Comput. chem. Engng, Vol. 12, No. 5, pp. 377–382, 1988
Printed in Great Britain.

0098-1354/88 $3.00 + 0.00

DIFFERENTIAL-ALGEBRAIC SYSTEMS, THEIR APPLICATIONS AND SOLUTIONS

G. D. BYRNE
Computing and Telecommunications Systems Division, Exxon Research and Engineering Company, Clinton Township, Route 22 East, Annandale, NJ 08801, U.S.A.

and

P. R. PONZI
Chemical Engineering Technology Division, Exxon Research and Engineering Company, P.O. Box 101, Florham Park, NJ 07932, U.S.A.

(*Received* 5 *January* 1988)

Abstract—Here, we describe what differential-algebraic systems (DASs) are and where they arise. We also sketch the numerical methods that underly the software packages that can be used to solve DASs. Finally, we describe a two-phase, time-dependent, reaction, diffusion, convection problem and the numerical solution of this prototypical DAS.

1. INTRODUCTION

Differential-algebraic systems (DASs) arise frequently in chemical engineering. They may occur directly from the use of first principles to model a physical phenomenon, such as cracking a hydrocarbon liquid, as we shall see later. DASs may also arise indirectly when solving a partial-differential equation with essential boundary conditions by a Galerkin procedure (Leaf *et al.*, 1977). Models of this type occur in the modeling of heat transfer, and other aspects of chemical processes. DASs also arise in applying a sensitivity analysis to a model, such as a process in a batch reactor (Caracotsios and Stewart, 1985). Linda Petzold (1986) lists several other applications including: chemical vapor deposition, neural network modeling, magma flow in volcanos, gas transfer, No_x reduction, analysis of a protein mixture, atomospheric physics and distillation. These only represent the more or less traditional applications in the chemical engineering field. In a more general setting, because DASs do arise indirectly in the numerical solution of partial differential equations (PDEs), several software packages for solving partial differential equations have DAS solvers as their foundation. These PDE solvers use either fixed or dynamic grid strategies. As the use of DAS solvers increases, other applications in the chemical engineering field will undoubtedly be found. Now that we have said something about the applicability of DASs, we can briefly review the contents of the balance of this paper.

In the remainder of this section, we will see what DASs are and work with an example. In Section 2, we will review the underlying methods in modern DAS software. Finally, in Section 3, we will look at an example of a two-phase reactive flow and its solution. By the way, much of the review in Sections 1 and 2 is based on material in Gear and Petzold (1984), Petzold (1982a, b, 1986) and Löstedt and Petzold (1983) as well as Gear (1971b).

A DAS can be represented as a system of N equations as follows:

$$\left.\begin{aligned} \mathbf{F}(t, \mathbf{y}, \dot{\mathbf{y}}) = 0, \quad t_0 \leqslant t \leqslant t_{\text{final}} \\ \mathbf{y}(t_0) = \mathbf{y}_0 \end{aligned}\right\}. \tag{1}$$

In (1), $\mathbf{F} = [F^1, F^2, \ldots F^N]^T$, where the superscript numerals are the indices of the components of $\mathbf{F}$ and the superscript T denotes the vector transpose. The dependent variable is $\mathbf{y} = [y^1, y^2, \ldots, y^N]^T$, t is the time-like independent variable, and $\dot{}$ is the time-like derivative $\mathrm{d}/\mathrm{d}t$. The initial value $\mathbf{y}_0$ is prescribed.

A DAS is a coupled system of ordinary differential equations (ODEs) and algebraic equations. That is, some of the equations in (1) do not have a corresponding component of $\dot{\mathbf{y}}$. Consequently, the matrix:

$$\frac{\partial \mathbf{F}}{\partial \dot{\mathbf{y}}} = \left[\frac{\partial F^i}{\partial \dot{y}^j}\right],$$

is singular. We can now turn to the following example Löstedt and Petzold (1983):

$$\left.\begin{aligned} \dot{y}^1 - y^2 = 0 \\ \dot{y}^2 - y^3 = 0 \\ y^1 - g(t) = 0 \end{aligned}\right\}. \tag{2}$$

