A new algorithm for the index determination in DAEs by Taylor series using Algorithmic Differentiation

René Lamour and Dagmar Monett*

Institute of Mathematics
Humboldt University of Berlin
Unter den Linden 6,
10099 Berlin, Germany
{lamour,monett}@math.hu-berlin.de

Abstract. We present an approach for determining the tractability index using truncated polynomial arithmetic. In particular, computing the index this way generates a sequence of matrices that contains itself derivatives. We realize the time differentiations using Algorithmic Differentiation techniques, specially by using the standard ADOL-C package with which calculating the derivatives becomes a simple shift and scaling of coefficients. We present the theory supporting the procedure we propose, as well as the implementation issues behind it to provide a convenient interface to the standard ADOL-C functionality. We give also examples of academic and practical problems and report several experimental results we have obtained with them.

Keywords: DAE, index determination, tractability index, AD, Algorithmic Differentiation, Automatic Differentiation
AMS Subject Classifications: 65L80, 65D25, 68W30

1 Introduction

The solution of a linear differential-algebraic equation (DAE) may in contrast to an ODE depend on the derivative of the function at the right-hand side. How often parts of the right-hand side are to differentiate depends on the index of the DAE. The index is a measure of the difficulty to solve a DAE. Therefore the determination of an index is important, because we have to know the expected difficulties solving this DAE.

Different index definitions needs different procedures to calculate the index.

The *differentiation index* and the definitions based on it like the *structural index* and the *strangeness index* calculate the index along a solution of the DAE implicitly using the equations of the DAE and its derivatives forming the so-called derivative array. The higher the index, the bigger the dimension of the system under consideration is. Additionally, the differentiation of the equations changes the properties of the DAE (cf. [11]), which makes its analysis and treatment more difficult.

The *tractability index* analyzes the inner structure of a DAE using the linearization of the DAE along an arbitrary function (which naturally has to fulfil

* Partial financial support from the DFG (German Research Foundation) Research Center Matheon Mathematics for key technologies, Project D7: *Numerical simulation of integrated circuits for future chip generations*, is acknowledged.

at least the same smoothness conditions as a solution). A sequence of matrices (with the same dimension as the DAE) is computed and the subindex of the first nonsingular matrix in that sequence indicates the index of the DAE. However, this matrix sequence contains in general itself a differentiation (see (2.4) in Section 2), which also complicates the DAE analysis.

For both families of index definitions the question is how to realize the differentiation which occurs. Realizations of the strangeness index require the explicitly given derivative array by formulae [8]. Pryce [20] and later together with Nedialkov [17–19] use the advantages of Algorithmic Differentiation (AD) combined with the structural index concept to overcome the necessary explicit differentiations. The structural index cannot analyze selected easy structured DAEs. Furthermore, there are DAEs with constant coefficients of index 1 which have arbitrarily high structural index (cf. [21]).

The present paper describes a realization of the tractability index matrix chain applying AD. Based on the DAE and a function $\bar{x}$, the linearization of the DAE along $\bar{x}$ has to be computed. The aim is to determine the index of the DAE in the fixed point $\bar{x}(t) \in \mathbb{R}^m$ for a $t \in \mathbb{R}$. Only two differentiations are needed: a differentiation to compute the Jacobian matrix of the DAE and second one to realize the matrix sequence.

Algorithmic or automatic differentiation means roughly speaking the calculation of Taylor series of the functions of interest by using the structure and the involved elementary functions (cf. [5, 2]). AD tools are available for many platforms like C/C++, FORTRAN, matlab, python.

We use the standard AD package ADOL-C [23] that yields Taylor polynomials for the matrices $A(t)$, $B(t)$, and $D(t)$ on which we then perform linear algebra in our own library for truncated polynomial arithmetic. How these matrices are obtained is subject of Section 5, where we give details about the algorithm we propose. We start with the necessary mathematical background in Section 2 and then we follow with the mathematical realization of the matrix sequence in Section 3. Then we give an overview about AD in Section 4. The examples from Section 6 and the experimental results from Section 7 complete this work.

2 Mathematical Background

DAEs arise in electrical network simulation, chemical kinetics, multi body problems. Beside these classical fields DAEs also arise using simulation packages combining equations of various topics. Here it is not clear which index the resulting DAE has, but this information is important to check the relevance of the simulation and the expecting difficulties in the computation.

We deal with DAEs given by the general equation with properly stated leading term

$$f((d(x(t),t))', x(t), t) = 0, \qquad t \in I, \tag{2.1}$$

with $I \subseteq \mathbb{R}$ being the interval of interest.

Properly stated means that

$$\ker \frac{\partial f}{\partial z}(z,x,t) \oplus \operatorname{im} \frac{\partial d}{\partial x}(x,t) = \mathbb{R}^n,\ \forall z \in \mathcal{D}_z \subseteq \mathbb{R}^n,\ x \in \mathcal{D}_x \subseteq \mathbb{R}^m,\ t \in I \quad (2.2)$$

and rank $\frac{\partial f}{\partial z} =: r_A$ and rank $\frac{\partial d}{\partial x} =: r_D$ are constant in the considered regions.

To compute the index as addressed in [14, 15] we begin by linearizing the DAE along a chosen path $\bar{x}$. The linearization of (2.1) is given by

$$\underbrace{\frac{\partial f}{\partial z}(\bar{z}(t),\bar{x}(t),t)}_{A(t)\in\mathbb{R}^{m\times n}}(\underbrace{\frac{\partial d}{\partial x}(\bar{x}(t),t)}_{D(t)\in\mathbb{R}^{n\times m}} z(t))' + \underbrace{\frac{\partial f}{\partial x}(\bar{z}(t),\bar{x}(t),t)}_{B(t)\in\mathbb{R}^{m\times m}} z(t) = \underbrace{-f((d(\bar{x}(t),t))',\bar{x}(t),t)}_{q(t)} \quad (2.3)$$

with $\bar{z}(t) = d'(\bar{x}(t),t)$.

The tractability index of a DAE bases upon the matrix sequence

$$\begin{aligned} G_0 &= A\,D, \quad B_0 = B, \\ G_{i+1} &= G_i + B_i\,Q_i = (G_i + W_i\,B_0\,Q_i)\left(I + G_i^-\,B_i\,Q_i\right), \\ B_{i+1} &= \left(B_i - G_{i+1}\,D^-\left(D\,P_0\cdots P_{i+1}\,D^-\right)'\,D\,P_0\cdots P_{i-1}\right)P_i, \end{aligned} \quad (2.4)$$

where Q_i is an admissible projector function such that $\operatorname{im} Q_i = \ker G_i$, $P_i := I - Q_i$ and $Q_i(I - P_0\cdots P_{i-1}) = 0$. W_i are projector functions such that $\ker W_i = \operatorname{im} G_i$. Further, D^- denotes a reflexive generalized inverse of D and G_i^- the reflexive generalized inverse of G_i such that $G_i^- G_i = P_i$ and $G_i G_i^- = I - W_i$. The first index μ, where the matrix G_μ becomes nonsingular, is called the *tractability index*.

3 Realizing the matrix sequence

The realization of (2.4) requires the determination of admissible projectors Q_i and the differentiation of matrix products to calculate B_{i+1}.

Crucial part of the procedure are the differentiations with respect to time, which at first were approximated by divided differences [9]. The results were dependent on the differencing increment and only reliable for low index systems. These problems could be overcome by realizing the repeated differentiations in the form of Taylor series arithmetic, where time differentiation becomes a simple shift and scaling of coefficients on the original specification.

Let us introduce first the algorithm to compute the matrix function sequence (2.4).

3.1 Checking the well matched condition

The computation of the QR decompositions of the matrices A and D by applying Householder transformations with column pivoting yields

$$A = U_A \begin{pmatrix} R_{A,1}\ R_{A,2} \\ 0 \end{pmatrix} \Pi_A^T \quad \text{and} \quad D = U_D \begin{pmatrix} R_{D,1}\ R_{D,2} \\ 0 \end{pmatrix} \Pi_D^T,$$

with orthogonal matrices U_A, U_D, upper triangular matrices $R_{A,1}$ and $R_{D,1}$ and column permutation matrices Π_A and Π_D.

We compute

$$G_0 = A\,D, \tag{3.1}$$

which is used in the matrix sequence calculation (see (2.4)). Then we apply a Householder decomposition of G_0 and obtain

$$G_0 = U_0 \begin{pmatrix} R_{01} & R_{02} \\ & 0 \end{pmatrix} \Pi_0^T. \tag{3.2}$$

Let $\operatorname{rank} G_0 = r_0$. The *well matched condition* [14, 15] is checked in that $\operatorname{rank} A = \operatorname{rank} D = \operatorname{rank} G_0$. If these equalities are not satisfied, then the leading term is not properly stated and the algorithm should terminate.

In the case r_0 equals m, then both the rank and the dimension of the DAE are also m and the index is 0. However, if $r_0 < m$, then the matrix sequence should be computed.

3.2 Computing generalized inverses

If we know a decomposition of a matrix

$$Z = U_Z \begin{pmatrix} R_Z & 0 \\ 0 & 0 \end{pmatrix} V_Z^{-1}$$

with nonsingular matrices U_Z, V_Z and R_Z then a generalized reflexive inverse of Z is given by

$$Z^- = V_Z \begin{pmatrix} R_Z^{-1} & m_{Z,2} \\ m_{Z,1} & m_{Z,1} R_Z m_{Z,2} \end{pmatrix} U_Z^{-1}$$

with arbitrary matrices $m_{Z,1}, m_{Z,2}$.

To compute the generalized inverse of G_0, i.e. G_0^-, we use the QR factorization of G_0 from (3.2) and obtain

$$G_0 = U_0 \begin{pmatrix} R_{01} \\ & 0 \end{pmatrix} \begin{pmatrix} I & R_{01}^{-1} R_{02} \\ & I \end{pmatrix} \Pi_0^T. \tag{3.3}$$

Then we calculate

$$G_0^- = \underbrace{\Pi_0 \begin{pmatrix} I & -R_{01}^{-1} R_{02} \\ & I \end{pmatrix}}_{V_0} \begin{pmatrix} R_{01}^{-1} & m_{02} \\ m_{01} & m_{01} R_{01} m_{02} \end{pmatrix} U_0^{-1}, \tag{3.4}$$

m_{01} and m_{02} being matrices that are initialized to zero entries.

The generalized inverse of D, i.e. D^-, is also used in the matrix sequence. It can be computed by operating in (3.1) as follows

$$D^- = G_0^-\, A \tag{3.5}$$

and it fulfills the following properties (see [14, 15, 9] for details):

$$\begin{aligned} D &= D D^- D, \\ D^- &= D^- D D^-, \end{aligned} \tag{3.6}$$

where DD^- and D^-D are projectors with the same rank. Let R be a matrix such that $R = DD^-$.

3.3 Computing the matrix sequence

Every step of the matrix sequence (2.4) calculates G_{i+1} and B_{i+1}. The former needs the calculation of G_{i+1}^- in such a way that the resulting nullspace projector $Q_{i+1} := I - G_{i+1}^- G_{i+1}$ is admissible with respect to the former nullspace projectors $Q_j,\ j = 0, \ldots, i$. The latter needs the derivative of $D\,P_0 \cdots P_{i+1}\,D^-$. We will explain now how the matrices G_{i+1}^-, G_{i+1} and B_{i+1} are computed.

