1983

Thoughts on a future equation-oriented flowsheeting system

Arthur W. Westerberg
Carnegie Mellon University

Dean R.jt. auth. Benjamin

THOUGHTS ON A FUTURE EQUATION-ORIENTED FLOWSHEETING SYSTEM

by

A\W. Westerberg & D.R. Benjamin

December, 1983

DRC-06-51-83

THOUGHTS ON A FUTURE EQUATION-ORIENTED FLOWSHEETING SYSTEM

by

Arthur W. Westerberg
Dean R. Benjamin

Department of Chemical Engineering
and
The Design Research Center

Carnegie-Mellon University
Pittsburgh, PA 15213

Paper presented at 25th CONICET Anniversary International Conference on New Developments Toward Technologies with Low Energy Requirements, INTEC, Santa Fe, Argentina, August 1983.

This material is based upon work supported by NSF Grant and by a grant from Amoco.

Abstract

This paper describes the desirable attributes of a future equation oriented flowsheeting system. Areas covered include user/program communications, modeling considerations for complex flowsheets and finally the solving of models.

1. Introduction

In this paper we shall "dream" about the capabilities of a future equation-based flowsheeting system.

Such a system should aid a design engineer to set up and solve complex process models. It should have much broader capabilities than current sequential modular programs, but these added capabilities should not make it unusable by an ordinary mortal.

Such a system should solve steady-state, dynamic and optimization problems. It should be friendly; in fact, we believe it can be intelligent in its interaction with the user. The designer should be able to save results and readily retrieve them for subsequent modeling or report writing.

The topics we will discuss are user/program interaction, how one could use such a system to create a complex model and how one could then solve such a model. The paper is only a start at examining these ideas. We hope it serves to broaden the reader's understanding of the large number of issues involved in creating a useful design aid of this type.

2. Communications

2.1. Types of Communication

We start by brainstorming the types of communications that will occur with a design aid of the type being considered. The system is to be useful for a designer to set up and solve large complex process models, to save results and to retrieve and report on them. Experience has shown clearly that such systems must contain extensive libraries of building blocks which the designer connects together to create his model. There are thus two types of users: those who create the libraries and the design engineers.

• The system users must enter and run their programs; they must therefore have facilities to debug them. Finally the need to save and retrieve earlier runs and to create arbitrarily formatted reports suggest the system should be tied to a database system.

An interesting question arises. Is it reasonable that the design engineers and the library creators should, be provided with the same tools? If possible then only one set of tools need be created, and both types of users would have the full power of such a system to use. In the next section we will discuss the nature of models and discover that a model of a complex system is often created as a hierarchy, i.e. as models which are built of models which are built of models, etc. A final flowsheet is nothing more than the highest level of model so far created. It can be constructed using the same language and tools used to construct lower level models. Indeed, the libraries of models can be viewed as being the same as saving results for flowsheet runs so perhaps even the saving and retrieving functions are the same.

Again then, we appear to need

1. an ability to construct models
2. an ability to run and debug them
3. an ability to save, retrieve and report results to the user.

We seem also to be implying the library creators and the designers are doing the same things so they can share common tools.

2*2. Form of the Communication

We believe the method of communication for building a model is through the use of a specially designed nonprocedural language. It should be one that can read and handle algebraic expressions, as the models will be constructed of such expressions. Until we examine the nature of the models, we will postpone our discussion of the language.

To run and debug, the tools must be interactive. The nature of the debugging begs for such an interface. Debugging normally consists of short "transactions[11] requiring fast "turnaround[11] and short "response times[11] (Benjamin, et al, 1981). Production runs which likely will not have short response times can be run in a batch mode through the use of "command files'* and so also are readily accommodated by interaactive tools.

Debugging requires easy access to "everything," a feature we tried in part to put into ASCEND-II by making every variable value and every equation residual available to the user by name and at any time during a run. One must also be able to "single step" through a run, again a feature we attempted in ASCEND-II by providing a wealth of commands that break down the execution of a model into a sequence of small steps, the consequence of each being understandable by the user. A useful feature which we do not have in ASCEND-II is to review very selectively the execution of a model after it has been run. This feature suggests the selective saving and retrieval of information during a run. The counterpart of this step with conventional flowsheeting systems is the printing of a thick pile of output during an execution.

As stated earlier, the capability to save and retrieve earlier models and runs made with them, along with report generation based on such results, would seem to suggest tying such an aid to a database management system. As we have not attempted such an exercise we cannot delve into any of the details of this idea here. We can, however, point out issues one must consider.

To be useful such a storage and retrieval system must allow the user to search and discover which of the old runs is the one he is attempting

to locate* If the system is to support a user who has not written down extensive notes to himself, it must provide the capability to search and to locate descriptive information. The notion of providing the user with a picture on a CRT display of an office with a desk, file cabinets, etc., to which he can point when searching is very appealing. Models worked on recently could be kept around on the "desk top.[11] Those worked on last month could be filed in a "cabinet." Such visual cues to aid in searching seem extremely appealing. Xerox has created such a file management system for their in-house system.

