

1983

A Newton-Raphson based strategy for exploiting latency in dynamic simulation

Selahattin Kuru
Carnegie Mellon University

Arthur W. Westerberg

Follow this and additional works at: <http://repository.cmu.edu/cheme>

This Technical Report is brought to you for free and open access by the Carnegie Institute of Technology at Research Showcase @ CMU. It has been accepted for inclusion in Department of Chemical Engineering by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:

The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

A NEWTON-RAPHSON BASED STRATEGY FOR
EXPLOITING LATENCY IN DYNAMIC SIMULATION

by

S. Kuru & A.W. Westerberg

December, 1983

DRC-O6-M7-33

**A NEWTON-RAPHSON BASED STRATEGY FOR
EXPLOITING LATENCY IN DYNAMIC
SIMULATION**

by

Selahattin Kurut

Arthur W. Westerberg

Department of Chemical Engineering

Carnegie-Mellon University

Pittsburgh, PA 15213

January, 1983

†Current address is Department of Computer Science, Bogazici
University, P. K. 2 Bebek - Istanbul, Turkey.

1. ABSTRACT

Latency¹ in dynamic simulation is a consequence of the difference between the dynamic behavior of different units that make up a flowsheet. Taking advantage of latency is essential to reduce the computation time in dynamic simulation. It is observed that in dynamic simulation there is a parallelism between the degree of dynamical activity of a unit and the computational effort required to realize this activity. An algorithm is presented to exploit this parallelism by avoiding some or all the steps involved in solving the corrector equations in the numerical integration process by the Newton-Raphson method. Unlike conventional ones, the proposed algorithm has no approximations built into it and solves the correct model of the flowsheet at each step of the integration. The algorithm is tested on an example problem and it is shown that significant savings in computation time can be achieved.

2. SCOPE

Very often in dynamic simulation, only a few of the units in a chemical flowsheet are dynamically active on a subrange of the integration interval. One reason for this is that some units in the flowsheet have large time constraints relative to those of others and hence the rate of change in the state of these units is not always appreciable. Time constraints of largely different magnitudes is one cause of stiffness in differential systems². Since the integration step-size is determined by the smallest time constant of a differential system, a dynamic simulator not taking special action for such a system would be forced to take small steps, thus demanding large amounts of CPU time. Another reason for some units to stay inactive for a period of time in dynamic simulation is that when a disturbance travels around the flowsheet, it affects only a few of the units at a time due to the dynamic lag between the units. This is because a lag weakens the coupling between the units. Note that a pure delay between two units decouples them completely if the integration step-size is less than the delay time in magnitude. The third reason for having dynamically inactive units is due to the action of control units in the flowsheet.

Latency in dynamic simulation is thus a consequence of the existence of dynamically inactive or relatively less active units in flowsheet. Taking advantage of latency is essential to reduce the computation time" in dynamic simulation.

There are basically two approaches used in the dynamic simulators existing in the

chemical engineering literature. One is to use the sequential modular approach³ and let the flow of material - and hence the path taken by a disturbance - determine the integration sequence⁴. This approach has two drawbacks: First, the existence of recycle streams introduces difficulties in applying this approach. Use of nested loops often results in excessive computation time. Second, the flow of information does not necessarily have to follow the flow of material.

The other approach⁵ is to simulate each unit separately for a subinterval, with step-sizes determined by individual units, as if the units are completely decoupled, and consider coupling only at the end of the subinterval. For each subinterval, coupling variables are kept at their values at the beginning of the subinterval. Though it may give reasonable solutions at times, this approach is not mathematically justifiable. The system solved in each subinterval is not a correct model of the flowsheet.

Here we introduce a different strategy where no artificial decoupling strategy is used so that the equations being solved are the correct model of the flowsheet. All the units in the flowsheet are simulated simultaneously and the step-size is determined by the fastest moving unit. This step-size is the smallest of the step-sizes that would be used by the individual units if the system were assumed to be decoupled unit by unit. The strategy is based on keeping track of a small subset of the trajectories followed by the units in the flowsheet. The information about the trajectory followed by a state variable in the past is stored as a requirement of the integration scheme, and the predictor formula predicts the value of the variable in the immediate future based on this information. Prediction of the algebraic variables for the model is shown to require keeping track of the trajectories of only a small subset of them. For a unit that follows a smoother (well approximated by a low order approximation) trajectory relative to the fastest moving unit, the predicted values of the state variables can be expected to be close to the corrected values. In other words, the predicted values may satisfy the corrector equations of the units experiencing slow dynamics within a tolerance, or at least, these units may require fewer corrector iterations to converge than the faster moving units in the flowsheet. The idea is to decompose the flowsheet unit by unit and to avoid some or all of the computational stages involved in the corrector iterations for those units that are dynamically inactive or relatively less active.

