15

OPTIMIZATION IN LARGE-SCALE PLANT DESIGN AND OPERATIONS

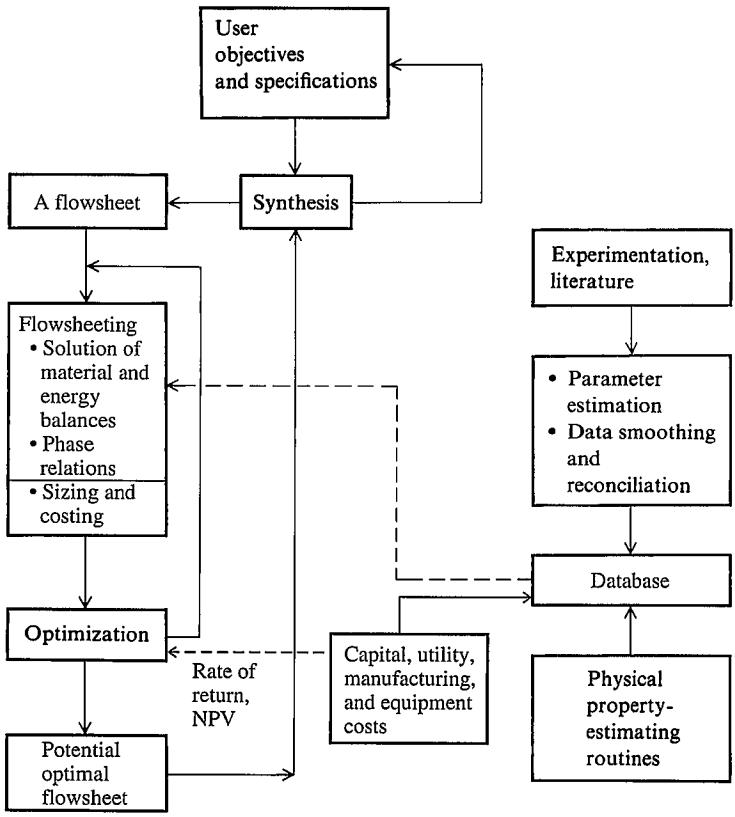

FIGURE 15.1
Information flow in the design process.

AS DISCUSSED IN Chapter 1, optimization of a large configuration of plant components can involve several levels of detail ranging from the most minute features of equipment design to the grand scale of international company operations. As an example of the size of the optimization problems solved in practice, Lowery et al. (1993) describe the optimization of a bisphenol-A plant via SQP involving 41,147 variables, 37,641 equations, 212 inequality constraints, and 289 plant measurements to identify the most profitable operating conditions. Perkins (1998) reviews the topic of plantwide optimization and its future.

An important global function of optimization is the synthesis of the optimal plant configuration (flowsheet). By *synthesis* we mean the designation of the structure of the plant elements, such as the unit operations and equipment, that will meet the designer's goals. Figure 15.1 shows the relation of synthesis to design and operation. You check a flowsheet for equipment that can be eliminated or rearranged, alternative separation methods, unnecessary feeds that can be eliminated, unwanted or hazardous product or byproducts that can be deleted, heat integration that can be improved, and so on. Even if no new technology is to be used, the problem is com-

binatorial in nature, and the number of alternatives increases substantially. For example, Gunderson and Grossman (1990) in synthesizing a heat exchanger network showed that for a net of five units below the pinch point (and three above), 126 different arrangements of exchangers exist. We have chosen not to discuss the general problem of synthesis in this chapter, but instead we treat examples of optimization applied to design of a specified configuration or flowsheet.

A major use of optimization is in the detailed *design* or retrofit of a plant for which the flowsheet is already formulated. Goals are to enhance profitability; reduce utility costs; select raw materials; size equipment; lay out piping; and analyze reliability, flexibility, and safety, and so on. Often, as a result of various case studies, a base case is developed by creating a detailed process flowsheet containing the major pieces of equipment. Then, process flow simulators are employed to achieve improved designs. The design team improves the database by getting vendor data and perhaps pilot plant data; simulates the base case design to find improvements and barriers to feasibility; and develops networks of heat exchangers, turbines, and compressors to satisfy the heating, cooling, and power requirements of the process. Refer to any of the process design books such as Seider et al. (1999) for details concerning the design process.

An even more widespread application of optimization is the determination of the optimal operating conditions for an existing plant, such as selecting particular feedstocks, temperatures, pressures, flow rates, and so on. Figure 15.2 traces the information flow involved in determining the optimal plant operating conditions. See Chapter 16 for a discussion of the optimization hierarchy in plant operations. As indicated in the figure, optimization occurs at intermediate stages of the process simulator as well as in the overall economic evaluation. The figure implies that effectively meshing optimization algorithms with process simulators requires more than just an optimization code and a process simulator containing the process model. The software functions involved are

1. A supervisor or director to manage overall control of the software components.
2. Data processing conditioning, reconciliation, and validation of the data evolving from the plant.
3. Estimation of process parameters and unmeasured variables.
4. Optimization of different kinds of problems.
5. Simulation of plant models (equations, modules, or both) of varying degrees of detail.
6. A database (historian) for process variables, costs and revenues, operating conditions, disturbances, and so on.
7. Communication links for data transfer and command signals.
8. Reports and analysis capability for unit and plant performance, economic performance, and hypothetical scenarios.

Although uncertainty exists in the results of all cases of the optimization of plants because of the uncertainty in the values of the parameters in the process models themselves, in the cost and revenue values in the objective function, and in potential changes in the process inputs, we avoid such issues in this chapter and focus solely on deterministic optimization.

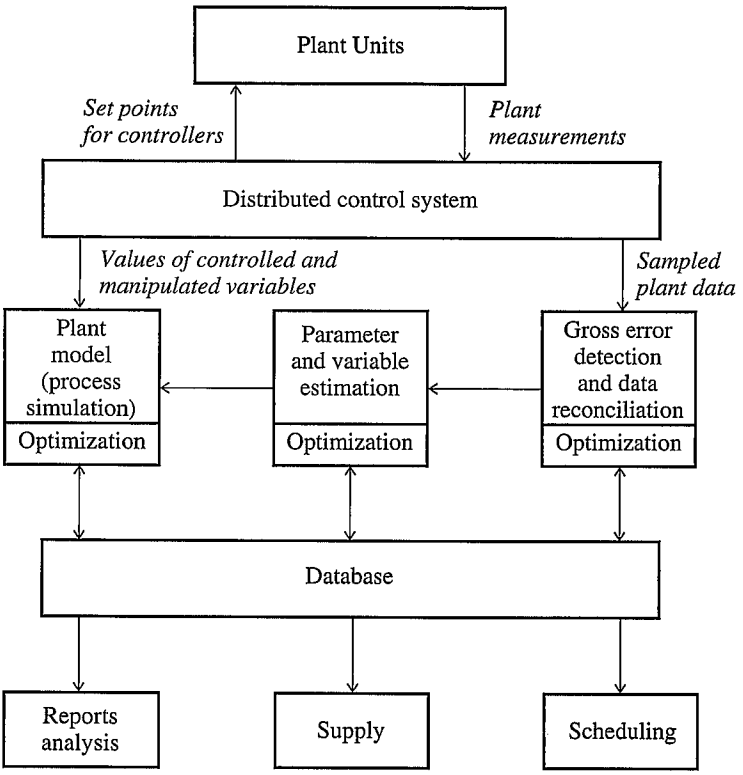

FIGURE 15.2
Information flow in developing the optimal operating conditions.

15.1 PROCESS SIMULATORS AND OPTIMIZATION CODES

Process simulators contain the *model* of the process and thus contain the bulk of the constraints in an optimization problem. The equality constraints ("hard constraints") include all the mathematical relations that constitute the material and energy balances, the rate equations, the phase relations, the controls, connecting variables, and methods of computing the physical properties used in any of the relations in the model. The inequality constraints ("soft constraints") include material flow limits; maximum heat exchanger areas; pressure, temperature, and concentration upper and lower bounds; environmental stipulations; vessel hold-ups; safety constraints; and so on. A *module* is a model of an individual element in a flowsheet (e.g., a reactor) that can be coded, analyzed, debugged, and interpreted by itself. Examine Figure 15.3a and b.

Two extremes are encountered in process simulator software. At one extreme the process model comprises a set of equations (and inequalities) so that the process model equations form the constraints for optimization, exactly the same as described in previous chapters in this book. This representation is known as an *equation-*

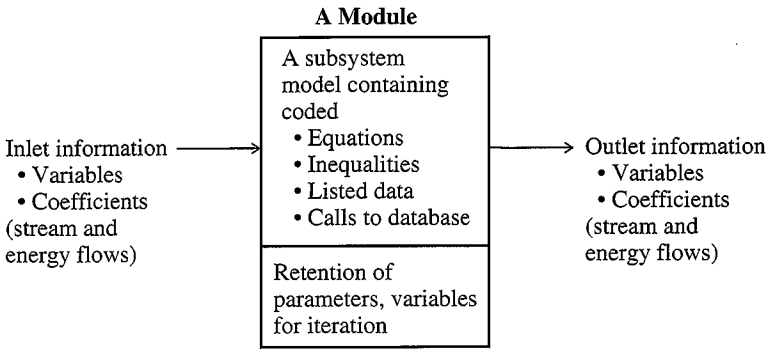

FIGURE 15.3a
A typical process module showing the necessary interconnections of information.

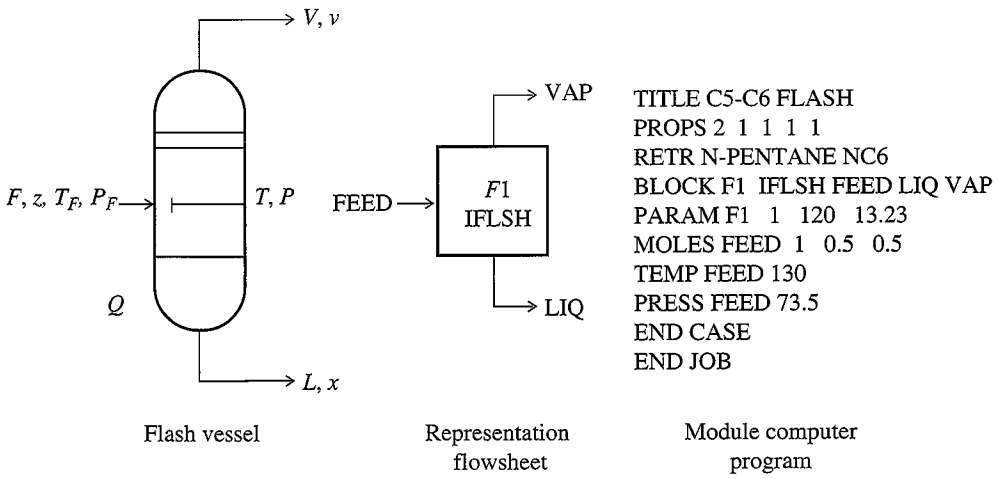

FIGURE 15.3b
A module that represents a flash unit. (Reproduced, with permission, from J. D. Seader, W. D. Seider, and A. C. Pauls. *Flowtran Simulation—An Introduction*. Austin, TX: CACHE, 1987.)

oriented process simulator. The equations can be solved in a sequential fashion analogous to the modular representation described in the next section, or simultaneously by Newton's method or by employing sparse matrix techniques to reduce the extent of matrix manipulations (Gill et al., 1981). Two of the better known equation-based codes are Aspen Custom Modeler (Aspen Technology 1998) and ASCEND (Westerberg 1998). Equation-based codes such as DMCC and RT-OPT (Aspen Technology), and ROMEO (Simulation Sciences, 1999) dominate closed-loop, real-time optimization applications (refer to Chapter 16). Section 15.2 covers meshing equation-based process simulators with optimization algorithms.

