EXPLOITING FINE BLOCK TRIANGULARIZATION AND QUASILINEARITY IN DIFFERENTIAL-ALGEBRAIC EQUATION SYSTEMS

NEDIALKO S. NEDIALKOV[†][§], GUANGNING TAN[†][¶], AND JOHN D. PRYCE[‡][‖]

Abstract. The Σ-method for structural analysis of a differential-algebraic equation (DAE) system produces offset vectors from which the sparsity pattern of DAE's system Jacobian is derived; this pattern implies a fine block-triangular form (BTF). This article derives a simple method for quasilinearity analysis of a DAE and combines it with its fine BTF to construct a method for finding the minimal set of initial values needed for consistent initialization and a method for a block-wise computation of derivatives for the solution to the DAE.

Key words. differential-algebraic equations, structural analysis, quasilinearity

AMS subject classifications. 34A09, 65L80, 41A58, 65F50

1. Introduction.

The authors have developed the MATLAB package DAESA, Differential-Algebraic Equations Structural Analyzer [6], aimed at analyzing the structure of a system of differential-algebraic equations (DAEs) of the general form

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

where the $x_j(t)$, $j = 1, \dots, n$, are state variables, and t is the time variable. The f_i can be arbitrary expressions built from the x_j and t using $+, -, \times, \div$, other analytic standard functions, and the d^p/dt^p operator.

DAESA implements the Σ-method for structural analysis [7]. Using operator overloading, this package extracts the signature matrix of (1.1), and then by solving a linear assignment problem, finds two offset vectors, from which it constructs coarse and fine block-triangular forms (BTFs) of the DAE. Using the fine BTF, DAESA performs *quasilinearity* (QL) analysis and then finds the *minimal set* of variables and derivatives of them that require initial values, and also constructs a *block-wise solution scheme.*

Some of the theory of these BTFs is presented in [9], where several results were left to be proved as future work. The companion article [8] proves them and presents new results on BTFs, and in particular related to the fine BTF. Describing the method for QL analysis was also left for future work in [9]: we derive this method here. We also present DAESA's algorithm for finding the minimal set of variables and derivatives that need to be initialized and the algorithm for producing a block-wise solution scheme.

Section 2 illustrates how the computation of derivatives for the solution to (1.1) was prescribed originally by the Σ-method. Section 3 derives a method for computing them based on a fine BTF of the DAE. A simple method for QL analysis is derived in Section 4. The overall solution scheme for computing derivatives for the solution to (1.1), building on its fine BTF and QL information, is given in Section 5. Conclusions are in Section 6.

[†]Department of Computing and Software, McMaster University, Hamilton, Canada

[‡]Cardiff School of Mathematics, Cardiff University, UK

[§]Supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC)

[¶]Supported in part by the Ontario Research Fund (ORF), Canada

[‖]Supported in part by The Leverhulme Trust

For brevity, we refer to the companion article [8] for definitions and concepts. A term that is explained in [8] is typeset here in slanted font on first occurrence, and the subsection where it appears in [8] is referenced as [§X].

We assume that (1.1) is *structurally well posed*; that is, its *signature matrix* $\Sigma = (\sigma_{ij})$ contains a *highest-value transversal (HVT)* with entries $> -\infty$ [§2.1].

2. Basic solution scheme. Let c and d be *valid offset vectors* [§2.1] for (1.1), and let $k_d = -\max_j d_j$. We can find derivatives for the solution to (1.1) in stages $k = k_d, k_d + 1, \ldots$, where at stage k we

$$\text{solve} \quad \{\, f_i^{(k+c_i)} = 0 \mid k + c_i \geq 0 \,\} \tag{2.1}$$

$$\text{for} \quad \{\, x_j^{(k+d_j)} \mid k + d_j \geq 0 \,\} \tag{2.2}$$

using values for $\{\, x_j^{(r)} \mid 0 \leq r < k + d_j \,\}$, which are found at stages $< k$ [7]. By a "derivative" $x_j^{(r)}$ we shall mean x_j and (appropriate) derivatives of it.

We say the DAE (1.1) is quasilinear (QL), if it is linear in the highest-order derivatives occurring in it, and non-quasilinear (NQL) otherwise (see also §4). To start this stage-wise process, we need to initialize

$$\{\, x_j^{(r)} \mid 0 \leq r \leq d_j - \gamma \,\}, \quad \text{where } \gamma = 1 \text{ if the DAE is QL and 0 otherwise.} \tag{2.3}$$

We refer to (2.1, 2.2) as basic (solution) scheme. It succeeds (locally), if the *System Jacobian* $\mathbf{J}$, defined as $\mathbf{J}_{ij} = \partial f_i / \partial x^{(\sigma_{ij})}$, if $\sigma_{ij} = d_j - c_i$ and 0 otherwise, is non-singular at a consistent point [§2.1], see also [7]. The systems at stages $k < 0$ are generally underdetermined. For stages $k \geq 0$ they are square, and for $k > 0$ always linear, where the matrix of the linear system is $\mathbf{J}$. If the DAE is QL, the system at $k = 0$ is also linear with a matrix $\mathbf{J}$.

In practice, when solving (1.1) numerically by Taylor series, we compute Taylor coefficients (TCs) $x_j^{(k+d_j)}/(k + d_j)!$ directly, where instead of derivatives in (2.1, 2.2) we have TCs. Such a computation is implemented in the DAETS solver; see [3, 4, 5] for details. In the present work, for simplicity of the exposition, we express the theory in terms of derivatives.

Example 2.1. Throughout this article, we use as an example the following DAE of differentiation index 7:

$$\begin{aligned} 0 &= A = x'' + x\lambda & 0 &= D = u'' + u\mu \\ 0 &= B = y'' + y\lambda + (x')^3 - G & 0 &= E = (v''')^2 + v\mu - G \\ 0 &= C = x^2 + y^2 - L^2 & 0 &= F = u^2 + v^2 - (L + c\lambda)^2 + \lambda''. \end{aligned} \tag{2.4}$$

The state variables are x, y, λ, u, v, and μ; L (length), G (gravity), and $c > 0$ are constants. These equations are obtained from a two-pendula problem [6] in which

$$B = y'' + y\lambda - G, \quad E = v'' + v\mu - G, \quad \text{and} \quad F = u^2 + v^2 - (L + c\lambda)^2.$$

The Σ matrix of (2.4) and its $\mathbf{J}$ are shown in Figure 2.1. In Table 2.1, we illustrate the basic scheme (2.1, 2.2) when applied to (2.4). This problem is NQL (because of $(v''')^2$), so $\gamma = 0$, and (2.3) implies we need to give initial values for $x^{(\leq 6)}$, $y^{(\leq 6)}$, $\lambda^{(\leq 4)}$, $u^{(\leq 2)}$, $v^{(\leq 3)}$, and μ; the notation $z^{(\leq r)}$ is short for $z, z', \ldots, z^{(r)}$. We show in §5.1 how the number of initial values can be drastically reduced by exploiting a fine BTF of (2.4).

$$
\Sigma = \begin{array}{cc|cccccc|c}
 & & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & \\
 & & x & y & \lambda & u & v & \mu & c_i \\
\hline
f_1 & A & 2^\bullet & & 0 & & & & 4 \\
f_2 & B & 1 & 2 & 0^\bullet & & & & 4 \\
f_3 & C & 0 & 0^\bullet & & & & & 6 \\
f_4 & D & & & & 2 & & 0^\bullet & 0 \\
f_5 & E & & & & & 3^\bullet & 0 & 0 \\
f_6 & F & & & 2 & 0^\bullet & 0 & & 2 \\
\hline
 & d_j & 6 & 6 & 4 & 2 & 3 & 0 &
\end{array}
, \quad
\mathbf{J} = \begin{array}{c|cccccc|}
 & x & y & \lambda & u & v & \mu \\
\hline
A & 1 & & x & & & \\
B & & 1 & y & & & \\
C & 2x & 2y & & & & \\
D & & & & 1 & & u \\
E & & & & & 2v''' & v \\
F & & & 1 & 2u & &
\end{array}
$$

blank denotes $-\infty$ blank denotes 0

FIG. 2.1. *Signature matrix and system Jacobian of (2.4) with labeling of equations, variables and offsets. A HVT in* Σ *is marked with* $\bullet$. $\mathbf{J}_{2,1} = 0$ *and* $\mathbf{J}_{6,5} = 0$, *since* $2 = d_1 - c_2 > \sigma_{2,1} = 1$ *and* $1 = d_5 - c_6 > \sigma_{6,5} = 0$, *respectively.*