3. CONCLUSIONS AND SIGNIFICANCE

The strategy presented for exploiting latency is based on the intuition that units with little or no dynamical activity in a flowsheet should introduce little or no computational work in the numerical integration process in dynamic simulation. Arguing on this parallelism qualitatively, an algorithm is obtained for dynamic simulators that use predictor-corrector methods. Although the algorithm is based on the Newton-Raphson technique as the equation solving method, the idea is readily applicable to other techniques as well.

It is shown on an example that the algorithm has significant advantage over the conventional strategy in terms of the computation time. We note that the new algorithm has no approximation built into it and the solution is identical with that of the conventional algorithm in terms of the trajectories of the variables, the number of steps taken, and the step-sizes used in the integration.

4. INTRODUCTION

In order to understand fully the concept of latency we shall consider the flowsheet in Figure 1. We shall first examine the dynamic behavior of the units in the flowsheet in qualitative terms. Then we shall discuss the conventional approaches for exploiting latency and present the idea of the strategy introduced in this paper. Later in the paper we shall discuss the strategy in detail and deduce an algorithm from the arguments. The flowsheet in Figure 1 will be used as an example to show that significant savings in computation time can be achieved with the algorithm presented. The flowsheet in Figure 1 represents a batch process where the reactants A and B are converted into product C in controlled reaction conditions.

The reaction



is a second order endothermic reaction with a constant heat of reaction and a rate expression

$$\frac{dc_A}{dt} = -k_{AB} c_A c_B$$

The operation of the batch process is composed of the following four phases:

Phase 1: The reactor is filled with the reactants to a specified level.

Phase 2: Reactants are heated to the reaction temperature. A heating coil is used to supply constant heat flow to the reactor.

Phase 3: Reactor contents are circulated through the heating loop in order to maintain the reaction temperature. The heating loop contains a steam heater, and the steam flowrate is manipulated to supply just enough heat to keep the reactor contents at reaction temperature.

Phase 4: Reactor contents are discharged to a storage tank through a line which contains a cooler to cool the products down to ambient temperature.

The above operation of the batch process is accomplished with the scenario given in Table 4-1 for the operation of the valves present in the flowsheet.

<u>Time</u>	<u>V1</u>	<u>V2</u>	<u>V3</u>	<u>V4</u>	<u>V5</u>
t_0	open	open	close	close	close
t_1	close	close	open	close	close
t_2	close	close	close	open	close
t_3	close	close	close	close	open
t_4	close	close	close	close	close

Table 4-1: Valve Operation for Scenario

The system is designed as one composed of seven subsystems. The equations for each subsystem are as follows (see nomenclature).

Subsystem 1: Reactor Level

$$A_R \frac{dh_R}{dt} = f_{m,A} + f_{IaB} - f_{out}$$

Subsystem 2: Reactor Temperature

$$A_R C_V \frac{d(h_R T)}{dt} = Q_H + Q_R - Q_L$$

Subsystem 3: Heating Coil

$$Q_H = q$$

Subsystem 4: Component Concentrations

$$A_R \frac{d(h_R C_A)}{dt} = f_{IaA} C_{Au} - r - f_{out} C_A$$

$$A_R \frac{d(h_R C_B)}{dt} = f_{JaB} C_{Be} - r - f_{out} C_B$$

$$*R \frac{d(h_R C_C)/dt \cdot r = f_{out} C_C$$

$$T * A_R h_R k_C A_C B$$

$$k * k_U e^{-E/RT}$$

Subsystem 5: Heating Loop

$$Q_R = \Delta H_R r$$

$$Q_L = F S X S$$

$$Q_R = Q_L$$

Subsystem 6: Cooler

$$Q_C = f_{out} C_V < T - V$$

$$Q_C = F_{CW} C_{CW} \Delta T$$

Subsystem 7: Storage Tank

$$A_T \frac{dh_T}{dt} = f_{out}$$

$$A_T \frac{d(h_T C_{AT})}{dt} = f_{out} C_A$$

$$A_{BT} \frac{d(h_T C_{BT})}{dt} = f_{out} C_B$$

$$A_C \frac{d(h_T C_{CJ})}{dt} = f_{out} C_C$$

Note the model consists of both algebraic and ordinary differential equations.

