# Infeasible Path Optimization with Sequential Modular Simulators

Infeasible path optimization algorithms have been effective with equation-solving or simultaneous modular process simulators. Using these simulators, successive quadratic programming is applied to large nonlinear programs. However, the algorithms are not directly applicable to commercial simulators of the sequential modular type.

Here an infeasible path algorithm is presented which can easily be integrated with most commercial process simulators. The optimization problem remains small (10 or 20 variables) and the user merely replaces the convergence block with an "optimization" block.

A simple process example demonstrates the operation, effectiveness, and potential of this algorithm.

**L. T. BIEGLER and R. R. HUGHES**

**Chemical Engineering Department**
**University of Wisconsin**
**Madison, WI 53706**

## SCOPE

Optimizing a process model based on a preprogrammed process simulation system is often cumbersome and slow. Because of the rigid information flow structure, most optimization studies have treated the flowsheet as a "black box" whose design variables are manipulated by an optimization algorithm. For each setting a complete simulation is performed to evaluate the objective function and constraints. Many process relations are highly nonlinear and even discontinuous, so that successful optimization often requires a robust method, such as the direct search methods of Box (1965), Hooke and Jeeves (1961), and Gall (1966); unfortunately, these are also among the least efficient optimization techniques.

In developing Quadratic Approximation Programming (QAP), Parker and Hughes (1981) have shown how gradient and other derivative information can be obtained from the flowsheet *without* resimulating the process for each perturbation. However, a converged flowsheet is still required for each base point and also for certain trial points. The modification by Biegler and Hughes (1981) termed Quadratic/Linear Approximation Programming (Q/LAP), used *successive quadratic programming* (Powell, 1977) as its nucleus. The method proved to be very effective in quickly optimizing realistic *sequential modular process simulations.*

Successive quadratic programming also has other desirable features. Because it seeks a stationary point of the Lagrangian, this method automatically converges equalities as it approaches the optimum. For chemical process optimization this feature means that the algorithm guarantees a converged flowsheet at the optimum, even if convergence of the flowsheet is not achieved for each function evaluation at trial variable values. This concept of simultaneous simulation convergence and optimization is termed *infeasible path optimization.* At each iteration, only linear approximations of the equality constraints are satisfied; if, as is usually the case, the problem includes nonlinear equality constraints, only the optimal point will be truly feasible.

Berna, Locke and Westerberg (1980) made the first application of this concept to chemical process simulators. At the heart of their algorithm is the successive quadratic programming method proposed by Powell (1977). These investigators used an equation solving simulator as their framework. Consequently, the optimization problem involves thousands of equations, and requires sparse matrix decomposition before the quadratic program is solved. Berna et al. applied their method to a small 10-variable, 31-constraint alkylation process with only 3 degrees of freedom. No results have yet been reported with this scheme on larger problems.

Jirapongphan (1980) applied the infeasible path concept to simultaneous modular simulators. Using, first, *linearized* models and, then, *reduced* analytic nonlinear models, he developed two algorithms with successive quadratic programming as their nucleus. Jirapongphan et al. (1980) reported remarkably short optimization times for small flash processes and an ammonia synthesis process simulated on FLOWTRAN.

However, no infeasible path algorithm has yet been applied to the large body of commercial sequential modular software. This paper presents an algorithm for such systems, and reports some computational results.

## CONCLUSIONS AND SIGNIFICANCE

An infeasible path optimization algorithm, IPOSEQ, has been developed by applying the Wilson-Han-Powell method of successive quadratic programming to an equality-constrained simulation in terms of both design and tear variables. From limited calculations with a simple flash example, IPOSEQ seems to be competitive with Q/LAP and far superior to direct search methods such as CPX.

The infeasible path algorithm is effective, robust and very easy for the engineer to use. The main advantages of this algorithm are its easy interface with many sequential modular simulators and its efficient performance because it optimizes and converges at the same time. Further improvements and testing should lead to an algorithm that permits frequent and efficient optimization studies with sequential modular simulators.

Present address of L. T. Biegler: Department of Chemical Engineering, Carnegie-Mellon University, Pittsburgh, PA 15213.

0001-1541-82-5880-0994-$2.00. © The American Institute of Chemical Engineers, 1982.

## CHEMICAL PROCESS SIMULATION AND OPTIMIZATION

Chemical process simulation requires the solution of square systems of equations, where the number of variables equals the number of equations. Optimization is a superset of this problem; the number of variables exceeds the number of equations, so that adjustment is possible to minimize (or maximize) an objective function, $\phi$, while still satisfying the equations.

In this paper, we distinguish $y$ and $x$ as vectors of process and design variables, respectively. Mathematically, these can be lumped into a single variable vector, but, for a process engineer, the process variables are values which are normally found by solving material and energy balance equations, while the design variables are values set by the engineer to specify the process design. We also distinguish a design equality constraint vector, $c$, from the process equation vector, $h$, which represents the material and energy balances. Thus, $h$ and $y$ have the same number of elements, while $c$ has fewer elements than $x$; the difference represents the degrees of freedom available for optimization. Both $x$ and $c$ are usually quite small (<100 elements) while $h$ and $y$ can have thousands of elements. The vector $g$ consists of any inequality constraints, which are required by the problem.

With these definitions, the process problems can be stated as:

| Simulation | Optimization |
|---|---|
| At preset $x^0$, force $h(x^0,y) = 0$ | $\min_{x,y} (\phi(x,y))$ subject to $g(x,y) \le 0$, $h(x,y) = 0$, $c(x,y) = 0$ (1) |

The feasible path procedure used in previous studies solves inner simulation loops to satisfy $h(x,y) = 0$ at fixed values of $x$, and then adjusts $x$ in the outer optimization loop. This strategy characterizes black box methods that use direct search algorithms as well as simple one-at-a-time case studies. Although QAP and Q/LAP evaluate gradients without converging flowsheets for each perturbation, they also require full simulations for each base point and some trial points.

If gradients are available the two loops can be executed simultaneously simply by solving an equality-constrained optimization problem. Using successive quadratic programming, we can construct an algorithm that parallels the convergence algorithm. Consider first the termination criteria for the simulation and optimization problems:

*Simulation* (satisfy equations)

$$h(x^0,y) = 0 \tag{2}$$

*Optimization* (Kuhn-Tucker conditions)

$$\nabla\phi(x,y) + u\nabla g(x,y) + v\nabla h(x,y) + t\nabla c(x,y) = 0 \tag{3}$$

$$ug(x,y) = 0;\ u \ge 0;\ g(x,y) \le 0 \tag{4}$$

$$h(x,y) = 0;\ c(x,y) = 0 \tag{5}$$

Except for the inequalities of Eq. 4, these criteria consist of sets of nonlinear equations. Their solution by a Newton-type iteration leads to similar matrix equations:

*Simulation*

$$J(x^0,y)\cdot\Delta y = -h(x^0,y) \tag{6}$$

*Optimization*

$$\begin{bmatrix} B_{xx} & B_{xy} & \nabla_x g & \nabla_x h & \nabla_x c \\ B_{xy} & B_{yy} & \nabla_y g & \nabla_y h & \nabla_y c \\ \nabla_x g_A^T & \nabla_y g_A^T & 0 & 0 & 0 \\ \nabla_x h^T & \nabla_y h^T & 0 & 0 & 0 \\ \nabla_x c^T & \nabla_y c^T & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ u \\ v \\ t \end{bmatrix} = - \begin{bmatrix} \nabla_x\phi(x,y) \\ \nabla_y\phi(x,y) \\ g_A(x,y) \\ h(x,y) \\ c(x,y) \end{bmatrix} \tag{7}$$

Here $g_A$ are the "active" elements in $g$, for which $u > 0$. The matrix $J(x^0,y)$ is the Jacobian (with respect to $y$) of $h(x^0,y)$, or its approximation, depending on the simulation approach. The matrix

$$\begin{bmatrix} B_{xx} & B_{xy} \\ B_{xy} & B_{yy} \end{bmatrix}$$

is the Hession of the Lagrangian of the problem specified by Eqs. 1, or its approximation. Note that recursive solution of Eq. 7 satisfies automatically Eqs. 3 and 5. Equation 4 can be satisfied by identifying the active inequality constraints, prior to solving each iteration of Eq. 7. This occurs automatically when a suitable quadratic programming problem is solved.

With equation solving simulators the number of process variables in Eq. 6 is on the order of 10000 or more. The number of variables in Eq. 7 is about twice this value. To avoid solving a quadratic program this large, Berna et al. (1980) decomposed Eq. 7 by using symbolic reduction and L/U decomposition. This leads to a quadratic program in $x$-space with only inequality constraints present. The reduced quadratic program is solved to evaluate $\Delta x$ and $u$; then $\Delta y$, $v$ and $t$ are found by back-substitution in Eq. 7. Because the system is large and sparse none of the matrices are actually multiplied or inverted and sparse L/U factorization is used throughout. The $B$ matrix, an approximated Hessian updated by the quasi-Newton BFGS formula, is large and dense, however. To avoid prohibitive storage requirements, Berna et al. store the updates and execute a symbolic reduction before computing the Hessian for the reduced quadratic program.

Jirapongphan (1980) uses a similar matrix decomposition strategy in his simultaneous modular optimization. Although the number of equations and variables is much less for this approach, the matrix of Eq. 7 is still large enough to require decomposition.

By applying successive quadratic programming and matrix decomposition, both Berna et al. and Jirapongphan were able to optimize their simulated processes efficiently with CPU times of the same order-of-magnitude as the simulations. Jirapongphan et al. (1980) mention another simultaneous modular method that combines their optimization strategy with the inside-out flash algorithm (Boston and Britt, 1978). The result is an efficient method that requires very few modular calculations.

## SEQUENTIAL MODULAR OPTIMIZATION

In the sequential modular environment, the only process equations and variables that need to be considered are those in the tear set. The fixed calculation order provides solution of all the other process equations within modules or as stream interconnections. Even for a detailed process model, the number of tear variables may be as small as 10 to 30, and common iterative techniques (Kehat and Shacham, 1973) can be used to solve Eqs. 2.

Perkins (1979) formulated, with a sequential modular simulator, an *extended* design problem:

$$\begin{aligned} c(z,y) &= 0 \\ h(z,y) &= 0 \end{aligned} \tag{8}$$

Here $z$ is a vector of design (or control) variables which are adjusted to satisfy the design constraints, $c(z,y)$ [i.e., $z$ is a subvector of $x$, above]. Perkins used a Broyden (1968) algorithm to solve the design and tear Eqs. 8 simultaneously. With a few design constraints, and modules which require lengthy computations, this approach is often superior to the use of controller blocks. It is especially valuable when there is strong interaction between the design and tear variables.

As a further extension, one can treat the optimization problem as a large design problem. Instead of Eqs. 8, the Kuhn-Tucker conditions must be satisfied. As described above, this can be done by applying successive quadratic programming as in Eq. 7. In the *sequential* modular environment, however, the system of equations *and* the quadratic program remain small. Therefore, with an efficient quadratic programming package, the matrix in Eq. 7 need not be decomposed, since only the equations for $c$, $g$ and $h$ are included.

In many simulators, the convergence calculations are performed in separate modules, interfaced to the model by insertion at the appropriate tear positions. Optimization can also be interfaced in this manner. Without affecting the system executive or other modules, the entire optimization algorithm may be programmed and tested separately. Acting as an extended convergence block, the optimization module then receives function and gradient information and manipulates tear and design variables with each pass through the simulation.

For simulators with a convergence block, the engineer simply substitutes the optimization block. Usually the tear sets and calculation order remain unchanged. He then chooses an objective function, design variables, dependent variables and constraints, and the algorithm simultaneously converges and optimizes, presenting the engineer with a feasible *and* optimal simulation in one run. Even if the simulator normally handles convergence within its executive program, it may be possible to insert an infeasible-path optimization block, and lock out the built-in convergence algorithm.

To achieve the same high performance as the other infeasible path methods, an efficient method must be used for gradient calculation. In the equation solving environment all equations have analytic gradients. Simultaneous modular methods either calculate gradients by perturbing each module or by constructing nonlinear reduced analytic models. For the sequential-modular case, gradients can be calculated in either of two ways:

First, all gradients may be specified or calculated for each equation within each module. All relationships between equipment, stream, cost and physical property variables must be examined and gradients specified. Clearly, this is a time-consuming task. Moreover, when nonsmooth functions are present, calculation of gradients is impossible. Finally, complete specification of gradients tends to destroy the "black box" characteristics of modules and the general nature of sequential modular simulation.

The second alternative is to calculate all gradients using numerical perturbation. Here we can decompose the flowsheet and use linear models as in the Q/LAP algorithms. A simpler method that involves much less bookkeeping and fewer variables is to perturb each tear and design variable and evaluate the response by passing through the calculation loop. This scheme involves slightly more subroutine calls than use of Q/LAP linear models, but sparse matrix decomposition is avoided and the sequential modular information flow is the same for perturbation as for convergence.

Numerical perturbation is obviously much slower than complete prespecification of gradients. However, evaluating gradients by perturbation preserves the "black box" nature of the process model and leads to an algorithm that is easily applied to any sequential modular simulator.

## INFEASIBLE PATH ALGORITHM

In this section, the algorithm for *i*nfeasible *p*ath *o*ptimization of *seq*uential modular systems with numerical perturbation (IPOSEQ) is formally stated. As in Q/LAP, the algorithm is centered around the successive quadratic programming algorithm (VFO2AD) of Powell (1977). Details of the modifications to that algorithm are not included here; instead, we concentrate on the steps pertaining to function and gradient calculation. Note that we redefine $y$ as the guessed tear variables and $h$ as the difference between guessed and calculated tear variables.

1. Choose an objective function, $\phi$, design variables, $x$, and process constraints, $g$ and $c$, for the simulated process. Use of the vector $c$ eliminates the need for control modules in the simulation. Identify dependent variables, $r$, whose values must be recovered from the module retention vectors to calculate the objective and constraint functions.

