SOLVING DIFFERENTIAL-ALGEBRAIC EQUATIONS BY TAYLOR SERIES (I): COMPUTING TAYLOR COEFFICIENTS *

Nedialko S. Nedialkov[1][†] and John D. Pryce[2][‡]

[1] *Department of Computing and Software, McMaster University, Hamilton, Ontario, L8S 4L7, Canada. email: nedialk@mcmaster.ca*
[2] *Computer Information Systems Engineering Department, Cranfield University, RMCS Shrivenham, Swindon SN6 8LA, UK. email: pryce@rmcs.cranfield.ac.uk*

Abstract.

This paper is one of a series underpinning the authors' DAETS code for solving DAE initial value problems by Taylor series expansion. First, building on the second author's structural analysis of DAEs (BIT 41 (2001) 364–394), it describes and justifies the method used in DAETS to compute Taylor coefficients (TCs) using automatic differentiation. The DAE may be fully implicit, nonlinear, and contain derivatives of order higher than one. Algorithmic details are given.

Second, it proves that either the method succeeds in the sense of computing TCs of the local solution, or one of a number of detectable error conditions occurs.

AMS subject classification: 34A09, 65L80, 65L05, 41A58.

Key words: Differential-algebraic equations (DAEs), structural analysis, Taylor series, automatic differentiation.

1 Introduction.

1.1 DAETS and the problem it handles.

The Taylor series expansion approach to ordinary differential equation (ODE) initial value problems (IVPs) has a long pedigree. Pryce's structural analysis [28] shows how the Taylor approach may be extended to differential-algebraic equations (DAEs). Advances in programming languages and automatic differentiation (AD) tools imply the symbolic preprocessing involved in Taylor methods no longer requires special-purpose packages or languages, and can be included seamlessly when coding in a general-purpose language. The authors' DAETS code solves IVPs by this method and is written in standard `C++`. This paper is one of a series on the theory underpinning DAETS.

For suitable problems, especially at high accuracy, Taylor series ODE methods can be orders of magnitude faster than traditional multistep or Runge–Kutta methods. They are also the basis of current *validated* ODE solvers, which produce rigorous bounds on the numerical solution [16, 23, 24]. Both facts motivated us in developing DAETS.

Two features mark out DAETS from most DAE codes. First, the differentiation operator d/dt has equal status with $+, \times$ and other algebraic functions, so that a system can be coded (in C++) in a form close to its natural mathematical formulation[1]. Second, Pryce's approach has no restriction on the DAE index. Numerical results for one high-index problem are given in Section 7.

We consider a fully implicit DAE initial-value problem comprising n equations $f_i = 0$ in n dependent variables $x_j = x_j(t)$, with t a scalar independent variable. In DAETS, the f_i can be arbitrary expressions built from the x_j and t using $+, -, \times, \div$, other standard functions, and $' = d/dt$. We informally write

$$(1.1) \qquad f_i(\, t, \text{ the } x_j \text{ and derivatives of them}\,) = 0, \quad 1 \leq i \leq n.$$

This encompasses such expressions as $f_1 = \big(x_2 \sin((tx_1)')\big)'$, which written with its arguments would be $f_1(t, x_1, x_1', x_1'', x_2, x_2')$. The f_i must be sufficiently smooth, see Theorem 3.1. We assume for simplicity they are defined by straight-line code without branches and loops — though DAETS allows these, provided the structural analysis is independent of the control path taken.

In exact arithmetic the method implemented by DAETS succeeds, roughly speaking, for any DAE whose sparsity structure correctly represents its mathematical structure: false in general, but true for many standard DAE forms, see [28, Theorem 5.3]. We prove a complementary result: if a certain matrix is nonsingular, the method has succeeded. Hence failure is detectable in practice.

The numerical results in Section 7 show DAETS can be very accurate and efficient, and particularly suitable for problems that are of too high an index for existing methods and solvers.

1.2   Background.

Taylor series solution of ODEs. Automatic computation of Taylor series to solve ODE IVPs has been known since the 1950's. Moore [22, pp. 107–130] presents an approach for efficient generation of Taylor coefficients (TCs). Barton, Willers, and Zahar [3] give one of the first such methods for IVPs for ODEs. Rall [29] details AD algorithms, generation of TCs, and application to ODEs. Griewank [12, chapter 10] describes automatic computation of TCs and solving IVPs for ODEs, and discusses briefly the extension to a DAE of the form $F(z(t), z'(t)) = 0$.

Packages for generating TCs include TADIFF [4], FADBAD++ [30], and ADOL-C [13]. Established codes for generating TCs, and solving ODEs thereby,

---

[1]We thank Ole Stauning for augmenting his FADBAD++ [30] tool with a d/dt operator. Without it, our approach is harder to implement. Packages such as ADOL-C [13] and TAYLOR [18] do not provide a d/dt.

include COSY [5], ATOMFT [7], TAYLOR [18] and Taylor Center [10]. All these in effect use a special-purpose language to formulate the ODE.

DAE structural analysis. The structural analysis of Pantelides [26] is essentially equivalent to Pryce's, but applies only to DAEs containing at most first derivatives, that is $f(t, x, x') = 0$. Campbell's *derivative array equation* theory [6] is more widely applicable but more complex, and harder to apply automatically to program code. More detailed comparisons are given in [28].

Taylor series solution of DAEs. Chang and Corliss [7] show how to generate Taylor series for the simple pendulum DAE (2.7), but in an *ad hoc* way. Corliss and Lodwick [8] extend this to validated solution of (2.7).

Pryce [27] outlines a prototype implementation of the present approach. A Prolog preprocessor converts an algebraic description of the DAE into a code list, which is read and interpreted by a MATLAB code that solves the DAE.

Hoefkens [14] uses Pryce's structural analysis to convert the DAE to an ODE, which is solved using Taylor series and the COSY package [5].

Barrio *et al.* [1, 2] apply Taylor methods to very high accuracy solution of dynamical system ODEs, and to DAEs using Pryce's approach. The implementation combines the Mathematica symbolic system and Fortran 77.

We believe the present paper is the first systematic description of methods for computing TCs for a general DAE — high-index, high-order, fully-implicit; and that the "nonsingularity implies validity" results of Section 5.2 are new.

1.3 Structure of this paper.

Section 2 summarizes the main steps of Pryce's structural analysis. Section 4 shows how TCs for the equations f_i can be generated, and how they are organized to find those of the x_j. It explains what a consistent point is, and describes the general method for computing TCs for a DAE. Section 3 derives needed smoothness results. Section 5 shows that hidden cancellations in the expressions (1.1) defining the DAE cannot cause the method to compute a solution when none exists. This is done in two stages: Subsection 5.1 investigates the effect of "non-canonical problem offsets" on the process of computing TCs; Subsection 5.2 shows cancellations have the effect of producing non-canonical offsets. The overall algorithm is given in Section 6. Section 7 gives numerical results from DAETS. Conclusions and notes on future work are in the final Section 8.

2   Outline of Pryce's structural analysis.

We present the main steps of Pryce's structural analysis [28].

The following definitions are needed. A *transversal* T of an $n \times n$ matrix (σ_{ij}) is a set of n positions in the matrix with one entry in each row and each column. That is, T is a set $\{(1, j_1), (2, j_2), \ldots, (n, j_n)\}$ where $(j_1, \ldots, j_n)$ are a permutation of $(1, \ldots, n)$. The *value* of T is $\operatorname{Val} T = \sum_{(i,j) \in T} \sigma_{ij}$.

Given a DAE in the form of (1.1), we perform the following steps.

1. Form the $n \times n$ *signature matrix* $\Sigma = (\sigma_{ij})$, where

$$\sigma_{ij} = \begin{cases} \text{order of the derivative to which the } j\text{th variable } x_j \\ \text{occurs in the } i\text{th equation } f_i; \text{ or} \\ -\infty \text{ if } x_j \text{ does not occur in } f_i. \end{cases}$$

2. Find a *highest value transversal* (HVT), which is a transversal T that makes $\operatorname{Val} T$ as large as possible. The value of a HVT is also, by definition, the *value of the signature matrix*, written $\operatorname{Val} \Sigma$.

   The value of any transversal, and of Σ, is either an integer or $-\infty$. The DAE is *structurally regular* if $\operatorname{Val} \Sigma$ is finite: that is, if there exists at least one transversal all of whose σ_{ij} are finite. Otherwise the DAE is *structurally ill-posed* — there is probably some error in problem formulation.

3. Find n-dimensional integer vectors $\mathbf{c}$ and $\mathbf{d}$, with all $c_i \geq 0$, that satisfy

   (2.1) $d_j - c_i \geq \sigma_{ij}$ for all $i, j = 1, \ldots, n$ and
   (2.2) $d_j - c_i = \sigma_{ij}$ for all $(i, j) \in T$.

   By [28, Lemma 3.3], if a transversal T and vectors $\mathbf{c}, \mathbf{d}$ are found such that (2.1) and (2.2) hold, then necessarily T is a HVT. Summing (2.2) over T gives an alternative formula for $\operatorname{Val} \Sigma$:

   $$\sum_{j=1}^{n} d_j - \sum_{i=1}^{n} c_i = \operatorname{Val} T = \operatorname{Val} \Sigma.$$

   From this follows for any $\mathbf{c}$ and $\mathbf{d}$:

   (2.3)   If (2.1, 2.2) hold for some HVT, then (2.2) holds for any HVT.

   We refer to $\mathbf{c}$ and $\mathbf{d}$ as the *offsets* of the problem. They are never unique. As discussed later, it is advantageous to choose the *smallest* or *canonical* offsets, smallest being in the sense of $\mathbf{a} \leq \mathbf{b}$ if $a_i \leq b_i$ for each i. Our method works correctly, however, when $\mathbf{c}$ and $\mathbf{d}$ are not canonical.

   Step 2 is a Linear Assignment Problem (LAP), a form of a linear programming problem. Step 3 defines its dual. The two formulae for $\operatorname{Val} \Sigma$ instance the fact that primal and dual have the same optimal value.

4. Form the $n \times n$ System Jacobian matrix

   (2.4) $$\mathbf{J} = \frac{\partial \left(f_1^{(c_1)}, \ldots, f_n^{(c_n)} \right)}{\partial \left(x_1^{(d_1)}, \ldots, x_n^{(d_n)} \right)}.$$

   By results in [28], (2.4) has the equivalent reformulations:

   (2.5) $$\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j^{(d_j - c_i)}} = \begin{cases} \dfrac{\partial f_i}{\partial x_j^{(\sigma_{ij})}} & \text{if } d_j - c_i = \sigma_{ij} \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

5. Seek values for the x_j and for appropriate derivatives, consistent with the DAE in the sense of Section 4.4, and at which $\mathbf{J}$ is nonsingular. If such values are found, they define a point through which there is locally a unique solution of the DAE. In this case we say the method "succeeds".

When the method succeeds:

- Val Σ equals the number of *degrees of freedom* (DOF) of the DAE, that is the number of independent initial conditions required.

- An upper bound for the differentiation index ν_d, see [6], of the DAE is given by the *Taylor index*

$$
(2.6) \qquad \nu_T = \max_i c_i + \begin{cases} 1 & \text{if some } d_j \text{ is zero,} \\ 0 & \text{otherwise.} \end{cases}
$$

In many cases, $\nu_T = \nu_d$.

Algorithms for steps 1 and 4 are discussed in the companion paper [25]. Steps 2 and 3 can be carried out by solving a suitable LAP as covered in [28] and briefly in Section 6. Step 5, the main topic of this paper, is covered in Section 4.

