Index Determination in DAEs Using the Library `indexdet` and the ADOL-C Package for Algorithmic Differentiation

Dagmar Monett[1], René Lamour[2], and Andreas Griewank[2]

[1] DFG Research Centre MATHEON, Humboldt-Universität zu Berlin, Unter den Linden 6, D–10099 Berlin, Germany,
`monett@math.hu-berlin.de`

[2] Institute of Mathematics, Humboldt-Universität zu Berlin, Unter den Linden 6, D–10099 Berlin, Germany,
`[lamour,griewank]@math.hu-berlin.de`

Summary. We deal with differential algebraic equations (DAEs) with properly stated leading terms. The calculation of the index of these systems is based on a matrix sequence with suitably chosen projectors. A new algorithm based on matrices of Taylor polynomials for realizing the matrix sequence and computing the index is introduced. Derivatives are computed using algorithmic differentiation tools.

Key words: Differential algebraic equations, tractability index, ADOL-C

1 Introduction

We have developed a new program, `daeIndexDet`, for the index determination in DAEs by using algorithmic differentiation (AD) techniques. `daeIndexDet` stands for *Index Determination in DAEs*. It uses the `indexdet` library, also implemented by the authors, which provides the utilities for the index computation. Both `daeIndexDet` and `indexdet` are coded in C++.

The main advantages of the `indexdet` library are: the index calculation is based on a matrix sequence with suitably chosen projectors by using AD techniques [4]; the evaluation of all derivatives uses the C++ package ADOL-C [15]. ADOL-C provides drivers that compute convenient derivative evaluations with only a few modifications to the original C++ code, thereby profiting from operator overloading features. The source code describing the functions to be derived requires the use of the special variable type `adouble` instead of `double`. We include classes for implementing Taylor arithmetic functionalities. Basic operations with and over both Taylor polynomials and matrices of Taylor polynomials, as well as Linear Algebra functions (several matrix multiplications and the QR factorization with column pivoting being the most relevant ones) that complement the index determination are provided by overloading built-in operators in C++. Furthermore, we extend the exception handling mechanisms from C++ to provide a robust solution to the shortcomings of traditional error handling methods, like those that may occur during execution time when computing the index. Apart from some predefined exception types from C++, we introduce a class, `IDException`, to define and handle new exception objects typical of the index computation.

Solving DAEs by Taylor series using AD has already been addressed in [2, 5, 11, 12, 13], to name only a few. Our work differs from others in the way we determine the index for nonlinear DAEs. It is based on the tractability index concept, which uses a matrix sequence from a linearization of the DAE along a given function, and it does not need derivative arrays. For example, Nedialkov and Pryce use the DAETS solver for DAE initial value problems and the AD package FADBAD++ [1] for doing AD. DAETS is based on Pryce's structural analysis [14]. FADBAD is a C++ package for AD that uses operator overloading.

2 Index Determination in DAEs

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

$$f((d(x(t),t))',x(t),t)=0, \qquad t\in I, \tag{1}$$

with $I\subseteq\mathbb{R}$ being the interval of interest. A detailed analysis on how to compute the tractability index of these DAEs is addressed in [8, 9, 10]. We will give a brief introduction. The tractability index concept of (1) is based on a linearization of (1) along a given trajectory $x(t)$. Such a linearization looks like

$$A(t)(D(t)x(t))'+B(t)x(t)=q(t) \tag{2}$$

with coefficients

$$A(t):=\left(\frac{\partial f}{\partial z}(z(t),x(t),t)\right), \qquad B(t):=\left(\frac{\partial f}{\partial x}(z(t),x(t),t)\right), \qquad \text{and} \qquad D(t):=\left(\frac{\partial d}{\partial x}(x(t),t)\right),$$

and $z(t)=d'(x(t),t)$ being the derivative of the dynamic d. The matrix functions

$$A(t)\in\mathbb{R}^{n\times m}, \qquad B(t)\in\mathbb{R}^{n\times n}, \qquad \text{and } D(t)\in\mathbb{R}^{m\times n}$$

are supposed to be continuous.