At the other extreme, the process can be represented on a flowsheet by a collection of modules (a modular-based process simulator) in which the equations (and other information) representing each subsystem or piece of equipment are coded so that a module may be used in isolation from the rest of the flowsheet and hence is portable from one flowsheet to another. Each module contains the equipment sizes, the material and energy balance relations, the component flow rates, temperatures, concentrations, pressures, and phase conditions. Examples of commercial codes are ASPEN PLUS (Aspen Technology, 1998), HYSYS (Hyprotech, 1998), ChemCAD (Chemstations, 1998), PRO/II 1998 (Simulation Sciences, 1998), and Batch Pro and Enviro Pro Designer (Intelligen, 1999). Section 15.3 covers meshing modular-based process simulators with optimization algorithms.

In addition to the two extremes, combinations of equations and modules can be used. Equations can be lumped into modules, and modules can be represented by their basic equations or by polynomials that fit the input–output information.

Although, as explained in Chapter 9, many optimization problems can be naturally formulated as mixed-integer programming problems, in this chapter we will consider only steady-state nonlinear programming problems in which the variables are continuous. In some cases it may be feasible to use binary variables (on–off) to include or exclude specific stream flows, alternative flowsheet topography, or different parameters. In the economic evaluation of processes, in design, or in control, usually only a few (5–50) variables are decision, or independent, variables amid a multitude of dependent variables (hundreds or thousands). The number of dependent variables in principle (but not necessarily in practice) is equivalent to the number of independent equality constraints plus the active inequality constraints in a process. The number of independent (decision) variables comprises the remaining set of variables whose values are unknown. Introduction into the model of a specification of the value of a variable, such as $T = 400°C$, is equivalent to the solution of an independent equation and reduces the total number of variables whose values are unknown by one.

In optimization using a process simulator to represent the model of the process, the *degrees of freedom* are the number of decision variables (independent variables) whose values are to be determined by the optimization, hence the results of an optimization yield a fully determined set of variables, both independent and dependent. Chapter 2 discussed the concept of the degrees of freedom. Example 15.1 demonstrates the identification of the degrees of freedom in a small process.

EXAMPLE 15.1 CALCULATION OF THE DEGREES OF FREEDOM

Figure E15.1 shows a simplified flowsheet for the conversion of N_2 and H_2 to ammonia (NH_3) when argon (A) is present in the feed. After the reaction of N_2 and H_2

$$N_2 + 3H_2 \rightarrow 2NH_3$$

the NH_3 is separated as a liquid from the gas phase. A purge gas stream prevents argon build-up in the system. Fresh feed is introduced in the proper ratio of N_2 to H_2 with

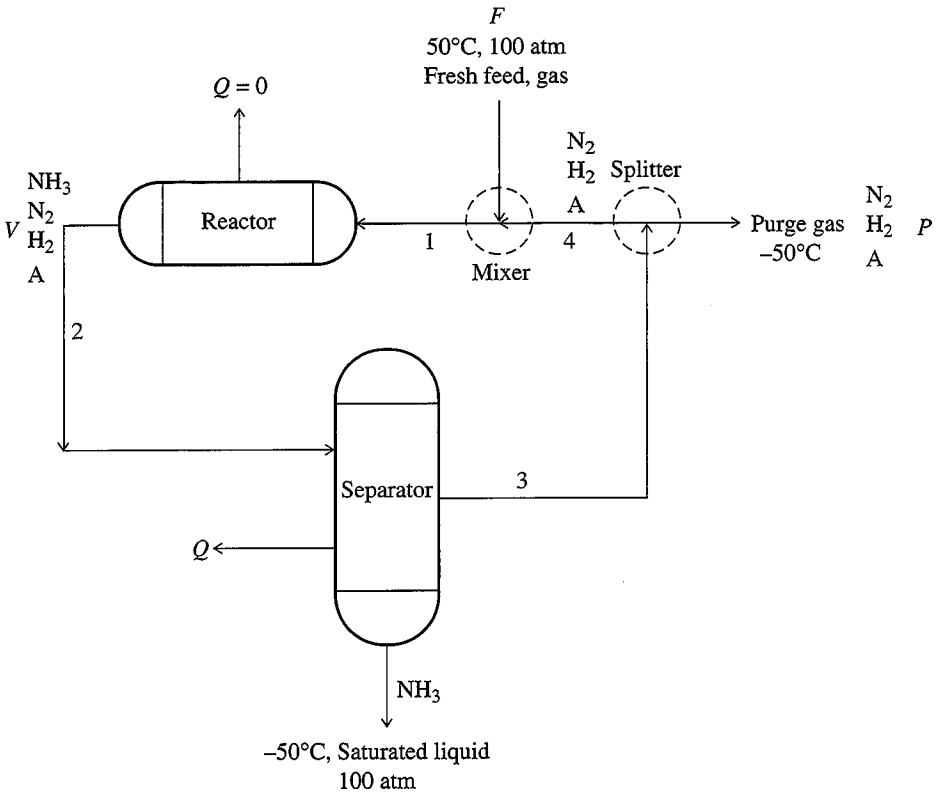

FIGURE E15.1

the accompanying argon of about 0.9 percent. Assume all of the units and the pipe lines are adiabatic ($Q = 0$). The fraction conversion in the reactor is 25 percent.

The process has four separate subsystems for the degree-of-freedom analysis. Redundant variables and redundant constraints are removed to obtain the net degrees of freedom for the overall process. The 2 added to N_{sp} refers to the conditions of temperature and pressure in a stream; +1 represents the heat transfer Q.

In this example

N_v is the number of variables
N_{sp} is the number of components (species) in a stream
N_r is the number of independent constraints
N_d is the degrees of freedom (number of decision variables)

The analysis of each subsystem is as follows.

Mixer:

$N_v = 3(N_{sp} + 2) + 1 = 3(6) + 1 =$ 19

N_r:

Material balances (H_2, N_2, A only)	3
Energy balance	1

Specifications:

NH_3 concentration is zero	3	
$T_P = -50°C$	1	
$T_F = 50°C$	1	
Assume that $p_F = p_{\text{mix out}} = p_{\text{split}} = 100$	3	
$Q = 0$	1	13
N_d: 19 − 13 =		6

Reactor:

$N_v = 2(N_{sp} + 2) + 1 = 2(6) + 1 =$ 13

N_r:

Material balances (H, N, A)	3	
Energy balances	1	
Specifications:		
NH_3 entering = 0	1	
$Q = 0$	1	
Fraction conversion	1	
$p_{\text{in}} = p_{\text{out}} = 100$ atm	2	
Energy balance	1	10
N_d: 13 − 10 =		3

Separator:

$N_v = 3(N_{sp} + 2) + 1 = 3(6) + 1 =$ 19

N_r:

Material balances	4	
Energy balance	1	
Specifications:		
$T_{\text{out}} = -50°C$	1	
$p_r = p_{\text{in}} = p_{NH_3} = 100$	3	
NH_3 concentration is 0 in recycle gas	1	
N_2, H_2, A are 0 in liquid NH_3	3	13
N_d: 19 − 13 =		6

Splitter:

$N_v = 3(N_{sp} + 2) = 3(6) =$ 18

N_r:

Material balances	1	
Specifications:		
NH_3 concentration = 0	1	
Compositions same $2(N_{sp} - 1)$	6	
Stream temperatures same = −50°C	3	
Stream pressures same = 100 atm	3	14
N_d: 18 − 14 =		4

The total number of degrees of freedom is 19 less the redundant information, which is as follows:

Redundant variables in interconnecting streams being eliminated:

Stream 1: $(4 + 2) =$	6
Stream 2: $(4 + 2) =$	6
Stream 3: $(4 + 2) =$	6
Stream 4: $(4 + 2) =$	6
	24

Redundant constraints being eliminated:

Stream 1:	
NH_3 concentration $= 0$	1
$p = 100$ atm	1
Stream 2:	
$p = 100$ atm	1
Stream 3:	
NH_3 concentration $= 0$	1
$p = 100$ atm	1
$T = -50°C$	1
Stream 4:	
NH_3 concentration $= 0$	1
$T = -50°C$	1
$p = 100$ atm	1
	9

Overall the number of degrees of freedom should be

$$N_d = 19 - 24 + 9 = 4$$

The redundant constraints and variables can be regarded as $24 + 9 = 33$ additional equality constraints in the optimization problem.

In optimization using a modular process simulator, certain restrictions apply on the choice of decision variables. For example, if the location of column feeds, draws, and heat exchangers are selected as decision variables, the rate or heat duty cannot also be selected. For an isothermal flash both the temperatures and pressure may be optimized, but for an adiabatic flash, on the other hand, the temperature is calculated in a module and only the pressure can be optimized. You also have to take care that the decision (optimization) variables in one unit are not varied by another unit. In some instances, you can make alternative specifications of the decision variables that result in the same optimal solution, but require substantially different computation time. For example, the simplest specification for a splitter would be a molar rate or ratio. A specification of the weight rate of a component in an exit flow stream from the splitter increases the computation time but yields the same solution.

Next, we need to clarify some of the jargon that you will find in the literature and documentation associated with commercial codes that involve process simulators. Two major types of optimization algorithms exist for nonlinear programming.

1. *Feasible path algorithms.* The equality constraints and active inequality constraints are satisfied at the end of every intermediate stage of the calculations.
2. *Infeasible path algorithms.* The equality constraints and active inequality constraints are satisfied only at the stage on which the optimal solution is reached.

Clearly option 1 incurs more computation time when process simulators are involved, but an abnormal termination yields a feasible solution.

Another classification of optimization codes relates whether a full set of variables is used in the search:

1. *Full vector.* All the independent and dependent variables constitute the vector of variables in the search.
2. *Reduced vector.* Only the independent variables are involved in the search; the dependent variables are then determined from the constraints.

With respect to process simulators, we can identify three types, with hybrid types often occurring:

1. *Equation-based.* Explained previously.
2. *Sequential modular.* Refers to the process simulator being based on modules, and the modules solved in a sequential precedence order imposed by the flowsheet information flow.
3. *Simultaneous modular.* The process simulator is composed of modules, but simplified, approximate, or partial representation of the modules enables solution techniques used in equation-based methods to be employed.

Other jargon you will encounter:

1. *Online.* Optimization calculations are carried out by computers that process plant data and transmit control signals.
2. *Offline.* Data is collected and used subsequently by separate computers for optimization so that the results are not directly available.
3. *Real time.* The clock cycle for the collection and transfer of process data and the optimization calculations is the same.

