*Computers chem. Engng*, Vol. 14, No. 2, pp. 161–177, 1990
Printed in Great Britain. 

0098-1354/90 $3.00 + 0.00


# THE SEQUENTIAL-CLUSTERED METHOD FOR DYNAMIC CHEMICAL PLANT SIMULATION

J. C. FAGLEY[1] and B. CARNAHAN[2]

[1]B. P. America Research and Development Co., Cleveland, OH 44128, U.S.A.
[2]Department of Chemical Engineering, The University of Michigan, Ann Arbor, MI 48109, U.S.A.

(*Received* 28 *May* 1987; *final revision received* 26 *June* 1989; *received for publication* 11 *July* 1989)

**Abstract**—We describe the design, development and testing of a prototype simulator to study problems associated with robust and efficient solution of dynamic process problems, particularly for systems with models containing moderately to very stiff ordinary differential equations and associated algebraic equations.

A new predictor–corrector integration strategy and modular dynamic simulator architecture allow for simultaneous treatment of equations arising from individual modules (equipment units), clusters of modules, or in the limit, all modules associated with a process. This "sequential-clustered" method allows for sequential and simultaneous modular integration as extreme cases.

Testing of the simulator using simple but nontrivial plant models indicates that the clustered integration strategy is often the best choice, with good accuracy, reasonable execution time and moderate storage requirements.

## INTRODUCTION

Two major stumbling blocks in the development of robust dynamic chemical process simulators are: (1) mathematical models for many important equipment types give rise to quite large systems of ordinary differential equations (ODEs); and (2) these ODE systems tend to be moderately to very stiff. The development of robust stiff numerical integrators and introduction of more powerful computers (larger memories, faster processors, new architectures) during the past few years now make the dynamic simulation of many plant operations a solvable problem. We need reliable, accurate dynamic chemical plant simulators that make efficient use of these new computational tools and hardware. Special attention must be given to the treatment of the large Jacobian matrices required by stiff-system algorithms, even for processes with only a few units.

This paper describes study into efficient treatment of the dynamic chemical plant simulation task. The work involved design and construction of a prototype simulator capable of handling chemical processes giving rise to large stiff ODE systems. This research simulator employs test models for controllers, distillation equipment and a double-pipe heat exchanger, and makes extensive use of plex data structures for storing numerical integrator, process and physical property system variables and constants.

We implement a novel approach in modular dynamic simulation, viz. a single simulator architecture that allows for simultaneous treatment of equations arising from individual modules (equipment units), clusters of modules, or in the limit, all modules associated with a process. This paper describes the most important features of the prototype simulator and includes results of tests on simulator performance. Direct numerical comparisons of alternative integration strategies are given for some relatively simple test plants. Conclusions and recommendations drawn from our experiences should be of interest to other workers developing dynamic simulation software.

## BACKGROUND

The mathematical modeling of transient chemical process operations gives rise to differential/algebraic equation systems (principally from mass and energy conservation laws) that must subsequently be solved during execution of a dynamic process simulator. In some cases, the well-mixed assumption applies, and the system boundary may be taken around an equipment subunit, or even an entire unit. Examples of such lumped parameter systems are stirred tank reactors, mixers and individual stages in a distillation column.

If spatial gradients within an equipment unit are important, distributed parameter systems of equations arise. The unsteady-state modeling equations are partial differential equations (PDEs), typically parabolic or hyperbolic, that depend on the nature and level of detail of the model. Examples of distributed parameter systems are plug flow reactors, packed separation columns and heat exchangers.

Transient distributed parameter models can usually be treated using the "method of lines" (Byrne and Hindmarsh, 1977; Haydweiller *et al.*, 1977; Carver, 1979) in which spatial derivatives are approximated by finite-differences. As a result, a single PDE will be represented as a system of ODEs, with time as the independent variable. By using the method of

lines, the dynamic model equation system for a chemical plant consists primarily of ODEs that can be solved with a single numerical integrator.

The overall equation system will also contain algebraic equations that are linked to the ODE-dependent variables. Many of these associated algebraic equations are required for physical or transport property evaluation (e.g. equations of state might be used to determine densities).

Until fairly recently most simulator research efforts have focused on the modeling of steady-state process operations. Historically, early steady-state simulators (see reviews by Hlavacek, 1977; Motard *et al.*, 1975), and the most popular commercial ones, employ a sequential-modular structure in which the procedure for solving the model equations for an individual equipment unit is incorporated into an individual subroutine associated with that unit and assigned to a unit program library. An executive routine calls upon the module subroutines in sequence (in an order specified by the user or determined by a flowsheet analyzer), usually following material flow through the process from feed to product streams. Typically the module routines (and occasionally the executive program) call upon a physical property package to generate essential property values.

If recycle streams are present, one or more streams are "torn" to render the system acyclic, and the sequential-modular solution strategy involves a multitiered iterative structure on calculation: (1) an outermost iteration involving solution of nonlinear equations associated with variables in the torn streams; (2) sequential calls upon the unit module routines; (3) possibly iterative solution of the model equations within a module; and (4) evaluation of physical properties needed by the modules during solution of the model equations (the property evaluations may again involve iterative numerical methods). Many strategies have been developed for optimal or near-optimal module calculation ordering (e.g. Duczak, 1986).

"Design" problems (in which a sufficient number of input stream, output stream and equipment parameters are specified to render the equation system solvable for the remaining process variables) are not solved in a "natural" way by modular-sequential simulators. Multiple versions of the same model equations (involving different "known" and "unknown" variables) may be needed, increasing the complexity of the modules. Or substantial low-level repetition of "simulation" unit module calculations may be required to satisfy internal stream specifications. Some design specifications (e.g. for process product streams) may even lead to iteration loops outside those required for converging the recycle calculations. Thus the overall computational scheme for design problems may involve nested iteration of a fairly high order. Process optimization may add yet another level of iteration to the calculations, although Biegler and Hughes (1983), Biegler and Cuthrell (1985) and Biegler (1988) show that recycle convergence and optimization calculations can be performed simultaneously using successive quadratic programming algorithms.

Principal alternative steady-state simulator structures are those incorporated into simultaneous simulators, which come in several generic forms variously called equation-based, simultaneous, simultaneous-modular and modular-simultaneous (Perkins, 1983; Biegler, 1983, Shacham *et al.*, 1982; Stadtherr and Vegeais, 1986; Morton and Smith, 1989). In these simulators, all (or almost all) of the equations or simplified (frequently linearized) forms of the equations (including model equations, physical property equations, stream connection equations, process specification equations, etc.) are solved simultaneously.

The simultaneous simulators have the prime virtue that "simulation" and "design" problems can be solved in essentially the same way, provided that the appropriate number (and combination) of variables have been assigned values to render the equations solvable. The highly nested iterative structure needed to solve "design" problems with sequential-modular simulators is eliminated and optimization calculations can be performed using infeasible path algorithms (Hutchison *et al.*, 1986).

However, the simultaneous simulators have their own problems, principally resulting from the numerical solution of the very large system of simultaneous nonlinear equations generated. Usually Newton or quasi-Newton algorithms are used to solve these nonlinear equations. This requires repeated evaluation of the (partial derivative) elements in the Jacobian matrices for the equation system (see later), and subsequent solution of high-order sparse linear equation systems. Initial guesses must be provided by the user or generated by the simulator for all (possibly thousands for large plants) of the unknown process variables.

Although most attention has been focused on steady-state simulation and design, a few general-purpose dynamic process simulators (Ingels and Motard, 1970; Ham, 1971; Franks, 1972; Lopez, 1974; Barney, 1975; Patterson and Rozsa, 1980) have been developed during the past 20 yr. The structures of most of these simulators are parallel to those for the steady-state simulators, i.e. modular-sequential and simultaneous.

Virtually all modular dynamic simulators consist of four principal parts: (1) unit model subroutines that incorporate the model equations for the associated equipment types; (2) a physical property subsystem that provides estimates of densities, enthalpies, vapor–liquid distribution coefficients, etc.; (3) a numerical integrator; and (4) a supervisory routine that performs input/output operations and oversees execution of the simulation calculations.

These simulators are called "modular" because each piece of equipment in the physical plant has its

dynamic behavior represented by a set of differential and algebraic equations incorporated into a unit model subroutine. The simulator framework is a general one, i.e. once the unit model subroutines for "standard" unit types have been constructed, they are available in a unit model subroutine library, and may be drawn together in different ways depending on the layout of the plant being simulated and the particular integration strategy used.

The unit model routines use input (material and information) stream and equipment parameter values, call upon the physical property system routines as needed, and compute estimates of the derivatives for the ODE-dependent variables associated with the unit. The numerical integrator uses these derivative estimates to generate updated values for the ODE-dependent variables at discrete times, which are then used by the unit module routines to calculate output stream parameter values. Unit-related algebraic equations are also solved during each module call using appropriate, robust numerical methods.

The steady-state equation-based (simultaneous) simulators can also be used to solve ordinary differential equations in addition to nonlinear algebraic equations. The ODEs are typically integrated using predictor–corrector algorithms, which require, at each time step, the iterative solution of the corrector equations. Since the corrector equations are nonlinear algebraic equations, they can be solved by the same (typically Newton or quasi-Newton) algorithm that is used to solve the other nonlinear algebraic equations for the process.

From an aesthetic viewpoint, the generality of the simultaneous equation-based approach to solving process problems is very attractive. Both steady-state and dynamic results can be produced, and because all equations are being solved simultaneously, mathematical rigor is assured.

Unfortunately, the addition of a potentially large number of corrector equations (one for each ODE) to an already large set of associated nonlinear algebraic equations leads to nonlinear equation systems of large order, for which our current methods are very heavily taxed and may not be sufficiently robust to guarantee reliable performance. It is not clear from the literature that large dynamic process plant models involving thousands of mixed algebraic and differential equations have been solved successfully and consistently using this approach.

