August, 1994
Eqn size parameters: 10 8 5 14 10

Equational-based process modeling

by
Arthur W. Westerberg
Peter C. Piela[1]
Department of Chemical Engineering
and the Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

These notes discuss two different approaches to process modeling, the sequential modular approach and the equational-based modeling approach, with an emphasis on the latter. We explore in detail the advantages and difficulties with equational-based modeling and show that using good modeling practice eases many of the difficulties for creating, debugging and solving complex chemical process models.

Introduction

In these notes, we are going to discuss process modeling. Two distinctly different approaches exist which chemical engineers use for the modeling of processes: the "sequential modular approach" underlying most of the commercially available modeling systems and the "equation-based approach." The equation-solving approach has a long history in research and only some commercial success in the last few years. Each approach has its advantages and disadvantages. These notes are intended to show how one can infuse the equation-solving approach with many of the advantages of the sequential modular approach, thus making it a much more competitive approach than it has been in the past.

We shall start by examining the sequential modular approach. For this approach, we shall examine the two main issues that occupied the earlier literature during its coming of age: how to select the order in which to solve the unit operation models used to define a flowsheet and how to converge the overall process model when it contains recycles streams.

The equation-based approach to modeling follows. There are two reasons to explore it in detail. First, the creators of a sequential modular modeling system have to develop the models for each of the units in that environment. These creators must approach this problem by developing the equations for each unit and then develop a robust approach for solving them; i.e., they must be

[1] Currently at AspenTech, Cambridge, MA

users of equation-based approaches for this subproblem. Secondly, the equation-based approach is itself a powerful way to solve a complete process model.

The emphasis for the equation-based approach will be on strongly separating the statement of the model from the debugging and solving of it. Thus we consider first how to state what is to be true at the solution (i.e., the equations for the model) and what can be said "declaratively" about solving it and secondly, and quite separately, the activities in finding a solution.

An equational-based model is a quantitative one described by algebraic and possibly ordinary and partial differential equations. We use equational-based models to determine the quantitative behavior of artifacts. For example, for a really simple problem, we may wish to determine the amount of metal required in constructing a cylindrical shaped tank given its diameter and height. The equations, based on simple geometry, state the needed relationships in terms of suitable variables. Solving would then involve giving values for some of the variables and computing the rest of them. For a more difficult problem, we may wish to determine the performance of a flash unit, and, for an even more difficult example, we may wish to compute the motion of a robot arm with time.

Models can become quite large and often very complex in their structure. None of us would be surprised to discover that correctly formulating and solving a large model is difficult. What we might not appreciate is that even small models can confound us with their subtle complexities. These notes are not a recipe for modeling. Rather they describe many of the difficulties we have found when modeling chemical processes. We shall pass along those ideas we believe can make such modeling less difficult. We shall also discuss issues involved in debugging and solving these models.

Debugging is inherently an interactive process. One typically attempts to solve a model and discovers the attempt fails. At this point a modeler must become a detective to find what is causing failure. Detectives look everywhere for clues and often have insights into the problem while walking home after work or at 2:00 am when trying to sleep. We shall discuss the types of tools that can aid in discovering what went wrong. Finally, it is important that failure is not the fault of the numerical methods for solving them so these must be as solid as possible. We shall discuss ideas related to good solution procedures.

The general outline for these notes will be first to discuss sequential modular modeling. When we switch to equation-based approaches, we shall discuss issues relating to stating a model and then to solving one assuming the model is well-posed. With these issues understood, we shall talk about the debugging of these models.

Sequential modular modeling - flowsheeting

Description

Sequential modular modeling underlies most of the flowsheet simulation programs developed since Kellogg announced their flexible flowsheeting program in 1958. The approach they and almost everyone following took was for skilled modelers to develop Fortran subroutines to model each of the

various types of unit operations that we use to construct complete processes. There are subroutines for the flash unit, distillation columns, absorbers, a variety of reactor types, compressors, pumps, valves, and so forth. One constructs a complete process model by *wiring* up an appropriate set of these building blocks. The flowsheeting system then solves the total process model by calling each of the unit models in turn, according to how they are wired together, iterating where necessary to coverge complex process models.

The clients for sequential modular flowsheeting systems are all those people who wish to develop the heat and material balances for chemical processes. These include process engineers, sales persons (who may have minimal technical training), PhD researchers, and the like. The systems must work and must warn of failure when they do not. Ideally they can suggest why they fail when they do in terms the user is likely to understand.

The main assumption for each unit subroutine in such systems is that its input streams are fixed and that it will compute the flows, temperature and pressure for each of the streams leaving the unit. Virtually all unit models, except for a stream mixer, require one to specify other parameters to fix their operation. For example, a conventional flash unit requires one to specify two other things about it, such as the temperature and pressure at which it operates or the heat added/ removed and the pressure, in order for it to be a well-posed model. We call the former an isothermal flash computation while the latter is a variant of an adiabatic flash computation.

The developers of these unit models include all sorts of special tricks in them to make these computations robust. Of highest priority is that a unit model will converge when there is a solution for its underlying equations and that it fail reasonably when there is not. These tricks include developing initial guesses from which the equations typically converge. There are detectors in many of these codes to discover lack of convergence and then tests to decide what to try next to gain convergence. Each is often like a mini-expert system, containing every bit of knowledge a modeler knows and learns about such a model to make it work. Most of the code in such routines is to assure this robustness.

Example

The easiet way to understand this approach is to work our way through a simple example process [Westerberg, et al, p131-8, 1978]. We shall be describing a hypothetical flowsheeting system.

Fig. 1 is a simple flowsheet for the conversion of propylene to 2-hexene by the reaction

$$2 \text{ propylene } (CH_3CH{=}CH_2) \rightarrow \text{2-hexene } (CH_2{=}CH(CH_2)_3CH_3)$$

The propylene feed is at 150 ^{o}C and 20 bar. It contains 2 mole% propane. Our goal is to investigate the performance of this process. In particular we would like to understand if the flash unit is a good enough separation device or if we should replace it with a distillation column.

Our flowsheeting system allows us to build the model interactively using a computer workstation on which we can place icons for each type of unit and "wire" these icons together with streams.

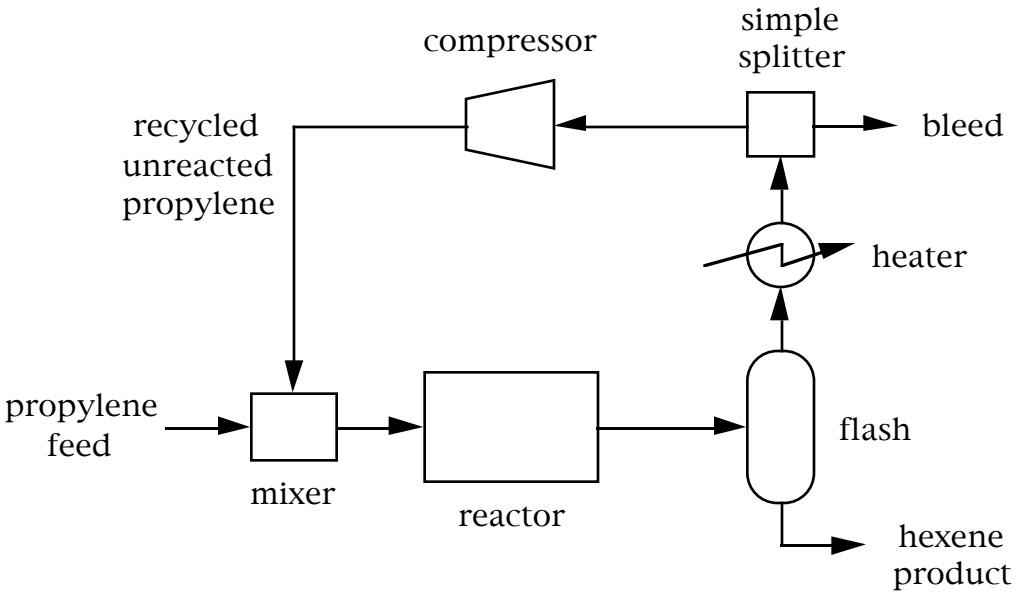


Fig. 1. Simple flowsheeting example

We use the graphical user interface to pull icons for each type of unit from a menu onto our workspace, placing them roughly as shown above. We ask the system to draw each stream connecting two units by clicking on the output node of one unit and the input node of another. In a matter of few minutes we have the above drawing on the computer screen. From this drawing, the flowsheeting system knows the existence of all our streams and units and how they are interconnected.

Our flowsheeting system next asks us the components we wish to use in our simulation. Different streams can have different species in them so we pick a stream and pick the species for it from a list the system offers to us. If a unit insists on the same species in two or more of its associated streams, it will propagate what we pick to these streams. For example, the compressor will insist that the species that enter must be exactly those that leave. The mixer can have different species in its input streams. The species for the output stream, on the other hand, must be all the species entering.

We must also pick the physical property methods we want the system to use. In our system, these options can vary from unit to unit. For a start we pick ideal for all units. The system will use ideal mixing models for vapor and liquid mixture Gibbs free energies, enthalpies and volumes (the information it will need to compute vapor/liquid equilibrium in the flash, effect of pressure changes for the compressor, and heat balances throughout).

The system analyzes our flowsheet and tells us the flowsheet has a recycle in it. It tells us we will have to guess conditions for one recycle stream before we start our computations. It lists the candidate streams from which we select the one we wish to guess: the compressor output, the reactor feed, the flash feed,

the simple splitter feed or the compressor feed. We pick the compressor output which we will later be asked guess.

Next we will have to select the unit model to use for each unit shown on our flowsheet. The sequential modular flowsheeting system will have in it one or more models for each of the types of unit it supports. If there are two or more models for a type of unit, they will typically range from simple to complex in their implementation. For example, the flash unit may have three models one could use for it. The first will be a simple component splitter where the we tell the model that 98% of the propylene and propane entering are to exit in the top vapor stream and 99% of the hexene in the bottom liquid stream. The second might be a constant relative volatility flash unit. For this type of flash unit we can either specify the relative volatilities for the species or ask the system to estimate them using Raoult's Law at a given temperature and pressure. Finally there will always be a rigorous flash model which uses the physical property library to compute nonideal equilibrium K-values and nonideal mixture enthalpies.

We select a unit, say the flash. The system gives us a menu of the three models we can use for it. We pick the rigorous flash model. The system then asks that we specify two added parameters out of a list of possibilities; we choose to specify the fraction of the incoming feed that will leave in the vapor stream (e.g., 50 mole%) and the pressure.

For the reactor unit, we pick a model that allows us to specify that 80% of the propylene entering will convert as it passes through this unit.

We repeat this activity for all remaining units.

The system now asks us for the needed input to fix the computation and for it to establish starting guesses. It first asks for the feed stream specifications; we tell it the flowrate, composition, temperature, pressure and that we think it is a liquid, which the system verifies. It asks for guesses for the flowrates, composition, temperature, pressure and phase (it had better be vapor) for the compressor output. Next it goes from unit to unit to ask for the operating parameters it needs to complete each unit model specification. It also asks for guesses of some of the variables for which it believes the user guesses will aid it to converge the total flowsheet model.

The flowsheeting system now solves the model. It does this by solving the mixer first as it knows the feed stream and has a guess for the recycle stream entering it. From that computation it know the reactor feed. It solves the reactor, then the flash, the heat exchanger, the simple splitter and finally the compressor. It compares the compressor output to that guessed. If these do not agree, it reguesses the recycle and repeats the solving of the units in the sequence done above. It tells us IN LARGE PRINT it was successful in converging the flowsheet in 23 interations. We interactively go from stream to stream and unit to unit to see the values associated with each. We find that the numbers produced look plausible as we investigate them. We ask for a print out of the total flowsheet.

We are now ready to play with our model. We can choose to do many different things with it. We first change from ideal to nonideal models for evaluating physical properties, choosing the methods an associated expert system

suggests to us. The expert system examines the species involved, asks us our intentions (do we want speed or do we want accuracy, for example), and suggest the options.

We resolve successfully and the numbers still look good. Now we wonder the effect of changing several of the parameters on the performance of the process. For example, we might wonder what is the impact of altering the fraction of the feed to the simple splitter which exits in the bleed stream. We set up a series of computations to be carried out one after the other where this parameter varies from a tenth of a percent to 20 percent. After running these cases, we ask the system to plot several of the variable values vs. the fraction we bleed from the process.

We decide the flash unit is not giving us a pure enough hexene product. We replace the flash unit with a small distillation column and start our simulations over again. Our first model for the column is a shortcut one capable of estimating the number of trays needed and the reflux flows needed for ideal behavior. After we switch to a rigorous column model, we need to play with the number of trays and feed tray location as well as the reflux ratio to get it to perform well. We find this to be a tedious exercise. We do not replace the bleed stream as we know separating propylene and propane is very difficult.

We next add computations for each of the units which estimate their capital investment and operating expenses. When we also add the value of the feed and product streams and things like the background interest rate for our company, we are able to compute the present worth of the process for each of the alternative ways we choose to run it. Now we are ready to turn on an optimizer that we find is available with the flowsheeting system.

We first ask for the plant requiring the minimum investment. After playing for a while, we solve and find the model is trying to converge to a plant with zero flows. That reminds us we forgot to place a production constraint on the process which we then add.

We will stop our example at this point as it has served its purpose.

Computational controllers

Often in a simulation we wish for the model to produce a value for a variable that we find is computed by the flowsheet model. For example, we may wish to solve our flowsheet to give us a hexene product which is 99% hexene. We decide to adjust the bleed fraction to attain this purity. It is not clear to us if we can adjust the bleed fraction enough to affect purtity this much, but we intend to try. One way is, of course, to set up a series of computations where we adjust the bleed fraction until we get the desired purity. We can also ask the flowsheeting system to carry out this search for us automatically.