The computation of the index is based on a matrix sequence for given coefficients $A(t)$, $B(t)$, and $D(t)$. By forming the sequence of matrices, suitably chosen projectors are computed using generalized inverses:

$$\begin{aligned} G_0 &:= AD,\\ B_0 &:= B,\\ G_{i+1} &:= G_i+B_iQ_i=(G_i+W_iB_0Q_i)(I+G_i^-B_iQ_i),\\ B_{i+1} &:= (B_i-G_{i+1}D^-(DP_0\dots P_{i+1}D^-)'DP_0\dots P_{i-1})P_i, \end{aligned} \tag{3}$$

where Q_i is a projector function such that im $Q_i=$ ker G_i. $P_i:=I-Q_i$ and W_i are projector functions such that ker $W_i=$ im G_i. Further, D^- denotes the reflexive generalized inverse of D and G_i^- the reflexive generalized inverse of G_i such that $G_i^-G_i=P_i$ and $G_iG_i^-=I-W_i$. The projectors Q_i play an important role in the determination of the tractability index.

Definition 1. *[8] An equation* (2) *with properly stated leading term is said to be a regular index μ DAE in the interval I, $\mu \in \mathbb{N}$, if there is a continuous matrix function sequence* (3) *such that*

(a) G_i has constant rank r_i on I,
(b) the projector Q_i fulfills $Q_iQ_j = 0$, with $0 \leq j < i$,
(c) $Q_i \in C(I, \mathbb{R}^{n\times n})$ and $DP_0 \dots P_i D^- \in C^1(I, \mathbb{R}^{m\times m})$, $i > 0$, and
(d) $0 \leq r_0 \leq \dots \leq r_{\mu-1} < n$ and $r_\mu = n$.

Since the index computation depends on the derivatives $(DP_0 \dots P_{i+1}D^-)'$, these should be computed as accurately as possible. Projector properties like $Q_iQ_j = 0$, $Q_i^2 = Q_i$ or $G_iQ_i = 0$ help in verifying the performance as well as the accuracy of the algorithm.

2.1 Algorithm for Computing the Index

In [7], an algorithm is proposed to realize the matrix sequence and to finally compute the index. It computes the numerical approximations of the continuous matrix functions $A(t)$, $B(t)$, and $D(t)$ by the MATLAB routine *numjac*. The time differentiations needed in the matrix sequence (i.e. to calculate B_{i+1} in (3)) are also computed via *numjac*. The reflexive generalized inverses D^- and G_i^- are computed by singular value decompositions (SVD) of D and G_i, respectively. The projectors are computed using the generalized inverses. The matrix sequence is computed until the matrix G_{i+1} in (3) is nonsingular.

We propose a new algorithm to compute the tractability index of DAEs introducing the following features: The approximations of the matrices $A(t)$, $B(t)$, and $D(t)$ are computed using specific drivers from the C++ package ADOL-C. Furthermore, the time differentiations to compute B_{i+1} in (3) are realized via a *shift operator* over Taylor series, i.e., no more calls to a differentiation routine are needed. 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. This allows to compute the derivative $(DP_0 \dots P_{i+1}D^-)'$ in (3) only by shifting Taylor series coefficients and by doing some multiplications. In particular, to operate over matrices of Taylor polynomials we apply the Taylor coefficient propagation by means of truncated polynomial arithmetic from [4] (see Sect. 10.2). We 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 [3].

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, which are less expensive than SVD. The function that implements QR follows the Algorithm 5.4.1 from [3] and makes use of Householder reflections.

In the rest of this article we define the DAE, the dynamic, and the trajectory as follows. The function $f : \mathbb{R}^{m_{dyn}} \times \mathbb{R}^{m_{tra}} \times \mathbb{R} \to \mathbb{R}^{m_{dae}}$ defines the DAE, the function $d : \mathbb{R}^{m_{tra}} \times \mathbb{R} \to \mathbb{R}^{m_{dyn}}$ defines the dynamic, and the function $x : \mathbb{R} \to \mathbb{R}^{m_{tra}}$ defines the trajectory, $m_{tra} = m_{dae}$ being the number of dependent variables. The trajectory depends only on the independent variable t. The algorithm determines the tractability index for a given trajectory x at a fixed point t_0.