In recent years, several new approaches (Palusinski, 1985; Hillestad and Hertzberg, 1986; Liu and Brosilow, 1987) have been suggested to accomplish dynamic simulation in a modular framework. First, there is the technique of partitioning the flowsheet and identifying streams that interconnect units or subsets of the plant units. Each unit or subset is integrated using a separate numerical algorithm, with values of interconnecting stream parameters being interpolated/extrapolated in time. This technique requires a supervisory program that keeps all units or subsets at roughly the same time level, oversees extrapolation and interpolation of essential stream parameter values and provides a strategy for retracing integration time steps when extrapolated values later prove to be inaccurate.

This approach is probably of greatest utility in cases where individual units display a weak interdependence. When there is heavy coupling of plant units, and the units show a strong degree of interdependence through either material or information (control signal) stream connections, extrapolation in time is often inaccurate, leading to retracing of integration time steps.

Other novel approaches, developed in recent years, include: (1) partitioning of plant structure, with tear-stream values guessed and iterated upon (Ponton, 1983); (2) use of a two-tiered approach with separation of flow-pressure equations from composition-dependent equations (Ponton and Vasek, 1986); and (3) a technique that takes advantage of "latency", the property that only a few units in a process are usually dynamically "active" at any given time (Kuru and Westerberg, 1985).

In each case, the structure of the simulator and the way in which the executive interacts with the unit modules is dictated by the adopted algorithm and its implementation. In turn, the best application of each depends on the type of plant being simulated. The techniques which use stream tearing, extrapolation/interpolation in time and iteration around recycle loops are best suited to plants with minimal integration of material, information (control) and energy flow. Conversely, the more rigorous approaches with global treatment of the ODE set tend to be better suited for treating plants with a high degree of recycle (material, energy and control signal) and which incorporate advanced, integrated control algorithms, i.e. feedforward and decoupling loops. In these highly integrated systems, accurate description of flow between the units is paramount in describing the interdependence of the units.

In this paper, another novel algorithm will be introduced which incorporates the most global and mathematically rigorous technique, total simultaneous integration, in the flexible sequential modular simulation framework. The simulator structure that allows implementation of the simultaneous algorithm will also be described.

## STIFFNESS

One of the more serious problems associated with dynamic simulation of complex processing plants is that the ODE system models for a wide variety of chemical and chemical engineering problems are moderate to very stiff (Barney, 1975; Johnson and Barney, 1976; Weimer and Clough, 1979).

Stiff problems are frequently encountered in modeling reactive systems, when kinetic rate con-

stants differ by more than one or two orders of magnitude (DeGrout and Abbott, 1965; Enright and Hull, 1976; Guertin *et al.*, 1977). Stiff ODE systems also arise in simulating separation units, where trace components on trays of a distillation column may have response times several orders of magnitude smaller than reboiler/condenser response times (Barney, 1975; Franks, 1972; Distefano, 1968; Mah *et al.*, 1962).

Stiff equations almost always arise in dynamic chemical plant simulation involving solution of PDEs by the method of lines. As the spatial mesh size for method-of-lines treatment of PDEs is decreased, the approximation of the spatial derivatives becomes more exact, but the resulting ODE set becomes more stiff. For example, for 1-D thermal diffusion with homogeneous Dirichlet boundary conditions, theoretical analysis shows that the stiffness ratio increases as the square of the number of spatial sections (Byrne and Hindmarsh, 1987). These types of stiff problems have been encountered, for example, in modeling gas absorption columns and tubular reactors (Haydweiller *et al.*, 1977).

Several stiff numerical integration techniques have been developed in the past 20 yr or so: semi- and fully-implicit Runge–Kutta techniques, exponential-fitting methods, spline fitting approaches, higher-order derivative methods and linear multistep algorithms employing backward differences (Byrne and Hindmarsh, 1977). All of these stiff techniques remain stable for integration step sizes much larger than for classical Runge–Kutta and multistep algorithms, so that even severely stiff stable problems may be treated in a reasonable number of time steps. Of course small time steps must be used when rapid transients are significant, if these are to be computed accurately and observed. One feature common to stiff numerical integrators is that the system Jacobian matrix plays a central role in the solution algorithm.

Currently, the most popular stiff integrators are those developed by Gear (1968, 1971). Gear's methods have been found by many workers to outperform other types of stiff integrators on a wide variety of chemistry and chemical engineering problems. The Gear algorithms are linear, multistep predictor–corrector techniques, based on backward difference formulae. Gear integration, as implemented in the most popular stiff-system codes, has many desirable features, including automatic startup, automatic step-size and method-order adjustment capability and built-in error control (Hindmarsh, 1974a,b, 1977, 1979; Byrne and Hindmarsh, 1977, 1987). It is a variable-step, variable-order algorithm.

Consider the initial-value ODE problem:

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y}, t), \tag{3}$$

$$\mathbf{y}(0) = \mathbf{y}_0, \tag{4}$$

with the system Jacobian matrix:

$$[\mathbf{J}_{ij}] = \partial f_i / \partial y_j. \tag{5}$$

Here $\mathbf{y}$ is a column vector of $n$ ODE-dependent variables, $\mathbf{f}$ is a column vector of $n$ functional relationships for the dependent variable derivatives and $t$ is the independent variable, time.

The system Jacobian J is an $n \times n$ matrix of partial derivatives; its $n$ eigenvalues $\lambda_i$ which may contain both real and imaginary parts, $[k = \sqrt{(-1)}]$:

$$\lambda_i = -1/\tau_i + k\omega_i. \tag{6}$$

For a stable system, all values of $\tau_i$ will be greater than zero. The imaginary parts $\omega_i$ represent local oscillations in components of the solution of the ODE system. The stiffness ratio $S$ is a standard measure of the stiffness of the ODE system:

$$S = \tau_{\max}/\tau_{\min} \tag{7}$$

Here, $\tau_{\max}$ and $\tau_{\min}$ are the maximum and minimum values of the $\tau_i$. For $S$ in the range 1–10, the problem set is nonstiff, and may be treated effectively with a conventional nonstiff integration technique such as an Adams–Moulton predictor–corrector or explicit Runge–Kutta method. For $S$ in the range 100–1000, the equation set is moderately stiff. For larger $S$, conventional techniques either cannot be used at all or are very inefficient; numerical stability considerations dictate a very large number of very small integration time steps to complete the integration to a useful maximum time $T$. (Note: for nonlinear ODE sets, the values of $\tau_i$ and therefore of $S$ vary with time.)

The term $\tau_{\max}$ represents a slowly changing component in the solution, while $\tau_{\min}$ represents a rapidly changing component. For conventional nonstiff integrators (e.g. classical explicit Runge–Kutta or Adams–Moulton predictor–corrector methods of orders 4–6), the time step of integration must always remain small (of the order of $\tau_{\min}$ to maintain numerical stability). However, integration must take place over a relatively long time in order to determine the nature of the slowly varying components.

The predictor equations are used to generate initial estimates of $\mathbf{y}$ at each new time level $t_{i+1}$:

$$\mathbf{y}_{i+1}^{(0)} = \sum_{j=1}^{q} a_j \mathbf{y}_{i+1-j} + hb_0 \mathbf{f}(\mathbf{y}_i, t_i) \tag{8}$$

and the corrector equations are iterated to convergence using a Newton–Raphson or quasi-Newton technique:

$$\mathbf{y}_{i+1}^{(s+1)} = \mathbf{y}_{i+1}^{(s)} + (I - hc_0\mathbf{J}^*)^{-1} \times \left[ hc_0 f(\mathbf{y}_{i+1}^{(s)}) - \mathbf{y}_{i+1}^{(s)} + \sum_{j=1}^{q} d_j \mathbf{y}_{i+1-j} \right]. \tag{9}$$

Here, $b_0$, $c_0$ and the $a$s and $d$s are method constants, $h$ is the time step of integration $(t_{i+1} - t_i)$, $q$ is the method order and $s$ is the Newton iteration counter. $\mathbf{J}^*$ is the Jacobian matrix evaluated at the current

or some previous values of **y** and $t$. The method is referred to as a quasi-Newton technique because a single Jacobian matrix may be used for several corrector iterations at more than one integration time step.

## INTEGRATION STRATEGIES

Dynamic simulators can be structured in either a modular or equation-oriented fashion. Both have advantages and disadvantages. For the process engineer, the modular approach is certainly easier to visualize, because processes are normally viewed as networks of individual processing units and most of the steady-state simulators in current industrial use are structured this way internally. In the remainder of this paper, we address the problem of effective dynamic simulation using a modular approach.

Once a modular simulation framework and numerical integrator have been selected, a strategy for incorporating the integrator into the simulator structure must be adopted. In the past, two general strategies have been used. The first general technique involves treating the ODE system arising from each individual unit model subroutine separately with the numerical integrator (Hlavacek, 1977) (possibly with different integrators for different units). Each piece of equipment gives rise to an individual ODE system which is a subset of the overall ODE system for the plant.

The integration of the ODE subsets is sequenced such that each subset is solved individually over a time horizon using a local time step. If the input variables for the units are fixed at their levels at the beginning of the time horizon interval, this approach is equivalent to decoupling the units completely during the time interval, a mathematically nonrigorous approximation.

In a modification of this approach, the time behavior of stream variables, both information (e.g. for control) and material, can be treated using interpolation in time (Liu and Brosilow, 1987). When material recycle or information feedback (e.g. from a controller) occurs, the simulator must extrapolate in time to estimate at least some missing ODE-dependent variables, so that each ODE subset can be solved over a given time horizon with different time steps (and possibly different integrators). Algorithms must be developed for ODE subset sequencing, interpolation and extrapolation, for checking the accuracy of extrapolated values and for retracing of the integration calculations when extrapolated values are later found to be insufficiently accurate. This approach is certainly sounder mathematically than the completely decoupled one, but is still less rigorous than simultaneous solution of all equations.