EXAMPLE 2.1. Throughout this paper, we give examples based on the simple pendulum, a DAE of differentiation-index 3. Though this system is small, solving it with our method displays almost all the algorithmic features. It is:

$$
(2.7) \qquad \begin{aligned} 0 &= f = x'' + x\lambda \\ 0 &= g = y'' + y\lambda - G \\ 0 &= h = x^2 + y^2 - L^2. \end{aligned}
$$

Here gravity G and length L of pendulum are constants, and the dependent variables are the coordinates $x(t)$, $y(t)$ and the Lagrange multiplier $\lambda(t)$.

A signature matrix Σ will be shown by a "tableau", which annotates it with the offsets c_i, d_j and the names of the functions and variables, and marks the positions of a HVT. For (2.7), there are two HVTs, marked $\bullet$ and $\circ$ in the tableau below. The canonical offsets are $\mathbf{c} = (0, 0, 2)$ and $\mathbf{d} = (2, 2, 0)$:

$$
\begin{array}{c|ccc|c}
 & x & y & \lambda & c_i \\
\hline
f & 2^{\bullet} & -\infty & 0^{\circ} & 0 \\
g & -\infty & 2^{\circ} & 0^{\bullet} & 0 \\
h & 0^{\circ} & 0^{\bullet} & -\infty & 2 \\
\hline
d_j & 2 & 2 & 0 &
\end{array}
$$

For this system, (2.5) then gives the system Jacobian

$$
(2.8) \qquad \mathbf{J} = \begin{bmatrix} \partial f/\partial x'' & 0 & \partial f/\partial \lambda \\ 0 & \partial g/\partial y'' & \partial g/\partial \lambda \\ \partial h/\partial x & \partial h/\partial y & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 2x & 2y & 0 \end{bmatrix}.
$$

From (2.6) the Taylor index is 3, agreeing with the differentiation index.

3   A needed smoothness result.

If Taylor coefficients of a solution $\mathbf{x}(t)$ to (1.1) exist to some order — that is, if $\mathbf{x}(t)$ is sufficiently smooth — the methods of Section 4 can compute them. However, DAEs of form (1.1) are not covered by standard theory, so that *a priori* there is no guarantee that solutions exist, let alone are smooth. Existence is proved by [28, Theorem 4.2] — ExT for short. This section extends ExT to a smoothness result.

Let $\mathbf{Y}$ denote the vector of derivatives $x_j^{(d_j)}$ ($j = 1, \ldots, n$) and $\mathbf{X}$ the vector formed by all lower derivatives $x_j^{(l)}$ ($0 \le l < d_j$, $1 \le j \le n$), in some standard sequence that does not matter for the present purpose. The equations $f_i^{(c_i)} = 0$, $i = 1, \ldots, n$, form a vector system

$$(3.1) \qquad\qquad \mathbf{F}(t, \mathbf{X}, \mathbf{Y}) = 0,$$

where the $x_j^{(l)}$ in $\mathbf{X}$ are regarded as independent algebraic variables.[2]

ExT shows, using the Implicit Function Theorem (ImFT), that near a point of $(t, \mathbf{X}, \mathbf{Y})$ space where $\mathbf{J}$ is nonsingular, (3.1) defines a locally unique functional dependence $\mathbf{Y} = \phi(t, \mathbf{X})$. That is,

$$(3.2) \qquad\qquad x_j^{(d_j)} = \phi_j(t, \mathbf{X}) \quad \text{for } j = 1, \ldots, n.$$

The x_j for which $d_j = 0$ ("algebraic" variables) do not appear on the right-hand sides.

Temporarily writing $x_j^{(l)}$ as $x_{j,l}$, one can recast (3.2) as a standard first-order ODE with $\left(\sum_j d_j \right)$ dependent variables $x_{j,l}$ for $0 \le l < d_j$, $j = 1, \ldots, n$. Namely it is the system formed by concatenating, for j such that $d_j > 0$, the equations

$$x'_{j,0} = x_{j,1},$$

$$(3.3) \qquad\qquad \vdots$$

$$x'_{j,d_j-2} = x_{j,d_j-1},$$

$$x'_{j,d_j-1} = \phi_j(t, \mathbf{X}).$$

Once this is solved, each algebraic variable is found explicitly from its corresponding equation (3.2), which for such variables has the form $x_j = \phi_j(t, \mathbf{X})$.

If the system (3.1) is linear in the elements $x_j^{(d_j)}$ of $\mathbf{Y}$, it takes the form $\mathbf{JY} = \mathbf{G}$, where $\mathbf{J}$ and $\mathbf{G}$ depend only on $(t, \mathbf{X})$. This holds when consistent points can live in $(t, \mathbf{X})$ space, rather than $(t, \mathbf{X}, \mathbf{Y})$ space — see Section 4.4 and Lemma 4.1.

ExT states that the solutions of the DAE (1.1) are just the solutions of ODE (3.3) that pass through a consistent point; more precisely, a solution of either can be reconstructed from a solution of the other in an obvious way. This leads to the main result of this Section:

---

[2] In the notation of Section 4, $\mathbf{X}$ is $\mathbf{x}_{J_0}$, $\mathbf{Y}$ is $\mathbf{x}_{J_{\le 0}}$, and (3.1) is the system $\mathbf{f}_{I_0} = 0$.

THEOREM 3.1. *Let* $\mathbf{x}(t)$ *be a solution of (1.1). Let* $(t^*, \mathbf{X}^*, \mathbf{Y}^*)$ *(or* $(t^*, \mathbf{X}^*)$ *in the linear case above) be the consistent point comprising the relevant derivatives of* $\mathbf{x}(t)$ *evaluated at* $t = t^*$. *Suppose* $\mathbf{J}$ *is nonsingular at this point and each function* f_i *in (1.1) has* $(N+c_i)$ *continuous derivatives in a neighbourhood of this point, where* $N \geq 1$.

Then the jth *component* $x_j(t)$ *has* $(N+d_j)$ *continuous derivatives in a neighbourhood of* t^*. *Hence, the algorithm for computing TCs can be carried out up to stage* $k = N$.

PROOF. That the n functions ϕ_j in (3.2) exist, comes from applying the ImFT to the system formed from the n functions $f_i^{(c_i)}$, with the $x_j^{(d_j)}$ as the unknowns. From its definition, $f_i^{(c_i)}$ is a sum of terms each of which is

$$\text{(a partial derivative of } f_i \text{ of order at most } c_i)$$

$$\times \text{ (a product of zero or more } x_j^{(l)}\text{'s).}$$

Hence $f_i^{(c_i)}$ has N continuous derivatives. By a standard corollary of the ImFT, this is true also of the functions ϕ_j, hence of each right-hand side in (3.3).

By a standard ODE result, this implies each component of any solution of (3.3) has $N+1$ continuous derivatives. In particular, each $x_{j,d_j-1}(t) = x_j^{(d_j-1)}(t)$ does so, whence each *differential* variable $x_j(t)$ has $N+d_j$ continuous derivatives. It is easy to extend this to the *algebraic* variables, which completes the proof. □

CHECK. This gives the expected result when the DAE is a first-order explicit ODE system $\mathbf{x}' = \mathbf{g}(t, \mathbf{x})$. In this case, all $c_i = 0$ and all $d_j = 1$. Then, Theorem 3.1 says, correctly, that if $\mathbf{g}$ is C^N then all solutions $\mathbf{x}(t)$ are C^{N+1}.

EXAMPLE 3.1. For the pendulum example, $\mathbf{X}$ is (x, y, x', y'), $\mathbf{Y}$ is (x'', y'', λ). We solve (3.1) — which in this case is linear in $\mathbf{Y}$ — to obtain x'', y'', λ as functions X, Y, Λ of x, y, x', y'. Renaming the latter as $x_0(t), y_0(t), x_1(t), y_1(t)$ gives the ODE (3.3) as

$$\begin{aligned}
x_0' &= x_1, \\
x_1' &= X(x_0, y_0, x_1, y_1), \\
y_0' &= y_1, \\
y_1' &= Y(x_0, y_0, x_1, y_1).
\end{aligned}$$

From a solution of this, one can then obtain $\lambda_0 = \lambda_0(t) = \Lambda(x_0, y_0, x_1, y_1)$.

4   How the method works.

We denote the pth TC of a function u of a real variable t at a point t^* by

$$(u)_p = u^{(p)}(t^*)/p\,!$$

The overall solution process goes in steps over the t range like a typical ODE solver. TCs (scaled by a power of the current step size to eliminate over- or

under-flow problems) are computed to a chosen order at the current t^* . Summing Taylor series with a stepsize h yields approximate values of solution components at the next point $t = t^* + h$, and this process repeats.

Compare a Taylor method for an *explicit* ODE. There, the TCs are *evaluated* in batches of n at a time by the AD methods of the next subsection: the given initial values specify the zero-order TCs $(x_j)_0$, then one computes all the $(x_j)_1$, then all the $(x_j)_2$, and so on. By contrast for the *implicit* DAE (1.1), we must *solve* for the $(x_j)_l$ by regarding the TCs $(f_i)_l$ as algebraic functions of them, equating the latter to zero, and using a numerical root-finding method. The role of the offsets c_i, d_j is to organize this process, as described in Section 4.2.

When solving an ODE, the solution computed at the previous step is used unchanged as the initial point for the current step. For a DAE however, this point is usually inconsistent in the sense that it does not satisfy the algebraic constraints. The solution process takes this point as an initial guess and (provided it converges) produces a consistent point. In effect it projects the initial guess on to the consistent manifold at the start of each step.

Subsection 4.1 shows how expressions for evaluating TCs can be generated. Subsection 4.2 shows how the solution process breaks into stages. Subsection 4.3 defines a compact notation and displays the block-triangular structure of the equations. Subsection 4.4 defines the notion of solutions of a DAE *living* in a certain space, and of *consistent point*. Subsection 4.5 shows how the block-triangular structure is exploited in a numerical method. Each subsection is illustrated by the simple pendulum example.

The companion paper [25] shows how to generate automatically the Jacobians that are used to solve the equations by a Newton-type method.

4.1   *Generating expressions for Taylor coefficients of the equations.*

An expression for evaluating $(f_i)_p$ can be constructed by applying well-known techniques for computing TCs (see the references in Section 1.2). For instance, if sufficient TCs of two functions u and v are known, we can compute the pth TCs for $u + cv$, where c is a constant, $u \cdot v$, u/v and the dth derivative of u by

$$(4.1) \qquad (u + cv)_p = (u)_p + c \cdot (v)_p,$$

$$(4.2) \qquad (u \cdot v)_p = \sum_{r=0}^{p} (u)_r (v)_{p-r},$$

$$(4.3) \qquad (u/v)_p = \frac{1}{(v)_0} \left((u)_p - \sum_{r=0}^{p-1} (v)_{p-r}(u/v)_r \right), \quad (v)_0 \neq 0, \quad \text{and}$$

$$(4.4) \qquad \left(u^{(d)} \right)_p = (p+1)(p+2) \cdots (p+d) \cdot (u)_{p+d}.$$

Similar formulas can be derived for the standard functions [12].