3 Program for Computing the Index and a Related Library

Figure 1 shows a general schema with the most important libraries and files we use in the index determination. The libraries `adolc.lib` and `indexdet.lib` provide the functionalities

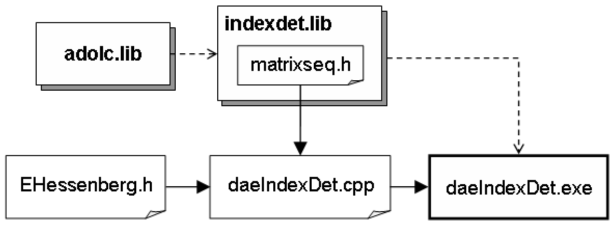

Fig. 1. The main libraries and files needed for the index determination.

for algorithmic differentiation and index determination, respectively. We use the former, the `adolc.lib` library, for the evaluation of derivatives using the C++ package ADOL-C. It can be downloaded from the ADOL-C's web site [15]. We provide the latter, the `indexdet.lib` library, for the index calculation based on the matrix sequence with suitably chosen projectors as it was already introduced in Sect. 2. Its code will be free as soon as we will have successfully finished both its implementation and testing.

The user header (i.e. `EHessenberg.h` in Fig. 1) contains the DAE whose index should be computed. That header also contains the dynamic and the trajectory. In other words, the user should provide the functions x, $d(x,t)$, and $f(z,x,t)$ in a C++ class that implements the abstract base class `IDExample` , which in turn is coded in the C++ header `IDExample.h` (provided in the library `indexdet.lib`). A user header has the following general structure (with `EHessenberg.h` used as example):

```
/** File EHessenberg.h */
#ifndef EHESSENBERG_H_
#define EHESSENBERG_H_
#include "IDExample.h"    // Abstract base class.

class EHessenberg : public IDExample {
    public:
        /** Class constructor. */
        EHessenberg( void ) : IDExample( 3, 4, 1.0, "EHessenberg" )
        { }

        /** Definition of the trajectory x. */
        void tra( adouble t, adouble *ptra )
        { //... }

        /** Definition of the dynamic d(x,t). */
        void dyn( adouble *px, adouble t, adouble *pd )
        { //... }

        /** Definition of the DAE f(z,x,t). */
        void dae( adouble *py, adouble *px, adouble t, adouble *pf )
        { //... }
};

#endif /*EHESSENBERG_H_*/
```

Note that the class constructor for the user class `EHessenberg` does not have any parameter. However, it calls the class constructor of the abstract base class `IDExample` with specific parameters: The first parameter corresponds to the number of dependent variables of the dynamic $d(x(t),t)$ and its type is `int`. It must be equal to the number of equations that define the dynamic (i.e. $m_{dyn} = 3$ in the second example from Sect. 4. The second parameter corresponds to the dimension of the DAE. Its type is also `int`. Its value must be equal to the number of equations that define the DAE, as well as equal to the number of equations that define the trajectory $x(t)$ (i.e. $m_{dae} = m_{tra} = 4$). The third parameter is a `double` that indicates the point at which the index should be computed, i.e., the value for the independent variable time (e.g. $t_0 = 1.0$). The fourth and last parameter is a character string used to denote the output files with numerical results. For example, the name of the class "*EHessenberg*" might be used.

The program `daeIndetDet` works with some global parameters. These parameters are declared in the header file `defaults.h`, provided in the library `indexdet.lib`. They have default values and allow to define the degree of Taylor coefficients or highest derivative degree, the threshold to control the precision of the QR factorization with column pivoting, the threshold to control the precision of I/O functionalities, as well as print out parameters to control the output. When necessary, the user can provide other values. This is the only information, in addition to the definition of the trajectory, the dynamic, and the DAE, that is required from the user.

3.1 Algorithmic Differentiation Using ADOL-C

As mentioned in Sect. 1, we use the C++ package ADOL-C to evaluate derivatives. This is why the vector functions related to the user's problems are written in C++.

In particular, specific ADOL-C drivers (i.e. the functions `forward` and `reverse` for the ADOL-C forward and reverse modes, respectively) are used to compute the Taylor coefficients of both the dependent and the independent variables of the DAE, the dynamic, and the trajectory, respectively. The Taylor coefficients are used in the construction of the matrices $A(t)$, $B(t)$ and $D(t)$ (see Sect. 2). The general procedure is shown in Fig. 2. The DAE, the dynamic, and the trajectory are evaluated via function pointers to the member functions that are coded by the user in its header class as addressed above.