2. Choose a calculation sequence that contains all design and dependent variables and all process loops within a single calculation loop. This step is needed so that perturbation can be performed in the appropriate sequence. Frequently, the optimization problem is posed such that the calculation sequence from the simulation problem may be used.

3. Identify tear streams to eliminate recycle calculations, and choose tear variables, $y$. For vapor-liquid systems in equilibrium containing $N_c$ components, each tear stream introduces $(N_c + 2)$ tear variables—the component flows, the stream pressure, and the specific enthalpy. If, at some point within the loop, the pressure is used as a design variable or is fixed as an equipment parameter, perturbing the assumed tear pressure will obviously produce no response. In this case, pressure could be eliminated as a tear variable, since change in pressure will occur through manipulation of design variables.

4. Initialize the design variables, $x = x^1$ and the tear variables, $y = y^1$.

5. Proceed through the process module calculations in the specified sequence to update the dependent variables, $r(x,y)$ and the *calculated* tear variables, $w(x,y)$. (As an option, if good initial values of $y$ are not available, step 5 can be repeated, with $y^i$ reset to $w$.)

6. Calculate $\phi(x^i,r)$, $g(x^i,r)$, $c(x^i,r)$, and $h(x^i,y) \equiv y^i - w(x^i,y^i)$. Set the iteration index $i = 1$.

7. Starting from the last module, search backward along the calculation sequence until an *un*perturbed design variable, $x_j^i$, is identified as a module input. Perturb this design variable, and recalculate the module and those following to the end of the sequence. Note that dependent variables in modules preceding the design variable remain unaffected. Evaluate the gradients:

$$\frac{\partial \phi}{\partial x_j} = [\phi(x^i(I + \xi E), r^*) - \phi(x^i,r)]/\Delta x_j$$

$$\frac{\partial g}{\partial x_j} = [g(x^i(I + \xi E), r^*) - g(x^i,r)]/\Delta x_j$$

$$\frac{\partial c}{\partial x_j} = [c(x^i(I + \xi E), r^*) - c(x^i,r)]/\Delta x_j$$

$$\frac{\partial h}{\partial x_j} = [h(x^i(I + \xi E), y^i) - h(x^i,y^i)]/\Delta x_j$$

where $r^*$ = dependent variables after perturbation
$\xi$ = perturbation factor $\equiv \Delta x_j/x_j^i$ (dimensionless)
$E$ = index matrix, where $e_{kk} = 1$ for $k = j$, and other elements of $E$ are zero

Restore $x_j^i$ to its unperturbed value.
Repeat Step 7 for all variables in $x$.

8. Perturb each tear variable in turn, and recalculate the modules in sequence, beginning with the module for which the associated tear stream is an input. From the recalculated dependent variables, $r^*$, and output tear variables, $w(x^i,y^i(I + \xi E))$, evaluate the gradients:

$$\frac{\partial \phi}{\partial y_j} = [\phi(x^i,r^*) - \phi(x^i,r)]/\Delta y_j$$

$$\frac{\partial g}{\partial y_j} = [g(x^i,r^*) - g(x^i,r)]/\Delta y_j$$

$$\frac{\partial c}{\partial y_j} = [c(x^i,r^*) - c(x^i,r)]/\Delta y_j$$

$$\frac{\partial h}{\partial y_j} = [h(x^i,y^i(I + \xi E)) - h(x^i,y^i)]/\Delta y_j$$

9. If $i = 1$, set $B^i = I$
If $i > 1$, update $B^i$ using the BFGS equation (Dennis and Moré, 1977; Powell, 1977).

10. Solve the following quadratic program:

$$\begin{aligned} \text{Min} \quad & \nabla\phi(x^i,r)^T d + \frac{1}{2} d^T B^i d \\ s.t. \quad & g(x^i,r) + \nabla g(x^i,r)^T d \leqq 0 \\ & c(x^i,r) + \nabla c(x^i,r)^T d = 0 \\ & h(x^i,y^i) + \nabla h(x^i,y^i)^T d = 0 \end{aligned}$$

Here the $\nabla$ operator is $\left\{\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right\}$

The results are a search vector, $d$, in confined $(x,y)$ space, and shadow price vectors $u$, $t$, and $v$, for $g$, $c$, and $h$, respectively.

11. If $|\nabla\phi(x^i,r)^T\cdot d| + |u\cdot g(x^i,r)| + |t\cdot c(x^i,r)| + |v\cdot h(x^i,y^i)| \leq \epsilon$, stop. [The point $(x^i,y^i)$ is then within a tolerance $\epsilon$ of a Kuhn-Tucker point, where $\epsilon$ has the same dimensions (units) as the objective, $\phi$.]

12. Use Powell's algorithm (1977) to make a line search with a Lagrangian and/or an exact penalty function, to find the stepsize, $\lambda$, along $d$ that yields an improved point. Each point along the line search requires a pass through the calculation sequence [as in Step 5] to permit evaluation of the functions:

$$\phi(z),\ g(z),\ c(z), \text{ and } h(z), \text{ where } z \equiv \begin{bmatrix} x^i \\ y^i \end{bmatrix} + \lambda d$$

If there is no improved point, set $B^i = I$, and return to Step 10.

13. Update for the next iteration:

$$\text{Set } \begin{bmatrix} x^{i+1} \\ y^{i+1} \end{bmatrix} = \begin{bmatrix} x^i \\ y^i \end{bmatrix} + \lambda d$$

Retrieve the function values for the point from Step 12.
Increment $i \rightarrow i + 1$
Return to Step 7.

## ADJUSTABLE PARAMETERS FOR INFEASIBLE PATH OPTIMIZATION

The optimization parameters to be specified by the user for the above algorithm consist of the Kuhn-Tucker tolerance, scale factors and perturbation sizes for the design and tear variables. In selecting these values, the user should keep in mind that the infeasible path approach is actually a hybrid of two algorithms. In the absence of any bounds or constraints the algorithm becomes an unconstrained quasi-Newton optimization method. If there are no degrees of freedom, it becomes Newton's method for solving a square system of equations in the space of the tear variables.

The Kuhn-Tucker tolerance, $\epsilon$, used in Step 10 of the algorithm, has the same units as the objective function. As in most methods, it indicates the allowable uncertainty in the optimal *function* value. The resulting uncertainty in the optimal *variable* values depends on the sensitivity of the objective and constraint functions to the design variables, and may be much larger. Moreover, for the infeasible path method, the Kuhn-Tucker tolerance also controls the recycle convergence. It is probably best to keep $\epsilon$ small, unless past experience with similar problems on the same simulator suggests possible difficulties in terminating the calculation.

In principle, methods based on successive quadratic programming should not require scaling. However, with poor scaling, the initialization of the Hessian as the identity matrix (Step 9) may be an ill-conditioned choice, especially if large initial step sizes are allowed. The best solution is to have the user specify a vector of scaling factors (integral powers of 10) so that the gradient vector, at least initially, will have elements all of the same order-of-magnitude. Elements of the scale vector divide into corresponding design or tear variable elements and multiply corresponding elements of all gradients. It is especially important that a balance be kept between design and tear variables so that excessive movement of one set does not underemphasize movement of the other.

