MODELING, IDENTIFICATION AND CONTROL, 1986, VOL. 7, NO. 3, 107–127
doi:10.4173/mic.1986.3.1

Dynamic simulation of chemical engineering systems by the sequential modular approach[†]

MAGNE HILLESTAD[‡] and TERJE HERTZBERG[‡§]

Keywords: *modular integration, sequential, coordination algorithm, prediction of tear variables, interpolation, multirate*

An algorithm for dynamic simulation of chemical engineering systems, using the sequential modular approach, is proposed. The modules are independent simulators, and are integrated over a common time horizon. Interpolation polynomials are used to approximate input variables. These input polynomials are updated before modules are integrated in order to interpolate output from preceding module(s) and thereby increase coupling and stabilize the computation. Tear stream variables have to be predicted at future time t_{n+1}, and various prediction methods are proposed.

1. Introduction

Steady-state process simulation is the most widely used tool for design and optimization of large scale process systems today. For these purposes steady-state models are adequate. There are also several dynamic process simulation systems available today (Cameron 1983; Hlavacek 1977), but for one reason or another, no system has yet been accepted as a general tool for large scale dynamic process simulation. A general tool for simulation of the dynamic behaviour of large scale processes is desired (Balchen, Fjeld and Saelid 1983), because there are so many areas where such a tool can be applied, which extend beyond those of steady-state simulation. For example

operability studies
control systems evaluation
start-up and shut-down studies
planning and control of plant commissioning
hazard analysis
response due to equipment failure
effect of changes in process and control parameters
operator training
studies of batch process operations

Dynamic process simulation systems will have to tackle many numerical and computer scientific problems; some of these are discussed here. Stiffness will be

Received 27 January 1986.
[†] Copyright 1986, Pergamon Press Ltd. This paper has been published in Computers and Chemical Engineering, Vol. 10, and is reprinted in MIC with permission from the publisher.
[§] Author to whom correspondence should be addressed. Institutt for Kjemiteknikk, NTH, 7034 Trondheim, Norway

restrictive on integration steplength when explicit integration is used. Implicit integration codes, on the other hand, require approximation of the Jacobian, which may be very laborious for large scale systems. The problem of stiffness can be reduced or even eliminated by applying a multi-rate method (Gear 1980) to the system. This paper will address this class of methods in more detail, and suggests the use of independent modules to be calculated in a sequential manner.

The method described here utilizes the fact that the equations represent chemical process units or modules, and exploits the information of process topology. Modular based flowsheeting will make the process modeling phase very convenient and user friendly. The proposed method is also computationally favourable when the system is sparse and the cost of function evaluations is high. This method will also enable existing numerical software, like integration codes, to be easily implemented independently of the overall system. Most of the commercially available steady-state process simulation systems are sequential modular systems. By the proposed method the use of these systems may be extended to dynamic simulation by modifying the executive system, and by adding dynamic modules. In this way the effort required to build a dynamic process simulation system will be relatively small.

Although this presentation does not include any large scale problems, the multi-rate nature is demonstrated more than adequately by two small examples. The subject of this paper will be continued in a following paper, which will treat more realistic problems.

There are many reasons that have prevented a widespread use of dynamic simulation of large scale process systems. The more significant reasons are that the development of dynamic models is time consuming, the cost of engineering time is high, that available tools are not easy to use, and there is a lack of confidence in the validity of the models. Moreover, the system of equations is not easy to solve, because of some properties this system of equations possesses. Typically one finds that

It is a system of differential/algebraic equations.

It is a system of large dimensionality (>1000), but sparse ($<1\%$ of the Jacobian are non-zero elements).

It is a system of stiff equations.

It is a system of non-linear equations.

There may occur discontinuities (time and state events).

These properties will cause trouble for the integration routine, if they are not given proper attention.

Modeling chemical engineering systems will often result in a mixed system of differential and algebraic equations (DAE). This equation system can be handled by diagonal implicit Runge–Kutta methods (DIRK) discussed by Cameron (1981), whereas Gear and Petzold (1982) apply the BDF method and discuss various problems related to the index of the DAE system.

The difficulties associated with discontinuities are in detecting them, in locating the time at which they occur, and in restarting the integration efficiently. For Runge–Kutta methods these difficulties are examined by Ellison (1981) and Enright, Jackson, Nørsett and Thomsen (1986); a paper by Gear and Østerby (1984) covers these problems for multi-step methods.

Stiffness is usually a problem when the equations have time constants that differ by orders of magnitude, and is characterized by the integration steplength no longer being controlled by accuracy, but rather by stability (see Fig. 4(*c*), where this is the case). Applying an explicity integration method to a system with widely different time constants will cause the integration steplength almost to vanish and the number of function evaluations to increase dramatically. An A-stable integration method is a method where the steplength seldom will be controlled by stability. In general no explicit integration method is A-stable and consequently they are unsuitable for solving large scale chemical engineering problems. By solving the composite system simultaneously it is necessary to apply an implicit integration method. This will lead to an implicit algebraic system that has to be solved at each integration step. This is usually done by a modified Newton iteration scheme where the Jacobian has to be updated from time to time. These computations are expensive and the Jacobian is updated only when the code has difficulties with convergence.

The philosophy of the multi-rate integration methods is to isolate some of the most rapidly decaying components and apply a suitable integration method and steplength to them. The slowly decaying components may use an explicit integration method and steplength best suited for them.

In this paper it is suggested that the well known sequential modular approach, where the equations are segregated by physical process units, be used. Modules may still have time constants which differ greatly in magnitude, but applying an implicit integration method to a module is more efficient than applying it to the whole system. This is because the dimension of the whole system is larger than the dimension of a module and consequently there is much less work involved in computing the Jacobian of a module.

2. Methods

In this section, a brief survey of different methods for dynamic process simulation is given and an attempt is made to clarify some terms used in the literature. As with steady-state simulation, dynamic simulation systems can be classified as either equation based or modular based systems. Cameron (1983) gives a thorough review of various systems and methods applied to large scale dynamic process simulation.

2.1. Equation based methods

The principal feature of equation based systems is that they operate on equations. In contrast, modular based systems see the module as a 'black box' and are not able to analyse the internal structure of the modules. With equation based systems, it is of utmost importance to solve the system of equations effectively. This can be done by using different ways of partitioning and by using sparse matrix techniques. A simulation system based on equation operations, like DPS (Wood *et al.* 1984), analyses the occurrence matrix and decomposes the system to get a block triangular system. It is then solved by direct or iterative techniques. In a recent paper by Kuru and Westerberg (1985) the flowsheet is decomposed so that the individual units can be treated separately. A predictor–corrector method is used for integrating the system of equations. The algorithm allows each unit, depending on its dynamic activity, to solve the corrector with different number of Newton–

Raphson iterations. Hachtel *et al.* (1981) give a survey of large scale decomposition theory.