The second general strategy for employing a numerical integrator in a modular, dynamic simulation framework involves using the same time step to numerically integrate each ODE subset in sequence, usually in an order determined by the flow of material (the path likely to be followed by a process disturbance). In this way, all ODE subsets are kept at the same time level, and problems with interpolation/extrapolation over more than one time step are avoided. The presence of recycle (either material or information, e.g. a control signal) can complicate this approach, since it may be necessary to iteratively reprocess the entire module calculation sequence to achieve required accuracy. This problem can be minimized by ordering the subsystem integration sequence based on the "tearing" of the cycle (in the sense of modular-sequential simulation) following the unit in the cycle having the longest time constant (Dudczak, 1986).

Each of these techniques has merits. The first approach tends to be favored in situations where some ODE subsets are difficult to integrate (i.e. require relatively small time steps) while other, large subsets are relatively easy to integrate. However, this approach tends to be unfavorable when the plant to be simulated has equipment units that show a high degree of interdependence and are sensitive to the dynamic behavior of one another. The work described here is concerned with the second approach, in which the solutions of the ODE subsets arising from each piece of equipment are kept at the same simulation time level.

### *Traditional modular integration*

One of the criticisms of the modular dynamic simulation structure is that due to the modular structure, where each type of equipment unit in the physical plant has its behavior represented by a procedure implemented as an individual subroutine, simultaneous treatment of the overall equation system is precluded. In other words, because the equation subsystems for computing the derivatives and solving any associated algebraic equations for each plant unit lie in separate subroutines, they cannot be treated collectively.

The structure of most of the early modular dynamic simulators reflects this approach. The simulators are constructed such that the unit model subroutines make calls on the (one or more) numerical integrator. In this sense, the unit model routines "drive" the numerical integrator, and this traditional modular (sequential) structure does preclude simultaneous treatment of the overall equation set.

Consider, for example, a traditional modular simulation of the simple, generic plant shown in Fig. 1.

At each time step, the following procedure would be used:

1. The subroutine modeling the behavior of unit A makes a call to the numerical integrator, providing derivative estimates [**f** in equation (2)] to the integrator. The numerical integrator then calculates estimates of the ODE-dependent variables

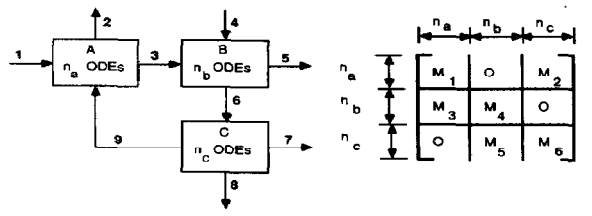


Fig. 1. Schematic of a plant with three units and nine streams. The system Jacobian matrix J is shown on the right; $M$ is a submatrix of J; $M_2$ results from recycle stream 9.

in equation (2) and control returns to the unit model subroutine for unit A.

2. Same as Step 1 with the subroutine associated with unit B making a call to the numerical integrator.
3. Same as Step 1 with the subroutine associated with unit C making a call to the numerical integrator.

*New modular integration approach*

Instead of having the unit model subroutines drive the numerical integrator, it is possible to use the numerical integrator to drive the unit model subroutines. The only known application of this approach in the dynamic simulation context is described by Patterson and Rozsa (1980). In one version of their DYNSYL simulator, the integrator calls directly on single modules corresponding to individual process units in a modular-sequential fashion, gathers system derivatives for all units and then implements simultaneous integration for the full ODE equation set.

The simulator structure developed in this work is organized such that the integrator makes indirect calls on the unit model routines which are individually responsible for evaluating derivatives and solving any associated algebraic equations for that unit. The ODEs and algebraic equations associated with an individual process unit can be solved simultaneously; in addition, ODEs and algebraic equations associated with any "cluster" of units (where a cluster consists of as few as one unit and as many as all units in the plant) can also be solved simultaneously by the integrator.

This strategy is implemented by: (1) creating an interface routine between the integrator and unit routines that is responsible for calling on units within a cluster collectively (a) first to solve all algebraic equations associated with the cluster, and (b) subsequently to evaluate all derivatives associated with ODE dependent variables within the cluster; and (2) modifying a standard integrator (we used a Gear integrator from Lawrence Livermore Laboratory) so that it can perform rigorous simultaneous integration within each cluster, for as many as $n$ individual equation clusters, where $n$ is the number of units in the plant.

As a result, it is possible to accomplish the integration using the traditional sequential approach (one unit assigned to each cluster) or using a simultaneous approach (all units assigned to a single cluster) or a hybrid of these two approaches, which we call the "sequential-clustered" approach. In the sequential-clustered approach, the equation subsystems in each cluster (containing one or more equipment units) are treated simultaneously, with integration remaining sequential in nature from cluster to cluster.

Several strategies for sequencing the integrator itself (for different kinds of integrator/interface calls) within the general structure described in the previous paragraph were postulated and tested as described later.

As an example of how this new modular simulator structure is used, consider simultaneous treatment of the generic plant shown in Fig. 1. On each integration time step (here, assume that there are no algebraic equations containing variables associated with more than one unit) the following procedure would be used:

1. The numerical integrator makes a derivative evaluation request by calling the unit model subroutine interface program.
2. The interface routine calls the unit model routine for unit A and the first $n_a$ derivatives are evaluated. Similar calls are made to the routine for unit B and then for unit C, and the next $n_b$ and last $n_c$ derivatives are evaluated.
3. Once all the derivative calculations have been made, control returns from the interface routine to the numerical integrator.

In this way, all the derivative evaluations are available to the numerical integrator at once, and the overall equation set may be treated simultaneously. A similar procedure is carried out when new estimates of **y** are generated by the numerical integrator. The routine interface program is used to transmit the new y-values to each individual unit model subroutine.

*Comparison of sequential and simultaneous integration*

For sequential integration of the plant shown in Fig. 1, each of the three units are treated in sequence on each time step; the first $n_a$ ODEs are numerically integrated for the time step, then the next $n_b$ and finally the last $n_c$. During the corrector iterations for each piece of equipment, three Jacobian submatrices are used, denoted $M_1$, $M_4$ and $M_6$ in Fig. 1.

For simultaneous integration of the same plant, $n_t$ is equal to $n_a$ plus $n_b$ plus $n_c$. Also, the overall $n_t \times n_t$ Jacobian matrix shown in Fig. 1 is used.

The principal advantage of sequential integration is that smaller Jacobian submatrices are used. The number of individual derivative evaluations needed to generate full submatrices by finite-differencing (perturbation) is equal to $n_a^2 + n_b^2 + n_c^2$. By comparison, generation of a full Jacobian matrix

employed during simultaneous treatment requires $(n_a + n_b + n_c)^2$ individual derivative evaluations.

In addition to the evaluation of the Jacobians, inversion calculations (usually by $LU$ decomposition) are also required. For a full Jacobian, the number of such calculations is proportional to the cube of the matrix order. Thus, the inversion of three small Jacobians will require far less work than for one large one. (This advantage for sequential integration is diminished somewhat for real systems, however, since actual plant Jacobians tend to be sparse, and sparse matrix methods can be used to reduce the number of operations from the theoretical maximum.)

The advantage of simultaneous integration is that by treating the equation set as a whole, no inaccuracies are introduced by recycle streams. For example, for sequential integration of the example plant, when the first $n_a$ ODEs are integrated on each time step, old values for streams $S_9$ must be used because final corrected values will not be available until unit C is integrated on the time step. By comparison, for simultaneous integration, recycle streams pose no special problem because all equations are predicted and corrected as a group.

In summary, the sequential approach is a shortcut technique which requires fewer calculations per time step, but may require small time steps (for a reintegration over the time step to generate "matching" recycle stream parameters) for accurate treatment of recycle. The simultaneous approach is a more rigorous technique, requiring more calculations per time step, but remaining accurate for larger time steps, even when extensive recycle is present. Note that recycle may be either recycle of material or feedback of information (e.g. a controller signal). Direct comparison of simultaneous, sequential-clustered and sequential integration in terms of execution times, storage requirements and accuracy on two test plants is described later on.

## OTHER CONSIDERATIONS

In designing a modular dynamic plant simulator, two other considerations should be taken into account. First, if simultaneous or sequential-clustered integration is used, the resulting Jacobian (sub)-matrices that arise can become quite large and therefore computationally expensive to invert. However, as plants are simulated that give rise to larger equation systems, the resulting Jacobian matrices tend to be more and more sparse. This is because each ODE-dependent variable typically only appears in a few of the many ODEs in the overall ODE system.

As larger and larger plants are simulated, it becomes more important to use sparse matrix techniques for solving the corrector equations at each time step. These sparse inversion techniques can take advantage of the high fraction of zero elements in both generating and inverting the Jacobian matrices. Using sparse matrix techniques, the equations and variables are re-ordered in a way that tends to minimize the number of calculations needed to effectively invert the Jacobian matrix.

Most sparse-matrix inversion techniques are based on the Markowitz criterion (Duff, 1981), which selects the pivots for Gaussian elimination primarily on the criterion of reducing the number of fill elements (nonzero elements generated where zero elements occur in the starting matrix).

Sparse matrix generation techniques (for finite-difference determination of system Jacobian matrix elements) are also very useful and involve a two-step process: (1) the full matrix is initially generated, column by column, and relationships between variables and functions (which variables affect which functions) are noted; this information is used to determine groupings of variables that do not affect the same functions so that several variables may be perturbed together; and (2) on subsequent evaluations of the matrix, the variable-groupings are employed.