Graphics could play a significant role in retrieval. Dreaming again, it would be extremely useful if a simple picture of the model could be retrieved when searching. The picture could be in the form of a simple process flow diagram (PFD), showing that the model is of a reactor followed by a flash with the flash bottoms recycling, etc.

Having located one or several old models together with results generated by them, one may wish to generate a. report and perhaps plots based on an analysis of the collected set of results. Cherry (1975) examined these ideas in his Ph.D. thesis. In addition though one may wish to construct a new model from parts of each of the old models. One needs a language to deal conveniently with such an exercise. Also one may wish to use the numbers generated for these earlier models to aid in initializing the calculations for the newly constructed model.

We shall Leave these issues here because, as yet, we have not thought seriously how to solve them.

2.3. Documentation

The means need to exist to document and retrieve that documentation for models which have been created. Thought must therefore be given to the modes of documentation possible. Online documentation seems appropriate. The best form would be for models to be self documenting, requiring the language to be close to English (or Spanish, etc.) and not confusing in intent. Documentation can be considerably more elegant than simply displaying back the program, however.

Let's dream a bit. It would be extremely useful if the online documentation were like a user's manual but with many access routes. First one should be able to read it like a book written in chapters, sections, subsections, etc. One should be able to access it through a table of contents. It would also be useful to allow crossreferencing within the documentation to other interesting sections related to the one being read. An entry by an index of keywords is also of value so one can consider a topic and find one's way to the correct part of the documentation. Finally it would be exceptionally convenient if, while the program is executing, it kept updated pointers to related portions in the documentation so a simple "help me[1]" would lead one directly to relevant aid. We discuss briefly such a help system in Benjamin et al (1981).

Would it not be ideal if the tools to create such documentation were available? Would it perhaps even be useful if the model developer had to document within such a system to create the model in the first place? And, would it not be convenient if somehow the document creation system could automatically use the model itself as part of the documentation, indexing it, etc.?

2.4. General Language Considerations

This section is to alert the reader to the problems associated with language design. The special language one invents for the user to communicate with the system should be very carefully designed. There has been considerable thought given to language design in computer science, and these resources should be tapped when developing a language. For example, one attribute the language should have is an "elegance" that allows a user to guess the command syntax for commands he has not yet used by assuming they must be similar to the syntax of commands he has already used. Questions about "strong typing" of variables arise. Also can the language cater to an expert? Should it? Perhaps not as it may be significantly less selfdocumenting.

In Benjamin et al (1981) we raise a number of issues relating to the language that one should create for a design engineer. We argue that for flowsheet calculations the appropriate level to interface the user is with the equations and variables. This will leave most readers aghast, but ultimately, when a calculation fails, it is to this level of detail that the engineer often must gain access to see why. He may look and decide it is not of help, but invariably he will ask to see the modeling equations.

We also argue, as we have already indicated earlier, that a major advantage of the equation solving approach is that the model can be stated independently of how it is to be solved. The solution algorithm becomes a separate aspect considered when solving but not when creating.

2.5. Nature of the Communication

We have considered various aspects of the user/program interface. There is another aspect worth considering which is whether the communication is active or passive. Three types of interaction will be mentioned here: friendly, graceful and intelligent.

Friendly interaction is a very popular concept at this time. It suggests that the user who is either a novice or who has been away for a while can discover how to use the system by searching through the help facilities provided. Graceful interaction goes one step further. It is friendly but also tolerant of errors, sometimes correcting them if they are obvious or suggesting alternatives if not. Also a graceful interaction will allow the user to invent his own abbreviations for routinely used commands. Graceful interaction has traces of being active in that it will attempt tor take some corrective actions on its own.

Intelligent: interaction can be thought of as a level of interaction where the computer does not simply let the user know what is permissible, it also advises the user which of the permissible next steps are the better ones to take. We can dream a bit again. It would be interesting to have a design aid that could understand the activity of setting up and solving complex models. It could know that the better strategy to do this activity Is to evolve from no model to the final complex model by setting up parts, testing them, correcting them, testing, and so forth. It could also advise the user on what steps to take when a test fails. It could perhaps know that distillation columns cannot be solved when asked to find solutions requiring too high a recovery for the key component. Such knowledge suggest a level of understanding beyond providing a user's manual. The system would have to detect what the user was doing to be able to offer advice.

The technology needed to develop this type of interaction is being developed in computer science under the label "expert systems.[11] These systems are an attempt to put knowledge into a computer so it can know when to apply the knowledge as well as sound like an expert when applying it.

We include this section to alert people who wish to develop computer aids as to the potential of new ideas that are becoming available.

3. Modeling

3«1« A Mathematical Representation

We consider here the mathematical language needed to formulate a model of the type needed for doing flowsheet simulation, design and optimization calculations. We shall introduce the concept of general models (Stateva and Westerberg (1983)) to broaden the range of models previously allowed in equation-oriented flowsheeting systems. Models are, in our definition, the equations and variables and not the solution method, which we consider to be a separate semantic entity.