Another different and promising way of solving sets of stiff differential equations is by partitioning the system into two sub-systems—a slow and a fast. Different integration methods are applied to each sub-system and different steplengths are used depending on the behaviour of the sub-systems. The two sub-systems are usually connected by means of polynomial interpolation or extrapolation.

There are several papers in the literature which treat the numerical aspect of these methods called multi-rate methods. Partitioning strategies, on the other hand, have not been treated exhaustively in spite of the fact that partitioning plays a very critical role in the overall performance of the numerical integration. By using a multi-rate method the stiffness problem is reduced or even eliminated. Among those who have contributed to the subject, a few are referred to here. Gear (1980) and Orailoglu (1979) apply Adams formulas on the two sub-systems, while Andrus (1979) uses Runge–Kutta formulas. Gomm (1981) analyses the stability of a multi-rate method. Skelboe (1984) uses the BDF integration methods for both sub-systems, and also gives a stability analysis of the method. Søderlind (1984) has developed a program system, DASP3, that integrates partitioned systems. It may be a stiff system of ordinary differential equations and a system of differential-algebraic equations. The nonstiff part is integrated by the classic fourth order Runge–Kutta method, and the third order BDF method is applied to the stiff part. Partitioning can also be done in other ways; for example, by dividing the system into a linear and a nonlinear part, as described by Palusinski and Wait (1978). These and other techniques for effective solution of large stiff equation systems are discussed in more detail by Hillestad (thesis, in preparation).

2.2. *Modular based methods*

The principal feature of modular based systems is that the system provides input and parameter values to the modules that return with the corresponding output. The modules, representing a process unit or a group of units, are linked together according to the process topology. There are two different approaches to modular dynamic simulation.

First, there is an approach which solves all the modules including interconnections simultaneously by a single integration routine. This is called *simultaneous modular* by Hlavacek (1977) and Fagley and Carnahan (1983), but by Patterson and Rozsa (1980) and Cameron (1983) it is called *coupled modular*. The former is more descriptive, but in steady-state simulation the same label is used on a totally different approach (two-tier). With this method each module is computed equally many times, as all modules are integrated by a common routine. The modules consist merely of model equations that provide the integration routine with derivatives. The general purpose process simulation system DYNSYL (Patterson and Rozsa 1980) has an option for this approach.

With the other approach, all modules are integrated over a common time interval or time horizon. Modules are provided with individual integration routines, and are in a sense independent simulators. This is called *sequential modular* by Hlavacek (1977), while Cameron (1983) uses *uncoupled modular*, and Fagley and Carnahan (1983) use a third term—*independent modular*. In fact, the latter label actually comprises the two former, because independent modules can either be computed in a sequential or parallel manner or by a combination of both. Liu and Brosilow (1983) name this approach *modular integration.*

3. Independent modular approach

As already mentioned, all modules are, by this approach, integrated over a common time horizon set by a coordination algorithm or coordinator. The philosophy is to have each module nearly independent of the overall system as illustrated by a typical module in Fig. 1. By integrator we normally mean a routine for solving a system of ordinary differential equations; but partial differential equations may also be solved if there are devices for proper discretisation of the equations.

The user or modeler is left with great freedom in choosing integrator, type of model, printout routines, etc., when composing an independent module. The model equations may be physically based as well as empirical input–output correlations. Depending on model type, different routines are selected to fit with the module. If the modules contain steady-state models, equation solvers are used for generating output from the module.

Normally integration routines are used and these will advance the modules in time with different speed or steplengths depending on the integration method, local error tolerance, and stiffness of the model. The independent modular approach is therefore a multi-rate integration method. Integration routines are chosen based on

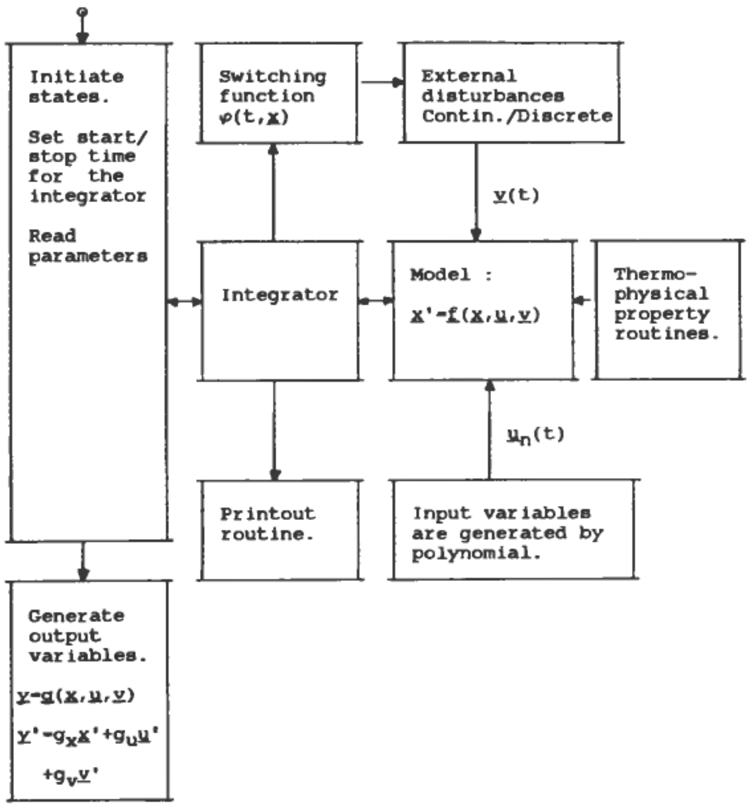

Initiate states.
Set start/stop time for the integrator
Read parameters
Switching function $\varphi(t,\underline{x})$
External disturbances Contin./Discrete
$\underline{v}(t)$
Integrator
Model : $\underline{x}'=\underline{f}(\underline{x},\underline{u},\underline{v})$
Thermo-physical property routines.
$\underline{u}_n(t)$
Printout routine.
Input variables are generated by polynomial.
Generate output variables.
$\underline{y}=\underline{g}(\underline{x},\underline{u},\underline{v})$
$\underline{y}'=g_x\underline{x}'+g_u\underline{u}'+g_v\underline{v}'$

Figure 1. Typical independent module.

properties of the module; fast or slow, discontinuities or not, differential–algebraic or purely differential equations.

On the other hand, this approach requires a coordinator to take care of communication between modules, predict tear stream variables, interpolate input variables, etc. The coordinator can be made in different ways and some proposals are found in the literature.

Liu and Brosilow (1983) have proposed three different algorithms for independent modular integration. The modules are parallel in the sense that all input variables are treated as iteration variables. These are iterated to some level of accuracy at each time horizon. The algorithms do not exploit information of flowsheet structure.

Ponton (1983) suggests using a sequential calculation order based on formation about the flowsheet. A criterion for tearing is also stated. It is suggested to tear a cycle by tearing the stream leaving the largest capacity. Input variables are held constant over the time horizon, which will give an approximation of order zero. In order to attain accuracy the time horizon will therefore have to be very small for all practical problems.