Choosing appropriate perturbation factors is easier than with other methods. Because the tear equalities are not satisfied at *each* iteration there is no recycle convergence error to affect the gradients. Iterative calculations in the modules, as in flash algorithms, generally have rather tight tolerances. Finally, the sequential nature of evaluating perturbation responses tends to "smooth out" nonlinearities, because linearity between input and output streams is no longer assumed, and linear coefficients are not calculated. Thus, derivatives can usually be calculated accurately with small perturbations, just big enough to guard against accumulation of round-off in the calculation sequence.

## EXAMPLE—A SIMPLE FLASH PROBLEM

### Description

A simple flash problem serves to demonstrate the performance of the infeasible path algorithm and the effect of the adjustable calculation parameters. The process flowsheet is presented as Figure 1. A light hydrocarbon feed is mixed with recycled bottoms and flashed adiabatically. The vapor is removed as a product, and the liquid split into a bottoms product and the recycle, which is pumped back to the feed.

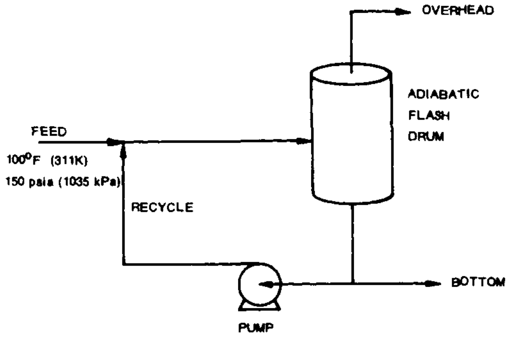

**Figure 1. Simple flash process flowsheet.**

***FEED* (kg-mol/h)**

| | |
|---|---|
| Propane | 10 |
| 1-Butene | 15 |
| N-Butane | 20 |
| Trans-2-Butene | 20 |
| Cis-2-Butene | 20 |
| Pentane | 10 |

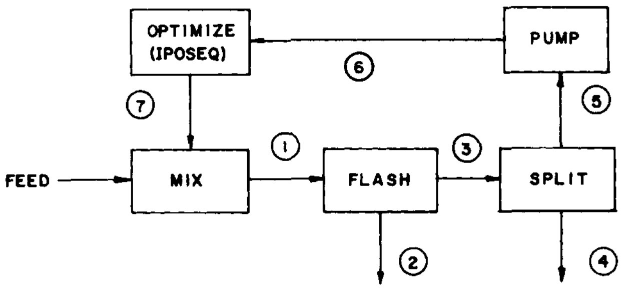

**Figure 2. Flash problem—simulation on SPAD.**

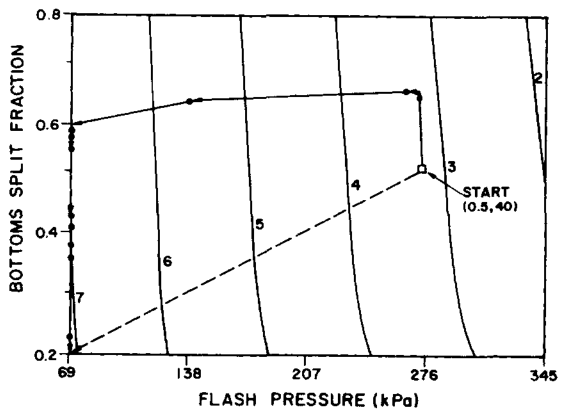

**Figure 3. Flash problem—monotonic response surface. Calculation paths: —— Run 4—poorly scaled - - - Run 5—well scaled**

TABLE 1. FLASH PROBLEM SPECIFICATIONS

| Objective Function | Constraints and Bounds |
|---|---|
| *Monotonic:* | $0.2 \le x_1 \le 0.8$ |
| $\max_{x,y} \{e_1\}$ | $69 \le x_2 \le 345$ |
| | $0 \le y_i \le 100\ (i = 1,6)$ |
| *Ridge:* | $0 \le y_7 \le 10^4$ |
| $\max\ [e_1^2 e_2 - e_1^2 - e_3^3 + e_4 - \sqrt{e_5}]$ | $y_j - w_j \equiv h_j = 0\ (j = 1,7)$ |

Independent Variables

$x_1$—Split fraction—Stream ④/Stream ③
$x_2$—Adiabatic flash pressure, kPa
$y_i (i = 1,6)$—Component flows—Stream ⑦ (kg-mol/h)
$y_7$—Specific enthalpy—Stream ⑦ (w/kg-mol)

Dependent Variables

$e_i\ (i = 1,6)$—Component flows—Stream ② (kg-mol/h)
$w_i\ (i = 1,6)$—Component flows—Stream ⑥ (kg-mol/h)
$w_7$ —Specific enthalpy—Stream ⑥ (w/kg-mol)

Initialization (Standard Procedure)

$x$—Tabulated starting point values
$y$—Direct substitution (from $w$), *after* a single pass through the simulation with $x = x$ and $y = 0$.

Scaling—Runs were made with one of the following sets of scaling factors for the independent variables:

| Set | $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ | $y_7$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 100 | 1 | 1 | 1 | 1 | 1 | 1 | 100 |
| 2 | 1 | 100 | 100 | $10^3$ | $10^3$ | $10^3$ | $10^3$ | $10^3$ | $10^4$ |
| 3 | 1 | 100 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | $10^3$ | 10 | 1 | 1 | 1 | 1 | 1 | 1 | 100 |
| 5 | 100 | 100 | 1 | 1 | 1 | 1 | 1 | 1 | 100 |
| 6 | 1 | 10 | 1 | 1 | 1 | 1 | 1 | 1 | 100 |
| 7 | 1 | 10 | 10 | 10 | 10 | $10^5$ | 100 | 100 | $10^4$ |
| 8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

This process was simulated and optimized on SPAD (Simulator for Process Analysis and Design) (Hughes, 1974), a small process simulator installed on the Univac 1100/82 at the Madison Academic Computing Center. With this simulator, processes can be calculated cheaply and alterations in the executive and optimization algorithms made easily. Figure 2 shows a block diagram of the simulated process. Note that instead of a convergence block for the recycle stream, an optimization block is specified.

The problem specifications are given in Table 1. There are *two* bounded design variables: the fractional split to the bottoms product, and the flash pressure; and *seven* tear variables, with the tear equations as equality constraints. Tear stream pressure is not included as a variable, because the loop pressure is fixed by the flash pressure. Two objective functions were tested. The first is *monotonic*, and has an optimum at the *lower* bounds of the design variables. It has only a slight dependence on the split fraction. The second objective is an arbitrary nonlinear combination of the first five components in the vapor stream. With this objective, the optimum lies on the *upper* bound of the split fraction, at an intermediate flash pressure, where the response surface has a *ridge*. Response surfaces of these two objective functions in the feasible design variable space are plotted in Figures 3 and 4.