Simple Steady State Model;

$$\underline{g}(\underline{x}) = \underline{0}$$

where $\underline{f}(\underline{x})$ is a set of m equations in n+m unknowns $\underline{x}$» To solve requires that n of the unknowns be specified by other means and the remaining are found by solving the m nonlinear algebraic equations in m unknowns.

Constrained Steady State Model;

where $g(\underline{x})$ is again a set of m equations in n+m unknowns. However, now some (or all) of the variables are restricted to be within specified bounds. (By the use of slack variables this formulation is equivalent to writing arbitrary inequality constraints.)

To solve one is asking to find a feasible point for the problem. This problem is typically solved as an optimization problem.

General Function Models (Stateva and Westerberg (1983)):

We now significantly enrich the structure of the models allowed by permitting the use of both continuous and discrete variables. Without this added structure, equation-oriented flowsheeting models cannot mimic the models used within conventional sequential modular flowsheeting systems.

We define a general function to be one of the form

$$g_i(\underline{x}) = \begin{cases} \hat{g}_i(x) & \text{if } i_i = \text{TRUE} \\ g_i(x) & \text{otherwise} \end{cases}$$

where $\pounds_i$ and $\%_i$ are equation alternatives to be used depending on the value of the logical (discrete) variable I_i. Either may be "null."

I_i is defined by

$$X_i * \text{logical expression}$$

An example is that $g_i(x)$ is the relationship between the fanning friction factor and Reynolds[1] number in a pipe. If the Reynolds[1] number exceeds 2100, turbulent flow is to be assumed giving one form. If it is below 2100, then laminar flow is assumed giving another form.

Stateva and Westerberg explore the range of problem types for which this structure is useful. An example is a traditional vapor/liquid flash model where the operation may be one phase (superheated vapor or subcooled liquid) or two phase. Another example is to evaluate

$$\Delta T_{lm} = \frac{(T_1 - t_1) - (T_2 - t_2)}{\ln\left(\frac{T_1 - t_1}{T_2 - t_2}\right)}$$

anywhere including the limit as T_1-t_1 becomes equal to T_2-t_2. A third example is to find the largest real root of a cubic polynomial of the form

$$x^3 + a_2x^2 + a_1x + a_0 = 0$$

and so forth.

Pictorially the model operates in "regions" as depicted by Figure 1. The given conditions may allow for a solution to reside in only one region while the initial guess may be in another. One can imagine a strategy which can move automatically from the incorrect region to the correct one.

Current models in sequential modular flowsheeting systems allow for general functions to be used in their definition, although this point is never stated nor perhaps fully appreciated. We are unaware of any equation-oriented flowsheeting systems attempting to deal with general function models, however.

Stateva and Westerberg discuss a simple strategy which appears to be effective for solving a number of problems of this form. Basically it involves guessing the discrete decision values which are not specified, solving the corresponding well-behaved region equations for the dependent continuous variables, reguessing the unspecified discrete decision values, etc. Contrast this with the current sequential modular "ad hoc" approach that imbeds the discrete decisions within each model so the outer flowsheeting system is being handed back discontinuous model behavior as it is trying to converge recycles.

Two approaches to handling regions are possible. One is to convert the problem to a feasible point problem, which is usually formulated as an optimization problem. Here the expressions defining the logical variables become constraints for the region model equations.