As we shall see, this example has several interesting features. If we differentiate the algebraic equation, substitute appropriately and continue this process until we have differentiated the algebraic equation three times, then we can *formally* reduce this DAS to the following system of ODEs:

$$\left.\begin{aligned} \dot{y}^1 = \dot{g}(t) \\ \dot{y}^2 = \ddot{g}(t) \\ \dot{y}^3 = \dddot{g}(t) \end{aligned}\right\}. \tag{3}$$

If g is twice continuously differentiable and if the initial conditions are compatible [i.e. $y^1(t_0) = g(t_0)$, $y^2(t_0) = \dot{g}(t_0)$, $y^3(t_0) = \ddot{g}(t_0)$] then the solution for this example is:

$$\left.\begin{aligned} y^1 &= g(t) \\ y^2 &= \dot{g}(t) \\ y^3 &= \ddot{g}(t) \end{aligned}\right\}. \tag{4}$$

If the continuity conditions or the compatibility conditions are not satisfied, then the solution to this example problem may not exist everywhere or it may not exist at all. This serves to illustrate that the solution of a DAS may not be as straightforward as it might first appear.

Note that (2) can also be put in the frequently found semiexplicit form of a DAS:

$$\left.\begin{aligned} \dot{\mathbf{x}} &= \mathbf{F}_1(t, \mathbf{x}, \dot{\mathbf{x}}, \mathbf{w}) \\ \mathbf{w} &= \mathbf{F}_2(t, \mathbf{x}, \mathbf{w}) \end{aligned}\right\}. \tag{5}$$

In the case of (2), $\mathbf{x} = [y^1, y^2]^T$, $\mathbf{w} = y^3$, $\mathbf{F}_1$ corresponds to the first two equations, and $\mathbf{F}_2$ corresponds to the third equation. It is worth noting that some semiexplicit equations can in turn be recast in the linearly implicit form:

$$\mathbb{A}(t, \mathbf{y})\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}), \tag{6}$$

where $\mathbb{A}$ is an $\mathbb{N} \times \mathbb{N}$ matrix, $\mathbf{y}$ is an $\mathbb{N}$ vector, and $\mathbf{f}$ is an $\mathbb{N}$ vector function. Systems of the type (6) arise from the use of the finite element method (Leaf *et al.*, 1977), heat transfer problems with nonlinear heat capacity (Leaf *et al.*, 1977) and implicit PDEs in reservoir simulation (Douglas, 1970), (Peaceman, 1977). Note that $\mathbb{A}$ may be a singular matrix. In particular, if we put the example (2) in the form (6), we have:

$$\mathbb{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{y} = [y^1, y^2, y^3]^T$$

$$\mathbf{f}(t, \mathbf{y}) = [y^2, y^3, -y^1 + g(t)]^T$$

So far, we have seen where DASs arise, reviewed some of the principal types of DASs, and discussed what a DAS is. We now turn to a sketch of the methods that are used in the most common software packages for solving DASs.

2. NUMERICAL METHODS AND SOFTWARE

The following is a *sketch* of the numerical methods implemented in DASSL (Petzold, 1982b). This software package is used for DASs in the form (1). As in the case of most DAS and ODE solvers. DASSL computes a discrete variable solution. The idea is to compute $\mathbf{y}_n$ an approximation to the exact solution of (1) at t_n. Here,

$$t_n = t_0 + \sum_{j=1}^{n} h_j, \quad h_j = t_j - t_{j-1}.$$

The basic idea for solving the DAS (1) (Gear, 1971b) is this: as it stands, $\mathbf{F}(t, \mathbf{y}, \dot{\mathbf{y}}) = 0$ needs to be solved for both $\mathbf{y}$ and $\dot{\mathbf{y}}$. Suppose we replace $\dot{\mathbf{y}}$ by an approximation, such as:

$$(\mathbf{y}_n - \mathbf{y}_{n-1})/h_n. \tag{7}$$

This gives the system of algebraic equations:

$$\mathbf{F}[t_n, \mathbf{y}_n, (\mathbf{y}_n - \mathbf{y}_{n-1})/h_n] = 0. \tag{8}$$