TABLE 2.1
Basic scheme for computing derivatives of the solution to (2.4). Nonlinear equations and the variables that appear nonlinearly in them are marked by ▨.

		stage k								
	c_i, d_j	-6	-5	-4	-3	-2	-1	0	$\cdots$	>0
solve	4			A	A'	A''	A'''	$A^{(4)}$	$\cdots$	$A^{(k+4)}$
	4			B	B'	B''	B'''	$B^{(4)}$	$\cdots$	$B^{(k+4)}$
	6	C	C'	C''	C'''	$C^{(4)}$	$C^{(5)}$	$C^{(6)}$	$\cdots$	$C^{(k+6)}$
	0							D	$\cdots$	$D^{(k)}$
	0							E	$\cdots$	$E^{(k)}$
	2					F	F'	F''	$\cdots$	$F^{(k+2)}$
for	6	x	x'	x''	x'''	$x^{(4)}$	$x^{(5)}$	$x^{(6)}$	$\cdots$	$x^{(k+6)}$
	6	y	y'	y''	y'''	$y^{(4)}$	$y^{(5)}$	$y^{(6)}$	$\cdots$	$y^{(k+6)}$
	4			λ	λ'	λ''	λ'''	$\lambda^{(4)}$	$\cdots$	$\lambda^{(k+4)}$
	2					u	u'	u''	$\cdots$	$u^{(k+2)}$
	3				v	v'	v''	v'''	$\cdots$	$v^{(k+3)}$
	0							μ	$\cdots$	$\mu^{(k)}$

3. Solution scheme through fine BTF.

The *coarse BTF* of (1.1) is based on the sparsity pattern of the DAE: $S = \{ (i,j) \mid \sigma_{ij} > -\infty \}$. For given valid offset vectors c and d, the corresponding *fine BTF* [§4.1] is based on the sparsity pattern of $\mathbf{J}$:

$$S_0 = S_0(c,d) = \{ (i,j) \mid d_j - c_i = \sigma_{ij} \}. \tag{3.1}$$

We refer to c and d as *global offsets*. By $\widehat{c}$ and $\widehat{d}$ we denote the vectors of *local offsets*. We assume that both global and local offsets are valid but not necessarily *canonical* [§5.1], except in §5.1, where the local offsets must canonical.

Example 3.1. For (2.4) there are two coarse (diagonal) blocks, see Figure 3.1, consisting of equations $E, D, F = 0$ in variables v, μ, u, and equations $A, B, C = 0$ in variables x, y, λ, respectively. The former can be decomposed into three fine blocks, while the latter cannot be decomposed into smaller fine blocks, as the block form of its sparsity pattern, S_0, is irreducible.

$$\Sigma = \begin{array}{cc|cccccc|cc} & & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & & \\ & & v & \mu & u & x & y & \lambda & c_i & \widehat{c}_i \\ \hline f_1 & E & 3 & 0 & & & & & 0 & 0 \\ f_2 & D & & 0 & 2 & & & & 0 & 0 \\ f_3 & F & 0 & & 0 & & & 2 & 2 & 0 \\ f_4 & A & & & & 2 & & 0 & 4 & 0 \\ f_5 & B & & & & 1 & 2 & 0 & 4 & 0 \\ f_6 & C & & & & 0 & 0 & & 6 & 2 \\ \hline & d_j & 3 & 0 & 2 & 6 & 6 & 4 & & \\ & \widehat{d}_j & 3 & 0 & 0 & 2 & 2 & 0 & & \end{array}, \quad \mathbf{J} = \begin{array}{c|cccccc|} & v & \mu & u & x & y & \lambda \\ \hline E & 2v''' & v & & & & \\ D & & u & 1 & & & \\ F & & & 2u & & & 1 \\ A & & & & 1 & & x \\ B & & & & & 1 & y \\ C & & & & 2x & 2y & \end{array}$$

FIG. 3.1. *Permuted Σ and $\mathbf{J}$ of (2.4) into fine BTF; c_i and d_j are global canonical offsets, and $\widehat{c}_i$ and $\widehat{d}_j$ are local canonical offsets.*

In the solution scheme below, we exploit the fine BTF (corresponding to the given global offsets): we assume that the equations and variables of (1.1) are permuted such that the resulting Σ and $\mathbf{J}$ are in fine BTF. For every position (i, j) below a diagonal block, $d_j - c_i > \sigma_{ij}$, and hence $\mathbf{J}_{i,j}$ is identically zero.

For simplicity in the notation, we denote the permuted equations and variables as before, $f_1, f_2, \dots, f_n$ and $x_1, x_2, \dots, x_n$. (In the companion paper [8] they are denoted by $\widetilde{f}_i$ and $\widetilde{x}_i$, $i = 1:n$.)

3.1. Solution scheme by blocks. Assume that there are p fine diagonal blocks, each of size N_l. Denote by B_l the set of indices of rows [resp. columns] in block l. That is,

$$B_l = \left\{ i \mid \textstyle\sum_{r=1}^{l-1} N_r < i \leq \sum_{r=1}^{l} N_r \right\}.$$

For example in Figure 3.1, $2 \in B_2$ and $5 \in B_4$. Denote also

$$B_{l:q} = \bigcup_{r=l}^{q} B_r.$$

We re-organize the basic scheme (2.1–2.2) as follows. At stage k, we solve blocks $l = p, p-1, \dots, 1$ in order such that for block l we

$$\text{solve} \quad \left\{ f_i^{(k+c_i)} = 0 \mid i \in B_l \text{ and } k + c_i \geq 0 \right\} \tag{3.2}$$

$$\text{for} \quad \left\{ x_j^{(k+d_j)} \mid j \in B_l \text{ and } k + d_j \geq 0 \right\} \tag{3.3}$$

using

$$\begin{aligned} &\left\{ x_j^{(r)} \mid 0 \leq r < k + d_j \right\} && \text{(computed at stages} < k\text{) and} \\ &\left\{ x_j^{(k+d_j)} \mid j \in B_{l+1:p} \text{ and } k + d_j \geq 0 \right\} && \\ & && \text{(available from blocks} > l \text{ at this stage).} \end{aligned} \tag{3.4}$$

We refer to (3.2–3.4) as block (solution) scheme.

TABLE 3.1
Block scheme for computing derivatives for the solution to (2.4).

			stage k								
		c_i, d_j	-6	-5	-4	-3	-2	-1	0	$\cdots$	>0
block 4	solve	4			A	A'	A''	A'''	$A^{(4)}$	$\cdots$	$A^{(k+4)}$
		4			B	B'	B''	B'''	$B^{(4)}$	$\cdots$	$B^{(k+4)}$
		6	C	C'	C''	C'''	$C^{(4)}$	$C^{(5)}$	$C^{(6)}$	$\cdots$	$C^{(k+6)}$
	for	6	x	x'	x''	x'''	$x^{(4)}$	$x^{(5)}$	$x^{(6)}$	$\cdots$	$x^{(k+6)}$
		6	y	y'	y''	y'''	$y^{(4)}$	$y^{(5)}$	$y^{(6)}$	$\cdots$	$y^{(k+6)}$
		4			λ	λ'	λ''	λ'''	$\lambda^{(4)}$	$\cdots$	$\lambda^{(k+4)}$
block 3							$\downarrow$	$\downarrow$	$\downarrow$	$\cdots$	$\downarrow$
	solve	2					F	F'	F''	$\cdots$	$F^{(k)}$
	for	2					u	u'	u''	$\cdots$	$u^{(k)}$
block 2									$\downarrow$	$\cdots$	$\downarrow$
	solve	0							D	$\cdots$	$D^{(k)}$
	for	0							μ	$\cdots$	$\mu^{(k)}$
block 1									$\downarrow$	$\cdots$	$\downarrow$
	solve	0							E	$\cdots$	$E^{(k)}$
	for	3				v	v'	v''	v'''	$\cdots$	$v^{(k+3)}$

Example 3.2. Using the fine BTF from Figure 3.1, we illustrate in Table 3.1 the scheme (3.2–3.4) when applied to (2.4). As we discuss in §5.1, for this problem we need to give initial guesses for x, x', y, y', u, v''', and initial values for v, v', v''.

