Towards Efficient DAE Solvers for the Solution of Dynamic Simulation Problems

A. J. Preston, M. Berzins, P. M. Dew, L. E. Scales

University of Leeds, Shell Research Ltd., Thornton Research Centre

1 Introduction

Dynamic simulation problems in the chemical engineering industry often involve computing the solution to large sparse systems of index two differential-algebraic equations (DAEs) with frequent discontinuities in the solution. This paper reports on work to improve the efficiency and robustness of SPRINT [1] for this class of problems. These improvements consist of an extension of SPRINT [1] to handle index two DAEs, a study of mechanisms for coping with discontinuities, and an initial investigation into the potential benefits and feasibility of porting SPRINT [1] to a highly parallel processor array machine, such as the Meiko Computing Surface.

The chemical and production processes employed in the petroleum and chemical industries involve increasingly specialised machinery and complex control systems. The efficient and robust control of such processes requires the accurate prediction of the dynamic response to changing operation conditions. The mathematical modelling of these systems requires the solution of a large set of sparse, stiff differential algebraic equations representing gas compositions, temperatures, pressures, flow rates and control parameters. Commercially available simulation software employs explicit time integration methods, needing very many time steps, and solves the algebraic equations by ad-hoc iterative techniques based on engineering insight into the connectivity of the system and the direction of information flows. SPRINT [1] is a general purpose software package for the numerical solution of time-dependent differential equations. The kernel of the software is a d.a.e integrator package based on backward differentiation formulas (BDFs). Although SPRINT [1] is widely used by mathematical modellers within Shell Research, it was not designed to cope with the particular difficulties arising from dynamic simulation problems. The aim of this paper is to describe the modifications needed to SPRINT [1] to accommodate this class of problems. To gain a better understanding of the problems encountered, the following case study has been investigated:

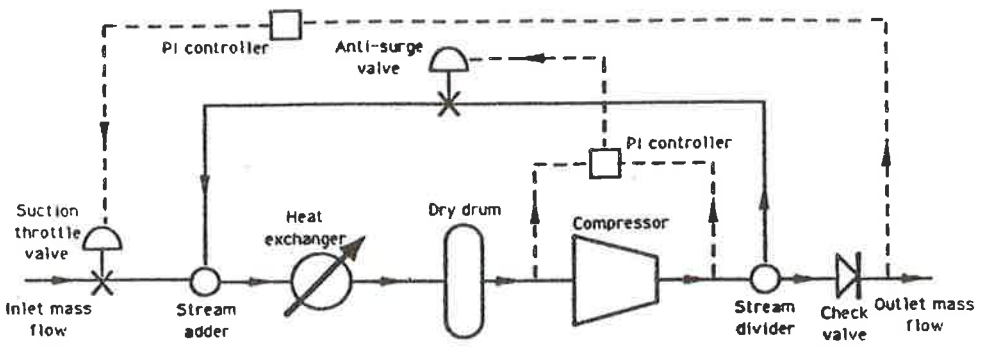

Figure 1. A dynamic simulation case study

The solid lines with arrows indicate the direction of physical output flow from each of the components (e.g. dry drum, compressor, etc.), and the dotted lines represent information flow that controls the position of the anti-surge and suction-throttle valves. The case study essentially models the behaviour of the components in the network as the inlet or outlet mass flow changes. There are a number of points that can be made about this case study. The first concerns the *check valve* after the *stream division* process, which is a device that is either fully open or fully closed. The inclusion of this device results in a system of DAEs that exhibit index two behaviour (see Section 2). The second point concerns the operation of the anti-surge and suction-throttle valves which can introduce discontinuities in the derivative of the variables that model the position of these valves (see Section 3). Finally, the number of variables in these simulations can be large, and it is of interest to consider how the solution could be obtained on a processor array (see Section 4).

2 A Robust Integrator for Dynamic Simulation Problems

The dynamic simulation problems result in a system of equations that belongs to the general class of DAEs given by

$$A(t, \mathbf{y}(t))\dot{\mathbf{y}}(t) = \mathbf{g}(t, \mathbf{y}(t)), \text{given } \mathbf{y}(0),\ \dot{\mathbf{y}}(0) \text{ and } \mathbf{g} : [t_0, T] \times \mathbb{R}^n \rightarrow \mathbb{R}^n \tag{2.1}$$

where A is a square matrix that can be singular. Gear and Petzold [4] have shown that DAEs can be classified according to the *index* of the system of equations. Broadly speaking, a system of equations has index two when at most two differentiations of each equation in the system are sufficient to formulate a first-order system of ordinary differential equations (ODEs), while one differentiation is not sufficient to do so. The case study