We can solve the system of equation (8) with a Newton-type method. In particular, define the Newton matrix $\mathbb{N}$ by

$$\begin{aligned} &\mathbb{N}[t_n, \mathbf{y}_n, (\mathbf{y}_n - \mathbf{y}_{n-1})/h_n] \\ &\quad = \mathbf{F}_y[t_n, \mathbf{y}_n, (\mathbf{y}_n - \mathbf{y}_{n-1})/h_n] \\ &\qquad + \left(\frac{1}{h_n}\right)\mathbf{F}_{\dot{y}}[t_n, \mathbf{y}_n, (\mathbf{y}_n - \mathbf{y}_{n-1})/h_n]. \end{aligned} \tag{9}$$

The $(m+1)$st approximation to $\mathbf{y}_n$ (denoted by $\mathbf{y}_{n(m+1)}$) can be computed as:

$$\left.\begin{aligned} &\mathbb{N}[t_n, \mathbf{y}_{n(m)}, (\mathbf{y}_{n(m)} - \mathbf{y}_{n-1})/h_n](\Delta\mathbf{y})_{(m)} \\ &\quad = \mathbf{F}[t_n, \mathbf{y}_{n(m)}, (\mathbf{y}_{n(m)} - \mathbf{y}_{n-1})/h]\,\mathbf{y}_{n(m+1)} \\ &\quad = \mathbf{y}_{n(m)} - (\Delta\mathbf{y})_{(m)} \end{aligned}\right\} \tag{10}$$

The starting value for the iteration $\mathbf{y}_{n(0)}$ can be obtained from a predicted value for $\mathbf{y}_n$. When the process has converged, the iterate is taken to be $\mathbf{y}_n$. In most codes for ODEs and DASs the number of iterations is limited to two or three. If convergence does not occur within this limit, then h_n is typically reduced and the step is retired.

In practice, a first-order approximation to $\dot{\mathbf{y}}_n$ of the type indicated above is computationally inefficient. The backward differentiation formula methods popularized by C. W. Gear can be used for this approximation. That is, (Gear, 1971a):

$$\dot{\mathbf{y}}_n = \frac{1}{h_n\beta_0}\left[\mathbf{y}_n - \sum_{i=1}^{q} \alpha_i \mathbf{y}_{n-1}\right]. \tag{11}$$

Here, q denotes the order of the BDF (relative to $\mathbf{y}_n$) and is limited to the range 1–5. The coefficients are known for fixed h_n and q and can be easily computed for variable step size h_n. Note that the Newton-type iteration needs to be revised somewhat. With some abuse of notation, we have:

$$\mathbb{N} = \mathbf{F}_y + \frac{1}{h_n\beta_0}\mathbf{F}_{\dot{y}}. \tag{12}$$

The functions are evaluated at $(t_n, \mathbf{y}_n, \dot{\mathbf{y}}_n)$, with $\dot{\mathbf{y}}_n$ replaced by its BDF approximation.

The term *Newton-type* has been used because the Newton matrix $\mathbb{N}$ is not recomputed during each iteration. Rather, it is computed at some point and retained until the convergence is deemed to be too slow. Typically this evaluation of $\mathbb{N}$ is done with a predicted value of $\mathbf{y}_n$.

It is worth noting that useful codes for solving ODEs and DASs automatically select the step size h_n so that the problem can be efficiently solved to the

user specified accuracy. In the case of the BDF based methods, q, the order is also automatically and dynamically selected for this purpose.