As a simple example of this technique, note that for a (tridiagonal system of equations, where variable $i$ affects only functions (derivatives) $i-1$, $i$ and $i+1$, full Jacobian matrix generation by finite-differencing (perturbation) requires evaluation of each of the $n$ functions $n$ times, for a total of $n^2+1$ evaluations. If a sparse generation technique is used, only four evaluations of the $n$ functions would be needed. The three groupings would be: variables $(1, 4, 7, \ldots)$, $(2, 5, 8, \ldots)$ and $(3, 6, 9, \ldots)$.

Another important consideration is the possibility of using shortcut techniques to reduce the amount of execution time spent estimating physical properties. For plant simulations of this type, we have found that these estimation calculations may consume 80–90% of the total execution time. Obviously, there is great incentive to try to reduce this percentage.

Use of shortcut techniques may be regarded as a direct method of reducing physical property costs. Indirect techniques, such as employing efficient numerical algorithms that complete the integration with as few time steps (and therefore as few physical property evaluations) as possible may prove to be just as effective. With the indirect approaches, the fraction of the total execution time spent estimating physical properties is not altered appreciably, but the total execution time is reduced.

## SOLUTION STRATEGIES FOR ODEs

We selected Gear's BDF (backward difference formula) methods as the most suitable integrators for the various stiff systems encountered in many chemical processes. Having chosen the integrator, we next attempted to find a suitable algorithm for incorporating these linear, multistep predictor–corrector algorithms into a simulator structure that

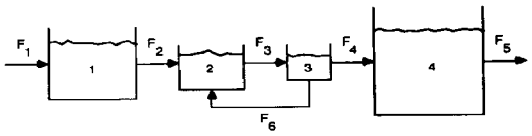


Fig. 2. Four well-mixed tanks in series with recycle.

would allow for sequential, sequential-clustered or simultaneous integration strategies.

The objective of our preliminary testing was to determine a single robust and accurate sequential integration strategy for Gear's methods (since sequential integration will place the greatest strain on method accuracy for systems containing recycle) and then to incorporate this strategy into a prototype chemical plant simulator. From the accuracy standpoint (particularly for processes containing material recycle or information feedback), it is clearly preferable to implement the predictor equations for all clusters before implementing the corrector equations for any clusters, and this strategy was used in all our tests.

For preliminary screening of algorithms, the simplified plant shown in Fig. 2 was chosen; it consists of four well-mixed tanks in series with an internal recycle stream. The tank sizes were chosen such that the governing ODE set had a stiffness ratio of $5 \times 10^6$.

The simplified plant was broken into two integration blocks or "clusters": the first two tanks were assigned to the first cluster, and the remaining two tanks to the second cluster. Also, the composition of the feed stream to the plant was perturbed at a specified time with a step change (discrete jump), to determine the suitability of the Gear integrator for handling discontinuities.

In all, five sequential integration schemes were tested. In each case, the integration of equations in cluster 1 preceded those in cluster 2 for a given time step; the predicted values of the recycle stream (stream 6) from the second cluster back to the first were used in the first block's corrector equations (implementing the predictor and corrector equations over both clusters as described above).

Other features of five sequential schemes are:

1. Predicted values of stream 3 are used in the second cluster's corrector equations.
2. Predicted values of stream 3 from the first cluster are used in the second cluster's corrector equations on the first corrector iteration and corrected values are used thereafter.
3. Corrected values of stream 3 from the first cluster are used in the corrector equations for the second cluster.
4. Same as Scheme 2 except that a final corrector iteration is forced for all equations in each cluster after the corrector cycle for each time step is completed.

Table 1. Results for four-tank problem of Fig. 2, for integration to time 200 h. $e$ = normalized r.m.s. truncation error tolerance, sim = simultaneous, seq = sequential, $n_s$ = number of time steps, DE = number of derivative evaluations, % increase = % increase in derivative evaluations beyond requirement for simultaneous integration

| $e$ | Method | $n_s$ | DE | Increase (%) |
|---|---|---|---|---|
| 0.001 | sim | 151 | 6754 | |
| 0.001 | seq-1 | 977 | 35,988 | 432.8 |
| 0.001 | seq-2 | 359 | 11,263 | 66.8 |
| 0.001 | seq-3 | 231 | 7532 | 11.5 |
| 0.001 | seq-4 | 207 | 8691 | 28.7 |
| 0.001 | seq-5 | 172 | 7185 | 6.4 |
| 0.01 | sim | 98 | 4884 | |
| 0.01 | seq-4 | 128 | 5305 | 8.6 |
| 0.01 | seq-5 | 112 | 5085 | 4.1 |
| 0.1 | sim | 63 | 3861 | |
| 0.1 | seq-4 | 87 | 3873 | 0.3 |
| 0.1 | seq-5 | 66 | 3873 | 0.3 |

5. Same as Scheme 3 except that a final corrector iteration is forced for all equations in each cluster after the corrector cycle for each time step is completed.

These five strategies are referred to as "seq-1" to "seq-5" in Table 1. Simultaneous (single-cluster) integration of this plant was also carried out. The results of these various tests are summarized in Table 1 for various values of $e$, the user-specified r.m.s. truncation error tolerance used by the Gear integrator.

From Table 1, we see that in each case the simultaneous scheme requires the fewest number of time steps to complete the integration. This is expected since it is the most rigorous approach. Also, note that sequential Scheme 5 fares the best of the sequential schemes tested. This scheme was subsequently adopted as the single sequential algorithm used in the prototype MUNCHEPS (Modular Unsteady-State Chemical Engineering Process Simulator) described in the rest of the paper.

To review the principal features of the simulator integrator:

1. Predictors are used for all equipment units or clusters of equipment units prior to initiating corrector iterations.
2. Corrected values are used as soon as they are available.
3. One extra corrector iteration is forced at the end of each time step.

While this third feature results in more derivative evaluations per time step (an extra number equal to the total number of ODEs in the process), the integration is completed in fewer (larger) time steps, with net computational savings. Forcing this extra corrector iteration and performing all predictor steps prior to beginning the iterative corrector steps helps very significantly to overcome the major weakness of sequential integration, inaccurate treatment of information and material recycle streams for large time steps.

Other conclusions drawn from this initial testing are:

1. The Gear integrator has little trouble in dealing with step change forcing functions. The algorithm automatically reduces the step size and resorts to a method order of one in the neighborhood of the step change, making it a single-step (backward Euler) method at the discontinuity.
2. As expected, introduction of recycle and stricter error tolerances tends to favor use of the more rigorous simultaneous approach over the sequential-clustered approach.
3. A sequential approach, with predicted values generated before beginning any corrector iterations, and an extra forced corrector iteration results in an 80% savings in execution time compared with a less sophisticated straightforward sequential integration approach (for comparable accuracy).

Comparison of simultaneous integration and sequential integration Scheme 5 on other plant units will be described later.

## SOLUTION OF PDEs

When temperature or concentration gradients within an equipment unit are important and the well-mixed assumption does not apply, PDEs govern. It is possible, using the method of lines, to reduce the governing PDE set to a larger ODE set, a process involving discretization of the space variable with a finite-difference grid. Partial derivatives with respect to spatial coordinates are then replaced with finite difference approximations, and partial derivatives with respect to time become ordinary derivatives with respect to time at each grid point.

As an example, consider the method-of-lines treatment of a double pipe heat exchanger model, with $n$ finite-difference grid points in the axial direction (and radial variations neglected) (Fagley, 1984). Originally, in the PDE representation, the temperature of fluid a is a single function of time and distance along the exchanger. After method-of-lines treatment, the temperature of fluid a becomes $n$ functions of time only. Similarly, the partial derivative of enthalpy with respect to axial distance becomes $n$ finite-difference approximations. The correspondence between the original PDE quantities and the method-of-lines quantities is shown in Table 2. A backward finite-difference expression is used for the enthalpy derivative, because this term appears in a bulk flow term, and only upstream enthalpies will have an effect (Haydweiller *et al.*, 1977).

As the number of finite-difference grid points used to represent the spatial coordinates increases, the accuracy of the finite-difference approximation of the spatial derivatives improves, but at the cost of larger ODE set sizes and increasing stiffness (Byrne and Hindmarsh, 1977).

Table 2. Correspondence between original PDE quantities and method-of-lines ODE quantities with $n$ method-of-lines segments

| Original PDE quantity | Method-of-lines ODE quantity | |
|---|---|---|
| Axial distance $x$, $0 < x < L$ | $x_i$ | $i = 1, 2, \ldots, n$ |
| 3 PDEs (heat balance, fluids a and b and metal) | $3n$ ODEs | |
| $T_a(x, t)$ | $T_{a,i}(t)$ | $i = 1, 2, \ldots, n$ |
| $\frac{\partial T_a}{\partial t}$ | $\frac{dT_{a,i}}{dt}$ | $i = 1, 2, \ldots, n$ |
| Enthalpy gradient, $\frac{\partial H}{\partial X}$ | $\frac{3H_i - 4H_{i-1} + H_{i-2}}{2(DX)}$ | $i = 1, 2, \ldots, n$ |

$DX$ = total length $L$ divided by $n$.

For the case of method-of-lines treatment of an equipment unit arising in a chemical plant simulation, the accuracy of the finite-difference representation should be in line with the accuracy requirements for the numerical integrator. For example, if the single-step r.m.s. truncation error tolerance specified to the numerical integrator corresponds to 0.1°F, then the error associated with the finite-difference approximation should be of the order of 0.01–0.05°F.