The kinds of optimization codes most often used together with process simulators include

1. Linear programming: LP (refer to Chapter 7).
2. Sequential linear programming: SLP (refer to Chapter 8).
3. Sequential quadratic programming: SQP (refer to Chapter 8).
4. Generalized reduced gradient: GRG (refer to Chapter 8).
5. Nonlinear programming: NLP—other than items 3 or 4 (refer to Chapter 8).
6. Mixed-integer nonlinear programming: MINLP (refer to Chapter 9).
7. Mixed-integer successive quadratic programming (refer to Chapter 9).
8. Random search (refer to Chapter 10 or Section 6.1).

Commercial process simulators mainly use a form of SQP. To use LP, you must balance the nonlinearity of the plant model (constraints) and the objective function with the error in approximation of the plant by linear models. Infeasible path, sequential modular SQP has proven particularly effective.

Finally, we should mention that in addition to solving an optimization problem with the aid of a process simulator, you frequently need to find the sensitivity of the variables and functions at the optimal solution to changes in fixed parameters, such as thermodynamic, transport and kinetic coefficients, and changes in variables such as feed rates, and in costs and prices used in the objective function. Fiacco in 1976 showed how to develop the sensitivity relations based on the Kuhn–Tucker conditions (refer to Chapter 8). For optimization using equation-based simulators, the sensitivity coefficients such as $(\partial h_i/\partial x_i)$ and $(\partial x_i/\partial x_j)$ can be obtained directly from the equations in the process model. For optimization based on modular process simulators, refer to Section 15.3. In general, sensitivity analysis relies on linearization of functions, and the sensitivity coefficients may not be valid for large changes in parameters or variables from the optimal solution.

15.2 OPTIMIZATION USING EQUATION-BASED PROCESS SIMULATORS

In this section we consider general process simulator codes rather than specialized codes that apply only to one plant. To mesh equation-based process simulators with optimization codes, a number of special features not mentioned in Chapter 8 must be implemented.

1. A method of formatting the equations and inequality constraints. Slack variables are used to transform the inequality constraints into equality constraints.
2. A possibility of using both continuous and discrete variables, the latter being particularly necessary to accommodate changes in phase or changes from one correlation to another.
3. The option of using alternative forms of a function depending on the value of logical variables that identify the state of the process. Typical examples are the shift in the relations used to calculate the friction factor from laminar to turbulent flow, or the calculation of $P - V - T$ relations as the phase changes from gas to liquid.
4. Efficient methods for solving equations in the physical property database (which often require up to 80% of the computation time needed to solve a plant optimization problem).
5. Efficient methods for solving large sets of linear equations, for example, the linearized constraints, particularly involving sparce matrices.
6. A good method of selecting initial guesses for the solution of the algebraic equations. Poor choices lead to unsatisfactory results. You want the initial guesses to be as close to the optimal solution as possible so that the procedure will converge, and converge rapidly. We recommend running the process simulator alone to develop one or more base cases that will serve feasible starting points for the optimization.

7. Provision for *scaling* of the variables and equations. By scaling variables we mean introducing transformations that make all the variables have ranges of the same order of magnitude. By scaling of equations we mean multiplying each equation by a factor that causes the value of the deviation of each equation from zero to be of the same order of magnitude. User interaction and analysis for a specific problem is one way to introduce scaling.
8. The code must carry out a structural analysis to determine if the model is well posed, that is, can it detect any inconsistencies among the equations in the model (Duff et al., 1989; Zaher, 1995)?

Figure 15.4 shows how the nonlinear optimization problem fits in with two widely used optimization algorithms: the generalized reduced gradient (GRG) and successive quadratic programming (SQP). The notation is in Table 15.1. Slack variables $\mathbf{x}_s$ have been added to the inequality constraints $\mathbf{g} \geq \mathbf{0}$ to convert them to equality constraints. The formulation in Figure 15.4 assumes that the functions and variables are continuous and differentiable (in practice, finite differences may be used as substitutes for analytical derivatives). Although we will not discuss optimization of dynamic processes in this chapter, in the NLP problem you can insert differential equations as additional equality constraints. Refer to Ramirez (1994) for details. In the execution of the optimization code, in some phases the specific assignment of independent and dependent variables within the code may differ from those you designate.

In formatting the inequalities g and equations h, you will find that the so-called open-equation representation is preferred to the closed-equation representation. One of the simplest examples is a heat exchanger model (the closed-equation format):

$$Q = F_C C_{p_C}(T_{C,\text{out}} - T_{C,\text{in}})$$

$$Q = F_H C_{p_H}(T_{H,\text{in}} - T_{H,\text{out}})$$

$$Q = UA\left[\frac{(T_{H,\text{in}} - T_{C,\text{out}}) - (T_{H,\text{out}} - T_{C,\text{in}})}{\ln\left[\dfrac{(T_{H,\text{in}} - T_{C,\text{out}})}{(T_{H,\text{out}} - T_{C,\text{in}})}\right]}\right]$$

where A = heat transfer area
C_p = heat capacity
F = flow rate
Q = heat transferred
T = temperature
U = heat transfer coefficient
H = hot
C = cold

If the temperatures, heat capacities, U, and A are known quantities, then you can directly calculate Q and the F's. On the other hand, if you know the stream flows,

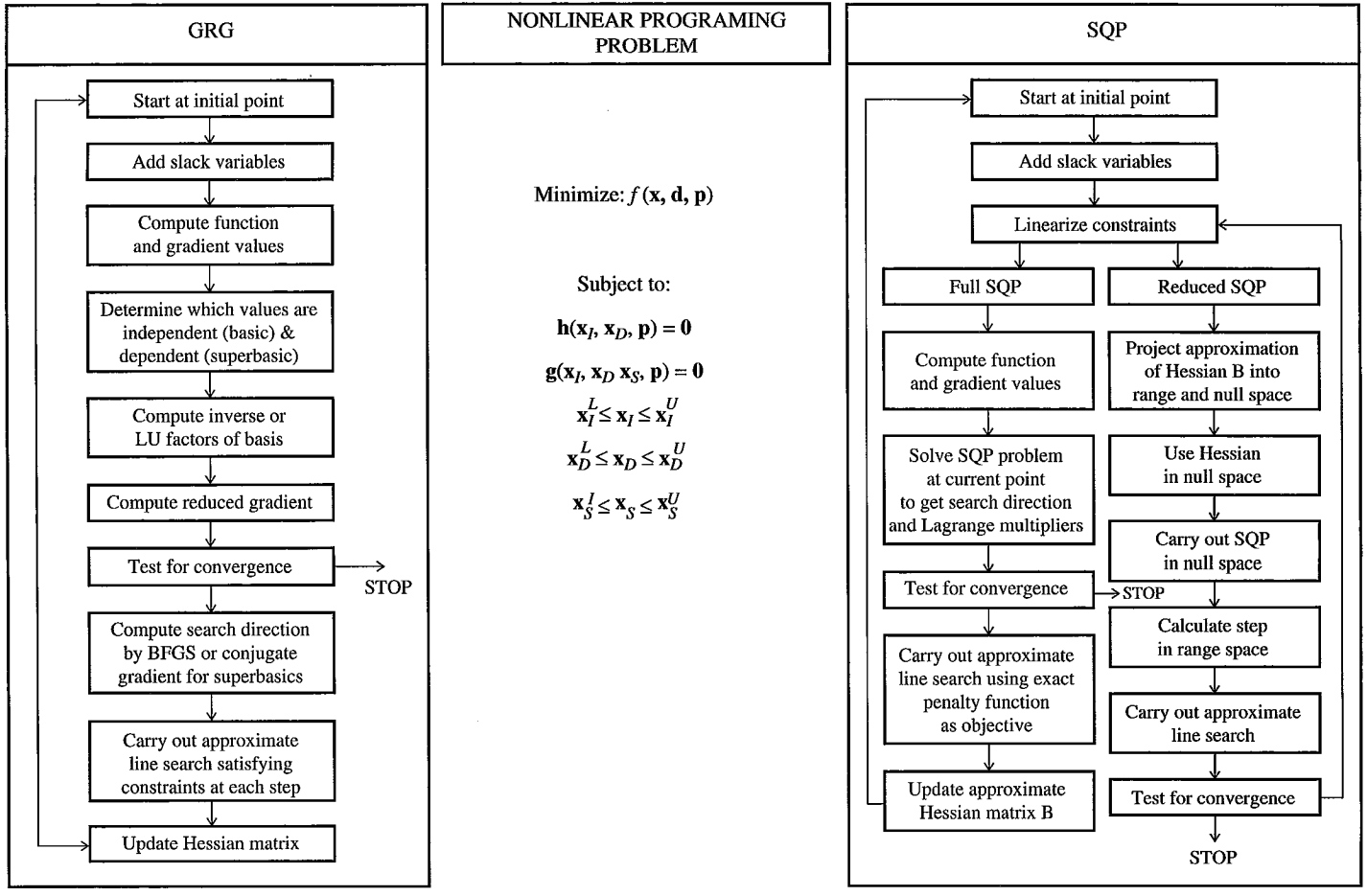

FIGURE 15.4
The nonlinear programming problem and typical GRG and SQP algorithms for solution; see notation in Table 15.1.

TABLE 15.1
Notation for Figure 15.4

f	Objective function
$\mathbf{g}$	Set of inequality constraints
$\mathbf{h}$	Set of equality constraints
$\mathbf{p}$	Vector of coefficients in the objective function and constraints
$\mathbf{x}_I$	Vector of independent (decision) variables
$\mathbf{x}_D$	Vector of dependent variables
$\mathbf{x}_S$	Vector of slack variables added to the inequality constraints
L	lower bound
U	upper bound

inlet temperatures, C_p's, U, and A, then the solution for Q and the outlet temperatures must be determined via iteration. This problem arises particularly for process models in which one unit that is underspecified is connected with another unit that is overspecified.

By using open-equation formats and infeasible path optimization algorithms, the type of difficulty described above can be avoided. All the equations in the NLP problem can be solved simultaneously, driving the residuals to zero. The open-equation format for the heat exchanger is

$$R_1 = Q - F_C C_{p_C}(T_{C,\text{out}} - T_{C,\text{in}})$$

$$R_2 = Q - F_H C_{P_H}(T_{H,\text{in}} - T_{H,\text{out}})$$

$$R_3 = Q \ln\left[\frac{(T_{H,\text{in}} - T_{C,\text{out}})}{(T_{H,\text{out}} - T_{C,\text{in}})}\right] - UA[(T_{H,\text{in}} - T_{C,\text{out}}) - (T_{H,\text{out}} - T_{C,\text{in}})]$$

where R_i is a residual. Note that division by a logarithm has been eliminated.

Another advantage of the open-equation format is that simple connection equations can be used rather than eliminating variables and equations that are connected. For example, the connections between two heat exchangers can be formulated as

$$R_1 = F_{C,\text{out},1} - F_{C,\text{in},2}$$

$$R_2 = T_{C,\text{out},1} - T_{C,\text{in},2}$$

$$R_3 = F_{H,\text{out},2} - F_{H,\text{in},1}$$

$$R_4 = T_{H,\text{out},2} = T_{H,\text{in},1}$$

More variables are retained in this type of NLP problem formulation, but you can take advantage of sparse matrix routines that factor the linear (and linearized) equations efficiently. Figure 15.5 illustrates the sparsity of the Hessian matrix used in the QP subproblem that is part of the execution of an optimization of a plant involving five unit operations.