We ask for a computation controller to be added to the model. We connect its *measured variable* to the hexene product stream and, in particular, to the mole fraction of hexene in it. We connect the variable it is to *manipulate* to the unit parameter we previously fixed that tells the simple splitter the fraction of its input stream it is to give to the bleed stream. We are asked for the target value

for the hexene mole fraction (0.99), an initial guess for the bleed fraction, the maximum step size to use to adjust it and maximum and minimum values to allow for the bleed fraction when searching.

When solving the model, the flowsheeting system automatically adjusts the bleed fraction while monitoring the hexene mole fraction in the flash liquid product stream. The computational controller uses a special search algorithm based on the secant method to reguess the bleed fraction as it attempts to converge the model to the value we desire. In this case it fails to get the mole fraction where we ask it to be.

Flowsheet ordering

There are three issues we will investigate in this section: partitioning, precedence ordering and tearing. We shall define these concepts by applying them to an example flowsheet found in the literature [Leesley, p624, 1982]. See Fig. 2.

Partitioning and precedence ordering

We would like to solve the flowsheet in Fig. 2 in the most efficient manner possible. If we look closely at it, we see that we can compute units A, B, C, D and E are in a recycle loop and will certainly have to computed together. A still closer look and we see we must add units F and G to the group. It appears that this group of six units can be solved first as we see no streams recycling from later units back to any of these units. To complete our analysis will take some careful examination. The units that we have to solve as a group are called partitions and finding these groups *partitioning*, while the order we must solve them is a *precedence ordering*. The grouping is unique; the ordering may or may not be, depending on the particular flowsheet we are examining.

This example is simple enough that we would have little trouble seeing the partitions and the ordering for them. Some flowsheets have tens to hundreds of units in them, and it is difficult to find the partitions in them and the ordering for those partitions. Fortunately a simple algorithm exists to find the partitions and a precedence ordering [Sargent and Westerberg, 1964]. It proceeds as follows for this example.

Start with any unit, unit I for example. Put it on a list we shall call list 1.

List 1: I

Extend list 1 by tracing output streams starting with the end object on the list 1. We continue until we find a unit repeating or until there is no output to trace. We find the following trace.

List 1: IJKLMNL

We discovered this sequence by noting that I has an output to J which has an output to K which has an output to L which has an output to M which has an output to L which has an output back to L. Unit L repeats. There is a loop that traces from L to M to N to L. These units must be in a group. Merge them into a group and treat them as a single entry on list 1.

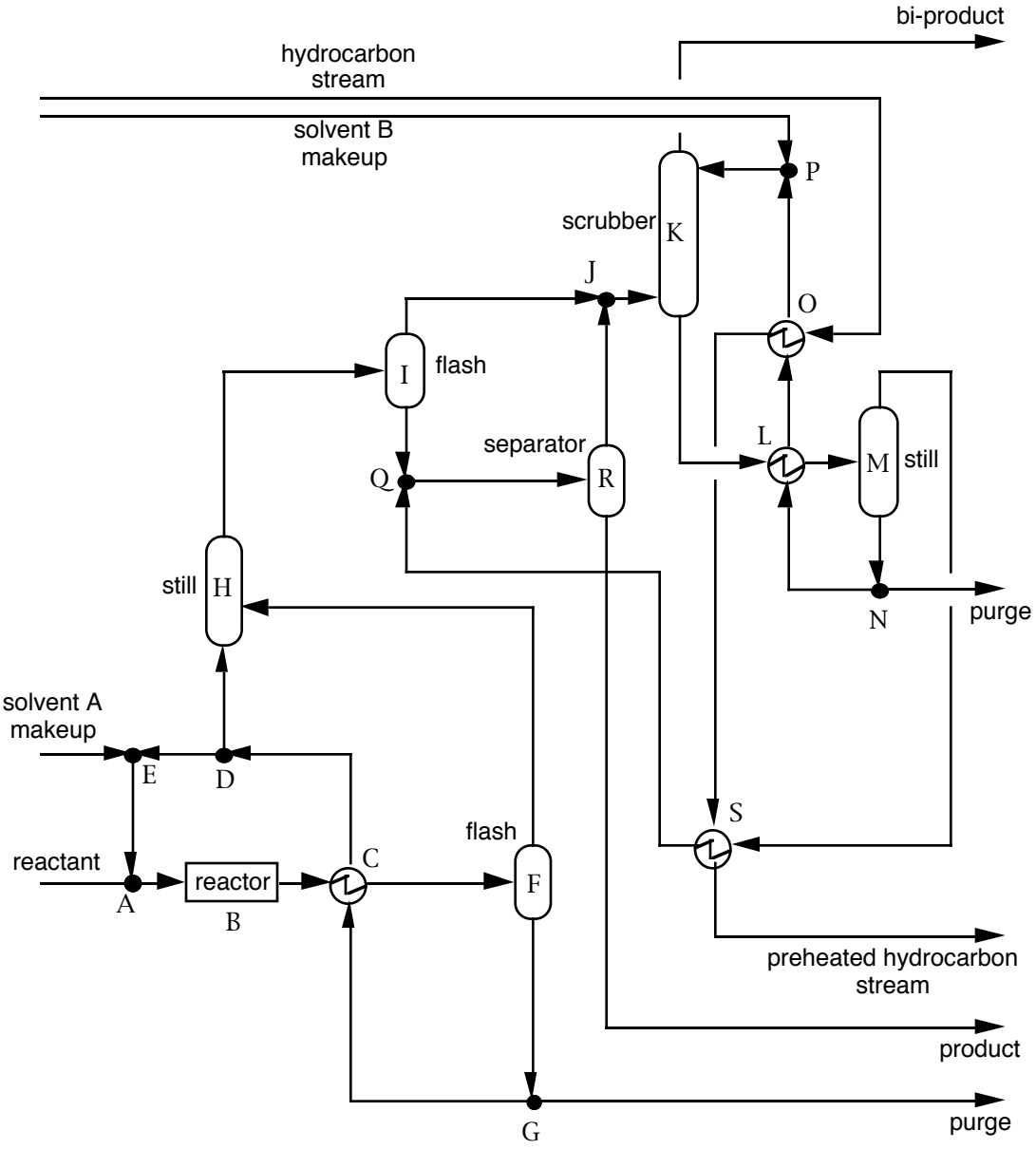


Fig. 2. Example flowsheet for partitioning and precedence ordering

List 1: IJK{LMN}

Continue tracing.

List 1: IJK{LMN}OPK

Unit K repeats. There is a loop from the group K to group {LMN} to O to P to K. Group the units in this loop, getting

List 1: IJ{KLMNOP}

Continue tracing outputs

List 1: IJ{KLMNOP}SQRJ

Unit J repeats, giving

List 1: I{JKLMNOPSQR}

When we try to continue tracing outputs, we discover that the units in the last group have no streams leaving from them to other units in the flowsheet. We remove this group from list 1 and place it on list 2.

List 2: {JKLMNOPSQR}

We cross off all these units from the flowsheet. We are done analyzing them. Returning to list 1

List 1: I

we look for more outputs from unit I. None exist that do not go to units removed from the flowsheet already. We remove unit I from list 1, place it at the head of list 2, and cross it off the flowsheet.

List 2: I{JKLMNOPSQR}
List 1:

List 1 is empty. Pick any unit in the flowsheet and place it onto list 1, say unit F.

List 1: F

Tracing outputs we get

List 1: FH

where we stop as H has no outputs except to units we already crossed out (and put onto list 2). Remove H from list 1 and place it as the head of list 2.

List 2: HI{JKLMNOPSQR}
List 1: F

Start tracing from F again, getting

List 1: FGCDEABC

Unit C repeats. Group it with the units between it two occurrences.

List 1: FG{CDEAB}

Continue tracing

List 1: FG{CDEAB}F

Group F and G with the other units

List 1: {FGCDEAB}

We find there are no other outputs to trace. Remove this last group from List 1, place it at the head of list 2, and remove these units from the flowsheet.

List 2: {FGCDEAB}HI{JKLMNOPSQR}
List 1:

There are no more units to place on list 1 so we are done. List 2 is our list of partitions in a precedence ordering. We can first solve the partition {FGCDEAB}, then unit H, then unit I and finally the remaining partition {JKLMNOPSQR}.

This algorithm works no matter which unit we start with on list 1. It gives a unique set of partitions - i.e., the units grouped together. However, the precedence order among the partitions may not be unique. Here it is, however. An example of nonuniqueness is, if there were two units, T and U each feeding reactor B from the outside, we could compute T first and then U or the reverse.

Tearing

The next issue is how we might solve each of the partitions containing more than a single unit. We have two such partitions in our last problem. We can illustrate an approach by examining the larger partition. The first is very simple, and we leave it as a homework problem for the reader.

We repeat this part of the flowsheet here in Fig. 3. We see a number of units in this part of the flowsheet for which a single stream enters and a single stream leaves. We can remove these units, Fig. 4, as they add nothing to the topology of the underlying network. Finally we straighten out the lines and redraw it as Fig. 5, where we label the streams.

Comparing Figs. 4 and 5, we see that if we were to choose to tear stream 8 (the connection between units S and K) we could tear any one of the actual streams along the path between those two units.

We shall look at two different algorithms to find a set of streams we can tear. Barkley and Motard (1973) proposed the first, Upadyhe and Grens the second. The first is very simple and looks for the fewest tear streams. The second looks for a set of tear streams which have better convergence characteristics.

Barkley and Motard Algorithm

In this algorithm we first convert the flowsheet into one the places the streams on the nodes and the units on the edges, what we can call a dual

representation; see Fig. 6. What we shall now look for are stream loops, e.g., streams 1, 2, 7 and 8 are in a loop. One of them has to be torn to break this loop. Our goal is to find the fewest we need to tear to break all stream loops.

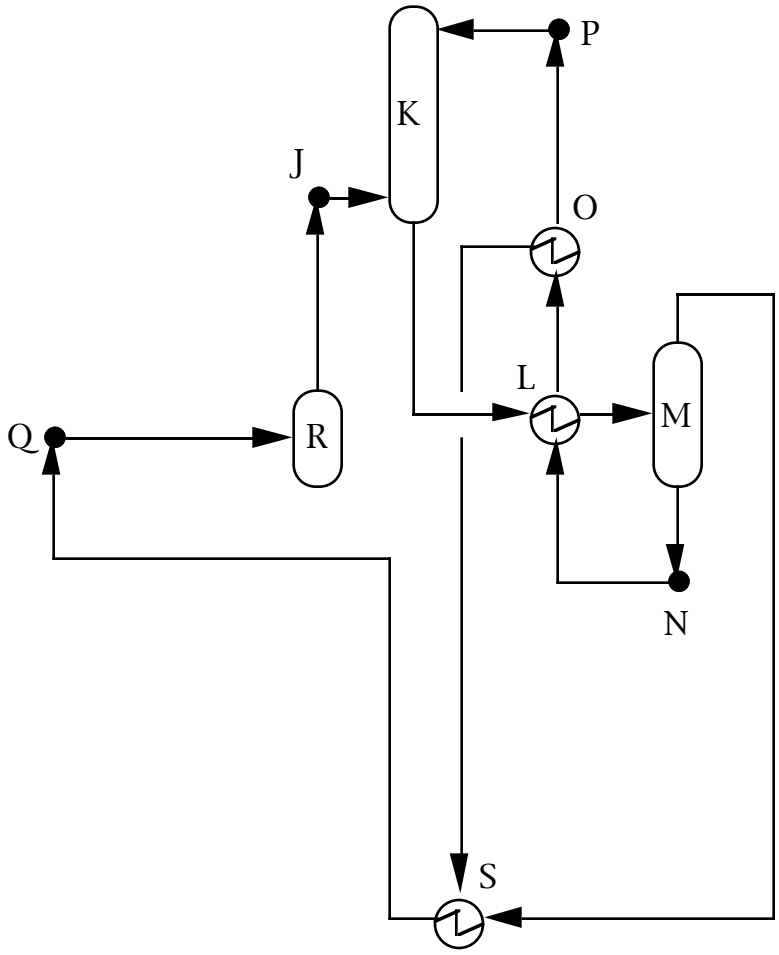


Fig. 3. The second partition for the flowsheet in Fig. 2. All streams and units outside this partition are removed.

We proceed by listing each stream and its inputs. If a stream has only one input stream feeding it, we remove it from the network and replace it by its single input, leading to columns 3 and 4 in the Table 1 and (a) in Fig. 7. All stream loops containing a stream with a single input must be torn and can be torn by tearing that single input stream.

We can place any stream which now feeds itself onto the list of tear streams as it is in a *self loop*. We find stream 2 in this category. Remove it from the problem, getting columns 5 and 6 in the above table.