Birta (1980) starts off with the composite system of equations and divides this into sub-systems. The method is thus not modular, but rather equation oriented. The method is mentioned here because the same philosophy also can be applied to independent modules. Off block-diagonal elements are treated as input variables and these are approximated by third order interpolation or extrapolation polynomials. The polynomials interpolate the input values (u_n, u'_n, u_{n-1}, u'_{n-1}). Birta (1980) also gives an analysis of the cost benefits of his method relative to the simultaneous solution of equations. This algorithm is also parallel because all input variables are iterated simultaneously.

4. The proposed method

The algorithm presented here is based on the use of independent modules, as shown in Fig. 1, computed in a sequential order. Input variables are described by interpolation polynomials. Before a module is to be computed, its input polynomials are updated to interpolate the last calculated output from the preceding module(s). This is done in order to increase coupling between modules, but also to stabilize the computation.

For each input variable the polynomial interpolates the values (u_n, u_{n-1}, …, u_{n-q} and u'_n). The corresponding discrete times (t_n, t_{n-1}, …, t_{n-q}) are not necessarily a constant mesh of points, but may have variable stepsize. All input polynomials have the same order $p = q + 1$, and also the same mesh of time points in order to minimise coordination work.

The interpolation polynomial with centre at t_k and order p is thus $u_k(t)$

$$u_k(t) = \sum_{i=0}^{p} b_{k,i} \cdot (t - t_k)^i = u(t) + O[(t - t_k)^{p+1}] \qquad (1)$$

This polynomial is such that

$$u_{n-j} = \sum_{i=0}^{p} b_{k,i} \cdot (t_{n-j} - t_k)^i; \quad j = 0,\ q \qquad (2)$$

$$u'_n = \sum_{i=1}^{p} b_{k,i} \cdot i \cdot (t_n - t_k)^{i-1} \tag{3}$$

The polynomial coefficients, $\mathbf{b}_k$, are not estimated by these equations, but by a more sophisticated updating procedure partly described by Gear (1971) and Cook and Brosilow (1980).

To control the accuracy of the polynomials, there are three factors that can be 'adjusted'

the principal truncation error constant ($b_{k,\,p+1}$)

the time horizon (th)

the polynomial odcer (p)

To illustrate, assume we are integrating a sequence of modules from t_n to t_{n+1}, and having polynomials, $\mathbf{u}_n(t)$, for all input variables. Before integrating a module, its polynomials should be updated to interpolate the last calculated input, namely $\mathbf{u}_{n+1}$ and $\mathbf{u}'_{n+1}$, and centralize at t_{n+1}, to get $\mathbf{u}_{n+1}(t)$. This sequential procedure will have a smaller principal truncation error constant, and it will also be computationally more stable than the parallel method (Hillestad, 1986).

The time horizon, $th = t_{n+1} - t_n$, should be as small as possible to obtain highest accuracy. However, a small time horizon is very inconvenient because it will increase the computational load drastically. Selecting the time horizon is discussed later.

The effect of polynomial order is not particularly obvious. The optimal order is not the highest possible order, but it can be calculated as in Gear (1971). In the algorithm proposed here the polynomial order is simply increased from one (initial value) to maximum order (user supplied), unless discontinuities occur. If the polynomial order is p and the integrators have at least order $p + 1$, then the overall integration order is p (Gear, 1980). This interpolation procedure will correspond to a q-step method of integration with variable stepsize and variable order $p = q + 1$. Since the centre t_k is chosen equal to the last calculated point, it will correspond to an implicit integration method. Observing this, it will be easier to discuss stability with respect to the time horizon and order.

4.1. Algorithm

After tear streams and precedence ordering are determined, the algorithm can be summarized by the following points

I. Predict tear stream variables at t_{n+1}.

II. Repeat for all modules in precedence order.

1. Compute the difference between calculated and extrapolated input variable at t_{n+1}.

$$\varepsilon = u_{n+1} - u_n(t_{n+1}) \tag{4}$$

Compute the difference between calculated and extrapolated derivative at t_{n+1}.

$$\delta = u'_{n+1} - u'_n(t_{n+1}) \tag{5}$$

Deviation values are computed for all input variables of the present module.

2. Update the interpolation polynomial by the following recursion formula

$$\mathbf{b}_{n+1} = D \cdot \mathbf{b}_n + \mathbf{L}_1 \cdot \varepsilon + \mathbf{L}_2 \cdot \delta \tag{6}$$

Where D is some variant of Pascal's triangle matrix. D, $\mathbf{L}_1$ and $\mathbf{L}_2$ vary from time horizon to time horizon, but a fast updating procedure is used to compute these (Hillestad, in preparation). This is a rapid recursion formula since the same matrix D and vectors $\mathbf{L}_1$ and $\mathbf{L}_2$ are used for all input variables in a sequence. The algorithm does not involve matrix inversion as long as the polynomial order is fixed. The resulting polynomial, $u_{n+1}(t)$, is now interpolating the following values: ($u_{n+1}, u_n, u_{n-1}, \ldots, u_{n-q+1}$ and u'_{n+1}).

3. Use $\mathbf{u}_{n+1}(t)$, when integrating the module over the time horizon to get estimates of states and their derivatives at the end: $\mathbf{x}_{n+1}$ and $\mathbf{f}_{n+1}$.
4. Compute output from module $\mathbf{y}_{n+1}$ and $\mathbf{y}'_{n+1}$.
5. Equate these to the input of the succeeding module(s).

III. Compute relative error between predicted and calculated tear stream variables.

IV. Update time horizon.

If the sequence outlined in II results in an error as calculated in III which is greater than the specified tolerance, then there are two possible actions. (1) We can recompute the sequence with the same time horizon, or (2) we can reduce the time horizon and recompute the sequence. When recomputing the sequence with the same time horizon, II.1 and II.2 are slightly modified to become:

II.1

$$\varepsilon = u_{n+1} - u_{n+1}(t_{n+1}) \tag{7}$$

$$\delta = u'_{n+1} - u'_{n+1}(t_{n+1}) \tag{8}$$

II.2.

$$\mathbf{b}_{n+1} = \mathbf{b}_{n+1} + \mathbf{L}_1 \cdot \varepsilon + \mathbf{L}_2 \cdot \delta \tag{9}$$

4.2. Tear streams

Perhaps the most critical point in succeeding with this method is having good estimates of tear stream variables at t_{n+1}. We have to predict these variables if we want to use the independent modular approach consistently. There are, however, several ways of obtaining estimates of these variables, because we have already calculated and stored historical data necessary to predict future values.