Let us consider one of the four phases of the batch process for the moment and observe the behavior of the system. Let us choose phase 4 for this purpose. Phase 4 is the discharge phase and starts at time t_3 by opening valve V5 while keeping the rest of the valves in the system closed. In phase 4, the reactor contents are discharged to the storage tank through the cooler where they are cooled down to ambient temperature, as stated earlier. We observe that the subsystems that are active in this phase are subsystem 1, which represents the reactor level, subsystem 6, which represents the cooler, and subsystem 7, which represents the storage tank. The other four subsystems are inactive in phase 4 in the sense that the variables that appear in the equations modeling these subsystems do not change their values during this phase. This is one form of latency that is encountered in dynamic simulation. In this case, the equations modeling the inactive subsystems need not be considered in the solution process of the numerical integration.

On the other hand, we may expect that subsystem 7 follows a smoother trajectory than subsystem 1 in phase 4 because the time constant of the storage tank is likely to be larger than that of the reactor in magnitude. This is another form of latency in which case the integration step-size is determined by the subsystem with the smaller time constant, but the variables associated with the subsystem of the larger time constant are expected to change faster.

The time that phase 4 ends, t_4 , is determined by the value of variable h_4 which represents the reactor level. When h_R reaches the value zero, valve V5 is closed.

By observing the system in all the four phases we obtain the information given in Table 4-2. The last column in Table 4-2 gives the specifications that characterize a particular phase. The specifications simply show the values of the control variables that force the system to act in the desired manner.

<u>phase</u>	<u>period</u>	<u>active subsystems</u>	<u>observed variable</u>	<u>specifications</u>
1	$t_0 - t_1$	1	h_R	$f_{out} = 0, k_0 = 0, q = 0$
2	$t_1 - t_2$	2, 3	T	$f_{in,A} = 0, f_{in,B} = 0,$ $f_{out} = 0, V_0$
3	$t_2 - t_3$	4, 5	c_C	$f_{in,A} = 0, f_{in,B} = 0,$ $f_{out} = 0, V_0$
4	$t_3 - t_4$	1, 6, 7	h_R	$f_{in,A} = 0, f_{in,B} = 0,$ $K_0 = 0, q = 0$

Table 4-2: Dynamical Activity of Subsystems in Each Phase

5. A GENERAL MODEL

We consider here the solving of a general process modeled by mixed sets of both algebraic and ordinary differential equations of the form

$$dx/dt = f(x,z,u,t) \quad (1)$$

$$g\{x,z,u,t\} = 0 \quad (2)$$

$$u(t) \text{ given} \quad (3)$$

where

- x are "n" state variables equal in number to the equations 1
 2 are "m" "algebraic" variables equal in number to equations 2
 u are "r" control variables representing the degrees of freedom for our problem. As equation 3 indicates, they are specified versus time.

A more conventional discussion on process modeling would consider that equations 2 were used to eliminate z by writing

$$2 = g(x,u,t)$$

and rewriting 1 in the form

$$dx/dt = f(x,g(x,u,t),u,t) = f(x,u,t)$$

We shall however consider the implications of solving our model explicitly in the form of equations 1 to 3.

Firstly in our discussion, we note that the chosen numerical integration scheme will convert equations 1 into (often implicit) algebraic equations to be solved at the next time step. For example, if we chose to use the trapezoidal rule, we would write

$$\begin{aligned} h(x(t_{k+1}), u(t_{k+1}), x(t_k), u(t_k)) = x(t_{k+1}) - \\ [x(t_k) \cdot (At/2)\{f(x(t_{k+1}), z(t_{k+1}), u(t_{k+1}), t_{k+1}) \\ + f(x(t_k), z(t_k), u(t_k), t_k)\}] \end{aligned} \quad (4)$$

Other schemes create similar equations.