Computing the matrix G_{i+1}^- To calculate the matrix G_1 we use the matrices G_0 and G_0^- already introduced above. However, the calculation of the successive G_i's is not straightforward anymore. Similarly to the expression (3.4) for G_0^-, we have the following expression for G_{i+1}^- if we assume a decomposition of

$$G_{i+1} = \mathcal{U}_{i+1} \begin{pmatrix} S_{i+1} & \\ & 0 \end{pmatrix} \mathcal{V}_{i+1}^{-1} \tag{3.7}$$

$$G_{i+1}^- = \mathcal{V}_{i+1} \begin{pmatrix} S_{i+1}^{-1} & m_{i+1,2} \\ m_{i+1,1} & m_{i+1,1} S_{i+1} m_{i+1,2} \end{pmatrix} \mathcal{U}_{i+1}^{-1}. \tag{3.8}$$

Suppose a decomposition of $G_i = \mathcal{U}_i \begin{pmatrix} S_i & \\ & 0 \end{pmatrix} \mathcal{V}_i^{-1}$ and a known generalized inverse $G_i^- = \mathcal{V}_i \begin{pmatrix} S_i^{-1} & m_{i,2} \\ m_{i,1} & m_{i,1} S_i m_{i,2} \end{pmatrix} \mathcal{U}_i^{-1}$. Then,

$$\begin{aligned} Q_i &= I - G_i^- G_i = \mathcal{V}_i \begin{pmatrix} 0 & \\ -m_{i,1} S_i & I \end{pmatrix} \mathcal{V}_i^{-1} = \mathcal{V}_i \begin{pmatrix} 0 & \\ & I \end{pmatrix} T_{l,i}^{-1}\, \mathcal{V}_i^{-1}, \\ W_i &= I - G_i G_i^- = \mathcal{U}_i \begin{pmatrix} 0 & -S_i m_{i,2} \\ & I \end{pmatrix} \mathcal{U}_i^{-1} = \mathcal{U}_i\, T_{u,i}^{-1} \begin{pmatrix} 0 & \\ & I \end{pmatrix} \mathcal{U}_i^{-1}, \end{aligned} \tag{3.9}$$

with upper and lower triangular matrices

$$T_{u,i} = \begin{pmatrix} I & S_i m_{i,2} \\ & I \end{pmatrix} \quad \text{and} \quad T_{l,i} = \begin{pmatrix} I & \\ m_{i,1} S_i & I \end{pmatrix}$$

and using (2.4)

$$\begin{aligned} G_{i+1} &= \mathcal{U}_i \left(\begin{pmatrix} S_i & \\ & 0 \end{pmatrix} + T_{u,i}^{-1} \begin{pmatrix} 0 & \\ & I \end{pmatrix} \underbrace{\mathcal{U}_i^{-1} B_0 \mathcal{V}_i}_{\bar{B}_i} \begin{pmatrix} 0 & \\ & I \end{pmatrix} T_{l,i}^{-1} \right) \mathcal{V}_i^{-1} F_i \\ &= \mathcal{U}_i\, T_{u,i}^{-1} \left(\begin{pmatrix} S_i & \\ & 0 \end{pmatrix} + \begin{pmatrix} 0 & \\ & I \end{pmatrix} \bar{B}_i \begin{pmatrix} 0 & \\ & I \end{pmatrix} \right) T_{l,i}^{-1}\, \mathcal{V}_i^{-1} F_i, \end{aligned} \tag{3.10}$$

with a nonsingular matrix $F_i := I + G_i^- B_i Q_i$. Only the lower right block $\bar{B}_{i,22}$ of the matrix $\bar{B}_i = \begin{pmatrix} \bar{B}_{i,11} & \bar{B}_{i,12} \\ \bar{B}_{i,21} & \bar{B}_{i,22} \end{pmatrix}$ influences G_{i+1}. The matrices $\tilde{U}_{i+1}^T$ and $\tilde{V}_{i+1}$ come from a Householder decomposition of the matrix $\bar{B}_{i,22}$:

$$\bar{B}_{i,22} = \tilde{U}_{i+1} \begin{pmatrix} R_{i+1,1} & \\ & 0 \end{pmatrix} \underbrace{\begin{pmatrix} I & R_{i+1,1}^{-1} R_{i+1,2} \\ & I \end{pmatrix} \tilde{\Pi}_{i+1}^T}_{\tilde{V}_{i+1}^{-1}}. \tag{3.11}$$

Using (3.11) in (3.10) we reach the decomposition we are looking for

$$G_{i+1} = \underbrace{\mathcal{U}_i\, T_{u,i}^{-1} \begin{pmatrix} I & \\ & \tilde{U}_{i+1} \end{pmatrix}}_{=:U_{i+1}} \underbrace{\left(\begin{pmatrix} S_i & \\ 0 & \end{pmatrix} + \begin{pmatrix} 0 & \\ & \begin{pmatrix} R_{i+1,1} & \\ & 0 \end{pmatrix} \end{pmatrix} \right)}_{=:\begin{pmatrix} S_{i+1} & \\ & 0 \end{pmatrix}} \underbrace{\begin{pmatrix} I & \\ & \tilde{V}_{i+1}^{-1} \end{pmatrix} T_{l,i}^{-1}\, \mathcal{V}_i^{-1}\, F_i}_{=:\mathcal{V}_{i+1}^{-1}}$$

In practice, $\operatorname{rank} \bar{B}_{i,22} =: r_{i+1}$ determines whether to continue with the algorithm (i.e., $\dim S_i + r_{i+1} < m$) calculating the matrix G_{i+1}^- or to terminate because G_{i+1} becomes nonsingular.

We create the projectors Q_{i+1} and W_{i+1} by G_{i+1}^-. The parameter matrices $m_{i+1,1}$ and $m_{i+1,2}$ influence the properties of the projectors Q_{i+1} and W_{i+1} respectively. We need admissible projectors Q_{i+1}, i.e. $Q_{i+1}(I - P_0 \cdots P_i) = 0$ or equivalently $Q_{i+1}Q_j = 0,\ j = 0, \ldots, i$.

Are these conditions qualified to determine parameter matrices $m_{i+1,1}$? The condition reads in detail

$$0 = Q_{i+1}Q_j = (I - G_{i+1}^- G_{i+1})Q_j = Q_j - G_{i+1}^- B_j Q_j \Leftrightarrow Q_j = G_{i+1}^- B_j Q_j.$$

Using the special structure of Q_j (cf. (3.9)) we obtain

$$\begin{aligned} \mathcal{V}_j \begin{pmatrix} 0 & \\ & I \end{pmatrix} T_{l,j}^{-1}\, \mathcal{V}_j^{-1} &= G_{i+1}^- B_j \mathcal{V}_j \begin{pmatrix} 0 & \\ & I \end{pmatrix} T_{l,j}^{-1}\, \mathcal{V}_j^{-1} \\ &= \mathcal{V}_{i+1} \begin{pmatrix} S_{i+1}^{-1} & m_{i+1,2} \\ m_{i+1,1} & m_{i+1,1} S_{i+1} m_{i+1,2} \end{pmatrix} \mathcal{U}_{i+1}^{-1} B_j \mathcal{V}_j \begin{pmatrix} 0 & \\ & I \end{pmatrix} T_{l,j}^{-1}\, \mathcal{V}_j^{-1} \end{aligned}$$

which leads to

$$\underbrace{\mathcal{V}_{i+1}^{-1} \mathcal{V}_j \begin{pmatrix} 0 \\ I \end{pmatrix}}_{=:w_j} = \begin{pmatrix} S_{i+1}^{-1} & m_{i+1,2} \\ m_{i+1,1} & m_{i+1,1} S_{i+1} m_{i+1,2} \end{pmatrix} \underbrace{\mathcal{U}_{i+1}^{-1} B_j \mathcal{V}_j \begin{pmatrix} 0 \\ I \end{pmatrix}}_{=:z_j}. \tag{3.12}$$

Equation (3.12) has to have fulfilled for $j = 0, \ldots, i$. Introducing $\widehat{W}_i := (w_0, \ldots, w_i)$ and $Z_i := (z_0, \ldots, z_i)$ we have to solve the equation

$$\widehat{W}_i = \begin{pmatrix} S_{i+1}^{-1} & m_{i+1,2} \\ m_{i+1,1} & m_{i+1,1} S_{i+1} m_{i+1,2} \end{pmatrix} Z_i.$$

We can prove that $Z_i = \begin{pmatrix} \tilde{Z}_i \\ 0 \end{pmatrix} \} r_m$ with $r_m = m - \dim S_{i+1}$, and Z_i and $\tilde{Z}_i$ are column regular (cf. [9]) which leads to the representation

$$\widehat{W}_i =: \begin{pmatrix} \widehat{W}_{i,1} \\ \widehat{W}_{i,2} \end{pmatrix} = \begin{pmatrix} S_{i+1}^{-1} \\ m_{i+1,1} \end{pmatrix} \tilde{Z}_i.$$

Finally, we have for the matrix we are looking for

$$m_{i+1,1} = \widehat{W}_{i,2} \tilde{Z}_i^- \tag{3.13}$$

because $\tilde{Z}_i^- \tilde{Z}_i = I$ for any generalized reflexive inverse $\tilde{Z}_i^-$. $\tilde{Z}_i^-$ is computed via a Householder factorization.

Computing the matrix B_{i+1} For calculating the matrix B_{i+1} from $i \geq 0$ on we do

$$B_{i+1} = \Big(B_i - G_{i+1}\, D^- \left(D\, P_0 \cdots P_{i+1}\, D^- \right)' D\, P_0 \cdots P_{i-1} \Big)\, P_i, \tag{3.14}$$

with $B_0 = B$, $G_{i+1} = G_i + B_i\, Q_i$, $P_0 = D^- D$, and $Q_0 = I - P_0$.

By applying the product rule we have that

$$\begin{aligned} \left(D\, P_0 \cdots P_{i+1}\, D^- \right)' &= (D \underbrace{P_0 \cdots P_i}_{\mathcal{P}_i} P_{i+1}\, D^-)' \\ &= \left(D\, \mathcal{P}_i\, D^-\, D\, P_{i+1}\, D^- \right)' \\ &= \left(D\, \mathcal{P}_i\, D^- \right)'\ D\, P_{i+1}\, D^- + D\, \mathcal{P}_i\, D^-\ \left(D\, P_{i+1}\, D^- \right)' \end{aligned} \tag{3.15}$$

where the only challenge is the calculation of $(D\, P_{i+1}\, D^-)'$ at each step since the derivative $(D\, \mathcal{P}_i\, D^-)'$ is already calculated in the step before. This is initialized to the derivative of $D\, D^-$.

The major advantage of using AD is that no call to a differentiation routine is needed every time a derivative is to be computed. Instead, we compute the time differentiations through a *shift operator* over Taylor series. When the elements of the matrices D, D^-, and P_{i+1} are Taylor series, then the computation of $(D\, P_{i+1}\, D^-)'$ can be done by shifting the Taylor coefficients and by doing some multiplications with them, after multiplying the matrices. The Taylor coefficient propagation makes use of truncated polynomial arithmetic from [5].

Computational realization of the matrix sequence Figure 3.1 shows the pseudo code corresponding to the matrix sequence loop. The mathematical background described above is generalized and only its most important aspects are included. The matrices $A(t)$, $B(t)$, and $D(t)$, for example, are assumed to be computed before the matrix sequence calculations starts. Their construction will be explained in later sections. Output from the program is either the tractability index or an error message that can depend on some special situations.