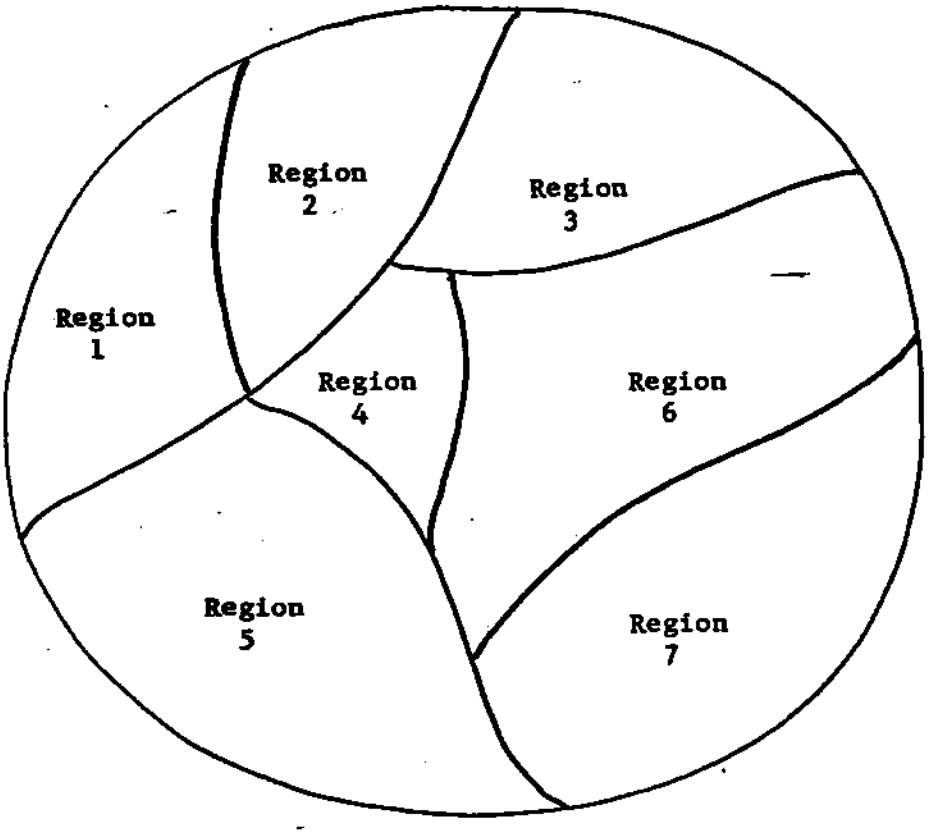

Figure 1. A Model Which Operates in Regions

The second approach is to guess which region is the correct one but ignore the constraints. If the solution resides outside the constraints, the place it falls can give help in reguessing which region to attempt next. This approach does not require an optimization problem formulation and is the one we are examining at present.

3.2. Dynamic Models

Kuru (1981) suggests writing a dynamic model in the form of a mixed set of ODE's and algebraic equations:

$$\dot{\underline{x}} = \underline{v}$$

$$\underline{f}(\underline{v},\underline{x},\underline{\pounds},\underline{u}) = \underline{0}$$

$$\underline{\pounds}(\underline{*}> \underline{\pounds}» H) = 2$$

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

$$\underline{x}(0) \quad \text{given}$$

where $\underline{x}$ are "state" variables equal in number to variables $\underline{v}$ and to equations $\underline{f}$, $\underline{v}$ are "velocity" variables, $\pounds$ are algebraic variables equal in number to equations $\pounds$ and $\underline{u}$ are independent variables whose time trajectories must be specified by other means before a unique solution is defined, $\pounds$ are the state equations, but here written in terms of variables $\underline{v}$ rather than as RHS's for the $\dot{\underline{x}}$ equations, and $\pounds$ are algebraic equations.

Defining $\dot{\underline{x}}=\underline{v}$ has the advantage that the numerical method which will be used to integrate the equations forward in time will have a simple predefined form of ODE to convert to the approximating algebraic equations defined by the method. For example a modified Euler's method will convert $\dot{\underline{x}}=\underline{v}$ into

$$\underline{x}_{k+1} = \underline{x}_k + \frac{\Delta t_k}{2} (\underline{v}_{k+1} + \underline{v}_k)$$

or

$$\underline{x}_{k+1} - \frac{\Delta t_k}{2} \underline{v}_{k+1} - \left(\underline{x}_k + \frac{\Delta t_k}{2} \underline{v}_k \right) = 0$$

which relates the unknown $\underline{x}$ and $\underline{v}$ values at time t_{k+1} to the known values at time t_k. Also the state equations $\underline{f}$ need not be algebraically rearranged so each f_i is in terms of only one $\dot{x}_i$ variable, often a tedious task.

3.3. Dynamics with Inequalities and General Functions

Inequalities can be added and any of the algebraic equations may be defined as general functions for a dynamic model, too.

3.4. Optimization

The only difference in an optimization model and one of the earlier ones is that a variable must be designated as the objective to be maximized or minimized. A point to be made here is that it should be possible to designate any variable. If so then problems are possible of the type: find the maximum recovery for this column if you are given 10% more money to operate it.

We will not discuss optimization further in this paper except to note that significant new results have occurred since Han (1975) provided the theoretical insights to allow the development of the very fast successive quadratic programming algorithm. Powell (1977) implemented the Han ideas, and we (Berna et al (1980), Locke and Westerberg (1982), Locke et al (1982)) extended them for use with large models.

3.5. Procedures

Perkins (1983) strongly advocates that an equation-oriented flowsheeting system must be able to use existing procedures within a model. A

"procedure" contains within it the equations and the code to solve them. One argument for allowing procedures is that some physical behavior could not be modeled in existing equation-oriented systems. We believe allowing general functions removes this argument. Another is that companies have many models developed which they will be unwilling to throw away, particularly for the calculation of physical properties. Finally the argument is made that some models defy normal solution methods and are best handled by a procedure that can incorporate all sorts of special ad hoc methods to help get to a solution for them. We agree with these last two arguments, particularly the former; the latter may become less valid as we learn how to set up and solve models containing general functions.

3.6. Model Hierarchies

Complex models are almost always built in a hierarchical fashion. An example is a distillation column, which is built up of trays, flash units, splitters, mixers, heat exchangers, pumps, etc. A flash unit itself is in fact a hierarchical structure. Embedded in it' is the need to evaluate K-values and enthalpies so in ccTncept it uses a library of models that already exist to evaluate the needed physical properties.