Knowing x, z and u at time t_R and u at t_{fc+1} we can solve equations 4 together with equations 2 at time t_{fc+1}

$$g(x(t_{k+1}), z(t_{k+1}), u(t_{k+1}), t_{k+1}) = 0 \quad (2')$$

as a set of n+m nonlinear algebraic equations in the n+m unknowns, $x(t_{fc+1})$ and $z(t_{k+1})$, using for example the Newton-Raphson method.

We note that equations 4 and 2' are in general implicit in variables $x(t_{k+1})$ and $z(t_{k+1})$. To aid in solving, values for these variables can be predicted as is done for only the state variables in a more conventional discussion. Prediction is usually done by fitting a polynomial in time to past behavior of x and z and using this polynomial to predict (extrapolate) the behavior at t_{k+1} . It would appear we may need therefore to save information on the previous behavior of not only the state variables, x , but also for all the algebraic variables, z . In a typical process the algebraic variables are far more numerous than the state variables (consider all the algebraic variables involved in evaluating physical properties). This requirement could in fact doom the approach we are about to suggest. Fortunately, we need usually only predict a very few of the algebraic variables by extrapolating their past behavior.

The algebraic equations are highly structured for a process and the algebraic variables and equations can be reordered such that their incidence matrix (see Westerberg at al³) is in the bordered triangular form shown in Figure 2. This reordering can be done automatically using any of several algorithms.

If we know values for only the m' *tear* variables, where m' is usually much smaller than m (the total number of algebraic variables, z), then we can use the first $m - m'$ algebraic equations to solve explicitly for the $m - m'$ *nontear* algebraic variables. These first $m - m'$ equations are lower triangular and thus can be solved directly using a forward substitution scheme. We would ignore the remaining m' *tear* equations in our prediction scheme.

This observation suggests we need only predict the m' *tear* variables using past behavior for them. The remaining ones can be predicted by $m - m'$ of the algebraic equations using a forward substitution scheme. Thus the "cost" of predicting is essentially the cost of saving past values for the state and algebraic *tear* variables, and then using the majority of the algebraic equations themselves to predict the remaining algebraic *nontear* variables. In terms of time required, prediction can be accomplished in a time similar therefore to the time of evaluating the right-hand side error vector for the Newton-Raphson scheme since the latter is an evaluation of equations 4 and 2. We discuss how this can be accomplished in practice.

As stated earlier, the *tear* variables can be found automatically using existing algorithms. However, since they are being found only to save storage space in terms of past values being needed for them, it hardly seems worth the effort to be

too careful to find a tear set having the fewest tear variables possible in it. Such complete automatic tearing algorithms are usually very expensive to execute both in computer time and space. Also, if a tear set is found for a set of equations, one is still faced with the problem of transforming the equations algebraically into a form suitable for fast forward elimination - i.e., one needs to rewrite each equation as it appears in the forward substitution scheme so the *new* variable occurring in it is found explicitly in terms of the other variables occurring in it. Doing this automatically using algebraic manipulation codes is so expensive as to be out of the question as an approach. Again only a good tear set need be found.

Suppose we analyze each unit model carefully to discover a set of tear variables for it, and we do this only the one time when the unit model is developed. We can then write a special prediction routine that can use the nontear equations to evaluate the algebraic nontear variables given predicted values for the state and selected tear variables for the unit. We write this prediction routine along with the routines to evaluate the Jacobian matrix and right-hand-side errors for the unit equations.

We can then have available for each such routine a list of the variables needed as input and a list of those predicted in terms of them. Predicting only the tear variables on these lists for all units and the state variables will then allow us to predict all remaining algebraic variables for an entire flowsheet. We can be cleverer than this approach and reduce the total number of tear variables needed if desired. A tear variable for a unit may already be predicted because it exists in a unit whose prediction routine has already been executed. If so, that variable can be dropped from the set of tear variables for the unit yet to have its variables predicted, and the space needed to save its past values can be eliminated. Discovering the variables which can be dropped can be done in a single preliminary pass through the prediction step. Clearly an optimal order for predicting variables for a flowsheet exists if we adopt this approach. (It is questionable however if the space saved will be worth the space and time needed for implementing such an ordering algorithm.)