Of particular interest are the time differentiations $(D\, P_{i+1}\, D^-)'$ at each step when computing B_{i+1}. The elements of the matrices D, P_{i+1}, and D^- are Taylor series, so the computation of the derivative is done by simply applying a shift operator as we have already addressed above. The precision of the computed derivative basically depends, however, on the number of Taylor coefficients that is defined, which is related to the index of the corresponding DAE. We will give more detail about AD and Taylor series in the following sections.

4 Algorithmic Differentiation

The DAEs we deal with contain derivatives. There are many ways to calculate and evaluate derivatives. We prefer the *Algorithmic Differentiation* approach [5] since the drawbacks from symbolic differentiation, numerical differentiation, and differentiation by hand are widely overcame by using AD. With AD the

```
int matrixSeq(A, D, B) {
   // It returns the tractability index value
   // m is the number of equations that define the DAE
   r_A = Householder(A);
   r_D = Householder(D);
   if r_A ≠ r_D then TerminalError;
   else {
      B_0 = B; G_0 = AD;
      r_0 = Householder(G_0);
      if r_0 ≠ r_A then TerminalError;
      else {
         Compute V_0;
         Compute G_0^-;
         D^- = G_0^- A;
      }
   }
   i = −1; dim = r_0;
   Initialize S_0 = R_01; b = V_02;
   Split V_0 after r_0 columns [V_01|V_02];
   while (i < m and dim < m)
      i++;
      Compute projectors P_i = G_i^- G_i, W_i, Q_i = I − P_i;
      Check out projectors' properties;
      Compute B̄_i and obtain B̄_{i,22};
      r_{i+1} = Householder(B̄_{i,22});
      dim += r_{i+1};
      if dim = m then return(index=i+1);
      else {
         if i > 0 {
            dDPiDm = shiftOperator(D P_{i+1} D^-);
            Apply product rule to obtain  D P_0 ··· P_{i+1} D^- ';
            Compute B_{i+1};
         }
         Compute G_{i+1} and V_{i+1};
         Ŵ_i = V_{i+1}^{-1} b;
         b = [b, V_{i+1,2}];
         Compute z_i and update Z = [Z, z_i];
         Compute Z̃^-;
         if Z̃^- has full rank then m_{i+1,1} = Ŵ_{i,2} Z̃^-;
         else TerminalError;
      }
   }
   return(index=i+1);
}
```

Fig. 3.1. Index determination with the matrix sequence.

computed derivatives are free of truncation and cancelation errors and they are exact up to machine precision. Furthermore, no explicit code development for derivatives by hand is needed and derivatives of arbitrary order can be computed automatically.

We then define derivatives of functions by computer programs so that AD can work. AD repeatedly applies the chain rule from the derivative calculus to these computer programs by implementing either a source code transformation of the original program or an operator overloading strategy. AD usually provides both the *forward mode* (or forward accumulation) and the *reverse mode* (or reverse accumulation) for the forward and reverse propagation of derivative values, respectively, when applying the chain rule. The former calculates derivatives of intermediate variables with respect to the independent variables. The latter calculates derivatives of the dependent variables with respect to the intermediate ones.

In particular we use the AD tool ADOL-C, a package to evaluate first and higher derivatives of vector functions that are defined by computer programs written in C or C++ [23]. Thus, our programs are written in C++ and we facilitate the user to provide the expressions for the random polynomial path $\bar{x}(t)$ to linearize the DAE, for the dynamic $d(\bar{x}(t), t)$ and for the DAE itself, as required for the construction of the matrices $A(t)$, $B(t)$, and $D(t)$ in (2.3). We consider then Taylor series expansions of the involved independent and dependent variables up to a degree defined by the user. For this purpose we provide new C++ classes, which overload built-in operators in C++ and implement several Taylor arithmetic functionalities when the coefficients of a matrix are Taylor series.

5 Algorithm for the index determination

The new algorithm we propose to compute the tractability index of DAEs introduces the following features: The approximations of the matrices $A(t)$, $B(t)$, and $D(t)$ and the evaluations of derivatives are computed using specific drivers from the C++ package ADOL-C. Furthermore, the time differentiations to compute B_{i+1} in (3.14) are realized via a *shift operator* over Taylor series, i.e., no more calls to a differentiation routine are needed. This allows to compute the derivative $(DP_0 \cdots P_{i+1}D^-)'$ only by shifting Taylor series coefficients and by doing some multiplications. In addition, the reflexive generalized inverses D^- and G_i^- and therefore the projectors, are computed using QR decompositions with column pivoting of the involved matrices. The Householder method is less expensive than the singular value decomposition, method that was applied in a precursor algorithm [9].

Figure 5.1 shows a flowchart that comprises the global operations concerning the index determination. The user only needs to provide the problem specification as a member of a certain abstract class (see top of the figure at the left side). We provide a convenient interface to standard ADOL-C functionality. This is why the vector functions related to the user's problems are written in C++.

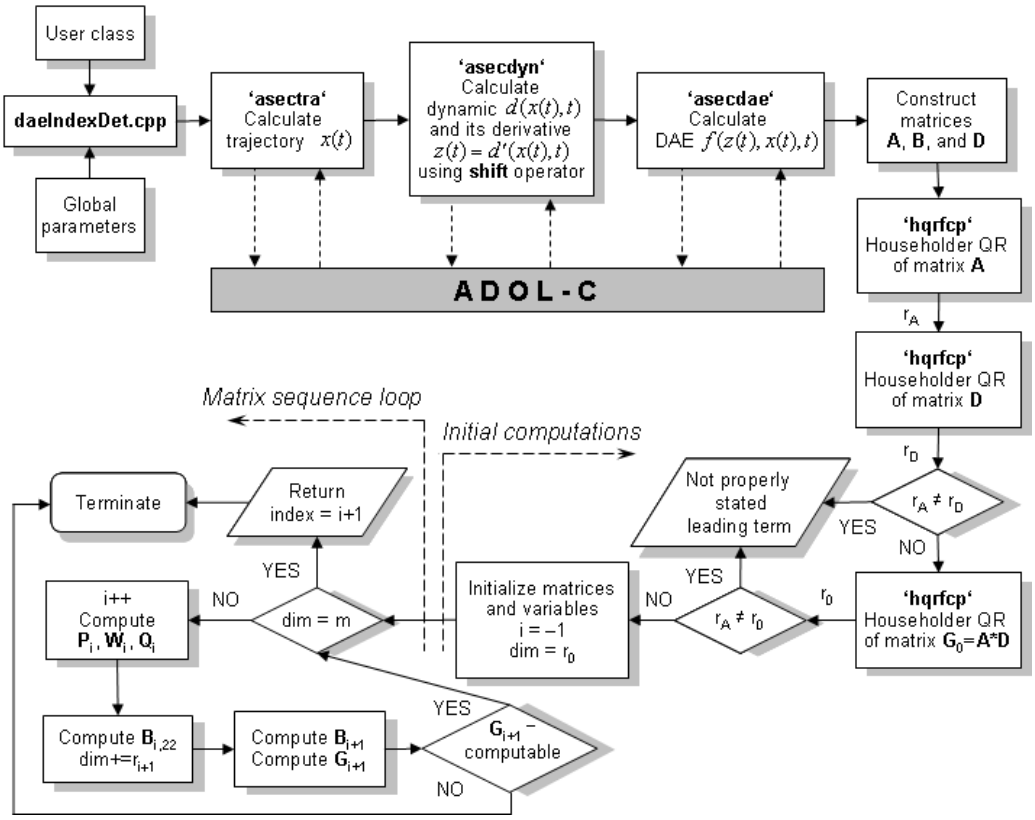

Fig. 5.1. Index determination by computing Taylor coefficients using ADOL-C.

The most relevant aspects of the flowchart above will be addressed in the next sections.

5.1 Computing Taylor coefficients

An *active section* in ADOL-C is a sequence of statements that contains the calculations involving the differentiable quantities at some time during the program execution. Their input and output variables can be expressed in terms of univariate Taylor expansions that are truncated after a certain derivative degree. This is the approach that ADOL-C [23] follows. We consider opportune to describe here the general expressions for the Taylor expansions ADOL-C work with. For the independent variables we have:

$$x(t+h) = \sum_{j=0}^{deg} x_j h^j + O(h^{deg+1}),$$

with $t, h \in \mathbb{R}$, deg the highest derivative degree defined by the user, and scaled derivatives of $x(t) : \mathbb{R} \to \mathbb{R}^n$ at $h = 0$, i.e., Taylor coefficients vectors at the parameter origin:

$$x_j = \frac{1}{j!} \frac{\partial^j}{\partial t^j} x(t). \tag{5.1}$$

For the dependent variables we have:

$$y(t+h) = \sum_{j=0}^{deg} y_j h^j + O(h^{deg+1}),$$

with $y(t) = F(x(t)) : \mathbb{R} \to \mathbb{R}^m$, F being deg times continuously differentiable, and $y(t)$ smooth with $(deg+1)$ Taylor coefficient vectors $y_j \in \mathbb{R}^m$. In particular, the Taylor coefficients of both the dependent and the independent variables of the trajectory, of the dynamic, and of the DAE are computed this way, as well as the Jacobian path $\frac{\partial F}{\partial x}$, when needed. The functions `forward` and `reverse` are used with this respect. They implement the ADOL-C forward and reverse modes we have introduced above, respectively.

A pseudo code of the algorithm we propose to compute the matrices of Taylor series $A(t)$, $B(t)$, and $D(t)$ is presented in Figure 5.2. It shows in some deep how these matrices are obtained. We will give now an overview of the general functioning.

Computing the trajectory The *trajectory*, i.e., the chosen polynomial path $\bar{x} : \mathbb{R} \to \mathbb{R}^m$ that is used to linearize the DAE, is computed in the function `asectra` from Figure 5.2, for a $t \in \mathbb{R}$ at which the index of the DAE should be determined. For doing this, an active section is defined where the expression for $\bar{x}$ is evaluated at t.

The Taylor coefficients $X_{tra} \in \mathbb{R}^{1,deg+1}$ of the independent variable, i.e., of t, are computed as described above in (5.1). They are initialized to the value t_0 given by the user. The dependent variable, i.e., $\bar{x}(t)$, has Taylor coefficients

```
int asectra() {
    // Compute the trajectory:
    Given t, compute x̄(t);
    // Compute the Taylor coeff. of t and x̄(t) (i.e. X_tra and Y_tra, resp.):
    forward( ..., X_tra, Y_tra);
}
int asecdyn() {
    // Compute the dynamic:
    Given x̄(t) and t, compute d(x̄(t), t);
    // Compute the Taylor coeff. of d(x̄(t), t) (i.e. Y_dyn):
    X_dyn = [Y_tra, X_tra];
    forward( ..., X_dyn, Y_dyn);
    // Compute the adjoints to obtain the Jacobian matrix ∂d/∂x:
    reverse( ..., Z_dyn, ... );
    // Compute z̄(t) = d'(x̄(t), t)
    Y_ddyn = shiftOperator(Y_dyn);
}
int asecdae() {
    // Compute the DAE:
    Given z̄(t), x̄(t), and t, compute f(z̄(t), x̄(t), t);
    // Compute the Taylor coeff. of f(z̄(t), x̄(t), t) (i.e. Y_dae):
    X_dae = [Y_ddyn, Y_tra, X_tra];
    forward( ..., X_dae, Y_dae);
    // Compute the adjoints to obtain the Jacobian matrix ∂f/∂y = ∂f/∂z, ∂f/∂x, ∂f/∂t :
    reverse( ..., Z_dae, ... );
}
int ABD() {
    // Compute the matrices A, B, and D:
    Using adjoints Z_dyn, construct D(t) = ∂d/∂x ;
    Using adjoints Z_dae, construct A(t) = ∂f/∂z and B(t) = ∂f/∂x ;
}
```

Fig. 5.2. How to compute the matrices of Taylor series using ADOL-C.