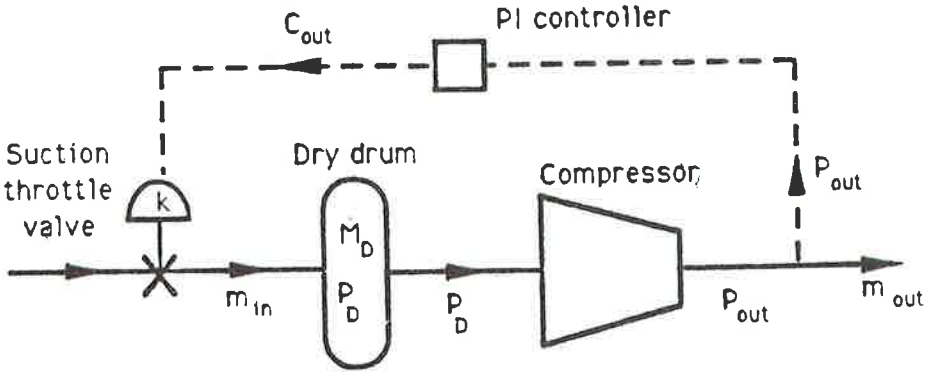

Figure 2. A model dynamic simulation problem

outlined above was found to exhibit index two behaviour, and this can be proven with considerable algebraic manipulation. To understand these problems in greater depth, it has been useful to consider a simplified form of the case study, as shown in the model above. This model consists of a compressor with a suction-throttle valve to control discharge pressure. A simplified mathematical model of this system is given by the following non-dimensional equations. Note that y_5 does not appear to the left of any of the equations.

$$y_1 = \text{valve position}(k). \quad \dot{y_1} = \begin{cases} \min\{(y_2 - y_1)/20, 0\} & \text{if } y_1 \geq 1 \\ \max\{(y_2 - y_1)/20, 0\} & \text{if } y_1 \leq 0 \\ (y_2 - y_1)/20 & \text{if } 0 < y_1 < 1 \end{cases} \tag{2.2}$$

$$y_1(0) = 0.25 \tag{2.3}$$

$$\begin{aligned} y_2 &= \text{controller output signal}(C_{out}). \\ \dot{y_2} &= -(\dot{y_3} + (y_3 - 99.10)/5)/15 \\ y_2(0) &= 0.25 \end{aligned} \tag{2.4}$$

$$\begin{aligned} y_3 &= \text{compressor output pressure}(P_{out}). \\ y_3/y_4 &= 3.35 - 0.075\ y_5 + 0.001 y_5^2 \\ y_3(0) &= 99.1 \end{aligned} \tag{2.5}$$

$$\begin{aligned} y_4 &= \text{compressor inlet pressure}(P_D). \\ y_4^2 &= 49.58^2 - (y_7/(1.2y_1))^2 \\ y_4(0) &= 36.7 \end{aligned} \tag{2.6}$$

$$\begin{aligned} y_5 &= \text{outlet mass flow}(m_{out}). \\ y_6 &= 20 y_4 \\ y_5(0) &= 10 \\ y_6 &= \text{mass in drum}(M_D). \end{aligned} \tag{2.7}$$

$$\begin{aligned} \dot{y}_6 &= y_7 - y_5 \\ y_6(0) &= 734 \qquad (2.8) \\ y_7 &= \text{inlet mass flow}(m_{in}). \\ y_7 &= 15 + 5\tanh(t-10),\ t \in [0,100], \\ y_7(0) &= 10 \qquad (2.9) \end{aligned}$$

Table 1. Comparison of the number of steps taken for the Index Two Model Problem

TOL	SPRINT [1]	New Module
0.1D-1	27*	56
0.1D-2	179*	88
0.1D-3	1000†	133

* Integration failure after about 10 seconds and
† Maximum number of steps taken to reach 9.19 seconds.

It is easy to see from this example how index one and two problems arise. When the inlet mass flow is explicitly stated, the dynamic simulation problem gives rise to an index two problem. On the other hand, when the outlet mass flow is explicitly stated, this results in an index one problem. A number of experiments have been carried out with index two DAEs, and improvements have been made to (i) the method of solution of the nonlinear equations, (ii) how the local error should be estimated, and (iii) the step-size and order selection strategy. For full details the reader is referred to [2]. This has resulted in an experimental SPRINT [1] module to handle index two problems. It uses an algorithm based on DASSL [9] with a modified stepsize and order mechanism. The local error estimate used is the reliable 'filtered' estimate proposed by Petzold [8], and used in the SPRINT software [1]. Results showing the number of steps taken by the unaltered SPRINT module, and the new module for the model problem are shown in Table 1.