To create complex models one therefore needs the ability to combine previously written models with the extra defining equations for the complex model. It is here that "language" considerations become important. This ability can be included with "elegance", or it can be unfortunately clumsy. We used "macros" in ASCEND-II to include this ability. Our next generation language (ASCEND-III) has a very natural language construction, so natural that we do not even have to raise the concept of "macros" to the model builder.

Let's get at some fine points about building models in hierarchies. Consider the problem of a flowsheet which "contains" a flash unit. The flash unit "contains" the physical property routines to calculate K-values, enthalpies and so forth. At the flowsheet level we wish to include the flash model and to tell it which variation of the physical property routines we wish it to include. How does one do this "routing" task with a natural language construct? We used defaults in ASCEND-II to pass information down to the flash unit so it could then create a statement to be added to the flowsheet through macro expansion that invoked the desired physical property models.

In ASCEND-III we are including the language construct that will allow any name used in any lower level model to be treated as a formal (i.e. dummy) name that can be replaced with an actual name by the higher level. Thus we can write a flash model to invoke the "ideal" option for liquid activities, but, when including this flash model in a flowsheet, we can overwrite at the top level the "ideal option" of the lower level by the "NRTL" or "Wilson" option.

Along with this powerful renaming potential goes the possibility that one might, for example, rename a variable of type "temperature" to be a variable- of type "pressure". ASCEND-III uses strong variable typing so the writer of the flash model can insist that a variable be of type "temperature."

Another language issue is the notion of equivalencing two names so they refer to the same variable. In most languages this equivalencing must be done at the highest level the variables are mentioned. In ASCEND-III, we are allowing equivalencing to occur at the lowest level. Why? Consider

a pipe model with input stream SI and output stream S2. In steady state the model does not change component flows from input to output. If it can equivalence them, the equations that otherwise are needed to equate them and the storage space for having these flows written twice can be eliminated from the model. It is conceivable a very large fraction of the modeling equations and variables could be eliminated for some problems.

We return to make a point on variable typing. In ASCEND-III variables and records made up of collections of variables can be strongly typed. Thus we can collect together a set of variables and type them as a "stream,[11] The normal notion of record typing is to fix completely all attributes when defining a type — thus a stream containing H_2, CH_4 and C_2H_6 might be of type "hydrocarbon stream[11] and a stream containing H_2O of type "water stream.[M] The flash model would have difficulty when it typed its inputs and outputs as stream records — which stream type should they be, the water stream type or the other? In ASCEND-III we get around this by allowing typing to be qualified. Thus the flash can insist that its inputs and outputs are of type "stream"; the components involved can be set by separately qualifying the type "stream" to be the type "hydrocarbon stream," for example. The flash model only cares that what it gets is at least of type "stream" for its input and output streams.

3.7. Variable Names

The final point of importance is the naming of variables. It is desirable that a variable can be accessed by any set of attributes it has. For example, if a temperature is that of a tank as well as of its outlet stream, which of course is the inlet stream to another unit, we ought to be able to access it by any of these names. In ASCEND-III naming is done by constructing qualifiers; e.g. a variable can be named

REACT0R1. LIQUID-OUT. TEMP

If it **is** also

HX2. INI. TEMP

it **can** be accessed by this name too.

4. **Model** Solving

All our current thinking supports solving models using Newton or quasi-Newton methods combined with sparse matrix methods. This approach first converts nonlinear algebraic equations into their linearized equivalents* These linearized equations are then solved iteratively, with the linear equation solving taking advantage of their structure.

The main alternative to solving is by "tearing[11]. Tearing procedures strive to discover the fewest variables one has to guess such that, by algebraically manipulating the equations, the remaining variables can be **solved** for using a forward substitution scheme. There will be a leftover **set** of equations equal in number to the set of < tear variables. These are **evaluated** as error functions and, if not equal to zero, require one to **reguess** the tear variables. A sequential modular flowsheeting system uses **tearing** to solve - recycle problems. Tearing has significant problems which **are** difficult to get around for a general equation solving package. First **one** usually discovers the tear variables for a set of equations using only **the** structure of the equations. It is very easy to create singular **solution** procedures using structure only. Second the approach suggests one **will** manipulate the equations to create the forward elimination scheme for **the** nontear variables. To date such algebraic manipulation is too costly to be done repeatedly. It can take seconds of time to rearrange an

equation* Third, the sensitivity information developed for Newton-based schemes is valuable for implementing optimization.

Therefore we shall confine our discussion here to Newton-based solving* The approach involves the following considerations.