$Y_{tra} \in \mathbb{R}^{m,deg+1}$ and are computed with a call to the ADOL-C function `forward`, which implements the ADOL-C forward mode, as it was already described. Both matrices X_{tra} and Y_{tra} are relevant to the construction of the matrices $A(t)$, $B(t)$, and $D(t)$ as we will see later on.

Computing the dynamic The *dynamic* $d : \mathbb{R}^{m+1} \to \mathbb{R}^n$ is computed similarly to the trajectory, thereby with a call to the function `asecdyn` from Figure 5.2. The variables $\bar{x}(t)$ and t are already known from the calculation of the trajectory so that we can easily compute $d(\bar{x}(t), t)$ within the corresponding active section. We then calculate the Taylor coefficients of the independent and dependent variables for this case, i.e., $X_{dyn} \in \mathbb{R}^{m+1,deg+1}$ (with m independents from $\bar{x}(t)$ and one from t) and $Y_{dyn} \in \mathbb{R}^{n,deg+1}$, similarly as we have done for the trajectory.

The initialization of X_{dyn}, necessary to call the function `forward` from ADOL-C, depends on the already computed matrices Y_{tra} and X_{tra} from the section above. This is why we only need to compute the Taylor coefficients Y_{dyn} of the dependent variable, i.e., $d(\bar{x}(t), t)$, that are later used to compute the derivative $\bar{z}(t) = d'(\bar{x}(t), t)$, with $\bar{z} : \mathbb{R}^{m+1} \to \mathbb{R}^n$.

The output $Z_{dyn} \in \mathbb{R}^{n,m+1,deg}$, after calling the ADOL-C function `reverse`, is used to compute the Jacobian matrix $\frac{\partial d}{\partial x}$. Z_{dyn} contains the *adjoints*, as defined in ADOL-C, needed for constructing the matrix $D(t)$.

Finally, we calculate $\bar{z}(t)$ by applying a *shift* operator. In this case, the coefficients of the Taylor polynomials are shifted and some multiplications are done. The new matrix $Y_{ddyn} \in \mathbb{R}^{n,deg+1}$ has a zeroed last column (the one belonging to the derivative of the independent terms).

Computing the DAE The *DAE* $f : \mathbb{R}^{n+m+1} \to \mathbb{R}^m$ is computed similarly but with a call to the function `asecdae` from Figure 5.2. The expression for $f(\bar{z}(t), \bar{x}(t), t) = 0$ is evaluated in the corresponding active section. Again, we do not have to calculate the Taylor coefficients of all involved variables. We have already the ones from the independent variables, i.e., from $\bar{z}(t)$, $\bar{x}(t)$, and t, calculated up to the section above: the matrices Y_{ddyn}, Y_{tra}, and X_{tra}, respectively.

Thus, the Taylor coefficients $X_{dae} \in \mathbb{R}^{n+m+1,deg+1}$ (with n independents from $\bar{z}(t)$, m from $\bar{x}(t)$ and one from t) do not need to be recomputed. The Taylor coefficients of the dependent variable, i.e., of $f(\bar{z}(t), \bar{x}(t), t)$, are calculated with a call to the ADOL-C function `forward`.

$Z_{dae} \in \mathbb{R}^{m,n+m+1,deg}$ is obtained after a call to the ADOL-C function `reverse`. It is used to compute the Jacobian matrix $\frac{\partial f}{\partial y} := \left(\frac{\partial f}{\partial z}, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial t}\right)$, which is later used in the construction of the matrices $A(t)$ and $B(t)$.

Constructing the matrices A, B, and D Using the already computed Jacobian matrices $\frac{\partial d}{\partial x}$ and $\frac{\partial f}{\partial y}$ we can straightforwardly construct the matrices $A(t)$, $B(t)$, and $D(t)$ as follows: $A(t) = \left(\frac{\partial f}{\partial z}\right)$, $B(t) = \left(\frac{\partial f}{\partial x}\right)$ and $D(t) = \left(\frac{\partial d}{\partial x}\right)$.

Some programming challenges The function that implements the *Householder QR with column pivoting* of a matrix basically follows the Algorithm 5.4.1 from [4]. We have extended the classic Householder method to work with matrices of Taylor series, introducing a new approach where the elements of the matrices are vectors of Taylor coefficients. For example, to select the pivot elements we analyze the first coefficients of the involved Taylor series, which correspond to the evaluation of the function. Important parameters passed into and out from our Householder C++ function are:

- (input/output) The original matrix to be factorized and its dimension.
- (output) The resulting matrix R (its diagonal and upper triangular part).
- (output) A vector of permutations on the columns of the original matrix.
- (output) The resulting orthogonal matrix Q.
- (input) The threshold value used in the stop criterion.
- (output) The calculated rank.

Furthermore, we have implemented and extended several Linear Algebra functionalities to operate over matrices of Taylor series. In particular, we apply the Taylor coefficient propagation by means of truncated polynomial arithmetic from [5] (Sect. 13.2, p. 303). To this end we provide special classes that overload several operators (i.e. arithmetic and assignment operators, relational and equality operators, and subscript operators) for Taylor arithmetic by redefining the meaning of some built-in operators in C++. We can then consider different types of matrix multiplications of the form $C = \alpha A \cdot B + \beta C$ (for transposed A and/or B, for inferior-right block of B being the identity matrix, for A or B where only the upper triangular part is of interest, among others) as well as solving equations like $U \cdot X = B$, thereby making only a few modifications to the back-substitution algorithm from [4].

6 Examples

In the sections that follow we will present the examples we have used to determine the index and to test our algorithm. The DAEs belong to the following cases:

1. Academic examples.
2. Classical pendulum.
3. Robotic arm.
4. Electromechanical problems: dynamo and plate-type capacitor.
5. Circuit simulations: bipolar ring oscillator, bipolar ring oscillator with inductor, and bipolar ring oscillator with cross-modulation.

The purpose of presenting several examples is to show different cases where the problem dimensions vary, as well as the conditions under which the index is calculated. We want to analyze critical situations from simple examples to more complex ones.

6.1 Academic examples

Example 1 ([7])

$$\begin{aligned} x_2' + x_1 - t &= 0, \\ x_2' + x_3' + x_1 x_2 + \eta x_2 - 1 &= 0, \\ x_2\left(1 - \frac{x_2}{2}\right) + x_3 &= 0, \end{aligned} \tag{6.1}$$

with $\eta \in \mathbb{R}$. By considering the differential terms x_2' and x_3' that appear in the first two equations of this DAE we define the dynamic function $d(x,t)$ as follows

$$d\,(x,t) = \begin{pmatrix} x_2 \\ x_2 + x_3 \end{pmatrix}.$$

We compute the linearization (2.3) along the trajectory[1] $\bar{x}$ and we choose:

$$\bar{x}(t) = \begin{pmatrix} t + c \\ 2 - 2e^{t-1} \\ \log(t+1) \end{pmatrix}$$

with $c \in \mathbb{R}$. The DAE has index 3 when $d := \det G_3 = \bar{x}_1 + \bar{x}_2' + \eta \neq 0$. The computation of the index depends on the derivative $\bar{x}_2'$. Choosing $\eta = 1$ and $t_0 = 1$ we obtain a singular matrix chain for $c = 0$ because of $\bar{x}_1 + \bar{x}_2' + \eta = t_0 + c - 2 + 1 = c$. In this case, the index is not defined.

With $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$, $D = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$ and $B = \begin{pmatrix} 1 & 0 & 0 \\ \bar{x}_2 & \bar{x}_1 + \eta & 0 \\ 0 & 1 - \bar{x}_2 & 1 \end{pmatrix}$ we obtain $G_0 = AD$, $Q_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ and $G_1 = G_0 + BQ_0 = \begin{pmatrix} 1 & 1 & 0 \\ \bar{x}_2 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}$. An admissible projector onto the nullspace of G_1 is given by $Q_1 = \begin{pmatrix} 0 & -1 + (1 - \bar{x}_2)\beta & \beta \\ 0 & 1 - (1 - \bar{x}_2)\beta & -\beta \\ 0 & (1 - \bar{x}_2)(-1 + (1 - \bar{x}_2)\beta) & (1 - \bar{x}_2)\beta \end{pmatrix}$ where β is an arbitrary smooth function. Computing G_2 using *Mathematica* we obtained a nullspace projector onto $\ker G_2$

$$Q_2 = \begin{pmatrix} 0 & \frac{(\bar{x}_2 - 1)(1 - \beta(d - \bar{x}_2'))}{d} & \frac{(1 - \beta(d - \bar{x}_2'))}{d} \\ 0 & \frac{(\bar{x}_2 - 1)(d\beta - 1)}{d} & \frac{(d\beta - 1)}{d} \\ 0 & \frac{(\bar{x}_2 - 1)((\bar{x}_2 - 1)(d\beta - 1) + d)}{d} & \frac{((x_2 - 1)(d\beta - 1) + d)}{d} \end{pmatrix}.$$

The explicit representation of the projectors allows a comparison with the computed values. We will report this in the experimental section.

[1] We recall that the function $\bar{x}$ is not a solution of (2.1) but a random polynomial path that is arbitrarily chosen to linearize the DAE.

Example 2 ([10])

$$\begin{aligned} x_1' + x_1 + x_4 &= 0, \\ x_2' + \alpha(x_1, x_2, x_3, t)x_4 &= 0, \\ x_3' + x_1 + x_2 + x_3 &= 0, \\ x_3 - p(t) &= 0, \end{aligned} \tag{6.2}$$

where α is a nonnegative C^1 function on $\mathbb{R}^3 \times \mathbb{R}$ and p is C^2 on $\mathbb{R}$. This DAE has index 3. The following function defines the dynamic, according to the differential terms x_1', x_2', and x_3' in (6.2):

$$d\,(x,t) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

We choose the following function $\bar{x} : \mathbb{R} \to \mathbb{R}^4$ to define the trajectory:

$$\bar{x}(t) = \begin{pmatrix} \sin(t) \\ \cos(t) \\ \sin(2t) \\ \cos(2t) \end{pmatrix}.$$

6.2 Classical pendulum

Example 3

A well known DAE is the one that describes the behavior of a classical pendulum:

$$\begin{aligned} x_1' - x_3 &= 0, \\ x_2' - x_4 &= 0, \\ x_3' + x_1 x_5 &= 0, \\ x_4' + x_2 x_5 - g &= 0, \\ x_1^2 + x_2^2 &= 1, \end{aligned} \tag{6.3}$$

with $g = 9.81$ being the gravity constant. This DAE has index 3. The dynamic function $d(x,t)$ is defined by

$$d\,(x,t) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix},$$

according to the differential terms x_1' to x_4' that appear in (6.3). We choose the following trajectory $\bar{x}(t)$:

$$\bar{x}(t) = \begin{pmatrix} \sin(t) \\ \cos(t) \\ \log(1+t) \\ \sin(2t) \\ 5 \end{pmatrix}.$$

6.3 Robotic arm

Example 4 ([3])

