Algorithm XXX: DAESA — a Matlab Tool for Structural Analysis of Differential-Algebraic Equations: Software

NEDIALKO S. NEDIALKOV, McMaster University
JOHN D. PRYCE, Cardiff University
GUANGNING TAN, McMaster University

DAESA, Differential-Algebraic Equations Structural Analyzer, is a MATLAB tool for structural analysis of differential-algebraic equations (DAEs). It allows convenient translation of a DAE system into MATLAB and provides a small set of easy-to-use functions. DAESA can analyze systems that are fully nonlinear, high-index, and of any order. It determines structural index, number of degrees of freedom, constraints, variables to be initialized, and suggests a solution scheme. The structure of a DAE can be readily visualized by this tool. It can also construct a block-triangular form of the DAE, which can be exploited to solve it efficiently in a block-wise manner.

Categories and Subject Descriptors: I.6.7 [**Computing methodologies**]: Simulation support systems; G.4 [**Mathematical software**]: Matlab, Algorithm design and analysis

Additional Key Words and Phrases: Differential-algebraic equations, structural analysis, modeling

ACM Reference Format:
Nedialkov, N., Pryce, J, Tan, G., 2012. DAESA— A MATLAB Tool for Structural Analysis of DAEs: Software ACM Trans. Math. Softw. V, N, Article B (YYYY), 21 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

We present DAESA, Differential-Algebraic Equations Structural Analyzer, a MATLAB tool for structural analysis (SA) of differential-algebraic equations (DAEs). It allows convenient translation of a DAE into MATLAB and provides (currently 18) easy-to-use functions for determining the structural index, the number of degrees of freedom (henceforth referred to as the DOF), the constraints and a solution scheme, and for visualizing the structure of the DAE. DAESA can also construct a block-triangular form (BTF), which can be exploited for efficient solution in a block-wise fashion.

Our package is applicable to DAE systems of the general form

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

where t is the independent variable, and the $x_j(t)$ are n state variables. The formulation (1) includes high-order systems and systems that are nonlinear in leading deriva-

Author's addresses: N. Nedialkov and G. Tan, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada. J. Pryce, Cardiff School of Mathematics, Cardiff University, UK.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

tives. Furthermore, (1) includes systems of ordinary differential equations (ODEs) and pure algebraic systems.

DAESA performs analysis similar to that of the C++ solver DAETS [Nedialkov and Pryce 2008; Nedialkov and Pryce 2009]. However, DAETS is not suitable for rapid investigation of DAEs, as it requires C++ knowledge and compiling the user code. The goal of this work is to produce a light-weight, easy-to-use tool with convenient facilities for rapidly exploring a DAE's structure. The present tool is based on Pryce's SA [Pryce 2001] and recent developments by the authors on improving this analysis using block-triangularization of the DAE. The choice of MATLAB is due to its ubiquity and ease-of-use, as well as its operator overloading, which is central to our implementation.

We do not define the terminology we use here, nor present the underlying theory—for this see the companion paper [Pryce et al. 2013] and the references therein.

This article is organized as follows. Section 2 gives an overview of DAESA. Section 3 presents several examples of analyzing DAEs. Section 4 investigates the performance of this package and the relative amount of work of various parts of it. Section 5 is a study of applying DAESA to a large number (1616 in total) of "pseudo-random" DAEs generated from matrices from the Florida Sparse Florida Sparse Matrix Collection [Davis and Hu 2011]. Conclusions are given in Section 6.

2. OVERVIEW OF DAESA

DAESA exploits MATLAB's operator overloading to process a DAE given by a user-supplied function for evaluating the f_i in (1). In particular, it extracts the *signature matrix*[1] and determines for each equation if it is *quasilinear* in the leading derivatives in the sense explained in the companion paper, see also [Nedialkov and Pryce 2008].

After the signature matrix is constructed, DAESA finds out if the problem is *structurally well-posed*, and if so, solves a *linear assignment problem* to calculate the *offsets* of the problem, and then determines *structural index* and *DOF*. Since it knows the structure of the analyzed DAE, DAESA reduces it to *block triangular form (BTF)*, finds *local offsets*, and determines *block by block quasilinearity*. Based on the offsets and linearity information, DAESA deduces which variables and derivatives of them need to be *initialized* and what the *constraints* are.

This package provides functions for reporting the constraints, initialization summary, and a solution scheme, and functions for displaying the original structure of the DAE, as well as for displaying *coarse* and *fine* BTFs of the DAE structure.

The DAESA package builds around three classes: `sigma`, `qla`, and `SAdata`. The signature matrix is obtained by executing the function defining the DAE with objects of the class `sigma`. The quasilinearity analysis is carried out by executing this function with objects of the class `qla`. In both cases, the processing of the DAE is done through operator overloading of the arithmetic operators and the elementary functions. The SA is performed by the function `daeSA`. It returns an object of the class `SAdata`, which encapsulates all the data obtained from the SA. Each of the remaining DAESA functions takes an object of this class as a parameter and extracts from it the data it needs. This mechanism (of the main function returning an object, and the remaining functions querying this object) ensures simple and consistent function interfaces.

[1]Concepts explained in the companion paper are typeset in slanted font on first occurrence.

Remark. The structural index computed by DAESA is an upper bound on the differentiation index, and often they are the same. Although successful on many problems of interest, the underlying SA theory (and DAESA) may fail to determine the correct structural, and therefore differentiation index on some problems; see e.g. [Pryce 2001; Nedialkov and Pryce 2007].

We would have a "certificate" that the SA is successful if the *system Jacobian* **J** is nonsingular at a *consistent point* [Nedialkov and Pryce 2007]. The present tool does not compute consistent points and does not evaluate **J**: it performs symbolic-type analysis of DAEs. We plan to incorporate the evaluation of **J** in a future version of DAESA.

3. DAESA EXAMPLES

In this section, we illustrate some of the capabilities of DAESA. Namely, we show SA on a simple DAE, the chemical Akzo Nobel problem [Mazzia and Iavernaro 2003] (§3.1), present results from analyzing a DAE arising from modeling a distillation column (§3.2), and show how DAESA reports structurally ill-posed (SIP) problems (§3.3). For details, see the DAESA user guide [McKenzie et al. 2013], which is also part of the distribution of this package.

3.1. Simple DAE: Chemical Akzo Nobel

We show DAESA's SA on the chemical Akzo Nobel problem, an index-1 DAE. The equations for this problem are given in §5.1 of the companion paper. The DAESA function for evaluating them is displayed in Figure 1.

```
function f = akzonobel(t,y)
 k1 = 18.7; k2 = 0.58;   k3 = 0.09; k4 = 0.42;
 K  = 34.4; klA = 3.3; CO2 = 0.9;    H = 737;
 Ks = 115.83;
 r1  = k1*y(1)^4*sqrt(y(2));
 r2  = k2*y(3)*y(4);
 r3  = k2/K*y(1)*y(5);
 r4  = k3*y(1)*y(4)^2;
 r5  = k4*y(6)^2*sqrt(y(2));
 Fin = klA*(CO2/H - y(2));
 f(1) = -Dif(y(1),1) - 2.0*r1 + r2 - r3 - r4;
 f(2) = -Dif(y(2),1) - 0.5*r1 - r4 - 0.5*r5 + Fin;
 f(3) = -Dif(y(3),1) + r1 - r2 + r3;
 f(4) = -Dif(y(4),1) - r2 + r3 - 2.0*r4;
 f(5) = -Dif(y(5),1) + r2 - r3 + r5;
 f(6) = Ks*y(1)*y(4) - y(6);
end
```

Fig. 1. DAESA function for evaluating the chemical Akzo Nobel problem

Structural analysis is performed by the call `daeSA(@akzonobel,6)`. The structure of a DAE, and its coarse and fine BTFs, are displayed graphically by the `showStruct` function. In Figure 2 we show the fine BTF for this problem as produced by the call `showStruct(sadata,'disptype','fineblocks')`: it decomposes into six fine blocks.