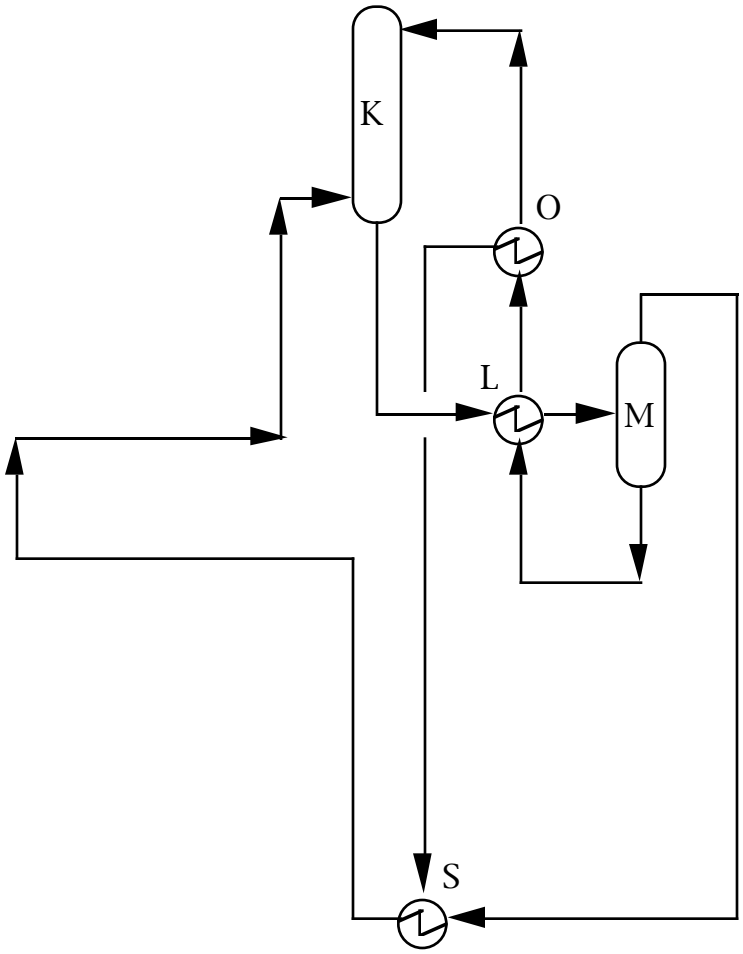


Fig. 4. A reduction of the second partition shown in Fig. 3. This reduction is formed by removing all units which have a single input/single output stream.

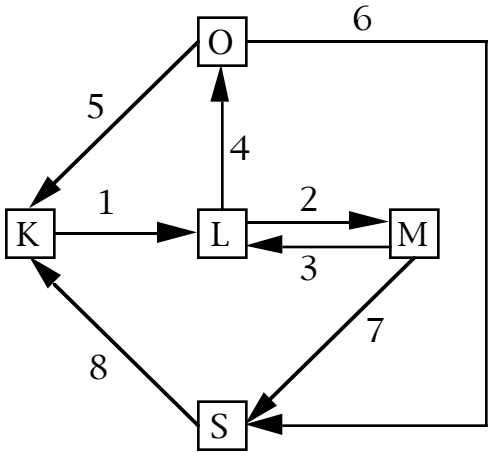


Fig. 5. The underlying topology for the partition in Fig. 4. The streams are now labeled to aid our analysis of this partition for tearing

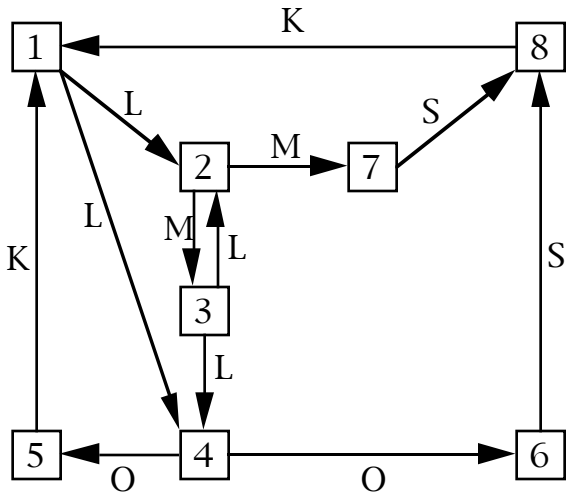


Fig. 6. Dual network for partition in Fig. 5

Table 1 Finding minimum tear streams for example problem using Barkley-Motard algorithm

first pass		second pass		third pass		fourth pass	
stream	inputs	stream	inputs	stream	inputs	stream	inputs
1	5,8	1	4,8	1	4,8	1	1
2	1,3	2	1,2				
3	2						
4	1,3	4	1,2	4	1		
5	4						
6	4						
7	2						
8	6,7	8	2,4	8	4		

We find streams 4 and 8 now have only one input. Replace them by their respective inputs streams in the problem, getting columns 7 and 8 above. We find stream 1 is in a self loop. It must be torn. Remove it from the problem. We are as there are no remaining streams in the problem.

Streams 1 and 2 form a minimum tear set.

There are problems where the above algorithm stalls as there will be no self loops and no streams with a single input. In this case, one can tear a stream which, if torn, will create a unit with a single input. If there are several options, one can choose to tear the stream which is listed as an input the most number of times. At this point there are no guarantees that one will produce a minimum tear solution.

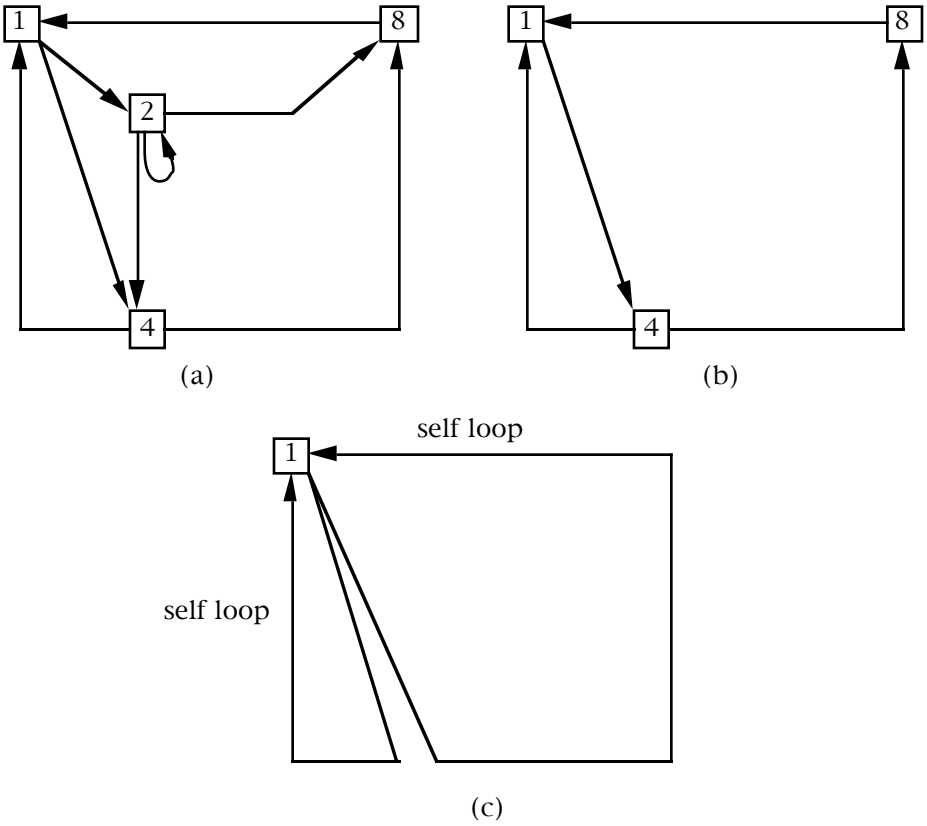


Fig. 7. Network reduction implied by Barkley-Motard Algorithm. (a) Network corresponding to first reduction when removing first set of streams having a single input (columns 3 and 4 in Table 1). (b) Network after removing stream 2 which is in a self loop (columns 5 and 6). (c) Network after removing second set of streams (unit 8 first and then unit 4) with single inputs, leading to stream 1 being in a self loop (columns 7 and 8).

Upadyhe and Grens Algorithm

Upadyhe and Grens [197x] noted there are families of tear sets that one might expect to have better convergence properties. Westerberg and Motard [19xx] developed a branch and bound algorithm for finding such tear sets efficiently both in terms of time taken and space required on the computer; it was the algorithm that the Aspen simulator used until 1994.

First we find all information loops in the partition. We do this by a stream tracing algorithm which is closely related to the stream tracing algorithm we used to partition the flowsheet. We will illustrate this algorithm by applying it to this example problem.

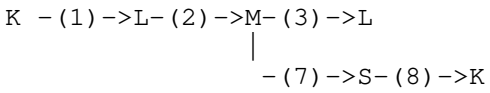
Start with any unit in the partition, for example unit K.

```
K -(1)->L-(2)->M-(3)->L
```

Unit L repeats. The two streams, 2 and 3, which connect the two appearances of unit L are placed on a list of loops, List 3.

List 3: {2,3}

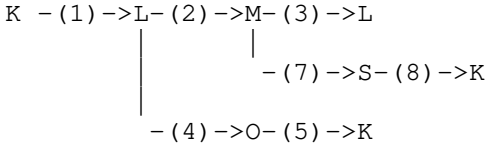
Start with the unit just before the repeated one trace any alternate paths from it.



K repeats. Place {1,2,7,8} on the list of loops.

List 3: {2,3}, {1,2,7,8}

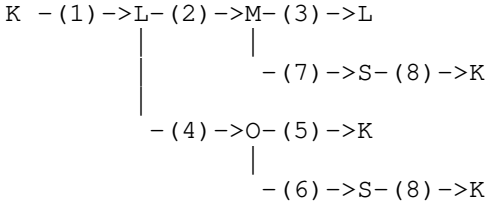
Back up to S and look for an alternate path leaving from it. There is none. Back up to M. Again there is no unexplored path leaving. Back up to L. This time there is another path. Continue with it.



K repeats. Place {1,4,5} on the list of loops.

List 3: {2,3}, {1,2,7,8}, {1,4,5}

Back to O for any alternate paths, which there is.



Again K repeats. Place {1,4,6,8} on the list of loops.

List 3: {2,3}, {1,2,7,8}, {1,4,5}, {1,4,6,8}

Return to S, to O, to L and finally to K, none of which have any alternate paths eminating from them. Since we have returned to the first unit on the list, we are done. There are four loops for this partition. We list them in a loop incidence array as shown in Table 2.

We now look for a set of streams which, if removed from the partition, will remove (tear) all loops. We want each loop removed exactly one time if possible. The loop incidence array aids in this process. For example, if we remove stream 1, loops 2, 3 and 4 are removed from the partition. Stream 3 will then remove loop 1. Each loop is broken precisely one time which is our goal.

Streams 1 and 3, therefore, are a *tear* set for this partition. Other tear sets are steams 2 and 4 and streams 3, 5, 6 and 7. This last set has four streams in it. There are reasons we might prefer to minimize the number of streams required making it less desirable as a tear set.

Table 2 Loop incidence array for partition

Stream Loop	1	2	3	4	5	6	7	8
1		x	x					
2	x	x					x	x
3	x			x	x			
4	x			x		x		x

It turns out we can find a whole family of tear sets. Find a first tear set, say streams 1 and 3. Now look for a unit, all of whose outputs are in that set. At least one will exist for which this will be true. We see that unit K is such a unit as stream 1 is its only output. Replace its output streams in the tear set with all of its input streams. Here we replace stream 1 by stream 5 and 8, getting the tear set 3, 5, 8; i.e., we get

```
{1,3} -[K]->{3,5,8}
```

We can repeat this until the entire family of "equivalent" tear sets is found. We will explain why we call them "equivalent" in a moment. Stream 8 is the only output from unit S. Replace stream 8 by steams 6 and 7; i.e.

```
{3,5,8}-[S]->{3,5,6,7}
```

The remaining steps are done similarly, generating

```
{3,5,6,7}-[M]->{2,5,6}-[O]->{2,4}-[L]->{1,3}  (where we started)
         |
          -[O]->{3,4,7}-[M]->{2,4}  (seen before)
```

There are no more sets which we can generate for this family. Among all these tear sets, we can choose among those that contain only two tears, namely sets {1,3} and {2,4}, rejecting the rest as candidates.

This "family" of tear sets is thus: {{1,3}, {3.5.8}, {3,5,6,7}, {2,5,6}, {2,4}, {3,4,7}. Let us remove one of these tear sets and partition and precedence order the units in this example. Choosing tear set {1,3} we see that we can compute unit L, which provides us with streams 2 and 4. We can compute units O and M in either order next, giving us streams 5, 6 and 7. That allows us to compute S and finally unit K. Fig. 8 shows the partial ordering that characterizes this precedence ordering for these units. It takes a 2-dimensional graph as shown in this figure to really show the precedence order for a group of units. We can see why the order is not necessarily unique. It can be either the ordering LOMSK or the ordering LMOSK.

Let's examine why we said all these tear sets are in the same family. We must tear all the outputs for the last unit in any ordering, here for unit K. If we move unit K to the beginning from the end of the ordering (i.e., form either

KLOMS or KLMOS), its output streams are no longer torn. However, all its inputs, streams 5 and 8, are. Since we have not moved units L and M relative to each other, stream 3 is still torn. The new tear set is {3,5,8}, the second tear set we discovered above. The algorithm above for finding all tear sets in the same family then moves unit S from the end to the beginning getting SKLOM or SKLMO (tear set {3,5,6,7}, then either unit M followed by O (getting MSKLO and then OMSKL) or unit M followed by O (getting (OSKLM and then MOSKL). We generate the remaining tear sets discovered above.

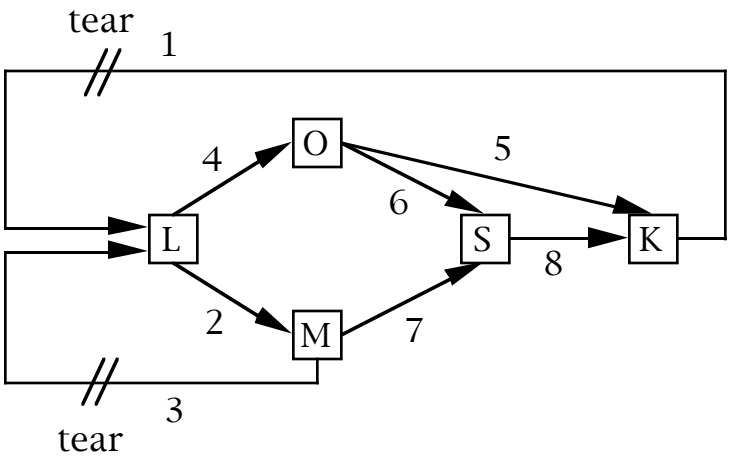


Fig. 8. Precedence order for partition in Fig. 5 when tearing streams 1 and 3

Suppose we choose to solve our flowsheet using successive substitution. We choose a tear set and guess values for each stream in it. Suppose we choose tear set {1,3}. We can then compute the units in the order LOMSK. We now have newly computed values for streams 1 and 3. We simply use them and start through the units again, getting

LOMSK LOMSK LOMSK·····

Suppose that instead of starting with L, we guess the streams needed to start with K in front (streams {3,5,8}). The successive substitution order would be

K(LOMS K)(LOMS K)(LOMS K·····

where the parentheses emphasize that we are really using the same ordering as above except we started with unit K. Guess streams {3,5,8} and then compute unit K, getting stream 1. Now treat this computed value as the guess we use for the ordering LOMSK. We would reproduce the same numbers for our computations as we attempt to converge these units using successive substitution. Therefore, the tear sets in the same family have the same numerical behavior when using successive substitution, a point made by Upadyhe and Grens [197x].