The following example describes the movement of a robotic arm:

$$\begin{aligned}
x_1' - x_4 &= 0,\\
x_2' - x_5 &= 0,\\
x_3' - x_6 &= 0,\\
x_4' - 2c(x_3)(x_4+x_6)^2 - x_4^2 d(x_3) - (2x_3 - x_2)(a(x_3) + 2b(x_3)) + & \\
-a(x_3)x_7 + a(x_3)x_8 &= 0,\\
x_5' + 2c(x_3)(x_4+x_6)^2 + x_4^2 d(x_3) - (2x_3 - x_2)(1 - 3a(x_3) - 2b(x_3)) + & \\
+a(x_3)x_7 - (a(x_3)+1)x_8 &= 0,\\
x_6' + 2c(x_3)(x_4+x_6)^2 + x_4^2 d(x_3) - (2x_3 - x_2)(a(x_3) - 9b(x_3)) + & \\
+2x_4^2 c(x_3) + (x_4+x_6)^2 d(x_3) + (a(x_3)+b(x_3))x_7 - (a(x_3)+b(x_3))x_8 &= 0,\\
\cos(x_1) + \cos(x_1+x_3) - p_1(t) &= 0,\\
\sin(x_1) + \sin(x_1+x_3) - p_2(t) &= 0,
\end{aligned} \tag{6.4}$$

with

$$\begin{aligned}
p_1(t) &= \cos(e^t - 1) + \cos(t-1), & p_2(t) &= \sin(1-e^t) + \sin(1-t),\\
a(x) &= \frac{2}{2-\cos^2(x)}, & b(x) &= \frac{\cos(x)}{2-\cos^2(x)},\\
c(x) &= \frac{\sin(x)}{2-\cos^2(x)}, & d(x) &= \frac{\cos(x)\sin(x)}{2-\cos^2(x)}.
\end{aligned}$$

This DAE has index 5. The dynamic $d(x,t)$ is defined by considering the differential terms x_1' to x_6' that appear in (6.4), as follows

$$d(x,t) = \begin{pmatrix} x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6 \end{pmatrix}.$$

The trajectory $\bar{x}(t)$ we choose is

$$\bar{x}(t) = \begin{pmatrix} 1-e^t\\ \cos(t)\\ e^t - t\\ 2\sin(t)\\ \sin(t)+\cos(t)\\ \log(3+t)\\ \sin(t)-\cos(t)\\ 4\cos(t)+1 \end{pmatrix}.$$

6.4 Electromechanical problems

Example 5 (Dynamo [1])

Figure 6.1 shows a mechanical system, our next example. It consists of a bike dynamo and its electrical circuit.

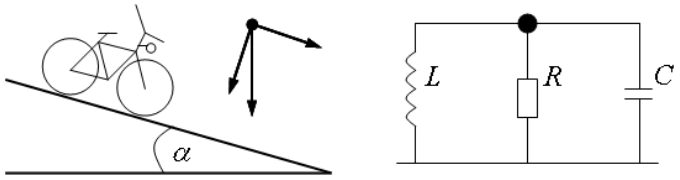

Fig. 6.1. Bike downhill with dynamo and corresponding electrical circuit.

The DAE is given by

$$\begin{aligned} p' - v &= 0, \\ v' - \tilde{f}(v, j_L, t) + \lambda &= 0, \\ p - z(t) &= 0, \\ Ce' + Ge - j_L &= 0, \\ \phi' - e &= 0, \\ \phi - \phi_L(v, t) &= 0, \end{aligned} \tag{6.5}$$

where $z(t)$ is a function to model the variation of the mass point position p depending on the time t, v is the velocity, e is the voltage at the knot, ϕ is the magnetic flow, j_L is the current through the inductance, λ is a constraining force, and C and $G = \frac{1}{R}$ are the capacity and the electrical conductance, respectively, with R the electrical resistance. Further we have:

$$\begin{aligned} \phi_L(v, t) &= k_0\, r \cos(2\pi k v t), \\ \tilde{f}(v, j_L, t) &= \frac{f_{mech}}{m} + \frac{2\pi}{m\,k} k_0\, j_L \sin(2\pi k v t), \\ f_{mech} &= g\, m \tan(\alpha), \\ z(t) &= \sin(t), \end{aligned}$$

where m is the mass of the bike, $g = 9.81$ is the gravity constant, α is the inclination angle, r is the axial radius of the inductor, and k and k_0 are problem specific constants.

The dynamic $d(x, t)$ has the following expression:

$$d\,(x, t) = \begin{pmatrix} p \\ v \\ e \\ \phi \end{pmatrix},$$

according to the differential terms p', v', e', and ϕ' in (6.5), respectively. The trajectory $\bar{x}(t)$ we choose in this case is defined by:

$$\bar{x}(t) = \begin{pmatrix} \sin(t) \\ \cos(t) \\ \cos(t) - t\sin(t) \\ \cos(t) \\ \sin(t) \\ \cos(t) \end{pmatrix},$$

for p, v, e, ϕ, λ, and j_L, respectively.

We can compute the solution of (6.5) explicitly. We obtain $p = z \Rightarrow v = p' \Rightarrow \Phi = \Phi_L(v,t) \Rightarrow e = \Phi' \Rightarrow j_L = Ce' + Ge \Rightarrow \lambda = \tilde{f}(v, j_L, t) - v'$.

The DAE from (6.5) has index 4.

Example 6 (Plate-type capacitor [1])

The DAE for this example is given by

$$\begin{aligned} p_1' - v_1 &= 0, \\ p_2' - v_2 &= 0, \\ v_1' + g - K(l_0 - p_1 + p_2) + \lambda &= 0, \\ v_2' + g + K(l_0 - p_1 + p_2) - f_{el}(e, p_2) &= 0, \\ p_1 - x_1(t) &= 0, \\ q' + j_L &= 0, \\ Lj_L' + e &= 0, \\ q - C(p_2)e &= 0, \end{aligned} \tag{6.6}$$

with

$$f_{el}(e, p_2) = -\tfrac{\epsilon_0 \epsilon_r A e^2}{2p_2^2}, \qquad C(p_2) = \tfrac{\epsilon_0 \epsilon_r A}{p_2}, \qquad x_1(t) = \sin(t),$$

and $g = 9.81$ being the gravity constant, $\epsilon_0 = 8.854 \cdot 10^{-12}$ the absolute permittivity, $\epsilon_r = 1.00059$ the relative permittivity of the air, and L, A, K and l_0 the inductance, the capacity area, the spring constant, and its length, respectively.

Figure 6.2 shows the plate-type capacitor. The DAE has index 3. The differential terms in (6.6) define the dynamic $d(x,t)$ as follows

$$d\,(x,t) = \begin{pmatrix} p_1 \\ p_2 \\ v_1 \\ v_2 \\ q \\ j_L \end{pmatrix},$$

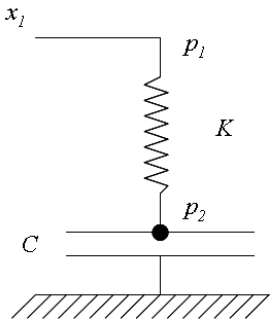

Fig. 6.2. Plate-type capacitor.

according to p_1', p_2', v_1', v_2', q', and j_L' from (6.6), respectively. For this example we choose the trajectory $\bar{x}(t)$ as follows:

$$\bar{x}(t) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ -1 \\ -1 \\ -1 \end{pmatrix},$$

for p_1, p_2, v_1, v_2, e, q, λ, and j_L, in this order.

6.5 Circuit simulations

We present in this subsection, the last one dedicated to give a flavor about the type of problems we have addressed, to three examples of DAEs that model electrical circuits. DAEs from electrical circuits that have neither loops consisting of capacitors and voltage sources (CV-loop) nor cutsets containing inductors and current sources (LI-cutset) have index 1. Adding at least one of such electrical components to the topology would make the index higher, as it was already shown in [22]. If the CV-loops and the LI-cutsets occur in a certain configuration, then the index is 3 or even higher.

Example 7 (Bipolar ring oscillator [6])

We start with a bipolar ring oscillator (BRO) as shown in Figure 6.3. The corresponding DAE has index 1 because it doesn't have any CV-loop nor a LI-cutset. The nonlinear capacity model q of the voltage v represents leading

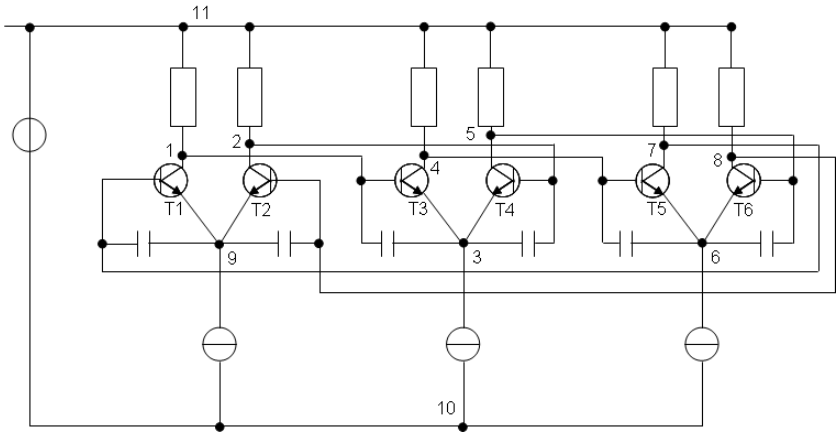

Fig. 6.3. Bipolar ring oscillator.

terms of the DAE, which is given by:

$$\begin{aligned}
\gamma\,\big(Ge_1 + j_{C1} + j_{B3} + q'(e_1 - e_3)\big) &= 0,\\
\gamma\,\big(Ge_2 + j_{C2} + j_{B4} + q'(e_2 - e_3)\big) &= 0,\\
\gamma\,\big(j_{E3} + j_{E4} + i - (q'(e_1 - e_3) + q'(e_2 - e_3))\big) &= 0,\\
\gamma\,\big(Ge_4 + j_{C3} + j_{B5} + q'(e_4 - e_6)\big) &= 0,\\
\gamma\,\big(Ge_5 + j_{C4} + j_{B6} + q'(e_5 - e_6)\big) &= 0,\\
\gamma\,\big(j_{E5} + j_{E6} + i - (q'(e_4 - e_6) + q'(e_5 - e_6))\big) &= 0,\\
\gamma\,\big(Ge_7 + j_{C5} + j_{B1} + q'(e_7 - e_9)\big) &= 0,\\
\gamma\,\big(Ge_8 + j_{C6} + j_{B2} + q'(e_8 - e_9)\big) &= 0,\\
\gamma\,\big(j_{E1} + j_{E2} + i - (q'(e_7 - e_9) + q'(e_8 - e_9))\big) &= 0,\\
j_V - 3i &= 0,\\
e_{10} + v &= 0,
\end{aligned} \tag{6.7}$$

with

$$q(e_a - e_b) = -4 \cdot 10^{-11} v_B \sqrt{1 - \frac{e_a - e_b}{v_B}},$$

$$\begin{aligned}
v_{BE} &= EB - EE, & v_{BC} &= EB - EC,\\
j_{BE} &= \frac{I_S}{\alpha}(e^{\frac{v_{BE}}{v_T}} - 1), & j_{BC} &= \frac{I_S}{\alpha_r}(e^{\frac{v_{BC}}{v_T}} - 1),\\
j_B &= j_{BE} + j_{BC}, & j_C &= \alpha\, j_{BE} - (1 + \alpha_r)\, j_{BC},
\end{aligned}$$

where EE, EC and EB denote the knot potentials at the emitter, at the collector and at the basis of the transistor, respectively, with constant values $\alpha = 100$, $\alpha_r = 10$, $v_T = 0.0258$, $G = 10^{-3}$, $I_S = 10^{-12}$, $v = 1.5$, $v_B = 0.8$, and $i = 3 \cdot 10^{-4}$. The scale factor γ is used to improve the behavior of the resulting nonlinear system.