The above sketch is intended to give some feeling for the way the DAS solver DASSL works. There are other software packages (Byrne and Hindmarsh, 1987), such as LSODI (Hindmarsh, 1980) and its ancestors, which have been used and are being used to solve DASs. These are intended to solve linearly implicit systems in the form (6), which we reproduce here:

$$\mathbb{A}(t, \mathbf{y})\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}),$$

$$\mathbf{y}(t_0) = \mathbf{y}_0.$$

These solvers follow the basic method sketched above $\dot{\mathbf{y}}_n$ is replaced by its BDF approximation and the following system is solved at each time step:

$$\mathbb{A}(t_n, \mathbf{y}_n)\left[\mathbf{y}_n - \sum_{i=1}^{q} \alpha_i \mathbf{y}_{n-i}\right] - h_n \beta_0 \mathbf{f}(t_n, \mathbf{y}_n) = \mathbf{0}. \quad (13)$$

The solvers for linearly implicit systems also use Newton type methods and select h_n and q automatically and dynamically. These methods have been used rather routinely to solve DASs for many years, in the setting of PDE software (Leaf *et al.*, 1977). By the way, electric and electronic circuits were the applications areas for much of the early work in DASs (Gear, 1971b; Hachtel *et al.*, 1971).

Now we turn to an example of an industrial problem that leads to a DAS.

3. AN EXAMPLE

This example is based on an industrial model of a two-phase time-dependent reaction, convection, diffusion problem. The system of equations for this model for cracking a hydrocarbon liquid includes:

- Mass balances
- Heat balance
- Equilibrium relations
- Mass fraction constraints

We can represent the model by the following system of partial differential equations and algebraic equations.

3.1. Mass balances

$$\begin{aligned}&\partial_t[\rho\epsilon y_i + \rho'(1-\epsilon)x_i] \\ &\quad + \partial_z[\rho\epsilon u y_i + \rho'(1-\epsilon)v x_i] \\ &= \partial_z[\rho\epsilon D\partial_z y_i + \rho'(1-\epsilon)D'\partial_z x_i] \\ &\quad - \epsilon R_i - (1-\epsilon)R_i' \end{aligned} \quad (14)$$

In the mass balance equation for composition i, above:

$\partial_t = \dfrac{\partial}{\partial t}$ and $\partial_z = \dfrac{\partial}{\partial z}$,

t = time,

z = distance along the reactor,

ρ = density of the gas (or discrete) phase,

ϵ = the gas phase volume fraction,

i = component index,

$'$ = liquid (or continuous) phase,

y_i = mass fraction of component i in gas phase,

x_i = mass fraction of component i in liquid phase,

u = velocity of the gas phase,

v = velocity of the liquid phase,

D = diffusion coefficient,

R = reaction rate term.

From top to bottom each line in the mass balance for component i (14) represents:

- Accumulation
- Convection
- Diffusion
- Reaction

When we specify the reaction terms and the number of components NS, we will then have a concrete representation of the mass balance. The boundary conditions for the x_i and y_i are the Dankwerts conditions at both the inlet z_{in} and the outlet z_{out}. That is:

$$\begin{aligned}&[\rho\epsilon u y_i + \rho'(1-\epsilon)v x_i]_{feed} \\ &\quad = [\rho\epsilon u y_i + \rho'(1-\epsilon)v x_i]_{z_{in}} \\ &\qquad - [\rho\epsilon D\partial_z y_i + \rho'(1-\epsilon)D\partial_z x_i]z_{in} \end{aligned} \quad (15)$$

at $z = z_{in}$ and

$$\partial_z x_i = \partial_z y_i = 0 \quad \text{at} \quad z = z_{out}. \quad (16)$$

3.2. Heat balance

$$\begin{aligned}&\partial_t[\rho\epsilon H + \rho'(1-\epsilon)H'] \\ &\quad + \partial_z[\rho\epsilon u H + \rho'(1-\epsilon)v H'] \\ &= \partial_z[(\epsilon\mathscr{K} + (1-\epsilon)\mathscr{K}')\partial_z T] - h_a(T - T_a), \end{aligned} \quad (17)$$

where the newly introduced symbols are as follows:

$H_i = \Sigma\, y_i h_i$ = total enthalpy of gas phase,

$H_i' = \Sigma\, x_i h_i'$ = total enthalpy of liquid phase,

h_i = enthalpy of gas phase of component i,

h_i' = enthalpy of liquid phase of component i,

$\mathscr{K}$ = thermal conductivity coefficient,

T = temperature of the system,

T_a = ambient temperature,

h_a = heat transfer coefficient.