Figure 15.4 shows the Hessian matrix for two different types of SQP algorithms for solving large-scale optimization problems. In the full-space SQP, all of

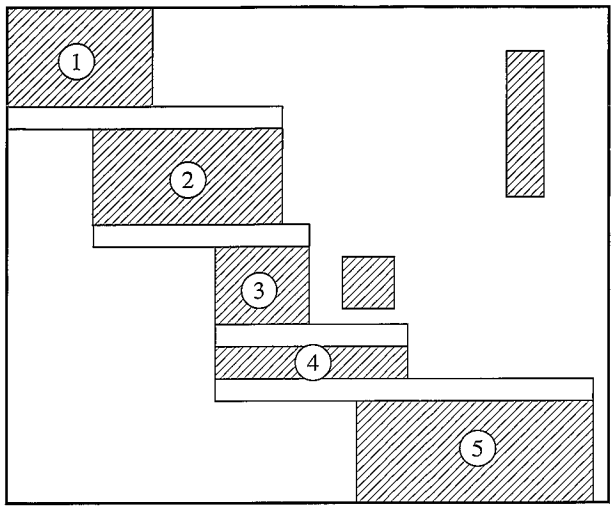

FIGURE 15.5
The Hessian matrix for the QP subproblem showing five units and the sparsity of the matrix.

the variables, both independent and dependent, are solved for simultaneously in the set of linear(ized) equations in the QP subproblem. The sparse structure of both **B** and $\Delta\mathbf{h}$ can be taken advantage of in their solution. In the reduced-space SQP only the sparse structure of $\Delta\mathbf{h}$ is used. For specific details of the execution of the reduced SQP, refer to the summary in Biegler et al. (1997) and the references therein, and to Schmid and Biegler (1994a).

As mentioned before, two contrasting classes of strategies exist for executing the SQP algorithms:

1. Feasible path strategies.
2. Infeasible path strategies.

With feasible path strategies, as the name implies, on each iteration you satisfy the equality and inequality constraints. The results of each iteration, therefore, provide a candidate design or feasible set of operating conditions for the plant, that is, suboptimal. Infeasible path strategies, on the other hand, do not require exact solution of the constraints on each iteration. Thus, if an infeasible path method fails, the solution at termination may be of little value. Only at the optimal solution will you satisfy the constraints.

To improve the formatting of the equations that represent a plant, many commercial codes partition the equations into groups of irreducible sets of equations, that is, those that have to be solved simultaneously. If a plant is represented by thousands of equations, the overall time consumed in their solution via either a GRG or SQP algorithm is reduced by partitioning and rearranging the order of the equations with the result indicated in Figure 15.6. Organization of the set of equations into irreducible sets can be carried out by the use of permutation matrices or by one of

$$
\begin{aligned}
h_1&: x_1^2 x_2 - 2x_3^{1.5} + 4 = 0\\
h_2&: x_2 + 2x_5 - 8 = 0\\
h_3&: x_1 x_4 x_5^2 - 2x_3 - 7 = 0\\
h_4&: -2x_2 + x_5 + 5 = 0\\
h_5&: x_2 x_4^2 x_5 + x_2 x_4 - 6 = 0
\end{aligned}
$$

(a) The n independent equations involving n variables ($n = 5$).

	x_1	x_2	x_3	x_4	x_5
h_1	1	1	1		
h_2		1			1
h_3	1		1	1	1
h_4		1			1
h_5		1		1	1

(b) The occurrence matrix (the 1's represent the occurrence of a variable in an equation).

	x_2	x_5	x_4	x_1	x_3
h_2	1	1	I		
h_4	1	1	II		
h_5	1	1	1	III	
h_3		1	1	1	1
h_1	1			1	1

(c) The rearranged (partitioned) occurrence matrix with groups of equations (sets I, II, and III) that have to be solved simultaneously collected together in the precedence order for solution.

FIGURE 15.6
Partitioning of sets of independent equations increases the sparsity of the occurrence matrix.

the many algorithms found in Himmelblau (1973). Feedback of information, materials, or energy ties equations together in irreducible groups.

We next solve an example optimization problem for a plant represented by equations and inequalities using the GRG method.

EXAMPLE 15.2 PROCESS OPTIMIZATION VIA GRG (EQUATION-BASED SOFTWARE)

Figure E15.2 shows the flowsheet for the process. Feed (stream 1) is a vapor mixture of ethane, propane, and butane (in the proportions shown in the figure) at 200°F and 500 psia. The product stream (stream 8) is a liquid at ≤ -20°F having the same composition but a reduced pressure. The notation for this example is defined in Table E15.2A.

1 Objective function. A simple objective function is used, namely, the minimization of the instantaneous cost of the work done by the three recycle compressors:

$$\text{Minimize:} \quad f = C\left[F_3\frac{(H_9 - H_3)}{0.65} + F_5\frac{(H_{11} - H_5)}{0.65} + F_7\frac{(H_{13} - H_7)}{0.65}\right] \qquad (a)$$

The value 0.65 is the efficiency factor.

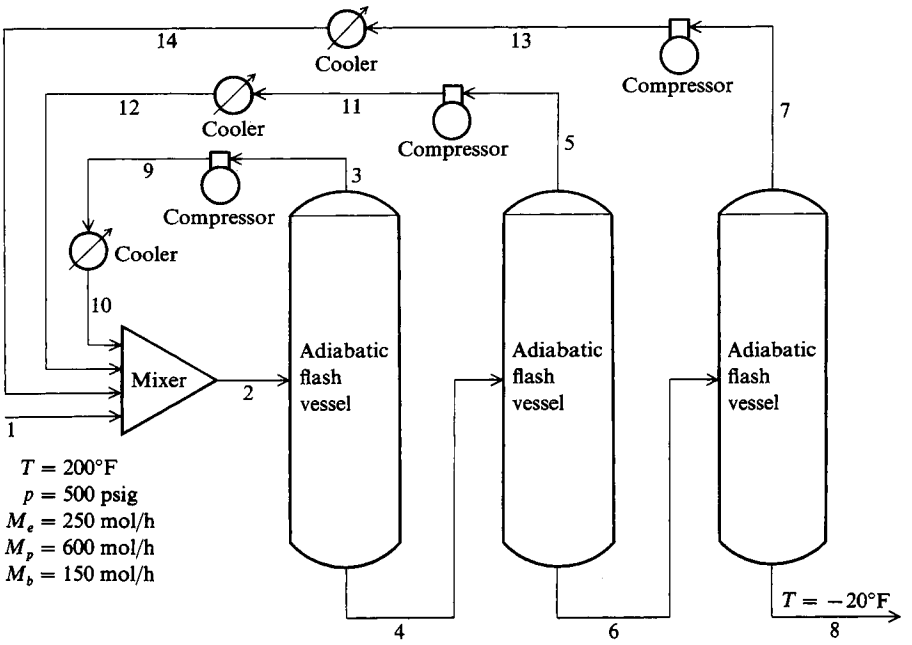

FIGURE E15.2
Flow diagram of light hydrocarbon refrigeration process.

TABLE E15.2A
Notation for Example 15.2

C	A constant denoting the cost of work per unit energy
F_j	Total molar flow rate of process stream j
H_j	Molar enthalpy of process stream j
K_i	Vapor–liquid equilibrium constant for component i
L_j	Liquid molar flow rate of process stream j
p_i	Pressure of process stream identified by subscript ($p_4 = p_3, p_6 = p_5$, and $p_8 = p_7$)
T_i	Temperature of process stream identified by subscript ($T_4 = T_3$, $T_6 = T_5$, and $T_8 = T_7$)
V_j	Vapor molar flow rate of process stream j
$x_{i,j}$	Liquid molar flow rate of component i in process stream j [i = 1 (ethane), 2 (propane), 3 (n-butane)]
$y_{i,j}$	Vapor molar flow rate of component i in process stream j [i = 1 (ethane), 2 (propane), 3 (n-butane)]

2 Inequality constraints. Three inequality constraints are involved: two relating pressures and one product temperature specification.

$$p_5 - p_3 \leq 0 \qquad (b\text{-}1)$$

$$p_7 - p_5 \leq 0 \qquad (b\text{-}2)$$

$$T_7 + 20 \leq 0 \qquad (b\text{-}3)$$

In addition, all 34 values of T_j, p_j, $x_{i,j}$, and $y_{i,j}$ have lower and upper bounds.

3 Equality constraints. The equality constraints (30 in all) are the linear and nonlinear material and energy balances and the phase relations.

3.1 *Mixer.*

Material balances:

$$\frac{0.5(y_{i,1} + x_{i,10} + x_{i,12} + x_{i,14} - y_{i,2} - x_{i,2})}{\max\{1, (y_{i,1} + x_{i,10} + x_{i,12} + x_{i,14} + y_{i,2} + x_{i,2})/2\}} = 0, \quad i = 1, \ldots, 3 \qquad (c)$$

Energy balance:

$$\frac{0.05(F_1H_1 + F_{10}H_{10} + F_{12}H_{12} + F_{14}H_{14} - F_2H_2)}{\max\{1, (F_1H_1 + F_{10}H_{10} + F_{12}H_{12} + F_{14}H_{14})/2\}} = 0 \qquad (d)$$

The denominators in this example are simply scaling factors in the respective constraints evaluated using the values of the variables in the numerator; 1 or the other term is picked, whichever is bigger. For example, the terms in the denominators of Equations (*c*) and (*d*) representing the average of the mass and energy, respectively, in and out, as well as the denominators of the following equations, are not needed for the balances—they are scaling factors (as are the multipliers 0.05 or 0.5) that are introduced to improve the conditioning of the matrices of partial derivatives of the constraints. Without such scaling, the non-linear programming code may not reach the optimal solution but instead terminate prematurely.

3.2 *Adiabatic flash vessels.*

Material balances:

$$\frac{0.5(y_{i,2} + x_{i,2} - y_{i,3} - x_{i,4})}{\max\{1, (y_{i,2} + x_{i,2} + y_{i,3} + x_{i,4})/2\}} = 0, \quad i = 1, \ldots, 3 \qquad (e)$$

$$\frac{0.5(x_{i,4} - y_{i,5} - x_{i,6})}{\max\{1, (x_{i,4} + y_{i,5} + x_{i,6})/2\}} = 0, \quad i = 1, \ldots, 3 \qquad (f)$$

$$\frac{0.5(x_{i,6} + y_{i,7} - x_{i,8})}{\max\{1, (x_{i,6} + y_{i,7} + x_{i,8})/2\}} = 0, \quad i = 1, \ldots, 3 \qquad (g)$$

Energy balances: In the energy balances the multiplier 0.05 is used to assist in scaling.

$$\frac{0.05(F_2H_2 - F_3H_3 - F_4H_4)}{\max\{1, (F_2H_2 + F_3H_3 + F_4H_4)/2\}} = 0$$

$$\frac{0.05(F_4H_4 - F_5H_5 - F_6H_6)}{\max\{1, (F_4H_4 + F_5H_5 + F_6H_6)/2\}} = 0$$

$$\frac{0.05(F_6H_6 - F_7H_7 - F_8H_8)}{\max\{1, (F_6H_6 + F_7H_7 + F_8H_8)/2\}} = 0$$

The values for the enthalpies of the streams in the database were based on the Curl–Pitzer correlations (Green, 1997). The enthalpies are calculated from correlations at zero pressure (functions of temperature and composition only) and then corrected via the enthalpy deviation:

$$H = H^0 - \left(\frac{H^0 - H}{T_c}\right)T_c \tag{h}$$

where H^0 is the stream molar enthalpy and the superscript 0 designates zero pressure, and T_c is the critical temperature. The enthalpy deviation term itself, $\Delta H/T_c$, is a function of the mole weighted average of the three critical properties: temperature, pressure, and compressibility.