4.1. Partitioning and Precedence Ordering (see Westerberg et al (1979)).

Based on the structure of the resulting problem it is often possible to partition the problem into a sequence of smaller irreducible ones which can be solved in a precedence order. Each irreducible problem requires the simultaneous solving of equations. For solving m equations in m unknowns (a square system), partitioning and precedence ordering is well understood, with an algorichm by Tarjan (1972), which is very similar to one by Sargent and Westerberg (1964), being extremely effective. Nousquare systems of ra equations in n+ro variables are not so readily dealt wich as here one can significantly affect the partitioning possible by choice of which 'n' variables to use as the independent ones. "Safe* partitioning is possible if *'a'* fictitious equations are added before partitioning where each is assumed to contain all variables. After partitioning these equations are dropped. Another alternative is to select the independenc variables by other means and then cu partition the resulting square sysucnu

If partitioning is done then one can solve the variable initialization problem, convergence problems, etc., as a sequence of smaller problems. Often initialization, for example, is easier to do well if values for variables feeding into a current partition from an earlier one are already calculated.

We find ourselves with a dilemma here using this argument to justify the use of partitioning. A good scheme to partition is to generate the

linearized equations and to perform an "analyze" pass for Gaussian elimination where a set of pivots are picked based on structural and numerical considerations. The $^{f}n^{*}$ variables not pivoted become the independent ones. We then partition and precedence order the leftover square set of 'm' equations and variables. But we needed initial guesses to generate the linearized equations. Full circle. Clearly we need to "tear" this circle and start somewhere. One could envision two (or more) passes. First one would crudely guess the variables and then perform an analyze pass to choose a set of 'n' independent variables. Then one could partition and precedence order the $^{v}m^{f}$ equations and 'm^{1} dependent variables. Then one could start over on initializing variables. Comparison studies are needed to decide if this two pass strategy is worthwhile for most problems.

4.2. Large Problems

To solve really large problems one can consider ways to decompose the irreducible computations or to solve them using other than Newton-like methods.

Problem decomposition using Newton-like schemes has been discussed by Westerberg and Berna (1978) and by Stadtherr and Hilton (1982). The idea is to decompose the linear equations corresponding to a flowsheet problem by taking advantage of their bordered block diagonal structure. The decomposition allows one to work on the equations unit by unit so the fast memory requirements of the computer need only be large enough for a unit's worth of equations. The penalty one pays is the substantial amount of data transfers to and from mass memory.

A sparse matrix package has been written by Clark (1980) based on these decomposition ideas. It is really very interesting in its capabilities, which were motivated by Kuru's (1981) thesis work on dynamic

simulation. It permits one to generate the Jacobian and RHS error functions for each unit separately and pass them to the sparse matrix package. Based on a preanalysis first pass, this package knows the extent to which it can reduce the equations for the unit by partial forward elimination, saving the bulk of the modified Jacobian matrix on fast memory and keeping the part that connects to other units available for subsequent reduction at the far end of the fast memory available. When fast memory fills to the extent permitted, the combined residual blocks from previously analyzed units are themselves reduced as if a unit. The back substitution~%Eep retrieves the modified Jacobians in the reverse order and carries out back substitution on a unit by unit basis.

If the variables for a unit have not changed since the last step, the Jacobian and RHS error generation can be suppressed. If needed for backward substitution the package will use the results of the last pass. If the RHS errors are essentially zero for a unit, the forward elimination step is not required. Again it can be skipped. A similar skipping for unnecessary back substitution steps is possible.

Finally, if two units are in the flowsheet in a way that would permit them to use the same pivot sequence, one can suppress the costly "analyze" step of a sparse matrix package and use the pivot sequence of one unit on another. This feature is particularly useful in dynamic simulation.

Kuru suppressed 45% of the computations needed for a very simple dynamic model using these capabilities. The "latent" portions of computations for the process were simply turned off when portions of the process ceased to be very active.

4.3. Variable Initialization

To solve a set of simultaneous nonlinear algebraic equations using a Newton based scheme requires that one has an initial guess for every variable. The user will be willing to provide guesses for only a few of the variables, and the system must be developed to provide the rest.

We believe that this step must be done with extreme care, as doing it poorly can often lead to no chance to solve the problem even though a solution exists. Newton schemes will usually converge rapidly if they will converge. Using Newton-Raphson we frequently solve problems in 4 to 6 iterations. The theory is very clear — one must get in the vicinity of the answer to get such convergence rates. Failing to do so will lead to slow or no convergence, with no convergence being useless to the user.

How can initialization be accomplished? We can imagine several schemes.

First one can use defaulting. The user can be asked for any guesses he is willing to provide. The system then proceeds to use default values for all the rest. Defaulting can be done by setting all variables to zero, or it can be done with some problem insight if the "type" of each variable is known. For example, if a variable is known to be a temperature, it could be defaulted to 300K. Or a mole fraction could be defaulted to $1/n_c$, where n_c is the number of components in the stream. To default with this level of insight requires that the system has considerable information available — in the case of mole fraction that the variable is a mole fraction and that it is part of a stream containing n_c components.