From top to bottom, the terms in (17) represent:

- Accumulation
- Heat transfer by convection
- Heat transfer by conduction
- Heat transfer between the system and its environs

The boundary conditions for the heat balance are also taken to be of the Dankwerts type. That is:

$$\begin{aligned}&[\rho\epsilon u H + \rho'(1-\epsilon)v H']_{feed} \\ &\quad = [\rho\epsilon u H + \rho'(1-\epsilon)v H']_{z_{in}} \\ &\qquad - [(\epsilon\mathscr{K} + (1-\epsilon)\mathscr{K}')\partial_z T]_{z_{in}}, \end{aligned} \quad (18)$$

at $z = z_{in}$ and

$$\partial_z T = 0 \quad \text{at} \quad z = z_{out} \tag{19}$$

3.3. Equilibrium relation

The equilibrium relation is given by:

$$y_i = K_i x_i, \tag{20}$$

where

K_i = equilibrium constant for the ith component.

3.4. Mass fraction constraints

The mass fraction constraints imposed on the system of equations are:

$$\sum_{i=1}^{NS} x_i = \sum_{i=1}^{NS} y_i = 1, \tag{21}$$

where

NS = number of components in the system.

At this stage, we have a system of partial-differential equations subject to constraints and boundary conditions. The constraints are algebraic equations. It is worth noting that the following are to be calculated by a physical properties software package: ρ, ρ', h_i, h_i', and K_i. The next step involves the conversion of the PDEs into ODEs. There are many techniques to do this. The basic idea is to treat the spatial variable z as a discrete variable and the time variable t as a continuous variable. We could do this by using a Galerkin procedure, finite differences, or a number of other techniques. Indeed, there are software packages which could be directly applied to these PDEs for the automatic discretization and solution of the system.

As it stands, the mathematical model for the reactive two-phase flow consists of $NS + 1$ PDEs (14) (17), $NS + 2$ algebraic equations (20) (21) and the boundary conditions (15) (16) (18) (19). In representing the accumulation terms in the PDEs describing mass balance and the PDE for heat balance, we can avoid some complexity by introducing $NS + 1$ artificial variables:

$$\left.\begin{aligned} G_i &= [\rho\epsilon y_i + \rho'(1-\epsilon)x_i] \\ H_T &= [\rho\epsilon H + \rho'(1-\epsilon)H'] \end{aligned}\right\}. \tag{22}$$

We can regard these artificially introduced variables as:

G_i = total mass density of component i in the system,

H_T = total enthalpy density.

To convert the PDEs in this model to ODEs, we discretize the z axis uniformly by setting $\Delta z = (z_{out} - z_{in})/M$. This gives $z_j = z_{in} + j(\Delta z)$. Next, we replace the first-order spatial derivatives in the convective terms by first-order backward (upwind) differences and the second-order derivatives in the diffusion terms by second-order central differences. We *could* have used a nonuniform grid, higher-order differences, a Galerkin procedure, etc.

In the following system of ODEs, j denotes evaluation at z_j, while i remains the composition index. The semidiscrete or spatially-discretized-continuous-time version of the mass balance (15) for component i at grid point j is:

$$\begin{aligned} \dot{G}_{i,j} &+ \frac{1}{\Delta z}[\rho_j\epsilon_j u_j y_{i,j} - \rho_{j-1}\epsilon_{j-1}u_{j-1}y_{i,j-1}] \\ &+ \frac{1}{\Delta z}[\rho'_j(1-\epsilon_j)v_j x_{i,j} \\ &- \rho'_{j-1}(1-\epsilon_{j-1})v_{j-1}x_{i,j-1}] \\ &= (\rho_{j-1/2}\epsilon_{j-1/2}D_{j-1/2}[y_{i,j-1} - y_{i,j}] \\ &+ \rho_{j+1/2}\epsilon_{j+1/2}D_{j+1/2}[y_{i,j+1} - y_{i,j}] \\ &+ \rho'_{j-1/2}(1-\epsilon_{j-1/2})D'_{j-1/2}[x_{i,j-1} - x_{i,j}] \\ &+ \rho'_{j+1/2}(1-\epsilon_{j+1/2})D'_{j+1/2}[x_{i,j+1} - x_{i,j}])\frac{1}{\Delta z^2} \\ &- \epsilon R_i - (1-\epsilon)R'_i. \end{aligned} \tag{23}$$