Using block 4, we can compute derivatives for x, y, λ independently of the other blocks. At stage -2, as soon as λ'' is available, we can find u in block 3 by solving (nonlinear) $F = 0$. At stage 0, as soon as u'' is available from block 3, we can find μ in block 2 by solving (linear) $D = 0$, and then find v''' in block 1 by solving (nonlinear) $E = 0$. When considering block 1, at each stage -3, -2, and -1 there is no equation to solve for v, v', and v'', respectively, and initial values are required for them.

3.2. Block scheme more formally. Denote

$$I_k^l = \{\, (i,r) \mid i \in B_l \text{ and } r = k + c_i \geq 0 \,\}, \tag{3.5}$$

$$J_k^l = \{\, (j,r) \mid j \in B_l \text{ and } r = k + d_j \geq 0 \,\}, \tag{3.6}$$

$$J_{<k} = \{(j,r) \mid 0 \leq r < k + d_j\}, \quad \text{and} \tag{3.7}$$

$$\begin{aligned} J_k^{>l} &= \{(j,r) \mid j \in B_{l+1:p} \text{ and } r = k + d_j \geq 0\} \\ &= \bigcup_{r>l} J_k^r. \end{aligned} \tag{3.8}$$

By $\mathbf{x}_{J_k^l}$ we mean a vector with components the elements of the set[1]

$$\{\, x_j^{(r)} \mid (j,r) \in J_k^l \,\};$$

[1]Some ordering must be agreed on but it does not matter.

similarly for $\mathbf{x}_{J_{<k}}$ and $\mathbf{x}_{J_k^{>l}}$. By $\mathbf{f}_{I_k^l}$ we denote a vector with components

$$\{\, f_i^{(r)} \mid (i,r) \in I_k^l \,\}.$$

Then (3.2–3.4) can be written concisely as: at stage k solve in order for blocks $l = p, p-1, \dots, 1$ the system

$$\mathbf{f}_{I_k^l}(t, \mathbf{x}_{J_{<k}}, \mathbf{x}_{J_k^l}, \mathbf{x}_{J_k^{>l}}) = 0 \tag{3.9}$$

for $\mathbf{x}_{J_k^l}$, where $\mathbf{x}_{J_{<k}}$ is found during previous stages, and $\mathbf{x}_{J_k^{>l}}$ is found at this stage but from previous blocks.

Example 3.3. To illustrate the above, consider stage $k = -2$. The sets (3.5, 3.6, 3.8) are (blank denotes $\emptyset$)

l	I_{-2}^l	J_{-2}^l	$J_{-2}^{>l}$
4	$\{(4,2),(5,2),(6,4)\}$	$\{(4,4),(5,4),(6,2)\}$	
3	$\{(3,0)\}$	$\{(3,0)\}$	$\{(6,2)\}$
2			
1		$\{(1,1)\}$	

and (3.7) is $J_{<-2} = \{\, (1,0), (4,<4)(4,<4), (5,<4), (6,<2) \,\}$, where $(j,<r)$ is short for $(j,0), (j,1), \dots (j, r-1)$. Then (3.9) can be illustrated as

l	$\mathbf{f}_{I_{-2}^l}$	$\mathbf{x}_{J_{-2}^l}$	$\mathbf{x}_{J_{-2}^{>l}}$
4	$A'', B'', C^{(4)}$	$x^{(4)}, y^{(4)}, \lambda''$	
3	F	u	λ''
2			
1		v'	

where $\mathbf{x}_{J_{<-2}} = \big(v;\, x, x', x'', x''';\, y, y', y'', y''';\, \lambda, \lambda'\big)$. Note that in block 1, there is nothing to solve, and we give an (arbitrary) initial value for v'.

For each fine block l, the difference between global and local offsets is a non-negative constant, the *lead time* [§5.1] of this block:

$$d_i - \widehat{d}_i = c_i - \widehat{c}_i = K_l \ge 0 \quad \text{for all } i \in B_l.$$

Let $k_l = k + K_l$, which we refer to as local stage. Then

$$k + c_i = k_l + \widehat{c}_i \quad \text{and} \quad k + d_j = k_l + \widehat{d}_j. \tag{3.10}$$

Using (3.10), we write (3.2, 3.3) as

$$\text{solve} \quad \big\{ f_i^{(k_l + \widehat{c}_i)} = 0 \mid i \in B_l \text{ and } k_l + \widehat{c}_i \ge 0 \big\} \tag{3.11}$$

$$\text{for} \quad \big\{ x_j^{(k_l + \widehat{d}_j)} \mid j \in B_l \text{ and } k_l + \widehat{d}_j \ge 0 \big\}. \tag{3.12}$$

For k such that $k_l = k + K_l \ge 0$, from (3.11, 3.12), we have a square system, and therefore (3.9) is square. For $k_l > 0$ it is always linear in its highest-order derivatives, as each of the equations is differentiated at least once.

LEMMA 3.1. *If* $\min_{i\in B_l} \widehat{c}_i = 0$*, then system (3.9) is underdetermined for any* k *such that* $k_l = k + K_l < 0$.

Proof. Since the sparsity pattern S_0 (3.1) of a fine block is irreducible, by the Strong Hall Property [8, §4.2], any set of $r \le N_l - 1$ columns [resp. rows] contains elements of at least $r + 1$ rows [resp. columns]. Therefore, any set of $r \le N_l - 1$ equations contains at least $r + 1$ variables x_j with $\sigma_{ij} = d_j - c_i$. The highest-order derivative of such a x_j in $f_i^{(k+c_i)} = f_i^{(k_l+\widehat{c}_i)}$ is

$$\sigma_{ij} + k_l + \widehat{c}_i = \widehat{d}_j - \widehat{c}_i + k + \widehat{c}_i = k_l + \widehat{d}_j \ge 0.$$

Since at least one $k_l + \widehat{c}_i < 0$, where $i \in B_l$, at local stage $k_l < 0$ there are $r \le N_l - 1$ equations with at least $r + 1$ variables. □

Remark 3.1. The condition $\min_{i\in B_l} \widehat{c}_i = 0$ ensures that the local offsets for block l are *normalized* [§2.1]. Canonical offsets are normalized, but not vice versa; see [8].

For brevity, write (3.9) as

$$\mathbf{F}_{I_k^l}(\mathbf{x}_{J_k^l}) = 0. \tag{3.13}$$

Let $k_l = k + K_l < 0$. If $I_k^l = \emptyset$, then there are no equations at this stage, and we simply give initial values for $\mathbf{x}_{J_k^l}$; otherwise we need initial guesses (trial values for a nonlinear solver) $\widetilde{\mathbf{x}}_{J_k^l}$ to find $\mathbf{x}_{J_k^l}$ by solving

$$\min \|\widetilde{\mathbf{x}}_{J_k^l} - \mathbf{x}_{J_k^l}\|_2 \quad \text{s.t.} \quad \mathbf{F}_{I_k^l}(\mathbf{x}_{J_k^l}) = 0.$$

We derive in §5.1 an algorithm for determining which derivatives to initialize and in §5.2 an algorithm for the overall solution scheme for (3.13).