Let's choose a tear set that puts a double tear into one of the loops. Returning to the loops listed in Table 1, such a tear set would be streams {1,2} where loop 2, {1,2,7,8}, is torn twice while the rest are torn one time. Note, it involves only two streams so from the point of view of involving the fewest streams, it appears to be a good tear set.

We can compute unit M first (its input stream, stream 2) is guessed. That gives us streams 1 (guessed) and 3 (just computed) which allows us to compute L. Continuing we find the order for computing the units is MLOSK as shown in Fig. 9.

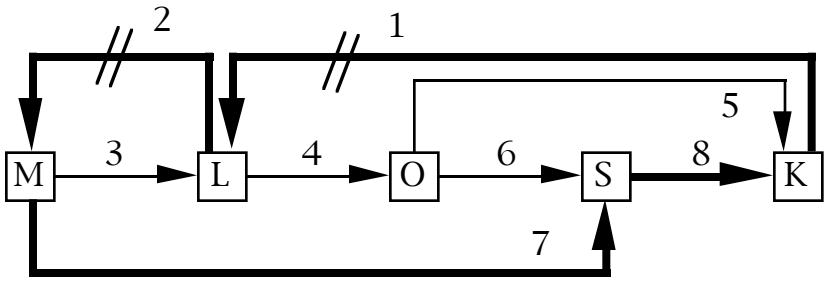


Fig. 9. Double tearing a loop

The impact of double tearing loop 2, highlighted in Fig. 9, is now evident. To solve, we guess streams 1 and 2 and compute once through all the units in the order shown. The new value that unit L computes for stream 2 impacts the next computation for stream M, but this computation is based on the old value for stream 1. The new value for stream 1 will impact the next computation for L but not for unit M. It impacts only unit L and the units following when we compute through the units the second time. Its new value will not impact unit M until we compute that unit a third time. This delays the transfer of information around this loop by one pass through the unit computations. Upadhye and Grens argued that such a delay slows the rate of convergence for successive substitution. It is for this reason we do not want tear sets that tear any loop more than one time. If we triple tear a loop, we will see the delay extend to two passes through the unit computations, and so forth.

An example where we cannot tear all the loops only one time each is shown in Fig. 10 We can write down all the loops for this example directly (although the reader might want to try the above loop detection algorithm to verify that we have them all).

All columns have two incidences in them so we can pick any one to be first. Pick stream 1. We would mark streams 2, 3, and 4 as ones we do not wish to choose so we will not double tear loop 1. Similarly, we mark stream 8 to avoid double tearing loop 3. That leaves us with streams 5, 6 and 7 only to tear the remaining loops. Symmetry of this problem say we can pick any one of these so we pick stream 5. To avoid double tearing loop 2 we mark streams 6, 7, and 8 (a second time) as ones we should not tear. All the streams are now marked as ones we do not wish to tear; loops 4 and 5 remain intact. To tear them we must choose two streams: one from the pair streams 2 and 7 and one from the pair streams 3 and 6. If we pick streams 2 and 3, we will triple tear loop 1. If we choose streams 6 and 7, we triple tear loop 2. Picking either streams 2 and 6 or streams 3 and 7 double tears both loops 1 and 2. As we argued above, we prefer double tearing loops to triple tearing them, so we choose either of the latter two options. Because of problem symmetry, we will always find we must at

least double tear two loops or triple tear one loop, no matter how we choose to carry out the tearing.

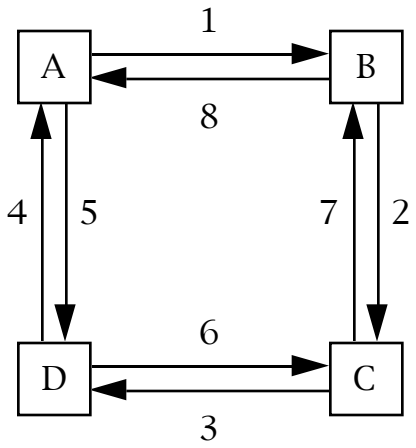


Fig. 10. Example where all loops cannot be torn exactly once

Table 3 Loop incidence array for partition in Fig. 10

Stream Loop	1	2	3	4	5	6	7	8
1	x	x	x	x				
2					x	x	x	x
3	x							x
4		x					x	
5			x			x		
6				x	x			

Convergence algorithms

When we tear streams to solve a group of units which are in a single partition, we have talked already of the possibility of converging them by using the newly computed value as the next guess. Experience has shown that such a convergence scheme will often converge, especially for partitions not containing reactors. Motard and Gena [197x] discuss why. We may, however, also put in special algorithms to do the reguessing. In this section we shall explore such algorithms. We can find many papers in the flowsheeting literature describing such algorithms which were written in the 1960s and 1970s. First let us explore computational controllers again for a moment, with the purpose of bringing the reguessing algorithms together with those for computational controllers.

Implicit vs. explicit iteration loops

We note that there is something different about the convergence loop we set up when using a computational controller and that we set up when converging a recycle stream. For a process stream, we start with a guess for

the stream and ultimately end up with a computed set of values for that same stream. We use the difference between what we guess and what we compute to be an error that tells us if we should carry out more iterations. The computed values for the stream could become our new guess for them. We noted thtat the method that uses these computed values as the new guess is called the successive substitution method.

A computational controller, on the other hand, adjusts one parameter to drive another to a desired value. We could not use the computed value for the measured variable as the next guess for the manipulated variable. One cannot use the successive method for this case.

We distinguish between these two types of convergence loops as being *explicit* (guess a set of values for some variables and compute new values for these same variables) or *implicit* (guess a set of values for a set of variables and compute values for a different set of variables). In the former the error is the change between what we guess and what we calculate using that guess while in the latter the error is the difference between what we calculate and what we specify that we want as the value.

Both these types of computational loops require iteration to solve. The explicit loop allows us to use successive substitution to reguess values; the implicit loop does not.

To see that the above really makes a difference, look again at the recycle loop for our process to manufacture hexene as shown in Fig. 1. We indicated that any of the streams around the loop could become the one we guess to get started. This is possible because the loop is an explicit loop. We are forced to tear an implicit loop in only one place -- between the measured and the manipulated variables.

It is worthwhile looking at the similarities as well as the differences between these forms of iteration. We can form the abstract concept of an information stream. Such a stream is formed by merging the measurement passing to a controller followed by the value it passes to the manipulated variable. Conversely, when we tear a process stream, we can mentally place a unit between the two torn parts that represents the convergence block we intend to use to reguess the stream value for the next iteration. Fig. 11 illustrates the similarity we are trying to establish here. The computational controller is playing the role of the convergence block for an information stream.

We gain two conceptually useful ideas here. Thought of in this manner, a computational controller becomes a stream in our flowsheet on which we can use the tearing algorithms we presented above. The key difference is that we must choose such a stream to be in the tear set when we are selecting which streams to tear because we cannot use successive substitution to converge such an information stream.

The other conceptual gain is that we may merge the computational controller algorithm with that used for the convergence block for streams, an idea we shall now explore. Thus we may converge all the tear streams and controllers in a single partition simultaneously using the same convergence method.

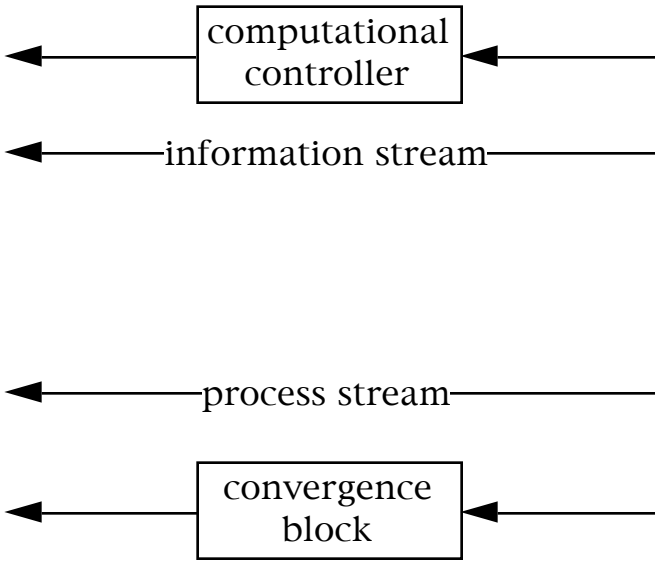


Fig. 11. Simularities between information streams and process streams

The convergence problem for flowsheets

We can characterize the convergence problem for a flowsheet as the solving of a set of *n* nonlinear equations in *n* variables.

$$\underline{h}(\underline{x}_{\text{guess}}) = \underline{0}$$

where the equations are

$$\underline{h}(\underline{x}_{\text{guess}}) = \begin{cases} \underline{x}_{\text{guess}} - \underline{x}_{\text{calc}} & \text{for process streams} \\ \underline{y}_{\text{calc}} - \underline{y}_{\text{setpoint}} & \text{for controlled variables} \end{cases}$$

For process streams, variables $\underline{x}$ are those characterizing it such as its flowrate, compositions, temperature and pressure. For a computational controller, $\underline{x}_{\text{guess}}$ is the manipulated variable while $\underline{y}_{\text{calc}}$ is variable whose value we wish to drive to the specified value $\underline{y}_{\text{setpoint}}$.

We shall discuss numerical methods to converge a set of simultaneous nonlinear equations in a later section after we first discuss developing the equations for a model in the next section.

Modeling - an example

What is a model

A model is much more than a set of equations. To be truly useful, we need to know, among other things, its purpose, the assumptions on which it is based and (considering its purpose) how we intend to decide if it is a good model.

Let us create a simple model, that of a simple binary flash unit, to see many of the issues involved.

Description of its purpose

To create a model we need first to describe its purpose. For example, consider that we wish to create a steady-state model for a binary flash model. We intend to use the model to study the separation of different mixtures of two species versus their relative volatility. In particular, we are curious about how large the relative volatility has to be for a simple flash separator unit to give us 99% recovery of the light species in the vapor stream while giving us a 99% recovery of the heavy species in the liquid stream.

This model is likely a very simple one to construct. For example, we will not need to couple it with a physical properties estimation package. It should consist of only a few equations, perhaps around 10 to 20 of them. If our purpose had been to predict azeotropic behavior between acetone and chloroform, the modeling problem would be very different and decidedly more difficult.

A significant part of the model is, therefore, a description of its purpose. Without it the modeler has little guidance as to the fidelity required for it. Also the purpose is the best first document to give to someone else when explaining a model.

Description of its underlying principles and assumptions

We shall assume the flash operates as an equilibrium flash. The underlying principles and assumptions for such a model are quite simple. We can model it using component material balances and a means to express equilibrium in terms of relative volatility. We characterize the species involved only by their relative volatility; there is no need to identify the exact species, therefore. We do not need a heat balance nor a momentum balance.

Explicitly stating these assumptions helps clarify the nature of the model we shall produce. We can measure these against the description of its purpose above to pass judgment on them.

Criteria to assess its success

We would like to be able to specify a feed for the flash and its relative volatilities. The desired output should be the composition of the top and bottom products. We would like answers to be accurate to at least three significant places. Just what that might mean for convergence tolerances, we shall have to discover when we do manage to solve our model.

We also want it to solve rapidly, i.e., perhaps within a fraction of a second. However, we have no need for this requirement. If the model does take longer than this, it should make us wonder if we stated it correctly. As we learn more about the model, we will likely revise these criteria for assessing its success.

Its defining equations

We can now start to write the equations we expect will be needed. This process is iterative in nature as we are likely to change our minds on how to structure the equations several times as we write them. Much of the restructuring is to *make it clearer to us* that we are writing just the right equations, neither writing more than we should nor forgetting any. This emphasis on creating an elegant form for the model may be frowned upon by many. The argument many modelers make is that of expediency: it is good enough as long as it works. Our bias is in the other direction. An elegant model is easier to examine and mentally check for correctness. A small crude model is okay, but this approach is fraught with danger if the model is large.

Fig. 12 illustrates a simple flash unit.

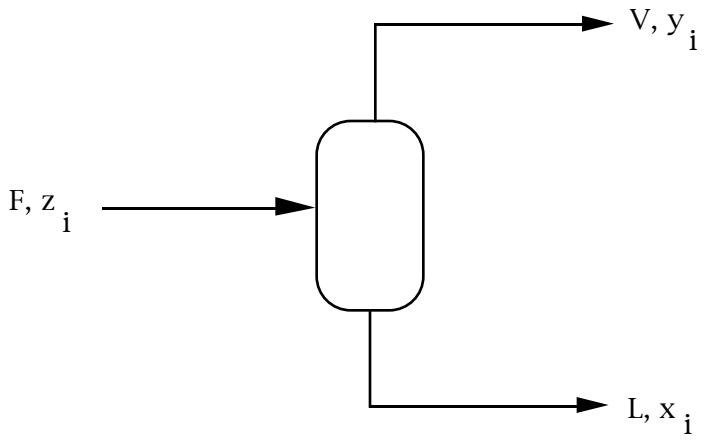


Fig. 12. A flash unit

The defining equations are first the component material balances. There are two species so we write two of them. We do have to decide whether to write them in terms of molar flows

$$f_1 = v_1 + l_1; \quad f_2 = v_2 + l_2 \tag{1,2}$$

where *f, v* and *l* are molar flow rates e.g., {gm_moles/s}. We could also write the material balances in terms of mole fractions, *z*, *y* and *x*, and overall stream flows, *F*, *V* and *L*.

$$z_1F = y_1V + x_1L; \quad z_2F = y_2V + x_2L$$

The former equations are linear in the variables we used to write them. Experience suggests this form is better for solving so we shall select it. To have this linearity in the equations, we pose the following general guideline for modeling processes.