The dynamic $d(x,t)$ of this system is defined by:

$$d\,(x,t) = \begin{pmatrix} q(e_1 - e_3) \\ q(e_2 - e_3) \\ q(e_4 - e_6) \\ q(e_5 - e_6) \\ q(e_7 - e_9) \\ q(e_8 - e_9) \end{pmatrix},$$

for the differential terms q' in (6.7). We choose the following trajectory $\bar{x}(t)$:

$$\bar{x}(t) = \begin{pmatrix} \cos(t) \\ \cos(t) \\ \cos(t) \\ \cos(t) \\ \cos(t) \\ \cos(t) \\ \cos(t) \\ \cos(t) \\ \cos(t) \\ \cos(t) \\ -1.5 \\ 9 \cdot 10^{-4} \end{pmatrix},$$

for e_1 to e_{10}, and j_V, in this order.

Example 8 (Bipolar ring oscillator with LI-cutset [6])

Now we add an inductor to the bipolar ring oscillator from Figure 6.3 obtaining a BRO with an LI-cutset. The DAE has then index 2. The electrical circuit is shown in Figure 6.4. The nonlinear capacity model q of the voltage v

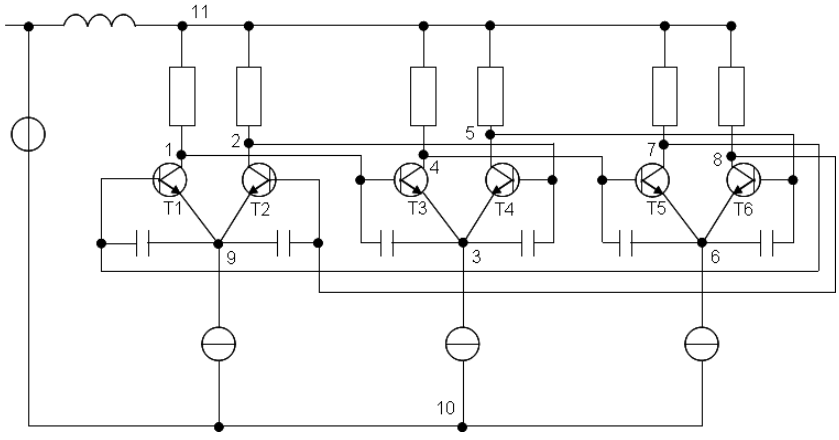

Fig. 6.4. Bipolar ring oscillator with inductor.

represents leading terms of the DAE, which is given by:

$$\begin{aligned}
\gamma\,\big(G(e_1-e_{11})+j_{C1}+j_{B3}+q'(e_1-e_3)\big)&=0,\\
\gamma\,\big(G(e_2-e_{11})+j_{C2}+j_{B4}+q'(e_2-e_3)\big)&=0,\\
\gamma\,\big(j_{E3}+j_{E4}+i-(q'(e_1-e_3)+q'(e_2-e_3))\big)&=0,\\
\gamma\,\big(G(e_4-e_{11})+j_{C3}+j_{B5}+q'(e_4-e_6)\big)&=0,\\
\gamma\,\big(G(e_5-e_{11})+j_{C4}+j_{B6}+q'(e_5-e_6)\big)&=0,\\
\gamma\,\big(j_{E5}+j_{E6}+i-(q'(e_4-e_6)+q'(e_5-e_6))\big)&=0,\\
\gamma\,\big(G(e_7-e_{11})+j_{C5}+j_{B1}+q'(e_7-e_9)\big)&=0,\\
\gamma\,\big(G(e_8-e_{11})+j_{C6}+j_{B2}+q'(e_8-e_9)\big)&=0,\\
\gamma\,\big(j_{E1}+j_{E2}+i-(q'(e_7-e_9)+q'(e_8-e_9))\big)&=0,\\
j_V-3i&=0,\\
e_{10}+v&=0,\\
G(6e_{11}-e_1-e_2-e_4-e_5-e_7-e_8)-j_L&=0,\\
Lj_L'+e_{11}&=0,
\end{aligned}\tag{6.8}$$

with q, v_{BE}, v_{BC}, j_{BE}, j_{BC}, j_B, and j_C calculated as in the former example. The constants α, α_r, v_T, G, I_S, v, v_B, and i have the same values as before. Further we have $L = 1$. The scale factor γ is used to improve the condition of the problem.

The dynamic $d(x,t)$ of this system is defined by:

$$d\,(x,t)=\begin{pmatrix} q(e_1-e_3)\\ q(e_2-e_3)\\ q(e_4-e_6)\\ q(e_5-e_6)\\ q(e_7-e_9)\\ q(e_8-e_9)\\ j_L \end{pmatrix},$$

according to the differential terms q' and j_L' that appear in (6.8). For this example we choose the following trajectory $\bar{x}(t)$:

$$\bar{x}(t)=\begin{pmatrix} \kappa\,\cos(t)\\ \kappa\,\cos(t)\\ \cos(t)\\ \kappa\,\cos(t)\\ \kappa\,\cos(t)\\ \cos(t)\\ \kappa\,\cos(t)\\ \kappa\,\cos(t)\\ \cos(t)\\ -1.5\\ 9\cdot 10^{-4}\\ 0.0\\ \cos(t) \end{pmatrix},$$

for e_1 to e_{10}, j_V, e_{11}, and j_L, in this order, with $\kappa = 1.01$.

Example 9 (Bipolar ring oscillator with cross-modulation [6])

Finally, we add a capacitor and a voltage source in a loop, i.e., a CV-loop, to the electrical circuit from Figure 6.4. We obtain then a bipolar ring oscillator with a *cross-modulation*, i.e., a BRO with both an LI-cutset and a CV-loop which is shown in Figure 6.5. The corresponding DAE has index 3.

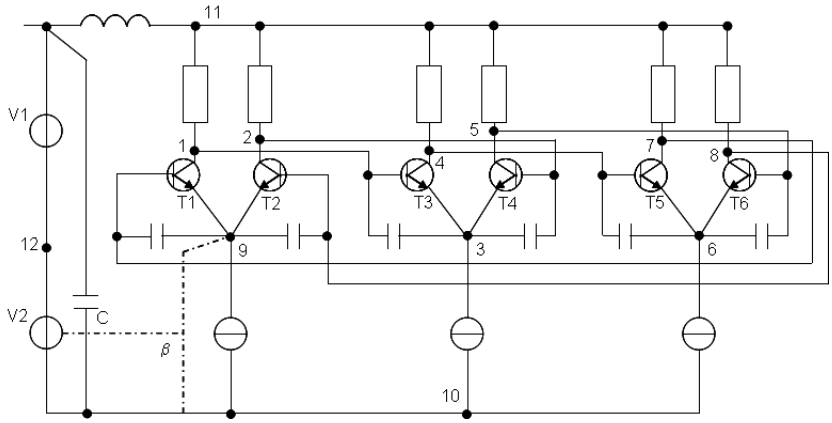

Fig. 6.5. Bipolar ring oscillator with cross-modulation.

The nonlinear capacity model q of the voltage v represents leading terms of the DAE, which is given by:

$$
\begin{aligned}
\gamma\,\big(G(e_1-e_{11})+j_{C1}+j_{B3}+q'(e_1-e_3)\big) &= 0,\\
\gamma\,\big(G(e_2-e_{11})+j_{C2}+j_{B4}+q'(e_2-e_3)\big) &= 0,\\
\gamma\,\big(j_{E3}+j_{E4}+i-(q'(e_1-e_3)+q'(e_2-e_3))\big) &= 0,\\
\gamma\,\big(G(e_4-e_{11})+j_{C3}+j_{B5}+q'(e_4-e_6)\big) &= 0,\\
\gamma\,\big(G(e_5-e_{11})+j_{C4}+j_{B6}+q'(e_5-e_6)\big) &= 0,\\
\gamma\,\big(j_{E5}+j_{E6}+i-(q'(e_4-e_6)+q'(e_5-e_6))\big) &= 0,\\
\gamma\,\big(G(e_7-e_{11})+j_{C5}+j_{B1}+q'(e_7-e_9)\big) &= 0,\\
\gamma\,\big(G(e_8-e_{11})+j_{C6}+j_{B2}+q'(e_8-e_9)\big) &= 0,\\
\gamma\,\big(j_{E1}+j_{E2}+i-(q'(e_7-e_9)+q'(e_8-e_9))\big) &= 0,\\
\gamma\,\big(C\,e'_{10}-j_{v_2}-3i\big) &= 0,\\
e_{12}+v_1 &= 0,\\
\gamma_2\,(G(6e_{11}-e_1-e_2-e_4-e_5-e_7-e_8)-j_L) &= 0,\\
-j_{v_1}+j_{v_2} &= 0,\\
\gamma_3\,\big(Lj'_L+e_{11}\big) &= 0,\\
e_{12}-e_{10}-\beta\,(e_9-e_{10}) &= 0,
\end{aligned}
\tag{6.9}
$$

with q, v_{BE}, v_{BC}, j_{BE}, j_{BC}, j_B, j_C, α, α_r, v_T, G, I_S, v, v_B, i, and L as before. Further we have $\beta = 0.1$, $C = 2\cdot 10^{-11}$. Scale factors γ, γ_2, and γ_3 are also used to improve the condition of the problem.

The dynamic $d(x,t)$ of this system is defined by:

$$d\,(x,t) = \begin{pmatrix} q(e_1 - e_3) \\ q(e_2 - e_3) \\ q(e_4 - e_6) \\ q(e_5 - e_6) \\ q(e_7 - e_9) \\ q(e_8 - e_9) \\ j_L \\ e_{10} \end{pmatrix},$$

according to the differential terms q', j_L', and e_{10}', respectively, from (6.9). We choose the following trajectory $\bar{x}(t)$:

$$\bar{x}(t) = \begin{pmatrix} \kappa\,\cos(t) \\ \kappa\,\cos(t) \\ \cos(t) \\ \kappa\,\cos(t) \\ \kappa\,\cos(t) \\ \cos(t) \\ \kappa\,\cos(t) \\ \kappa\,\cos(t) \\ \cos(t) \\ \cos(t) \\ \cos(t) \\ -1.5 \\ \cos(t) \\ \cos(t) \\ \cos(t) \end{pmatrix},$$

for e_1 to e_{12}, j_L, j_{v_1}, and j_{v_2}, in this order. The constant $\kappa = 1.01$ is also used.

In the following section we will discuss some aspects concerning tests and experiments we have done with each of the examples presented so far.

7 Experimental results

In this section we report several experimental results we have conducted with the examples introduced above. Other theoretical insights have been also presented in [13].

Example 1

In [16] we presented some results for the DAE from (6.1). We analyzed the accuracy of the projectors Q_i, since exact Taylor expansions at the point $t_0 = 1$, for $\eta = 1$, computed from the theoretical expression with *Mathematica*, were available. Our algorithm computes Q_0, Q_1 and Q_2 as given in Example 1 of Section 6.1 with a function β which has the Taylor expansion

$$\beta(t) = 0 + 2(t-1) + (t-1)^2 - \frac{23}{3}(t-1)^3 - \frac{10725}{900}(t-1)^4 + O(t-1)^5$$

at the point $t_0 = 1$, for $\eta = 1$ and $c = 10^{-2}$. The Taylor expansion for

$$Q_{2_{13}} = \frac{1 - \beta\,(x_1 + \eta)}{x_1 + x_2^{'} + \eta}$$

at the point $t = 1$, for $\eta = 1$, computed from its theoretical expression with *Mathematica* is

$$Q_{2_{13}}(t) = 100 + 9598(t-1) + 969399(t-1)^2 + 9.790447\bar{4}3 \cdot 10^7 (t-1)^3 + \\ + 9.887711261\bar{9}16 \cdot 10^9 (t-1)^4 + O(t-1)^5.$$

The Taylor coefficients of $Q_{1_{13}}$ and $Q_{2_{13}}$ computed with our algorithm are presented in Table 1. The computed projectors values agreed with the theoretical

Table 1. Computed Taylor coefficients for $Q_{1_{13}}$ and $Q_{2_{13}}$ and their respective relative errors.