4. Quasilinearity analysis.

Consider the basic scheme. When $k > 0$, an $f_i^{(k+c_i)}$ with $k + c_i > 0$ is linear in its highest-order derivatives. If $k \le 0$ and c_i is such that $k + c_i = 0$, the corresponding f_i can be nonlinear in the derivatives we solve for, which are the $x_j^{(\sigma_{ij})}$ with $\sigma_{ij} = d_j - c_i$.

For an f_i, let

$$Y_i = \left\{\, x_j^{(\sigma_{ij})} \mid \sigma_{ij} = d_j - c_i \,\right\}. \tag{4.1}$$

DEFINITION 4.1. *Equation* $f_i = 0$ *is QL, if it is linear in all* $y \in Y_i$*, and NQL otherwise.*

DEFINITION 4.2. *System (2.1) is NQL (at stage k), if it contains an undifferentiated NQL* f_i*, and QL otherwise. The DAE (1.1) is NQL, if it is NQL at stage 0, and QL otherwise.*

Example 4.1. Consider (2.4) and Table 2.1. At stage $k = -2$, we have $k + c_i = 0$ only for equation $f_6 = F = 0$. Here, $Y_6 = \{\lambda'', u\}$ as $x_3^{(d_3-c_6)} = x_3^{(4-2)} = \lambda''$ and $x_4^{(d_4-c_6)} = x_4^{(2-2)} = u$; $x_1^{(d_5-c_6)} = x_5' = v'$ does not appear in F (note $1 = d_5 - c_6 > \sigma_{6,5} = 0$). This equation and the system are NQL at stage $k = -2$.

Our goal is to determine automatically if a system at stage $k \le 0$ is QL. (For stages $k > 0$, they are always QL.) We achieve this by propagating the *offset* and *QLity* (pronounced *cuellity*) for each variable in a code list of the DAE. In §4.1, we specify what we mean by code list, and in §4.2 and §4.3, we introduce the offset and QLity of a variable. The derivations in these two subsections are summarized in the algorithm in §4.4. In §4.5, we give a simple modification to it, so it works correctly for the block scheme. Appendix §A suggests a simple approach for an implementation of QL analysis using operator overloading, and also suitable for source code translation.

4.1. Code list. An f_i is described by an expression containing arithmetic operations, standard functions, and the d^p/dt^p operator. Such an expression can be represented by code list containing input, intermediate, and an output variables as follows. The input variables are t and x_j for $j = 1:n$. We rename them as $v_{-n} = t$ and $v_{j-n} = x_j$. Then each subsequent variable v_r for $r > 0$ is defined using previous variables and an operation of arity $m > 0$:

$$v_r = \phi_r(v_{i_1}, v_{i_2}, \dots, v_{i_m}), \quad -n \le i_q < r \quad \text{for each } q = 1:m. \tag{4.2}$$

ϕ_r can be an arithmetic operation, elementary function, the identity function, d^p/dt^p, or a user-defined function. We refer to a constant (an operation of arity 0) directly instead of assigning it to a variable. We refer to the last variable in the code list of an f_i as an output variable.

Assuming that we evaluate all the f_i's and that the last n variables are output variables, we can illustrate the above as

$$\big[\underbrace{v_{-n}}_{t}, \underbrace{v_{j-n}, \dots, v_0}_{x_j}, \quad v_1, \dots, v_q, \ \underbrace{v_{q+1}, \dots, v_{q+n}}_{f_i}\big];$$

$v_1, \dots, v_q$ are intermediate variables.

4.2. Offset of a variable. Let v be a variable[2] in a code list of an f_i.

DEFINITION 4.3. *The* signature vector *of v is the n vector with jth component*

$$\sigma_j(v) = \begin{cases} \textit{the highest-order of derivative of } x_j \textit{ on which } v \textit{ formally depends, or} \\ -\infty \quad \textit{if } v \textit{ does not depend on } x_j. \end{cases}$$

Note that $\sigma_j(v) \le \sigma_{ij}$. "Formally" means that we do not consider symbolic simplifications in the expressions that arise; e.g. the highest-order derivative of x in $x''' + x'' + \lambda x - x'''$ is 3 not 2. (See [4] for more details and an algorithm for the computation and propagation of these vectors to compute the signature matrix of a DAE.)

For an equation number i, denote by M_i the set of indices for which $\sigma_{ij} = d_j - c_i$:

$$M_i = \{\, j \mid \sigma_{ij} = d_j - c_i \,\}.$$

Since we assume the DAE is structurally well posed, each entry in a HVT is ≥ 0, and therefore $\sigma_{ij} = d_j - c_i$ for at least one j. Hence $M_i \ne \emptyset$.

DEFINITION 4.4. *The* offset *of v, with respect to f_i, is*

$$\alpha_i(v) = \min_{j \in M_i} \big(\sigma_{ij} - \sigma_j(v)\big). \tag{4.3}$$

Example 4.2. For $f_6 = F$, $M_6 = \{\, 3, 4 \,\}$. Then (4.3) becomes

$$\begin{aligned} \alpha_6(v) &= \min_{j=3,4} \big(\sigma_{6,j} - \sigma_j(v)\big) = \min\big\{\, \sigma_{6,3} - \sigma_3(v),\ \sigma_{6,4} - \sigma_4(v) \,\big\} \\ &= \min\big\{\, 2 - \sigma_3(v),\ -\sigma_4(v) \,\big\}. \end{aligned}$$

Table 4.1 shows, for the code list in the first column, the corresponding $\sigma_3(v_r)$, $\sigma_4(v_r)$, and $\alpha_6(v_r)$.

[2]This v and later u are not to be confused with the v and u in (2.4).

Table 4.1
Code list and offsets for $f_6 = F$

	code list		expression	$\sigma_3(v_r)$	$\sigma_4(v_r)$	$\alpha_6(v_r)$
	v_{-3}	$= \lambda$	λ	0	$-\infty$	2
	v_{-2}	$= u$	u	$-\infty$	0	0
	v_{-1}	$= v$	v	$-\infty$	$-\infty$	∞
	v_1	$= v_{-2}^2 + v_{-1}^2$	u^2+v^2	$-\infty$	0	0
	v_2	$= (L + cv_{-3})^2$	$(L+c\lambda)^2$	0	$-\infty$	2
	v_3	$= v''_{-3}$	λ''	2	$-\infty$	0
$f_6 =$	v_4	$= v_1 - v_2 + v_3$	$u^2+v^2-(L+c\lambda)^2+\lambda''$	0	0	0

For convenience in the notation below, by "v is an algebraic function ϕ of a set U of variables u" we mean $v = v_r$ in (4.2) is obtained through a $\phi = \phi_r$ with arguments $u \in U = \{ v_{i_1}, \ldots, v_{i_m} \}$.

We propagate variable offsets through the code list of a DAE based on the following lemma.

Lemma 4.5.

(i) If v is $v_{-n} = t$, then $\alpha_i(v_{-n}) = \alpha_i(t) = \infty$.

(ii) If v is $v_{j-n} = x_j$ then

$$\alpha_i(v_{j-n}) = \alpha_i(x_j) = \begin{cases} \sigma_{ij} & \text{if } j \in M_i \\ \infty & \text{otherwise.} \end{cases}$$

(iii) If v is an algebraic function ϕ of a set U of variables u, then

$$\alpha_i(v) = \min_{u\in U} \alpha_i(u).$$

(iv) If $v = d^p u/dt^p$, where $p \geq 0$, then

$$\alpha_i(v) = \alpha_i(u) - p.$$

(v) If v is an output variable (corresponding to f_i), then $\alpha_i(v) = \alpha_i(f_i) = 0$.

Proof.