Part of the testing of the prototype MUNCHEPS simulator involved demonstrating the ability to incorporate a method-of-lines equipment unit into a general, modular, unsteady-state simulator framework. A double-pipe heat exchanger model was selected for this purpose. The system chosen for testing involved countercurrent flow and exchange of heat between a hydrocarbon stream and a cooling water stream. The details of the heat exchanger model formulation and testing are given in Fagley (1984) and the results of this testing are summarized in Table 3. Statistics are for execution on an Amdahl 5860 mainframe with a scalar processing capacity of about 6 mips; for comparison, a current scalar IBM 3090 processor operates at about 15 mips.

As shown in Table 3, the accuracy in the water and hydrocarbon exit temperatures improves with an increasing number of method-of-lines sections. The cost of this improved accuracy is an increase in both execution time and storage requirements. Also, the stiffness ratio increases with the number of method-of-lines sections. The initial stiffness ratio for the system (defined here as the final time of simulation divided by the inverse of the largest negative

Table 3. Statistics for testing method-of-lines heat exchanger model. $N_s$ = number of sections, integrator r.m.s. error tolerance $e = 0.001$ in all cases. Accuracy for water and hydrocarbon temperatures is the time-averaged error, assuming solution for $N_s = 20$ to be exact

| | | Accuracy (°R) | | |
|---|---|---|---|---|
| $N_s$ | CPU time(s) | Water | Hydrocarbon | Storage (kbyte) |
| 5 | 4.26 | $q$ 2.5 | $q$ 2.0 | 193.8 |
| 10 | 6.69 | $q$ 0.92 | $q$ 0.74 | 212.9 |
| 15 | 13.56 | $q$ 0.35 | $q$ 0.23 | 236.8 |
| 20 | 14.18 | — | — | 269.9 |

real eigenvalue part) increased from 160 for five method-of-lines sections to 840 for 20 method-of-lines sections, meaning that this is a moderately stiff equation set.

Note that as the number of sections increases from five to 20, the ODE set size increases from 15 to 60, and the number of elements in the full system Jacobian matrix increases by a factor of 16. However, we see from Table 3 that the storage requirements increase by only 40%. This is because the storage space needed for the machine code associated with simulator and accompanying utility routines (hereafter referred to as background storage) is larger than the storage space needed for storing numerical integrator quantities. It should be noted, however, that as larger plants are simulated, these background storage requirements will become relatively less and less significant; numerical integrator quantities will dominate demand for fast memory.

Another interesting point is brought out by close inspection of this table. For 15 method-of-lines sections, the execution time is unexpectedly high—only 4% smaller than that required for 20 sections. This can be explained by the fact that for the run with 15 sections, the integrator required that three Jacobian matric evaluations be made, while only two were required for the other three cases. This type of behavior is typical for a numerical integrator. Time statistics (and the number of derivative evaluations and Jacobian matrix evaluations) for a family of parameter values are somewhat stochastic in nature. A single set of values such as those shown in Table 3 are "point estimates" and are useful for showing trends rather than making fixed absolute value predictions. For example, a miniscule increase in the error tolerance might well have allowed the integrator to proceed with just two rather than three Jacobian matrix evaluations in the 15-section case, changing the statistics for that case significantly.

The conclusions from these studies are: (1) it is not particularly difficult to include method-of-lines unit model routines in a general modular dynamic chemical plant simulator, such as the MUNCHEPS simulator; and (2) the optimal number of method-of-lines sections to be used depends on the desired accuracy of numerical integration of the resulting ODE set. A tradeoff exists between accuracy and CPU time/storage requirements. For the heat exchanger we considered, 10–15 sections results in a fair tradeoff.

## SIMULATOR TESTING

The prototype research simulator MUNCHEPS was developed primarily to determine the feasibility of performing simultaneous and sequential-clustered integration in a modular simulation framework. Subsequently, the simulator was tested on example chemical plants to compare these competing strategies in terms of execution times, storage requirements and accuracies. Some of the noteworthy features of the simulator are listed below.

### *Plex data structures*

The MUNCHEPS simulator makes extensive use of plex structures (Evans *et al.*, 1977). Plex structures are used to store equipment parameters, stream parameters, physical property constants and 1- and 2-D arrays that are associated with the Gear integrator. Plex structures have several advantages over conventional matrix structures: (1) more efficient use of storage; (2) savings in physical property system computations by logical classification of streams; and (3) greater flexibility in simulating a variety of plants, both in terms of defining new types of stream parameters and constructing different types of equipment unit model routines. The plex data structures are also quite compatible with the MUNCHEPS preprocessor, described next.

### *Preprocessor*

Simulator input data are entered through a preprocessor. The preprocessor is executed separately from and prior to the simulator itself. The preprocessor is used to read in information about the layout of the process and the number and kinds of chemical species present. During execution of the preprocessor, the user also specifies information regarding numerical integration, e.g. r.m.s. single-step error tolerance, initial conditions and integration strategy (simultaneous, sequential-clustered or total sequential). The preprocessor uses this information to set up the various plex data structures, including determination of pointer values and total plex vector lengths. The preprocessor then writes a small main program (in FORTRAN-77) for the simulation, where plex vectors are dimensioned to exactly the length needed for the given simulation. Thus the simulation uses only that storage which is needed by the particular process plant and dynamic memory allocation (unsupported in ANSI FORTRAN-77) is unnecessary.

### *Sparse matrix decomposition*

The MUNCHEPS simulator makes use of a sparse matrix $LU$ decomposition procedure which is based on the Markowitz criterion for reducing decomposition operations (Duff, 1981). This technique is particularly useful for treatment of the Jacobian (sub)matrices that arise. The first time a Jacobian matrix is generated, the sparse matrix algorithm is used to determine a strategy for performing its $LU$ decomposition by Gaussian elimination. This sparse strategy is then recorded in integer code lists.

The decomposed matrix is used to process new right-hand-side vectors on each corrector iteration for each time step. Subsequently, as new instances of the Jacobian matrix are encountered every several time steps, the integer code strings are used to decompose the matrix. This feature allows for

significant time savings, since the decomposition strategy need be determined only once in many cases. However, the MUNCHEPS sparse matrix solver (called SLUD) allows for re-determination of the decomposition strategy should the structure of the matrix change (as it does sometimes for nonlinear ODEs) or should a Gaussian pivot element become too small in magnitude. The current version of the MUNCHEPS simulator does not make use of sparse matrix generation techniques involving grouped variables described earlier, although such a feature is certainly desirable, especially for simultaneous and sequential-clustered simulation of large plants.

*Simulator structure*

The simulator was designed so that the numerical integrator "drives" the unit model routines indirectly through an interface routine. This allows for sequential-clustered and simultaneous integration, where the ODEs from different equipment pieces may be treated together in spite of the modularity of the routines generating the essential derivative functions.

Figure 3 shows the basic block structure of the MUNCHEPS simulator. The arrows show the routine calling procedure that is used when new values of the ODE-dependent y are to be stored in the stream and equipment data structures. Values of y are calculated by the Gear numerical integration package, denoted GEAR. GEAR then calls the unit model routines with the new values of y through the unit model routine calling program ROCALL. The unit model routines in turn make calls on the physical property system through the interface routine PROP. Also, the routines GET and PUT are used to store and retrieve numerical values from the plex stream data structure.

After all values of y have been stored and associated physical property estimates have been generated, control returns back to the GEAR routine and then to the supervisory routine SUPER. Also shown in Fig. 3 is the sparse matrix solver package SLUD, which is called by the GEAR integrator when it is necessary to decompose a new Jacobian matrix, and during each corrector iteration. The main program,

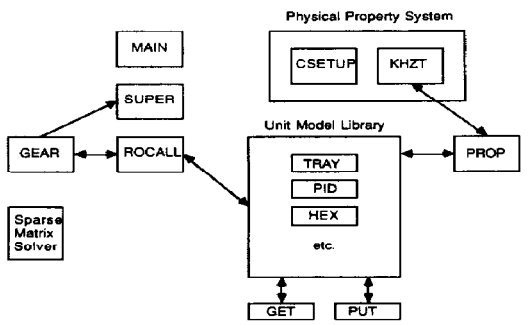


Fig. 3. Routine calling procedure for storing y values after a corrector iteration or for storing perturbed y values during Jacobian element calculation.

written by the preprocessor, is also shown. Further details of the routine calling procedure for various types of operations carried out by the MUNCHEPS simulator and details of the data and routine structures are given in Fagley (1984).

The critical design feature of the simulator architecture shown in Fig. 3 is that the numerical integrator "drives" the unit model routines by calling them through the calling program ROCALL. It is this feature of the program structure that allows for simultaneous and sequential-clustered integration in the modular simulation framework.

Development of the procedure by which the numerical integrator drives the unit model routines involved very extensive modification of the Gear integration software (acquired from Livermore Laboratories); in its original form, the software is capable of integrating just one ODE set. The package was altered so that $N_c$ (potentially as large as the number of units in a plant) clusters of ODE sets could be handled. Modifications to the original code were also required to allow for implementing the predictor equations for all clusters prior to beginning corrector equation iteration for any clusters and for forcing an extra corrector iteration at the end of each successful integration step.

The automatic stepsize and method-order logic was also altered so that multiple sets of ODEs could be handled. In the original Gear code, the allowable time steps of integration that meet the specified user-supplied error tolerances are estimated for orders $q+1$, $q$ and $q-1$ every $(q+1)$th time step; here $q$ is the method order being used on the current step. The integrator then chooses the largest of these three estimated time steps for its next stepsize, and switches (if necessary) the order of the method accordingly.

This order and stepsize-changing logic was altered as follows for the MUNCHEPS simulator. Every $(q+1)$th time step an acceptable time step of integration is estimated for each ODE set (i.e. for each of the $N_c$ clusters) for each of three method orders, $q+1$, $q$ and $q-1$. For each of these three orders, a minimum acceptable integration time step (taken over all $N_c$ ODE sets) is found. The maximum of these three minimum time steps is then selected as the next stepsize; the method order is changed, if necessary, to that producing the selected stepsize.