3.3 *Energy balances for compressors.* For isentropic compression

$$\frac{0.05[T_9 - (T_3 + 459.69)(500.0/P_3)^{0.200} - 459.69]}{\max\{1, (T_3 + T_9)/2\}} = 0 \tag{i}$$

$$\frac{0.05[T_{11} - (T_5 + 459.69)(500.0/P_5)^{0.200} - 459.69]}{\max\{1, (T_5 + T_{11})/2\}} = 0 \tag{j}$$

$$\frac{0.05[T_{13} - (T_7 + 459.69)(500.0/P_7)^{0.200} - 459.69]}{\max\{1, (T_7 + T_{13})/2\}} = 0 \tag{k}$$

3.4 *Phase equilibria relations.* Evaluation of the K values for phase equilibria was based on the relation

$$K_i = \frac{\gamma_{il}\nu_i}{\phi_i} \tag{l}$$

where γ_{il} = activity coefficient in the liquid phase of component i evaluated from Hildebrand and Scott (Green, 1997)
ν_i = fugacity coefficient of component i in the liquid phase evaluated from Chao-Seader (Green, 1997)
ϕ_i = fugacity coefficient of component i in the vapor phase evaluated from Redlich-Kwong (Green, 1997)

Based on the notation of Table E15.2a, in stream j

$$K_i = \frac{y_{i,j-1}/V_j}{x_{i,j}/L_j} \tag{m}$$

To assist in scaling, Equation (m) is rearranged as follows:

$$x_{i,j}K_i\left(\frac{V_j}{L_j}\right) + x_{i,j} = y_{i,j-1} + x_{i,j}$$

$$x_{i,j} = \frac{(x_{i,j} + y_{i,j-1})L_j}{K_iV_j + L_j}$$

or

$$x_{i,j} - \frac{(x_{i,j} + y_{i,j-1})L_j}{K_iV_j + L_j} = 0$$

and divided by

$$\max\left\{1, x_{i,j} + \frac{(x_{i,j} + y_{i,j-1})L_j}{K_iV_j + L_j}\right\}$$

and multiplied by the factor 0.01:

$$\frac{0.01\{x_{i,2} - [(x_{i,2} + y_{i,2})L_2/(K_iV_2 + L_2)]\}}{\max\{1, x_{i,2} + [(x_{i,2} + y_{i,2})L_2/(K_iV_2 + L_2)]\}} = 0, \quad i = 1, \ldots, 3 \tag{n}$$

$$\frac{0.01\{x_{i,4} - [(x_{i,4} + y_{i,3})L_4/(K_iV_3 + L_4)]\}}{\max\{1, x_{i,4} + [(x_{i,4} + y_{i,3})L_4/(K_iV_3 + L_4)]\}} = 0, \quad i = 1, \ldots, 3 \tag{o}$$

$$\frac{0.01\{x_{i,6} - [(x_{i,6} + y_{i,5})L_6/(K_iV_5 + L_6)]\}}{\max\{1, x_{i,6} + [(x_{i,6} + y_{i,5})L_6/(K_iV_5 + L_6)]\}} = 0, \quad i = 1, \ldots, 3 \tag{p}$$

$$\frac{0.01\{x_{i,8} - [(x_{i,8} + y_{i,7})L_8/(K_iV_7 + L_8)]\}}{\max\{1, x_{i,8} + [(x_{i,8} + y_{i,7})L_8/(K_iV_7 + L_8)]\}} = 0, \quad i = 1, \ldots, 3 \tag{q}$$

In summary, the problem consists of 34 bounded variables (both upper bound and lower bounds) associated with the process, 12 linear equality constraints, 18 nonlinear equality constraints, and 3 linear inequality constraints.

4 Solution of the problem. It was not possible to use analytical derivatives in the nonlinear programming code because the energy balance equality constraints

and the process stream phase equilibria constraints involve the stream molar enthalpy H_j and the phase equilibrium constant K_{ij}, respectively. H_j was calculated at zero pressure and then corrected using the Watson acentric factor (Green, 1997). The correction for nonideality was based on correlated experimental data that cannot be differentiated analytically. The component phase equilibrium constant K_{ij} was calculated via the Redlich–Kwong equation of state; the vapor phase mixture compressibility factor z^v was determined as the largest of the three real roots from the virial equation:

$$z^3 - z^2 + C_1 z + C_2 = 0$$

where C_1 and C_2 are functions of the critical properties of the mixture. An analytical derivative of the vapor phase mixture compressibility with respect to the stream variables cannot be determined explicitly, and therefore, the derivative of the component phase equilibrium constant K_{ij} cannot be determined analytically.

As a consequence, the gradient of the objective function and the Jacobian matrix of the constraints in the nonlinear programming problem cannot be determined analytically. Finite difference substitutes as discussed in Section 8.10 had to be used. To be conservative, substitutes for derivatives were computed as suggested by Curtis and Reid (1974). They estimated the ratio μ_j of the truncation error to the roundoff error in the central difference formula

$$\frac{\partial f}{\partial x_j} = \frac{f(x + d_j) - f(x - d_j)}{2d_j}$$

where d_j is the step size, as follows:

$$\mu_j = \frac{-(d_j/2)[f(x + d_j) - 2f(x) + f(x - d_j)]}{d_j^2} \Bigg/ p\left|\frac{\partial f}{\partial x_j} \cdot x_j\right|$$

where p is the magnitude of the error incurred in the storage of a number in the computer. The Curtis–Reid method updates d_j on each calculation of a partial derivative from the relation

$$d_j^{k+1} = (d_j^k)\min\left\{1000, \sqrt{\frac{\mu_j^*}{\max\{u_j, 1\}}}\right\}$$

where u_j^* is the target value of the error ratio. To ensure that the truncation error calculation was not dominated by round-off error, Curtis and Reid suggested a value for u_j^* of 100 with an acceptable range of 10 to 1000.

The solution listed in Table E15.2B was obtained from several nonfeasible starting points, one of which is shown in Table E15.2C, by the generalized reduced gradient method.

TABLE E15.2B
Final solution of light hydrocarbon refrigeration process

Stream	$F/1000$ $\left(\frac{\text{lb mol}}{\text{h}}\right)$	T (°F)	p (psia)	$H/1000$ $\left(\frac{\text{Btu}}{\text{lb mol}}\right)$	Molar flow rates: Liquid/100 C_2H_6	Liquid/100 C_3H_8	Liquid/100 nC_4H_{10}	Vapor/100 C_2H_6	Vapor/100 C_3H_8	Vapor/100 nC_4H_{10}
1	1.00	200	500	6.90	0.00	0.00	0.00	2.50	6.00	1.50
2	2.97	115	500	2.07	10.6	9.15	1.66	5.63	2.46	0.221
3	1.29	80.6	306	4.69	0.00	0.00	0.00	9.04	3.58	0.259
4	1.68	80.6	306	0.0651	7.19	8.03	1.62	0.00	0.00	0.00
5	0.412	33.7	143	4.48	0.00	0.00	0.00	2.86	1.89	0.0749
6	1.27	33.7	143	−1.36	4.34	6.84	1.55	0.00	0.00	0.00
7	0.272	−20.0	511	4.09	0.00	0.00	0.00	1.84	0.843	0.0451
8	1.00	−20.0	511	−2.85	2.50	6.00	1.50	0.00	0.00	0.00
9	1.29	136	500	4.72	0.00	0.00	0.00	9.04	3.58	0.259
10	1.29	50.0	500	−0.373	9.04	3.58	0.259	0.00	0.00	0.00
11	0.412	174	500	6.06	0.00	0.00	0.00	2.86	1.19	0.0749
12	0.412	50.0	500	−0.386	2.86	1.19	0.0749	0.00	0.00	0.00
13	0.272	234	500	7.32	0.00	0.00	0.00	1.84	0.843	0.0451
14	0.272	50.0	500	−0.415	1.84	0.843	0.0451	0.00	0.00	0.00

TABLE E15.2C
Starting point 1 of light hydrocarbon refrigeration optimization

Stream	$F/100$ $\left(\frac{\text{lb mol}}{\text{h}}\right)$	T (°F)	p (psia)	$H/100$ $\left(\frac{\text{Btu}}{\text{lb mol}}\right)$	Molar flow rates: Liquid/1000 C_2H_6	Liquid/1000 C_3H_8	Liquid/1000 nC_4H_{10}	Vapor/100 C_2H_6	Vapor/100 C_3H_8	Vapor/100 nC_4H_{10}
1	1.00	200	500	6.90	0.00	0.00	0.00	2.50	6.00	1.50
2	3.50	103	500	1.79	1.48	0.952	0.167	6.85	1.95	0.149
3	1.47	68.3	300	4.46	0.00	0.00	0.00	1.14	3.12	0.208
4	2.02	68.3	300	−0.156	1.03	0.834	0.161	0.00	0.00	0.00
5	0.372	34.3	175	4.33	0.00	0.00	0.00	2.88	0.793	0.0471
6	1.65	34.3	175	−1.16	0.741	0.755	0.156	0.00	0.00	0.00
7	0.652	−8.17	150	3.46	0.00	0.00	0.00	4.91	1.55	0.0607
8	1.00	−81.7	150	−4.18	0.250	0.600	0.150	0.00	0.00	0.00
9	1.47	125	500	4.70	0.00	0.00	0.00	11.4	3.12	0.208
10	1.47	50.0	500	−0.260	1.14	0.312	0.0208	0.00	0.00	0.00
11	0.372	150	500	5.49	0.00	0.00	0.00	2.88	0.793	0.0471
12	0.372	50.0	500	−0.259	2.88	0.793	0.0471	0.00	0.00	0.00
13	0.652	303	500	8.55	0.00	0.00	0.00	4.91	1.55	0.0607
14	0.652	500	500	−0.290	0.491	0.155	0.0607	0.00	0.00	0.00

15.3 OPTIMIZATION USING MODULAR-BASED SIMULATORS

Over the past 40 years an enormous amount of time and considerable expense have been devoted to the development of modular-based process simulator codes. Figure 15.7 shows typical icons of modules found in a steady-state process simulator, and Figure 15.3 showed the details of one such module. In current practice, optimization meshed with modularly organized simulators prevails because (1) modules are easy to construct and understand, (2) addition and deletion of modules to and from a flowsheet is easily accomplished via a graphical interface without changing the solution strategy, (3) modules are easier to program and debug than sets of equations, and diagnostics for them easier to analyze, and (4) modules already exist and work, whereas equation blocks for equipment have not been prevalent. It seems appropriate, then, to mesh process models in the form of modules with optimization algorithms so that computer codes do not require wholesale rewriting.