Using the algorithm discribed here, we have to predict both $\mathbf{u}_{n+1}$ and $\mathbf{u}'_{n+1}$ for all tear streams. The following six ways of predicting these variables are suggested below

1. Keep the polynomials, $\mathbf{u}_n(t)$, of the tear streams as they are. This corresponds to extrapolation of the tear variables by means of existing polynomials.
2. Extrapolate the states and their derivatives of the module preceding a tear stream

$$\mathbf{x}_{n+1} = \sum_{j=0}^{q} \gamma_j \cdot \mathbf{x}_{n-j} \tag{10}$$

$$\mathbf{f}_{n+1} = \sum_{j=0}^{q} \gamma_j \cdot \mathbf{f}_{n-j} \tag{11}$$

3. States and their derivatives of the module preceding a tear stream are estimated by using the predictor

$$\mathbf{x}_{n+1} = \sum_{j=1}^{q} \alpha_j \cdot \mathbf{x}_{n+1-j} + \beta_1 \cdot \mathbf{f}_n \tag{12}$$

$$\mathbf{f}_{n+1} = \sum_{j=0}^{q} \eta_j \cdot \mathbf{x}_{n+1-j} \tag{13}$$

4. Use a linearized model to predict states and their derivatives for the module preceding the tear stream.

$$\mathbf{x}_{n+1} = P_n \mathbf{x}_n + \mathbf{r}_n \tag{14}$$

$$\mathbf{f}_{n+1} = Q_n \mathbf{f}_n + \mathbf{s}_n \tag{15}$$

   The matrices P_n and Q_n and the vectors $\mathbf{r}_n$ and $\mathbf{s}_n$ are calculated with the Jacobian of the module at t_n and previously calculated states (neglecting input and external disturbances).

5. Predict the tear stream variables by rigorous integration of the preceding module.
6. Calculate the whole sequence of modules to obtain an estimate of tear variables. This method, however, may abandon the advantage of using fewer function evaluations than straightforward methods. By increasing integration tolerance and by using simplified thermo-physical models as suggested by Barratt and Walsh (1979), this method of predicting tear stream variables will perhaps not be more expensive than conventional methods. Equations (7)–(9) are used when recomputing the sequence.

The prediction of derivatives in eqns. (11), (13) and (15) may be exchanged with a call to the model routine. The first three methods are based on serially correlated models, while the latter three are more or less rigorous models. Methods 2–5 are based on predicting tear stream variables by first predicting internal states of the preceding module to a tear stream and then computing output stream variables. By using methods 2–5, we assume that these modules contain sufficient state information to be able to predict future states and derivatives. The philosophy of using such prediction modules is that they contain sufficient memory to be able to predict future states almost independently of the form of the input of these modules. Each tear stream of a flowsheet may, of course, use different prediction methods and should obviously use the method best suited. Other methods for prediction may be constructed as well, but the critical points will always be *stability*, *accuracy* and *speed* of the prediction method.

The network theory is, naturally, the same for dynamic and steady-state simulation, but the criteria for choosing the best tear set are different. If we want to do tearing automatically as described by Gundersen and Hertzberg (1983), it is obvious that several additional criteria for weighting the streams have to be implemented for 'dynamic' tearing. Ponton (1983) suggests that a general rule is to tear the stream leaving the unit in a loop having the largest capacity or time constant. This is reasonable because it is easier and more accurate to predict output from a slow unit than a rapid one. There are other criteria that should also be considered.

Number of stream variables: We wish to use the least possible number of iteration variables, which is related to the total error estimate of the tear stream variables.

Number of prediction modules: Using methods 2–5, we want to predict the necessary tear stream variables by using as few prediction modules as possible, which is related to the speed of prediction.

Time constant and time delay: It is convenient to approximate process units by a time delay and by first order dynamics. Using methods 2–5, which are 'one module methods', the time constant and time delay are obviously measures of how well the output is predicted. Since we do not know the exact input to these prediction modules a priori, the sensitivity of the input over the time horizon should be as small as possible. The time horizon, time constant and the time delay should be such that the output (tear stream) at t_{n+1} is mainly affected by the internal states of the module. If there is a choice, one should pick the tear set with prediction modules having the largest time delay and time constant.

Ease in computing the Jacobian: Using method 4, which is a linearized model, the Jacobian should be as easy as possible to evaluate in order to increase speed.

Ease in integrating the module: Using method 5, the stiffness ratio of the module should be low in order to integrate with high speed.

Discontinuities: External disturbances imposing discontinuities on modules preceding tear streams can only be handled by a rigorous integration, preferably by using an integrator made to detect these. Method 5 or 6 will have to be used.

There is much more to be said about determining the best tear set and choosing the right method for prediction of tear variables in dynamic simulation, as almost nothing has been done in this area.

4.3. Time horizon

Liu and Brosilow (1983) suggest adjusting the time horizon (th) by comparing the calculated error of tear variables and the maximum tolerance limit ($e_{\max}$). Knowing that all stream variables are of order p, one finds

$$err = O((th)^{p+1}) \tag{16}$$

This gives the following updating scheme for the time horizon, which is implemented in most integration codes for steplength adjustment as

$$(th) := (th)(e_{\max}/err)^{1/(p+1)}\, 0{\cdot}9 \tag{17}$$

The absolute error, err, may be estimated as the norm of the difference between predicted and calculated tear variables; thus

$$err = \| \mathbf{u}_{pt} - \mathbf{u}_{ct} \| \tag{18}$$

The time horizon will certainly also be restricted by stability as well. Since we already are controlling the error, the stability is automatically ensured. However this requires that we recompute the sequence with a reduced time horizon when the error is greater than the specified limit. Much effort will be wasted in this way, so it would be desirable to know, a priori, if the chosen time horizon is within the region of stability. How this can be done is not quite clear, but evidently the stability region of the corresponding multi-step integration method should be used. As already mentioned, the time horizon should be chosen also by considering the time constant and time delay of prediction modules.

Another criterion for choosing the right time horizon will also be considered. The time horizon is set equal to the maximum steplength calculated by the different integrators

$$(th) := \max_{i=1,n}(h_i) \tag{19}$$

In eqn. (19), h_i is the calculated steplength of module number i provided by the integration routine. This method does not require any initial value of the time horizon as eqn. (17) does. Furthermore, since it is not desirable to let the user specify an initial time horizon, eqn. (19) can at least be used for calculating the initial time horizon.

4.4. Implementation

The user gives initial states for all modules. But the system will also be able to compute steady-state values to be used as initial states in a dynamic simulation. Error tolerance and maximum order of polynomials are user supplied variables. The system will also store all variables, when terminating, to make it possible to restart the simulation with other experimental conditions, if desirable. External disturbances are user specified subroutine(s) that may be connected to any module. Disturbances may be physically derived models of the surroundings or they may be empirical correlations. The surroundings may, for example, be the pressure relief system, the ambient temperature, or the procedure for starting up a pump. Disturbances will often create discontinuities in the unit model and these are more likely to happen when a state variable has reached a certain threshold (state event), or at a certain time (time event). State events are determined by a user supplied switching function as illustrated in Fig. 1. They are taken care of by letting the particular integration routine detect the discontinuity and locate the time. Then, the time horizon is reduced to match this time, and the sequence is recomputed. To restart after a discontinuity has occurred, the same procedure will be used as proposed for multi-step integration methods by Gear and Østerby (1984).