A second approach is to use "tearing" concepts to analyze a problem to discover a minimal set of variables, which if guessed, will allow the

rest to be determined in terms of these. Here tearing is only used to obtain an initial guess. We use such an approach in ASCEND-II. However, the tearing is not done automatically, nor did we strive to provide minimum tear sets. Rather, each unit to be put into the library was analyzed by the library creator to find a set of tear variables which if known would allow the remaining variables to be initialized. The input stream variables to a unit were assumed to be known or to have been guessed by the user on entry to initialization. These together with a few "tear" variables permitted initialization of the remaining variables for the unit. The tear variable names are known and printed in the user manual for each unit.

By sequencing through a flowsheet much as for solving in a sequential modular fashion, all the units can be initialized by guessing only recycle stream values and the tear variables for each unit. Locke (1982) describes the approach in more detail. Kuru and Westerberg (1983) describe similar ideas to predict algebraic variable values as one is stepping forward in time when solving initial value dynamic simulations.

There is another approach to initialization that is very appealing. It is to "evolve" from simple models up to the final complex one that one wishes to solve. For example, one could first use approximate models that consider only material balance and equilibrium equations, the latter with constant relative volatilities, etc. A system composed of these simple or partial models could be solved first. Based on these numbers, more complexity can be added — for example, the heat balances, the new variables initialized and the system solved again. Note that here the entire complex flowsheet is solved at each step but with a sequence of more complete models.

It is our experience that the simple models will converge from almost any first guess so one can readily get started with simple defaults and perhaps the unit-by-unit initialization of ASCEND-II. Then based on these results, more complexity can be added to the models and unit-by-unit initialization repeated to initialize the added variables just introduced. Does this sound like the special tricks used by many distillation column programs?

The use of general function modeling permits one to develop models that can evolve from simple to complex by the changing of a single discrete variable from values of "simple" to "intermediate[11] to "complete." This capability we feel will have major impact on solving difficult problems.

4.4. Variable and Equation Scaling

Scaling is a "black art." Experience is poor with schemes that rescale variables and equations and do not consider the nature of the problem. The Newton-Raphson method predicts a step which is theoretically scale invariant; it is said to be a scale invariant method. It is not in practice, however, as scaling is necessary to allow a stable pivot sequence to be selected for a finite word length machine.

Quasi-Newton methods are frequently not theoretically scale-invariant* Perkins (1983) discusses this point. He presents an appealing case for developing scale invariant Broyden-like methods and discovers on testing them that theoretical scaling invariance has not helped particularly.

Still, scaling itself is necessary as it is needed to prevent excessive roundoff errors when pivoting. How can one do it if completely blind scaling seems to be poor?

In ASCEND-II we scale variables by knowing the variable type and scale using nominal values* We divide temperature perturbations by 300K, mole fractions by unity and so forth. Flowrate scaling is based on current values.

To scale equations we develop the terms in the nonlinear equation (based on current variable values) which are added together to form the equation and scale by the largest of these in magnitude or unity, whichever is larger.

In ASCEND-II, variable and equation rescaling can be requested at any time. We have had problems not converge until they were rescaled, indicating it was not that the problems did not converge, but, because of poor scaling, we could not detect convergence.

Two implications occur here for the flowsheeting system. Again it must be aware of variable types, and we find it very useful to solve problems interactively, at least the first few times, to see if rescaling can improve the solution process.

4.5. Converging

The real problem here is to decide what to do if one fails to converge. From a numerical point of view, one can investigate such topics as using bounded Newton steps, or steps such as those predicted by a Marquardt-Levenberg algorithm, or use a continuation method. Seader (1983) has just presented an excellent paper which uses continuation methods to converge complex distillation structures. Perkins (1983) also discusses attempting to create more stable and convergent quasi-Newton methods, by, for example, making them (theoretically) independent of scaling. Many people have suggested starting up dynamic models to find the desired

steady state solutions. This is one aspect of flowsheeting systems well covered by the literature, and one we will not spend time on here. The continuation methods look exciting if they can be developed to run faster. At the present time they seem to take an order of magnitude too long.

We will look at another aspect to converging. It is quite different from the above and involves, again, sneaking up on the problem by evolving from less complex calculations to more difficult ones. The idea we bring up here which is different from that in initialization is to change the set of variables used as the independent ones for a problem. We will illustrate with an example.

Suppose we wish to solve a distillation column model. We give it a fixed number of trays, top and bottom, and want to solve it to recover 90%, of the light key in the top product and 95% the heavy key in the bottom product. In ASCEND-II, the approach we would take is first to solve the column in a manner we know intuitively has a solution. We would fix the feed, top product flow and reflux rate, the last at a somewhat higher than necessary value. We almost certainly will find a solution (or we will "evolve" to it through a sequence of less complex models as discussed under initialization). We would then trade the top product flow and reflux specifications for the two recovery specifications. This trade means we will change the sets of variables identified as dependent and independent.