> Modeling rule: Express material balances in terms of molar or mass flows.

We express equilibrium using relative volatilities as follows. This form is a generalization of the form we might remember from when we were first taught about relative volatilities (substitute $\alpha_2=1$ and $x_2 = 1-x_1$ to get the more familiar form).

$$y_1 = \frac{\alpha_1}{\alpha_1 x_1 + \alpha_2 x_2} x_1 \qquad (3)$$

$$y_2 = \frac{\alpha_2}{\alpha_1 x_1 + \alpha_2 x_2} x_2 \qquad (4)$$

We just wrote two such relationships. At this point we might wonder to ourselves if we are allowed to write two of them. After all, in our first course on distillation, we wrote only one equilibrium equation for the more volatile species. We put this thought aside for the moment with a mental flag set to return to it as we proceed. Also, we just wrote these equations in terms of mole fractions rather than molar flows. At the moment this appeals, but we may wonder if we will end up liking it. Again we put this question aside to be considered later.

There are other equations we know are likely to play a role. First the mole fractions for our problem have to sum to unity:

$$x_1 + x_2 = 1; \quad y_1 + y_2 = 1 \qquad (5,6)$$

Also, if we intend to use molar flows for the two components as well as mole fractions, we need to define their interrelationship to each other. We write

$$x_1 = \frac{l_1}{L}; \quad x_2 = \frac{l_2}{L}; \quad y_1 = \frac{v_1}{V}; \quad y_2 = \frac{v_2}{V} \qquad (7,8,9,10)$$

Note that we related all mole fractions to their respective molar flows. We may wonder if we have the right to do so. Our reason is that we want to treat all components equally when writing equations. We would definitely want this symmetry to occur if this were a model for more than two components.

Do we need to define *L* and *V*? Summing eqns 7 and 8 and applying eqn. 5 gives:

$$\frac{l_1 + l_2}{L} = 1$$

which implicitly defines *L*. Summing eqns 9 and 10, together with 6, also implicitly define *V*. We do not need to write extra equations to define *L* and *V*. They are the "scale factors" that make the mole fractions defined by eqns. 7 to 10 add to unity.

Did we write too many equations?

We have written ten equations here (excluding the second form for the material balances). Are they all independent? Did we forget any? Are they correct equations for our problem? We have little doubt they are each correct, but we should have serious doubts that they are all necessary and/or that we did not forget any.

To decide if we wrote too many equations, let us see if we can derive any we wrote from others we wrote. Adding the two equilibrium equations, we get

$$y_1 + y_2 = \frac{\alpha_1 x_1 + \alpha_2 x_2}{\alpha_1 x_1 + \alpha_2 x_2} = 1$$

This equation says that the vapor mole fractions sum to unity without our having to say they do. We should remove either one of the equilibrium equations or the equation summing the vapor mole fractions to unity. We might select to remove the second equilibrium equation; however, this may insult our sense of symmetry for the problem. On the other hand, removing the statement that the mole fractions add to unity also bothers us as it means that we must treat the vapor stream different from the liquid stream leaving the flash unit.

Another approach, similar to our experience above with L and V, is to rewrite the equilibrium equations as follows, introducing a new variable $\overline{\alpha}$ to scale the equations.

$$y_1 = \frac{\alpha_1}{\overline{\alpha}} x_1 \qquad (3')$$

$$y_2 = \frac{\alpha_2}{\overline{\alpha}} x_2 \qquad (4')$$

If we sum these two equations, we get

$$\overline{\alpha} = \alpha_1 x_1 + \alpha_2 x_2$$

Now we can leave both of these equilibrium equations and both of the equations indicating that the mole fractions add to unity. This is more satisfying because of its symmetry in defining things. The new variable $\overline{\alpha}$ will, when solved for by the model, yield the value shown by the last equation above. This last equation cannot be a part of the model if all the other equations are. However, we do see an easy way to make an initial guess for it. It must be somewhere between the two relative volatilities, α_1 and α_2.

The last, and definitely satisfying characteristic for this model, is that we see that the y values are proportional to the x values with the relative volatilities being the constants of proportionality, a nice interpretation for this form of expressing equilibrium.

We still do not know if the model is correct, but we might be ready to try using it anyway. If it "works," we will assume it is correct.

Selecting which variables to fix and which to compute

To solve the flash unit, we count the variables used in the model, discovering there are 15 of them: y_1, y_2, x_1, x_2, α_1, α_2, $\overline{\alpha}$, f_1, f_2, l_1, l_2, v_1, v_2, L and V. As noted, we wrote 10 equations. To make this problem a *square* one (i.e., one with an equal number of unknown variables and equations), we can select five variables and provide values for them. In the problem description, we stated that we wanted to specify the feed fully which happens if we specify f_1 and f_2. Also we can specify the two relative volatilities. If our analysis for degrees of freedom is correct, we can also chose one more. Let us require that the total liquid flow leaving the unit equal half the flow of the feed entering, i.e., we add the equation

$$L = 0.5F \qquad (11)$$

This specification is one that most chemical engineers would think reasonable for a flash unit.

Checking if model will solve

For a problem as simple as this one, we can now attempt to solve the equations *mentally* to see if the model "works."

1. Fix f_1 and f_2 to 1 {gm_mole/s} each.

2. Fix α_1 to 2 and α_2 to 1.

3. From eqn. 11, we compute L = 1 {gm_mole/s}.

4. We can go no further using the equations as they are now written. We must do some algebra (often difficult or impractical to do) or some guessing and iterating. Let us guess l_1 = 0.4 {gm_mole/s} (less than half the more volatile species will exit in the liquid product) .

5. We can use the following five equations in the order indicated to compute

   eqn 7 --> $x_1 = 0.4$
   eqn 5 --> $x_2 = 0.6$
   eqn 1 --> $v_1 = 0.6$ {gm_mole/s}
   eqn 8 --> $l_2 = 0.6$ {gm_mole/s}
   eqn 2 --> $v_2 = 0.4$ {gm_mole/s}

6. We are stuck again. Some simple algebra will, in fact, rescue us. We can actually compute $\overline{\alpha}$ using the definition given earlier to be

   $$\overline{\alpha} = \alpha_1 X_1 + \alpha_2 X_2 = 2(0.4) + 1(0.6) = 1.4$$

   and V

$$V = v_1 + v_2 = 0.6 + 0.4 = 1\{gm_mole / s\}$$

Neither of these two equations are in our set so they should replace two which are, e.g., eqns 6 and 9.

7. We can now use the following eqns to compute

   eqn 3' --> $y_1 = 0.57143$
   eqn 4' --> $y_2 = 0.42857$

   We note these add to unity automatically as we used the right value for $\overline{\alpha}$ in step 6. This observation justifies our replacing eqn 6 when we used the definition we did in step 6 for $\overline{\alpha}$.

7. We have one unused equation left and no unevaluated variables. Earlier we guessed l_1. We can use the leftover equation as an error function whose value would be zero had we guessed the right value for l_1. Evaluating the error between the LHS and RHS of eqn 10, we get

$$y_2 - \frac{v_2}{V} = 0.57143 - \frac{0.4}{1} = 0.17143$$

   which is not zero.

8. We need now to return to step 4 where we guessed a value for l_1. We should reguess it until the error in step 7 is reduced to zero.

If we do this we shall find the solution finally to be

f_1=1 {gm_mole/s}
f_2=1 {gm_mole/s}
α_1=2
α_2=1

for the variables whose values we fixed and

L= 1.00000 {gm_mole/s}
V= 1.00000 {gm_mole/s}
l_1=0.41421 {gm_mole/s}
l_2=0.58579 {gm_mole/s}
v_1=0.58579 {gm_mole/s}
v_2=0.41421 {gm_mole/s}
x_1=0.41421
x_2=0.58579,
y_1=0.58579
y_2=0.41421
$\overline{\alpha}$ =1.41421

for the variables that we need to calculate. The solution procedure seems to work and give reasonable answers. For a problem as simple as this one, we

shall assume, therefore, that the equations are correct. We seem to have a well-posed model.

Our purpose in creating the above solution procedure was not to solve the equations this way but to demonstrate that we might have a correct model. In fact, what we are really trying to discover is if we have an obviously incorrect model.

Homework 1

1. Consider computing the mass of metal required to construct a thin walled cylindrical vessel as shown in Fig. 13. Wall thickness is t. Write the description of purpose, the underlying principles and assumptions, the criteria for success and the defining equations for the model. Check the model to see if it appears to "work."

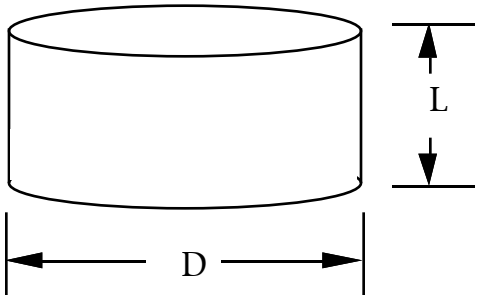


Fig. 13. A simple tank

2. Repeat question 1 for a closed fixed volume vessel containing water and water vapor as shown in Fig. 14. One can estimate the vapor pressure for water using the Antoine equation (Reid et al, 1977):

$$P^{sat}_{water}\{\text{mm Hg}\} = \exp\left(18.3036 - \frac{3816.44}{T\{\text{K}\} - 46.13}\right)$$

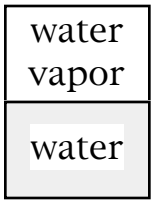


Fig. 14. A simple boiler

Solving the equations using Newton's method

One can solve the equations for our binary flash model numerically using Newton's method. Newton's method linearizes the equations and solves the linear equations as a way to approximate the solution to the nonlinear ones. Fig. 15 illustrates the method for a single equation in one unknown. Guessing

the unknown, *x*, at x_0, the method evaluates $h(x_0)$ as well the slope of the function, $(dh/dx)_{x_0}$ at x_0. A linear equation that goes through the function at x_0 is:

$$y(x) = h(x_0) + \left[\frac{dh}{dx}\right]_{x_0} (x - x_0)$$

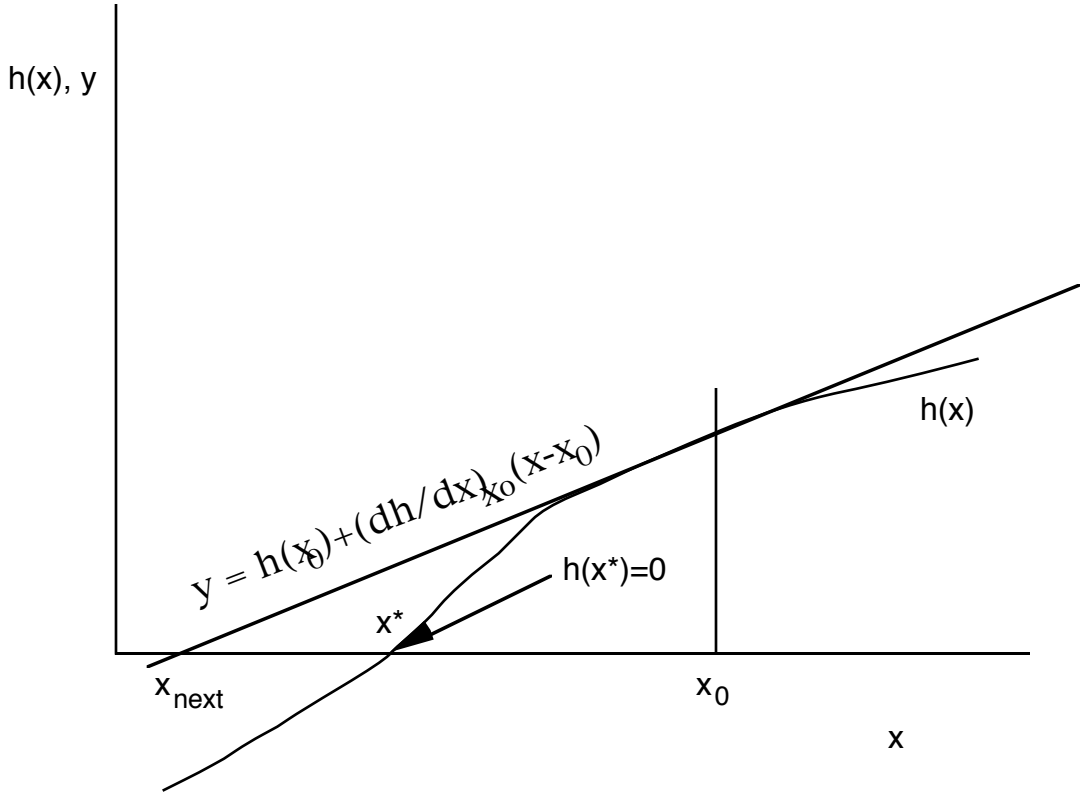


Fig. 15. Newton method for one equation in one unknown

Solving for *x* when *y* is set to zero gives us the next guess for an x that will solve $h(x)=0$.

$$x_{\text{next}} = x_0 - \left[\frac{dh}{dx}\right]^{-1}_{x_0} h(x_0) \equiv x_0 + \Delta x_0$$

To solve the ten equations above using Newton's method requires us to linearize all ten equations, which we can do as follows. Remember that we have fixed f_1, f_2 ,L, α_1 and α_2 to get a set of ten equations in ten unknowns.