These results are encouraging. More importantly, a similar improvement to the performance of the integrator was observed for the case study [2]. However, there still remain other unresolved issues beyond the scope of this paper, such as the best way of calculating accurate and consistent initial conditions [7].

3 Discontinuity Handling

To highlight the main features of the discontinuity problem, consider the Case Study. The position of the anti-surge and suction throttle valves will be either open ($y_1 = 1$) or closed ($y_1 = 0$), or else somewhere in between the two ($0 < y_1 < 1$), and discontinuities will be formed in $\dot{y}_1$ when the valve changes its state.

The valve model (2.2) allows values of y_1 outside the interval $[0, 1]$. This is because the valve position is fixed (by setting $\dot{y}_1 = 0$) when the valve opens at (or 'beyond') fully open (i.e. $\dot{y}_1 > 0$ and $y_1 = 1 + \varepsilon$ for $\varepsilon > 0$), or when the valve closes at (or 'beyond') fully closed (i.e. $\dot{y}_1 < 0$ and $y_1 = -\varepsilon$ for $\varepsilon > 0$). Petzold [8] explains that, even if the solution components are exact before the discontinuity, then a code based on BDF may well fail on the step after the discontinuity if the change in the derivative is large enough for the error test never to be satisfied. Even if the code steps away from the discontinuity, then it may generate unphysical (and possibly catastrophic) values for some of the component variables. It is therefore necessary to locate the discontinuity accurately and efficiently and then restart the integration from that point. Essentially there are two types of discontinuity to consider, which are referred to here as *explicit* and *implicit* respectively. In the explicit case, once a valve changes its state to fully open (or fully closed respectively), the discontinuity is located efficiently by performing a linear interpolation backwards on the valve position (using the predicted value after the discontinuity, corrected value before the discontinuity, and known state of the valve after the discontinuity). It is possible to include regular checks for this kind of discontinuity through the use of a switching function which changes sign when the valve opens or closes. Once a sign change occurs, there is a discontinuity and it is necessary to interpolate backwards to find the value of t at which the discontinuity occurred. This is principally the same approach as that used by Smith [11] and others.

The implicit case appears when a valve changes its state from fully open (or equivalently from fully closed). Suppose that an implicit discontinuity in $\dot{y}_1$ is expected to occur at some time t_D, bounded between integration times t_k and t_{k+1}. In the locality of the discontinuity, the valve position is constant prior to $t = t_D$ ($y_1 = 1$ for $t \leq t_D$), and linear immediately after $t = t_D$. In this case the valve is fully open prior to the discontinuity, so attempting to estimate its position using linear interpolation gives the estimate of t_D as t_k.

Interpolation on two values beyond the discontinuity would involve using possibly unphysical estimates of the valve position, and does not seem to be a sensible approach. Instead, a switch function can be defined using the C^0 continuous values of y_1, or the piecewise constant values of $\dot{y}_1$ in

Table 2. Handling of discontinuities

CASE	TOL	STEP	CALLS	JAC	ITS
A	0.1	35	2351	51	127
A	0.01	35	2135	46	129
A	0.001	103*	-	-	-
B	0.1	32	2165	47	112
B	0.01	32	2042	44	119
B	0.001	52	2214	45	207

the following way for a valve closing from fully open:

$$f(t, y_1, \dot{y}_1) = \begin{cases} 1 & \text{if } y_1 = 1 \text{ (or alternatively if } \dot{y}_1 = 0) \\ g(y_1, \dot{y}_1) & \text{if } y_1 < 1 \text{ (or alternatively if } \dot{y}_1 < 0) \end{cases}$$

To satisfy the requirements of a switch function, $g(y_1, \dot{y}_1)$ should be either monotonically decreasing from some non-positive value, or should remain constant negative after the discontinuity. Practical work shows that the best rate of convergence to the estimated location of the discontinuity that can be achieved in the general implicit case amounts to taking $g(y_1, \dot{y}_1) = -1$. This is essentially equivalent to the bisection method used by Gear and Osterby [3] for locating the root of an implicit discontinuity in a system of ODEs. Integration is then restarted where necessary.

An alternative approach would be to use the valve control signal as part of the switch function. This approach, which will be the subject of our future investigations, should provide a switch function which varies more smoothly, and so enables the discontinuity to be located more efficiently.