The solution scheme, for computing derivatives of the solution, is produced by `printSolScheme`, the 'compact' form of whose output is on the left of Figure 3. This

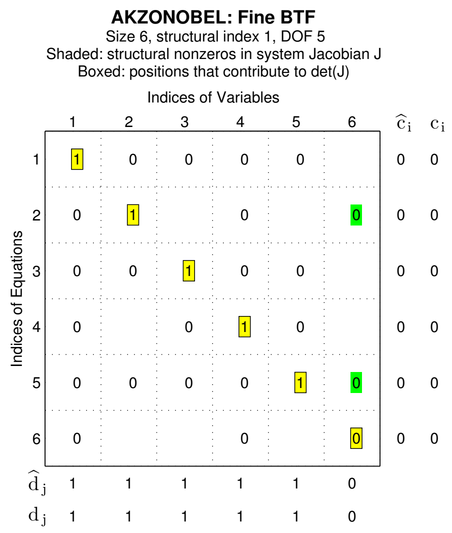

AKZONOBEL: Fine BTF
Size 6, structural index 1, DOF 5
Shaded: structural nonzeros in system Jacobian J
Boxed: positions that contribute to det(J)
Indices of Variables
Indices of Equations
1 2 3 4 5 6 $\widehat{c}_i$ c_i
1: 1 0 0 0 0 | 0 0
2: 0 1 0 0 | 0 0
3: 0 0 1 0 0 | 0 0
4: 0 0 1 0 | 0 0
5: 0 0 0 0 1 0 | 0 0
6: 0 0 0 | 0 0
$\widehat{d}_j$ 1 1 1 1 1 0
d_j 1 1 1 1 1 0

Fig. 2. Fine BTF and solution scheme for the chemical Akzo Nobel problem

```
Solution scheme for 'akzonobel' problem
-----------------------------------------------------------------------
Initialization summary:
y1, y2, y3, y4, y5
-----------------------------------------------------------------------
k = -1:  [] : y5
         [] : y4
         [] : y3
         [] : y2
         [] : y1
k =  0:  [f6] : y6
         [f5] : y5'
         [f4] : y4'
         [f3] : y3'
         [f2] : y2'
         [f1] : y1'
```

$k=-1$	Give y_1, y_2, y_3, y_4, y_5 as initial values							
$k=0$	solve	for	using					
			initial values					computed
	$f_6=0$	y_6	y_1			y_4		
	$f_5=0$	y_5'	y_1	y_2	y_3	y_4		y_6
	$f_4=0$	y_4'	y_1		y_3		y_5	
	$f_3=0$	y_3'	y_1	y_2		y_4	y_5	
	$f_2=0$	y_2'	y_1			y_4		y_6
	$f_1=0$	y_1'		y_2	y_3	y_4	y_5	

Fig. 3. 'Compact' solution scheme for the chemical Akzo Nobel problem and inset, extra data provided by the 'full' solution scheme.

says solution may be performed by giving initial values (IVs) for y_1, y_2, y_3, y_4 and y_5 at stage -1. Then using these IVs, at stage $k=0$ we solve six linear scalar equations, one per fine block. More detail is given by the 'full' form of the output, summarized in the inset: e.g., it says one solves $f_5=0$ for y_5', using the values of y_1 to y_4 given as IVs, and the previously computed value of y_6.

The computations at stage $k>0$ are deduced from stage $k=0$. That is, we solve first $f_6^{(k)}=0$ for $y_6^{(k)}$ and then solve $f_i^{(k)}=0$ for $y_i^{(k+1)}$ in the order $i=5,4,3,2,1$—all linear, and using previously computed values for the derivatives of the solution.

In [Mazzia and Iavernaro 2003], this problem is classified as nonlinear, requiring IVs for all variables and their first derivatives. Without the information produced by our BTF it would be regarded as fully coupled, needing solution of a non-linear system of size six to determine values for $y_i^{(k+1)}$, $i=1,\ldots,5$, and $y_6^{(k)}$ at stage $k=0,1,\ldots$.

3.2. Chemical engineering application: distillation column

Systems of DAEs are commonly used in chemical engineering to describe mass/energy balances and constitutive relations, such as mass/energy transfer rates, reaction kinetics, thermodynamic properties and relations and control laws. Depending on the assumptions used to construct the model, a DAE system with an index exceeding one is possible. Then typically an index reduction is performed, and the resulting system is simulated with a standard, index-1 DAE solver (e.g., SUNDIALS [Hindmarsh et al. 2005] or DASSL [Brenan et al. 1996]).

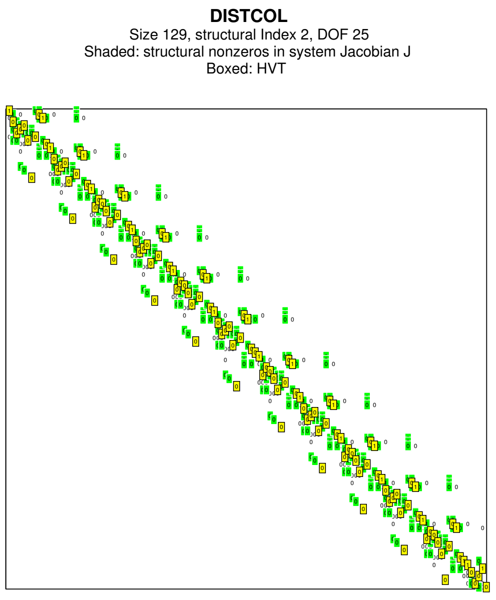
From a modeler's perspective, one might be interested in determining the indices of equations that need to be differentiated to reduce to a lower-index DAE. These are

$$\{i \mid c_i > 0,\ i = 1, \ldots, n\},$$

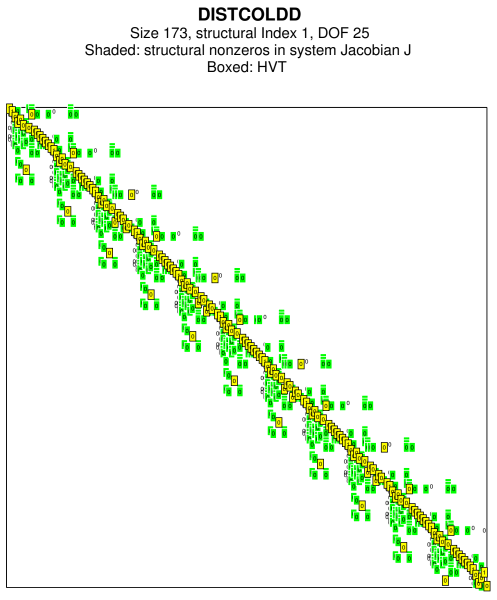
where c_i is the global offset of equation i, and they can be readily obtained by the `getOffsets` function of DAESA.

In this section, we show results of DAESA's SA on a separation model[2] for a distillation column with a partial reboiler and total condenser. The present model is an index-2 system of 129 equations. We refer to this system as DISTCOL. In [Washington and Swartz 2011], it is reduced to index-1 using dummy derivatives [Mattsson and Söderlind 1993], resulting in 173 equations, and then simulated with SUNDIALS [Hindmarsh et al. 2005]. We refer to this system as DISTCOLDD. Their structures are shown in Figure 4.

[2]This model is provided by Ian Washington of the Department of Chemical Engineering of McMaster University.

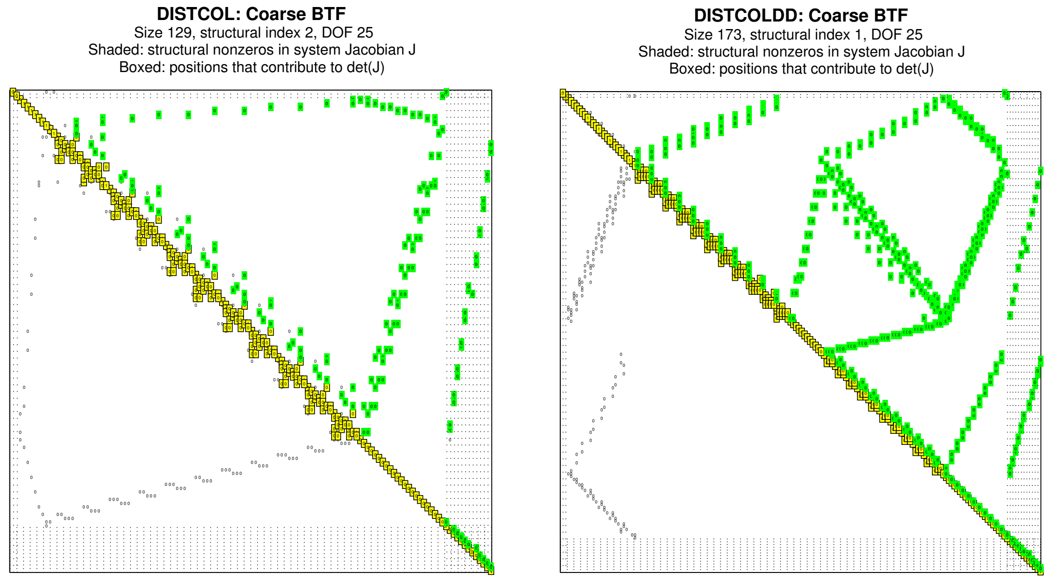


(a) DISTCOL



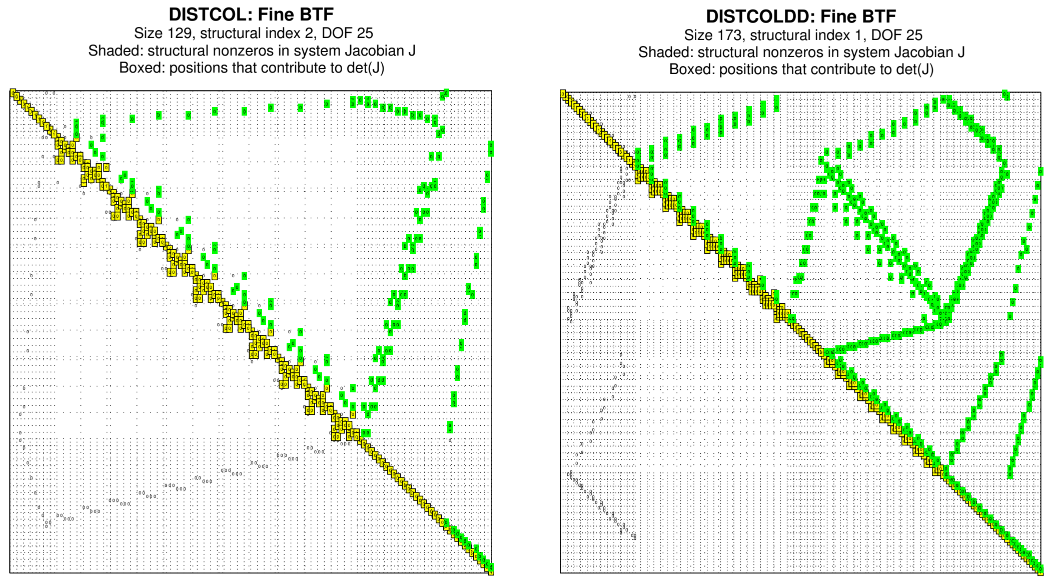
(b) DISTCOLDD

Fig. 4. Distillation model: structure of index-2 and index-1 formulations. Offsets, and equations and variable indices are not displayed for problems of size $n > 40$.