Similarly, the semidiscrete form of the heat balance (17) is:

$$\begin{aligned} \dot{H}_{Tj} &+ \frac{1}{\Delta z}[\rho_j\epsilon_j u_j H_j - \rho_{j-1}\epsilon_{j-1}u_{j-1}H_{j-1}] \\ &+ \frac{1}{\Delta z}[\rho'_j(1-\epsilon_j)v_j H'_j - \rho'_{j-1}(1-\epsilon_{j-1}) \\ &\times v_{j-1}H'_{j-1}] \\ &= ([\epsilon_{j-1/2}K_{j-1/2} + (1-\epsilon_{j-1/2})K'_{j-1/2}] \times [T_{j-1} - T_j] \\ &+ [\epsilon_{j+1/2}K_{j+1/2} + (1-\epsilon_{j+1/2})K'_{j+1/2}] \\ &\times [T_{j+1} - T_j])\frac{1}{\Delta z^2} - h_a[T_j - T_a]. \end{aligned} \tag{24}$$

In the ODEs for mass and heat balance (23) and (24), the subscript j runs from 1 to M. The Dankwerts boundary conditions for the x–s, y–s, and T–s are similarly discretized to give additional algebraic equations.

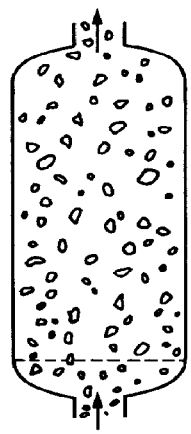

Fig. 1. Sketch of two-phase flow in reactor—liquid (continuous) phase and discrete (gas) phase.

We can now specify a prototypical four-component liquid cracking model. The reaction mechanism is:

$$\left.\begin{aligned} &A + b_1 B \rightarrow C, \quad k_1 \\ &C + b_2 B \rightarrow D, \quad k_2 \end{aligned}\right\}. \tag{25}$$

Next, we identify the components A, B, C and D with the subscripts 1–4 respectively, with the reaction rate coefficients k_1 and k_2. This gives:

$$\left.\begin{aligned} R_1 &= -k_1 x_1 x_2 \\ R_2 &= -b_1 k_1 x_1 x_2 - b_2 k_2 x_2 x_3 \\ R_3 &= (1 + b_1) k_1 x_1 x_2 - k_2 x_2 x_3 \\ R_4 &= (1 + b_2) k_2 x_2 x_3 \end{aligned}\right\}. \tag{26}$$

We used DASSL to solve this problem. Consequently we ordered the components of the dependent variable y for the integrator in this way:

$$\begin{aligned} \mathbf{y}^{\mathrm{T}} = [&x_{1,1}, \ldots, x_{NS,1}\, y_{1,1}, \ldots, y_{NS,1} \\ &G_{1,1}, \ldots, G_{NS,1}, H_{\mathrm{T},1}, u_1, v_1, T_1, \\ &x_{1,2}, \ldots, T_2, \ldots, T_M]^{\mathrm{T}}. \end{aligned} \tag{27}$$

Here, the dependent variables have been indexed by going through all of the unknowns at a grid point and then moving to the next grid point. The idea is to keep the underlying Jacobian matrix (and hence the Newton matrix) as tightly banded as possible. This helps to reduce both computer storage and run time requirements. We have a system of $16 \times M$ equations for this model.

We obtained the initial conditions for this model as follows. A vapor–liquid equilibrium calculation of the feed streams was used to calculate the initial value of the components and the velocities in each finite difference cell. The time derivatives were set to zero for each variable in this calculation. This computation gave a consistent starting value for the model, as long as no reactions were present in the system. The reactions were then turned on by using a multiplicative factor. This factor varied continuously from 0 to 1 as t increased from 0 and as steady state was approached. Because we were really interested in the steady state solution of this model, this "false transient" approach was acceptable.