6. THE STRATEGY AND THE ALGORITHM

Integration by predictor-corrector methods⁶ requires a prediction and up to a prespecified number of corrector iterations at every step of the integration process. In general, it is the corrector step that is computationally expensive. At the corrector step, the corrector equations, which are nonlinear in general, are solved using an appropriate iterative scheme - the Newton-Raphson method in our case. A

Newton-Raphson iteration corresponds to solving a set of linear equations obtained by a Taylor series expansion of the nonlinear equations around the current point. Gaussian elimination with LU factorization is the usual technique to solve the resulting linear equations. This comprises four computational stages: pivot selection, LU factorization, forward elimination, and backward substitution. Pivot selection is related to the structure of the coefficient matrix (the Jacobian matrix for nonlinear systems) of the system, and, since this structure does not change from step to step in integration, pivot selection is not performed at every step. There are two cases where pivot selection is redone: 1. If the decision variable set is changed, then repivoting is necessary since some pivots may no longer be variables in the problem. This occurs at the end of each phase in the example problem. 2. In the case of numerical singularity, where some pivots are no longer eligible because of the numerical values of the variables. LU factorization is done only when the coefficient matrix is regenerated. This is usually done in the following cases: 1. Some large amount of steps have been taken since the last regeneration, 2. Corrector iterations do not converge even when the step-size is reduced, 3. Step-size has changed significantly.

The numerical behavior of a unit in corrector iterations (e.g. the number of iterations needed for convergence, stages that can be skipped in one Newton-Raphson iteration, etc.) depends on how close the predicted values are to the corrected values, its interaction with other units, and the existence of a change in decision variables associated with it. For a system with latency, the numerical behavior of the units will be different from each other, by definition. The approach that we take in taking advantage of latency is based on exploiting the difference between the units in their numerical behavior in corrector iterations. This, of course, requires decomposing the flowsheet so that individual units can be treated separately. A code that is developed by Clark⁷, based on a paper by Westerberg and Berna⁸, gives us this capability.

The idea in the Westerberg and Berna decomposition strategy is to perform pivot selection and LU factorization stages of a Newton-Raphson iteration on a unit basis. In the pivot selection phase, pivots for a unit are selected only among its internal variables. The rows that are not assigned pivots are merged with the rows of other units in the same situation. The blocks that are obtained by this merging - the so-called residual blocks - are pivoted when they get large enough (according to memory space considerations). LU factorization, forward elimination and backward

substitution are performed in the same way.

Let us discuss the possible situations that might be seen during the calculations performed for a step of numerical integration in dynamic simulation. As stated earlier, the computational activity at a step comprises a prediction and several corrector iterations, where a corrector iteration corresponds to a forward elimination and a backward substitution. If the predicted values for the variables for a unit are the same as their values at the previous step, then the corrector equations for this unit will already be satisfied by the predicted values. In this case, there is no need for doing forward elimination for this unit.

Another situation that might be seen is the case where the predicted values are different than the values at the previous step, but the equation residuals are still zero within a tolerance. If the variables have not moved enough to require a re-evaluation of the Jacobian matrix for the Newton-Raphson equations, then we can suppress the forward elimination step for this case. The equation residuals for the predictor equations become the negative of the right-hand-side for their corresponding Newton-Raphson equations. If they are essentially zero, the right-hand-side after the forward elimination step will also be essentially zero.

In backward substitution the perturbations for the shared variables for a unit have already been determined before the equations for the unit are ready to be back substituted to find the predicted perturbations for the variables for the unit. If the unit was already skipped for forward elimination, implying its equation residuals are essentially zero, and if the predicted shared variable perturbations feeding back to the unit are also essentially zero, then the unit can be skipped for the backward substitution step.

Still another situation possible is the case where the value of a control variable changes at some point. In this case, all four stages of a Newton-Raphson iteration are performed for the unit to which the control variable belongs.

We also need criteria on the maximum number of allowable corrector iterations in one integration step and on the maximum number of allowable integration steps without Jacobian re-evaluation. Again, since the computational behavior of different units of a system with latency are expected to be different from each other, the maximum allowable corrector iterations and the maximum allowable integration steps without Jacobian re-evaluation will be different for different units.