4 Examples

This section introduces two academic examples to test various problem dependent features as well as the program functionality.

Example 1 ([6])

$$\begin{aligned} x_2' + x_1 - t &= 0, \\ x_2' + x_3' + x_1 x_2 - \eta x_2 - 1 &= 0, \\ x_2\left(1 - \frac{x_2}{2}\right) + x_3 &= 0, \end{aligned} \tag{4}$$

with $\eta \in \mathbb{R}$. By considering the differential terms that appear in the first two equations of this DAE we define the dynamic function, $d(x(t),t)$, as follows:

$$\begin{aligned} d_1(x(t),t) &= x_2, \\ d_2(x(t),t) &= x_2 + x_3. \end{aligned} \tag{5}$$

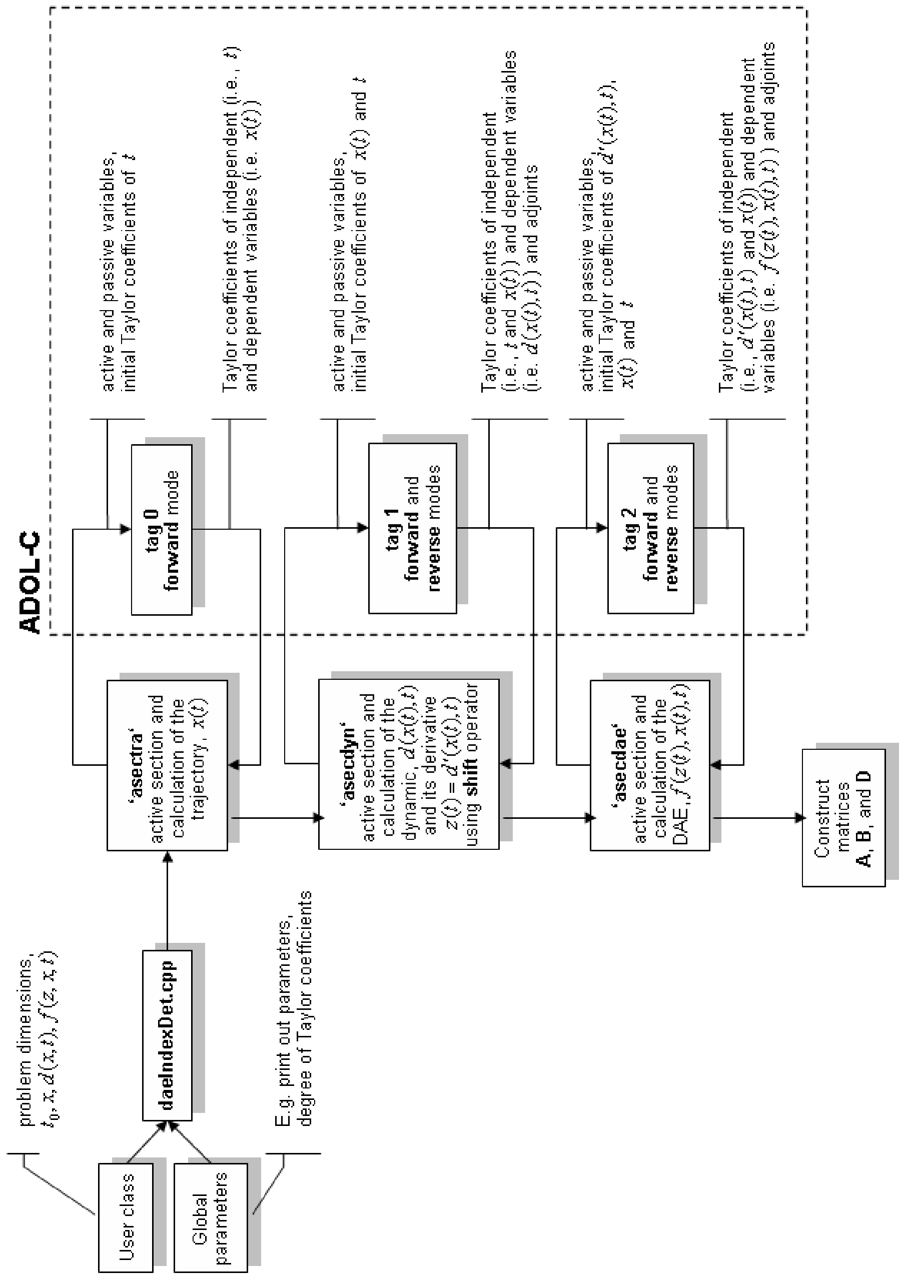
ADOL-C
User class
Global parameters
problem dimensions, $t_0, x, d(x,t), f(z,x,t)$
E.g. print out parameters, degree of Taylor coefficients
daeIndexDet.cpp
'asectra' active section and calculation of the trajectory, $x(t)$
'asecdyn' active section and calculation of the dynamic, $d(x(t),t)$ and its derivative $z(t) = d'(x(t),t)$ using shift operator
'asecdae' active section and calculation of the DAE, $f(z(t),x(t),t)$
Construct matrices A, B, and D
tag 0 forward mode
tag 1 forward and reverse modes
tag 2 forward and reverse modes
active and passive variables, initial Taylor coefficients of t
Taylor coefficients of independent (i.e., t) and dependent variables (i.e. $x(t)$)
active and passive variables, initial Taylor coefficients of $x(t)$ and t
Taylor coefficients of independent (i.e., t and $x(t)$) and dependent variables (i.e. $d(x(t),t)$) and adjoints
active and passive variables, initial Taylor coefficients of $d'(x(t),t)$, $x(t)$ and t
Taylor coefficients of independent (i.e., $d'(x(t),t)$ and $x(t)$) and dependent variables (i.e. $f(z(t),x(t),t)$) and adjoints