EXAMPLE 4.1.   For the pendulum example (2.7), the TCs will be written without parentheses for brevity: x_p rather than $(x)_p$, etc. Applying (4.1), (4.2),

and (4.4) to (2.7), we obtain for the f equation

$$f_p = (x'')_p + (x\lambda)_p = (p+1)(p+2)x_{p+2} + \sum_{r=0}^{p} x_r \lambda_{p-r},$$

and similarly for the g and h equations. The first three coefficients are:

p	0	1	2
f_p	$1{\cdot}2x_2 + x_0\lambda_0,$	$2{\cdot}3x_3 + x_0\lambda_1 + x_1\lambda_0,$	$3{\cdot}4x_4 + x_0\lambda_2 + x_1\lambda_1 + x_2\lambda_0$
g_p	$1{\cdot}2y_2 + y_0\lambda_0 - G,$	$2{\cdot}3y_3 + y_0\lambda_1 + y_1\lambda_0,$	$3{\cdot}4y_4 + y_0\lambda_2 + y_1\lambda_1 + y_2\lambda_0$
h_p	$x_0^2 + y_0^2 - L^2,$	$2x_0x_1 + 2y_0y_1,$	$2x_0x_2 + x_1^2 + 2y_0y_2 + y_1^2$

Note that for a constant c such as G and L^2, $(c)_0 = c$ and $(c)_p = 0$ for $p > 0$.

4.2 The stages of solving for Taylor coefficients of the solution.

It is not obvious by inspection how the various equations obtained by differentiating the original DAE equations f_i can be organized to solve for the TCs of the state variables x_j. This is made clear by the offsets c_i and d_j, which tell us that the computation forms a sequence of stages indexed by integer k. At stage k we solve a system of equations containing

$$(4.5) \qquad (f_i)_{k+c_i} = 0 \quad \text{for all } i \text{ such that} \quad k + c_i \geq 0$$

to determine values for

$$(4.6) \qquad (x_j)_{k+d_j} \quad \text{for all } j \text{ such that} \quad k + d_j \geq 0.$$

All previously computed $(x_j)_l$ in any equation (4.5) are to be *treated as constants*.

Clearly in the above, each $(f_i)_l$ $(i = 1, \ldots, n;\ l = 0, 1, \ldots)$ is used exactly once and each $(x_j)_l$ $(j = 1, \ldots, n;\ l = 0, 1, \ldots)$ is solved for exactly once. Let

$$k_c = -\max_i c_i \quad \text{and} \quad k_d = -\max_j d_j.$$

Then the set of equations is empty for all $k < k_c$, and the set of unknowns is empty for all $k < k_d$. It follows from the definition (2.1, 2.2) of the offsets that $k_d \leq k_c \leq 0$. Hence the process effectively starts at stage $k = k_d$ and is performed for stages $k = k_d, k_d+1, \ldots$.

Except when (1.1) is purely algebraic with no derivatives at all, one always has $k_d < 0$. The algebraic constraints, and on the first integration step the initial conditions, are applied during stages $k < 0$. It is after stage $k = 0$ that a consistent point has been computed, so this is a good time to output solution values to the user.

Suppose that each function f_i in (1.1) has at least $(N + c_i)$ continuous derivatives, where $N \geq 1$, in a neighbourhood of a point $(t^*, \mathbf{x}^*)$ at which $\mathbf{J}$ is nonsingular. Theorem 3.1 shows that, for any solution $\mathbf{x}(t)$ of (1.1) in a neighbourhood of $t = t^*$, and with $\mathbf{x}(t^*)$ sufficiently near $\mathbf{x}^*$, the jth component of $\mathbf{x}(t)$ has

$(N + d_j)$ continuous derivatives in a neighbourhood of t^*. Hence one can find TCs up to at least stage $k = N$. If the f_i are C^∞, one can find indefinitely many TCs.

EXAMPLE 4.2. For the pendulum, (4.5, 4.6) give the following recipe:

Stage	uses equations	to obtain
$k = -2$	$0 = h_0$	x_0, y_0
$k = -1$	$0 = h_1$	x_1, y_1
$k = 0$	$0 = f_0, g_0, h_2$	x_2, y_2, λ_0
$k = 1$	$0 = f_1, g_1, h_3$	x_3, y_3, λ_1
$\cdots$	$\cdots$	$\cdots$

Thus at stage $k = k_d = -2$, we find x_0, y_0 that satisfy

$$(4.7) \qquad 0 = h_0 = x_0^2 + y_0^2 - L^2.$$

At stage $k = -1$, we find x_1, y_1 that satisfy

$$(4.8) \qquad 0 = h_1 = 2x_0 x_1 + 2y_0 y_1$$

taking the previously computed x_0, y_0 as known. At stage $k = 0$, we find x_2, y_2, λ_0 that satisfy

$$0 = f_0 = 1 \cdot 2x_2 + x_0 \lambda_0$$
$$0 = g_0 = 1 \cdot 2y_2 + y_0 \lambda_0 - G$$
$$0 = h_2 = 2x_0 x_2 + x_1^2 + 2y_0 y_2 + y_1^2$$

taking the previously computed x_0, y_0, x_1, y_1 as known. This forms the linear system

$$(4.9) \qquad 0 = \begin{bmatrix} 2 & 0 & x_0 \\ 0 & 2 & y_0 \\ 2x_0 & 2y_0 & 0 \end{bmatrix} \begin{pmatrix} x_2 \\ y_2 \\ \lambda_0 \end{pmatrix} + \begin{pmatrix} 0 \\ -G \\ x_1^2 + y_1^2 \end{pmatrix} = \mathbf{A}_0 \xi + \mathbf{b}, \text{ say.}$$

Matrix $\mathbf{A}_0$ is a diagonally scaled version of the pendulum's System Jacobian $\mathbf{J}$, given in (2.8). Namely

$$\mathbf{A}_0 = \text{diag}(1, 1, 2)^{-1} \mathbf{J} \text{ diag}(2, 2, 1).$$

Now $\mathbf{J}$ is nonsingular since its determinant is $-2(x_0{}^2 + y_0{}^2) = -2L^2 \neq 0$. Hence (4.9) has a unique solution.

In general, for every stage $k \geq 0$, (4.5) and (4.6) tell us to find $x_{k+2}, y_{k+2}, \lambda_k$ satisfying $f_k, g_k, h_{k+2} = 0$, subject to all the already found values. The resulting system is linear with a matrix $\mathbf{A}_k$ that is a scaled $\mathbf{J}$:

$$\mathbf{A}_k = \begin{bmatrix} (k+1)(k+2) & 0 & x_0 \\ 0 & (k+1)(k+2) & y_0 \\ 2x_0 & 2y_0 & 0 \end{bmatrix}$$

$$= \text{diag}\big(k!, k!, (k+2)!\big)^{-1} \mathbf{J} \text{ diag}\big((k+2)!, (k+2)!, k!\big).$$

Hence $\mathbf{A}_k$ is nonsingular, and we can solve for $\big(x_{k+2}, y_{k+2}, \lambda_k\big)^T$.

4.3   Structure of the equations for Taylor coefficients of the solution.

We start with a compact notation for the equations and the unknowns in (4.5, 4.6). Let $\mathcal{I}$ and $\mathcal{J}$ be index sets labeling the complete set of $(f_i)_l$ and $(x_j)_l$ respectively. For the general case (1.1), we take both $\mathcal{I}$ and $\mathcal{J}$ to be the set of pairs $\{(i,l) \mid i = 1, \ldots, n; \ l = 0, 1, \ldots\}$.

If I is some (in practice finite) subset of $\mathcal{I}$, define $\mathbf{f}_I$ to be the vector function whose components are $(f_i)_l$ as (i,l) ranges over I — in some standard sequence that does not matter at present.

Similarly, if $\mathbf{x}$ is an arbitrary numeric vector indexed over $\mathcal{J}$, and J some subset of $\mathcal{J}$, define $\mathbf{x}_J$ to be the sub-vector of $\mathbf{x}$ whose components are $(x_j)_l$ as (j,l) ranges over J.

The equation system (4.5) to be solved at stage k can then be written as

$$\mathbf{f}_{I_k} = 0 \quad \text{where} \quad I_k = \big\{(i,l) \mid 0 \le l = k + c_i\big\} = \big\{(i,l) \in \mathcal{I} \mid l = k + c_i\big\}$$

whose unknowns are the vector

$$\mathbf{x}_{J_k} \quad \text{where} \quad J_k = \big\{(j,l) \mid 0 \le l = k + d_j\big\} = \big\{(j,l) \in \mathcal{J} \mid l = k + d_j\big\}.$$

In general $\mathbf{f}_{I_k}$ is also a function of the $\mathbf{x}_{J_\kappa}$ for all $\kappa < k$. That is, the complete set of equations and unknowns is organized into the block triangular structure

(4.10)
$$\begin{cases} 0 & = & \mathbf{f}_{I_{k_d}}\big(t, \mathbf{x}_{J_{k_d}}\big), \\ 0 & = & \mathbf{f}_{I_{k_d+1}}\big(t, \mathbf{x}_{J_{k_d}}, \mathbf{x}_{J_{k_d+1}}\big), \\ & \vdots \\ 0 & = & \mathbf{f}_{I_k}\big(t, \mathbf{x}_{J_{k_d}}, \mathbf{x}_{J_{k_d+1}}, \ldots, \mathbf{x}_{J_k}\big), \\ & \vdots \end{cases}$$

To simplify discussing the root-finding process, let $\mathbf{F}_k$ denote $\mathbf{f}_{I_k}$ regarded purely as a function of the stage-k unknowns. That is, for $k = k_d, k_d + 1, \ldots$ we have to solve

(4.11)
$$0 = \mathbf{F}_k(\mathbf{x}_{J_k})$$
$$=_{\text{def}} \mathbf{f}_{I_k}\big(t, \mathbf{x}_{J_{k_d}}, \mathbf{x}_{J_{k_d+1}}, \ldots, \mathbf{x}_{J_k}\big).$$

When we wish to speak of the previously-found $\mathbf{x}_{J_\kappa}$ it is convenient to pack them up into one vector $\mathbf{x}_{J_{<k}}$: this denotes $\big(\mathbf{x}_{J_{k_d}}, \mathbf{x}_{J_{k_d+1}}, \ldots, \mathbf{x}_{J_{k-1}}\big)$.

We also write $\mathbf{x}_{J_{\le k}} = (\mathbf{x}_{J_{<k}}, \mathbf{x}_{J_k})$, so $\mathbf{f}_{I_k}$ with its arguments can be written

$$\mathbf{f}_{I_k}\big(t, \mathbf{x}_{J_{<k}}, \mathbf{x}_{J_k}\big) \quad \text{or} \quad \mathbf{f}_{I_k}\big(t, \mathbf{x}_{J_{\le k}}\big).$$

For each k, let m_k and n_k be the number of equations and unknowns respectively in (4.11). Thus by (4.5, 4.6),

(4.12)
$$m_k \text{ is the number of } i \text{ for which } k + c_i \ge 0; \quad \text{and}$$
$$n_k \text{ is the number of } j \text{ for which } k + d_j \ge 0.$$

It is proved in [28] that, first, the system $\mathbf{F}_k = 0$ is nonlinear in general for $k \leq 0$, but linear for $k > 0$. Second, by a counting argument using (2.1, 2.2),

$$m_k \leq n_k \quad \text{for all } k.$$

Thus $\mathbf{F}_k = 0$ is square ($m_k = n_k$) or underdetermined ($m_k < n_k$) but never overdetermined. It is clear from (4.12) that m_k [resp. n_k] increases from 0 when $k < k_c$ [resp. $k < k_d$] to n when $k \geq 0$, so only the negative stages $k = k_d, \ldots, -1$ can have $m_k \neq n_k$.