(i) $v_{-n} = t$ does not depend on any x_j, so $\sigma(v_{-n})$ contains only $-\infty$'s, and $\alpha_i(v_{-n}) = \infty$ for all $i = 1:n$.

(ii) $\sigma_k(x_j) = 0$ if $k = j$ and $-\infty$ otherwise. Hence

$$\alpha_i(v_{j-n}) = \alpha_i(x_j) = \min_{k\in M_i}\bigl(\sigma_{ik} - \sigma_k(v)\bigr) = \begin{cases} \sigma_{ij} - \sigma_j(x_j) = \sigma_{ij} & \text{if } j \in M_i \\ \infty & \text{otherwise.} \end{cases}$$

(iii) Using[3] $\sigma_j(v) = \max_{u\in U} \sigma_j(u)$,

$$\begin{aligned} \alpha_i(v) &= \min_{j\in M_i}\bigl(\sigma_{ij} - \sigma_j(v)\bigr) = -\max_{j\in M_i}\bigl(\sigma_j(v) - \sigma_{ij}\bigr) = -\max_{j\in M_i}\bigl(\max_{u\in U}\sigma_j(u) - \sigma_{ij}\bigr) \\ &= -\max_{j\in M_i}\max_{u\in U}\bigl(\sigma_j(u) - \sigma_{ij}\bigr) = -\max_{u\in U}\max_{j\in M_i}\bigl(\sigma_j(u) - \sigma_{ij}\bigr) \\ &= \min_{u\in U}\min_{j\in M_i}\bigl(\sigma_{ij} - \sigma_j(u)\bigr) = \min_{u\in U}\alpha_i(u). \end{aligned}$$

(iv) $\alpha_i(v) = \min_{j\in M_i}\bigl(\sigma_{ij} - (\sigma_j(u) + p)\bigr) = \alpha_i(u) - p$.

(v) If $v = f_i$, then $\alpha_i(v) = \min_{j\in M_i}\bigl((\sigma_{ij} - \sigma_j(f_i)\bigr) = \sigma_{ij} - \sigma_{ij} = 0$.
$\square$

[3]proved in [4]

4.3. QLity of a variable.

DEFINITION 4.6. *The* QLity *of a variable v in a code list of an f_i is*

$$\mathtt{Q}_i(v) = \begin{cases} \mathtt{I} & \textit{if } v \textit{ does not depend on any } y \in Y_i; \\ \mathtt{N} & \textit{if } v \textit{ depends nonlinearly on some } y \in Y_i; \quad \textit{and} \\ \mathtt{L} & \textit{if } v \textit{ depends only linearly on } y \in Y_i. \end{cases}$$

Remark 4.1. As in Definition 4.3, we mean "formal" dependence. For example v depends nonlinearly on x'' in $v = (x'')^2 + x'' + x\lambda - (x'')^2$, while the true dependence is linear.

If v is an algebraic function of a set U of variables u, denote

$$U_0 = \{\, u \in U \mid \alpha_i(u) = 0 \,\}. \tag{4.4}$$

We propagate QLity information based on the following lemma.

LEMMA 4.7.

(a) $\alpha_i(v) > 0$ *iff* $\mathtt{Q}_i(v) = \mathtt{I}$.

(b) If $\alpha_i(v) = 0$*, we have the following cases.*

(i) If v is an input variable x_j, then $\sigma_{ij} = 0$ and

$$\mathtt{Q}_i(x_j) = \mathtt{L}.$$

(ii) If v is an algebraic function ϕ of a set U of variables u then, see (4.4),

$$\mathtt{Q}_i(v) = \begin{cases} \mathtt{L} & \textit{if } \mathtt{Q}_i(u) = \mathtt{L} \textit{ for all } u \in U_0 \textit{ and } \phi \textit{ is linear in all } u \in U_0; \textit{ and} \\ \mathtt{N} & \textit{otherwise.} \end{cases}$$

(iii) If $v = d^p u/dt^p$, where $p > 0$, then

$$\mathtt{Q}_i(v) = \mathtt{L}.$$

(iv) If v is an output variable (corresponding to f_i), then $\mathtt{Q}_i(v) = \mathtt{L}$ *or* $\mathtt{N}$.

Proof.

(a) We have $\alpha_i(v) > 0$ iff $\sigma_{ij} > \sigma_j(v)$ for all $j \in M_i$ iff v does not depend on any $y \in Y_i$ iff $\mathtt{Q}_i(v) = \mathtt{I}$.

(b) If $\alpha_i(v) = 0$, then v depends on at least one $y \in Y_i$.

(i) If $v = x_j$ then $\sigma_{ij} = \sigma_j(x_j) = 0$. That is, the highest-order derivative of x_j in f_i is $\sigma_{ij} = 0$, and $\mathtt{Q}_i(x_j) = \mathtt{L}$.

(ii) If $\mathtt{Q}_i(u) = \mathtt{L}$ for all $u \in U_0$, and ϕ is linear in those u's, then v depends only linearly on $y \in Y_i$, and $\mathtt{Q}_i(v) = \mathtt{L}$. Otherwise, $\mathtt{Q}_i(u) = \mathtt{N}$ for some $u \in U_0$ or ϕ is nonlinear in some $u \in U_0$. In both cases, $\mathtt{Q}_i(v) = \mathtt{N}$.

(iii) Since $\alpha_i(v) = 0$, v will not be differentiated further in the code list (of f_i), and since $p > 0$, v depends linearly on its highest-order derivatives. Hence $\mathtt{Q}_i(v) = \mathtt{L}$.

(iv) This follows from (a) and definition 4.6.

∎

Since $\alpha_i(v) = 0$ implies v will not be differentiated further, we have

Corollary 4.8. *If $\mathtt{Q}_i(v) = \mathtt{N}$ for some intermediate v in a code list for f_i, then $\mathtt{Q}_i(f_i) = \mathtt{N}$.*

That is, we can conclude from the first v with $\mathtt{Q}_i(v) = \mathtt{N}$ that f_i is NQL.

Example 4.3. Consider $f_1 = A$ in (2.4). Here $M_1 = \{1, 3\}$, $Y_1 = \{x'', \lambda\}$; we wish to determine if A is QL in x'' and λ. In Table 4.2, $\mathtt{Q}_1(f_1) = \mathtt{L}$, and A is QL.

Table 4.2
Code list and propagation of offsets and QLities for $f_1 = A$

	code list		expression	$\alpha_1(v_r)$	$\mathtt{Q}_1(v_r)$
	v_{-5}	$= x$	x	2	I
	v_{-3}	$= \lambda$	λ	0	L
	v_1	$= v''_{-5}$	x''	0	L
	v_2	$= v_{-5}v_{-3}$	$x\lambda$	0	L
$f_1 =$	v_3	$= v_1 + v_2$	$x'' + x\lambda$	0	L

Example 4.4. Now consider F. We have $M_6 = \{3, 4\}$, $Y_6 = \{u, \lambda''\}$, and we need to determine if F is QL in λ'' and u. In Table 4.3 $\mathtt{Q}_6(v_1) = \mathtt{N}$, and we can conclude from it that F is NQL.

Table 4.3
Code list and propagation of offsets and QLities for $f_6 = F$

	code list		expression	$\alpha_6(v_r)$	$\mathtt{Q}_6(v_r)$
	v_{-3}	$= \lambda$	λ	2	I
	v_{-2}	$= u$	u	0	L
	v_{-1}	$= v$	v	∞	I
	v_1	$= v_{-2}^2 + v_{-1}^2$	$u^2 + v^2$	0	N
	v_2	$= (L + cv_{-3})^2$	$(L + c\lambda)^2$	2	I
	v_3	$= \lambda''$	λ''	0	L
$f_6 =$	v_4	$= v_1 - v_2 + v_3$	$u^2 + v^2 - (L + c\lambda)^2 + \lambda''$	0	N

4.4. Quasilinearity analysis algorithm. From §4.2 and §4.3, we derive in Figure 4.1 algorithm QL_ANALYSIS, which determines if an f_i is QL/NQL. Since $\alpha_i(v) > 0$ iff $\mathtt{Q}_i(v) = \mathtt{I}$, in practice, we need to propagate only $\mathtt{L}$ and $\mathtt{N}$.