Fig. 2. Computing Taylor coefficients using ADOL-C.

We compute the linearization (2) along the trajectory[3]:

$$\begin{aligned} x_1(t) &= t + c, \\ x_2(t) &= 2 - 2e^{t-1}, \\ x_3(t) &= \log(t+1), \end{aligned} \tag{6}$$

with $c \in \mathbb{R}$. The DAE has index 3 when $x_1 + x_2' + \eta = 0$. The computation of the index depends on the derivative x_2'. Choosing $\eta = 1$ and $t_0 = 1$ we obtain a singular matrix chain for $c = 0$ because of $x_1 + x_2' + \eta = t_0 + c - 2 + 1 = c$. In this case, the index is not defined.

Example 2

$$\begin{aligned} x_1' + x_1 + x_4 &= 0, \\ x_2' + \alpha(x_1, x_2, x_3, t)x_4 &= 0, \\ x_3' + x_1 + x_2 + x_3 &= 0, \\ x_3 - p(t) &= 0, \end{aligned} \tag{7}$$

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 and dimension 4. The function $d(x(t),t)$ defines the dynamic:

$$\begin{aligned} d_1(x(t),t) &= x_1, \\ d_2(x(t),t) &= x_2, \\ d_3(x(t),t) &= x_3. \end{aligned} \tag{8}$$

Let the function $x : \mathbb{R} \to \mathbb{R}^4$ define the chosen trajectory:

$$\begin{aligned} x_1(t) &= \sin(t), \\ x_2(t) &= \cos(t), \\ x_3(t) &= \sin(2t), \\ x_4(t) &= \cos(2t). \end{aligned} \tag{9}$$

Coding the DAEs, the trajectories, and the dynamics for both examples into their respective headers (i.e., coding the functions `tra`, `dyn`, and `dae` already introduced in Sect. 3) is straightforward. Besides the value of the point t at which the indexes should be computed and the expressions for the functions α and p (for the second example), no other information is needed from the user.

5 Experiments

We have conducted several experiments to study both the performance and robustness of our algorithm by measuring the computation time, the memory requirements, and the accuracy of the results. The experiments were performed on both a laptop PC with AMD Athlon™XP 1700+ main processor, 1.47 GHz, 512 MB RAM, Microsoft Windows XP Professional Version 2002, MinGW32/MSYS, gcc v. 3.4.2 (MinGW-special), and a desktop PC with Intel™Pentium™4 main processor, 2.8 GHz, 100 GB RAM, Linux version 2.6.5-7.276.smp, and gcc v.3.3.3 (SuSE Linux).