Experimental results for the case study (without check valve), using the modified SPRINT [1] module described above, are given in Table 2. In CASE A, the positions of the valves have been set to 1 only when they were predicted greater than 1, and to 0 when predicted to be less than 0. We then proceeded to integrate in the normal way. In CASE B, we have incorporated the discontinuity handling techniques as described above. These results clearly illustrate the need for special handling of discontinuities.

```
STEP    =   Steps taken

CALLS   =   Function Evaluations

JAC     =   LU Decompositions of Jacobian matrix

ITS     =   Total number of iterations
```

```
(* SPRINT [1] was unable to calculate the initial derivatives
shortly after restarting the integration)
```

4 Towards a Parallel Version of SPRINT for Dynamic Simulation Problems

The SPRINT [1] software is a modular package that was specifically designed to simplify the task of including variable step integration and linear algebra software. As a result there is choice of four integrators and three different linear algebra routines (full, banded and sparse) within SPRINT [1]. Another advantage of the modular structure is that it makes it easier to port the code to parallel computers, by simply replacing the computationally intensive modules. The main components of the present software are: (i) a STEP routine to advance the solution one timestep; (2) NLSLVR, a nonlinear solver called by STEP and (3) RESID to evaluate the residual equations. The module NLSLVR can be sub-divided further into three routines: (1) Jacobian matrix formation; (2) LU factorization; and (3) back-substitution.

From our practical experience we have found that a large proportion of the execution time in solving dynamic simulation problems (up to 80 percent) is taken up in the formation and decomposition of the Jacobian matrix. It is for this reason, as a first step, we have considered parallel linear algebra routines to increase the efficiency of the integration. An alternative approach would be to use the matrix-free methods being developed by Hindmarsh and Brown [6] and Seward [10], which involve only matrix-vector multiplications and so can be made to work in parallel more easily than sparse LU decomposition routines.

The experimental environment for deveioping a parallel version of SPRINT [1] is a Meiko Computing Surface, which is based on an array of INMOS T800 transputers. This machine belongs to a general class of processor network machines. The programming model is a set of sequential processors with a logical, fully connected network. Communications occur by commands of the type *write to destination* where the final destination of a message is specified rather than the channel through which it is to proceed. The new operating systems (e.g. CS-TOOLS) for the Meiko Computing Surface support this abstraction.

The most obvious approach to parallelizing SPRINT [1] is to adopt a master–slave model, where the master processor contains the user code, set-up routines, time integration driver and the STEP routine. The code for RESID, Jacobian matrix formulation, LU factorization and back-substitution are replicated on the slave processors. The slave processors can either pass back the part of a vector or its norm. The latter approach of passing norms reduces communication cost.

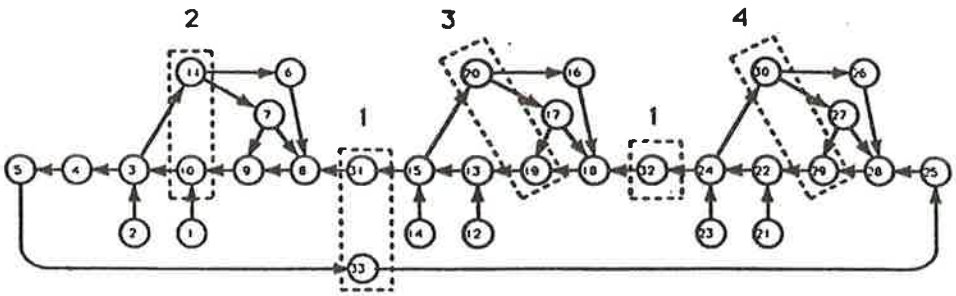

Figure 3. Graph of large dynamic simulation problem

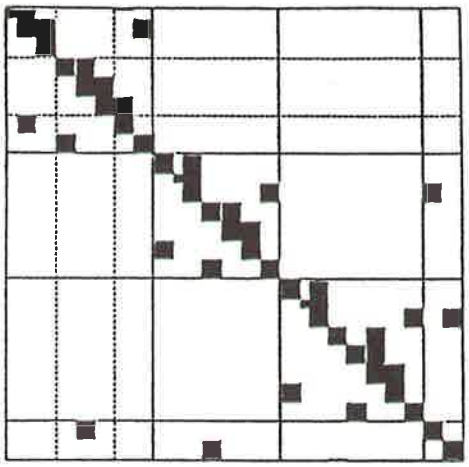

Figure 4. Sparsity pattern for a large dynamic simulation problem

The master-slave model requires that the linear algebra is partitioned into blocks and 'parcelled' out to the slave processors. For dynamic simulation problems, it is necessary to partition the graph of the network associated with the model. One approach that is being investigated is a variant of *Nested Dissection* [5]. This can be illustrated by the above digraph taken from a larger dynamic simulation problem: Here, the vertices represent variable sets and the digraph represents the block adjacency structure of the Jacobian matrix for a given vertex numbering. The first level decomposition is a trisection using the small separator vertex set 1, which results in three blocks of similar size. These can be dissected further using the separator sets 2, 3, and 4. This nested dissection is said to be incomplete because it has been terminated before reaching the bottom level, and the vertex numbering shown has been completed arbitrarily. The Jacobian matrix for the digraph has the sparsity pattern: One attraction