We simply note that we did try the option in DASSL that causes the code to try calculating the initial time derivatives. It is because we were unsuccessful in this that we turned to the false transient technique.

The numerical results for a coarse 10 point grid are depicted in Figs 2–5. We solved this problem on an IBM 3081 in 1.5 min of CPU time. The profile of the Jacobian matrix for this prototypical example is

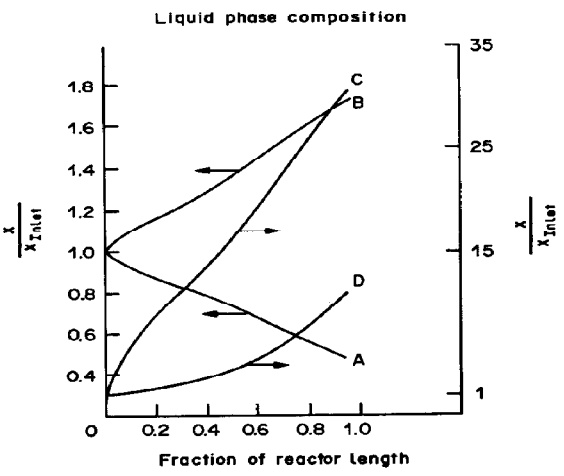

Fig. 2. Liquid phase composition vs fraction of reactor length.

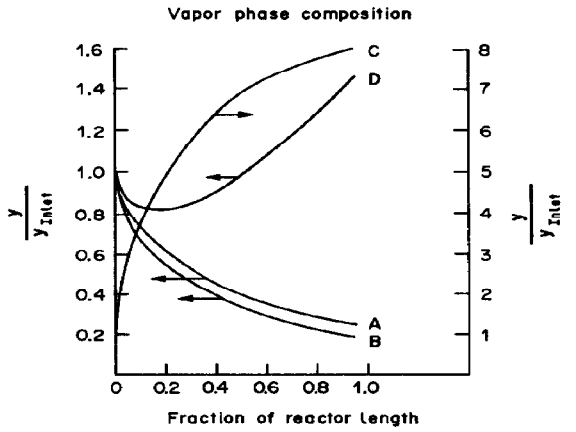

Fig. 3. Vapor phase composition vs fraction of reactor length.

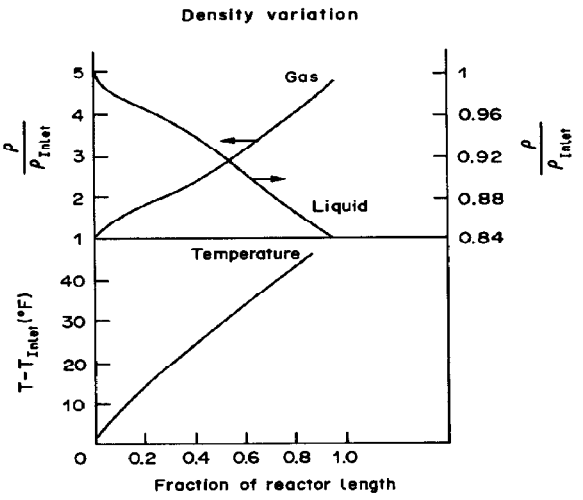

Fig. 4. (Top) Density variation vs fraction of reactor length.
Fig. 5. (Bottom) Temperature change vs fraction of reactor length.

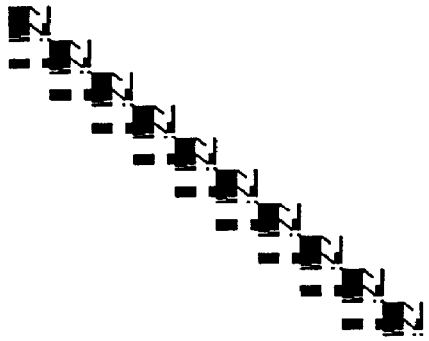

Fig. 6. Profile of nonzero elements of the Jacobian matrix for the prototype problem.

depicted in Fig. 6. In this case, we used $k_1 = 1$, $k_2 = 7$, $b_1 = 0.001$, and $b_2 = 0.005$.