Term	$Q_{1_{13}}$	Rel.err. $Q_{1_{13}}$	$Q_{2_{13}}$	Rel.err. $Q_{2_{13}}$
$(t-1)^0$	0.0	0.0	$9.999999999999321 \cdot 10$	6.790e–14
$(t-1)^1$	2.0000000000000010	5.000e–16	$9.597999999998652 \cdot 10^3$	1.404e–13
$(t-1)^2$	0.9999999999999998	2.000e–16	$9.693989999997970 \cdot 10^5$	2.094e–13
$(t-1)^3$	–7.6666666666666780	1.478e–15	$9.790447433330607 \cdot 10^7$	2.785e–13
$(t-1)^4$	–11.9166666666667400	6.153e–15		

ones with working accuracy[2] and our algorithm improved the performance of the algorithm presented in [9]. We obtained accurate results, especially around the points where the index might vary (i.e., around the DAE singular points). Not only are the derivatives obtained with machine precision, but also the determination of the index does not suffer from problem dependent singularities, at least very close to the singular points.

Even at the singular point, when $c = 0$ (for $t = 1$ and $\eta = 1$), our algorithm accurately identifies it in a closer neighborhood. This is not the case of the algorithm from [9] where the value of c should be set to $3 \cdot 10^{-5}$ to allow successfully comparing pivot elements in the QR factorization. With a value smaller than $3 \cdot 10^{-5}$, neither the singular point is identified by that algorithm nor the index is correctly computed.

Our algorithm behaves much better. We studied with which precision the singular point could not be found any more. To this end we tried with different values for the constant c and decremented them systematically. Table 2 shows the obtained results. We found that, for a value of c greater or equal than $3.4 \cdot 10^{-16}$, the singular point is always identified and the index is correctly computed (i.e., the index is 3). For smaller values of c, however, the index is said to be 3 when it should be actually 2 in such cases. Even up to 10^{-64}, our algorithms correctly detects the singularity.

[2] Each differentiation reduces the relevant Taylor coefficients in one term. We considered a number of five Taylor coefficients in the computations. This is why the last values for $Q_{2_{13}}$ are not shown in the Table.

Table 2. Threshold's assessment in a neighborhood of the singular point of the DAE from (6.1).

Value of c	Last pivot element that holds the condition when applying Householder to $\tilde{Z}_i^-$	Rank	Singular point identified?
10^{-5}	$1.00000000021172 \cdot 10^{10}$	3	
10^{-6}	$1.000000002347046 \cdot 10^{12}$	3	
10^{-7}	$1.000000027468525 \cdot 10^{14}$	3	
10^{-8}	$1.000000165636668 \cdot 10^{16}$	3	y
10^{-9}	$1.0000019869449 \cdot 10^{18}$	3	
10^{-10}	$1.000021456205492 \cdot 10^{20}$	3	e
10^{-11}	$1.00017219237835 \cdot 10^{22}$	3	
10^{-12}	$1.002183093489067 \cdot 10^{24}$	3	s
10^{-13}	$1.016062706685979 \cdot 10^{26}$	3	
10^{-14}	$1.139904008044362 \cdot 10^{28}$	3	
10^{-15}	$3.549874073494557 \cdot 10^{30}$	3	
$5 \cdot 10^{-16}$	$1.577721810442028 \cdot 10^{30}$	3	y
$3.5 \cdot 10^{-16}$	$1.577721810442028 \cdot 10^{30}$	3	e
$3.4 \cdot 10^{-16}$	$1.577721810442028 \cdot 10^{30}$	3	s
$3.3 \cdot 10^{-16}$	1.0,	2	y
10^{-32}	with rank of $\tilde{Z}_i = 1$	2	e
10^{-64}	$\neq$ nr. of columns of $\tilde{Z}_i = 2$	2	s

Example 2

For the second academic example we conducted some experiments to test the robustness of the algorithm we propose, as well as to measure the programm running time including the computation of derivatives with ADOL-C. For this purpose we defined matrices where each element is a polynomial with more than 10000 Taylor coefficients. We should emphasize that the degree of the Taylor expansions that is needed to compute the index is at most equal to the dimension of the DAE, as we will address in short. With the experiments we just wanted to work with big matrices and to check whether the algorithm worked as expected.

Some of the results we obtained have been published in [16]. In short: the algorithm correctly computes the index (i.e. it is 3 for the DAE from (6.2) with $\alpha = x_1$, $p(t) = e^{10t}$ and $t = 1$) and the overall computation time does not exceed a second for over 500 Taylor coefficients. For more than 10000 coefficients the computation time was slightly greater than 6 minutes. Furthermore, the computation effort grows theoretically quadratically (see Figure 7.1) but over the range of practical interest only linearly with respect to the degree, i.e., with respect to the number of Taylor coefficients.

We also conducted one experiment to analyze the memory requirements of ADOL-C, i.e., the size of the tapes used by ADOL-C for evaluating the underlying functions and their derivatives. The size of a tape depends on the program code segment that is to be automatic differentiated.

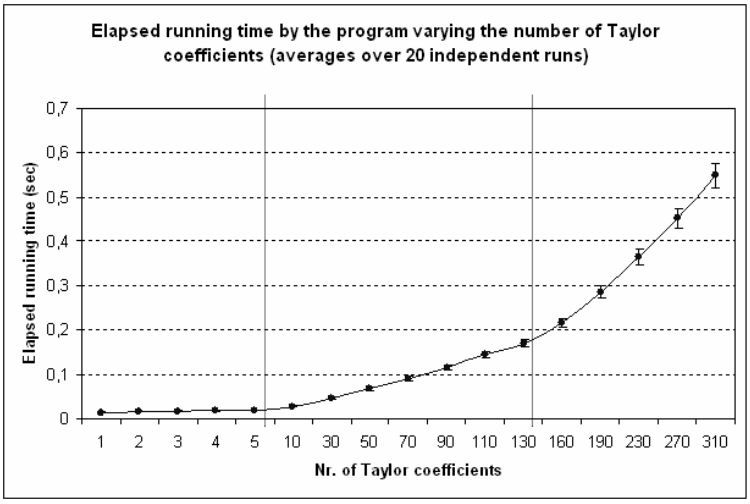

Fig. 7.1. Programm running time when varying the number of Taylor coefficients.

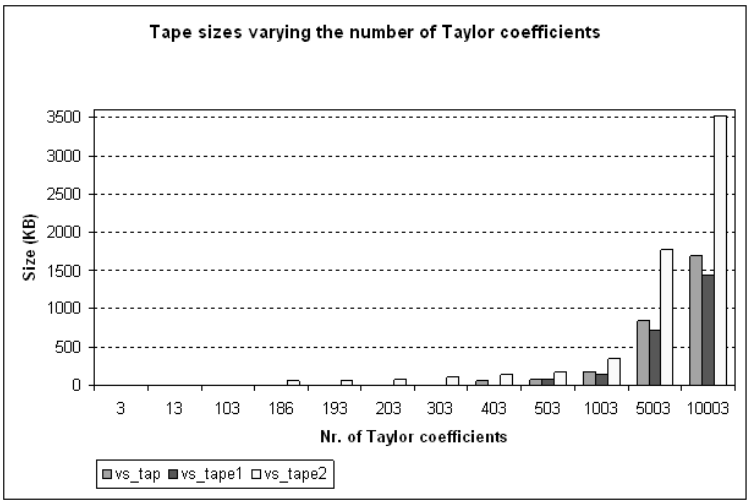

Fig. 7.2. Programm running time when varying the number of Taylor coefficients.

Figure 7.2 shows that for a large number of Taylor coefficients the size of the tapes remains acceptable: about 3500 KB for more than 10000 Taylor coefficients.

Example 3

Another known example of DAE is the one from (6.3) describing a classical pendulum. This DAE has index 3. A critical situation arises when the direction of motion reverses, for instance, when the pendulum is in vertical position and the velocity is zero, because the DAE has a singular point there and the computation of the index may have difficulties. Our algorithm however detects without any problem the singularities of the DAE and can correctly compute the index even at these points.

Example 4

The DAE from (6.4) that describes the motion of a robotic arm is an example of a high index DAE. Computing the index of high index DAEs could be very difficult.

When applying our algorithms and once the projectors are computed as described in Section 3.3, there should be analyzed their properties. To check out projector properties like $Q_{i+1}Q_i = 0$, $Q_i^2 = Q_i$, $G_iQ_i = 0$, and $P_0 = D^-D$ we use the *Frobenius norm* defined as the square root of the sum of the absolute squares of the matrix elements. Since the elements of these matrices are vectors of Taylor coefficients, we consider the Frobenius norm either concerning the first Taylor coefficients, i.e., the evaluation of the corresponding polynomials, or all the Taylor coefficients as well. The decision is up to the user, which defines a particular parameter for that. No matter what the case is, the Frobenius norm should not be greater than a threshold $eps \in \mathbb{R}$ given by the user; otherwise the calculation of the projectors might have problems.

We did some tests for a threshold $eps = 10^{-14}$ and we defined the number of Taylor coefficients to be 6, i.e., we deal with polynomials up to grade 5 in the case of the robotic arm DAE.

When running the program with the above settings, it prints out some warning errors inside the matrix loop. They indicate that the calculation of the projectors might have problems because at least for one Taylor coefficient does not hold that $\|Q_i^2 - Q_i\|_F < eps$, being i the iterations concerning the matrix sequence. However, as it can be seen in Figure 7.3, the values of the norm are nevertheless very small.

We obtain similar results for the Frobenius norm $\|G_iQ_i\|_F$, where at least for one Taylor coefficient the norm value is greater that eps (see Figure 7.4). Again, the values are acceptable.

Not only our algorithm correctly computes the index at problem singularities (for example, when $t = 0$) but the results remain accurate when we vary the number of Taylor coefficients. This is the kind of test the next experiment shows. We expected that the accuracy of the involved matrices depends on the degree of the Taylor expansions used in the computations, i.e., we should obtain more accurate results when we define more Taylor coefficients before truncation. In general, the number of Taylor coefficients NTC can be calculated using the following formula:

$$NTC = (index - 1) + 2,$$

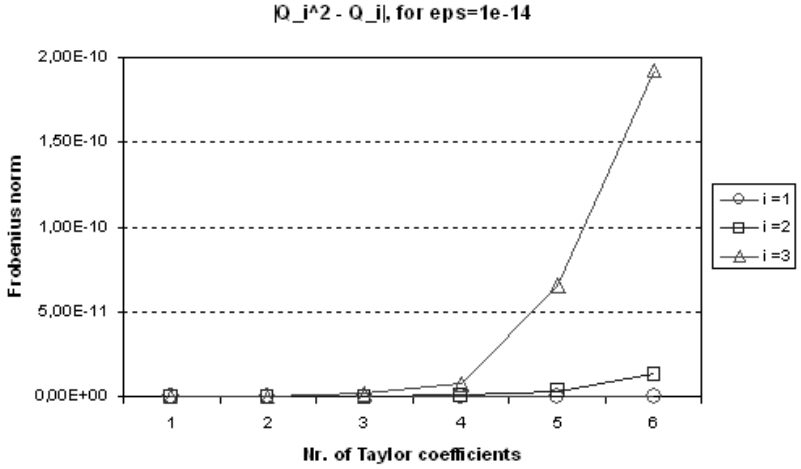

Fig. 7.3. Frobenius norm of $Q_i^2 - Q_i$.