of nested dissection is the regular structure that is obtained which makes it possible to partition the problem into a number of subproblems. Solid lines correspond to first level partitions, dotted lines exemplify the second level. For example, ignoring pivoting, elimination under all of the diagonal blocks, at any level, can take place in parallel. Pivoting row interchanges must be confined to diagonal blocks if the structure required for parallelism is not to be lost. If the separator sets are small, the vast majority of pivots will be available. Failing this, in the d.a.e context the Jacobian can always be made sufficiently diagonally dominant by reducing the step size.

The residual code will be distributed over all the slave processors but each processor will only be responsible for calculating a subset of the residuals as determined by the block of the Jacobian matrix it is allocated. The Jacobian matrix is evaluated in distributed form by finite differences using only the subset of residuals available to each processor. Experimental and theoretical work is underway at both Leeds and Shell Research to better understand the problems of parallelising SPRINT [1], but it is too early to report on the results of this work.

5 Conclusions

The paper has reported on work in progress to improve the efficiency of DAE codes for dynamic simulation problems. The modifications of SPRINT [1] to handle index two DAEs has been successfully tested on realistic problems and found to work satisfactorily. The results of the work on developing techniques for handling of the discontinuities shows promise but more work is needed to evaluate the techniques on realistic networks. Our early studies of parallelism suggest that it is worth investigating parallel algorithms based on nested dissection.

Acknowledgements

We wish to thank Shell Research Limited for permission to publish this paper and for supporting the SERC CASE studentship for Andrew Preston. We are also grateful for the help received from Simon Frost of Shell Research Ltd., and Jim Griffiths at Leeds has helped with developing a parallel version of SPRINT [1].

References

[1] Berzins M., Dew P.M.and Furzeland R.M. (1989) Developing P.D.E Software Using the Method of Lines and Differential Algebraic Integrators, *Appl. Numer. Math.*, 5, 375–397.

[2] Berzins M., Dew P.M. and Preston A.J. (1988) Integration Algorithms for the Dynamic Simulation of Production Processes, Report

88.20, School of Computer Studies, University of Leeds. (A condensed version of this report to appear in Proc. 3rd European Conference for Maths in Industry, held at University of Strathclyde, August 1988, Pub. Teubner-Reidel, Ed. McKee S., Owens D., Manley J.)

[3] Gear C.W., Osterby O. (1981) Solving Ordinary Differential Equations with Discontinuities, Report UIUCDCS-R-81-1064, Dep. Comp. Sci., University of Illinois IL61801.

[4] Gear C.W., Petzold L.R. (1984) O.D.E Methods for the Solution of Differential Algebraic Systems, *SIAM J. Num. Anal.*, **21**, 716-728.

[5] George A. (1973) Nested Dissection of a Regular Finite Element Mesh, *SIAM J. Numer. Anal.*, **10**, (2), 345-363.

[6] Hindmarsh A.C., Brown P.N. (1987) Reduced Storage Techniques in the Numerical Method of Lines, Report UCRL96261, Lawrence Livermore National Laboratory CA94550.

[7] Leimkuhler B.J., Petzold L.R. and Gear C.W. (1991) On the Consistent Initialization of Differential Algebraic Equations, *SIAM. J. Numer. Anal.*, 28, 205-226.

[8] Petzold L.R. (1982) Differential Algebraic Equations are not ODEs, *SIAM J. Sci. Stat. Comp.*, **Vol. 3**, 367-384.

[9] Petzold L.R. (1984) A Description of DASSL, Proc. of IMACS World Congress, Montreal.

[10] Seward W.L. (August 1989) Solving Large ODE Systems Using a Reduced System Iterative Matrix Solver, Rep. CS-89-38, Dep. Comp. Sci., University of Waterloo, Waterloo, Ontario, Canada.

[11] Smith G.J. (1985) Dynamic Simulation of Chemical Engineering, PhD. Thesis, University of Cambridge.