5. **Results**

The proposed algorithm is currently being tested, so every aspect of it is not yet fully investigated. Here, two simple example problems are shown, where steplength control and accuracy are compared with the simultaneous solution of the equations.

5.1. Problems

These problems are constructed arbitrarily by means of two building blocks—SP and MT. SP and MT are types of modules and the number in parentheses corresponds to module number. Here, SP is a splitter with two output streams and one input stream. MT is a mixer with two input streams and one output stream. In

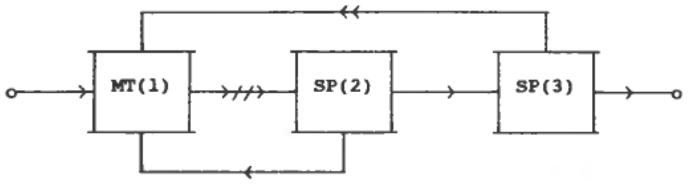


Figure 2. Example 1. Precedence ordering (2)-(3)-(1).

addition, MT has the possibility of having an external disturbance, which is an extra input stream. The models describing these units are very simple with only two state variables each (see appendix), but the main point is to illustrate two aspects of the approach—namely steplength and accuracy.

5.2. Steplength

To illustrate the steplength control of the two solution methods, the number of function evaluations are compared in Table 1. This very clearly illustrates the savings in the number of function evaluations by using the proposed algorithm. For comparison, Euler's method (embedded with the modified Euler's method for estimation of local error) and the same integration tolerance (10^{-4}) are applied to all integration. Theoretically, the number of evaluations with the sequential approach will never exceed that of the simultaneous solution in any module provided that the same integration method and tolerance are used. However this is not quite true for module 5 of example 2. This discrepancy stems from the fact that after every new time horizon, the initial steplength is set to an unnecessarily small value. But, although this can be avoided by simply recalling the last steplength from the previous time horizon, the extra evaluations, by doing this, are surprisingly few. Of course, we should use an initial steplength equal to the last estimate from the previous time horizon if that is possible with the routines available.

The plots in Fig. 4 are steplength as a function of time for example 1. Only accepted steps are plotted, except in Table 1 both accepted and rejected steps are included (two evaluations per step).

In Fig. 4, it can be seen that module 2 is the restrictive module with respect to steplength up to time $\simeq 1{\cdot}3$, and from there, module 3 is the restrictive one. From these two problems, which are not even stiff problems, it is obvious that there is a

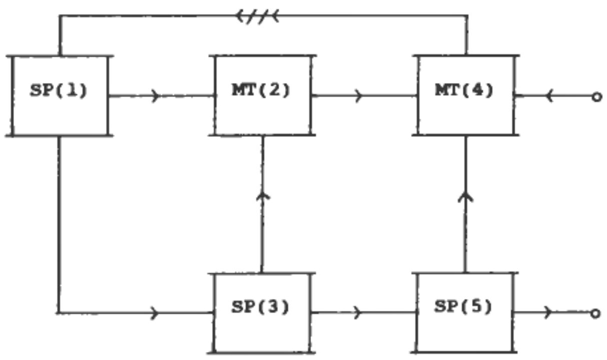


Figure 3. Example 2. Precedence ordering (1)-(3)-(2)-(5)-(4).

	Module number	Simultaneous solution	Sequential solution
Example 1	1	884	297
	2	884	641
	3	884	773
	sum	2652	1711
Example 2	1	513	127
	2	513	208
	3	513	319
	4	513	140
	5	513	519
	sum	2565	1313

Table 1. Number of function evaluations.

lot to be saved in the number of function evaluations. Even when using an initial integration steplength of 0·001, as illustrated, 35 and 49 per cent savings are achieved in example 1 and 2, respectively. For stiff and large scale problems the savings are expected to be much higher.

This comparison is not quite fair as the overhead is not taken into account, but for large scale problems the overhead will become relatively less significant (Birta 1980). The overhead will also be taken into account in the comparison when larger and more realistic problems are treated in a later study.

5.3. *Accuracy*

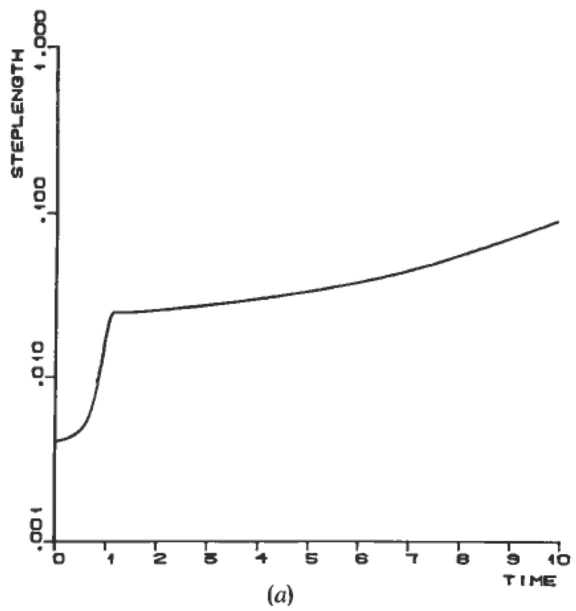
By reducing the number of function evaluations one expects accuracy to be reduced, but that is not the case. The saved evaluations are those not necessary to attain integration accuracy. However, the expected accuracy of the sequential method is not as high as that of the simultaneous solution, because prediction is involved in the sequential method. Prediction may also be involved in the simultaneous solution, but the prediction is then done over a much shorter time interval.

The two examples shown below indicate that there are no significant deviations between the solution trajectories of the two methods. The worst case which is module 2 of example 1 is shown in Fig. 5. The deviation between the solutions of x_2 is due to its strong dependency on x_1 when x_1 is small. Method 3 (eqns. (12) and 13)) is used for prediction of tear stream variables for the two examples.