Recently, we solved a larger industrial model similar to this. There were 17 species and 40 grid points. The Jacobian matrix had a lower-half bandwidth of 56. This took 4.8 megabytes of storage for double precision computation. This model required about 40 min of CPU time on the IBM 3081. There were 2500 function calls and 14 Jacobian calls for the computation. However, there were 30,000 calls to the physical properties package. These calls took about 75% of the CPU time. This is a significant comment on where efficiencies might be realized. Simply put, if all computations outside the physical properties were reduced to 0 CPU s, we would achieve only a 25% improvement in speed. Consequently, for problems of this type we might focus our attention on speeding up the physical properties package and/or reducing the number of calls to it.

We feel that DAS solvers can and will play a significant role in chemical engineering. One of the keys is the simultaneous treatment of both the differential equations and the algebraic equations in a mixed system or DAS. This eliminates the need for hooking together separate methods and software for the ODEs and the algebraic equations in an *ad hoc* way (Byrne and Ho, 1983).

REFERENCES

Byrne G. D. and A. C. Hindmarsh, Stiff ODE solvers: a review of current and coming attractions. *J. Comput. Phys* **70**, 1–62 (1987).

Byrne G. D. and W. S. Ho, A solution of differential-algebraic systems: a trick. Exxon Research and Engineering Co. Report CTSD.2DI.83 (1983).

Caracotsios M. and W. E. Stewart, Sensitivity analysis of initial value problems with mixed ODEs and algebraic equations. *Comput. chem. Engng* **9**, 359–365 (1985).

Douglas J. Jr, The numerical solution of a composition model in petroleum reservoir engineering. *Numerical Solution of Field Problems in Continuum Physics. SIAM-AMS Proceedings, Vol. II*, pp. 54–59. American Mathematical Society, Providence (1970).

Gear C. W., *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, Englewood Cliffs, New Jersey (1971a).

Gear C. W., Simultaneous numerical solution of differential algebraic equations. *IEEE Trans. Circuit Theory* **CT-18**, 89–95 (1971b).

Gear C. W. and L. R. Petzold, ODE methods for the solution of differential algebraic systems. *SIAM J. Numer. Anal.* **21**, 716–728 (1984).

Hachtel G. D., R. K. Brayton and F. G. Gustavson, The sparse tableau approach to network analysis and design. *IEEE Trans. Circuit Theory* **CT-18**, 101–113 (1971).

Hindmarsh A. C., LSODE and LSODI, two new initial value ordinary differential equation solvers. *ACM SIGNUM Newslett.* **15**, 10–11 (1980).

Leaf G. K. *et al.*, DISPL: A software package for one and two spatially dimensioned kinetics–diffusion problems. Argonne National Laboratory Report ANL-77-12 (1977).

Löstedt P. and L. R. Petzold, Numerical solution of nonlinear differential equations with the algebraic constraints. Sandia National Laboratory Report SAND83-8877 (1983).

Löstedt P. and L. R. Petzold, *Ibid.*, Part 1: convergence results for backward differentiation formulae. *Math. Comput.* **49**, 491–516 (1986); and Part 2: practical implications. *SIAM J. Sci. Statist. Comput.* **7**, 720–733 (1986).

Peaceman D. W., *Fundamentals of Reservoir Simulation*, Elsevier, New York (1977).

Petzold L. R., Differential/algebraic equations are not ODEs. *SIAM J. Sci. Stat. Comput.* **3**, 367–384 (1982a).

Petzold L. R. A description of DASSL: a differential-algebraic system solver. Sandia National Laboratory Report SAND82-8637; also in *IMACS Transactions on Scientific Computing* Vol. 1 (R. S. Stepleman *et al.*, Eds). pp. 65–68. North Holland, Amsterdam (1982b).

Petzold L. Numerical methods for differential/algebraic systems. Paper presented at the *1986 ODE Conf.*, Albuquerque (1986).