(a) DISTCOL (b) DISTCOLDD

Fig. 5. Distillation model: coarse BTFs



(a) DISTCOL (b) DISTCOLDD

Fig. 6. Distillation model: fine BTFs

3.2.1. Coarse BTF. In Figure 5(a) and (b), we show the coarse BTFs of DISTCOL and DISTCOLDD, respectively. Consider DISTCOL. With the `getBTF` function, we determine that there are 14 blocks of size 1 and one block of size 115. Using `getQLdata`, we determine that blocks 1:1 and 3:117 are non-quasilinear. The rest are quasilinear. For

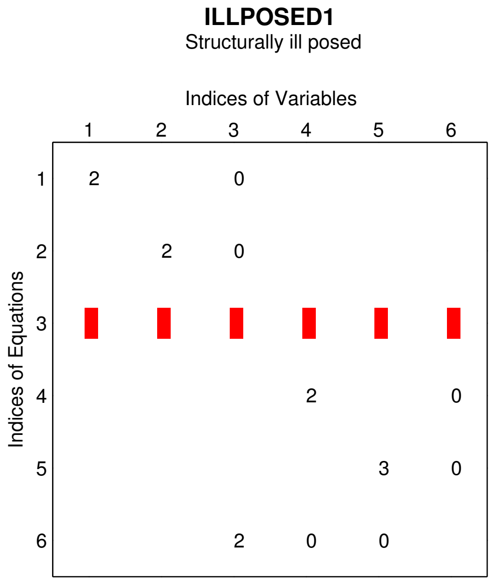
DISTCOLDD, we have 14 blocks of size 1 and one block of size 159. Blocks 2:2 and 3:161 are non-quasilinear; the rest are quasilinear.

3.2.2. Fine BTF. Figure 6(a) and (b) show the fine BTFs of DISTCOL and DISTCOLDD, respectively. DISTCOL has 52 blocks of size 1, of which 13 are non-quasilinear, and 11 quasilinear blocks of size 7. Obviously, the 115×115 block (from the coarse BTF) has decomposed into smaller blocks. For DISTCOLDD, we have 85 blocks of size 1, of which 13 are non-quasilinear, 11 non-quasilinear blocks of size 3, and 11 quasilinear blocks of size 5. Either way, the BTF shows great potential for speeding up numerical solution.

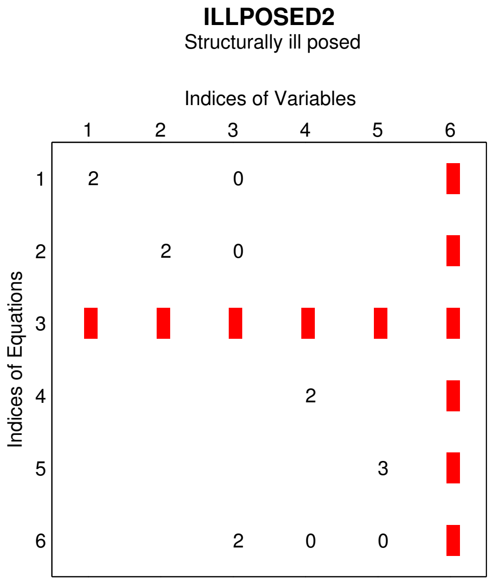
3.3. Troubleshooting

DAESA can report missing equations and/or variables in problem formulation, and can produce a diagnostic BTF, indicating over- and under-determined parts of the system. This is illustrated here.

3.3.1. Reporting missing equations and/or variables. In the MOD2PEND problem from the companion paper, we remove the third equation, apply `daeSA`, and then call `showStruct`; see Figure 7(a). Then we remove the occurrences of variable μ and apply these two functions; see Figure 7(b). The function `showStruct` displays missing equations and/or variables in red.



(a) $f_3 = 0$ removed from MOD2PEND



(b) $f_3 = 0$ and μ removed from MOD2PEND

Fig. 7. Displaying the structure of two ill-posed problems

For large problems, one may not be able to easily find missing equation and/or variable indices from the displayed structure. The function `getMissing` produces indices of equations and variables that are missing in the problem. For example, calling

```
[meqn, mvar] = getMissing(sadata)
```

```
function f = illPosed3(t, z)
  x1 = z(1); x2 = z(2); x3 = z(3);
  x4 = z(4); x5 = z(5); x6 = z(6);
  f(1) = x1   + x2 +           x5   + x6;
  f(2) = x1^3 + x2 +           x5   + x6  ;
  f(3) =                       x5   * x6  ;
  f(4) =                     - x5^3 + x6^4;
  f(5) = x1   + x2 + x3 + x4 + x5   + x6^3;
  f(6) =                       x5   + x6  ;
end
```

Fig. 8. Example of a SIP problem

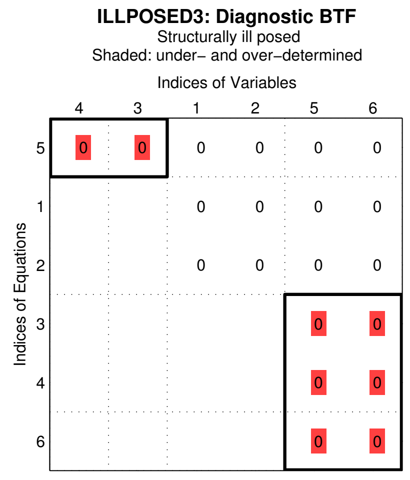
on the second DAE (with missing equation 3 and variable 6) gives `meqn=3`, `mvar=6`.

3.3.2. Diagnosing SIP problems. When a DAE does not have missing equations or variables but is nevertheless SIP, we can use `getBTF` to identify under-, over-, and well-determined subsystems. As an example, the function in Figure 8 gives a SIP problem. Executing the commands

```
sadata3 = daeSA(@illPosed3,6);
[pe,pv,cb,fb] = getBTF(sadata3)
showStruct(sadata3,'disptype','blocks');
```

produces the output in Figure 9. The output vectors `pe` and `pv` give the permutation of equations and variables, respectively; `cb` is empty; and `fb` specifies the boundaries of the under-, well- and over-determined blocks. Here, we have a structurally underdetermined equation $0 = f_5$ in variables x_3 and x_4, and a structurally over-determined set of equations $0 = f_3, f_4, f_6$ in variables x_5 and x_6.

```
pe =
     5     1     2     3     4     6
pv =
     4     3     1     2     5     6
cb =
     []
fb =
     1     2     4     6     7
     1     2     3     5     7
```



$$\begin{array}{c|cccccc} & 4 & 3 & 1 & 2 & 5 & 6 \\ \hline 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & & & 0 & 0 & 0 & 0 \\ 2 & & & 0 & 0 & 0 & 0 \\ 3 & & & & & 0 & 0 \\ 4 & & & & & 0 & 0 \\ 6 & & & & & 0 & 0 \end{array}$$

Fig. 9. Output and diagnostic BTF of the SIP problem from Figure 8

.

4. PERFORMANCE: PROBLEMS OF ADJUSTABLE SIZE

In this section we study the performance of the main function, `daeSA`. We apply it on a fully dense problem and five sparse problems. We describe these problems in §4.1 and then present timing and profiling data in §4.2 and §4.3, respectively.

4.1. Problems used

4.1.1. Dense DAE. The Layne Watson exponential cosine curve example [Watson 1979] is a standard problem for assessing continuation methods: we want to find a fixed point of the nonlinear function $\mathrm{g} : \mathbb{R}^n \to \mathbb{R}^n$ defined by

$$g_i(\mathbf{x}) = g_i(x_1, \dots, x_n) = \exp\left(\cos\left(i \cdot \sum_{k=1}^{n} x_k\right)\right), \quad i = 1, \dots, n;$$

that is a root of $\mathbf{x} = \mathbf{g}(\mathbf{x})$.