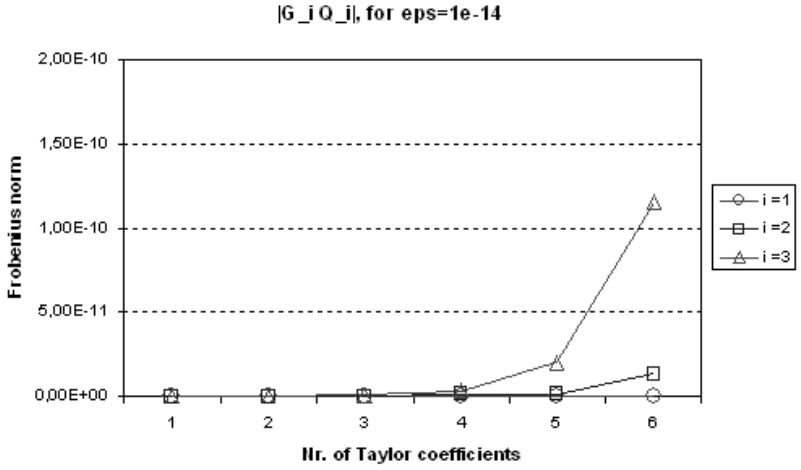

Fig. 7.4. Frobenius norm of $G_i Q_i$.

where *index* is the index of the system (if it is known). It ensures a good precision after truncating the univariate Taylor expansion. At most the index is equal to the dimension of the DAE, what would be an upper bound.

We varied the number of Taylor coefficients from 1 to 7 in the robotic arm example and analyzed the Frobenius norm of the derivative $(D\,P_{i+1}\,D^{-})'$ that is computed in order to obtain the matrix B_{i+1} from (3.14). Actually, any variation on the number of Taylor coefficients would only affect the calculation of $(D\,P_{i+1}\,D^{-})'$ since it is computed by shifting the Taylor coefficients of the polynomials of each element in the involved matrices (and by doing some multiplications with them after multiplying the matrices). Hence we wanted to see whether the further calculation of the matrix sequence would be also affected by this variation.

The number of Taylor coefficients does not have a direct effect on the results accuracy since the Householder QR with column pivoting just considers the column norms of the matrices, calculated these with the first coefficients of the involved Taylor series, which correspond to the evaluation of the function (as it was already explained in Section 5.1). So it is enough to consider an appropriate number for the degree of Taylor polynomials to deal with, i.e., at most the DAE's dimension in case NTC above cannot be previously determined.

Examples 5 and 6

The DAEs of the electromechanical problems from Section 6.4, i.e., the ones modeling the bike dynamo and the plate-type capacitor, have index 4 and 3, respectively.

For the bike dynamo we simplified the DAE from (6.5) making C, R, G, k_0, k, m, and r equal to 1, and eliminating the factors with value 2π. Furthermore we set $\alpha = 10°$. For the plate-type capacitor we set the values of m, L, A, K and l_0 to 1 in (6.6).

We should stress that for these practical problems there was not always possible to determine the index at problem singularities by using other methods [1]. The algorithm we propose, however, computes the index satisfactorily for both ones, even at problem singularities as in the case of $t = 0$ for the bike dynamo.

Examples 7, 8 and 9

In the case of the circuit simulations from Section 6.5 we analyzed the DAEs of three bipolar ring oscillators. These DAEs have index 1, 2, and 3, respectively, that depend on the topology of the electrical circuit. The complexity of the circuit and of the DAE increase when particular transformations to the topology are considered. These effects should be taken into account in the simulation of real chips. The so called *crosstalk effect*, for example, is quite important in practice; however, it has not been resolved satisfactorily from the numeric point of view.

We had manually to tune the scale factors of these three examples until we obtained reasonable results. The final values are shown in Table 3. The more complex the problem is, the more difficult to tune the scale factors are. Our program correctly computed the index for all cases. Special attention deserved Example 9, the bipolar ring oscillator with cross-modulation. It is a case of an ill-conditioned system with a high condition number, with solutions very sen-

Table 3. Settings for the scale factors in the circuit simulations (Examples 7, 8, and 9).

Scale factor	BRO DAE from (6.7)	BRO-LI DAE from (6.8)	BRO-LI-CV DAE from (6.9)
γ	10^9	10^6	10^9
γ_2	–	–	10^3
γ_3	–	–	10^6

sitive to small changes in the data. There we have encountered problems with numerical conditioning. This is why we are still investigating that issue and have considered to introduce some changes in the mathematical algorithm to compute the index (although the ill-conditioning does not depend on the algorithm or floating point accuracy used but on the properties of the involved matrices). Our idea is to obtain projectors with better characteristics that allow us to apply further the Linear Algebra functionalities we have already implemented to operate over matrices of Taylor series. These are topics of the ongoing research work.

8 Conclusions and Outlook

We have presented a new approach to compute the tractability index based on Algorithm Differentiation capabilities in the context of DAEs. Our algorithm correctly computes the index very fast and without any human intervention, disregarding problem singularities even in quite high index DAEs.

We have developed and implemented a new matrix-algebra package with operations to deal with AD (e.g. implementation of special matrix-matrix multiplications, QR decomposition of matrices of Taylor polynomials, and much more) using operator overloading in C++. We now have a novel program and a library for the index determination with truncated polynomial arithmetic that we have tested on various examples. Our implemented algorithm is also the background for the extension of our method to compute consistent initial values.

The main achievements we have obtained are the ones concerning accuracy and runtime efforts: they are entirely satisfactory. No truncation errors are present because we now have exact differentiations without explicit specification of derivatives expressions. Furthermore, the algorithm's complexity is quadratic in degree: the computational effort grows theoretically quadratically but over the range of practical interest only linearly with respect to the degree, i.e., with respect to the number of Taylor coefficients. The results are obtained with working accuracy and are compared to exact calculations.

We also have documented the project, in two ways: internally, i.e., by including comments in the code explaining the code to the reader, and externally, by creating an on-line documentation browser (in HTML) and an off-line reference manual (in LaTeX) from the documented source files. The former easily helps understanding and maintaining the software. The latter provides a more com-

pleted view of the whole project because almost all global variables, functions, and classes are there documented.

The work in progress includes the extension of our method for computing consistent initial values from time-invariant linear problems to general smooth DAEs [12].

References

1. E. Abram. Netzwerkbasierte Analyse von elektromechanischen DAE-Systemen. Diplomarbeit, Institut für Mathematik, Technische Universität Berlin, 2008.
2. C. Bendtsen and O. Stauning. FADBAD, a flexible C++ package for Automatic Differentiation. Technical Report IMM–REP–1996–17, IMM, Dept. of Mathematical Modelling, Technical University of Denmark, August 1996.
3. A. De Luca and A. Isidori. Feedback Linearization of Invertible Systems. *2^{nd} Colloq. Aut. & Robots*, Duisburg 1987.
4. G. H. Golub and Ch. F. Van Loan. *Matrix Computations.* The John Hopkins University Press, Baltimore, MD, USA, 3^{rd} edition, 1996.
5. A. Griewank and A. Walter. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* SIAM, second edition, 2008.
6. M. Günther and U. Feldmann. CAD based electric circuit modeling in industry I: Mathematical structure and index of network equations. *Surv. Math. Ind.*, 8:97–129, 1999.
7. D. König. Indexcharakterisierung bei nichtlinearen Algebro-Differentialgleichungen. Master's thesis, Institut für Mathematik, Humboldt-Universität zu Berlin, 2006.
8. P. Kunkel, V. Mehrmann, and I. Seufer. GENDA – Homepage. http://www.math.tu-berlin.de/numerik/mt/NumMat/Software/GENDA/.
9. A. Lamour. Index Determination and Calculation of Consistent Initial Values for DAEs. *Computers and Mathematics with Applications*, 50:1125–1140, 2005.
10. R. Lamour, R. März, and C. Tischendorf. *Proyector-Based DAE Analysis.* (In preparation), 2009.
11. R. Lamour, R. Mattheij, and R. März. On the Stability Behaviour of Systems obtained by Index Reduction. *Journal of Comp. and Applied Math.*, 56:305–319, 1994.
12. R. Lamour and F. Mazzia. Computation of consistent initial values for properly stated index 3 DAEs. In *Int. Conf. on Scientific Computation and Differential Equations, SciCADE'07*, Saint-Malo, France, 2007.
13. R. Lamour and D. Monett. Index Determination of DAEs – A Wide Field for Automatic Differentiation. In T.E. Simos, G. Psihoyios, and Ch. Tsitouras, editors, *International Conference on Numerical Analysis and Applied Mathematics*, volume 2 of *AIP Conference Proceedings 1168*, pages 727–730, Rethymno, Crete, Greece, September 2009.
14. R. März. The index of linear differential algebraic equations with properly stated leading terms. In *Result. Math.*, volume 42, pages 308–338. Birkhäuser Verlag, Basel, 2002.
15. R. März. Differential Algebraic Systems with Properly Stated Leading Term and MNA Equations. In K. Antreich, R. Bulirsch, A. Gilg, and P. Rentrop, editors, *Modeling, Simulation and Optimization of Integrated Circuits, International Series of Numerical Mathematics*, volume 146, pages 135–151. Birkhäuser Verlag, Basel, 2003.
16. D. Monett, R. Lamour, and A. Griewank. Index Determination in DAEs Using the Library *indexdet* and the ADOL–C Package for Algorithmic Differentiation. In C. H. Bischof, H. M. Bücker, P. Hovland, U. Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, volume 64 of *Lecture Notes in Computational Science and Engineering*, pages 247–257, Berlin, 2008. Springer.
17. N.S. Nedialkov and J.D. Pryce. Solving Differential-Algebraic Equations by Taylor Series (I): Computing Taylor coefficients. *BIT Numerical Mathematics*, 45:561–591, 2005.
18. N.S. Nedialkov and J.D. Pryce. Solving Differential-Algebraic Equations by Taylor Series (II): Computing the System Jacobian. *BIT Numerical Mathematics*, 47:121–135, 2007.
19. N.S. Nedialkov and J.D. Pryce. Solving Differential-Algebraic Equations by Taylor Series (III): the DAETS Code. *Journal of Numerical Analysis, Industrial and Applied Mathematics*, 1(1):1–30, 2007.

20. J. D. Pryce. A Simple Structural Analysis Method for DAEs. *BIT*, 41(2):364–294, 2001.
21. G. Reissig, W.S. Martinson, and P. I. Barton. Differential-Algebraic Equations of Index 1 may have an arbitrarily high Structural Index. *SIAM J. Sci. Comput.*, 21(6):1987–1990, 2000.
22. C. Tischendorf. *Coupled systems of differential algebraic and partial differential equations in circuit and device simulation. Modeling and numerical analysis.* Habilitationsschrift, Institut für Mathematik, Humboldt-Universität zu Berlin, 2004.
23. A. Walther and A. Griewank. *ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++, Version 2.0.0*, December 2008.