At every step of the numerical integration process, the number of iterations needed for the convergence of corrector equations is counted for each unit. An average allowable number of corrector iterations is computed for each unit by averaging the number of iterations needed in the last several steps. If the number of iterations needed for convergence of the corrector equations of a unit at any step exceeds this average, then the portion of the Jacobian matrix that belongs to this unit is updated and an LU factorization is performed on that portion. At the same time, the frequency of Jacobian matrix evaluations is monitored and an average number of steps is computed to use as a limit on the maximum number of steps that can be taken without re-evaluating the portion of the Jacobian matrix belonging to that unit.

From these arguments we deduce the following algorithm for the solution of the corrector equations.

0. Initialization at the first step:
 - Set maxstep_i and maxiter_i to their initial values for each unit i .
 - Set stepcount_i to 1.
1. Set itercount_i to 0.
 - {The next two steps correspond to comments made at the end of the first paragraph in this section.}
2. Set $\text{itercount}_i = \text{maxstep}_i$
 - Update J_i
 - Perform LU factorization on J_i
 - Set stepcount_i to 1.
3. If $\text{itercount}_i = \text{maxiter}_i$ then
 - Update J_i
 - Perform LU factorization on J_i
 - Set itercount_i to 0.
4. Forward elimination:
 - If $\|Ax\| \leq c_{1\#}$ then skip unit i .
 - Otherwise, calculate resid_i

If $|\text{resid}_j| \leq \epsilon_2$, then skip unit i .

Otherwise, perform forward elimination on unit i .

5. Backward substitution:

If forward elimination was skipped for this unit and if $|\text{As}_j| \leq \epsilon_3$ for the shared variables, then skip unit i .

Otherwise, perform backward substitution on unit i .

6. If a forward elimination or a backward substitution is performed on unit i then increment itercount_i by 1.

7. Convergence check:

If not converged then go to 2

If converged then update maxiter_i , stepcount_i , and maxstep_i

7. AN EXAMPLE AND A COMPARISON

The flowsheet in Figure 1 is simulated using a predictor-corrector method with the Heun method being the predictor formula and the trapezoidal rule being the corrector formula⁹. Step-size change is allowed at every step but restricted to a maximum allowable step-size in order to display the solution frequently enough. The local truncation error, which is estimated by Richardson extrapolation, is used as the basis for step-size adjustment.

The Clark implementation of the Westerberg and Berna decomposition algorithm is used as the decomposition and linear equation solving code for both the conventional strategy and the new one proposed here. This implementation allows the calling program to request different steps of the Newton-Raphson scheme be performed selectively on selected subsystems.

The following observation was made during the simulation. Remember the process passes through 4 phases as described earlier. A total of 131 integration steps were taken to complete the simulation. Two steps were unsuccessful at the first two tries of phase 3 due to unacceptable truncation error. Phase 1 took 10 steps, each converging in one iteration with no need for forward elimination. That was because only one subsystem with a linear equation was active in phase 1. Phase 2 took 11 steps, again each converging in one iteration. Only one of the two subsystems that

were active in this phase required forward elimination. Phase 3 took 90 successful steps. At 73 of the 90 steps, convergence was achieved in 2 iterations and for 17 of them it was achieved in one iteration. The predictor accurately predicted the behavior of one of the two subsystems active in this phase. That subsystem required no forward elimination. Phase 4 took 20 steps, each converging in 3 iterations. Only one of the three subsystems that were active in this phase required forward elimination. Pivot selection and LU factorization had to be performed a total of 6 times, 4 of them being at phase changes and 2 at step failures. We note that, although we cite the results of a run where pivot selection was performed forcibly every time there was a change in the operation phase, this is not a requirement of the algorithm. In another test, the system itself successfully detected if a new pivot selection was required, even at operation phase changes.

<u>activity</u>	<u>conventional millisec</u>	<u>new millisec</u>
pivot selection	806	808
LU factorization	20307	11964
forward elimination	8226	4261
backward substitution	4226	1022
shared variable check	-	467
Jacobian evaluation	561	241
right-hand side generation	922	434
total time	35048	19197

table 7-1: Computation Time Comparisons of the Two Strategies

Table 7-1 gives statistics on the performance of the conventional strategy and the

proposed strategy. It is seen that the new algorithm reduces the CPU time by about 45%. One may estimate a maximum possible time saving potential for the steps of LU factorization, forward elimination and backward substitution for the Newton-Raphson scheme by taking the number of equations to be solved at each phase of operation as a basis. These estimates show that the CPU time could be reduced up to 60% for the example problem. The actual number obtained for the example is 48%. The difference of 12% between the estimated and the actual values reflects the overhead occurring in detecting if various steps can be skipped in the algorithm being proposed here.