Also from [28], the total "amount of underdetermination",

$$D = \sum_{k=k_d}^{-1} (n_k - m_k)$$

is the number of DOF of the system, also given by $\mathrm{Val}\,\Sigma$, see Section 2.

For all $k \geq 0$, the Jacobian $\mathbf{F}'_k$ of $\mathbf{F}_k$ equals the system Jacobian $\mathbf{J}$ up to diagonal scaling (that is, $\mathbf{F}'_k = D_1 \mathbf{J} D_2$ where D_1, D_2 are nonsingular diagonal matrices). Thus $\mathbf{F}'_k$ is nonsingular if $\mathbf{J}$ is nonsingular at the current point.

Finally, for $k < 0$, $\mathbf{F}'_k$, again up to scaling, is the result of deleting those rows i and columns j of $\mathbf{J}$ for which $k + c_i < 0$ and $k + d_j < 0$, respectively. Moreover, $\mathbf{F}'_k$ is of full row-rank if $\mathbf{J}$ is nonsingular.

EXAMPLE 4.3. The pendulum, with $\mathbf{c} = (0, 0, 2)$ and $\mathbf{d} = (2, 2, 0)$, has m_k, n_k as follows:

k	-2	-1	≥ 0
m_k	1	1	3
n_k	2	2	3

Thus the $k = -2$ system (4.7) and the $k = -1$ system (4.8) each introduce one DOF, as we expect, since one position and one velocity condition are required to specify the motion. They both have the same Jacobian, obtained by deleting the λ column and f, g rows of $\mathbf{J}$, namely

$$\mathbf{F}'_{-2} = \mathbf{F}'_{-1} = (2x_0, 2y_0).$$

4.4 The consistent manifold.

For a DAE, not all initial conditions (ICs) are compatible with a solution. An IC that is so is called *consistent*. For a DAE system as general as (1.1) it is not obvious what collection of values of the x_j and their derivatives makes up a valid IC. This subsection makes the notion precise.

A useful comparison is with an explicit ODE, not of the first-order form $\mathbf{x}' = \mathbf{f}(t, \mathbf{x})$ but one with higher orders on the left side, for instance

(4.13)
$$\begin{aligned} u'' &= p(t, u, v), \\ v''' &= q(t, u, v), \end{aligned}$$

with scalar u, v. Then an IC consists of given values for u, u', v, v', v'' at a given t. We say solutions of (4.13) *live in* in (t, u, u', v, v', v'') space in the sense that there is a unique solution through each point of this space (assuming the ODE is defined there, is smooth, etc.). Solutions do *not* live in (t, u, u', v) space, say, because it does not give enough information to specify solutions uniquely.

The following definition applies to points at which our method applies, namely, points that give a nonsingular system Jacobian.

DEFINITION 4.1. *Let J be a set of indices $\big((j_1, l_1), (j_2, l_2), \ldots\big)$ in $\mathcal{J}$. Thus $\mathbf{x}_J = \big((x_{j_1})_{l_1}, (x_{j_2})_{l_2}, \ldots\big)$ represents a list of TCs of the variables. The variable t should be added if the DAE is non-autonomous: that is, if t appears explicitly in it.*

Solutions of the DAE live in $\mathbf{x}_J$ space, or $(t, \mathbf{x}_J)$ space in the non-autonomous case, if there is at most one solution having specified values of these TCs at $t = 0$, or at a specified t in the non-autonomous case.

If solutions live in such a space, a point of the space for which a solution of the DAE exists is a (globally) consistent point. *The set of all consistent points is the* consistent manifold $\mathcal{M}$ *for this space.*

The interesting J's when using our method are $J_{\leq k}$'s as defined above. Namely, the solution is uniquely determined once (4.10) has been solved as far as $k = 0$. This is clear from the fact that for $k > 0$, the equations $\mathbf{F}_k = 0$ are square, linear and nonsingular (it is proved directly in [28]). This is equivalent to saying that any vector $\mathbf{x}_{J_{\leq 0}}$ that satisfies $\mathbf{F}_k = 0$ for $k \leq 0$ is consistent, and that solutions live in $\mathbf{x}_{J_{\leq 0}}$, or $(t, \mathbf{x}_{J_{\leq 0}})$, space.

A very common situation is that the DAE is linear in the derivatives that appear in $\mathbf{x}_{J_0}$, that is the $x_j^{(d_j)}$. The following is easily proved:

LEMMA 4.1. *System $\mathbf{F}_0 = 0$ is linear in the $x_j^{(d_j)}$ iff every f_i, whose offset c_i is zero, is linear in the $x_j^{(d_j)}$.*

In this case, solutions actually live in $\mathbf{x}_{J_{<0}}$, or $(t, \mathbf{x}_{J_{<0}})$, space. This is because then $\mathbf{F}_0 = 0$ is also square, linear and nonsingular, so that the solution is uniquely determined by the stages up to $k = -1$. This situation is detectable during preprocessing and might be used to generate more efficient code, for both operator overloading and source translation implementations. Currently DAETS does not detect such a situation.

EXAMPLE 4.4. (2.7) illustrates the linear case mentioned above. It is linear in the highest derivatives, so already after stages $-2, -1$, a unique solution is specified. Thus solutions can live in $\mathbf{x}_{J_{<0}}$ space, that is in (x, y, x', y') space.

If we change the first equation to $0 = (x'')^2 - (x\lambda)^2$, which does not change the signature matrix or offsets, the DAE is now nonlinear in the highest derivatives. Then for a given solution to stages $k = -2, -1$, there is usually more than one solution to stage $k = 0$. Solutions must now live in $\mathbf{x}_{J_{\leq 0}}$ space, namely $(x, y, x', y', x'', y'', \lambda)$ space.

For the pendulum, if living in (x, y, x', y') space, $\mathcal{M}$ is the solution set of $(0 = h = x^2 + y^2 - L^2, \, 0 = h' = 2xx' + 2yy')$, a 2-dimensional manifold in $\mathbb{R}^4$. If we choose to make the problem live in $(x, y, x', y', x'', y'', \lambda)$ space, $\mathcal{M}$ is

the solution set of $0 = (h, h', f, g, h'')$ — now in $\mathbb{R}^7$, but still 2-dimensional and easily seen to be equivalent to the previous $\mathcal{M}$.

4.5 Numerics of solving for Taylor coefficients of the solution.

At each integration time-step, input data is available in the form of initial "guesses" to solution components. The first step is different from subsequent steps so we discuss them separately. In either case, the guesses are organized as approximations $\mathbf{x}^a_{J_k}$ to the solution of $\mathbf{F}_k(\mathbf{x}_{J_k}) = 0$ in (4.11) for relevant k. At present, DAETS uses the full generality of the IPOPT [31] optimization package in both cases, but it should be possible to perform the general step robustly with a simpler algorithm.

4.5.1 The general step.

Here the guesses are values from advancing the previous step by summing relevant Taylor series. The error control selects a stepsize h to keep the estimated truncation error within a tolerance.

We try to find the closest point (in an appropriate Euclidean norm related to the error estimate criterion) $\mathbf{x}^*_{J_k}$ to $\mathbf{x}^a_{J_k}$ such that $\mathbf{F}_k(\mathbf{x}^*_{J_k}) = 0$ in (4.11) holds. Thus we solve a sequence of constrained optimization problems

$$(4.14) \qquad \min_{\mathbf{x}} \|\mathbf{x} - \mathbf{x}^a_{J_k}\|^2 \quad \text{subject to} \quad \mathbf{F}_k(\mathbf{x}) = 0,$$

to find $\mathbf{x}^*_{J_k}$ for $k = k_d, k_d + 1, \ldots, 0$. The concatenation of these $\mathbf{x}^*_{J_k}$ is a point $\mathbf{x}^*$ on $\mathcal{M}$ in $\mathbf{x}_{J_{\leq 0}}$ space that we may call "stage-wise closest" to the guess.

By the remarks in Subsection 4.3, the Jacobians $\mathbf{F}'_k$ have full row rank. If the error control was successful, the $\mathbf{x}^a_{J_k}$ form a point $\mathbf{x}^a$ sufficiently close to the consistent manifold $\mathcal{M}$ that each problem (4.14) has a unique solution, and convergence is rapid.

Subsequent stages are square and linear, all with a matrix equal to the last value of $\mathbf{J}$ found (at the end of process (4.14)) up to diagonal scaling. Thus they can all be solved with a single LU factorization.

In the common case that the stage 0 equations are linear (but (1.1) as a whole need not be), $\mathcal{M}$ can live in $\mathbf{x}_{J_{<0}}$ space instead, and one can save computation by only running (4.14) as far as $k = -1$.

If any $\mathbf{F}_k$ for $k \leq 0$ is linear — in particular in the important case when (1.1) is entirely linear — (4.14) can be solved more simply, as the minimum-norm solution of an underdetermined linear system if $m_k < n_k$, or a nonsingular linear system if $m_k = n_k$.

Currently DAETS does not have mechanisms to detect such linear cases.

4.5.2 The first step.

At the initial step, the task can also be written in the form of (4.14). If a good guess of a consistent point is available this process will find it. If not, finding a consistent point can be a difficult numerical problem, but our experience is that good optimization software, such as IPOPT, usually succeeds.

In practice the user may want to add extra constraints besides those in (4.14). This may involve fixing some initial values; or imposing further equations on the variables; or imposing upper/lower bounds on variables. For instance in the pendulum problem one might specify the angular position by an equation $c_1 x + c_2 y = 0$, and the velocity by an equation $(x')^2 + (y')^2 = c_3$ together with a bound such as $y' \geq 0$. DAETS does not yet have such a feature but using a standard optimizing code makes it easy to add.

EXAMPLE 4.5. For the pendulum, for $k = -2$, (4.14) is the nonlinear problem to find the closest point $(x_0, y_0)^T$ to $(x_0^a, y_0^a)^T$ that satisfies $0 = h_0 = x_0^2 + y_0^2 - L^2$.

For $k = -1$, it is the linear problem to find the closest $(x_1, y_1)^T$ to $(x_1^a, y_1^a)^T$ that satisfies $0 = h_1 = 2x_0 x_1 + 2y_0 y_1$, with $(x_0, y_0)^T$ known.

5   The method cannot be deceived.

This section shows that an event that looks potentially fatal to the method is in fact not a problem. Namely, the system Jacobian $\mathbf{J}$ is computed from (2.4) using the offsets, which come from the signature matrix Σ, which in turn comes from analysing the program code that defines the DAE functions f_i — the DAE code, built from the input x_j as described in Subsection 1.1 from algebraic functions and $' = d/dt$.

Suppose the code contains something like $w = (x_1' + x_1) - x_1'$ or, less obviously, $w = (x_1 x_2)' - x_1' x_2$. In either case, "formally" computing the order of highest derivative of x_1 in w yields 1, not the correct value 0. By "formal" is meant:

- If a variable w in the code is a purely algebraic function of variables $u, v, \ldots$, then the order of highest derivative of x_j occurring in w is taken to be the *maximum* of the corresponding orders in $u, v, \ldots$;

- If $w = u'$, then the order of highest derivative of x_j occurring in w is taken to be one more than that in u.

When such *hidden cancellation* occurs, the computed $\widetilde{\Sigma}$ may not be the true Σ. No clever symbolic manipulation can overcome the hidden cancellation problem, because the task of determining whether some expression is exactly zero is known to be *undecidable* in any algebra closed under the basic arithmetic operations together with the exponential function.