Using DAETS [Nedialkov and Pryce 2009], we sought a fixed point of the parameterized problem $\mathbf{x} = \lambda\mathbf{g}(\mathbf{x})$, that is a root of

$$\mathbf{f}(\lambda, \mathbf{x}) = \mathbf{x} - \lambda\mathbf{g}(\mathbf{x}) = \mathbf{0}. \tag{2}$$

When $\lambda = 0$, equation (2) has the trivial solution $\mathbf{x} = \mathbf{0}$. We tracked solutions to the desired root at $\lambda = 1$.

This was implemented in DAETS using *arc-length continuation*. Namely, we rewrote the system (2) as n equations in $n{+}1$ unknowns:

$$\mathbf{f}(\mathbf{y}) = \mathbf{0}, \tag{3}$$

where $\mathbf{y} = (\mathbf{x}; \lambda)$, and using a new independent variable s, we added to (3) the equation $\|\mathrm{d}y/\mathrm{d}s\|_2^2 = 1$, that is

$$\sum_j {y_j'}^2 - 1 = 0. \tag{4}$$

Combining (3) and (4), we have an index-1, fully dense, non-quasilinear system of $n+1$ equations and variables. We refer to the resulting systems as LW.

There is no nontrivial block structure, either coarse or fine, as shown in Figure 10(a), where the dimension is 12.

4.1.2. Sparse DAEs. The problems below consist of P chained pendula, where each pendulum is of size 3. The resulting systems are of size $n = 3P$, and each pendulum is quasilinear. The aim is to produce parameterized problems of similar sparsity patterns but differing significantly in index and block structure.

MULTIPEND. The first and the ith pendula ($2 \le i \le P$) are

first pendulum	ith pendulum
$0 = x_1'' + \lambda_1 x_1$	$0 = x_i'' + \lambda_i x_i$
$0 = y_1'' + \lambda_1 y_1 - G$	$0 = y_i'' + \lambda_i y_i - G$
$0 = x_1^2 + y_1^2 - L^2$	$0 = x_i^2 + y_i^2 - (L + c\lambda_{i-1})^2.$

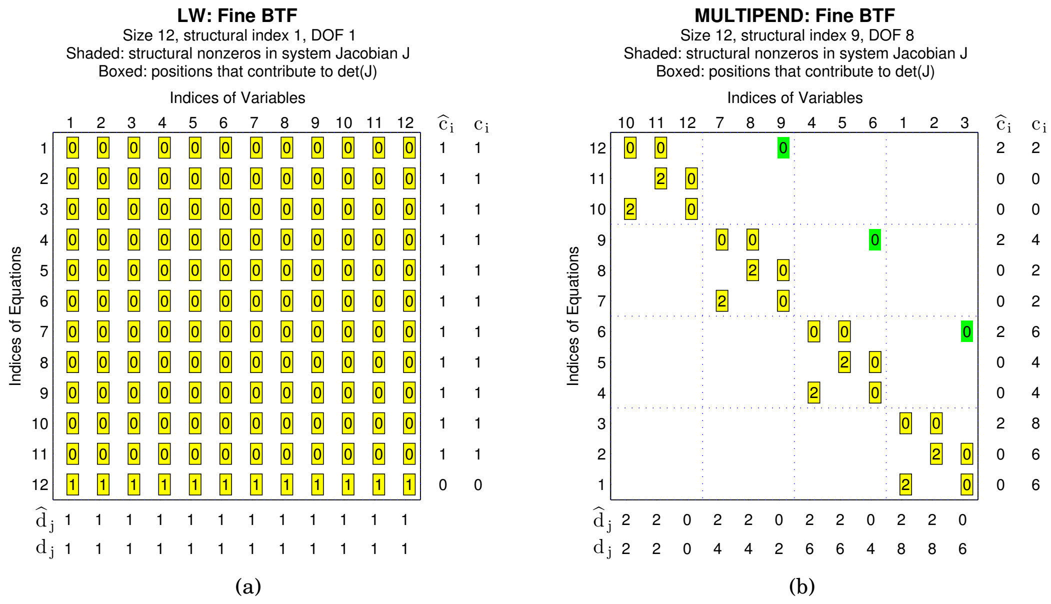

LW: Fine BTF
Size 12, structural index 1, DOF 1
Shaded: structural nonzeros in system Jacobian J
Boxed: positions that contribute to det(J)
Indices of Variables
Indices of Equations
(a)
MULTIPEND: Fine BTF
Size 12, structural index 9, DOF 8
Shaded: structural nonzeros in system Jacobian J
Boxed: positions that contribute to det(J)
Indices of Variables
Indices of Equations
(b)

Fig. 10. LW and MULTIPEND: fine BTFs

The state variables of the ith pendulum ($i \geq 1$) are x_i, y_i, and λ_i; $G > 0$ is gravity, $L > 0$ is the length of the first pendulum, and c is a constant. A system of P pendula has index $2n/3 + 1 = 2P + 1$.

The coarse and fine BTFs are the same; we display in Figure 10(b) the fine BTF, where $P = 4$.

MULTIPENDA. This problem is obtained by replacing the third equation in each pendulum, except in the first one, by

$$0 = x_i^2 + y_i^2 - (L + cx'_{i-1})^2.$$

The index is 3, for any P. There are P coarse blocks of size 3, and each such block except the top-left one is decomposed into 3 fine blocks of size 1.

The coarse and fine BTFs for $P = 4$ are displayed in Figure 11(a), 11(b).

MULTIPENDB. The pendula are

first pendulum	ith pendulum, $2 \leq i \leq P-1$	Pth pendulum
$0 = x_1'' + \lambda_1 x_1 + cx_2$	$0 = x_i'' + \lambda_i x_i + cx_{i+1}$	$0 = x_P'' + \lambda_P x_P$
$0 = y_1'' + \lambda_1 y_1 - G$	$0 = y_i'' + \lambda_i y_i - G$	$0 = y_P'' + \lambda_P y_P - G$
$0 = x_1^2 + y_1^2 - L^2$	$0 = x_i^2 + y_i^2 - (L + cx'_{i-1})^2$	$0 = x_P^2 + y_P^2 - (L + cx'_{P-1})^2$

The index is $P+2$, for any P. Here there is only one coarse block, which decomposes into P fine blocks of size 3.

The coarse and fine BTFs for $P = 4$ are displayed in Figure 12(a), 12(b).

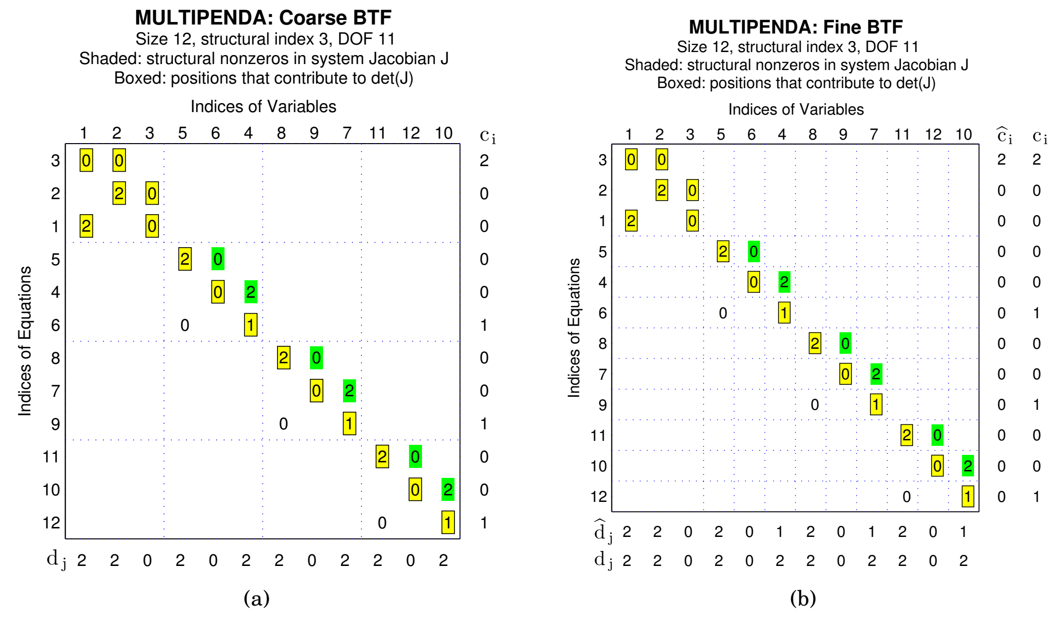

MULTIPENDA: Coarse BTF
Size 12, structural index 3, DOF 11
Shaded: structural nonzeros in system Jacobian J
Boxed: positions that contribute to det(J)
Indices of Variables
Indices of Equations
MULTIPENDA: Fine BTF
(a)
(b)