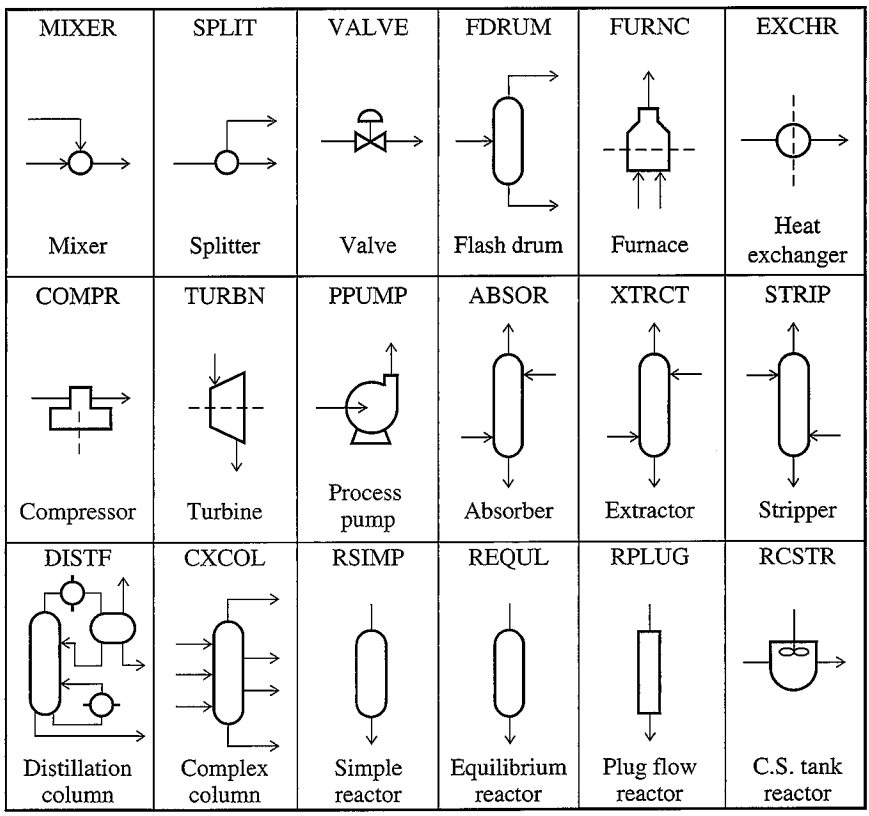

FIGURE 15.7
Typical process modules used in sequential modular-based flowsheeting codes with their subroutine names.

However, certain difficulties arise in doing this:

1. The input and output variables in a computer module are fixed so that you cannot arbitrarily introduce an output and generate an input, as can be done with an equation-based code.
2. When the modules are connected to one another as represented in a flowsheet, a long train of units may become coupled together for calculations. Thus, a set of modules may require a fixed precedence order of solution so that convergence of the calculations may be slower than in equation-based codes.
3. The modules require some effort to generate reasonably accurate derivatives or their substitutes, especially if a module contains tables, functions with discrete variables, discontinuities, and so on. Perturbation of the input to a module is the primary way in which a finite-difference substitutes for derivatives can be generated.
4. To specify a parameter in a module as a design variable, you need to feed back information around the module and adjust the parameter so that design specifications are met. This arrangement creates a loop exactly the same as a feedback of material or energy creates a recycle loop. Examine Figure 15.8. If the values of many design variables are to be determined, you might end up with several nested loops of calculations (which do, however, enhance stability).
5. Conditions imposed on a process (or a set of equations for that matter) may cause the unit physical states to move from a two-phase to a single-phase operation, or the reverse. As the code shifts from one module to another to represent the process properly, a severe discontinuity occurs in the objective function surface (and perhaps a constraint surface). Derivatives or their substitutes may not change smoothly, and physical property values may jump about.

In Section 15.1 we mentioned that two basic approaches for modular-based process simulators exist:

1. Sequential modular methods.
2. Simultaneous modular methods.

We next consider both methods.

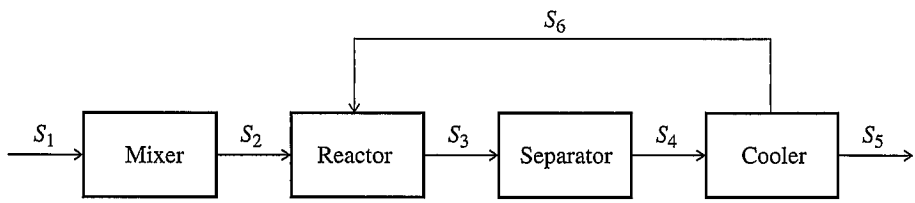

FIGURE 15.8
Modules in which recycling occurs; information (material) from the cooler module is fed back to the reactor, causing a loop.

15.3.1 Sequential Modular Methods

Two procedures are needed to implement efficient computations using sequential calculations in modular-based process simulators: one is *precedence ordering* and the other is *tearing*. Precedence ordering was briefly touched on at the end of Section 15.2 in connection with the partitioning and ordering of equations. The same concept applies to modules connected by loops of information flow. Partitioning the modules in a flowsheet into minimum-size subsets of modules that must be solved simultaneously can be executed by many methods. As with solving sets of equations, to reduce the computational effort you want to obtain the smallest block of modules that constitutes a loop in which the individual modules are tied together by the information flow of outputs and inputs. Between blocks, the information flow occurs serially.

How can you find all of the blocks connected together by information flows? A simple algorithm to isolate blocks is to trace a path of the flow of information (material usually, but possibly energy or a signal) from one module to the next through the module output streams. The tracing continues until either (1) a module in the path is encountered again, in which case all the modules in the path up to the repeated module form a group together that is collapsed and treated as a single module in subsequent tracing, or (2) a module or group with no output is encountered, in which case the module or group of modules can be deleted from the block diagram. As a simple example, examine the block diagram in Figure 15.9, which can be partitioned by the following steps.

Start with an arbitrary unit, say 4, and start tracing the path of information flow in any selected sequence; call this path I:

Start tracing:	$4 \rightarrow 5 \rightarrow 6 \rightarrow 4$	collapse as one set (456)
Continue tracing:	$(456) \rightarrow 2 \rightarrow 4$	collapse as one set (4562)
Continue tracing:	$(4562) \rightarrow 1 \rightarrow 2$	collapse as one set (45621)
Continue tracing:	$(45621) \rightarrow 7 \rightarrow 8 \rightarrow 7$	collapse as one set (78)
Continue tracing:	$(45621) \rightarrow (78) \rightarrow 9$	terminate tracing (no output)

The precedence order for path I is as follows:

$$(45621) \rightarrow (78) \rightarrow 9$$

To complete the search and add more modules to the precedence order, start on path II:

Start tracing:	$10 \rightarrow 3 \rightarrow 2$	terminate with 2 as 2 is in path I

The precedence order for path II is

$$(10) \rightarrow (3) \rightarrow (45621)$$

All of the modules from the block diagram have been included in the tracing, and no more paths have to be searched. The procedure identifies all the nested and outer loops. The overall precedence order is (10) → (3) → (45621) → (78) → (9).

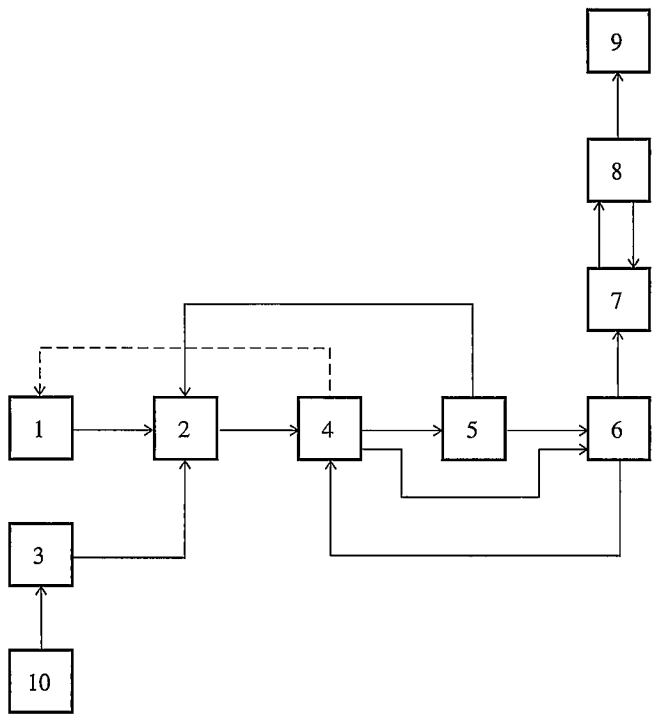

FIGURE 15.9
Block diagram to be partitioned.

Computer techniques to partition complex sets of modules besides the one described earlier can be found in Montagna and Iribarren (1988) and in Mah (1990). Simple sets can be partitioned by inspection.

From a computational viewpoint, the presence of recycle streams is one of the impediments in the sequential solution of a flowsheeting problem. Without recycle streams, the flow of information would proceed in a forward direction, and the calculational sequence for the modules could easily be determined from the precedence order analysis outlined earlier. With recycle streams present, large groups of modules have to be solved simultaneously, defeating the concept of a sequential solution module by module. For example, in Figure 15.8, you cannot make a material balance on the reactor without knowing the information in stream S6, but you have to carry out the computations for the cooler module first to evaluate S6, which in turn depends on the separator module, which in turn depends on the reactor module. Partitioning identifies those collections of modules that have to be solved simultaneously (termed **maximal cyclical subsystems, loops,** or **irreducible nets**).

To execute a sequential solution for a set of modules, you have to tear certain streams. **Tearing** in connection with modular flowsheeting involves decoupling the interconnections between the modules so that sequential information flow can take place. Tearing is required because of the loops of information created by recycle

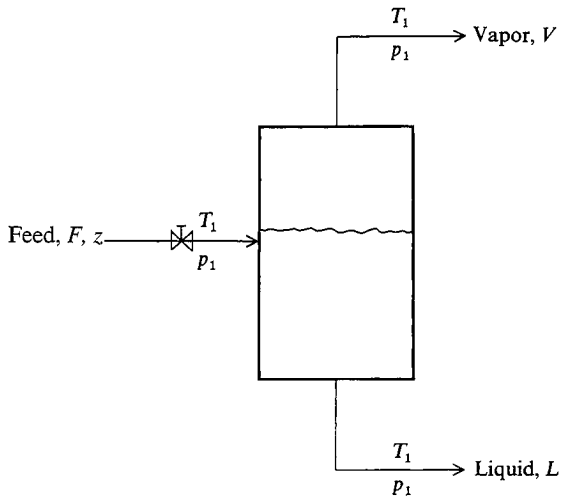

FIGURE 15.10
Vapor–liquid separator.

streams. What you do in tearing is to provide initial guesses for values of some of the unknowns (the tear variables), usually but not necessarily the recycle streams, and then calculate the values of the tear variables from the modules. These calculated values form new guesses, and so on, until the differences between the estimated and calculated values are sufficiently small. **Nesting** of the computations determines which tear streams are to be converged simultaneously and in which order collections of tear of tear streams are to be converged.