We show that this essentially does not matter and the approach of deriving Σ formally gives a robust algorithm. As the above examples suggest (and is proved in [25]) any error will be an overestimation, that is $\widetilde{\Sigma} \geq \Sigma$ elementwise.

For the rest of the section it is assumed that

- The computed signature matrix $\widetilde{\Sigma}$ satisfies $\widetilde{\Sigma} \geq \Sigma$.

- The numerical algorithm for $\mathbf{J}$ computes (2.4) exactly up to roundoff error.

Under these assumptions we show that the computed $\mathbf{J}$ *either* is nonsingular, and the method is using valid, but non-canonical, offsets for the true Σ; *or* is structurally singular, so that the method fails in a way that can be detected.

Thus, symbolic simplification may help, converting a structurally singular case to a "success" case. But it is not a prerequisite of making the method robust.

5.1 Valid offsets and corresponding $\mathbf{J}$.

It was not pointed out in [28], but is needed here, that [28, Theorem 4.2] and other results used here *do not require the offsets to be canonical.* That is, they must satisfy the inequality $d_j - c_i \geq \sigma_{ij}$ with equality on some transversal; but they need not be the minimal offsets subject to $c_i \geq 0$, guaranteed by [28, Theorem 3.6] and computed by [28, Algorithm 3.1].

Therefore, Theorem 3.1 holds whether the offsets are canonical or not, provided only that the $\mathbf{J}$ defined by (2.4) is nonsingular. It is possible to have different offsets for the same DAE system. Their effect is to produce the same TCs, but in a different sequence. This leads to the following, where the term "offset vector" is used to refer to the complete list of values $(\mathbf{c}, \mathbf{d}) = (c_1, \ldots, c_n, \ d_1, \ldots, d_n)$

DEFINITION 5.1. *An offset vector for a given signature matrix Σ is called* valid *if and only if*

(i) $d_j - c_i \geq \sigma_{ij}, \ (i, j = 1, \ldots, n)$.

(ii) *Equality holds on some and, by (2.3), on every HVT.*

(iii) $\min_i c_i = 0$. *(This can be assumed without loss, because adding a constant to all the c_i and d_j does not essentially change the solution process.)*

Also define the *essential sparsity pattern* S_{ess} of Σ to be the union of the HVTs of Σ: that is, the set of all (i, j) positions that lie on any HVT.

The situation is summarized by the following result, which underlies the important Theorem 5.2.

THEOREM 5.1. *Suppose one offset vector for the signature matrix of a DAE system (1.1) gives a nonsingular $\mathbf{J}$ as defined by (2.5) at some consistent point. Then all valid offset vectors give a nonsingular $\mathbf{J}$ there, and thus define a solution scheme for the TCs. All resulting $\mathbf{J}$ are equal on S_{ess}, and all have the same value of the determinant $\det \mathbf{J}$.*

PROOF. Equality on S_{ess} comes at once from (2.5) and (2.3).

The determinant of $\mathbf{J}$ can be written as the sum of all products $\mathrm{prod}(T) = \mathrm{sign}(T) \prod_{(i,j) \in T} \mathbf{J}_{ij}$, taken over all $n!$ transversals T, where $\mathrm{sign}(T)$ is ± 1 according as T, regarded as a permutation, is even or odd. Consider any $\mathbf{J}$ and any transversal T. Summing the inequalities $d_j - c_i \geq \sigma_{ij}$ over $(i, j) \in T$ shows that if T is *not* a HVT then σ_{ij} must be strictly less than $d_j - c_i$ for some $(i, j) \in T$. By the definition of $\mathbf{J}$, its (i, j) entry is therefore zero, and the product $\mathrm{prod}(T)$ corresponding to T is also zero.

Thus, only $\mathrm{prod}(T)$'s belonging to HVTs contribute to $\det \mathbf{J}$ — that is, $\det \mathbf{J}$ depends only on those entries of $\mathbf{J}$ that lie within S_{ess}. But all $\mathbf{J}$ are equal on S_{ess}, whence they all have the same value of $\det \mathbf{J}$.

Therefore, if $\mathbf{J}$ is nonsingular for one valid offset vector, it is nonsingular for all. From remarks preceding Theorem 5.1, this implies that all valid offset vectors specify a solution scheme for the TCs. This completes the proof. ☐

EXAMPLE 5.1. The following examples illustrate how much freedom there may be in choosing the offsets, how this affects sequencing the computation of TCs, and how $\mathbf{J}$ varies according to the offsets chosen.

Consider some variants on an ODE system with $n = 2$. The tableau will be shown, and to its right some equations that could give rise to it. Version 1 is

$$(5.1) \qquad \begin{array}{c} \\ f \\ g \\ d_j \end{array} \begin{array}{cc} x & y \\ \begin{pmatrix} 1^\bullet & 1 \\ 1 & 1^\bullet \end{pmatrix} \\ 1 & 1 \end{array} \begin{array}{c} c_i \\ 0 \\ 0 \end{array} \qquad \begin{array}{l} \text{Possible realization} \\ 0 = f(t,\ x, y,\ x', y') \\ 0 = g(t,\ x, y,\ x', y') \end{array}$$

The offsets shown, namely $\mathbf{c} = (0,0)$, $\mathbf{d} = (1,1)$, are clearly the only valid ones. This reflects the fully coupled nature of the equations, which forces the TCs to be obtained in the sequence

Stage	uses equations	to obtain
$k = -1$	none	x_0, y_0
$k = 0$	f, g	x_1, y_1
$k = 1$	f', g'	x_2, y_2
$\ldots$	$\ldots$	$\ldots$

The entry "none" for $k = -1$ signifies that x_0, y_0 are arbitrary initial values. The associated system Jacobian is

$$\begin{bmatrix} f_{x'} & f_{y'} \\ g_{x'} & g_{y'} \end{bmatrix}.$$

Next consider

$$(5.2) \qquad \begin{array}{c} \\ f \\ g \\ d_j \end{array} \begin{array}{cc} x & y \\ \begin{pmatrix} 1^\bullet & 0 \\ 0 & 1^\bullet \end{pmatrix} \\ 1 & 1 \end{array} \begin{array}{c} c_i \\ 0 \\ 0 \end{array} \qquad \begin{array}{l} \text{Possible realization} \\ 0 = f(t,\ x, y,\ x') \\ 0 = g(t,\ x, y,\ y') \end{array}$$

The equations are still implicit, but less tightly coupled than in (5.1). Definition 5.1 gives the constraints

$$d_1 - c_1 = 1,\ d_2 - c_2 = 1; \quad d_2 - c_1 \geq 0,\ d_1 - c_2 \geq 0; \quad \min_i c_i = 0.$$

The tableau (5.2) shows the *canonical* offsets $\mathbf{c} = (0,0)$, $\mathbf{d} = (1,1)$. There are two other possibilities: $\mathbf{c} = (1,0)$, $\mathbf{d} = (2,1)$ and $\mathbf{c} = (0,1)$, $\mathbf{d} = (1,2)$. This reflects the weaker coupling of the equations compared with (5.1), which permits the TCs to be obtained in either of the sequences

Stage	uses eqns	to obtain		Stage	uses eqns	to obtain
$k = -2$	(none)	x_0		$k = -2$	(none)	y_0
$k = -1$	f	x_1, y_0	or	$k = -1$	g	x_0, y_1
$k = 0$	f', g	x_2, y_1		$k = 0$	f, g'	x_1, y_2
$k = 1$	f'', g'	x_3, y_2		$k = 1$	f', g''	x_2, y_3
$\ldots$	$\ldots$	$\ldots$		$\ldots$	$\ldots$	$\ldots$

That is, either of x_k and y_k can "lag" behind the other by up to one stage, but not more. The system Jacobians in each case are tabulated below:

$\mathbf{c}, \mathbf{d}$	$(0,0), (1,1)$	$(1,0), (2,1)$	$(0,1), (1,2)$
$\mathbf{J}$	$\begin{bmatrix} f_{x'} & 0 \\ 0 & g_{y'} \end{bmatrix}$	$\begin{bmatrix} f_{x'} & f_y \\ 0 & g_{y'} \end{bmatrix}$	$\begin{bmatrix} f_{x'} & 0 \\ g_x & g_{y'} \end{bmatrix}$

As a contrast, compare the DAE $0 = f(t, x, x')$; $0 = g(t, x, y, x', y')$, where the reader may verify the constraints

$$d_1 - c_1 = 1, \ d_2 - c_2 = 1; \quad d_1 - c_2 \geq 1; \quad \min_i c_i = 0,$$

whose general solution is $\mathbf{c} = (K, 0)$, $\mathbf{d} = (1 + K, 1)$ for any integer $K \geq 0$. Thus the TCs of the first equation can (if desired) be generated to some given order K, independently of the second equation, and stored. After this the TCs of the second equation can be generated to that same order.

5.2 Nonsingularity implies success.

Under the assumptions at the start of Section 5, the computed signature matrix $\widetilde{\Sigma}$ satisfies $\widetilde{\Sigma} \geq \Sigma$. Let $\widetilde{\mathbf{c}}, \widetilde{\mathbf{d}}$ be any valid offsets defined by $\widetilde{\Sigma}$, and $\widetilde{\mathbf{J}}$ the resulting Jacobian defined by (2.4), equivalently (2.5).

Structural singularity is a property of a matrix that depends on parameters. Here, the DAE code defines each of its variables, hence also $\widetilde{\mathbf{J}}$, as a real function over some domain D in some $\mathbb{R}^M$, a space of certain x_j's and appropriate derivatives. A *structural zero* of $\widetilde{\mathbf{J}}$ is an (i, j) position where $\widetilde{\mathbf{J}}_{ij}$ is identically zero for all $\xi \in D$. Then, $\widetilde{\mathbf{J}}$ is *structurally singular* if every matrix $B \in \mathbb{R}^{n \times n}$, with $B_{ij} = 0$ in $\widetilde{\mathbf{J}}$'s structural zero positions, is singular — equivalently, if every transversal of $\widetilde{\mathbf{J}}$ contains a structural zero. $\widetilde{\mathbf{J}}$ is *structurally nonsingular* otherwise.

We can now prove the main result of this section. In inexact arithmetic, a structural zero may become a non-zero near roundoff level, requiring standard methods of rank estimation [11] to decide which alternative of the Theorem has occurred. Recall $\mathrm{Val}\,\Sigma$ was defined in Section 2 step 2.

THEOREM 5.2. *Let the notation be as at the start of this subsection. In exact arithmetic, just one of two alternatives must occur:*

(i) $\mathrm{Val}\,\widetilde{\Sigma} = \mathrm{Val}\,\Sigma$. *Then every HVT of Σ is a HVT of $\widetilde{\Sigma}$, and $\widetilde{\mathbf{c}}, \widetilde{\mathbf{d}}$ are also valid offsets for Σ. Thus the method computes the correct TCs, but in a possibly different sequence from those of the canonical offsets for Σ; and $\widetilde{\mathbf{J}}$ is the correct System Jacobian for the sequence used.*

(ii) $\mathrm{Val}\,\widetilde{\Sigma} > \mathrm{Val}\,\Sigma$. *In this case $\widetilde{\mathbf{J}}$ is structurally singular.*

PROOF. By definition, $\widetilde{\mathbf{c}}, \widetilde{\mathbf{d}}$ satisfy

(5.3) $$\widetilde{d}_j - \widetilde{c}_i \geq \widetilde{\sigma}_{ij} \geq \sigma_{ij},$$

with $\min \widetilde{c}_i = 0$.