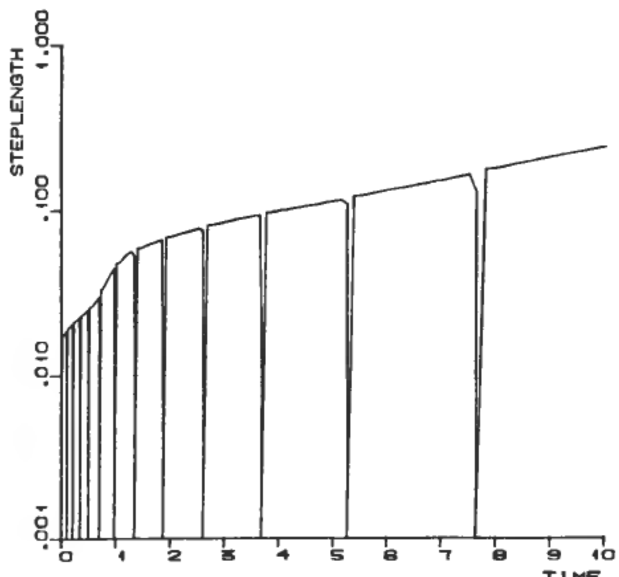
6. Discussion

Perhaps the greatest advantage of the independent modular approach is the multi-rate nature of the method. Slow units, like a flash tank, need few function evaluations over the time horizon, while fast units, like control units, have to do several more evaluations to attain the same integration accuracy. The total number of function evaluations required is therefore reduced when compared to the simultaneous solution, as illustrated. This advantage is very attractive, when the function evaluations are laborious as is often the case in process simulation with chemical unit operations involving heavy thermo-physical property estimations. However, the independent modular approach requires more administration, so increased program overhead should not override any savings. Small non-stiff problems involving few equations will certainly not benefit from using this approach with respect to computer time. In general, a multi-rate method can only be competitive if one of two conditions is true; either the cost of function evaluation is high, or the system of equations is sparse (few interpolations). In large scale process simulation it is likely that both are true.

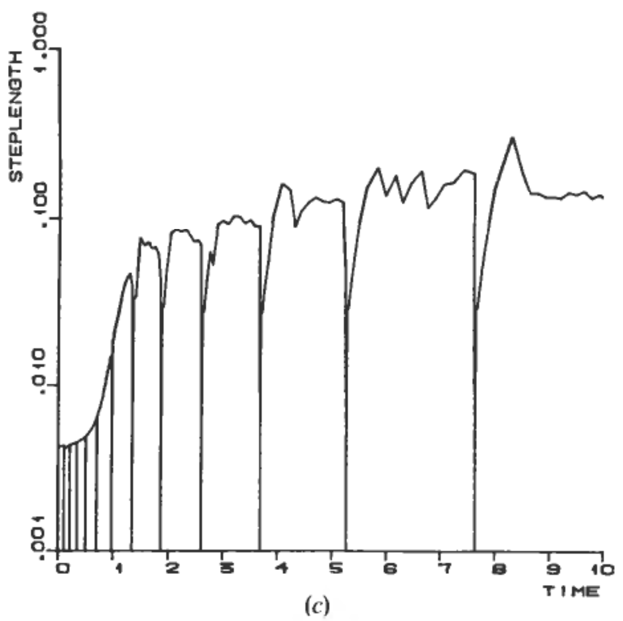
For a modular system in general and an independent modular system in particular it is easier to locating modeling or data errors. The software will be less complex and therefore easier to maintain. This is a well known fact from steady-state simulation. Also, modeling or setting up new process configurations will be easier. By this



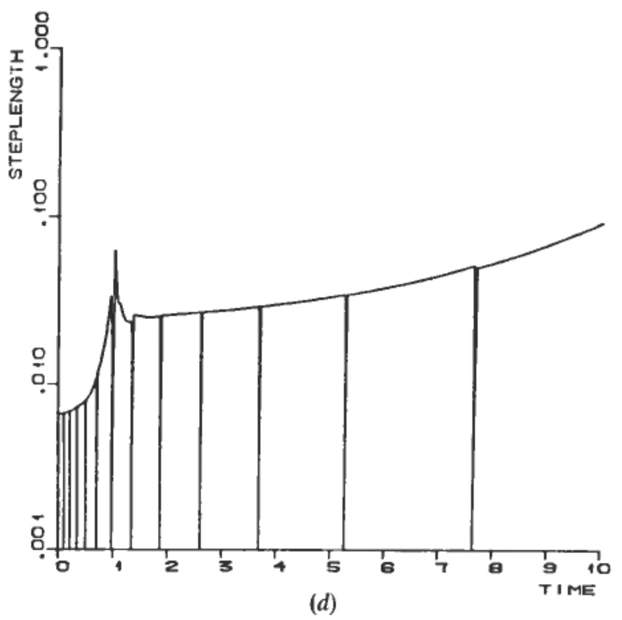
(*a*)



(*b*)



(*c*)



(*d*)

Figure 4. (*a*) Integration steplength of simultaneous solution of example 1. (*b*) Steplength of module 1. (*c*) Steplength of module 2. (*d*) Steplength of module 3.

approach it is possible to do 'modeling in the large' and the process engineer may test different control or process structures and designs within a short time.

The independent modular approach makes it very easy to exchange integration routines with newer and better ones. We are not tied to one particular choice of integration method. It should also be mentioned, as an advantage of the independent modular approach, that it is even possible to use the analytical solution by integrating the model using the approximative input polynomials. In addition, the integration routine does not even need to solve an initial value problem, but may

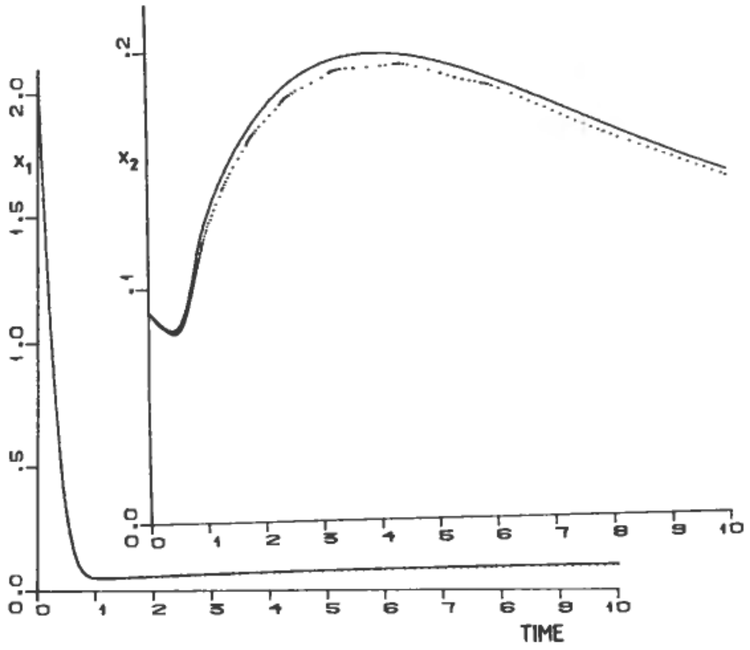


Figure 5. Solutions of module 2 of example 1. Sequential modular solution (dotted line) and simultaneous solution (solid line).

equally well be solving a two point boundary value problem in time. One or several states may be design variables, which are to meet specified design requirements within a specified time.

Another attractive aspect of the independent modular approach is that there is also great freedom with respect to choice of model type. Any module may use any type of model, provided that it is consistent with the rest of the module. Using a simultaneous solution, it seems to be difficult to combine differential equations with input–output models.