Suppose we leave the variable values, even for the new independent variables, at the values found above. We ask the system to attempt to solve. It will spot a singularity in the Jacobian matrix for the new set of dependent variables if the trade was bad among the dependent and independent variable sets. We can change what we specify until we obtain a

nonsingular problem. We can then attempt to impose the 907. and 957» recovery specifications. If the problem fails to converge, it can be because we are too far from the answer initially or because the specifications are beyond the ability of the column to meet them. It is not because we chose a bad set of variables to use as independent variables.

Failing to converge we would, in ASCEND-II, return to our original simulation and play with the reflux rate and top product flow to see how far we can push the recoveries. If we cannot get close to those desired, we would almost certainly see the impossibility of our original problem. Note we are using the "power[11] of ASCEND-II to allow us to alter the sets of dependent and independent variables quickly to discover why we cannot converge.

This altering of sets is a matter of changing the flags associated with variables that indicate if the variable is to be calculated or fixed in the subsequent calculation. It costs us only a "reanalyze" step in the solving of linear equations (about 1 second for several hundred equations on a VAX-780).

4.6. Procedures

We can propose three approaches to solving models containing procedures. Let's assume our procedure calculates £ given x and that it is equivalent to

$$Z - £(*>$$

Including this procedure is the same as including the equations

$$KZ»*> - Z " E(2S) " 9.$$

into our model* The Jacobian elements for a Newton-Raphson scheme become

$$\frac{\partial}{\partial} = \quad \cdot \quad \frac{\partial}{\partial}$$

the latter we must estimate numerically, either using perturbations or quasi-Newton update methods.

An alternative for using procedures is to develop an approximate nonlinear model that reproduces the local behavior of the procedure adequately in the vicinity of the current solution. For example, one might approximate a complex thermodynamic model for K-values by

$$K_i = \exp(A_i/T + B_i)/P \quad .$$

The procedure is then used to obtain K_i values vs T and P at the current compositions of the vapor and liquid streams which are in equilibrium with each other, to allow A_i and B_i to be approximated. Lucia and Westman (1983) points out that this last approach can cause one to lose the superlinear convergence characteristics of a quasi-Newton method.

5. Conclusion

We have used this paper to discuss many aspects we view to be important in equation solving approaches to process flowsheeting. The presentation is not meant to be complete or even near complete.

We believe some new ideas on language design for such a system and the use of "general functions[11] in models are real contributions to developing these systems in the future.

6. References

Benjamin, D.R., Locke, M.H., Westerberg, A.W., "Interactive Programs for Process Design," Report 06-28-81 paper 25b, DRC, 1981, presented as paper 25b, AlChE Summer National Meeting, Detroit.

Berna, T.J., Locke, M.H., Westerberg, A.W., "A New Approach to the Optimization of Chemical Processes," AIChE J., Vol. 26, No. 1, 1980, pp. 37-43.

Cherry, D.H., Data Organization in Chemical Plant Design, Ph.D. Thesis, Univ. of Cambridge, Cambridge, England (1975).

Clark, P.A., An Implementation of a Decomposition Scheme Suitable for Process Design Calculations, M.S. Thesis, Carnegie-Mellon Univ., Pittsburgh, PA (1980).

Han, S*P., A Globally Convergent Method for Nonlinear Programming, Dept. of Computer Science, Cornell Univ., Report No. 75-257 (1975).

Kuru, S., Dynamic Simulation with an Equation Based Flowsheeting System, Ph.D. Thesis, Carnegie-Mellon Univ., Pittsburgh, PA (1981).

Kuru, S», Vesterberg, A.W., "A Newton-Raphson Based Strategy for Exploiting Latency in Dynamic Simulation," submitted, 1983.

Locke, M.H., A CAD Tool Which Accommodates an Evolutionary Strategy in Engineering Design Calculations, Ph.D. Thesis, Carnegie-Mellon Univ., Pittsburgh, PA (1981).

Locke, M.H., Edahl, R.H., Westerberg, A.W.., "An Improved Successive Quadratic Programming Optimization Algorithm for Engineering Design Problems," Tech. report, Design Research Center, Carnegie-Mellon Univ., Pittsburgh, PA, 1982. Accepted for publication, AIChE J.

Locke, M.K., Westerberg, A.W., "The ASCEND-II System - A Flow-sheeting Application of a Successive Quadratic Programming Meth-, odology," Tech. report. Presented Annual AIChE Meeting, Los Angeles, CA, 1982. Accepted for publication, Computers and Chem. Engng. J.

Lucia, A. and K.E. Westman, Low Cost Solutions to Multistage, Multicomponent Separation Problems by a Hybrid Fixed Point Algo-rithm, Contributed paper, FOCAPD-83, Snowmass, CO (1983).

Perkins, J.D., Equation Oriented Flowsheeting, Keynote Lecture, FOCAPD-83, Snowmass, CO, (1983).

Powell, M.J.D., A Fast Algorithm for Nonlinearly Constrained Optimi-zation Calculations, presented at 1977 Dundee Conference on Numerical Analysis (1977).