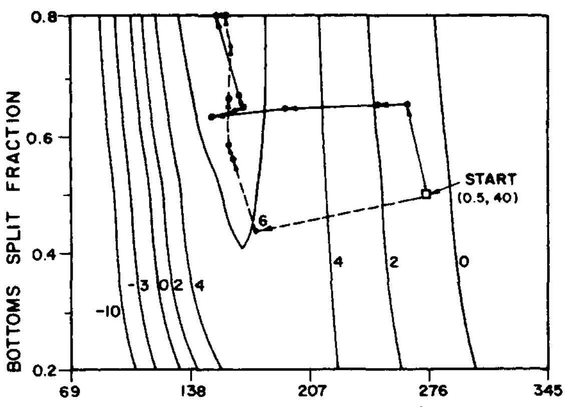

**Figure 4. Flash problem—ridge response surface. Calculation paths: —— Run 13—poorly scaled --- Run 14—well scaled**

## Results

Calculations were made with both objective functions, at various perturbation sizes, scale factors, and starting points. The results are reported in Table 2. Optimization was successfully achieved from *two* starting points, with *both* objectives, at different scale factors and perturbation sizes. The most efficient calculation, with the monotonic objective, required only 6.62 seconds or 2.58 STE's. (A Simulation Time Equivalent or STE is defined as the CPU run time divided by some basis simulation time.) The best run for the ridge objective was longer (12.86 s), especially in terms of STE's

TABLE 2. INFEASIBLE PATH OPTIMIZATION OF FLASH PROBLEM WITH IPOSEQ

| Run No. | Starting Point | Perturb'n Size | Scale Set | No. of Iter. | CPU Time (s) | Total STE's[a] | Best objective value found | Kuhn-T error |
|---|---|---|---|---|---|---|---|---|
| Monotonic Objective | | | | | | | | |
| *Successful Runs* | | | | | | | | |
| 1 | 0.4,30 | $10^{-3}$ | 1 | 24 | 27.55 | 10.75 | | $<10^{-5}$ |
| 2 | | | 2[b] | 5 | 6.660 | 2.60 | | $<10^{-4}$ |
| 3 | 0.5,40 | $10^{-2}$ | 1 | 20 | 22.679 | 8.85 | 7.233[c] | $<10^{-5}$ |
| 4 | | $10^{-3}$ | 1 | 21 | 23.879 | 9.32 | | $<10^{-4}$ |
| 5 | | | 2[b] | 5 | 6.620 | 2.58 | | $<10^{-4}$ |
| *Unsuccessful Runs* | | | | | | | | |
| 6 | 0.5,40 | $10^{-4}$ | 1 | 12 | 13.670 | 5.33 | 7.085 | $<10^{-5}$ |
| 7 | | | 3 | 3 | 3.262 | 1.27 | 3.268 | $<10^{-3}$ |
| 8 | | $10^{-3}$ | 4 | 8 | 8.917 | 3.48 | 7.089 | $<10^{-4}$ |
| 9 | | | 5 | 8 | 8.912 | 3.48 | 7.089 | $<10^{-4}$ |
| 10 | | | 6 | 8 | 8.872 | 3.40 | 7.090 | $<10^{-3}$ |
| Ridge Objective | | | | | | | | |
| *Successful Runs* | | | | | | | | |
| 11 | 0.4,30 | $10^{-3}$ | 5 | 16 | 19.044 | 28.21 | 6.937[c] | $<10^{-3}$ |
| 12 | | | 7[b] | 14 | 17.366 | 25.73 | | $<10^{-3}$ |
| 13 | 0.5,40 | | 5 | 19 | 22.845 | 33.84 | | $<10^{-4}$ |
| 14 | | | 7[b] | 10 | 12.856 | 19.05 | | $<10^{-3}$ |
| *Unsuccessful Run* | | | | | | | | |
| 15 | 0.4,30 | $10^{-3}$ | 8 | 17 | 20.722 | 30.70 | 4.888 | $<10^{-3}$ |

[a] STE's = CPU time/$\tau$, where $\tau$ = CPU time for simulation at the optimum. $\tau$ = 2.563 s for monotonic obj., 0.675 s for ridge obj.—both starting at $y = 0$.
[b] With indicated scale factors, *scaled* gradient components at the starting point satisfy the conditions $10 \le |(\nabla\phi)_j| \le 100;\ j = 1,9$.
[c] Optimal values.

TABLE 3. COMPARISON OF ALGORITHM PERFORMANCE ON FLASH PROBLEM BEST VALUES

| Starting Point | Problem | Algorithm | CPU Time (s) | No. of Iter.* |
|---|---|---|---|---|
| 0.5,40 | Monotonic | IPOSEQ | 6.620 | 5 |
| 0.5,40 | Monotonic | Q/LAP | 7.090 | 2 |
| 0.5,40 | Monotonic | CPX | 252.115 | 36 |
| 0.5,40 | Ridge | IPOSEQ | 12.856 | 10 |
| 0.5,40 | Ridge | Q/LAP | 28.981 | 10 |
| 0.5,40 | Ridge | CPX | 119.873 | 37 |
| 0.4,30 | Monotonic | IPOSEQ | 6.660 | 5 |
| 0.4,30 | Monotonic | Q/LAP | 6.426 | 2 |
| 0.4,30 | Monotonic | CPX | 49.279 | 26 |
| 0.4,30 | Ridge | IPOSEQ | 17.366 | 14 |
| 0.4,30 | Ridge | Q/LAP | 12.811 | 6 |
| 0.4,30 | Ridge | CPX | 136.468 | 46 |

* A CPX (Complex Search) iteration is defined as the evaluation of one new simplex.

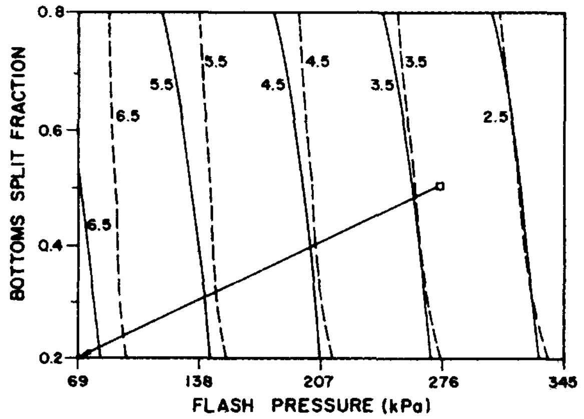

**Figure 5. Contour approximation—run 5—monotonic, well scaled.**
**—— Iteration 1**
**- - - Actual response surface**

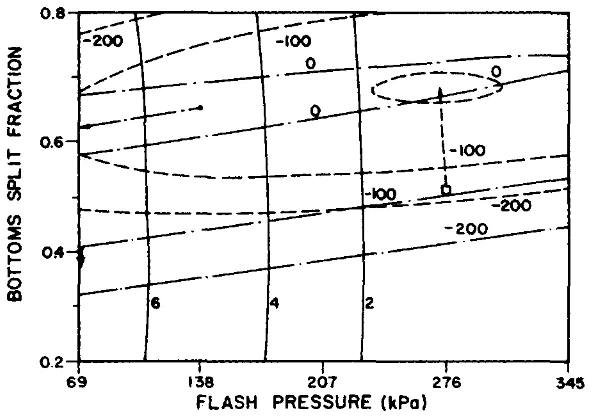

**Figure 6. Contour approximation—run 4—monotonic, poorly-scaled.**
**- - - Iteration 1**
**– – Iteration 5**
**—— Iteration 12**