By solving the modules simultaneously with a single integration routine, the integrator will solve a system of equations which also includes the algebraic interconnections. This is avoided when an independent modular approach is used. The total system of equations that is seen by the integration routines will therefore be less complex than if all modules were solved simultaneously.

The sequential modular approach is well investigated, and much well-established theory and software exists for steady-state simulation. This may well be built upon for extension to dynamic simulation. Dynamic modules will have to be built, but with some modifications, the executive system will certainly suffice. Using the sequential modular approach in dynamic simulation, the tear stream convergence problem will normally not be any problem at all. This is due to the capacities and dead times of process units, which act like filters to changes in tear variables. Convergence may only be a problem when the time horizon is chosen greater than the time lag of a loop.

Discontinuities have to be handled efficiently by a dynamic process simulator. Using the independent modular approach, the detection and localization of the time when the discontinuity occurs will be left to the particular integration routine. The time horizon will then be reduced to match the discontinuity point, and the integration is then restarted. When a discontinuity occurs, there is no point in representing the stream variables by polynomials interpolating points established before the discontinuity. The polynomial order has to be reduced according to Gear and Østerby (1984). They also propose how the new time horizon should be chosen when restarting after a discontinuity has occurred.

By comparison of the sequential and parallel variants of the independent modular approach, it is obvious that the sequential involves fewer iteration variables and will therefore give a smaller error estimate of tear streams. Using the sequential approach, less work is involved in predicting tear variables for the same reason. The sequential variant will also be numerically more stable because the sequential and the parallel approach will correspond to Gauss–Seidel and Jacobi iteration, respectively. The sequential procedure has stability properties similar to an implicit integration, while the parallel procedure has stability properties similar to an explicit integration. The parallel approach may be computationally preferable only when using a multi-processor computer where each module is computed on an individual processor, i.e. parallel processing.

The sequential approach is limited by the assumption of directed information streams. The simulation will break down when an information stream changes direction unless the system automatically recovers by finding a new precedence ordering, tear streams and prediction modules. Unfortunately the information streams are not always identical to mass and energy streams. Quite often process units are affected by down stream units through the output pressure. This is handled by letting the output pressure become an input information stream to the actual module.

The independent modular approach will impose an external error control in addition to the error control of the integrators. This is unwanted, but is a necessary part of the coordination algorithm. However, as already stated, by calculating the modules sequentially there will normally be no problem with convergence of tear streams. When compared to the simultaneous solution of equations, the accuracy will normally be poorer due to prediction of tear variables. But if predictions are accurate, the same accuracy may be attained with the proposed method. This was the case when running examples 1 and 2 using prediction method 3.

Finally, the stability analysis for a multi-rate method, like the independent modular approach, is non-trivial. In fact, having merely two sub-systems, each with different step-control, will involve complex mathematics for finding the region of stability, as demonstrated by Gomm (1981). Since we are primarily interested in the region of stability for the time horizon, an analysis of the corresponding multi-step integration method will suffice assuming that the order of the integrators is at least $p + 1$ (Gear 1980). The internal integrator steplengths are controlled by the local error tolerances and will therefore lie within their regions of stability.

7. Conclusion

The proposed algorithm for sequential modular dynamic process simulation is promising. This is due to the multi-rate nature of the method which will be most competitive when the function evaluations are expensive, and the system of equations is sparse. Besides building on well investigated theory, the sequential modular approach also leaves the modeler with great freedom in the choice of integration method and type of model. In sequential modular dynamic simulation, the tear variables will not cause any convergence problems as long as the time horizon is chosen properly. With modifications, existing steady-state process simulation systems may be extended to dynamic simulation.

ACKNOWLEDGMENT

We thank Professor S. P. Nørsett at the Department of Numerical Mathematics for his help with the presentation and for his comments and suggestions during the work.

List of symbols

$\mathbf{b}_n$	Polynomial coefficients defined in eqn. (1).
D	Pascal's triangle matrix.
$e_{\max}$	Maximum error tolerance of tear variables; user-supplied.
err	Absolute error of tear variables; calculated.
$\mathbf{f}, f_x$	State equations and the Jacobian.
$\mathbf{g}, g_x, g_u, g_v$	Output equations, and partial derivatives of $\mathbf{g}$ w.r.t. $\mathbf{x}$, $\mathbf{u}$ and $\mathbf{v}$.
h_i	Steplength of module no. i.
$\mathbf{L}_1, \mathbf{L}_2$	Vectors from eqn. (6).
p	Degree of interpolation polynomials.
P_n	Matrix used in eqn. (14).
q	Number of time intervals used in the interpolation procedure ($q = p - 1$).
Q_n	Matrix used in eqn. (15).
$\mathbf{r}_n$	Vector used in eqn. (14).
$\mathbf{s}_n$	Vector used in eqn. (15).
t_n	Discrete time.
th	Time horizon $= t_{n+1} - t_n$.
$u(t)$	Exact input function.
$\mathbf{u}_n(t), u_n(t)$	Approximated input functions; polynomials of degree p and centralized at t_n.
$\mathbf{u}_n, u_n$	Input value at t_n.
$\mathbf{u}_{pt}, \mathbf{u}_{ct}$	Tear variables; predicted and calculated.
$\mathbf{v}(t)$	External disturbance functions; user-supplied.
$\mathbf{x}_n$	State vector at t_n.
$\mathbf{y}_n$	Output vector at t_n.
α_j, β_1	Coefficients in the predictor eqn. (12).
γ_j	Coefficients in the extrapolation eqn. (10) and (11).
η_j	Coefficients in the extrapolation eqn. (13).
ε	Difference calculated in eqn. (4).
δ	Difference calculated in eqn. (5).

REFERENCES

ANDRUS, J. F. (1979). Numerical solution of systems of ordinary differential equations separated into subsystems. *SIAM J. Numerical Anal.*, **16,** 606–611.

BALCHEN, J. G., FJELD, M., and SALID, S. (1983). Significant problems and potential solutions in future process control, paper presented at the Annual AIChE Meeting, Washington DC (1983).

BARRETT, A. and WALSH, J. J. (1979). Improved chemical process simulation using local thermodynamic approximations. *Comp. and Chem. Engg.*, **3,** 397–402.

BIRTA, L. G. (1980). A Quasi-Parallel Method for the Simulation of Loosely Coupled Continuous Subsystems. *Mathematics and Computers in Simulation*, **22,** 189–199.

CAMERON, I. T. (1981). (Ph.D. thesis) *Numerical Solution of Differential-Algebraic Equations in Process Dynamics.* Dept. of Chemical Engineering and Chemical Technology, Imperial College of Science and Technology, London SW7.

CAMERON, I. T. (1983). Large scale transient analysis of processes—the state of the art. *Lat. Am. J. Chem. Eng. Appl. Chem.*, **13,** 215–228.

COOK, W. J. and BROSILOW, C. B. (1980). A Modular Dynamic Simulation for Distillation Systems, Paper presented at the 73rd Annual AIChE Meeting, Chicago Ill. (1980).