4.5. Quasilinearity of a fine block. At local stage $k_l > 0$, system (3.13) is linear in $\mathbf{x}_{J^l_k}$. We are interested in determining if it is linear in $\mathbf{x}_{J^l_k}$ at local stage $k_l \le 0$. Similarly to the definition of Y_i in (4.1), used to define quasilinearity of an f_i in (3.13), let

$$Z_i = \left\{ x_j^{(\sigma_{ij})} \mid j \in B_l \text{ and } \sigma_{ij} = d_j - c_i \right\}.$$

Definition 4.9. *Equation $f_i = 0$ with $i \in B_l$ is QL, if it is linear in all $y \in Z_i$, and NQL otherwise.*

Definition 4.10. *System (3.13) is NQL if it contains an undifferentiated NQL f_i, and QL otherwise.*

When considering block l, algorithm QL_ANALYSIS can be readily adapted by changing the line

$$M_i \leftarrow \{ j \mid \sigma_{ij} = d_j - c_i \} \quad \text{to} \quad M_i \leftarrow \left\{ j \mid j \in B_l \text{ and } \sigma_{ij} = d_j - c_i \right\}$$

```
Algorithm. QL_ANALYSIS
INPUT
    code list for evaluating an $f_i$ in (1.1), $\sigma_{ij}$ for all $j = 1 : n$
OUTPUT
    $\mathtt{Q}_i(f_i)$
COMPUTE
    $M_i \leftarrow \{ j \mid \sigma_{ij} = d_j - c_i \}$
    for each variable $v$ in the code list
        if $v$ is an input variable $x_j$ then
            if $j \in M_i$ then
                $\alpha_i(v) \leftarrow \sigma_{ij}$
                if $\alpha_i(v) = 0$ then $\mathtt{Q}_i(v) \leftarrow \mathtt{L}$
            else $\alpha_i(v) \leftarrow \infty$
        elseif $v$ is $t$ then
            $\alpha_i(v) \leftarrow \infty$
        elseif $v$ is an algebraic function $\phi$ of a set $U$ of previous variables $u$ then
            $\alpha_i(v) \leftarrow \min_{u \in U} \alpha_i(u)$
            if $\alpha_i(v) = 0$
                $U_0 \leftarrow \{ u \in U \mid \alpha_i(u) = 0 \}$
                if $\mathtt{Q}_i(u) = \mathtt{L}$ for all $u \in U_0$ and $\phi$ is linear in $U_0$ then
                    $\mathtt{Q}_i(v) \leftarrow \mathtt{L}$
                else $\mathtt{Q}_i(f_i) \leftarrow \mathtt{N}$, return
        else        % $v$ is $d^p u/dt^p$
            $\alpha_i(v) \leftarrow \alpha_i(v) - p$
            if $\alpha_i(v) = 0$ then $\mathtt{Q}_i(v) \leftarrow \mathtt{L}$
```

FIG. 4.1. *Algorithm for determining if an f_i in the basic scheme is QL.*

and keeping the rest of this algorithm the same.

Example 4.5. Consider block $l = 3$. For equation $f_3 = F = 0$, $3 \in B_3$. Then

$$M_3 = \left\{ j \mid j \in B_3 \text{ and } \sigma_{3,j} = d_j - c_3 \right\} = \{ 3 \},$$

and we need to determine if F is QL in $x_3 = u$. Now the index of $x_6 = \lambda$ is $6 \notin M_3$ and therefore $\lambda'' \notin Z_3$; cf. Example 4.4. We show in Table 4.4 the initialization and propagation of offsets and QLity information for the code list in Table 4.3.

TABLE 4.4
Code list and propagation of offsets and QLities for $f_3 = F$ of block 3 in the BTF of (2.4)

	code list		expression	$\alpha_3(v_r)$	$\mathtt{Q}_3(v_r)$
	v_{-5}	$= v$	v	∞	I
	v_{-3}	$= u$	u	0	L
	v_0	$= \lambda$	λ	∞	I
	v_1	$= v_{-3}^2 + v_{-1}^2$	$u^2 + v^2$	0	N
	v_2	$= (L + cv_{-3})^2$	$(L + c\lambda)^2$	2	I
	v_3	$= \lambda''$	λ''	0	L
$f_3 =$	v_4	$= v_1 - v_2 + v_3$	$u^2 + v^2 - (L + c\lambda)^2 + \lambda''$	0	N

5. Overall algorithm.

In §5.1 we present first an algorithm for finding the indices of derivatives that need to be initialized, and then in §5.2 we give an overall algorithm implementing the block solution scheme.

5.1. Initialization.

Consider block l. For k such that $k_l = k + K_l < 0$, we need to initialize $\mathbf{x}_{J^l_k}$, that is, give values for

$$\big\{ x_j^{(r)} \mid j \in B_l \text{ and } r = k + d_j \geq 0 \big\}. \tag{5.1}$$

We have $k + d_j = k_l + \widehat{d}_j \geq 0$ when

$$k \geq -\max_{j \in B_l} d_j \quad \text{or equivalently} \quad k_l \geq -\max_{j \in B_l} \widehat{d}_j.$$

When $k_l < -\max_{i \in B_l} \widehat{c}_i$, there are no equations at this stage and in this block, and (5.1) are *initial values*, otherwise (5.1) are *initial guesses* (trial values).

When $k_l = 0$ and (3.13) is NQL (see §4.5), also initial guesses for

$$\big\{ x_j^{(\widehat{d}_j)} \mid j \in B_l \big\}$$

are required.

This is expressed by algorithm FINE_BLOCK_INITIALIZATION, Figure 5.1, which finds the set of indices (j, r) for derivatives $x_j^{(r)}$ that require initial values and the set of indices for derivatives that require initial guesses. By [9, Theorem 4.5], these are minimal sets.

```
Algorithm. FINE_BLOCK_INITIALIZATION
INPUT
    local canonical offsets $c_i^*$, $d_j^*$,   $i, j = 1 : N_l$
    $\gamma = 1$ if block is QL and 0 otherwise
    $b = \sum_{i=1}^{l-1} N_i$, block $l$ starts at index $b + 1$
OUTPUT
    $J_v$  set of indices for derivatives that require initial values
    $J_g$  set of indices for derivatives that require initial guesses
COMPUTE
    $J_v \leftarrow \emptyset$
    $J_g \leftarrow \emptyset$
    for $q \leftarrow -\max d_j^*$ to $-\gamma$
        $M \leftarrow \{ j \mid q + d_j^* \geq 0 \}$
        $J \leftarrow \big\{ (j + b, q + d_j^*) \mid j \in M \big\}$
        if $q < -\max c_i^*$ then        (no equations)
            $J_v \leftarrow J_v \cup J$
        else $J_g \leftarrow J_g \cup J$
```

FIG. 5.1. *Initialization for a fine block. To simplify the notation in this algorithm, for block l we denote by c_i^* and d_j^* its local canonical offsets and assume they are indexed from 1 to N_l; that is $c_i^* = \widehat{c}_{b+i}$ and $d_j^* = \widehat{d}_{b+j}$.*

Example 5.1. For the blocks in our example, this algorithm produces sets as follows.