(19.05). However, the latter figure is partly an artifact caused by the choice of basis simulation times. Some possible candidates for this basis are:

| | | |
|---|---|---|
| Convergence Tolerance | $10^{-3}$ | $10^{-4}$ |
| Simulation Times (s) | | |
| Starting Points—(0.4,30) | 0.941 | 1.203 |
| —(0.5,40) | 0.648 | 0.980 |
| Monotonic Optimum | 1.575 | 2.563* |
| Ridge Optimum | 0.513 | 0.675* |

The * identifies the values used. All values tabulated are for calculations with initial tear variable values of zero. It is apparent that the monotonic optimum occurs at a point where convergence is especially slow.

In Table 3, the best runs from Table 2 are compared with similar calculations using two other algorithms: —Q/LAP (Biegler and Hughes, 1981, and CPX, a modification of Complex Search (Box, 1965). The latter treats the simulation as a "black box" function generator, and is a good example of a direct search optimization algorithm. As expected, both IPOSEQ and Q/LAP are much faster than CPX. For runs from the first starting point (0.5,40), IPOSEQ was faster than Q/LAP; from the other starting point (0.4,30), Q/LAP was the best. Based on this limited example, we infer that the efficiencies of the two algorithms are about the same.

## ANALYSIS OF IPOSEQ PERFORMANCE

### Variable Scaling

From study of Table 2, it is clear that the scale factors play a major role in controlling the optimization of the example problem. The fastest of the successful calculations—runs 2, 5, 12, and 14—occurred when the initial *scaled* objective function gradient elements all had absolute values between 10 and 100. This heuristic keeps the Hessian approximation well-conditioned and leads to good initial movement with an initial Hessian approximation of *I*.

The significance of scaling with this algorithm is not unexpected. Many variables with different units and effects are involved. In particular, the user must strike a workable balance between convergence of the tear variables and optimization with the design variables. As an extreme, consider Run 7, which uses scale set 3. Here, design variable movement is completely *under*emphasized and the tear variables quickly converge to a feasible, but nonoptimal point.

With somewhat better scaling, the algorithm may find the optimum, but only after quite a few iterations. To illustrate this, calculation paths for Runs 4 and 5 are included in Figure 3. Because Run 4 is poorly scaled, initial movement is *away* from the optimum. In Run 5, on the other hand, the calculation moves directly to the lower variable bounds and in 4 more iterations converges to a feasible point without further change in the design variables. The approximating contours used as the objective function of the quadratic program (QP) are presented in Figure 5 for iteration 1 of Run 5 and in Figure 6 for iterations 1, 5, and 12 of Run 4. Note how the contours used in the QP approach the actual contours as the optimum is approached.

The same comparison can be seen for the ridge problem with Runs 13 and 14. Traces of their performance are presented in Figure 4. Approximating contours for iterations 1, 3, and 7 of Run 14 are shown in Figure 7 while Figure 8 shows contours for iterations 1, 6, and 15 of Run 13. Again, as the optimum is approached, the approximating contours begin to resemble the true response surface shown in Figure 4.

*In both of these comparisons, the response values shown on the*

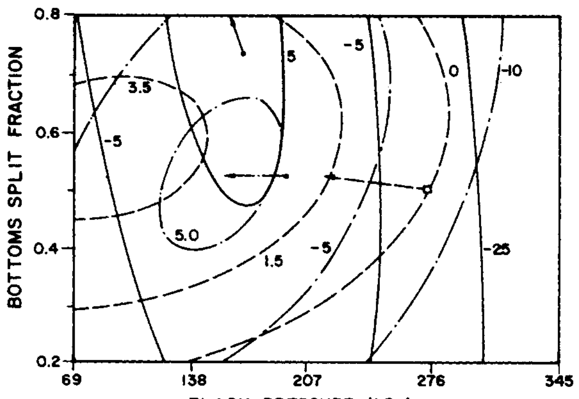

**Figure 7. Contour approximation—run 14—ridge, well-scaled.**
--- Iteration 1
-- Iteration 3
— Iteration 7

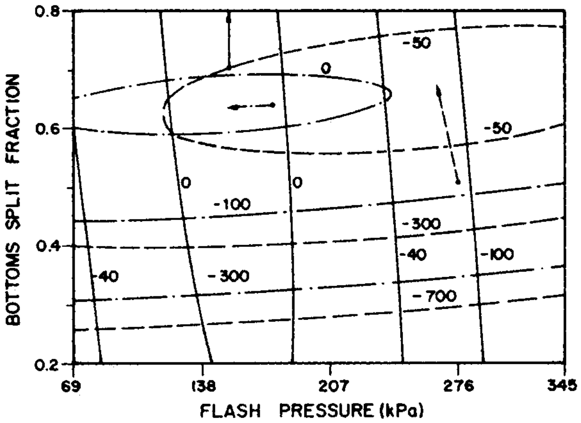

**Figure 8. Contour approximation—run 13—ridge, poorly-scaled.**
--- Iteration 1
-- Iteration 6
— Iteration 15

approximating contour lines are somewhat larger in magnitude than those on corresponding lines of actual surface. This occurs because the equality constraint curvature terms ($v\nabla^2 h$) in the Hessian matrix become significant as one moves away from the current base point.

## Other Control Parameters

The results for Runs 3 and 4 show that IPOSEQ is relatively insensitive to perturbation size, *if* it exceeds a certain minimum. In Run 6, with a perturbation factor of $10^{-4}$, the algorithm took small steps and terminated prematurely. This may have been due to errors introduced by the iterative flash calculations, which have convergence tolerances on the order of $10^{-5}$ to $10^{-6}$.

In choosing the Kuhn-Tucker tolerance, $\epsilon$, one should examine the objective function response to a perturbation with factor $\xi$. We suppose that a good choice for $\epsilon$ is on the same order of magnitude as this response. For the two flash problems, the responses were about the same size as the perturbation factor, so a good Kuhn-Tucker tolerance should then equal $\xi$. This is certainly true for the runs in Table 2; the Kuhn-Tucker *error* after the final iteration is always below the tabulated perturbation size, $\xi$, and examination of the detailed results shows that this error after the preceding iteration was usually greater than $\xi$. However, the Kuhn-Tucker error is calculated with scaled variables. If the scaling is poor, movement of design (or tear) variables will be small and too large a Kuhn-Tucker tolerance may lead to premature termination.