We first look at linearizing eqns 1 and 2. We repeat them here for convenience.

$$f_1 = v_1 + l_1; \quad f_2 = v_2 + l_2 \qquad (1,2)$$

Since f_1 is fixed, linearizing eqn 1 gives:

$$f_1 = v_1 + \Delta v_1 + l_1 + \Delta l_1$$

which we rearrange to put the unknowns (the perturbations for the variables) to the left-hand- (LHS) side while putting the known terms to the right.

$$-\Delta v_1 - \Delta l_1 = -(f_1 - v_1 - l_1)$$

As with the case of a single equation in one unknown considered above, the right hand side is the negative of the original equation (rewritten so it is in the form *f(x)=0* by subtracting its right hand side from its left hand side). If it is zero, then the original equation is satisfied. We, therefore, use the right hand side of the perturbation equations to detect convergence.

If we then use the values we guessed for the first pass earlier through the equations for all the variables above, we can evaluate the right hand side to get

$$-\Delta v_1 - \Delta l_1 = -(1 - 0.6 - 0.4) = 0 \qquad (1")$$

Linearizing eqn 2 gives

$$-\Delta v_2 - \Delta l_2 = -(f_2 - v_2 - l_2) = -(1 - 0.4 - 0.6) = 0 \qquad (2")$$

We can rearrange equation (3') to a form that is easier to linearize by multiplying through by $\overline{\alpha}$:

$$\overline{\alpha} y_1 = \alpha_1 x_1 \qquad (3')$$

Linearizing (and remembering that α_1 is fixed) gives:

$$\begin{aligned}\overline{\alpha} y_1 + \Delta(\overline{\alpha} y_1) &= \overline{\alpha} y_1 + \overline{\alpha}\Delta y_1 + y_1 \Delta\overline{\alpha} \\ &= \alpha_1 x_1 + \Delta(\alpha_1 x_1) = \alpha_1 x_1 + \alpha_1 \Delta x_1\end{aligned}$$

Rearranging to put the perturbation variables to the left hand side gives:

$$\overline{\alpha}\Delta y_1 + y_1 \Delta\overline{\alpha} - \alpha_1 \Delta x_1 = -(\overline{\alpha} y_1 - \alpha_1 x_1)$$

We substitute in current values for the variables, getting:

$$\begin{aligned}&1.4\Delta y_1 + 0.57143\Delta\overline{\alpha} - 2\Delta x_1 \\ &\quad = -(1.4 * 0.57143 - 2 * 0.4) = 0 \qquad (3")\end{aligned}$$

We could continue with the remaining equations, and, hopefully, the approach is clear. When done, one will have ten **linear** equations (1" to 10") in ten perturbation variables, Δy_1, Δy_2, Δx_1, Δx_2, $\Delta\overline{\alpha}$, Δl_1, Δl_2, Δv_1, Δv_2 and V. One can solve these using Gaussian elimination. The next guesses for each of the variables are the current values plus these perturbations. If all the right-hand-sides are zero, we have solved the equations.

The equations are then relinearized at this new point, again getting a solution for the perturbations, which are again added to the then current variable values. If one is successful, the method converges, usually quite quickly.

Homework 2

1. Linearize the remaining equations for the two component flash model. Guess the original point to be the values determined above. Set up the Newton equations and solve the linear equations that result for the next guess for all the variables.

2. Repeat for the equations for the cylindrical vessel defined in problem 1 of the first set of homework problems.

3. And repeat again for the closed vessel contain water and water vapor (Homework 1, problem 2).

Improved Newton based solving methods

One must often have an extremely good estimate of the answer to get close enough that the Newton method for solving nonlinear equations will converge. There are many things one can do numerically to increase the "ball" of convergence. We summarize a few of them here.

Line search

One option is to take a full Newton step and see if that point leads to a reduction in the sum of squares of the functions (remember that our goal is for the functions to be zero so this sum of squares measures how far we are from satisfying our goal). If it does not, then the approach is to shorten the step until one finds a point where such a reduction does occur.

Levenberg/Marquardt search

If one is going to shorten the step taken from that of a full Newton step, the direction to search should move more in the direction of *steepest descent* , the direction where one reduces the sum of squares of the functions most rapidly if one were to take an extremely short step. This is the idea behind methods first developed independently by Levenberg (194x) and Marquardt (195x). Fig. 16 illustrates.

Shown in this figure are the larger oval contours where the sum of squares of the functions are constant. Assume we are currently at the point shown in the lower left. We show circles surrounding this point of constant step length. We compute two directions: the Newton direction and the direction of steepest

descent. If the functions are linear, the Newton step will point precisely to the point where the functions are all zero, both in terms of the direction taken and the length of the step, so we show it pointing here close to the point where the sum of the functions is zero. One can prove that the steepest descent direction is orthogonal to the contour lines, which is how we show it here. The curve starting from the end of the Newton step back toward the starting point and moving always closer to the steepest descent direction as the distance from the starting point shortens is the Levenberg/Marquardt direction.

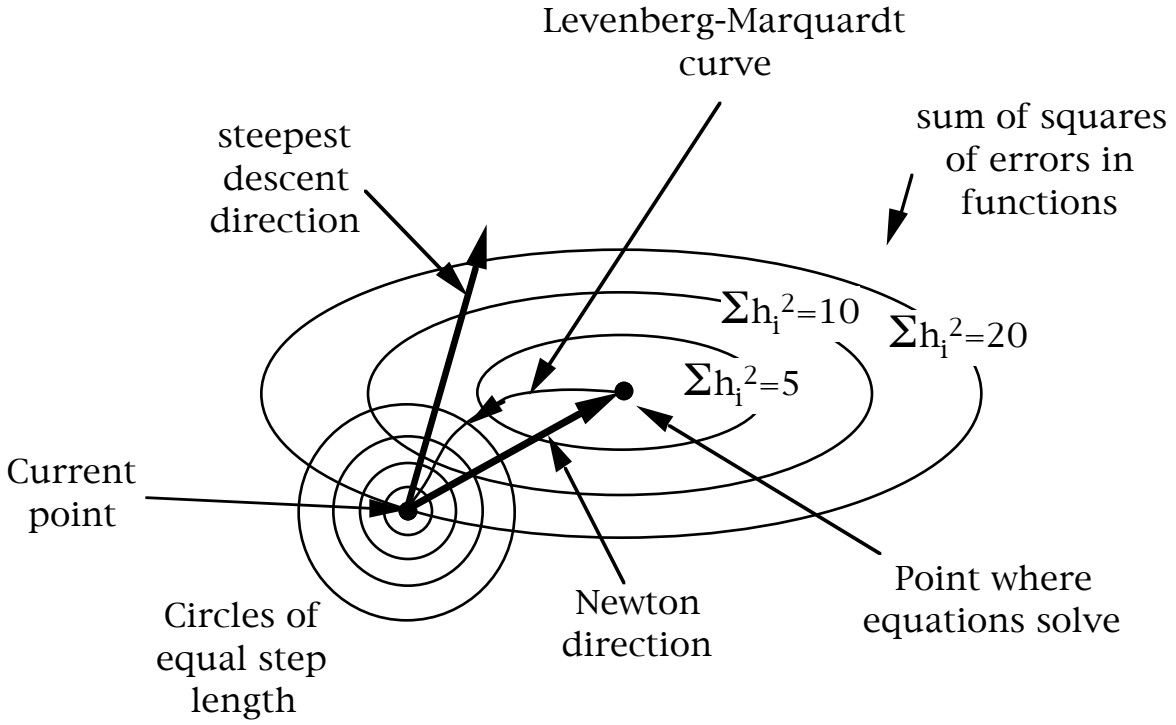


Fig. 16. Levenberg/Marquardt steps

If our goal is to take a step of fixed length - say to one of the circles surround the current point, then the size of the step is on the Levenberg/Marquardt trajectory. To see this, move around one of these circles to the point where the sum of squares of the functions is minimized on it. If one were to illustrate this trajectory in more than two directions, it would move out of the plane defined by the Newton and steepest descent directions.

We use a modification of this method in the ASCEND system (Westerberg and Director, 197x). The modification keeps the step taken in the plane defined by the two directions and, as a result, retains the sparsity and numerical conditioning of the original problem. The original Levenberg/Marquardt method is twice as dense and has a condition number the square of that for the original problem (which makes it a worse condition number).

Trust region/linear programming

Another approach is to put constraints on the maximum size of the step one is willing to take. These constraints form what is called a *trust region*. To compute a step to take, we can linearize each function, h_i, writing

$$h_i(\underline{x} + \Delta\underline{x}) \approx h_i(\underline{x}) + \sum_j \frac{\partial h_i}{\partial x_j}\Delta x_j = p_i - n_i \quad \text{for all i}$$

$$-a_j \le \Delta x_j \le a_j \text{ for all j.}$$

where p_i and n_i are positive numbers. If we do not satisfy the linearization, at least one of these two will be nonzero.

Then we form the linear objective

$$Min \sum_i (p_i + n_i)$$

which attempts to drive the deviations to zero. The trust region constraints bounding the step size can prevent us from driving the deviations to zero. Note that since both p_i and n_i must be positive, at most one of them will be away from zero with this objective. With a simple transformation on the Δx_i, one can make all of this into a linear program where one minimizes by choice of the Δx_i, p_i and n_i. Solving gives the "best" direction in which to move subject to remaining within the bounds we put on the step sizes. The best value is when p_i and n_i are both zero, in which case we compute a Newton step. One can also add bounds on the variables themselves to this formulation. Bullock and Biegler (199x) explored this approach to computing directions for solving nonlinear equations and found it works very well.

Continuation methods

The last class of methods we shall briefly discuss here is the class of continuation methods. To get the idea, suppose we evaluate the functions at some initial guess, getting each equal to $h_i(\underline{x}_0)$ (i.e., the value of h_i at the initial guess, $\underline{x}_0$). Rather than attempting to drive the functions all the way to zero, we could try to move them a small way toward zero, say to $0.9h_i(\underline{x}_0)$ for each h_i. In other words we solve

$$h_i(\underline{x}) - 0.9h_i(\underline{x}_0) = 0$$

Our hope is that by staying close to where we started we are within the ball of convergence for the method we are using to converge the equations. If we succeed in solving with 0.9, we reduce it to 0.8 and repeat, finally reducing the multiplier to 0, at which point we have solved our original equation. One can see two issues here even for this approach: (1) just how fast can one reduce the multiplier and (2) this approach is a lot more work so we would likely use it only if we fail to converge without using continuation. The use of a multiplier as shown here is a form of algebraic continuation method. There are two other forms worth mentioning. One is to convert the problem into a problem of solving differential equations in "time" where each step attempts to move

closer to where the functions are zero. The speed with which one can reduce the multiplier now is a problem for the integration package to decide.

Another is to use what are called *natural continuation* formulations. As an example, chemical engineers often write nonideal vapor/liquid equilibrium in the following form:

$$y_i = \frac{\gamma_i f_i^0}{\phi_i P} x_i$$

where γ_i is an activity coefficient, f_i^0 a standard state fugacity which is close to being the vapor pressure for the pure component i, ϕ_i is a fugacity coefficient, P is the pressure, and y_i and x_i are mole fractions. The nonideality is in the activity and fugacity coefficients. We can alter this relationship by adding in the factor t as follows:

$$y_i = \left(\frac{\gamma_i}{\phi_i}\right)^t \frac{f_i^0}{P} x_i$$

and creep up numerically on the solution by letting t move slowly from 0 (where the nonideal behavior is suppressed) to 1 where we solve the original problem. Taylor (198x) suggested this last approach and demonstrated how well it worked by solving some really difficult reboiled absorber problems.

Scaling

Convergence is detected when the right hand sides to the linearized equations are all very small. However, what is small depends on the scaling used. Scaling also affects the numerical performance of many convergence algorithms.

Interestingly, Newton's method is scale invariant for an infinitely accurate computer. To be scale invariant means that the computed step does not depend on how the scaling used for the variables and equations. We can see this to be the case by examining the Newton equations after we have rescaled by dividing each variable and each equation by a nominal value for it.

Our original functions are

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

We expand these equations in a Taylor series to first order to obtain the Newton equations (using matrix/vector notation), getting

$$\underline{h}(\underline{x} + \Delta\underline{x}) \approx \underline{h}(\underline{x}) + \left[\frac{\partial \underline{h}}{\partial \underline{x}^T}\right]_{\underline{x}} \Delta\underline{x} = \underline{0}$$

The latter part can be rearranged to give us Δx in the form we usually associate with the Newton method:

$$\left[\frac{\partial \underline{\underline{h}}}{\partial \underline{\underline{x}}^T}\right]_{\underline{x}} \Delta \underline{x} = -\underline{h}(\underline{x})$$

Scaling the variables says that we divide each variable by its nominal value. We can accomplish this by premultiplying the vector of variables by a diagonal matrix (only the diagonal has nonzero terms), where the diagonal element is the reciprocal of the nominal value of the variables.

We can see this with the following example where we rescale the vector of variables $\underline{x}$. The nominal value for x_1 is 10 and for x_2, 0.3. We call the newly rescaled variables $\underline{y}$.

$$\begin{bmatrix} \frac{1}{10} & 0 \\ 0 & \frac{1}{0.3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{10} x_1 \\ \frac{1}{0.3} x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

In a similar fashion we can rescale the functions by premultiplying them by a diagonal matrix. The resulting equations become:

$$\underline{\underline{D}}_h \underline{h}(\underline{\underline{D}}_x \underline{\underline{D}}_x^{-1} \underline{x}) = \underline{\underline{D}}_h \underline{h}(\underline{\underline{D}}_x \underline{y}) = \underline{0}$$

where $\underline{y} = \underline{\underline{D}}_{\underline{x}}^{-1} \underline{x}$.