We also found after extensive testing that integrator performance could be improved significantly (see Table 9) by modifying the Jacobian matrix updating strategy used in the original Livermore software. Originally, a new Jacobian matrix calculation was triggered whenever the quantity $hc_0$ [see equation (9)] changed by more than 30% as a result of stepsize and order changes. We modified this criterion as follows:

1. The Jacobian matrix J is updated only when the corrector equations fail to converge.

2. Whenever $hc_0$ changes by more than 2%, the nonzero elements of the matrix $hc_0$J are rescaled, and a new sparse matrix $LU$ decomposition is performed using the currently saved calculation strategy.

Plex data structures were also incorporated into the Gear integrator to facilitate storage of numerical integrator quantities for clustered sets of ODEs. One feature of the original Gear integrator is that only one type of call is made to the derivative evaluation routine, independent of whether this call is made during a predictor step, a corrector step, a call to evaluate derivatives for finite-difference Jacobian matrix determination, a call after an unsuccessful attempted step, etc. In the type of dynamic simulation described here, it is important to distinguish among these and other types of calls because coupled algebraic equations are involved, some of which involve running sums.

For example, the integral portion of the action taken by a PID controller involves numerical estimation of the time integral in the offset of a sensed process variable. In order to be able to estimate this running sum, it is important that information be passed to the unit model routine for the PID controller which indicates what type of call it is. For instance, if this call represents a normal continuation of integration, a term is added onto the running sum. If this is the first call after an unsuccessful time step, a term must be subtracted from the running sum, and so on. Modifications were made to the Gear integrator to allow for the passage of this type of information through an integer code in the calling argument list to the interface routine ROCALL.

Another feature of dynamic simulation of this sort is that the governing ODE sets are coupled with algebraic equation (AE) sets. In the most general case, unit model routines can contain algebraic equations which include variables calculated by other unit model routines. It would then be possible to formulate simulation problems with an algebraic equation set, characteristic of steady-state chemical plant simulation, "imbedded" inside the dynamic problem. Rigorous solution would then involve convergence of the overall algebraic equation set (similar to the steady-state simulation case) during each corrector iteration for numerical solution of the overlying ODE description.

In the simplest (and most common) case all algebraic equations coupled with the ODE set for a unit model routine only contain variables that appear in that unit routine model so that algebraic equation solution can be accomplished internal to the unit model routine.

For the formulation of the coupled ODE and AE set description of units encountered during development of the MUNCHEPS simulator, some cases were encountered where AE and ODE variables of interest in one routine were indeed calculated in another routine. For example, determination of component balances on an individual tray in a distillation column involve both liquid flows down from the tray above and vapor flows upward from the tray below.

It was found that by using a two-round calling procedure, these types of problems could be overcome. For instance, suppose a new value of the ODE-dependent variables are determined at a given time level. A first series of calls is then made to all tray routines, and all vapor flows and liquid flows are determined. Subsequently, a second round of calls is made during which derivatives are determined. Since all liquid and vapor flows have been determined during the first round of calls, the derivatives may now be determined accurately. Details of this two-round calling procedure are given in Fagley (1984).

Another example involves control of liquid level in a mixing tank. Calculation for material balance ODEs for the tank requires that the outlet flow rate be known, which in turn requires a known controller setting, which in turn relies on liquid level in the tank. On the first round of calls to the mixing tank–controller subprocess, the liquid level (an algebraic variable that depends on the current values of the molar holdup, composition, density and physical dimensions of the mixing tank) is computed by the tank model and a controller setting is determined by the controller model, given the current liquid level. On the second round of calls, the correct controller setting, and therefore the correct outlet flowrate, are available for accurate derivative evaluation by the tank model.

## DESCRIPTION OF TEST PLANTS

We include here tests for two example plants simulated with MUNCHEPS. The first plant is the controlled distillation system shown in Fig. 4. The accompanying MUNCHEPS topology map is shown in Fig. 5. This distillation system is used to separate a mixture of benzene, toluene and *o*-xylene. The control scheme used is a material-balance control scheme suggested by Shinskey (1977). Results shown for this plant are for a total simulation time of 25 min with a perturbation in the feed rate occurring at a simulation time of 3 min.

This controlled fractionation plant may be described as an underdamped oscillatory system with a period of roughly 10 min. Each TRAY unit routine model contains four ODEs, one for each component (chemical species) and one for total liquid holdup. The reboiler routine model contains five ODEs, one for each component, one for liquid holdup and one for the metal temperature in the reboiler (the reboiler routine includes the reboiler heat exchanger and column bottoms). The overhead condenser routine model contains five ODEs, one for each component, one for the total liquid holdup and one for the enthalpy of the liquid in the overhead condenser

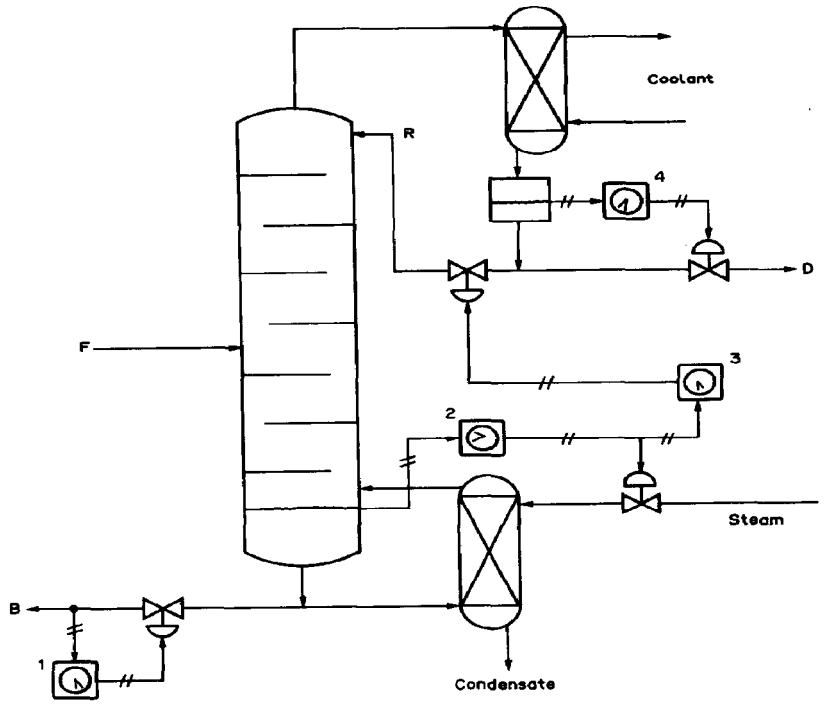


Fig. 4. Layout of a seven-tray distillation column. Four PID controllers, denoted 1 to 4, are employed. Hashed lines denote information (controller signal) streams.

accumulator. Details of the modeling equations (with a total of 38 ODEs) used to describe the dynamic behavior of each piece of equipment, and further discussion of the control strategy and system response, are given in Fagley (1984).

Results are also shown for a second plant, consisting of the seven-tray column shown in Fig. 4 with a double-pipe heat exchanger added to exchange heat between the cold feed entering the fractionation system and the hot liquid leaving the column bottoms. For this application, 10 method-of-lines sections were selected for the heat exchanger, so that the overall system size was 68 ODEs, with 38 ODEs for the column and 30 for the heat exchanger. As before, the results shown are for the plant simulated up to a simulation time of 25 min, with a feed rate perturbation introduced at a time of 3 min.

## TEST RESULTS

The seven-tray plant of Fig. 4 was simulated using three different equipment grouping strategies: simultaneous (one integration block with 38 ODEs); sequential-clustered (the reboiler and bottom three trays in one integration block and the condenser and top four keys in a second); and total sequential with nine integration blocks (one for each tray, one for the condenser and one for the reboiler).

Table 4 gives a summary results for these tests. Execution times are for the Amdahl 5860 computer with a single scalar processor. Results for a simultaneous integration with a user-specified, normalized, r.m.s. truncation error tolerance of 0.0005 was assumed to be the "exact" solution. "Error" refers to the time-averaged absolute value of the normalized error in the parameter values for the two outlet streams (distillate and bottoms product): total

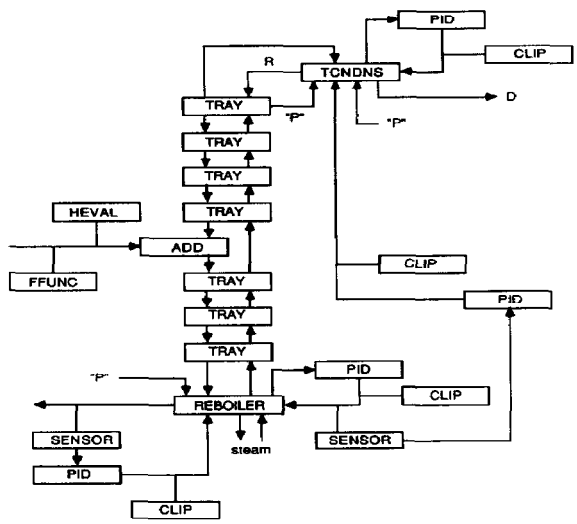


Fig. 5. MUNCHEPS topology of plant shown in Fig. 4. Each box represents a unit model routine. Streams denoted *P* are information streams containing pressures, and are used to determine flowrates through valves.