Nomenclature

A_R	reactor cross-sectional area, given
A_T	storage tank cross-sectional area, given
c_A, c_B, c_C	concentrations of A, B, and C in reactor
c_{A0}, c_{B0}	concentrations of A and B in respective inlet streams
$c_{A,T}, c_{B,T}, c_{C,T}$	concentrations of A, B, and C in the storage tank
C_V	heat capacity of reactants and products, constant, given
C_{CW}	heat capacity of cooling water, given
E	activation energy for the reaction, given
f	rhs functions for state equations
$f_{in,A}, f_{in,B}$	inlet flowrates of A and B to reactor, volumetric, given
f_{out}	outlet flowrate from reactor, given
F_S	steam flowrate through the heater
F_{CW}	cooling water flowrate through the cooler
g	algebraic equations
h_R	reactor fluid level
V	storage tank fluid level
i	unit index
$itercount_i$	iteration counter for unit i in corrector iterations
J_{ij}	portion of Jacobian matrix belonging to unit i
k	time index
k	reaction rate constant
k_0	Arrhenius constant
m	number of algebraic equations, g , and algebraic variables, z
m'	number of algebraic <i>tear</i> variables
$maxstep_i$	maximum number of steps allowed for unit i without regenerating J_i

maxiter_i	maximum number of corrector iterations allowed for unit i
n	number of state variables, x
Q_H	heat supply rate to the reactor through the heating coil
Q_R	rate of heat absorbed by the reaction
Q_L	heat supply to the heater
r	number of control variables, u
r	rate of reaction
R	universal gas constant
resid_i	residuals of corrector equations belonging to unit i
s_i	variables shared with unit i and all other units
stepcount_i	number of steps take without regenerating J_i
t	time
T	reactor temperature
u	control variables
x	state variables
x_i	variables associated with unit i
ΔH_R	heat of reaction, constant, given
ΔT	allowable temperature increase for cooling water, given
z	algebraic variables
ϵ_1 to ϵ_3	tolerance parameters
λ_s	latent heat of steam, given

REFERENCES

1. Rabbat, N.B.G., Sangiovanni-Vincentelli, A.L. Hsieh, H.Y., "A Multilevel Newton Algorithm with Macromodeling and Latency for the Analysis of Large-Scale Nonlinear Circuits in the Time Domain/" *IEEE Trans. Circuits Systems*, Vol. CAS-26, No. 9, 1979, pp. 733-741.
2. Shampine, L.F., Gear, C.W., "A User's View of Solving Stiff ODE's," *SIAM Review*, 1979, pp. 21.
3. Westerberg, A.W., Hutchison, H.P., Motard, R.L., Winter, P., *Process Flow sheet ing*, Cambridge Univ. Press, Cambridge, England, 1979.
4. Alfonso, LL, *DYSCO: An Interactive Executive Program for Dynamic Simulation and Control of Chemical Processes*, PhD dissertation, Univ. of Mich., 1974.
5. Patterson, G.K., Rozsa, R.B., "DYNSYL: A General-Purpose Dynamic Simulator for Chgmical Processes," *Computers and Chem. Eng. J.*, Vol. 4, 1980, pp. 1-20.
6. Gear, C.W., *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, 1971.
7. Clark, P.A., "An Implementation of a Decomposition Scheme Suitable for Process Design Calculations," Master's thesis, Carnegie-Mellon Univ., Pittsburgh, PA 15213, 1980.
8. Westerberg, A.W., Berna, T.J., "Decomposition of Very Large Scale Newton-Raphson Based Flowsheeting Problems," *Computers and Chem. Eng. J.*, Vol. 2, 1978, pp. 61-63.
9. Kuru, S., *Dynamic Simulation with an Equation Based Flowsheeting System*, PhD dissertation, Carnegie-Mellon Univ., Pittsburgh, PA 15213, 1981.