If we carry out a Taylor series expansion on these to first order to derive the Newton equations for them, we get:

$$\begin{aligned} \underline{\underline{D}}_h \underline{h}(\underline{y} + \Delta \underline{y}) &\approx \underline{\underline{D}}_h \underline{h}(\underline{y}) + \underline{\underline{D}}_h \left[\frac{\partial \underline{\underline{h}}}{\partial \underline{\underline{x}}^T}\right]_{\underline{x}} \left[\frac{\partial \underline{x}}{\partial \underline{y}^T}\right] \Delta \underline{y} \\ &= \underline{\underline{D}}_h \left\{ \underline{h}(\underline{y}) + \left[\frac{\partial \underline{\underline{h}}}{\partial \underline{\underline{x}}^T}\right]_{\underline{x}} \underline{\underline{D}}_{\underline{x}} \underline{\underline{D}}_{\underline{x}}^{-1} \Delta \underline{x} \right\} \\ &= \underline{\underline{D}}_h \left\{ h(\underline{y}) + \left[\frac{\partial \underline{\underline{h}}}{\partial \underline{\underline{x}}^T}\right]_{\underline{x}} \Delta \underline{x} \right\} = \underline{\underline{D}}_h \underline{0} \end{aligned}$$

Multiplying the last row above through by the inverse of $\underline{\underline{D}}_h$ gives us back our original Newton equations. Thus the Newton method will compute the same step whether we have scaled or not.

Most other numerical methods are not scale invariant. Broyden's method and the steepest descent methods are not, for example. Thus altering the scale for these methods will alter the steps taken.

Computers have finite accuracy and as a result there is an impact of scaling even on Newton's method. The accuracy of solving linear equations on a finite word length computer can be strongly affected by the pivot sequence one selects. For example, if we do all computations to four digits of accuracy (using rounding), the solution to the following equations changes with the pivot sequence used.

$$\begin{bmatrix} 1 & 10^{-6} \\ 10^{-6} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Pivoting on the large elements *(1)* leads to $x_1=1$ and $x_2=2$. Pivoting on the small elements (10^{-6}) leads to $x_1=1$ and x_2 equals zero. The 2 on the RHS has no affect on the solution. The first is the correct solution.

Scaling of equations as a part of solving them numerically has been the theme of many publications. The general consensus is that scaling should be done based on physical arguments. Automatic scaling often leads to inappropriate scaling. We can offer an argument for this observation. Suppose one is computing a model in which some of the variables are mole fractions. How should they be scaled? We might at first glance decide that their nominal values should all be unity. However, suppose one of the species is a trace species whose values is in parts per billion and that its value is being calculated very specially to allow us to maintain an accurate number in that range. Scaling by unity is totally incorrect for this variable. In another case one part per billion may be effectively zero to us.

If the variables and equations are properly scaled, then we can use absolute measures to decide on what is small. Suppose all variables are scaled to be near to 1.0 in value. Then we can test the change to the rescaled variable to decide on the magnitude of a change in it. We often consider a change of 10^{-8} to be small. Similarly, suppose that equations are rescaled so all terms that are added or subtracted to form them are of magnitude 1.0. Then a small error is again discernible by its absolute value. We often consider an equation to be converged if the magnitude of its error is of the order of 10^{-8}.

We argue that scaling has to be done as follows.

Scale all variables so their values are near to unity. We define the nominal value of a variable as the positive number by which we shall divide that variable to rescale it. It is easy to select a nominal value for an absolute temperature. We can choose room temperature (about 300 {K}). Similarly, it is often easy to scale pressures by dividing by a nominal value of 10 {atm}. Mole fractions typically have a nominal value of unity.

Mass and energy flows are not so easy. Flows can vary by many orders of magnitude in the same problem.

A good rule of thumb for flows in chemical processes is that a typical process feed flow is about 1 {kg/s}. A large one, such as a coal liquefaction process, is about 30 {kg/s}. Internal flows in columns will generally be no more than ten times their feed flow. We generally rescale by performing two or three iterations in an attempt to solve them using an initial nominal value equivalent to 1 {kg/s}. Then we reset the nominal value to be the larger of the magnitude of the actual flowrate or 0.01 {kg/s}.

To rescale equations, we evaluate each of the terms which are added/subtracted to form it, using nominal values for all the variables. Generally this will lead to a rescaling by unity.

Let us examine scaling for eqn 1 in the above model using these guidelines. We repeat eqn 1 for convenience.

$$f_1 = v_1 + l_1 \qquad (1)$$

All the variables in it are molar flows. The feed flow f_1 is fixed for the problem and is the sum of the other two. f_1 has a fixed value of 1 {gm_mole/s}. This value, therefore, is an excellent nominal value to use for all three flows. Divide each flow by 1 {gm_mole/s}.

If none of the flows is known in this equation, then we could proceed as follows. Assume the molecular weight of the material is about 100 g/mole. Then 10 mole is 1 kg. Set the nominal value for flowrate to be 10 {gm_mole/s}. After performing two Newton iterations, reset the nominal value to the larger of the flow and 0.1 {gm_mole/s}.

Let us suppose we have rescaled this equation. It then has the form:

$$\frac{f_1}{1\ \{\text{gm_mole / s}\}} = \frac{v_1}{1\ \{\text{gm_mole / s}\}} + \frac{l_1}{1\ \{\text{gm_mole / s}\}}$$

If we assign nominal values to each of the variables above, each term is unity. We use unity as the factor by which we rescale this equation.

One more example is eqn 3 as we rewrote it to linearize it just above:

$$\bar{\alpha} y_1 = \alpha_1 x_1 \qquad (3')$$

Relative volatilities tend to be near to unity (say, from 1/3 to 3). We can set the nominal value for them to 1.0. The nominal value for mole fractions is also 1.0. The terms will also evaluate to unity if nominal values are used for all the variables. Thus we rescale the equation by dividing it by 1.0.

Homework 3

1. Develop appropriate scaling for the rest of variables and equations for the two component flash model. Guess the original point to be the values determined above. Set up the Newton equations and solve the linear equations that result for the next guess for all the variables.

2. Repeat for the variables and equations for the cylindrical vessel defined in problem 1 (Homework 1, problem 1; Homework 2, problem 2).

3. And repeat again for the closed vessel contain water and water vapor (Homework 1, problem 2; Homework 2, problem 3).

4. Prove that the solution changes depending on the pivot sequence for the problem when all arithmetic is done to 4 significant digits with rounding. Show that the solution does not depend on the 2 on the right-hand-side if one pivots using the smaller numbers.

$$\begin{bmatrix} 1 & 10^{-6} \\ 10^{-6} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Solving when derivatives are not available explicitly

Generalized secant

Assume we are solving

$$\underline{h}(\underline{x}) = \underline{0}, \text{ n equations in } n \text{ unknowns}$$

by guessing $\underline{x}$ at several points; i.e., we compute

$\underline{x}$	$\underline{h}(\underline{x})$
$\underline{x}_0$	$\underline{h}(\underline{x}_0)$
$\underline{x}_1$	$\underline{h}(\underline{x}_1)$
$\underline{x}_n$	$\underline{h}(\underline{x}_n)$

where each $\underline{h}(\underline{x}_i) \neq 0$ or else we would have solved the equations.

We assume

$$\underline{h} = \underline{\underline{A}}\,\underline{x} + \underline{b}$$

Then for each i, we can write

$$\Delta\underline{h}_i = \underline{h}_i - \underline{h}_{i-1} = \underline{\underline{A}}(\underline{x}_i - \underline{x}_{i-1})$$

giving

$$\left[\Delta\underline{h}_1, \Delta\underline{h}_2, \cdots, \Delta\underline{h}_n\right] = \underline{\underline{A}}\left[\Delta\underline{x}_1, \Delta\underline{x}_2, \cdots, \Delta\underline{x}_n\right]$$

which we can write in vector/matrix form as

$$\underline{\underline{\Delta H}} = \underline{\underline{A}}\,\underline{\underline{\Delta X}}$$

where we can solve for the coefficient matrix

$$\underline{\underline{A}} = \underline{\underline{\Delta H}}\,\underline{\underline{\Delta X}}^{-1}$$

Since we choose the steps we can create a matrix $\underline{\underline{\Delta X}}$ which has an inverse. We compute the vector of constants $\underline{b}$ as follows

$$\underline{b} = \underline{h}_n - \underline{\underline{A}}\,\underline{x}_n$$

Finally, we want to choose the next value for $\underline{x}$ such that

$$\underline{h}^* \approx \underline{\underline{A}}\,\underline{x}^* + \underline{b} = \underline{0}$$

Solving, we get

$$\underline{x}^* = -\underline{\underline{A}}^{-1}\underline{b} = -\underline{\underline{A}}^{-1}(\underline{h}_n - \underline{\underline{A}}\,\underline{x}_n) = \underline{x}_n - \underline{\underline{A}}^{-1}\underline{h}_n$$

or

$$\Delta\underline{x}^* \equiv \underline{x}^* - \underline{x}_n = -\underline{\underline{A}}^{-1}\underline{h}_n = \underline{\underline{\Delta X}}\,\underline{\underline{\Delta H}}^{-1}\underline{h}_n$$

One proceeds when applying this method to replace a previous $\underline{x}$ with this new one. Usually one replaces the oldest vector, but sometimes this replacement will lead to a singular matrix $\underline{\underline{\Delta H}}$ so we must then replace a different previous vector.

Since we are replacing only one vector in this matrix by another, one often uses rank one updates methods to maintain the inverse for the matrix $\underline{\underline{\Delta H}}$.

Rank one updating

We can derive the Sherman-Morris/Householder formula for rank 1 updating by the following approach. Assume we have have the following matrix of four matrices where those matrices on the diagonal are square. We append an appropriately partitioned identity matrix. If we then do a block gaussian elimination on this matrix and also carry out the same operations on the appended identity matrix, we convert the identity matrix into the inverse of the original matrix.

$$\begin{bmatrix} \underline{\underline{A}} & \underline{\underline{B}} \\ \underline{\underline{C}} & \underline{\underline{D}} \end{bmatrix} \begin{bmatrix} \underline{\underline{I}} & 0 \\ 0 & \underline{\underline{I}} \end{bmatrix} \quad \begin{matrix} r_1 \\ r_2 \end{matrix}$$

Step 1 of a block gaussian elimination (we suppress underlining of matrices in the following)

$$\begin{matrix} I & A^{-1}B & A^{-1} & 0 & r_1' = A^{-1}r_1 \\ 0 & D - CA^{-1}B & -CA^{-1} & I & r_2' = r_2 - Cr' \end{matrix}$$

Step 2 of the above gaussian elimination

$$\begin{array}{cccc} I & 0 & A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ 0 & I & -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{array} \quad \begin{array}{l} r_1'' = r_1' - A^{-1}Br'' \\ r_2'' = (D - CA^{-1}B)^{-1} r_2' \end{array}$$

Let us repeat the above elimination process, but this time we shall start by pivoting in the second row, second column first. We might expect that we should get the identical result, but we do not.

$$\begin{array}{cccc} A - BD^{-1}C & 0 & I & -BD^{-1} \\ D^{-1}C & I & 0 & D^{-1} \end{array} \quad \begin{array}{l} r_1' = r_1 - Br_2' \\ r_2' = D^{-1}r_2 \end{array}$$

Continuing with step 2 of the elimination, we get

$$\begin{array}{cccc} I & 0 & (A - BD^{-1}C)^{-1} & -(A - BD^{-1}C)^{-1}BD^{-1} \\ 0 & I & -D^{-1}C(A - BD^{-1}C)^{-1} & D^{-1} - D^{-1}C(A - BD^{-1}C)^{-1}BD^{-1} \end{array} \quad \begin{array}{l} r_1'' = (A - BD^{-1}C)^{-1} r_1' \\ r_2'' = r_2' - D^{-1}Cr_1'' \end{array}$$

If we examine the two different forms for the element that is sitting in the first row, first column of the inverse for the above two cases, we see their form is very different. However, they must represent the same thing. We have derived the following equality (known as the Sherman-Morris/Householder formula).

$$(A - BD^{-1}C)^{-1} = A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1}$$

We note that the matrix on the left-hand-side is the inverse of the matrix A modified by subtracting a term from it. The right-hand-side has the inverse for $\underline{\underline{A}}$ with a term added to it. Let us consider the special case which leads to a rank one update formula.

Let $\underline{\underline{B}}=\underline{u}$ a column vector
$\underline{\underline{C}}=\underline{v}^T$ a row vector
$D=1$ a scalar

which corresponds to the matrix

$$\begin{bmatrix} A & u \\ v^T & 1 \end{bmatrix}$$

for which we get the following rank one update formula.

$$(\underline{\underline{A}} - \underline{u}\,\underline{v}^T)^{-1} = \underline{\underline{A}}^{-1} + \frac{\underline{\underline{A}}^{-1}\underline{u}\,\underline{v}^T\,\underline{\underline{A}}^{-1}}{1 - \underline{v}^T\,\underline{\underline{A}}^{-1}\underline{u}}$$

We note that $\underline{\underline{A}}$ is modified by adding the outer product of two vectors. An outer product produces an *nxn* matrix which is at most of rank one - thus the name for the formula. The inverse embedded within the second term of the Sherman-Morris/Householder formula for this case is a scalar. As such we can factor it out and put it into the denominator, as we have done here.