– Block $l = 4$ is QL, $\gamma = 1$, $b = 3$, $\max d_j^* = \max c_i^* = 2$. When $q = -2$, $M = \{ 1, 2 \}$, $J_g = J = \{ (4, 0), (5, 0) \}$; when $q = -1$, M is the same, $J = \{ (4, 1), (5, 1) \}$, and

$$J_g = \{ (4, 0), (5, 0), (4, 1), (5, 1) \} \quad \text{and} \quad J_v = \emptyset.$$

– Block $l = 3$ is NQL, $\gamma = 0$, $b = 2$, $\max d_j^* = 0$, and $\max c_i^* = 0$. We have one iteration with $q = 0$, $M = \{ 1 \}$, and

$$J_g = \{ (3, 0) \} \quad \text{and} \quad J_v = \emptyset.$$

– Block $l = 2$ is QL, $\gamma = 1$, $\max d_j^* = 0$, the loop does not execute, and $J_g = \emptyset$ and $J_v = \emptyset$
– Block $l = 1$ is NQL, $\gamma = 0$, $b = 0$, $\max d_j^* = 3$, and $\max c_i^* = 0$. After iterations $q = -3, -2, -1$, we have

$$J_v = \{ (1, 0), (1, 1), (1, 2) \},$$

and after iteration $q = 0$,

$$J_g = \{ (1, 3) \};$$

$M = \{ 1 \}$ through all iterations.

Since the variables are in the order v, u, μ, x, y, λ, the above J_g and J_v sets imply that

$x, y,\ x', y'\ u,\ v'''$	require initial guesses and;
v, v', v''	require initial values.

cf. Table 3.1.

5.2. Block scheme algorithm. Here we assume that Σ, fine BTF, canonical local and global offsets are computed, the quasilinearity of each f_i is determined, and appropriate derivatives are given initial values or guesses (according algorithm QL_ANALYSIS). Also, we assume that the K_l are available (see the companion paper). Finally, algorithm SOLVING_FOR_DERIVATIVES, Figure 5.2, gives the algorithm for stage k of the block scheme.

6. Conclusion.

We showed how to perform QL analysis of a DAE and how to combine it with its fine block triangularization to obtain a solution scheme. Provided the problem can be decomposed into many small fine blocks, this scheme results in solving much smaller problems compared to the basic scheme. For instance, for the distillation column in [6], we have 52 blocks of size 1 and 11 blocks of size 7. With the basic scheme, we would solve systems of size 129 at stages ≥ 0, while with the block scheme, the largest system is of size 7. Furthermore, a DAE that is NQL may decompose into subproblems that are all QL: this is the case e.g. with the Chemical Akzo Nobel problem, a NQL DAE of size 6 that decomposes into 6 linear equations of size 1; see [6, 9].

In the context of computing a numerical solution using Taylor series, the block scheme has the advantage that we could exploit multiple processing units (CPUs, cores) to pipeline the computation of Taylor coefficients (TCs). Profiling the DAETS solver [5] has shown that typically more than 80%, and in some cases above 90%, of the total computing time is spent in automatic differentiation (AD) for evaluating TCs for the f_i's. Our hope is that, by assigning blocks of the DAE to different processing

```
Algorithm. SOLVING_FOR_DERIVATIVES
INPUT
    $k$ stage number
    $K_l$ for each block $l$
    $\widehat{c}$, $\widehat{d}$ local canonical offsets
    $\gamma_i = 1$ if $f_i$ is QL, 0 otherwise, $i = 1:n$
    $B_l$ for each block $l$
OUTPUT
    $\mathbf{x}_{J_k} = (\mathbf{x}_{J_k^1}, \dots, \mathbf{x}_{J_k^p})$
COMPUTE
    for $l = p, p-1, \dots, 1$
        $k_l \leftarrow k + K_l$
        if $k_l < -\max_{i \in B_l} \widehat{c}_i$ then continue
        $I_k^l \leftarrow \{ (i,r) \mid i \in B_l \text{ and } r = k_l + \widehat{c}_i \geq 0 \}$
        $J_k^l \leftarrow \{ (j,r) \mid j \in B_l \text{ and } r = k_l + \widehat{d}_j \geq 0 \}$
        if there is $(i,0) \in I_k^l$ and $\gamma_i = 0$ then
            qlflag $\leftarrow$ false
        else qlflag $\leftarrow$ true
        if $k_l < 0$ then
            if qlflag = false
                solve min $\|\widetilde{\mathbf{x}}_{J_k^l} - \mathbf{x}_{J_k^l}\|_2$ s.t. nonlinear $\mathbf{F}_{I_k^l}(\mathbf{x}_{J_k^l}) = 0$
            else solve min $\|\widetilde{\mathbf{x}}_{J_k^l} - \mathbf{x}_{J_k^l}\|_2$ s.t. linear $\mathbf{F}_{I_k^l}(\mathbf{x}_{J_k^l}) = 0$
        elseif $k_l = 0$ and qlflag = false then
            solve nonlinear $\mathbf{F}_{I_k^l}(\mathbf{x}_{J_k^l}) = 0$
        else solve linear $\mathbf{F}_{I_k^l}(\mathbf{x}_{J_k^l}) = 0$
```

FIG. 5.2. *Algorithm for computing derivatives at stage k using fine BTF of the DAE. When $k_l < 0$, $\mathbf{F}_{I_k^l}(\mathbf{x}_{J_k^l}) = 0$ is underdetermined and determined otherwise.*

units and computing in a block-wise manner, we can reduce the overall time in AD to compute these coefficients. We outline this pipelining idea here and discuss briefly some of the challenges involved.

Using AD, to evaluate the $(k+c_i)$ TC of an f_i requires $O(k+c_i)$ operations, where the constant in the big-O notation depends on the number of nonlinear operations ($\times$, $\div$, sin, exp, etc.) in f_i [1, 2]. (The work is $O(1)$ if f_i contains only $+, -$, and d^p/dt^p.)

Denote this constant by τ_i and write $O(k + c_i) = \tau_i(k + c_i)$. Using the basic scheme, at stage $k \geq 0$ we need to evaluate N such coefficients. Thus, the work is

$$\sum_{i=1}^{N} \tau_i(k + c_i) = k \sum_{i=1}^{N} \tau_i + \sum_{\substack{i=0 \\ c_i \neq 0}}^{N} \tau_i c_i = \tau k + \sum_{\substack{i=0 \\ c_i \neq 0}}^{N} \tau_i c_i = O(\tau k),$$

where $\tau = \sum_{i=1}^{N} \tau_i$ depends on the problem, and if the c_i's are canonical, $\sum_{\substack{i=0 \\ c_i \neq 0}}^{N} \tau_i c_i$ depends on the problem as well. To evaluate $q+1$ coefficients for stages $k = 0:q$, the work in AD is $\sum_{k=0}^{q} O(\tau k) = O(\tau q^2)$.

For our example problem, we could assign the 3×3 fine block to one core, say P2, and the coarse block consisting of the 3 fine blocks to another core, say P1. Then when P2 is at stage k, P1 would be at stage $k-1$. When P2 finishes its last stage q, P1 needs to complete stage q. If we assume that the work per each of these cores/blocks is $O(\tau q^2/2)$, that is the constant is $\tau/2$, P2 would finish in $O(\tau q^2/2)$ and P1 would finish in $O(\tau q/2)$ after P2. Under these simplistic assumptions, and ignoring rest of the work (e.g. in linear algebra), we would expect the computation time to be reduced to about half.

It is desirable to assign blocks to processing units and perform a pipelined computation, such that each stage of the pipeline takes about the same amount of time and there are no "gaps" in the pipeline. However, developing such a method is a challenging task on its own.

One difficulty is estimating the work per TC of an f_i. Another is, if the number of fine blocks is much larger than the number of processing units, how to agglomerate blocks such that each unit does nearly the same amount of work. A third challenge comes from the structure of the problem. For example, assume that fine blocks 3, 2, and 1 are assigned to cores P3, P2, and P1, respectively. In Table 6.1, P3 needs

Table 6.1
Pipelining blocks 3,2, and 1. $k, w : z$ means solving at stage k for w which depends (at least) on z.

core	time slots $1, 2, \dots$							
P3	$0, u'' : v''$	$\times$	$\times$	$1, u''' : v'''$	$\times$	$\times$	$2, u^{(4)} : v^{(4)}$	$\cdots$
P2		$0, \mu : u''$	$\times$	$\times$	$1, \mu' : u'''$	$\times$	$\times$	$\cdots$
P1			$0, v''' : \mu$	$\times$	$\times$	$1, v^{(4)} : \mu'$	$\times$	$\cdots$

$v^{(k)}$ to find $u^{(k)}$, and P1 needs $\mu^{(k)}$, which in turn depends on $u^{(k)}$ to find $v^{(k)}$. That is, P3 has to wait for $v^{(k)}$ to be determined by P1, which leads to the above gaps, marked by $\times$.