Suppose case (i), $\operatorname{Val}\widetilde{\Sigma} = \operatorname{Val}\Sigma$. Let T be an arbitrary HVT of Σ. Then summing (5.3) over $(i,j) \in T$ gives $\operatorname{Val}\widetilde{\Sigma}$ on the left and $\operatorname{Val}\Sigma$ on the right. Since these are equal, we must have equality term by term: $\widetilde{d}_j - \widetilde{c}_i = \widetilde{\sigma}_{ij} = \sigma_{ij}$ on T. Thus $\widetilde{\mathbf{c}}, \widetilde{\mathbf{d}}$ are valid offsets for Σ in the sense of Definition 5.1. Theorem 5.1 then completes the proof of case (i).

In case (ii), $\operatorname{Val}\widetilde{\Sigma} > \operatorname{Val}\Sigma$. By a similar argument, for any transversal T,

$$\widetilde{d}_j - \widetilde{c}_i > \sigma_{ij} \quad \text{for at least one } (i,j) \in T.$$

For such an (i,j), we see from (2.5) that $\widetilde{\mathbf{J}}_{ij} = \partial f_i / \partial x_j^{(\widetilde{d}_j - \widetilde{c}_i)}$ is a structural zero in exact arithmetic, because σ_{ij} is the highest order of derivative of x_j on which f_i truly depends; cf. (2.5). That is, every transversal contains a structural zero of $\widetilde{\mathbf{J}}$, and $\widetilde{\mathbf{J}}$ is structurally singular. This completes the proof of case (ii). □

EXAMPLE 5.2. The following examples illustrate some ways in which computing a $\widetilde{\Sigma}$, different from Σ, can change the TC calculation.

	v	w	x	y	c_i
f_1	$1^{\bullet}$	—	—	—	0
f_2	1	$1^{\bullet}$	—	—	0
f_3	—	—	$0^{\bullet}$	—	1
f_4	—	—	1	$0^{\bullet}$	0
d_j	1	1	1	0	

Case (a), $\operatorname{Val}\Sigma = 2$

	v	w	x	y	c_i
f_1	$1^{\bullet}$	—	—	—	1
f_2	1	$1^{\bullet}$	—	—	1
f_3	—	1	$0^{\bullet}$	—	1
f_4	—	—	1	$0^{\bullet}$	0
d_j	2	2	1	0	

Case (b), $\operatorname{Val}\widetilde{\Sigma} = 2$

Case (a) shows[3] the true Σ. There is one HVT, marked with asterisks. This tableau represents a system of size 4 with two degrees of freedom (DOF), comprising two independent subsystems of size 2. Assume for illustration that they are a partially implicit ODE system for v and w (2 DOF), and an index-2 DAE system for x and y, (0 DOF), say

$$\begin{array}{ll} 0 = f_1 = v' - g_1(t,v), & \\ 0 = f_2 = w' - g_2(t,v,w,v') & \end{array} \quad \text{and} \quad \begin{array}{l} 0 = f_3 = x - g_3(t), \\ 0 = f_4 = y - g_4(t,x,x'). \end{array}$$

The canonical offsets are shown. The corresponding $\mathbf{J}$ is the identity matrix.

Case (b) shows the *same* system, but with the code written in such a way as to introduce into f_3 an apparent dependence on w'. Write

$$f_3 = x - g_3(t) + \zeta(w'),$$

where $\zeta(a,b,\ldots)$ stands for a generic expression that apparently depends on $a, b, \ldots$ but is identically zero.

The computed $\widetilde{\Sigma}$ is now different from Σ. There is still just one HVT, but the canonical offsets have changed. Indeed, if the w'-dependence were real, this would be an index-3 DAE with 2 DOF. Since $\operatorname{Val}\widetilde{\Sigma} = \operatorname{Val}\Sigma = 2$, by Theorem 5.2, these offsets give a valid TC evaluation scheme for the original problem.

---

[3] "—" denotes $-\infty$.

Now consider further changes:

	v	w	x	y	c_i
f_1	$1^\bullet$	$-$	0°	$-$	1
f_2	1°	$1^\bullet$	$-$	$-$	1
f_3	$-$	1°	$0^\bullet$	$-$	1
f_4	$-$	$-$	1	$0^{\bullet\circ}$	0
d_j	2	2	1	0	

Case (c), $\mathrm{Val}\,\widetilde{\Sigma}' = 2$

	v	w	x	y	c_i
f_1	1	$-$	$-$	$0^\bullet$	0
f_2	$1^\bullet$	1	$-$	$-$	0
f_3	$-$	$1^\bullet$	0	$-$	0
f_4	$-$	$-$	$1^\bullet$	0	0
d_j	1	1	1	0	

Case (d), $\mathrm{Val}\,\widetilde{\Sigma}'' = 3$

Case (c) again shows the *same* system, but with $\zeta(w')$ added to f_3 and $\zeta(x)$ added to f_1. This creates a second transversal of value 2, marked $^\circ$. Since $\mathrm{Val}\,\widetilde{\Sigma}' = 2$, we again have a valid TC evaluation scheme for the original problem — identical in fact with that of case (b).

If the $\zeta(x)$ of case (c) is changed to $\zeta(y)$, case (d) arises. Here, a transversal, and HVT, of value 3 has been created. If the apparent dependencies were real, this would be a fully coupled index-2 DAE system with 3 DOF. Since $\mathrm{Val}\,\widetilde{\Sigma} > \mathrm{Val}\,\Sigma$, Theorem 5.2 shows that we no longer have a valid solution scheme, and that $\widetilde{\mathbf{J}}$ (in exact arithmetic) is structurally singular. Indeed,

$$\widetilde{\mathbf{J}} = \frac{\partial(f_1, f_2, f_3, f_4)}{\partial(v', w', x', y)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\partial g_2/\partial v' & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

6   The complete method.

The overall method for computing TCs for the solution of (1.1) is summarized in the following algorithm.

ALGORITHM 6.1 (TAYLOR COEFFICIENTS).

Input for preprocessing, steps 1,2:
     the code defining the DAE.
Input for run time, steps 3 onwards:
     value of t;
     approximations to the relevant x_j derivatives, $\mathbf{x}_{J_{\leq 0}}$.

1.   Generate code for the System Jacobian as described in [25].
2.   Generate code for computing the TCs $(f_i)_l$
       using the approach of Subsection 4.1.
3.   for stages $k = k_d \ldots 0$
4.   try to solve (4.14)
5.   if failure, exit
6.   Factorize the $\mathbf{J}$ produced at the end of stage $k = 0$.
7.   for $k = 1 \ldots k_{\max}$
8.   solve linear system (4.11), noting that its matrix is a scaled
       version of $\mathbf{J}$, hence using the factorization from step 6.

Some implementation-related remarks follow.

Solving the LAP. An efficient method for the Linear Assignment Problem in Section 2 step 2 is the shortest augmenting path algorithm. DAETS uses the program LAP implementing it [17]. It inputs a cost matrix, here Σ, and outputs a HVT. Then DAETS uses [28, Algorithm 3.1] to find the canonical offsets.

Note that, if the computed HVT contains $-\infty$, Algorithm 6.1 should exit before computing the offsets c_i, d_j. In this case, the DAE is structurally ill-posed.

Stages $k \leq 0$. Currently DAETS always uses IPOPT [31] to solve (4.14). Numerical experience suggests that at integration steps after the first, a simple Gauss-Newton [19] iteration would suffice.

In step 4, the solution of (4.14) fails if a rank-deficient matrix arises. In this case, we exit this algorithm, and know that the System Jacobian is singular. Otherwise, each $\mathbf{J}_k$ would have been of full row rank; see [28, Prop. 4.1].

Stages $k > 0$. After the first step, DAETS uses the (currently) computed TCs as an initial guess for the TCs for the next step. That is, $\mathbf{x}_{J_k}^a$ in (4.11) is set to the computed $\mathbf{x}_{J_k}^*$ on the previous step. This essentially happens automatically since TCs are stored in an array: the values in this array are used as an initial guess and overwritten with the new TCs.

If approximations for the TCs for stages $k > 0$ are not available, we can set them to zero since the equations are linear. But using such approximations has accuracy advantages, since the solution process can be regarded as a step of *iterative refinement* [11] of an already reasonable solution.

Scaled Taylor coefficients. When integrating a DAE using Taylor series, to reduce the possibility of overflows and underflows, it is desirable to compute TCs multiplied by the appropriate power of a stepsize $h = t - t^*$. Instead of computing $(x_j)_l = x_j{}^{(l)}/l!$ and then forming $(x_j)_l\, h^l$, DAETS directly computes the scaled TCs $x_j^{(l)}/l!\, h^l$.

Problem specification. A DAE should be encoded in a form as close as possible to its mathematical formulation. Both source-text translation and operator overloading allow such an encoding. The way it is done in DAETS is shown by example in the next section.

7 Numerical results.

Experience with DAETS suggests the performance of Taylor methods on non-stiff or mildly stiff DAEs is broadly similar to that for ODEs. We give results of running the complete DAETS code. This involves many algorithms beyond the scope of this paper, notably: evaluating Σ and solving the linear assignment problem; finding an initial consistent point; evaluating $\mathbf{J}$; stepsize and order control, and resulting error control; output to the user.

First we show some results from solving the Car Axis problem from the Test Set for IVP Solvers [21] (the former CWI test set [20]). Second, we illustrate DAETS solving an index-5 problem. Third, and of particular relevance to this paper, we show how total work depends on the Taylor order and tolerance.

7.1   Car Axis.

This problem is a moderately stiff DAE of index 3. It is a simple model of a car axis going over a bumpy road. For solution in [21], the original second-order form is converted to a first-order form. In DAETS, we solve directly using the original form

$$
\begin{aligned}
Kp'' &= f(t, p, \lambda), \\
0 &= \phi(t, p),
\end{aligned}
\tag{7.1}
$$

with p, f of dimension 4, and λ, ϕ of dimension 2. Here, $p = (x_l, y_l, x_r, y_r)^T$, and (x_l, x_r) and (y_l, y_r) are the coordinates of the left and right wheels, respectively; $\lambda = (\lambda_1, \lambda_2)^T$ are Lagrange multipliers.

Figure 7.1 displays the code used to define the DAE to DAETS. This (template) code is employed when computing the Σ matrix and generating a computational graph by FADBAD++ for evaluating TCs for the f_i and needed gradients for stages $k \leq 0$. In this figure, the function `Ftpl` evaluates $f(t, p, \lambda)$, and the function `CarAxis` computes

$$
F(t, p, p'', \lambda) := \left(\begin{array}{c} -Kp'' + f(t, p, \lambda) \\ \phi(t, p) \end{array} \right).
$$

The form of the DAE problem solved by DAETS is $F(t, p, p'', \lambda) = 0$.

Figure 7.2 shows resulting solutions for p with the initial condition given in [21].

7.1.1   Accuracy results.

We investigate the accuracy of the numerical solutions computed by DAETS on (7.1), over a range of tolerances, and the accuracy of the reference solutions reported in [20] and [21]. We use the same initial condition as in [20, 21].

If we denote the ith component of a reference solution at t_{end} by r_i and the ith component of a solution computed by DAETS by x_i, we estimate the relative error in x_i at t_{end} by $|x_i - r_i|/|r_i|$. By SCD, significant correct digits, we denote the number of minimum correct digits in a numerical solution at the end of an integration interval [20, 21]:

$$
\text{SCD} := -\log_{10}\big(\|\text{relative error at the end of integration interval}\|_\infty\big),
$$

where we take the norm of a vector of relative errors.