Table 4. Results for seven-tray plant of Fig. 4. sim, seq-clu, and seq refer to simultaneous, sequential-clustered and sequential integration, respectively

| $e$ | Method | Error (%) | CPU time (s) | Storage (kbyte) |
|---|---|---|---|---|
| 0.001 | sim | 0.030 | 14.6 | 286 |
| | seq-clu | 0.10 | 12.3 | 228 |
| | seq | 0.063 | 23.0 | 205 |
| 0.01 | sim | 0.42 | 4.85 | 286 |
| | seq-clu | 0.36 | 4.80 | 228 |
| | seq | 0.79 | 16.1 | 205 |
| 0.1 | sim | 0.78 | 3.67 | 286 |
| | seq-clu | 1.8 | 3.45 | 228 |
| | seq | 2.7 | 11.9 | 205 |

molar flow rate, temperature, enthalpy and mole fractions.

From Table 4, we see that:

1. Simultaneous integration tends to be a little more accurate than the other two techniques for a given value of error tolerance $e$.
2. Simultaneous and sequential-clustered integration are roughly equivalent to each other and superior to sequential integration in terms of execution time.
3. The simultaneous integration scheme requires the most storage.

The latter observation is to be expected, since storage for a $38 \times 38$ Jacobian matrix (and other associated integrator arrays) must be assigned by the preprocessor; much smaller Jacobian submatrices are used in the other two cases. The fact that the simultaneous scheme requires only 40% more storage than the sequential scheme indicates that a large portion of storage is being used to store the simulator object code. For larger plants, however, the difference in storage requirements between simultaneous and sequential integration will be much more pronounced.

In view of the results shown in Table 4 for this plant, either simultaneous or sequential-clustered integration would be preferred, giving better accuracy at less cost than for sequential integration. Figure 6

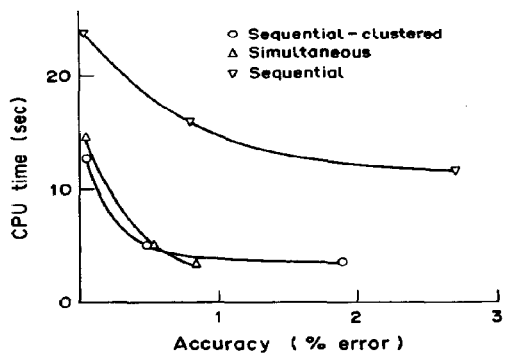


Fig. 6. CPU time vs accuracy (percentage error) for sequential, sequential-clustered and simultaneous integration for the plant of Fig. 4.

Table 5. Integration statistics for seven-tray column of Fig. 4 with $e = 0.01$. $n$ = number of time steps, DE = number of derivative evaluations, JE = number of Jacobian matrix evaluations for each cluster in order

| | Method | | |
|---|---|---|---|
| | sim | seq-clu | seq |
| $n_s$ | 32 | 34 | 114 |
| DE | 3344 | 4417 | 13, 532 |
| JE | 1 | 1, 2 | 2, 18, 17, 16, 16, 15, 14, 14, 14 |
| % DE for JE | 43 | 27 | 19 |
| CPU time (s) | 4.85 | 4.80 | 16.1 |

illustrates the tradeoff between cost (CPU time) and accuracy for the results shown in Table 4.

Table 5 shows final numerical integrator statistics for the three integration strategies with an error tolerance $e$ of 0.01.

Notice that simultaneous integration, being the most rigorous technique, requires the fewest number of time steps to complete the simulation, though the sequential-clustered method requires only a few more. The cost of evaluating the Jacobian (sub)-matrices is also clearly indicated in this table. Even though simultaneous integration requires only one Jacobian matrix evaluation, the evaluation of the elements in the relatively large $38 \times 38$ matrix requires 43% of the total individual derivative evaluations.

By comparison, the sequential scheme requires its nine Jacobian submatrices to be evaluated an average of 16 times. However, these submatrices are relatively small ($4 \times 4$ and $5 \times 5$) and evaluation of the Jacobian elements by perturbation requires only 19% of the total individual derivative evaluations. Note that for this plant, the sum of the squares of the number of ODEs in each piece of equipment (the total number of elements in all nine Jacobians) is one-ninth the square of the sum (the number of elements in the Jacobian for simultaneous integration).

The second plant tested was the distillation system of Fig. 4 with heat integration between the hot bottoms stream and the cold incoming feed. Three equipment-grouping strategies were used:

1. Simultaneous—one integration block with 68 ODEs.
2. Sequential—10 integration blocks: one for the feed heat exchanger (30 ODEs), one for the reboiler and one for the overhead condenser (5 ODEs each) and one for each of the seven trays (four ODEs each).
3. Sequential-clustered—three equipment clusters, one for the feed-bottoms heat exchanger (30 ODEs), one for the reboiler and bottom three trays (17 ODEs) and one for the overhead condenser and top four trays (21 ODEs).

The plant was tested with a normalized single-step truncation error tolerance of 0.01 specified for the numerical integrator. The results are summarized in Table 6.

Table 6. Integration statistics for seven tray column of Fig. 4 with feed heat exchanger. $e = 0.01$, $n$ = number of time steps, DE = number of derivative evaluations, JE = number of Jacobian matrix evaluations for each cluster in order

| | Method | | |
|---|---|---|---|
| | sim | seq-clu | seq |
| $n_s$ | 38 | 36 | 89 |
| DE | 13,464 | 8687 | 18,184 |
| JE | 2 | 1, 1, 1 | 3, 3, . . . , 3, 3 |
| % DE for JE | 68.7 | 20 | 21.5 |
| Storage (kbyte) | 362 | 290 | 236 |
| CPU time (s) | 10.2 | 7.34 | 20.2 |

Note that the sequential-clustered scheme is particularly efficient for this problem, requiring 38% less CPU time than the simultaneous run, and 64% less CPU time than the sequential run. As before, the sequential-clustered approach makes good use of both execution time and storage. However, the benefits of sequential-clustered relative to simultaneous integration are somewhat exaggerated here because, on average, we would expect the more rigorous simultaneous approach to require fewer Jacobian matrix evaluations.

Table 6 shows once again that the cost of evaluating the large (68 × 68) Jacobian matrix is quite expensive in the simultaneous case, requiring over two-thirds of the individual derivative evaluations. The large size of this Jacobian matrix is reflected in the total storage requirements. The simultaneous scheme requires an additional 53% memory and the sequential-clustered approach an additional 23%, as compared to the sequential case.

The sequential scheme is particularly inefficient in terms of execution time. This is due to the high degree of recycle internal to the column. However, from these results we may also conclude that the degree of coupling is not great enough to demand simultaneous treatment of the entire column and heat exchanger (the results for all cases differ by less than 1%).

Table 7 shows the CPU time breakdown for simultaneous and sequential integration of the distillation test plant shown in Table 5. Note that in both cases, most execution time is spent in computing physical properties. We used a proprietary property package supplied by Professor Motard of Washington University of St Louis for test purposes; although a new property system interface program was written for the MUNCHEPS simulator, the property routines themselves were not modified.

Table 7. Breakdown (in percent) of CPU time for the simultaneous and sequential integration cases shown in Table 5

| | Simultaneous (%) | Sequential (%) |
|---|---|---|
| Physical properties | 79.4 | 82.9 |
| Unit model routine calculations | 5.7 | 4.7 |
| Plex structure storage/retrieval | 4.2 | 4.3 |
| Sparse $LU$ strategy/decomposition | 3.9 | 0.8 |
| Sparse $LU$ solution | 3.5 | 3.0 |
| Gear integrator | 1.6 | 2.2 |
| Supervisory routines | 1.7 | 2.1 |
| Total | 100.0 | 100.0 |

Table 8. Time statistics for sparse and full $LU$ decomposition and equation solution for simultaneous (38 × 38 system) and sequential (two 5 × 5 and seven 4 × 4 subsystems) integration of the seven-tray plant up to a simulation time of 3 min. Results are for simulation of seven-tray column with error tolerance $e = 0.01$

| Simultaneous integration | CPU time (s) | Savings (%) |
|---|---|---|
| Full decomposition | 0.500 | |
| Sparse decomposition | 0.156 | 68.8 |
| Full solution | 0.206 | |
| Sparse solution | 0.140 | 32.0 |
| Total full | 0.706 | |
| Total sparse | 0.296 | 58.5 |
| Sequential integration | | |
| Full decomposition | 0.0411 | |
| Sparse decomposition | 0.0320 | 22.1 |
| Full solution | 0.189 | |
| Sparse solution | 0.121 | 36.0 |
| Total full | 0.230 | |
| Total sparse | 0.153 | 33.5 |

There are two approaches one might take to improve overall execution times. One is to reduce the amount of calculation needed for each physical property evaluation. This would require modification or replacement of the property routines or some shortcut approach such as interpolation or extrapolation of infrequently computed property values, and would reduce the percentage of execution time spent in the physical properties routines.

Another way of decreasing overall execution time is to use more efficient numerical integration techniques so that a given simulation is accomplished in fewer steps and, consequently, with the need for fewer physical property evaluations. This second method would not necessarily show a change in execution-time breakdown percentages.

Table 8 shows the relative advantage of using sparse matrix decomposition techniques. Note that for the small Jacobian submatrices used in sequential integration, the savings in matrix decomposition and solution times are relatively small. By contrast, for the larger and more sparse Jacobian matrix used in simultaneous integration, more pronounced savings in execution times are realized with the sparse technique. As shown here, using a sparse technique for this 38 × 38 matrix results in a 58.5% saving in execution time. For larger systems, greater savings would be realized.

Table 9 summarizes one final set of numerical experiments performed on the seven-tray distillation plant. The strategy used in the original GEAR software involves updating the Jacobian matrix whenever the corrector equations fail to converge or whenever the quantity $hc_0$ [see equation (9)] changes by more than 30%. Results using this strategy are reported as "Orig" in the table.