## Tear Variable Initialization

For all of the results given in Table 2, tear variables were initialized by the standard procedure given in Table 1—direct substitution after a single flowsheet pass. This procedure was apparently effective, but one wonders whether the calculation efficiency would improve if the starting values were closer to the converged simulation. Accordingly, some runs were recalculated with starting values calculated by *two* iterations through the flowsheet and/or by complete convergence to a feasible simulation.

The results are compared in Table 4. For the monotonic objective, improved initial feasibility has little effect, except for the poorly-scaled Run 9, where a converged starting point results in premature termination of the optimization. For the ridge objective, similar results are obtained, with the additional problem, for well-scaled Run 14, that improved feasibility greatly increases the computation time.

It is evident that, at least for this problem, the infeasible path algorithm works better if the approach to feasibility is limited to the standard procedure of Table 1. This conclusion can be explained with the help of the illustration in Figure 9.

Consider a response surface in the $x$-$y$ plane with a nonlinear equality constraint $h(x,y) = 0$. A cross section of this surface at the starting point shows the behavior of the exact penalty function used for the line search. In the quadratic programming step, the constraint is linearized and a search direction is determined. For the infeasible point large steps can be taken because the search direction points to an improved and more feasible location. The (near) feasible point, however, is (nearly) at the bottom of the trough in the cross section, while the search direction probably extends

TABLE 4. EFFECT OF APPROACH TO FEASIBILITY AT THE STARTING POINT ON FLASH PROBLEM OPTIMIZATION WITH IPOSEQ

| Run No. | Scale Set | Initializ'n Option[a] | No. of Iter. | CPU Time (s) | Best Objective Value Found |
|---|---|---|---|---|---|
| Monotonic Objective | | | | | |
| 4 | 1 | 1 cycle | 21 | 23.879 | 7.233[b] |
| | | 2 cycle | 21 | 24.151 | 7.233[b] |
| | | feasible | 8 | 9.936 | 7.134 |
| 5 | 2 | 1 cycle | 5 | 6.620 | 7.233[b] |
| | | 2 cycle | 5 | 6.816 | 7.233[b] |
| | | feasible | 5 | 6.498 | 7.233[b] |
| 9 | 5 | 1 cycle | 8 | 8.912 | 7.089 |
| | | feasible | 8 | 10.032 | 7.134 |
| Ridge Objective | | | | | |
| 13 | 5 | 1 cycle | 19 | 22.845 | 6.937[b] |
| | | feasible | 9 | 11.097 | 6.111 |
| 14 | 7 | 1 cycle | 10 | 12.856 | 6.937[b] |
| | | 2 cycle | 20 | 25.325 | 6.937[b] |
| | | feasible | 20 | 25.061 | 6.937[b] |

[a] Options are: 1-cycle: standard procedure of Table 1 (results duplicate those of Table 2). 2-cycle: after first cycle, substitute $y = w$, repeat calculation sequence, and again set $y = w$ for starting point. feasible: complete convergence to $h = 0$ at the starting point, before beginning infeasible path optimization.
[b] Optimal values. Note: Starting point for all runs is $x^0 = [0.5,40]$.

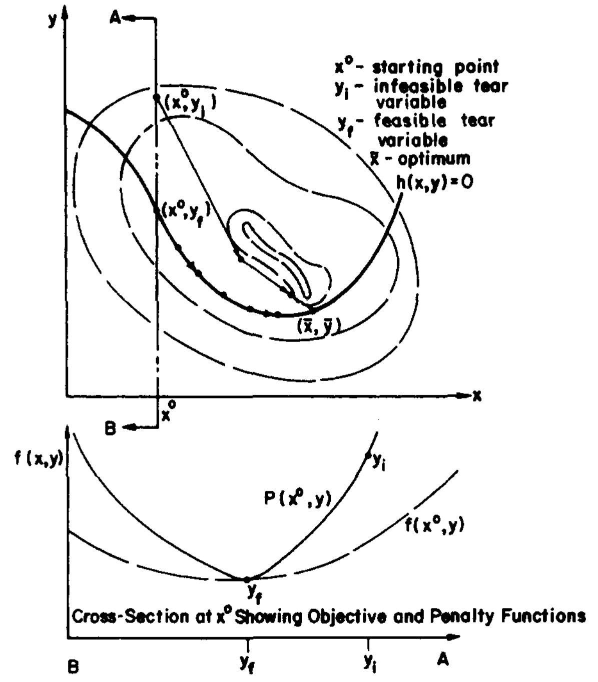

Figure 9. Effect of feasible starting points on hypothetical example with one degree of freedom.

outside of the trough. Thus, a line search will probably find an improved point near the bottom of the trough with a small step size. Unless the objective function gains are large along the search direction, the sequence of points for a feasible start will be nearly feasible and close together. Such a series of small steps either leads to slow convergence or the premature termination seen in Table 4.

This discussion leads to two important points for restarting infeasible path optimizations that terminate prematurely. First, the trends that are observed early in the run may not be in the direction of the optimum if the problem is poorly scaled. Second, a feasible restart of the optimization is not recommended because it may generate small steps or lead to premature termination.

## Summary of Recommendations

Based on theoretical arguments, and limited test calculations, the following heuristics are recommended for the control parameters of the algorithm:

a) Performance is generally insensitive to the perturbation factor, $\xi$, as long as it is larger than the convergence error in iterative calculations within the modules.

b) The best performance usually occurs when the absolute values of the *scaled* elements of the objective function gradient at the starting point are between 10 and 100.

c) Use of converged or nearly-converged simulations as starting points is *not* recommended because this may result in small steps or lead to premature termination.

d) The Kuhn-Tucker tolerance, $\epsilon$, should usually be set approximately equal to the estimated objective function responses for the specified perturbation.

In the algorithm given above, gradients were calculated by numerical perturbation. The alternative, prespecifying all gradients and applying the chain rule, would be tedious and time-consuming but leads to a very efficient algorithm. Perhaps a hybrid of both methods could be used where gradients would be specified with respect to stream variables and design variables could be perturbed. For lengthy or complicated calculations simplified models as applied by Komatsu (1968) or Jirapongphan et al. (1980) may also be useful.

## ACKNOWLEDGMENTS

The authors wish to acknowledge the support of the Paul A. Gorman Fellowship from the International Paper Company Foundation (L.T.B.) and of the Engineering Experiment Station of the University of Wisconsin-Madison.