We consider three reference solutions:

1. computed by DAETS on Ultra Sparc 10 in IEEE double precision with order 15, atol = rtol = 10^{-16}, and tolerance dtol = 10^{-14} for the IPOPT package[4];

---

[4]atol and rtol are used to control the truncation error in the Taylor series solution, and dtol is the tolerance used by IPOPT; 10^{-14} is the smallest tolerance it can satisfy on this problem.

```
// Various #include and #define statements omitted
template <typename T> void Ftpl( T *f, const T *y, const T & t )
{
  T yb = R*sin(W*t);                                    yb = R sin(Wt)
  T xb = sqrt( sqr(L) - sqr(yb) );                      xb = √(L² − yb²)
  T Ll = sqrt( sqr(xl) + sqr(yl) );                     Ll = √(xl² + yl²)
  T Lr = sqrt( sqr(xr-xb) + sqr(yr-yb) );   Lr = √((xr − xb)² + (yr − yb)²)

  f[0] = (L0-Ll)*xl/Ll + lam1*xb + 2.0*lam2*(xl-xr) ;
```
$$f_0 = \frac{(L_0 - L_l)x_l}{L_l} + \lambda_1 x_b + 2\lambda_2(x_l - x_r)$$
```
  f[1] = (L0-Ll)*yl/Ll + lam1*yb + 2.0*lam2*(yl-yr) - EPS_M;
```
$$f_1 = \frac{(L_0 - L_l)y_l}{L_l} + \lambda_1 y_b + 2\lambda_2(y_l - y_r) - \frac{\epsilon^2}{M}$$
```
  f[2] = (L0-Lr)*(xr-xb)/Lr - 2.0*lam2*(xl-xr);
```
$$f_2 = \frac{(L_0 - L_r)(x_r - x_b)}{L_r} - 2\lambda_2(x_l - x_r)$$
```
  f[3] = (L0-Lr)*(yr-yb)/Lr - 2.0*lam2*(yl-yr) - EPS_M;
```
$$f_3 = \frac{(L_0 - L_r)(y_r - y_b)}{L_r} - 2\lambda_2(y_l - y_r) - \frac{\epsilon^2}{M}$$
```
}

// function for evaluating the DAE
template <typename T> void CarAxis( T *f, const T *y, const T & t )
{
// set the differential equations
  Ftpl( f, y, t );
  for ( int i = 0; i < 4; i++ )                         For i = 0, . . . , 3

    f[i] = -EPS_M*diff( y[i], 2 ) + f[i];               fi = −(ε²/M)yi″ + fi

// set the algebraic equations
  T yb = R*sin(W*t);                                    yb = R sin(Wt)
  T xb = sqrt( sqr(L) - sqr(yb) );                      xb = √(L² − yb²)
  f[4] = xl*xb+yl*yb;                                   f4 = xl xb + yl yb
  f[5] = sqr(xl-xr) + sqr(yl-yr) - sqr(L);   f5=(xl−xr)²+(yl−yr)²−L²
}
```

Figure 7.1: `C++` code defining the Car Axis DAE, with the mathematical form of the equations on the right. $\epsilon, M, L, L_0, W, R$ are constants. Input array `y` holds the state variables $x_l, y_l, x_r, y_r, \lambda_1, \lambda_2$; for example, x_l is encoded (for clarity) as `#define xl y[0]`, and the rest of the variables are encoded similarly.

   2. computed by PSIDE [9] on Cray C90 with Cray's double precision and atol = rtol = 10^{-16}, as reported in [20];

   3. computed by GAMD [15] on an Alpha server DS20E in quadruple precision with atol = rtol = 10^{-24}, as reported in [21].

In Figure 7.3(a) we plot, for each of these reference solutions, the SCD of the numerical solution produced by DAETS with respect to the corresponding reference solution. Since, in [20, 21], the relative errors in p, p', and λ are in-
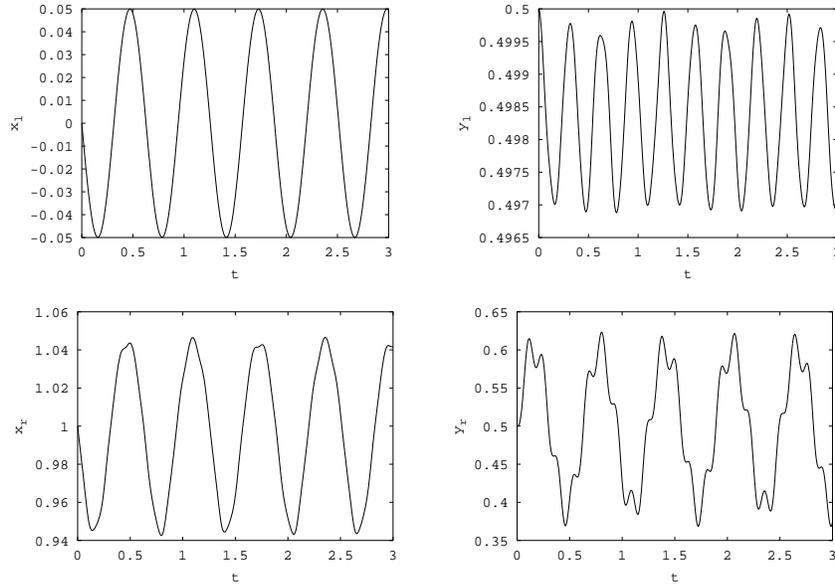
Figure 7.2: Car Axis: plots of x_l, y_l, x_r, and y_r versus t.
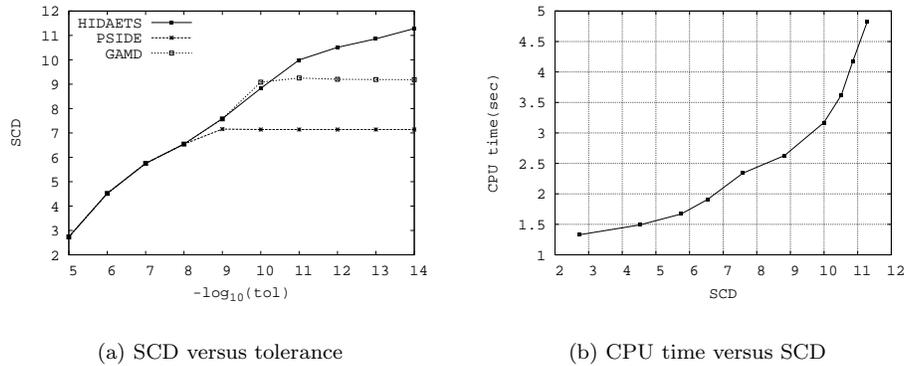
(a) SCD versus tolerance   (b) CPU time versus SCD

Figure 7.3: Car Axis: accuracy and work-precision diagrams. DAETS refers to the SCD in DAETS with respect to its reference solution; PSIDE refers to the SCD computed with the numerical solutions from DAETS and the reference solution from PSIDE; similar for GAMD.

cluded when computing SCD, we include them here too. The numerical solutions with DAETS are computed with atol = rtol = 10^{-m} and dtol = $0.5 \cdot$ atol for $m = 5, \ldots, 13$, and atol = rtol = 10^{-14} and dtol = 10^{-14}.

Although we do not have a guarantee that the DAETS reference solution is the most accurate one, based on our numerical experience and the above plots,

we believe it is. From this figure, the reference solution from PSIDE has about 7.1 SCD, and the reference solution from GAMD has about 9.2 SCD.

7.1.2   Timing results.

Figure 7.3(b) shows the work-precision diagram for executing DAETS on the Car Axis problem. We have used order 15 in DAETS and tolerances and DAETS reference solution as described before. The timing is performed on Sun Ultra 5/10, Ultra SPARC-IIi 360MHz CPU, 4GB memory, Solaris 9. The compiler is gcc 3.2 with optimization flag -O2.

On this problem, for the accuracy PSIDE and GAMD can achieve, DAETS is less efficient (cf. [20, 21]), but it is much more accurate at small tolerances. Generally, on problems for which a solver based on a Runge-Kutta or a multistep method is already very efficient, DAETS may not be competitive. However, it will be competitive on problems that current methods cannot handle because of high index, or if high accuracy is required.

7.2   Two pendula: index-5 DAE.

This artificial problem of index 5 comprises two simple pendula coupled in such a way that the length of the second, "driven" pendulum depends on tension (effectively λ) in the first, "driving" one. The equations are

$$(7.2) \quad \begin{aligned} 0 &= x'' + x\lambda, & 0 &= u'' + u\kappa, \\ 0 &= y'' + y\lambda - G, & 0 &= v'' + v\kappa - G, \\ 0 &= x^2 + y^2 - L^2, & 0 &= u^2 + v^2 - (L + c\lambda)^2, \end{aligned}$$

where G, L, c are constants, $x(t)$, $y(t)$, $u(t)$, $v(t)$ are the position variables of the pendulum masses, and $\lambda(t)$, $\kappa(t)$ are Lagrange multipliers.

We illustrate the behaviour of two numerical solutions to (7.2) computed by DAETS. We integrate this problem from $t_0 = 0$ to $t_{\text{end}} = 50$ using

- constants $G = 1$, $L = 1$, and $c = 0.1$;

- order = 20, atol = rtol = 10^{-10}, dtol = $0.5 \cdot 10^{-10}$;

- two initial conditions at $t_0 = 0$:

$$(7.3) \quad \begin{aligned} x &= 1, & x' &= 0, & u &= 1, & u' &= 0, \\ y &= 0, & y' &= 1, & v &= 0, & v' &= 1; \end{aligned}$$

and

(7.4)   the same as (7.3) except $v = 0$ is replaced by $v = 0.001$.

The values for x, x', y, and y' are consistent for the first pendulum, but the values for u, u', v, and v' are not consistent for the second pendulum. Table 7.1 contains the consistent values computed by DAETS for these variables.

	consistent values	
	(7.3)	(7.4)
u	1.1	1.0999994500004
u'	$3 \cdot 10^{-1}$	$2.9899985100011 \cdot 10^{-1}$
v	0	$1.0999994500004 \cdot 10^{-3}$
v'	1	1.0002989998510

Table 7.1: Consistent initial values for the second pendulum corresponding to initial values (7.3) and (7.4).
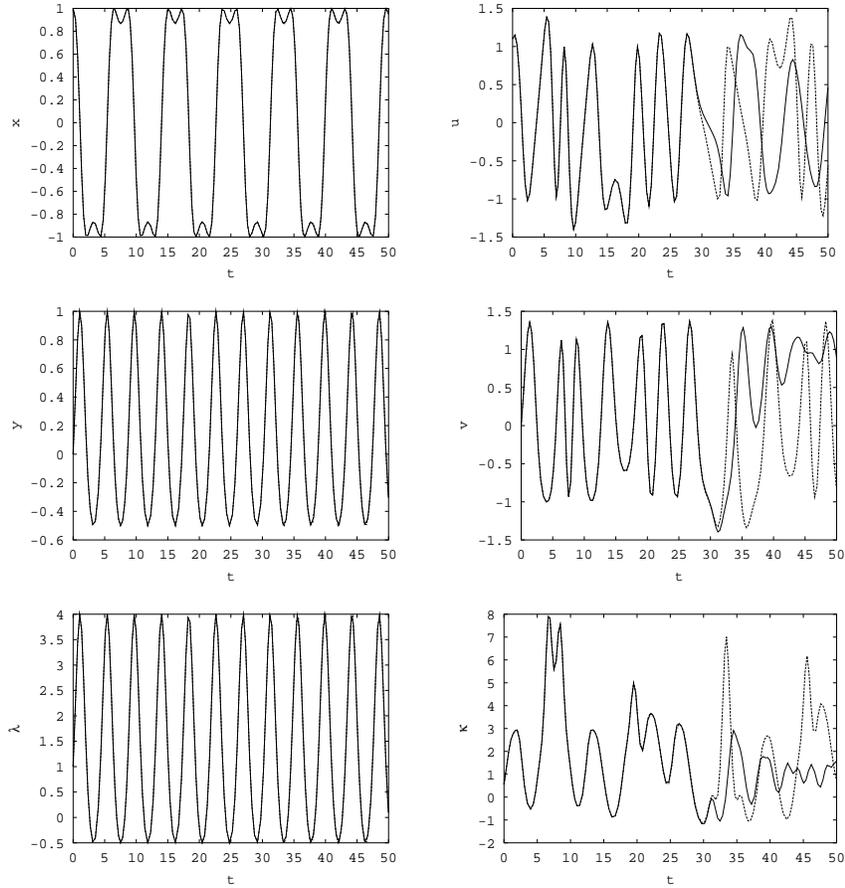
Figure 7.4: Two Pendula: plots of x, y, λ, u, v, and κ versus t. The solid and dashed lines denote the solutions corresponding to (7.3) and (7.4) respectively.