After experimentation with other Jacobian matrix recalculation strategies, we adopted a somewhat different approach and modified the integration software accordingly. The Jacobian matrix is updated only when the corrector equations fail to converge. Whenever the quantity $hc_0$ changes by

Table 9. Execution time statistics and storage requirements for three different Jacobian matrix updating strategies. Orig cases use the original Gear (Livermore) software heuristic (see text), New cases use the modified updating criterion (see text), Diag cases use the original heuristic with a diagonal Jacobian matrix approximation only. Results are for simulation of seven-tray column with error tolerance $e = 0.01$

| Method | Run | CPU time (s) | Increase (%) | Storage (kbyte) |
|---|---|---|---|---|
| Simultaneous | New | 4.85 | | 286 |
| | Orig | 11.4 | 135 | 286 |
| | Diag | 132 | 2620 | 274 |
| Sequential | New | 16.1 | | 205 |
| | Orig | 33.0 | 105 | 205 |
| | Diag | 76.0 | 372 | 204 |

more than 2%, the nonzero elements of the matrix $hc_0$J are rescaled (note that the Jacobian elements themselves need not be recalculated) and the rescaled matrix is decomposed. The results of these tests (shown as "New" in the table) illustrate that the new strategy is clearly superior, especially for simultaneous integration (the "Orig" and "New" run results are of comparable accuracy). Effectively the new strategy allows a calculated Jacobian matrix to be used over more steps (because of the rescaling) without sacrificing accuracy, and reduces the number of Jacobian evaluations required for the complete integration.

Table 9 also includes results for cases where a diagonal approximation to the Jacobian matrix is used (labelled "Diag"). While this strategy results in some storage savings, there is a heavy penalty in terms of increased execution time. It should be mentioned here that all other results presented in this paper have been obtained using the new Jacobian matrix updating strategy.

## CONCLUSIONS

Design, construction and testing of a prototype modular, dynamic, chemical plant simulator capable of simulating plants that give rise to large, stiff equation sets has revealed several features important to this type of simulation. First, total sequential, sequential-clustered and simultaneous integration in a single modular simulation framework has been demonstrated. Simultaneous treatment of two or more individual unit model routines is made possible through use of a novel simulator structure in which the numerical integrator "drives" the unit model routines indirectly through an interface; the integrator incorporates features to allow for solution of many potentially large ODE systems.

Subsequent testing of the simulator showed that a well-chosen sequential-clustered scheme is often the optimal choice, giving good execution time response with moderate storage requirements and acceptable accuracy. The sequential-clustered approach allows for simultaneous treatment of units that display a strong interdependence, while integration remains sequential from cluster to cluster.

Testing of the simulator has also demonstrated the benefits of sparse matrix solution techniques and the benefits of a new Jacobian matrix updating strategy. A sparse matrix grouped-variable Jacobian generation technique would be beneficial especially for the sequential-clustered and simultaneous integration approaches, but this feature is not currently incorporated into the prototype simulator.

Additional testing of the simulator demonstrated the use of the method of lines for treatment of distributed parameter models in the dynamic, modular simulation framework. The resulting ODE sets were easily handled by the stiff integrator. In addition, experience in constructing and testing the simulator has shown that a preprocessor and plex data structures are well suited to this type of simulation, allowing for enhanced flexibility and making good use of available storage.

## REFERENCES


Barney J. R., Dynamic simulation of large stiff systems. Ph.D. Thesis, McMaster University, Hamilton, Ontario (1975).

Biegler L. T., Simultaneous modular simulation and optimization. *Proc. Second Int. Conf. on Foundations of Computer-Aided Process Design* (A. W. Westerberg and H. H. Chien, Eds), p. 369. CACHE, Austin (1983).

Biegler L. T., On the simultaneous solution and optimization of large scale engineering systems. *Comput. chem. Engng* **12**, 357 (1988).

Biegler L. T. and J. E. Cuthrell, Improved infeasible path opimization for sequential modular simulators—I. The interface; II: The optimization algorithm. *Comput. chem. Engng* **9**, 245 (1985).

Biegler L. T. and R. R. Hughes, Process optimization: a comparative case study. *Comput. chem. Engng* **7**, 645 (1983).

Byrne G. D. and A. C. Hindmarsh, *Numerical Solution of Stiff Ordinary Differential Equation Sets*. AIChE Today Series, New York (1977).

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

Carver M. B., FORSIM-VI: a program package for the automatic solution of arbitrarily-defined differential equations. *Comput. Phys. Commun.* **17**, 239 (1979).

DeGrout J. J. and M. J. Abbott, A computation of one-dimensional combustion of methane. *AIAA Jl* **3**, 381 (1965).

Distefano G. P., Stability of numerical integration techniques. *AIChE Jl* **14**, 946 (1968).

Dudczak J., Optimal structuring of modular computations of chemical engineering systems. *Comput. chem. Engng* **10**, 7 (1986).

Duff I. S., *Sparse Matrices and Their Uses*. Academic Press, New York (1981).

Enright W. H. and T. E. Hull, Comparing numerical methods for the solution of stiff systems arising in chemistry. *Numerical Methods for Differential Systems* (L. Lapidus and W. E. Schiesser, Eds). Academic Press, New York (1976).

Evans L. B., B. Joseph and W. D. Seider, System structures for process simulation. *AIChE Jl* **23**, 658 (1977).

Fagley J. C., Flexibility and efficiency in modular dynamic chemical plant simulation. Ph.D. Thesis, University of Michigan (1984).

Franks R. G. E., *Modeling and Simulation in Chemical Engineering*. Wiley, New York (1972).

Gear C. W., The automatic integration of stiff ordinary differential equations. *Inform. Process.* **68,** 187 (1968).
Gear C. W., *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, New York (1971).
Guertin E. W., J. P. Sorenson and W. E. Stewart, Exponential collocation of stiff reactor models. *Comput. chem. Engng* **1,** 197 (1977).
Ham P. G., REMUS: the transient analysis of integrated chemical processes. Ph.D. Thesis, University of Pennsylvania (1971).
Haydweiller J. C., R. F. Sincovec and L. Fan, Dynamic simulation of chemical processes described by distributed and lumped parameter models. *Comput. chem. Engng* **1,** 125 (1977).
Hillestad H. and T. Hertzberg, Dynamic simulation of chemical engineering systems by the sequential modular approach. *Comput. chem. Engng* **10,** 377 (1986).
Hindmarsh A. C., Construction of math software, Part III. The control of error in the Gear package. Report UCID 30050 Part 3, Lawrence Livermore Laboratories (1974a).
Hindmarsh A. C., Gear . . . ODE system solver. Report UCID 3000, 1, Rev. 3, Lawrence Livermore Laboratories (1974b).
Hindmarsh A. C., Linear multistep methods for ODEs . . . Method formulations, stability, and the methods of Nordsieck and Gear. Report UCRL-51186, Lawrence Livermore Laboratories (1977).
Hindmarsh A. C., Stiff system problems. Report UCRL-87406, Lawrence Livermore Laboratories (1979).
Hlavacek V., Analysis of a complex plant—steady state and transient behavior. *Comput. chem. Engng* **1,** 1 (1977).
Hutchison H. P., D. J. Jackson and W. Morton, The development of an equation-oriented flowsheet simulator and optimization package, I and II. *Comput. Chem. Engng* **10,** 1 (1986).
Ingels D. M. and R. L. Motard, PRODYC—a simulation for process dynamics and control. Report Re. 4-70, University of Houston (1970).
Johnson A. I. and J. R. Barney, Numerical solution of large systems of stiff ordinary differential equations in a modular simulation framework. *Numerical Methods for Differential Systems* (L. Lapidus and W. E. Schiesser, Eds). Academic Press, New York (1976).
Liu Y. C. and C. B. Brosilow, Simulation of large scale dynamic systems. *Comput. chem. Engng* **11,** 241 (1987).
Kuru S. and A. W. Westerberg, A Newton-based strategy for exploiting latency in dynamic simulation. *Comput. chem. Engng* **9,** 175 (1985).
Lopez L., DYSCO: an interactive executive program for dynamic simulation and control of chemical processes. Ph.D. Thesis, University of Michigan (1974).
Mah, R. S. H., S. Michaelson and R. W. H. Sargent, Dynamic behavior of multi-component multi-stage systems. Numerical methods for solution. *Chem. Engng Sci.* **1,** 3 (1962).
Morton W. and G. J. Smith, Time delays in dynamic simulation. *Comput. chem. Engng* **13,** 631 (1989).
Motard R. L., M. Shacham and E. M. Rosen, Steady-state chemical process simulation. *AIChE Jl* **21,** 417 (1975).
Palusinski O. A., Simulation of dynamic systems using multirate integration techniques. *Trans. Soc. Comput. Sim.* **2,** 11 (1985).
Patterson G. K. and R. B. Rozsa, DYNSYL: a general-purpose dynamic simulator for chemical processes. *Comput. chem. Engng* **4,** 1 (1980).
Perkins J. D., Equation-oriented flowsheeting. *Proc. Second Int. Conf on Foundations of Computer-Aided Process Design* (A. W. Westerberg and H. H. Chien, Eds), p. 309. CACHE, Austin (1983).
Ponton J. W., Dynamic process simulation using flowsheet structure. *Comput. chem. Engng* **7,** 13 (1983).
Ponton J. W. and V. Vasek, A two-level approach to chemical plant and process simulation. *Comput. chem. Engng* **10,** 277 (1986).
Shacham M. *et al.*, Equation oriented approach to process flowsheeting. *Comput. chem. Engng* **6,** 79 (1982).
Shinskey F. G., *Distillation Control*. McGraw-Hill, New York (1977).
Stadtherr M. A. and J. A. Vegeais, Recent progress in equation-based process flowsheeting. *Proc. 1985 Summer Computer Simulation Conf.* Society for Computer Simulation (1986).
Weimer A. W. and D. E. Clough, A critical evaluation of the SIRK methods for stiff systems. *AIChE Jl* **25,** 730 (1979).