ELLISON, D. (1981). Efficient automatic integration of ordinary differential equations with discontinuities. *Mathematics and Computers in Simulation*, **23,** 12–20.

ENRIGHT, W. H., JACKSON, K. R., NØRSETT, S. P. and THOMSEN, P. G. Interpolants for Runge–Kutta formulas. *Trans. of Math. Software* (submitted).

FAGLEY, J. and CARNAHAN, B. (1983). Efficiency and flexibility in dynamic chemical plant simulation, *Proceedings of PAChEC-83, The Third Pacific Chemical Engineering Conference*, Seoul, Korea, 8–11 May (1983), pp. 78–84.

GEAR, C. W. (1971). *Numerical Initial Value Problem in Ordinary Differential Equations* (Prentice Hall Inc., Englewood Cliffs, N.J.).

GEAR, C. W. (1980). Automatic multirate methods for ordinary differential equations, Department of Computer Science, University of Illinois at Urbana-Champaign, Report UIUCDCS-R-80-1000.

GEAR, C. W. and PETZOLD, L. R. (1982). ODE methods for solution of differential/algebraic systems, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Report UIUCDCS-R-82-1103.

GEAR, C. W. and ØSTERBY, O. (1984). Solving ordinary differential equations with discontinuities. *Trans. on Math. Software*, **10,** 23–44.

GOMM, W. (1981). Stability analysis of explicit multirate methods. *Mathematics and Computers in Simulation*, **23,** 34–50.

GUNDERSEN, T. and HERTZBERG, T. (1983). Partitioning and tearing of networks—applied to process flowsheeting. *Modeling Identification and Control*, **4,** 139–165.

HACHTEL, G. D. and SANGIOVANNI-VINCENTTELLI, A. L. (1981). A survey of third-Generation simulation techniques. *Proceedings I.E.E.E.* **69,** 1264–1280.

HILLESTAD, M. (1986). Thesis for the degree of dr.ing., Lab. of Chemical Engineering, The Norwegian Institute of Technology, N-7034 Trondheim-NTH (in preparation for submission).

HLAVACEK, V. (1977). Analysis of a complex plant—steady state and transient behavior. *Comp. and Chem. Engg.*, **1,** 75–100.

KURU, S. and WESTERBERG, A. W. (1985). A Newton–Raphson based strategy for exploiting latency in dynamic simulation. *Comp. and Chem. Engg.*, **9,** 175–182.

LIU, Y. C. and BROSILOW, C. B. (1983). Modular Integration Methods for Large Scale Dynamic Systems, Paper presented at AIChE Diamond Jubilee Meeting, Washington DC (1983).

ORAILOGLU, A. (1979). A multirate ordinary differential equation integrator, Dept. of Computer Science, University of Illinois at Urbana-Campaign, Report UIUCDCS-R-79-959.

PATTERSON, G. K. and ROZSA, R. B. (1980). DYNSYL: A general-purpose dynamic simulator for chemical processes. *Comp. and Chem. Engg.*, **4,** 1–20.

PALUSINSKI, O. A. and WAIT, J. V. (1978). Simulation method for combined linear and nonlinear systems. *Simulation*, **30,** 85–95.

PONTON, J. W. (1983). Dynamic process simulation using flowsheet structure, *Comp. and Chem. Engg.*, **7,** 13–17.

SKELBOE, S. (1984). Multirate Integration Methods, Elektronik Centralen, DK-2970 Hørsholm, Denmark.

SØDERLIND, G. (1980). DASP3—A Program for Numerical Integration of Partitioned Stiff ODEs and Differential-Algebraic Systems, Report TRITA-NA-8008, Numerical Analysis and Computer Science, The Royal Institute of Technology, S-10044 Stockholm 70, Sweden.

WOOD, R. K., THAMBYNAYAGAM, R. K., NOBLE, R. G. and SEBASTIAN, D. J. (1984). DPS—A digital simulation language for process industries, *Simulation*, 221–234.

Appendix

In both modules, MT and SP, the states x_1 and x_2 are level and concentration of the liquid in the tanks, respectively. The initial conditions and the parameters were chosen arbitrarily for the modules shown in Figs. 2 and 3. The parameter vector **p** and the initial conditions are figures attached to the particular module number.

Module MT

Input variables

u_1	Flowrate of stream 1
u_2	Concentration of stream 1
u_3	Flowrate of stream 2
u_4	Concentration of stream 2

External disturbances

$v_1 = p_3$	Flowrate is constant
$v_2 = p_4$	Concentration is constant

State equations

$$x_1' = (v_1 + u_1 + u_3 - p_2\sqrt{(x_1)})/p_1 \ldots x_1(a) = x_{10}$$
$$x_2' = (v_1(v_2 - x_2) + u_1(u_2 - x_2) + u_3(u_4 - x_2))/(p_1 x_1) \ldots x_2(a) = x_{20}$$

Output equations

$y_1 = p_2\sqrt{(x_1)}$	Flowrate
$y_2 = x_2$	Concentration
$y_1' = 0{\cdot}5 p_2 x_1'/\sqrt{(x_1)}$	
$y_2' = x_2'$	

Module SP

Input variables

u_1	Flowrate
u_2	Concentration

State equations

$$x_1' = (u_1 - (p_2 + p_3)\sqrt{(x_1)})/p_1;\ x_1(a) = x_{10}$$
$$x_2' = (u_1(u_2 - x_2))/(p_1 x_1);\ x_2(a) = x_{20}$$

Output equations

$y_{1,1} = p_2\sqrt{(x_1)}$	Flowrate of stream 1
$y_{1,2} = x_2$	Concentration of stream 1
$y_{2,1} = p_3\sqrt{(x_1)}$	Flowrate of stream 2
$y_{2,2} = x_2$	Concentration of stream 2
$y_{1,1}' = 0{\cdot}5 p_2 x_1'/\sqrt{(x_1)}$	
$y_{1,2}' = x_2'$	
$y_{2,1}' = 0{\cdot}5 p_3 x_1'/\sqrt{(x_1)}$	
$y_{2,2}' = x_2'$	

Numerical values for example 1

Module 1	Module 2	Module 3
MT	SP	SP
$p_1 = 10{\cdot}0$	$p_1 = 1{\cdot}0$	$p_1 = 1{\cdot}8$
$p_2 = 1{\cdot}2$	$p_2 = 1{\cdot}3$	$p_2 = 0{\cdot}9$
$p_3 = 1{\cdot}2$	$p_3 = 3{\cdot}2$	$p_3 = 0{\cdot}5$
$p_4 = 0{\cdot}03$		
$x_{10} = 0{\cdot}5$	$x_{10} = 2{\cdot}1$	$x_{10} = 4{\cdot}6$
$x_{20} = 0{\cdot}02$	$x_{20} = 0{\cdot}09$	$x_{20} = 0{\cdot}89$