Fig. 11. MULTIPENDA: coarse and fine BTFs

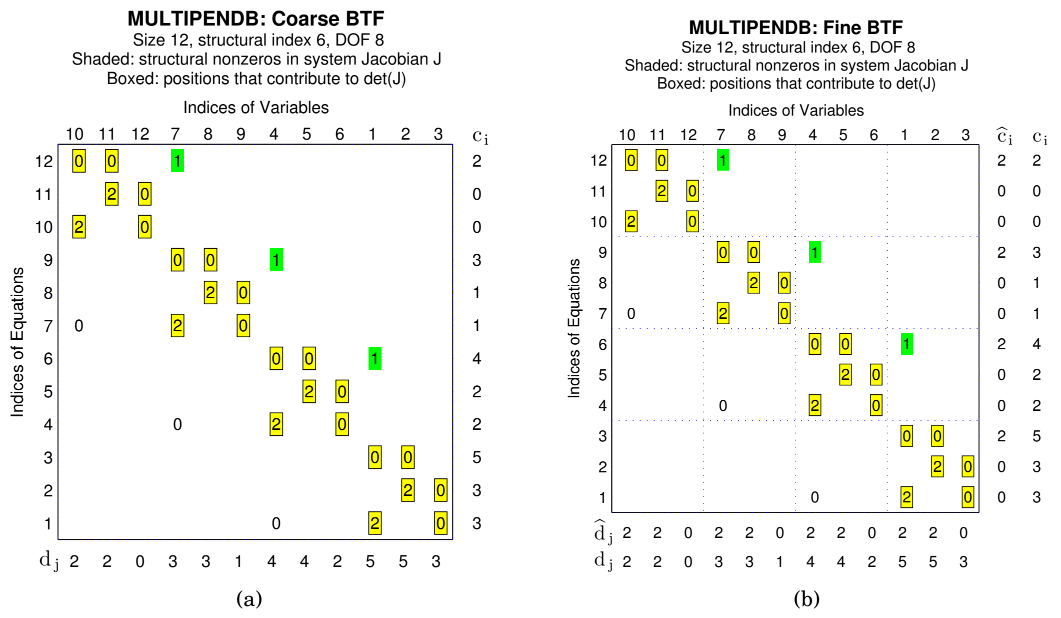

MULTIPENDB: Coarse BTF
Size 12, structural index 6, DOF 8
Shaded: structural nonzeros in system Jacobian J
Boxed: positions that contribute to det(J)
Indices of Variables
Indices of Equations
MULTIPENDB: Fine BTF
(a)
(b)

Fig. 12. MULTIPENDB: coarse and fine BTFs

MULTIPENDC. We replace the first equation in each pendulum $i < P$ in MULTIPENDA by

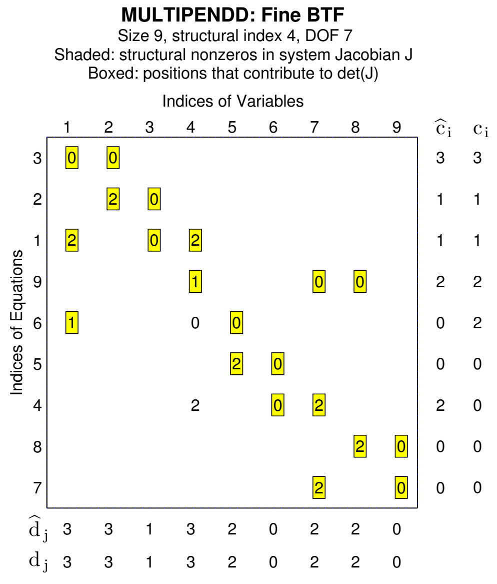
$$0 = x_i'' + \lambda_i x_i + c x_{i+1}'.$$

The index is $P+2$, for any P. There is one coarse block, which does not decompose into fine blocks (figure omitted).

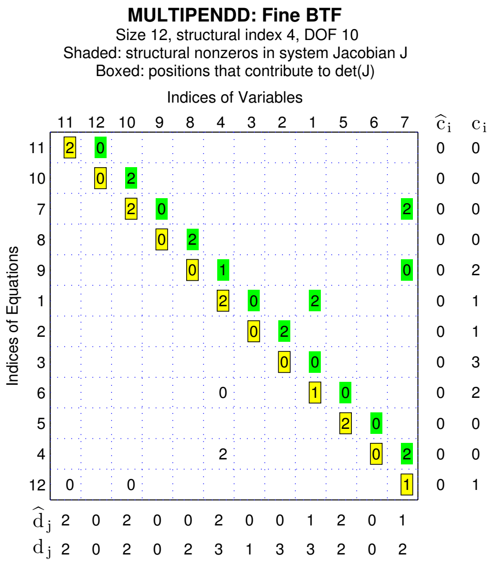
MULTIPENDD. Now we replace the first equation in each pendulum $i < P$ in MULTIPENDA by

$$0 = x_i'' + \lambda_i x_i + c x_{i+1}''.$$

The index is $P/2+(9-(-1)^P)/4$. For any P, there is one coarse block. However, if P is odd, there is one fine block, and if P is even, there are n fine blocks. In Figure 13(a), 13(b), we show the fine BTFs for $P=3$ and $P=4$ respectively.


MULTIPENDD: Fine BTF
Size 9, structural index 4, DOF 7
Shaded: structural nonzeros in system Jacobian J
Boxed: positions that contribute to det(J)
Indices of Variables
Indices of Equations

(a)


MULTIPENDD: Fine BTF
Size 12, structural index 4, DOF 10
Shaded: structural nonzeros in system Jacobian J
Boxed: positions that contribute to det(J)
Indices of Variables
Indices of Equations

(b)

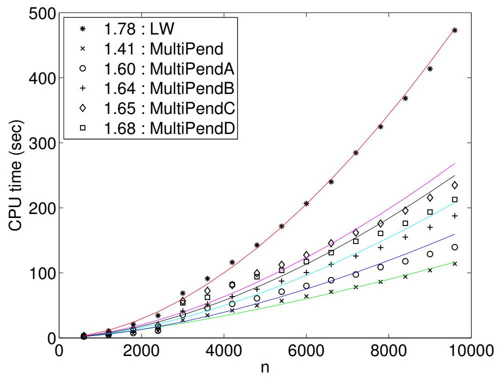
Fig. 13. MULTIPENDD: fine BTFs

4.2. CPU time vs. size

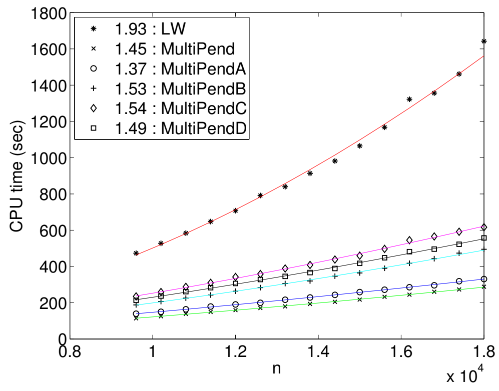
The results in this subsection, and later in §5.3.3, are produced with MATLAB R2013a on a Linux server with 16 CPUs and 64GB of RAM. The CPUs are Intel Xeon E7-8870 2.4GHZ. The implementation of DAESA does not take advantage of the parallel capabilities of MATLAB; that is, it is a serial implementation.

Figure 14, plots CPU time versus n for $n = 600 : 600 : 9600$, 14(a), and for $n = 9600 : 600 : 18000$, 14(b). Denoting by t_i the CPU time corresponding to size n_i, we find the constants in αn^β using a least-squares fit on $\log\alpha + \beta\log n_i = \log t_i$ and plot these fits.

For dense problems, we might expect $O(n^3)$ work due to the complexity for finding a HVT; here it is more like $O(n^2)$. For the sparse pendulum problems, the CPU time behaves more like $O(n^{\approx 1.5})$.



(a) $n = 600 : 600 : 9600$



(b) $n = 9600 : 600 : 18000$

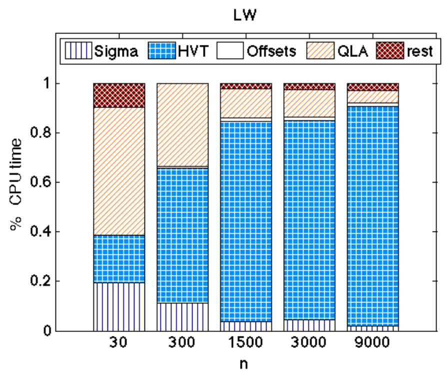
Fig. 14. CPU time versus n. The constants in the legends are the computed β's in αn^{β}.

4.3. Work breakdown

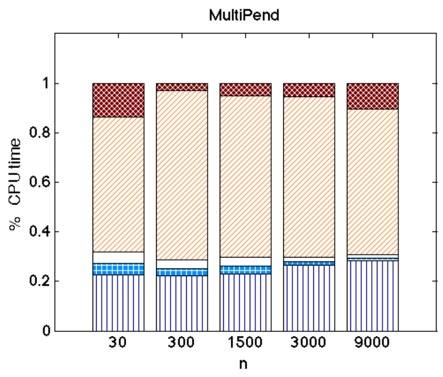
We profile the execution of daeSA on the above problems, using $P = 10, 100, 500, 1000$ and 3000 pendula, with problem sizes $n = 3P$. We use the same n's for the LW problem.

In Figure 15, we display the percentages of time for computing Σ, HVT, offsets, and performing quasilinearity analysis (QLA) for the LW and MULTIPEND problems. For the remaining problems, the plots are very similar to Figure 15(b), and we omit them here.

For the LW problem, as n increases, the computing time is dominated by the time for finding a HVT, Figure 15(a). For the remaining problems, most the time is in computing Σ and finding QLA information, the latter being the dominating time, Figure 15(b).