Physical insight and experience in numerical analysis are important in selecting which variables to tear. For example, Figure 15.10 illustrates an equilibrium vapor–liquid separator for which the combined material and equilibrium equations give the relation

$$\sum_{j=1}^{C} \frac{z_j(1 - K_j)}{1 - (V/F) + VK_j/F} = 0$$

where z_j is the mole fraction of species j out of C components in the feed stream, $K_j = y_j/x_j$ is the vapor–liquid equilibrium coefficient, a function of temperature, and the stream flow rates are noted in the figure. For narrow-boiling systems, you can guess V/F, y_j, and x_j, and use the preceding summation to calculate K_j and hence the temperature. This scheme works well because T lies within a narrow range. For wide-boiling materials, the scheme does not converge well. It is better to solve the preceding summation for V/F by guessing T, y, and x, because V/F lies within a narrow range even for large changes in T. Usually, the convergence routines for the code constitute a separator module whose variables are connected to the other modules via the tear variables. Examine Figure 15.11.

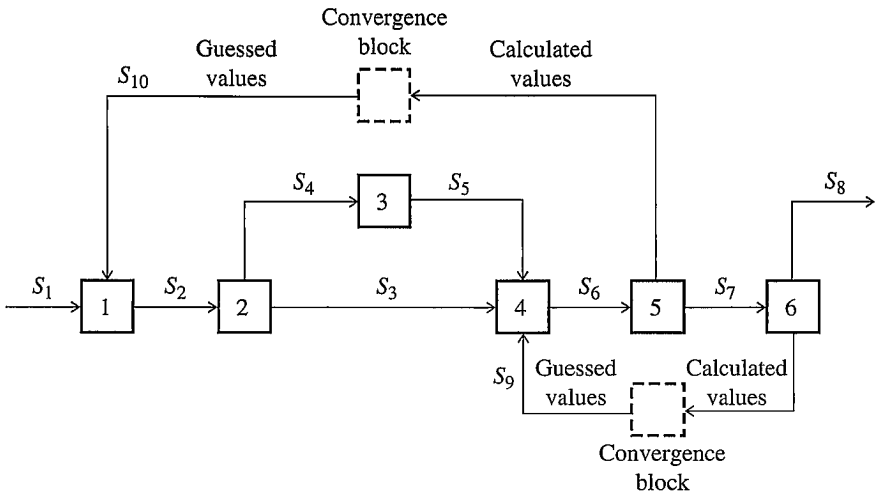

FIGURE 15.11
A computational sequence for modular flowsheeting. Initial values of both recycles are guessed, then the modules are solved in the order 1, 2, 3, 4, 5, and 6. Calculated values for recycle streams S9 and S10 are compared with guessed values in a convergence block, and unless the difference is less than some prescribed tolerance, another iteration takes place with the calculated values, or estimates based on them, forming the new initial guessed values of the recycle streams.

If the objective in selecting streams to tear is to minimize the number of the tear variables (Pho and Lapidus, 1973) subject to the constraint that each loop be broken at least once, this problem is an integer programming problem known as the covering set problem. Refer to Biegler et al. (1997) and Section 8.4.

Although it would logically be quite straightforward to nest the process simulator within the optimization code, and iteratively first satisfy the constraints represented by the simulator by running the simulator, and then applying the optimization code, this procedure is not particularly efficient. The preferred strategy is to insert into the nonlinear optimization problem format, Figure 15.4, the equations corresponding to the convergence blocks in Figure 15.11, namely

$$\tilde{\mathbf{h}}(\mathbf{x}, \mathbf{p}) = \mathbf{0} = \mathbf{x}_T^{(k+1)} - \tilde{\mathbf{f}}(\mathbf{x}_I, \mathbf{x}_D, \mathbf{x}_T^{(k)}, \mathbf{p})$$

where $\tilde{\mathbf{h}}$ = set of equations involving the tear variables.
$\tilde{\mathbf{f}}$ = set of functions that compute the values of the tear variables for the next iteration $(k + 1)$ as the output of a module using the values of tear variables from the previous iteration k.
$\mathbf{x}_T$ = vector of tear variables.
$\mathbf{p}$ = equipment parameter vector

in lieu of using the convergence blocks in the process simulator to determine the values of the tear variables. This procedure saves many iterations through nested loops in the process simulator.

With the preceding implementation, the optimization problem can be solved either via a GRG algorithm or SQP algorithm. Each evaluation of the constraints and objective function requires a full pass through the process simulator. Additional passes are needed to develop the gradients with respect to $\mathbf{x}$ (including the tear variables). Then, the search direction can be obtained as indicated in Figure 15.4 by solving the following QP subproblem.

$$\begin{aligned}
\text{Minimize:} \quad & \nabla^T f(\mathbf{x}_I, \mathbf{x}_D, \mathbf{x}_T, \mathbf{p})\mathbf{s} + \tfrac{1}{2}\mathbf{s}^T\mathbf{B}\mathbf{s} \\
\mathbf{s} & \\
\text{Subject to:} \quad & \mathbf{h}(\mathbf{x}_I, \mathbf{x}_D, \mathbf{x}_T, \mathbf{p}) + \nabla^T \mathbf{h}(\mathbf{x}_I, \mathbf{x}_D, \mathbf{x}_T, \mathbf{p})\mathbf{s} = \mathbf{0} \\
& \tilde{\mathbf{h}}(\mathbf{x}_I, \mathbf{x}_D, \mathbf{x}_T, \mathbf{p}) + \nabla^T \tilde{\mathbf{h}}(\mathbf{x}_I, \mathbf{x}_D, \mathbf{x}_T, \mathbf{p})\mathbf{s} = \mathbf{0} \\
& \mathbf{g}(\mathbf{x}_I, \mathbf{x}_D, \mathbf{x}_T, \mathbf{x}_S, \mathbf{p}) + \nabla^T \mathbf{g}(\mathbf{x}_I, \mathbf{x}_D, \mathbf{x}_T, \mathbf{x}_S, \mathbf{p})\mathbf{s} \geq \mathbf{0}
\end{aligned}$$

After determining the search direction $\mathbf{s}$, an approximate line search is carried out to get the values of $\mathbf{x}_I$ and $\mathbf{x}_T$ for the next iteration.

Of the various versions of the SQP algorithm, the infeasible path reduced SQP has been the most widely used in commercial process simulators. One technique favored by programmers (Lang and Biegler, 1987) is to make just one pass through the process flowsheet simulator before adjusting the values of the decision and tear variables rather than spending considerable computation time satisfying the constraints involved in loops. This procedure has some merit because the value of the variables determined by a fairly precise solution of the loops on one iteration of the optimization program will probably no longer be satisfactory on a subsequent iteration.

15.3.2 Simultaneous Modular Methods

One of the earlier approaches to emulating equation-based optimization using process simulators was to develop by least squares polynomial functions (quadratic being the simplest) to approximate the input–output relations for a module, and for the phase relations (Mahelec et al., 1979; Biegler, 1985; Chen and Stadtherr, 1984; Parker and Hughes (1981); Schmid and Biegler, 1994a). Then, the equations could be used as constraints in an optimization code. Some disadvantages of such an approximation strategy are that (1) adequate approximation of the module may not be possible with simple relations, and (2) the optimum of the approximate model may not lie near the optimum of the rigorous model as ascertained via a more rigorous solution. Nevertheless, such modeling schemes avoid some of the difficulties encountered in closure and convergence of the recycle loops each time the process simulator is called. You obtain the speed and flexibility of the equation-based mode while using as models equations representing the modules.

Use of the reduced space SQP mentioned in Section 15.1 has facilitated the implementation of simultaneous modular optimization. The modeling equations representing the individual modules are not explicitly made part of the optimization problem. Instead, the equations are solved by taking successive steps using Newton's

TABLE 15.2
Comparison of the results of equation-based and simultaneous modular-based optimization for two connected distillation columns

Number of variables:	
Decision	4
Outputs, inputs, and so on	47
Internal	114
Number of equality constraints:	
Simultaneous modular strategy*	47
Equation-based strategy	161
Number of iterations (CPU time in seconds)	
Simultaneous modular, SQP	4 (3.4)
Equation-based, SQP	4 (3.3)

Abbreviations: CPU = central processing unit; SQP = successive quadratic programming.
*Method of Schmid and Biegler (1994b)

method for the individual modules. In addition, as proposed by Schmid and Biegler (1994b), a line search is employed that does not require that the Lagrange multipliers associated with the equality constraints be calculated explicitly, an important saving in the case of code modifications. Derivatives are presumed calculated analytically or by finite-difference methods as described in Section 15.3.3. As an example, Table 15.2 lists the results of Schmid and Biegler for the optimization of a hydrodealkylation process (1994b). Comparison of a simultaneous modular strategy with an equation-oriented strategy indicates that both yield equivalent results.

15.3.3 Calculation of Derivatives

Effective computer codes for the optimization of plants using process simulators require accurate values for first-order partial derivatives. In equation-based codes, getting analytical derivatives is straightforward, but may be complicated and subject to error. Analytic differentiation ameliorates error but yields results that may involve excessive computation time. Finite-difference substitutes for analytical derivatives are simple for the user to implement, but also can involve excessive computation time.

For modular-based process simulators, the determination of derivatives is not so straightforward. One way to get partial derivations of the module function(s) is by perturbation of the inputs of the modules in sequence to calculate finite-difference substitutes for derivatives for the torn variables. To calculate the Jacobian via this strategy, you have to simulate each module $(C + 2)\, n_T + n_F + 1$ times in sequence, where C is the number of chemical species, n_T is the number of torn streams, and n_F is the number of residual degrees of freedom. The procedure is as follows. Start with a tear stream. Back up along the calculation loop until an unperturbed independent variable $x_{I,i}$ in a module is encountered. Perturb the independent variable,

and calculate the resulting dependent and tear variables in that module and all downstream modules in the calculation loop. (Dependent variables upstream are not affected.) Evaluate the finite-difference approximations for the gradients of f, $\mathbf{g}$, $\mathbf{h}$, and $\tilde{\mathbf{h}}$ with respect to each $\mathbf{x}_{I,i}$ by using a forward-difference formula in which the values of $\mathbf{x}_D$ are those from the perturbed calculations and the values of $\mathbf{x}_I$, except $\mathbf{x}_{I,i}$ are perturbed values.

One at a time, perturb the elements of the tear variable $\mathbf{x}_{T,i}$. Calculate the dependent variables, and evaluate the tear equations. Calculate the gradients of f, $\mathbf{g}$, $\mathbf{h}$, and $\tilde{\mathbf{h}}$ with respect to each $\mathbf{x}_{T,i}$ by a forward difference equation in which the $\mathbf{x}_D$ are the perturbed values and $\mathbf{x}_I$ are the unperturbed values.

Another way to calculate the partial derivatives is possible. Figure 15.12 represents a typical module. If a module is simulated individually rather than in sequence after each unknown input variable is perturbed by a small amount, to calculate the Jacobian matrix, $(C + 2)n_{ci} + n_{di} + 1$ simulations will be required for the ith module, where n_{ci} = number of interconnecting streams to module i and n_{di} = number of unspecified equipment parameters for module i. This method of calculation of the Jacobian matrix is usually referred to as full-block perturbation.