[3] We recall that the function $x(t)$ is not a solution of (1).

The calculation of the projector Q_2 from Example 1 above depends on the calculation of both projectors Q_0 and Q_1. These projectors can be computed exactly. For example, the element $Q_{2,13}$ (projector Q_2, first row, third column) has the following expression:

$$Q_{2,13} = \frac{1-\beta\,(x_1+\eta)}{x_1+x_2'+\eta}, \tag{10}$$

where β is an at least once differentiable function that appears in Q_1 (in particular, $Q_{1,13}=\beta$) and $x_1+x_2' \neq -\eta$. The function β has the following exact Taylor expansion at the point $t_0=1$, for $\eta=1$ and $c=10^{-2}$

$$\beta(t) = 0+2(t-1)+(t-1)^2-\frac{23}{3}(t-1)^3-\frac{10725}{900}(t-1)^4+O(t-1)^5. \tag{11}$$

The Taylor expansion for $Q_{2,13}$ at the point $t=1$, for $\eta=1$, computed from its theoretical expression with Mathematica is

$$\begin{aligned} Q_{2,13}(t) = {} & 100+9598(t-1)+969399(t-1)^2+9.7904474\overline{3}\cdot 10^7(t-1)^3+ \\ & +9.88771126191\overline{6}\cdot 10^9(t-1)^4+O(t-1)^5. \end{aligned} \tag{12}$$

The Taylor coefficients of $Q_{1,13}$ and $Q_{2,13}$ computed with our algorithm are presented in Table 1. The computed values agree with the theoretical ones with high accuracy. Concerning

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	$9.886594928579903\cdot 10^9$	1.129e–4

accuracy, our algorithm improves the performance of the algorithm presented in [7]. We have obtained more accurate results, especially around the points where the index might vary (i.e., around the DAE singular points). Not only are the derivatives calculated 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 $\eta=1$), the algorithm from [7] has to set the threshold to compare pivot elements in the QR factorization to $3\cdot 10^{-5}$. With a threshold smaller than this value neither the singular point is identified nor the index is correctly computed. Instead, our algorithm works well with a threshold value up to 10^{-12}. Furthermore, the singular point is accurately identified in a closer neighborhood and the index is always correctly computed (i.e., it is equal to 3).

The computation time or program running time concerns the computation of derivatives, as well as the running time of the algorithm that determines the index. For Example 2 and for $\alpha=t$ and $p(t)=1$, we can observe a slow quadratic growth as the number of Taylor coefficients increases (see Fig. 3). The computation time starts rising from 30 Taylor coefficients

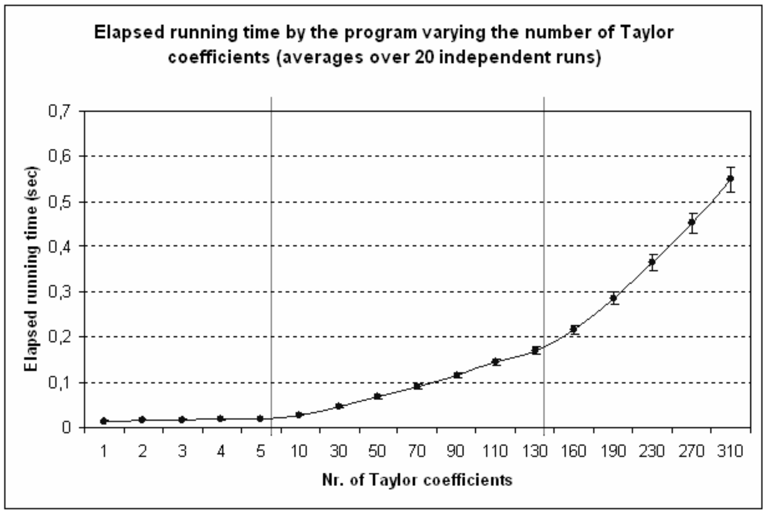

Fig. 3. Program running time for Example 2.

onwards. It its worthwhile noticing that, even for calculations with over 190 Taylor coefficients, the overall computation time does not exceed a second. To test the robustness of the algorithm and the time dependence when varying the number of Taylor coefficients, which are the main goals of this experiment, computations were performed for more than 10000 Taylor coefficients. The computation time in these cases was slightly greater than 6 minutes.