(a)



(b)

Fig. 15. Profiling daeSA. The percentages of the total time are denoted as: Sigma for computing the Σ matrix; HVT for finding HVT(s); Offsets for computing canonical offsets c and d; QLA for performing quasilinearity analysis; Rest the rest of daeSA.

5. PERFORMANCE: PROBLEMS GENERATED FROM FMC MATRICES

In this section, we report results from a study using DAESA on DAEs generated from matrices from the Florida Sparse Matrix Collection [Davis and Hu 2011]. We describe how we generate such DAEs in §5.1. In §5.2, we study sample of around 70 to 100 such DAEs for each of three matrices. In §5.3, we experiment over a large number of matrices, and respectively DAEs.

5.1. A way to generate DAEs with a given sparsity

To test the performance of DAESA, we generated a large number of "pseudo-random" DAEs by a simple algorithm. It takes just two inputs: a matrix A and a non-negative integer "seed vector" $s = (s_1, \dots, s_r)$, and outputs (as MATLAB code accepted by DAESA) a linear constant-coefficient DAE whose signature matrix Σ has the same sparsity pattern as does A. The N nonzero entries of A are listed column-by-column as N triples (i, j, a_{ij}), as done by the MATLAB statement `[i,j,aij] = find(A)`. Then s is repeated cyclically to form a vector of length N which replaces the vector of a_{ij}'s to form the entries (i, j, σ_{ij}) of Σ (the remaining entries being $-\infty$).

The entries of A were used in the actual DAE constructed by this scheme, the DAE function f_i is the sum $\sum_{j:a_{ij}\neq 0} a_{ij} x_j^{(\sigma_{ij})}$, plus a constant b_i. In these experiments, we only used vectors s with entries 0 or 1. The DAE coefficients $a_{ij} \neq 0$ and b_i are irrelevant to the structure of the DAE—which only depends on A's sparsity pattern and the seed vector—but were included to make the DAE more interesting should we experiment with numerical solution in future.

For example,

$$A = \begin{pmatrix} & 1.2 & & -3.4 \\ 5.6 & & & 7.8 \\ & & -2.3 & 4.5 \\ 6.7 & & & \end{pmatrix} \text{ and } s = (0, 1, 2) \text{ generate } \Sigma = \begin{pmatrix} & 2 & & 1 \\ 0 & & & 2 \\ & & 0 & 0 \\ 1 & & & \end{pmatrix},$$

where s has been replicated to form $(0, 1, 2, 0, 1, 2, 0)$ whose 7 entries replace the 7 nonzeros of A, columnwise, to form Σ. The actual DAE, with variables $x_1, \dots, x_4$, is

$$\left.\begin{array}{rcrrrrr} 0 &=& f_1 = & & 1.2x_2'' & & -\ 3.4x_4' + 1 \\ 0 &=& f_2 = & 5.6x_1 & & & +\ 7.8x_4'' + 1 \\ 0 &=& f_3 = & & & -\ 2.3x_3 & +\ 4.5x_4 + 1 \\ 0 &=& f_4 = & 6.7x_1' & & & + 1 \end{array}\right\},$$

and the MATLAB code describing f_1 might be

```
f(1) = 1.2*Dif(x(2),2) - 3.4*Dif(x(4),1) + 1;
```

In general, this method generates from A a DAE of the form

$$0 = A_0 x(t) + A_1 x'(t) + \cdots + A_p x^{(p)}(t) + b,$$

where p is the largest number occurring in s, A is decomposed as $A = A_0 + A_1 + \cdots + A_p$, and the A_r have disjoint sparsity patterns. Namely, A_r has a nonzero entry equal to a_{ij} for those (i, j) such that $\sigma_{ij} = r$, and is zero elsewhere.

This testing scheme, like others based on generating artificial problems, faces the criticism that the results may not represent real-life problems. By choosing real-life matrices A from the Florida Matrix Collection, we hope to lessen the force of this criticism, though the artificiality of the seed vector process remains. We call a DAE generated in this way "pseudo-random" in that generally we do not expect a particular pattern in Σ, though sometimes we have observed a "resonance" of s with the structure of A, giving a high index, or a large change in the DOF, compared with other similar s.

5.2. Study of DAEs derived from some FMC matrices

In this group of tests a few FMC matrices A were chosen, and for each one a sample of DAEs having the sparsity structure of A was generated as described above. Each DAE was analyzed by DAESA, and we recorded: the CPU time taken by `daeSA` to analyze the DAE; the index; the DOF; the sequence of sizes of coarse blocks; the same for fine blocks.

As a crude numerical measure of how effective block decomposition had been, we recorded `cbratio` $=$ (matrix size)$/$(maximum coarse block size) and `fbratio` similarly for fine blocks. (The coarse structure depends only on the matrix A; the fine structure usually depends on the DAE, i.e., the seed vector.) We aimed to answer questions such as the following:

(a) For a particular A, how do coarse block and fine block structure, and DOF and index, vary across the sample?

(b) Can we relate answers to (a) to the underlying sparsity pattern (that of A)?

(c) Are any of the metrics significantly correlated?

(d) Do the answers to these questions vary significantly with A of different structure: e.g., banded sparsity versus apparently unstructured sparsity?

Three FMC matrices were chosen. Each DAE m-file was given a name made up of the FMC ID number and the s-vector—for instance `DAE_330_00000001.m` was generated from FMC330, that is matrix #330, using seed vector $s = (0,0,0,0,0,0,0,1)$. It was hard to see general patterns, so the report is mainly anecdotal.

FMC1. (a power network problem, name `1138_bus` in the `HB` group, of size 1138 with 4054 nonzeros) is symmetric with a concentration of entries near the diagonal and a somewhat uneven scattering elsewhere. We formed 76 DAEs from it with various s-vectors of lengths from 6 to 13, all having just two 1's. There was no coarse block structure (i.e., `cbratio` $= 1$). All cases had a definite fine block structure in the sense that all blocks were much smaller than the matrix size: the lowest `fbratio` was 3.9, while the highest, for `DAE_1_00000011`, was around 60, with largest fine block of size 19. However, no pattern was observed in the fine blocks; in each case the blocks of size 1 far outnumbered the larger blocks, but the latter seemed to be distributed randomly among the former. This contrasts with FMC217, where the fine block sizes repeat in a fairly regular way.

The index varied from 3 to 6 and the DOF from 345 to 687. There were weak, but statistically significant correlations of DOF negatively with CPU time, and positively with `fbratio`.

FMC217. (a structural problem, name `nos1` in the `HB` group, of size 237 with 1017 nonzeros) is symmetrical and banded, with bandwidth 4 either side of the diagonal. We formed 85 DAEs from it with s-vectors of length 4 to 13. The band conceals a coarse block structure: a block of size 79, then one of size 158.

For most cases, the fine block structure consisted of many small blocks, with `fbratio` as high as 79 several times, indicating the fine blocks have size at most 3. E.g., `DAE_217_001001100` has 164 blocks of size 1, 11 of size 2 and 17 of size 3.

Various s of length 13 showed an index-raising "resonance": for instance `DAE_217_0101001010000` has index 39, while cyclically shifting its s one to the left gives `DAE_217_1010010100000`, which only has index 1. These high-index cases had a fine block of size around 155, the rest being of size 1 or 2.

Both the lowest and highest DOF values (DOF 119, index 39; DOF 196, index 21) belonged to these resonance cases; for the rest, the index varied from 1 to 4 with no strong correlation to the DOF.

FMC330. (a robotics problem, name `rbsb480` in the `Bai` group, of size 480 with 17088 nonzeros) has a sparsity pattern made up of five similar horizontal bands but has no other clear structure. We formed 96 DAEs from it, using s-vectors having either one or two 1's followed by several zeros, e.g. $(1, 1, 0, 0, 0, 0, 0)$; and all cyclic permutations of these, e.g. $(0, 0, 0, 1, 1, 0, 0)$.

The index was either 1 or 2, except for `DAE_330_00000001` and one other which had index 3. The DOF varied from 371 to 477, tending to be smaller for s-vectors with a smaller proportion of 1's. There was no coarse block structure. Mostly there was little fine block structure (`fbratio` close to 1), but `DAE_330_00000001` had a less trivial decomposition with a largest block of size 293 giving `fbratio` $= 1.64$. The largest fine block was almost always at or near the top left, but `DAE_330_00010000`'s largest block, of size 324, was a good way down the diagonal.

There were weak but statistically significant correlations of CPU time positively with index, and negatively with DOF and with `fbratio`.

From these results—especially those for FMC1 and FMC330, where there is no reason to expect resonance between s and the sparsity—it seems probable that, on the one hand, many real-life DAEs with a nontrivial sparsity pattern will also have a nontrivial fine-block decomposition that can be exploited to speed up numerical solution; while on the other hand, for other DAEs such fine-block structure may be weak or absent. Resonance leading to high index may occur: physically, this probably relates to coupling between components of a repetitive or otherwise strongly structured system.