## NOTATION

| | |
|---|---|
| $B$ | = quasi-Newton approximation to Hessian matrix of Lagrangian |
| $c$ | = design equality constraints |
| $d$ | = search direction from quadratic program |
| $e_i$ | = molar flows of stream ② |
| $E$ | = perturbation matrix, all elements zero except $e_{kk} = 1$ for $k = j$ |
| $g$ | = inequality constraints |
| $g_A$ | = active (or tight) inequality constraints |
| $h$ | = process equalities; tear equations $(y - w = 0)$ |
| $I$ | = identity matrix |
| $J$ | = Jacobian matrix $\partial h_i/\partial y_j$ |
| $P(x,y)$ | = exact penalty function |
| $r$ | = retention variable vector |
| $r^*$ | = perturbed retention variable vector |
| $T$ | = transpose |
| $t$ | = quadratic program (qp) shadow price for $c$ |
| $u$ | = qp shadow price for $g$ |
| $v$ | = qp shadow price for $h$ |
| $w$ | = calculated tear variable |
| $x$ | = design variable |
| $y$ | = process variable; guessed tear variable |
| $z$ | = subset of $x$ for design problem; $\begin{bmatrix} x \\ y \end{bmatrix}$ |

### Greek Letters

| | |
|---|---|
| $\Delta$ | = change in variable due to iteration or perturbation |
| $\epsilon$ | = Kuhn-Tucker tolerance |
| $\phi$ | = scalar objective function |
| $\lambda$ | = stepsize between zero and one |
| $\xi$ | = perturbation factor |

### Math Symbols

$$\nabla_\eta \equiv \left[\frac{\partial \zeta_i}{\phi \eta_j}\right], \text{ where vector } \zeta \text{ is a function of vector } \eta.$$

If $\eta$ is not specified, differentiation occurs with respect to all variable vectors of $\zeta$.

## LITERATURE CITED

Berna, T. J., M. H. Locke, and A. W. Westerberg, "A New Approach to Optimization of Chemical Processes," *AIChE J.*, **26**, 1, p. 37 (1980).

Biegler, L. T., and R. R. Hughes, "Approximation Programming of Chemical Processes with Q/LAP," *Chem. Eng. Prog.*, **77**, 4, p. 76 (1981).

Boston, J. F., and H. I. Britt, "A Radically Different Solution of the Single Stage Flash Problem," *Comp. & Chem. Engr.*, **2**, p. 109 (1978).

Box, M. J., "A New Method of Constrained Optimization and a Comparison with Other Methods," *Computer J.*, **8**, 1, p. 42 (1965).

Broyden, C. G., "A Class of Methods for Solving Nonlinear Simultaneous Equations," *Math. Comp.*, **19**, p. 577 (1965).

Dennis, J. E., and J. J. More, "Quasi-Newton Methods, Motivation and Theory," *SIAM Review*, **19**, 1, p. 46 (1977).

Hughes, R. R., *SPAD User's Manual*, University of Wisconsin-Madison (1974, 1981).

Jirapongphan, S., "Simultaneous Modular Convergence Concept in Process Flowsheet Optimization," Sc.D. Thesis, M. I. T. (1980).

Jirapongphan, S., J. F. Boston, H. I. Britt, and L. B. Evans, "A Nonlinear Simultaneous Modular Algorithm for Process Flowsheet Optimization," 80th AIChE Meeting, Chicago, Preprint #3b (1980).
Kehat, E., and M. Shacham, "Chemical Process Simulation Programs-3," *Process Technol.*, 18, 4, p. 181 (1973).
Komatsu, S., "Application of Linearization to Design of a Hydrodealkylation Plant," *IEC Oper. Res. Symp.*, 60, 2, p. 36 (1968).
Parker, A. L., and R. R. Hughes, "Approximation Programming of Chemical Processes—1," *Comp. & Chem. Engr.*, 5, 3, p. 123 (1981).
Perkins, J. D., "Efficient Solution of Design Problems Using a Sequential Modular Flowsheeting Programme," *Comp. & Chem. Engr.*, 3, p. 375 (1979).
Powell, M. J. D., "A Fast Algorithm for Nonlinearly Constrained Optimization Calculations," *Numerical Analysis*, Dundee (1977, Lecture Notes in Math., #630, p. 144, Springer, Berlin (1978)).
Westerberg, A. W., "Optimization in Computer-Aided Design," *Foundations of Computer-Aided Chemical Process Design*, Mah and Seider, eds., p. 149, Engineering Foundation, New York (1980).

*Manuscript received October 6, 1981; revision received February 17, and accepted March 4, 1982.*

# Oscillatory Behavior of a Gas Bubble Growing (or Collapsing) in Viscoelastic Liquids

**HEE JU YOO and CHANG DAE HAN**
**Department of Chemical Engineering**
**Polytechnic Institute of New York**
**Brooklyn, NY 11201**

**A theoretical study was carried out to achieve a better understanding of the oscillatory behavior of a gas bubble growing (or collapsing) in a viscoelastic liquid, by taking into account both the hydrodynamic and diffusion effects. The Zaremba-DeWitt model was chosen to represent the rheological properties of the suspending medium. The finite difference method was employed to solve the governing system equations.**

**The computational results show that, in the case of *very fast* diffusion (i.e., constant bubble pressure), the oscillatory behavior of a bubble takes place only when the ratio of the initial pressure difference between the gas bubble and the liquid phase to the elastic modulus of the suspending medium is below a certain critical value. On the other hand, in the case of *very slow* diffusion, the oscillatory behavior of a bubble persists, regardless of the magnitude of the rheological properties of the suspending medium. Our study indicates further that the diffusivity of a gas has a profound influence on the occurrence of oscillatory behavior, that the elastic property of the suspending medium enhances oscillatory behavior while its viscosity plays the opposite role, and that even a Newtonian medium can give rise to an oscillatory pattern of bubble growth (or collapse), although it dampens out very quickly.**

## SCOPE

**Today, plastics foam processing is one of the fastest growing polymer processing techniques for manufacturing the cellular products, either extruded or injection-molded. The success of plastic foam processes, however, depends largely on the ability of controlling cell size and its distribution in the final product, because the physical/mechanical properties of foamed products are very much affected by the cell structure (Frisch and Saunders, 1972). It is, therefore, very important to have a better understanding of the phenomena associated with the formation of gas bubbles and their growth in molten polymers during processing.**

**In the manufacture of foamed products using thermoplastic resins, two basic types of additives are available: (1) organic or inorganic compounds, often referred to as chemical blowing agents, which, after being blended with the resin, decompose on heating to give off gas which expands the molten polymer; and (2) volatile liquids, often referred to as physical blowing agents, which, after being injected into the molten polymer, vaporize at the boiling point, leaving gas cells in the resin. In the use of either type of blowing agent, the control of bubble growth in the molten polymer is of fundamental importance in obtaining cellular products of uniform cell size.**

**As may be surmised, the control of cell size and its distribution in foaming processing is dependent upon the following factors: (1) the physical and thermodynamic properties of the molten polymer/blowing agent mixture; (2) the rheological properties of molten polymer/blowing agent mixture; and (3) the temperature and pressure of the molten polymer/blowing agent mixture. Several researchers have carried out theoretical investigations of bubble growth (or collapse) in viscoelastic liquids (Street, 1968; Fogler and Goddard, 1970; Tanasawa and Yang, 1970; Ting, 1975; Zana and Leal, 1975) and in Newtonian liquids (Plesset and Zwick, 1954; Scriven, 1959; Barlow and Langlois, 1962; Marique and Houghton, 1962; Ruckenstein and Davis,**

0001-1541-82-6467-1002-$2.00. © The American Institute of Chemical Engineers, 1982.