Appendix A. QL analysis: implementation issues.

Algorithm QL_ANALYSIS (Figure 4.4) can be implemented using operator overloading or source code translation as follows.

In an initialization phase, we associate with each input variable x_j an n-vector with ith component

$$\alpha_i(x_j) = \begin{cases} \sigma_{ij} & \text{if } \sigma_{ij} = d_j - c_i \text{ and } i \text{ and } j \text{ in the same block} \\ \infty & \text{otherwise;} \end{cases}$$

see Table A.1. According to Lemma 4.5, for a unary operator, the offset vector remains the same; for each binary operator, we take a component-wise minimum of the associated offset vectors; and for the d^p/dt^p operator, we can subtract p from each component. Since

$$\mathtt{Q}_i(x_j) = \begin{cases} \mathtt{L} & \text{if } \alpha_i(x_j) = 0 \text{ and} \\ \mathtt{I} & \text{otherwise,} \end{cases}$$

we can assume that offset 0 encodes `L` and offset > 0 encodes `I`. Further, we can use e.g. "offset" -1 to encode `N`.

For an intermediate variable v, if $\mathtt{Q}_i(v) = \mathtt{N}$, then the subsequent intermediate variables in the code list for the f_i will be of QLity `N`. Hence, for a v with $\mathtt{Q}_i(v) = \mathtt{N}$, we can set the ith component of its offset vector to -1. Then, if the ith component of the offset vector of f_i is 0, $\mathtt{Q}_i(f_i) = \mathtt{L}$, and if it is -1, $\mathtt{Q}_i(f_i) = \mathtt{N}$.

In passing, we note that, if a variable v is in the code list of f_i but not in the code list of f_j, $i \neq j$, then $\alpha_j(v)$ is ignored. However, we propagate the whole n-vector, as we do not know in advance (in particular with operator overloading) if v appears in the code list of more than one f_i.

Table A.1
Propagation of variable offsets and QLity information through code list of (2.4). The QLity of each equation (output variable) is marked by ▨ *; $-$ denotes ∞.*

code list			expression	A	B	C	D	E	F
				α_1	α_2	α_3	α_4	α_5	α_6
	v_{-5}	$= x$	x	2	–	0	–	–	–
	v_{-4}	$= y$	y	–	2	0	–	–	–
	v_{-3}	$= \lambda$	λ	0	0	–	–	–	–
	v_{-2}	$= u$	u	–	–	–	–	–	0
	v_{-1}	$= v$	v	–	–	–	–	3	–
	v_0	$= \mu$	μ	–	–	–	0	–	–
	v_1	$= v''_{-5}$	x''	0	–	−2	–	–	–
	v_2	$= v_{-5}v_{-3}$	$x\lambda$	0	0	0	–	–	–
A	$= v_3$	$= v_1 + v_2$	$x'' + x\lambda$	▨ 0	0	−2	–	–	–
	v_4	$= v''_{-4}$	y''	–	0	−2	–	–	–
	v_5	$= v_{-4}v_{-3}$	$y\lambda$	0	0	0	–	–	–
	v_6	$= v_4 + v_5$	$y'' + y\lambda$	0	0	−2	–	–	–
	v_7	$= v'_{-5}$	x'	1	–	−1	–	–	–
	v_8	$= v_7^2$	$(x')^2$	1	–	−1	–	–	–
	v_9	$= v_6 + v_8$	$y'' + y\lambda + (x')^2$	0	0	−2	–	–	–
B	$= v_{10}$	$= v_9 - G$	$y'' + y\lambda + (x')^2 - G$	0	▨ 0	−2	–	–	–
	v_{11}	$= v_{-5}^2$	x^2	2	–	−1	–	–	–
	v_{12}	$= v_{-4}^2$	y^2	–	2	−1	–	–	–
	v_{13}	$= v_{11} + v_{12}$	$x^2 + y^2$	2	2	−1	–	–	–
C	$= v_{14}$	$= v_{13} - L^2$	$x^2 + y^2 - L^2$	2	2	▨ −1	–	–	–
	v_{15}	$= v''_{-2}$	u''	–	–	–	–	–	−2
	v_{16}	$= v_{-2}v_0$	$u\mu$	–	–	–	0	–	0
D	$= v_{17}$	$= v_{15} + v_{16}$	$u'' + u\mu$	–	–	–	▨ 0	–	−2
	v_{18}	$= v'''_{-1}$	v'''	–	–	–	–	0	–
	v_{19}	$= (v''')^2$	$(v''')^2$	–	–	–	–	−1	–
	v_{20}	$= v_{-1}v_0$	$v\mu$	–	–	–	0	3	–
	v_{21}	$= v_{19} + v_{20}$	$(v''')^2 + v\mu$	–	–	–	0	−1	–
E	$= v_{22}$	$= v_{21} - G$	$(v''')^2 + v\mu - G$	–	–	–	0	▨ −1	–
	v_{23}	$= v_{-2}^2$	u^2	–	–	–	–	–	−1
	v_{24}	$= v_{-1}^2$	v^2	–	–	–	–	3	–
	v_{25}	$= v_{23} + v_{24}$	$u^2 + v^2$	–	–	–	–	3	−1
	v_{26}	$= cv_{-3}$	$c\lambda$	0	0	–	–	–	–
	v_{27}	$= L + v_{26}$	$L + c\lambda$	0	0	–	–	–	–
	v_{28}	$= v_{27}^2$	$(L + c\lambda)^2$	−1	−1	–	–	–	–
	v_{29}	$= v_{25} - v_{28}$	$u^2 + v^2 - (L + c\lambda)^2$	−1	−1	–	–	3	−1
	v_{30}	$= v_{-3}$	λ''	−2	−2	–	–	–	–
F	$= v_{31}$	$= v_{29} + v_{30}$	$u^2 + v^2 - (L + c\lambda)^2 + \lambda''$	−2	−2	–	–	3	▨ −1

REFERENCES

[1] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, Philadelphia, PA, USA, second ed., 2008.

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

[3] N. S. Nedialkov and J. D. Pryce, *Solving differential-algebraic equations by Taylor series (I): Computing Taylor coefficients*, BIT Numerical Mathematics, 45 (2005), pp. 561–591.
[4] ———, *Solving differential-algebraic equations by Taylor series (II): Computing the system Jacobian*, BIT Numerical Mathematics, 47 (2007), pp. 121–135.
[5] ———, *Solving differential algebraic equations by Taylor series (III): the DAETS code*, JNAIAM J. Numer. Anal. Indust. Appl. Math, 3 (2008), pp. 61–80.
[6] N. S. Nedialkov, J. D. Pryce, and G. Tan, *DAESA — a Matlab tool for structural analysis of DAEs: Software*, ACM Transactions on Mathematical Software, to appear (2014). 15 pages.
[7] J. D. Pryce, *A simple structural analysis method for DAEs*, BIT, 41 (2001), pp. 364–394.
[8] J. D. Pryce, N. S. Nedialkov, and G. Tan, *Graph theory, irreducibility, and structural analysis of differential-algebraic equation systems.* Submitted to SIAM J. Sci. Comput., November 2014, 24 pages.
[9] J. D. Pryce, N. S. Nedialkov, and G. Tan, *DAESA — a Matlab tool for structural analysis of DAEs: Theory*, ACM Transactions on Mathematical Software, to appear (2014). 20 pages.