5.3. Further problems derived from FMC

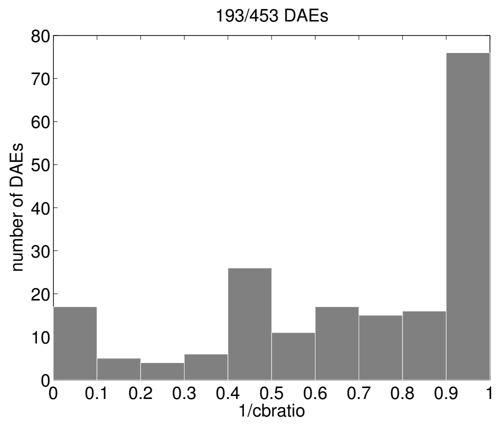
Using `UFget` [Davis and Hu 2011], we create a list of matrices of size $n \le 10,000$. If there is more than one matrix of the same size, and/or more than one matrix with the same number of nonzeros, we select the first one. This results in total of 535 matrices. Then we use the seed vectors

$$[1\ 1\ 0\ 0],\ [0\ 1\ 0\ 0\ 0\ 1],\ \text{and}\ [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

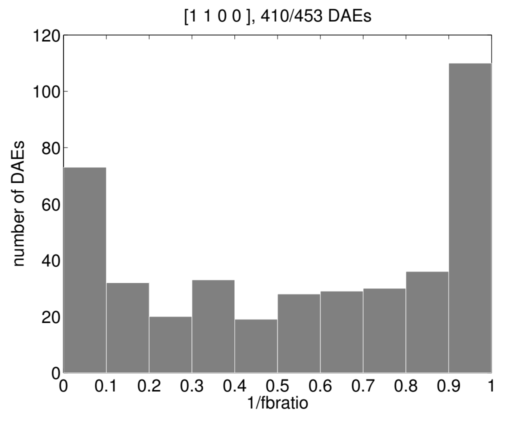
to generate three sets of DAEs (each consisting of 535 DAEs), which we analyze with `daeSA`. 82 of them are SIP, so finally we have three sets of 453 DAEs. The smallest and largest in terms of n and number of nonzeros are:

n		nnz	id	name	kind
min = 5		19	904	cage3	directed weighted graph
max =	10,000	40,000	532	G67	undirected weighted random graph
nnz		n			
min = 15		7	449	b1_ss	chemical process simulation problem
max =	2,269,500	6001	1235	exdata_1	optimization problem

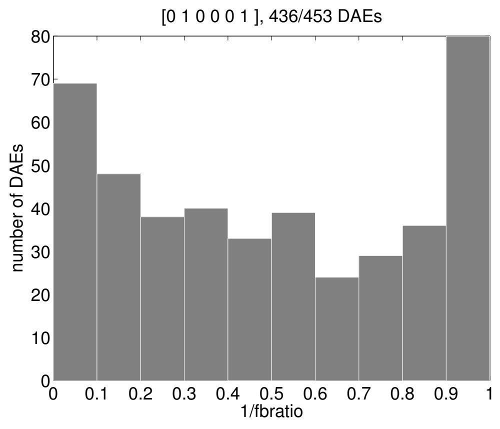
5.3.1. Block sizes. In Figure 16(a), we show a histogram of $1/$`cbratio` for the problems for which this ratio is < 1; that is, for problems that have at least two coarse blocks. In Figure 16(b),16(c), 16(d), we show histograms of $1/$`fbratio` for DAEs obtained using each of the seed vectors above, and for which this ratio is < 1.



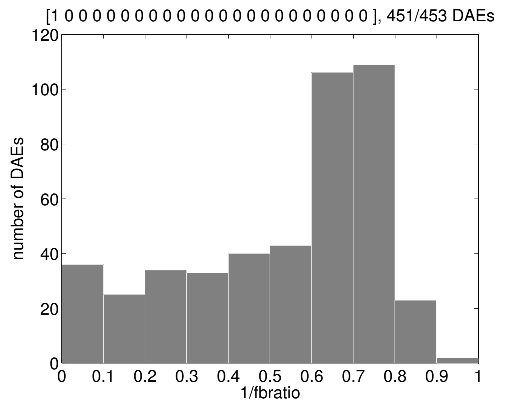
(a)



(b)



(c)



(d)

Fig. 16. `1/cbratio` is (maximum coarse block size)/(matrix size); `1/fbratio` is (maximum fine block size)/(matrix size). In the titles, M/N means in M of the N problems, the ratios are < 1.

Although it is difficult to find definite patterns, it seems that there is a large number of structures, where we can exploit the fine BTFs for efficient numerical solution. E.g., with seed vector $(1, 1, 0, 0)$, over 70 matrices have $1/$`fbratio` < 0.1, meaning no fine block is larger than a tenth of the matrix size.

5.3.2. Fine blocks and index. In Figure 17, we display boxplots of `1/fbratio` versus index. For indices with a non-trivial number of samples (e.g. more than 10 DAEs per index), generally the DAEs seem to have finer block structure as the index increases.

Remark 5.1. These boxplots are produced with MATLAB's `boxplot` command; see Figure 18 for an interpretation.

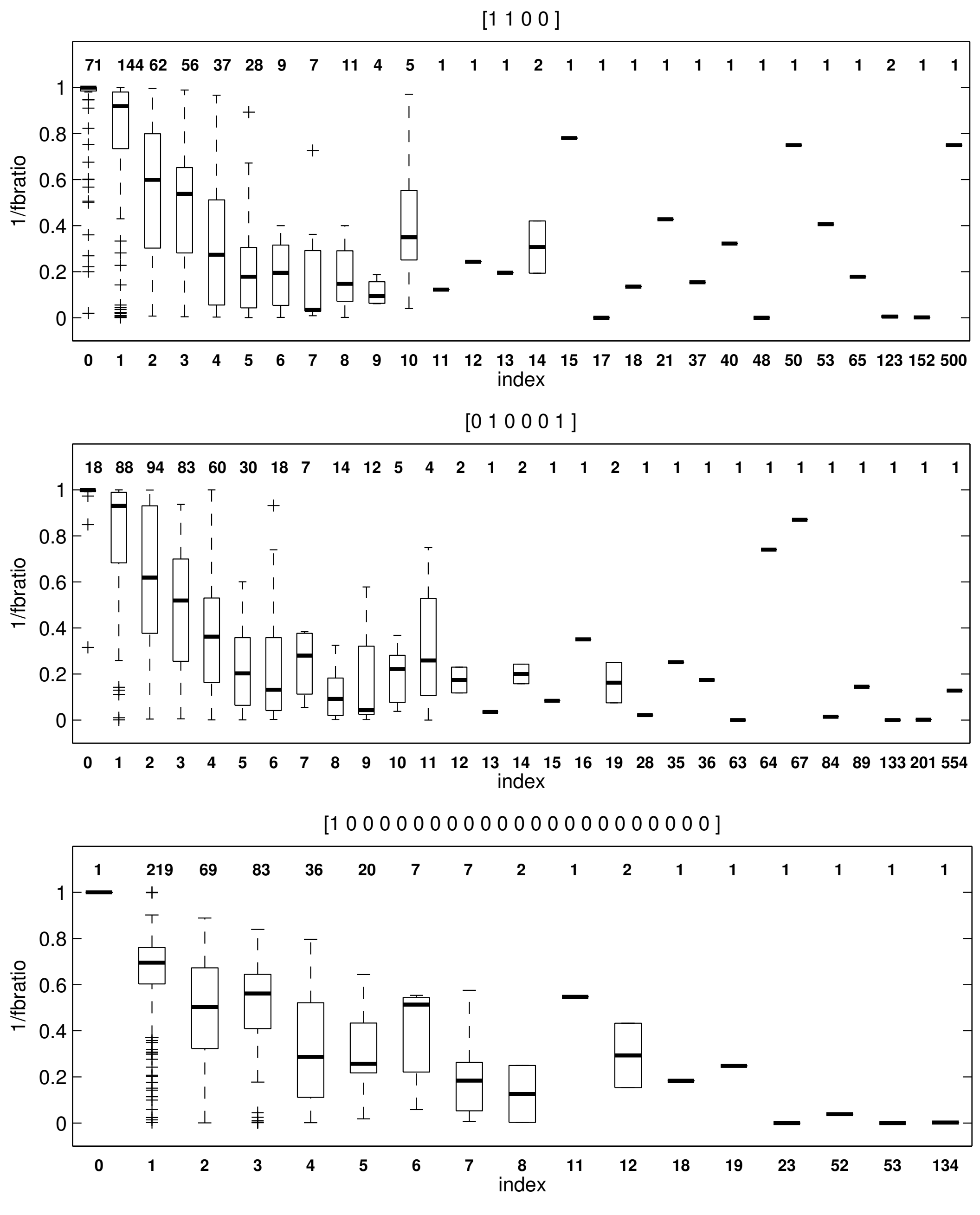
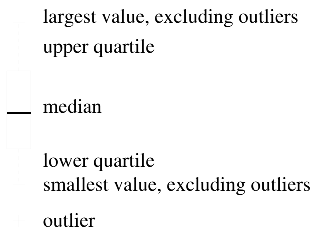


Fig. 17. Boxplots of `1/fbratio` versus index. The number above a box, corresponding to an index, is the number of DAEs of this index.


largest value, excluding outliers
upper quartile
median
lower quartile
smallest value, excluding outliers
outlier

50% of the values (here number of DAEs) are above the median (thick line). The top edge of the box divides the top 25% percent (upper quartile) from the rest; the bottom edge of the box divides the bottom 25% (lower quartile) from the rest. The difference between the top and bottom quartiles is the interquartile range *(IQR). Outliers are marked at* $1.5 \times$ *IQR.*

Fig. 18. Boxplot interpretation

5.3.3. CPU time. We record the CPU time for each execution of `daeSA` and plot it versus n and number of nonzeros (nnz) in Figure 19. A rough estimate of the complexity (using least-squares fits) shows about $O(n^{1.52})$ and $O(\mathrm{nnz}^{1.19})$ running times; cf. §4.2.