Wolbert et al. in 1991 proposed a method of obtaining accurate analytical first-order partial derivatives for use in modular-based optimization. Wolbert (1994) showed how to implement the method. They represented a module by a set of algebraic equations comprising the mass balances, energy balance, and phase relations:

$$\boldsymbol{\phi}_k(\mathbf{u}_k, \mathbf{x}_k) = \mathbf{0} \tag{15.1}$$

where $\boldsymbol{\phi}_k(\mathbf{u}_k, \mathbf{x}_k)$ = set of functions representing the behavior of the kth module, i.e., the model for module k
$\mathbf{u}_k$ = vector of inputs to the kth module
$\mathbf{x}_k$ = vector of outputs from the kth module

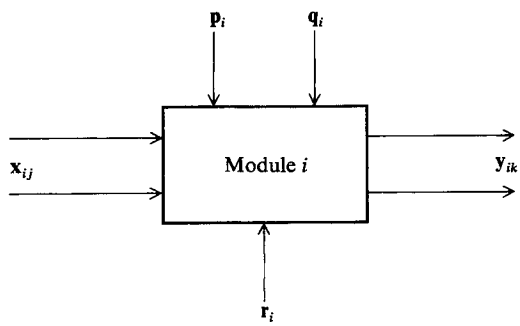

FIGURE 15.12
A typical module showing the input stream vectors $\mathbf{x}_{ij}$, output stream vectors $\mathbf{y}_{ik}$, specified equipment parameter vector $\mathbf{p}_i$, unspecified equipment parameter vector $\mathbf{q}_i$, and the retention (dependent) variable vector $\mathbf{r}_i$.

The analytical derivatives $(\partial \phi_{k_i}/\partial x_{k_j})$ and $(\partial \phi_{k_i}/\partial u_{k_j})$, and the sensitivity coefficients $(\partial x_{k_i}/\partial u_{k_j})$ can be obtained directly from Equation (15.1).

As to the automatic generation of exact derivatives in existing modular-based process simulator codes directly from the code itself, refer to Griewank and Corliss (1991) or Bischof et al. (1992).

EXAMPLE 15.3 EXTRACTIVE DISTILLATION DESIGN

This example shows the application of optimization of a process using HYSYS software. Refer to the website, www.mhhe.com/edgar, associated with this book.

EXAMPLE 15.4 MAXIMIZING OPERATING MARGIN

This example shows the application of optimization of a process using Aspen software. Refer to the website, www.mhhe.com/edgar, associated with this book.

15.4 SUMMARY

Commercial process simulators have added optimization capabilities the specific details of which are naturally proprietary, but the general features of these codes are described in this chapter. Very large scale optimization problems of considerable economic value can be treated as shown by the examples presented earlier, and in the future improvements in power, robustness, speed of execution, and user-friendly interfaces of computers and software can be expected to expand the scope of optimization of large scale problems.

REFERENCES

Aspen Technology, Inc. *Aspen Plus, Aspen Custom Modeler, Dynaplus, Split, Advent, Adsim.* Cambridge, MA (1998).

Biegler, L. T. "Improved Infeasible Path Optimization for Sequential Modular Simulators—I: The Interface." *Comput Chem Eng* **9:** 245–256 (1985).

Biegler, L. T.; I. E. Grossmann; and A. W. Westerburg. *Systematic Methods of Chemical Process Design.* Prentice-Hall, Upper Saddle River, NJ (1997).

Bischof, C.; A. Carle; G. Corliss; A. Griewank; et al. *ADIFOR Generating Derivative Codes for Fortran Programs.* Preprint MCS-P263-0991, Argonne National Lab. (1992).

ChemCAD. Chemstations. Houston, TX (1998).

Chen, H. S.; and M. A. Stadtherr. "A Simultaneous-Modular Approach to Process Flowsheeting and Optimization: I. Theory and Implementation." *AIChE J* **30:** 1843–1856 (1984).

Curtis, A. R.; and J. K. Reid. "The Choice of Step Lengths When Using Differences to Approximate Jacobian Matrices." *J Inst Math Its Appl* **13:** 121–140 (1974).

Duff, I. S.; A. M. Erisman; and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford Univ. Press, New York (1989).

Fiacco, A. V. "Sensitivity Analysis of Nonlinear Programming Using Penalty Function Methods." *Math Program* **10:** 287–311 (1976).

Gill, P. E.; W. Murray; and M. H. Wright. *Practical Optimization*. Academic Press, New York (1981).

Green, D. W., ed. *Perry's Chemical Engineering Handbook*. Section 4, 7th edition. McGraw-Hill, New York (1997).

Griewank, A.; and G. F. Corliss, eds. *Automatic Diffentiation of Algorithms: Theory, Implementation and Application*. SIAM, Philadelphia (1991).

Gunderson, T.; and I. E. Grossmann. "Improved Optimization Strategies for Automated Heat Exchanger Network Synthesis Through Physical Insights." *Comput Chem Eng* **14:** 925–944 (1990).

Himmelblau, D. M., ed. *Decomposition of Large Scale Problems*. North-Holland Publ., Amsterdam (1973).

Hyprotech Ltd. *HYSYM, HYSYS, HYCON*. Calgary, Alberta (1998).

Intelligen, Inc. *Documentation for EnviroPro Designer and BatchPro Designer*. Scotch Plains, NJ (1999).

Lang, Y. D.; and L. T. Biegler. "A Unified Algorithm for Flowsheet Optimization." *Comput Chem Eng* **11:** 143–158 (1987).

Lowery, R. P.; B. McConville; F. H. Yocum; and S. R. Hendon. *Closed-Loop Real Time Optimization of Two Bisphenol-A Plants*. Paper presented at the National AIChE Meeting, Houston, TX, Mar. 28–Apr. 1, 1993.

Mah, R. H. S. *Chemical Process Structures and Information Flows*. Butterworths (1990).

Mahalec, V.; H. Kluzik; and L. B. Evans. *Simultaneous Modular Algorithm for Steady State Flowsheet Simulation and Design*. Paper presented at the 12th European Symposium on Computers in Chemical Engineering. Montreaux, Switzerland (1979).

Montagna, J. M.; and O. A. Iribarren. "Optimal Computation Sequence in the Simulation of Chemical Plants." *Comput Chem Eng* **12:** 12–14 (1988).

Parker, A. P.; and R. R. Hughes. "Approximate Programming in Chemical Processes—1." *Comput Chem Eng* **5:** 123–133 (1981).

Perkins, J. D. "Plantwide Optimization—Opportunities and Challenge." In *Foundations of Computer-aided Process Operations*. J. F. Pekny; G. E. Blau, eds. American Institute of Chemical Engineering. New York (1998), pp. 15–26.

Pho, T. K.; and L. Lapidus. "An Optimum Tearing Algorithm for Recycle Streams." *AIChE J* **19:** 1170–1181 (1973).

Ramirez, W. F. *Process Control and Identification*. Academic Press, New York (1994).

Schmid, C.; and L. T. Biegler. "Quadratic Programming Algorithms for Reduced Hessian SQP." *Comput Chem Eng* **18:** 817–832 (1994a).

Schmid, C.; and L. T. Biegler. "A Simultaneous Approach for Flowsheet Optimization with Existing Modeling Procedures." *Trans Inst Chem Eng* **72A:** May (1994b).

Seider, W. D.; J. D. Seader; and D. R. Lewin. *Process Design Principles*. Wiley, New York (1999).

Simulation Sciences, Inc. *Documentation for ROMEO (Rigorous On-line Modeling with Equation-based Optimization*. Brea, CA (1999).

Simulation Sciences, Inc. *PRO/II, Provision, Protiss, Hextran*. Brea, CA (1998).

Westerberg, A. *Advanced System for Computations in Engineering Design*. Report No. ICES 06-239-98, Institute for Complex Engineered Systems, Carnegie-Mellon University (1998).

Wolbert, D.; X. Joulia; B. Koehret; and L. T. Biegler. "Flowsheet Optimization and Optimal Sensitivity Analysis Using Analytical Derivatives." *Comput Chem Eng* **18:** 1083–1095 (1994).

Wolbert, D.; X. Joulia; B. Koehret; and M. Pons. "Analyse de Sensibilitie pour l'Optimisation des Procedes Chimique." 3 d Congres de Genie des Procedes, Compiegne, France. *Rec Prog Genie Proc* **5:** 415–420 (1991).

Zaher, J. J. *Condition Modeling*. Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh PA (1995).

SUPPLEMENTARY REFERENCES

Alkaya, D.; S. Vasantharajan; and L. T. Biegler. "Generalization of a Tailored Approach for Process Optimization." *Ind Eng Chem Res* **39** (6): 1731–1742 (2000).

Balakrishna, S.; and L. T. Biegler. "A Unified Approach for the Simultaneous Synthesis of Reaction, Energy and Separation Systems." *Ind Eng Chem Res* **32:** 1372–1382 (1993).

Chen, H. S.; and M. A. Stadtherr. "A Simultaneous Modular Approach to Process Flowsheeting and Optimization." *AIChE J* **31:** 1843–1856 (1985).

Diwekar, U. M.; I. E. Grossmann; and E. S. Rubin. "MINLP Process Synthesizer for a Sequential Modular Simulator." *Ind Eng Chem Res* **31:** 313–322 (1992).

Grossmann, I. E.; and M. M. Daichendt. "New Trends in Optimization-Based Approaches to Process Synthesis." *Comput Chem Eng* **20:** 665–683 (1996).

Kisala, T. P.; R. A. Trevino-Lozano; J. F. Boston; H. I. Britt; et al. "Sequential Modular and Simultaneous Modular Strategies for Process Flowsheet Optimization." *Comput Chem Eng* **11:** 567–579 (1987).

Kokossis, A. C.; and C. A. Floudas. "Optimization of Complex Reactor Networks—II. Nonisothermal Operation." *Chem Engr Sci* **49** (7): 1037–1051 (1994).

Kravanja, Z.; and I. E. Grossmann. "Prosyn: An MINLP Process Synthesizer." *Comput Chem Engr* **14:** 1363–1378 (1990).

Lang, Y. D.; and L. T. Biegler. "A Unified Algorithm for Flowsheet Optimization." *Comput Chem Eng* **11:** 143–158 (1987).

Pistikopoulos, E. N.; and I. E. Grossmann. "Optimal Retrofit Design for Improving Process Flexibility in Nonlinear Systems—I. Fixed Degree of Flexibility." *Comput Chem Eng* **13:** 1003–1016 (1989).

Quesada, I.; and I. E. Grossmann. "Global Optimization of Bilinear Process Networks with Multicomponent Streams." *Comput Chem Eng* **19:** 1219–1242 (1995).

Raman, R.; and I. E. Grossmann. "Symbolic Integration of Logic in Mixed Integer Linear Programming Techniques for Process Synthesis." *Comput Chem Eng* **17:** 909–928 (1993).

Turkay, M.; and I. E. Grossmann. "Logic-Based MINLP Algorithms for the Optimal Synthesis of Process Networks." *Comput Chem Eng* **20:** 959–978 (1996).