The memory requirements for the last example show that for a large number of Taylor coefficients the size of the ADOL-C tapes remains acceptable (about 3500 KB for 10003 Taylor coefficients).

6 Conclusions

We presented a new algorithm for the index computation in DAEs. It successfully applies AD techniques for calculating derivatives using new matrix-algebra operations especially developed for matrices of Taylor polynomials. By applying our algorithm we obtained more accurate results for some academic examples that were previously computed with a finite-differences-based approach. Both, algorithm performance and robustness were tested for thousands of Taylor coefficients. Last, but not least, we implemented a user friendly C++ algorithm to be used as an experimental tool for accurate index computation and we extensively documented all programs, headers, and classes.

Our current work concentrates on consistent initialization and the specific AD functionalities. Future work will center on comparing the accuracy to already existing techniques for index calculation as well as applications to more realistic problems.

References

1. Bendtsen, C., Stauning, O.: FADBAD, a flexible C++ package for Automatic Differentiation. Tech. Rep. IMM–REP–1996–17, IMM, Dept. of Mathematical Modelling, Technical University of Denmark (1996)
2. Chang, Y.F., Corliss, G.F.: ATOMFT: Solving ODEs and DAEs using Taylor series. Computers & Mathematics with Applications **28**, 209–233 (1994)
3. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3^{rd} edn. The John Hopkins University Press, Baltimore, MD, USA (1996)
4. Griewank, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. No. 19 in In Frontiers in Applied Mathematics. SIAM, Philadelphia, PA (2000)
5. Hoefkens, J., Berz, M., Makino, K.: Efficient high-order methods for ODEs and DAEs. In: G. Corliss, F. Ch., A. Griewank, L. Hascoët, U. Naumann (eds.) Automatic Differentiation of Algorithms: From Simulation to Optimization, Computer and Information Science, chap. 41, pp. 343–348. Springer, New York, NY (2002)
6. König, D.: Indexcharakterisierung bei nichtlinearen Algebro-Differentialgleichungen. Master's thesis, Institut für Mathematik, Humboldt-Universität zu Berlin (2006)
7. Lamour, A.: Index Determination and Calculation of Consistent Initial Values for DAEs. Computers and Mathematics with Applications **50**, 1125–1140 (2005)
8. März, R.: The index of linear differential algebraic equations with properly stated leading terms. In: Result. Math., vol. 42, pp. 308–338. Birkhäuser Verlag, Basel (2002)
9. März, R.: Differential Algebraic Systems with Properly Stated Leading Term and MNA Equations. In: K. Antreich, R. Bulirsch, A. Gilg, P. Rentrop (eds.) Modeling, Simulation and Optimization of Integrated Circuits, International Series of Numerical Mathematics, vol. 146, pp. 135–151. Birkhäuser Verlag, Basel (2003)
10. März, R.: Fine decoupling of regular differential algebraic equations. In: Result. Math., vol. 46, pp. 57–72. Birkhäuser Verlag, Basel (2004)
11. Nedialkov, N., Pryce, J.: Solving Differential-Algebraic Equations by Taylor Series (I): Computing Taylor coefficients. BIT Numerical Mathematics **45**, Springer (2005)
12. Nedialkov, N., Pryce, J.: Solving Differential-Algebraic Equations by Taylor Series (II): Computing the System Jacobian. BIT Numerical Mathematics **47**, Springer (2007)
13. Nedialkov, N., Pryce, J.: Solving Differential-Algebraic Equations by Taylor Series (III): the DAETS Code. Journal of Numerical Analysis, Industrial and Applied Mathematics **1**(1), Springer (2007)
14. Pryce, J.D.: A Simple Structural Analysis Method for DAEs. BIT **41**(2), 364–294 (2001)
15. Walther, A., Kowarz, A., Griewank, A.: ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++, Version 1.10.0 (2005)