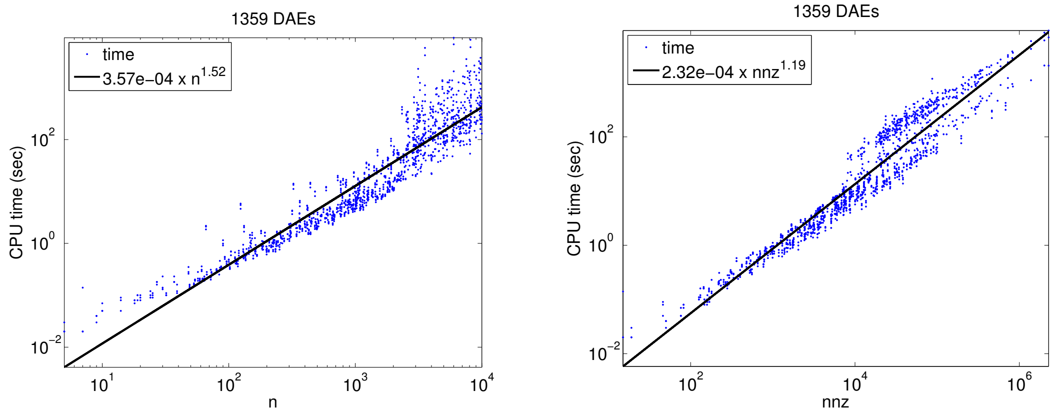


Fig. 19. CPU time vs. n and vs. nnz

6. CONCLUSION AND FUTURE WORK

The PhD thesis of P. Bunus [2004] clearly stated the goal of building software to analyze DAEs generated by modeling languages, to diagnose errors in model construction and to make numerical solution more efficient. He attacked it using methods based on sparsity, but the relation of sparsity to DAE block structure was then insufficiently understood.

Much progress has been made since then in such structural analysis software; we believe the DAESA package currently goes furthest towards Bunus's goal. It is at present unique in being based on a systematic theory for DAEs that can be fully implicit and of arbitrary order and index. Its features:

— report structural index and DOF;

— report solution scheme plus information about initial values and constraints;

— display structure and BTF graphically;

are present in other systems, but not in so far-reaching a way. We believe no other system exploits fine BTF to minimize the number of initial values required for numerical solution.

DAESA's abilities to diagnose ill-posedness, and report under- and over-determined parts of a system, potentially go beyond what is currently available in terms of suggesting ways of correcting them. However, this needs considerable further development.

As for performance, tests on on a large number of problems of various sizes and sparsities show DAESA is well able to analyze DAEs of size up to 10,000. Though the current version exploits sparsity only partially, the expected n^3 dependence on problem size has not materialized. The tests reported here give an indication where to concentrate effort to improve the algorithms. We are developing faster methods to compute the offsets, while future work will focus on reducing the time for QLA and computing the Σ matrix; cf. Figure 15(b).

Further developments of this tool include implementation of the computation of the system Jacobian, which will provide a check for the success of the SA. That is, if this Jacobian is nonsingular (up to roundoff), the SA has succeeded. We have a prototype implementation, which will be incorporated in a future version of DAESA.

A complete DAE solver (with the capabilities of DAETS) in MATLAB would enable researchers and practitioners to use it readily within existing MATLAB code and to experiment within this problem-solving environment. Building such a solver requires a tool for computing Taylor coefficients (TCs). A suitable candidate is the ADTAYL [Pryce et al. 2010] package. It is designed to compute TCs for an explicit function, given TCs of its argument(s), but it is inefficient for computing TCs for the solution to an ODE, let alone a DAE. We are working on extending it to compute such coefficients. Finding consistent initial point, stepsize control, and projections on each integration step can be handled as in DAETS.

An exciting direction for research and implementation is parallelizing the algorithms for DAE solving based on Pryce's SA. The DAETS solver can handle efficiently a few hundred equations, but would be unacceptably slow for thousands of equations. For large and sparse problems, one could take advantage of the BTFs in two ways. First, we could use the coarse BTF to integrate the DAE system in a block-wise manner, similarly to how block-triangular linear systems are solved. Then we could pipeline the integration of these subproblems, where the integrations advance asynchronously in time (companion paper). Second, we could use the fine BTF to generate TCs for smaller subproblems, versus computing TCs for the whole system. As a result, we need to solve smaller linear systems, while in the original method (and DAETS), we solve $n \times n$ systems [Nedialkov and Pryce 2005; Nedialkov and Pryce 2008].

The studies in §5 reveal that we can have fine BTFs with the size of the largest fine block much smaller than the size of the problem: this would lead to efficient, block-wise numerical integration.

In its present form, DAESA is a collection of MATLAB functions, so that its facilities are invoked at the MATLAB command line or within code. We are developing a GUI interface to DAESA that will allow interactive study of DAEs. This will allow zooming into the structure of large matrices and identifying lines of code that lead to a given pattern in the signature or system Jacobian matrix.

ACKNOWLEDGMENTS

The authors are grateful to Ian Washington from the Department of Chemical Engineering of McMaster University for numerous discussions and suggestions on improving DAESA, providing DAE problems and encoding them in DAESA, and in particular the model in §3.2.

We acknowledge with thanks the support given to JDP by the Leverhulme Trust and the Engineering and Physical Sciences Research Council, both of the UK, and to GT and NSN by the Canadian Natural Sciences and Engineering Research Council and the McMaster Centre for Software Certification.

REFERENCES

BRENAN, K., CAMPBELL, S., AND PETZOLD, L. 1996. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations* second Ed. SIAM, Philadelphia.

BUNUS, P. 2004. Debugging techniques for equation-based languages. Ph.D. thesis, Linköping University, Sweden, Department of Computer and Information Science.

DAVIS, T. A. AND HU, Y. 2011. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw. 38,* 1, 1:1–1:25.

HINDMARSH, A. C., BROWN, P. N., GRANT, K. E., LEE, S. L., SERBAN, R., SHUMAKER, D. E., AND WOODWARD, C. S. 2005. SUNDIALS, Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Trans. Math. Softw. 31,* 3, 363–396.

MATTSSON, S. E. AND SÖDERLIND, G. 1993. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM J. Sci. Comput. 14,* 3, 677–692.

MAZZIA, F. AND IAVERNARO, F. 2003. Test set for initial value problem solvers. Tech. Rep. 40, Department of Mathematics, University of Bari, Italy. `http://pitagora.dm.uniba.it/~testset/`.

MCKENZIE, R., NEDIALKOV, N. S., PRYCE, J., AND TAN, G. 2013. DAESA user guide. Tech. rep., Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, L8S 4K1. `http://www.cas.mcmaster.ca/~nedialk/daesa/daesaUserGuide.pdf`.

NEDIALKOV, N. AND PRYCE, J. 2008–2009. DAETS user guide. Tech. rep., Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, L8S 4K1.

NEDIALKOV, N. S. AND PRYCE, J. D. 2005. Solving differential-algebraic equations by Taylor series (I): Computing Taylor coefficients. *BIT 45*, 561–591.

NEDIALKOV, N. S. AND PRYCE, J. D. 2007. Solving differential-algebraic equations by Taylor series (II): Computing the System Jacobian. *BIT 47,* 1, 121–135.

NEDIALKOV, N. S. AND PRYCE, J. D. 2008. Solving differential-algebraic equations by Taylor series (III): The DAETS code. *JNAIAM 3,* 1–2, 61–80. ISSN 17908140.

PRYCE, J., GHAZIANI, R. K., WITTE, V. D., AND GOVAERTS, W. 2010. Computation of normal form coefficients of cycle bifurcations of maps by algorithmic differentiation. *Mathematics and Computers in Simulation 81,* 1, 109 – 119.

PRYCE, J., NEDIALKOV, N. S., AND TAN, G. 2013. DAESA — a Matlab tool for structural analysis of DAEs: Theory. Accepted for publication in ACM Trans. on Math. Softw.

PRYCE, J. D. 2001. A simple structural analysis method for DAEs. *BIT 41,* 2, 364–394.

WASHINGTON, I. AND SWARTZ, C. 2011. On the numerical robustness of differential-algebraic distillation models. In *61st Canadian Chemical Engineering Conference*. London, Ontario, Canada.

WATSON, L. T. 1979. A globally convergent algorithm for computing fixed points of C^2 maps. *Appl. Math. Comput. 5*, 297–311.