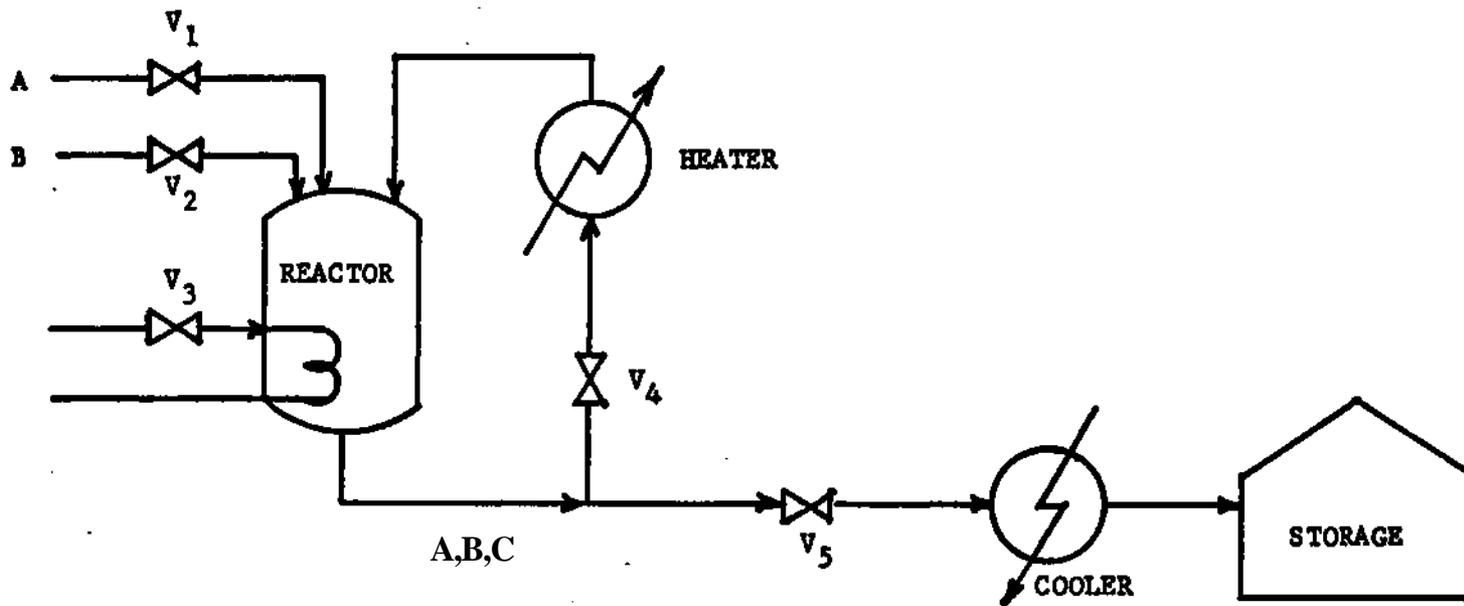


Figure 1: Flowsheet for a Batch Process

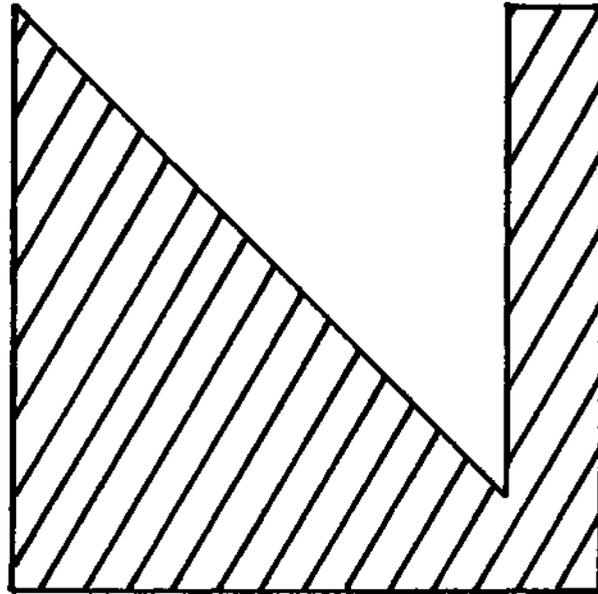


Figure 2: Bordered Lower Triangular Structure
for Unit Model Equations