In Figure 7.4, we plot the solution components corresponding to (7.3, 7.4). The two solutions for u, v and κ are close until $t \approx 30$ and diverge afterwards,

indicating a sensitive dependence on initial conditions. Although the motion of the driven pendulum appears chaotic, from our experiments, the computed solutions by DAETS are consistent over a wide range of tolerances, giving confidence that DAETS solves this high-index problem correctly.

7.3   Work versus order of Taylor series.

We have performed experiments in which DAETS was run repeatedly on a problem, varying the error tolerance and the Taylor order (the latter being held fixed for the run) and with its standard stepsize control. The CPU time for each DAETS call was measured. Tests on several problems suggest that, for problems of little or moderate stiffness, an order in the range 20 to 30 appears a good choice at all tolerances. For illustration, Figure 7.5 plots of CPU time against order at various tolerances for the Two Pendula (left) and Car Axis problem (right).
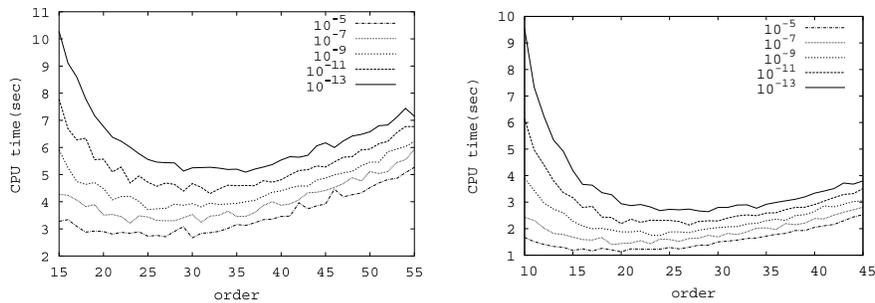

Figure 7.5: CPU time versus Taylor series order.

8   Conclusions.

We have presented a method for computing TCs for the solution of a general DAE, and some examples of the performance of a `C++` code based upon it. Given a DAE described by computer program, the necessary structural analysis data and System Jacobian can be obtained via operator overloading, as in DAETS; or in a pre-processing stage, for an implementation by source-text translation. Either the structural analysis succeeds, or one can know for certain (subject to obstacles of inexact arithmetic) that it has failed.

With this approach, simulation software that automatically converts a model to a DAE need not to produce it in a particular (first-order or lower-index) form: it can be a direct, compact translation of the model.

Computing an initial consistent point, which is quite separate from stepping in most DAE codes, here is just the implementation, for the first step, of stages $k \leq 0$ of the stepping algorithm. It only differs from subsequent steps in that a more robust nonlinear least squares algorithm is used.

The restrictions noted in Subsection 1.1 (no branching, etc., in the code for the DAE) are to simplify the exposition. If they are violated, the theory applies

locally. Of the resulting difficulties, those due solely to discontinuities are well-studied in the AD community. In the extra case that the DAE's structure changes during solution — for instance in a chemical plant model, opening or closing a valve may change the index — run time re-analysis is needed.

The structural analysis has also been implemented in `C++` as a stand-alone structural analyzer. When it succeeds, it reliably finds the degrees of freedom and an upper bound for the differentiation index, and can give a recipe for converting the DAE to ODE form.

Useful insights into Taylor methods are in Barrio's recent work. First, high orders can be used without fear of accuracy loss by cancellation, etc. In [2], dynamical system ODEs are solved to very high accuracy using a multi-precision package. The variable step, variable order method chose orders around 150 at the smallest tolerance used, namely 10^{-128}. Second, in this work, and in [1] where they solve DAE problems in the CWI test set using Pryce's approach, Taylor methods were far faster at high accuracies than others tested.

Third, they show empirically that the stability region of the Taylor method approaches a semicircle in the left half-plane, whose radius increases linearly with the order. Hence Taylor methods can be highly efficient on moderately stiff problems. With DAETS, we have observed this is also true for DAEs.

Owing to the explicit nature of Taylor series methods, they cannot be very efficient for highly stiff DAEs . A promising approach is to generalize the Taylor series approach to Hermite-Obreschkoff (HO) methods [23], which are known to have much better stability, in the ODE sense, than Taylor series methods. Techniques for efficient computation of the relevant Jacobians are being developed.

Future work will include study of HO methods, description of the whole integration process, more detailed reports of numerical experience with DAETS, and study of the applicability of this method to engineering problems.

REFERENCES

1. R. BARRIO, *Performance of the Taylor series method for ODEs/DAEs*, Applied Mathematics and Computation, (2005). To appear.

2. R. BARRIO, F. BLESA, AND M. LARA, *VSVO formulation of the Taylor method for the numerical solution of ODEs*. Preprint, University of Zaragoza. Submitted for publication, 2005.

3. D. Barton, I. M. Willers, and R. V. M. Zahar, *The automatic solution of ordinary differential equations by the method of Taylor series.*, Computer J., 14 (1970), pp. 243–248.

4. C. Bendsten and O. Stauning, *TADIFF, a flexible C++ package for automatic differentiation using Taylor series*, Tech. Rep. 1997-x5-94, Department of Mathematical Modelling, Technical University of Denmark, DK-2800, Lyngby, Denmark, April 1997.

5. M. Berz, *COSY INFINITY version 8 reference manual*, Technical Report MSUCL–1088, National Superconducting Cyclotron Lab., Michigan State University, East Lansing, Mich., 1997.

6. S. L. Campbell and C. W. Gear, *The index of general nonlinear DAEs*, Numerische Mathematik, 72 (1995), pp. 173–196.

7. Y. F. Chang and G. F. Corliss, *ATOMFT: Solving ODEs and DAEs using Taylor series*, Computers and Mathematics with Applications, 28 (1994), pp. 209–233.

8. G. F. Corliss and W. Lodwick, *Role of constraints in the validated solution of DAEs*, Tech. Rep. 430, Marquette University Department of Mathematics, Statistics, and Computer Science, Milwaukee, Wisc., March 1996.

9. J. J. B. de Swart, W. M. Lioen, and W. A. van der Veen, *PSIDE — parallel software for implicit differential equations*, December 1997. `http://www.cwi.nl/archive/projects/PSIDE/`.

10. A. Gofen, *The Taylor Center for PCs: exploring, graphing and integrating ODEs with the ultimate accuracy*, in Computational Science: ICCS 2002, P. Sloot et al., eds., no. 2329 in Lecture notes in Computer Science, Amsterdam, 2002, Springer.

11. G. H. Golub and C. F. V. Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, 3rd ed., 1996.

12. A. Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Frontiers in applied mathematics, SIAM, Philadelphia, PA, 2000.

13. A. Griewank, D. Juedes, and J. Utke, *ADOL-C, a package for the automatic differentiation of algorithms written in C/C++*, ACM Trans. Math. Software, 22 (1996), pp. 131–167.

14. J. Hoefkens, *Rigorous Numerical Analysis with High-Order Taylor Models*, PhD thesis, Department of Mathematics and Department of Physics and Astronomy, Michigan State University, East Lansing, MI 48824, August 2001.

15. F. Iavernaro and F. Mazzia, *Block-boundary value methods for the solution of ordinary differential equation*, SIAM J. Sci. Comput., 21 (1999), pp. 323–339. GAMD web site is `http://pitagora.dm.uniba.it/~mazzia/ode/gamd.html`.

16. K. R. Jackson and N. S. Nedialkov, *Some recent advances in validated methods for IVPs for ODEs*, Applied Numerical Mathematics, 42 (2002), pp. 269–284.

17. R. Jonker and A. Volgenant, *A shortest augmenting path algorithm for dense and sparse linear assignment problems*, Computing, 38 (1987), pp. 325–340. The assignment code is available at `www.magiclogic.com/assignment.html`.

18. A. Jorba and M. Zou, *A software package for the numerical integration of ODE by means of high-order Taylor methods*, Tech. Rep. , Department of Mathematics, University of Texas at Austin, TX 78712-1082, USA, 2001.

19. C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, no. 16 in Frontiers in Applied Mathematics, SIAM, Philadelphia, 1995.

20. W. M. Lioen and J. J. B. de Swart, *Test set for initial value problem solvers*, Tech. Rep. MAS-R9832, CWI, Amsterdam, The Nederlands, December 1998. `http://www.cwi.nl/cwi/projects/IVPtestset/`.

21. F. Mazzia and F. Iavernaro, *Test set for initial value problem solvers*, Tech. Rep. 40, Department of Mathematics, University of Bari, Italy, 2003. `http://pitagora.dm.uniba.it/~testset/`.

22. R. E. Moore, *Interval Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 1966.

23. N. S. Nedialkov, *Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation*, PhD thesis, Department of Computer Science, University of Toronto, Toronto, Canada, M5S 3G4, February 1999.

24. N. S. Nedialkov, K. R. Jackson, and G. F. Corliss, *Validated solutions of initial value problems for ordinary differential equations*, Applied Mathematics and Computation, 105 (1999), pp. 21–68.

25. N. S. Nedialkov and J. D. Pryce, *Solving differential-algebraic equations by Taylor series (ii): Computing the System Jacobian.* Submitted to BIT, 2005.

26. C. C. Pantelides, *The consistent initialization of differential-algebraic systems*, SIAM. J. Sci. Stat. Comput., 9 (1988), pp. 213–231.

27. J. D. Pryce, *Solving high-index DAEs by Taylor Series*, Numerical Algorithms, 19 (1998), pp. 195–211.

28. J. D. Pryce, *A simple structural analysis method for DAEs*, BIT, 41 (2001), pp. 364–394.

29. L. B. Rall, *Automatic Differentiation: Techniques and Applications*, vol. 120 of Lecture Notes in Computer Science, Springer Verlag, Berlin, 1981.

30. O. Stauning and C. Bendtsen, *FADBAD++ web page*, May 2003. FADBAD++ is availabe at `www.imm.dtu.dk/fadbad.html`.

31. A. Wächter, *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*, PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2002.