Example

Suppose we wish to find the inverse of a slightly modified identity matrix where we find a one in the first column of the second row.

$$\underline{\underline{A}}_{\text{mod}} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - (-1)\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} = \underline{\underline{A}} - \underline{u}\,\underline{v}^T$$

We show how this matrix can be related to the identity matrix by adding in a rank 1 update. The one in the second row of the column vector establishes which row we are modifying while the one in the first column of the row vector establishes the column. Using the rank one update formula above (noting that the inverse of the identity matrix is the identity matrix), we get

$$\underline{\underline{A}}^{-1}_{\text{mod}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \frac{(-1)\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}}{1 + \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Homework 4

1. Develop a rank two update formula. Describe what it says in words and then find the inverse to the following matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 3 & 1 \end{bmatrix}.$$

Broyden's method

Another method to converge our equations is to update the coefficient matrix for the Generalized secant method by using a rank one update that satisfies only the latest computed point while making the least change to this matrix. In other words we want $\underline{\underline{A}}_{k+1}$ to satisfy

$$\underline{\underline{A}}_{k+1}\Delta\underline{x}_k = \Delta\underline{h}_k$$

while also satisfying

$$\underline{\underline{A}}_{k+1} = \underline{\underline{A}}_k + \underline{u}_k\,\underline{v}_k^T$$

$\underline{u}_k$ and $\underline{v}_k$ are vectors. Note, the second term on the right-hand-side is a vector outer product resulting in an nxn matrix.

We want to minimize the change to $\underline{\underline{A}}$, i.e., we wish to minimize the norm of the change. This norm is bounded as follows.

$$\left\| \underline{u}_k \underline{v}_k^T \right\|_2 \le \left\| \underline{u}_k \right\|_2 \left\| \underline{v}_k \right\|_2 = \left[(\underline{u}^T \underline{u})(\underline{v}^T \underline{v}) \right]^{1/2}$$

Carrying out this optimization leads one to the following algorithm.

Algorithm

1. Have $\underline{\underline{A}}_k$ to start (for example, start with $\underline{\underline{A}}_0 = \underline{\underline{I}}$ or estimate $\underline{\underline{A}}_0$ using numerical perturbation).

2. Compute the next $\Delta \underline{x}_k$ from

$$\Delta \underline{x}_k = \underline{\underline{A}}_k^{-1} \underline{h}_{k-1}$$

3. Compute $\underline{h}_k \equiv \underline{h}(\underline{x}_{k-1} + \Delta \underline{x}_k)$. If $\left\| \underline{h}_k \right\|$ small enough, stop.

4. Estimate

$$\underline{\underline{A}}_{k+1} = \underline{\underline{A}}_k + \underline{u}_k \underline{v}_k^T$$

where

$$\underline{u}_k = \frac{\Delta \underline{h}_k - \underline{\underline{A}}_k \Delta \underline{x}_k}{\Delta \underline{x}_k^T \Delta \underline{x}_k}$$

Note that $\underline{u}_k$ is proportional to the error in using the existing matrix A to $\Delta \underline{h}_k$.

and

$$\underline{v}_k = \Delta \underline{x}_k$$

Note that this vector is in the direction of the step just taken.

5. Increment k to k+1 and repeat from step 2 .

Good and poor model formulations

Earlier, when developing the model for a binary flash unit, we wondered if we should perhaps reformulate the equilibrium relationships in terms of molar flows rather than in terms of mole fractions. We could substitute the definitions of mole fractions in terms of component molar flows into equilibrium eqns 3' and 4' to get

$$\bar{\alpha} v_1 (l_1 + l_2) = \alpha_1 l_1 (v_1 + v_2); \quad \bar{\alpha} v_2 (l_1 + l_2) = \alpha_2 l_2 (v_1 + v_2)$$

If we do this, we can then eliminate completely any mention of mole fraction in our model. Is this a good idea?

It is NOT a good idea for the following reason. If we look closely at these last two equations, we see that they can be satisfied at two spurious points which are not solutions for the original equations. If all the vapor flows are zero, they are satisfied; similarly if all liquid flows are zero, they are also satisfied. These roots are spurious roots introduced by the way we stated the equilibrium equations. Superficially we do not seem to have altered the model. In fact we have in a very subtle way. In doing so we have introduced two spurious roots.

One lesson we can draw here is that these things happen when modeling. In other words, modeling can be a treacherous activity.

However, there is another lesson to be extracted from this example that is particular to expressing equilibrium between phases. Stating equilibrium between phases is the writing of relationships among intensive variables. It should not be expressed in terms of the molar amounts of the species which are extensive variables. To explain, consider the process of cooling a mixture such that it passes from the vapor phase through the two phase region to the liquid phase. In particular, consider that last bubble of vapor disappearing just as the mixture becomes all liquid. The amount of material in the vapor phase goes to zero. The mole fractions characterizing that bubble do not. Mole fractions are the limits in this case of a ratio of molar flows, all of which are heading to zero. Mole fractions are well defined numbers. Equilibrium equations involving them should never be written in terms of the molar amounts of the species as the equations lose their meaning in this limiting process.

> Modeling rule: Express thermodynamic equilibrium in terms of mole fractions and not in terms of molar amounts.

Modular versus equation-based approaches to modeling

In the modeling done above, we captured the model in two distinct forms. In one we presented the model as the equations defining it, that is, as eqns 1 to 8. In the second, we presented the model as the equations as well as the steps to solve it (used above to discover if the model formulation is obviously defective). The form we need to prepare of course depends on the environment we intend to use to solve the model. The *equation solving* approach separates the equations from solving them. The *sequential modular* approach to modeling, which is used in most flowsheeting systems (such as Aspen or Pro/II), falls into this latter class of approaches. In the modular approach, the model is in the form of a subroutine that will provide a solution to a model given values for a prescribed set of variables. The equations are embedded into the code within the subroutine. Since the equations and the code to solve them are intertwined, it is quite difficult to retrieve the underlying equations from the code at a later date.

In the equation-solving approach, it is assumed that the modeling environment provides the capability to solve any model we give to it. It is easy to solve the simple model we just developed; virtually any equation-solving environment will have little difficulty in doing so. Unfortunately, many models are less well-behaved in this respect. Large interconnected models, such as those for a distillation column or a multieffect evaporative process, can be very difficult to solve; they require assistance from the modeler to get to a solution.

We can list a number of relative advantages and disadvantages for these two approaches to modeling.

- The equation-solving approach requires only the statement of the equations to define the model, an activity which is about 20% of the effort to define the equations and then write a subroutine that can solve them as required by the modular approach.

- Given the equations, the use of the model for many different purposes is straight forward. For example, it is easy to convert an equational-based model so it can be used within an optimization environment even though it was originally developed to be solved as "square" set of equations. It is not as easy to move a modular model from a solving-only environment to an optimization one.

To write the code for a modular model, the modeler has to decide which variables are to be fixed and, by default, which are then to be computed from the model. For example, in a traditional sequential modular flowsheeting model for a unit operation, the system assumes the modeler has selected to fix all the feed streams into the model along with a sufficient set of unit parameters to fix the performance of the unit. The subroutine then must compute the output streams from the unit. Preselecting which variables must be fixed has the following advantage and disadvantage for anyone subsequently using such a model.

- The user of the model has *does not have to select* which variables to fix for the model to be well posed.

- The user of the model *cannot select* variables to be fixed other than those selected by the person creating the model.

Selecting variables to fix is a difficult task, especially for a large model. Imagine looking at 5000 equations and trying to discover that last variable to fix (or the one too many that is fixed and should not be). It is a task that has caused many to dislike the equation-solving approach to modeling. We deduce the following requirement for any effective equation-solving environment:

> Modeling environment requirement: A modeling environment must allow a modeler to aid a user of a model to select a legitimate set of variables to be fixed.

Finally, there is the following real advantage for the modular approach.

- To solve really tough models requires more than just a statement of the equations from the modeler. Also needed are good initial guesses (often obtained by sneaking up on the solution), proper scaling and so forth. The modular approach allows the modeler considerable flexibility in sneaking up on a solution and in scaling, with perhaps 90% of the code in such a model being to get close enough to the solution that the nonlinear solving methods can converge.

The last advantage that is attributed to the modular approach cannot be lost in an equation-based approach. We suggest, therefore, the following requirement in the creation of such systems.

> Modeling environment requirement: An equation-based approach must provide the modeler with the means to aid a subsequent user in the process of getting a good initial guess for the solution.

Declarative vs. procedural parts of a model

In the equation-based approach to modeling, one declares the variables and the equations that hold among them. These form the declarative part of a model. Setting the initial guesses for the variables can be either declarative or procedural. Sneaking up on the solution as a precursor to solving is almost certainly procedural. If the above guideline is valid, an equational-based model must have both parts in its statement.

Modeling problems

We shall spend some time examining many of the modeling problems exposed in the above example. Our main thesis is that modeling is really difficult. No one should underestimate the problems involved. The first issue we consider is that of not writing redundant equations.

Writing redundant equations

One of the very difficult issues in modeling is determining if one has written a redundant equation. We shall illustrate how easy this is to do by showing two rather subtle examples. We have no general method to cure this problem. Our goal here is simply to show how easy it is to do.

Simple stream splitter problem

The first example we shall show for creating redundant equations is in the modeling of a simple stream splitter. This model appears to be very straight forward, but virtually everyone modeling it has, at some time, written too many equations in its formulation and been confounded in trying to find out why.

Let us create the model for the splitter shown in Fig. 17. It has one feed stream and n_{out} output streams. Each stream is characterized by the following variables.

F_{tot}	total molar flowrate
f_i, $i=1..n_{comp}$	molar flows for each of the components
y_i, $i=1..n_{comp}$	mole fraction for each of the components

The equations we can very naturally write for a stream defined in terms of these variables are the following.

$$F_{tot} = \sum_{i=1}^{n_{comp}} f_i$$

$$f_i = y_i F_{tot} \quad \text{for } i = 1 \cdots n_{comp}$$

By adding up the second set of equation over all components and comparing to the first, we discover that we can derive that the mole fractions add to unity. We must not write this as an independent equation.

We have $2n_{comp}+1$ variables and $n_{comp}+1$ equations for each stream. Thus a simple stream such as this has n_{comp} degrees of freedom for it. If we fix the molar flowrates for each of the species, we know that we have completely specified this stream.

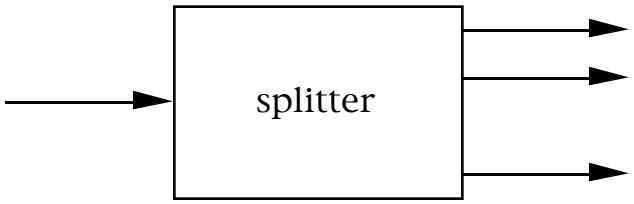


Fig. 17. Simple stream splitter

(A way to think about these streams, then, is to allow ourselves to have all these variables for them, knowing we shall write the above equations in terms of these variables and leave ourselves with n_{comp} degrees of freedom for it.)

There are n_{out} streams leaving and one stream entering the splitter. We can model the splitter by equating all the composition variables in all the streams:

$$s_{in} \cdot y_i = s_{out_j} \cdot y_i \quad \text{for } i = 1 \cdots n_{comp},\ j = 1 \cdots n_{out}$$

where $s_{in}.y_i$ is a composition variable for the inlet stream, etc. Note that we label the stream variables by placing the stream identifier as a qualifier at the beginning of the variable names.

We also need to write the following equations to describe how the inlet stream splits.

$$s_{out_j}.F_{tot} = \sigma_j * s_{in}.F_{tot} \quad \text{for } j = 1 \cdots n_{out}$$

where the splits σ_j must sum to unity:

$$\sum_{j=1}^{n_{out}} \sigma_j = 1$$

Our intuition tells us that we can specify all the splits except for one (the last is 1 less the sum of the others). We should also be able to fix the feed stream; namely, we can specify the molar flows for each of the components in it.

If we write all these equations and make these specifications, we will discover to our chagrin that the problem is overspecified. It isn't just a little overspecified; rather, it has n_{out} too many specifications or there are n_{out} too many equations written or some combination of both these problems.

What went wrong? This number seems surprisingly large.

For each stream we can derive the equation that its mole fractions add to unity, as we noted above. Equating all the mole fractions means that we need this equation only one time, but it exists $n_{out} + 1$ times. It is implied n_{out} too many times.

How do we cure this problem? One possibility is that we equate all but one of the mole fractions for the streams. Another is that we eliminate the equation relating F_{tot} to the molar flows of the components for all the output streams. This last option is not a preferred one as this equation belongs to our concept of a stream, a concept we wish to keep intact. Thus we can adopt the first cure. The real point to be made here is that we wrote far too many equations and hardly noticed we did it when modeling. We have seen modelers searching for hours to find this error (and sometimes we have been those modelers).

The concept of a state

Another way to cure the problem for the splitter is to rethink the basic modeling concepts we use for streams. Motivated by this example, let us define the concept of the state of a stream as all the variables and all equations we can write among them that define its *intensive* characteristics. In this case we define the state in terms of

y_i, $i=1..n_{comp}$ mole fraction for each of the components

and the relationship that these add to unity:

$$\sum_{i=1}^{n_{comp}} y_i = 1$$

A stream is then a state and a flowrate. We can add the variable

F_{tot} total molar flowrate

to complete our stream definition. For convenience we can add the individual flows for the components

f_i, $i=1..n_{comp}$ molar flows for each of the components

and an equal number of equations to define them

$$f_i = y_i F_{tot} \quad \text{for } i = 1 \cdots n_{comp}$$

On closer examination we note that we cannot add that F_{tot} is the sum of the individual flows for each of the components since we can derive this equation from those already written.

It would seem we have done almost exactly what we did before, but there is a subtle difference which we shall now exploit when developing a splitter model. This time we write our model to say that the states for all the streams are equal; thus there is only one state with only one equation that says the mole fractions add to unity. We augment this model with the equations

$$s_{out_j}.F_{tot} = \sigma_j * s_{in}.F_{tot} \quad \text{for } j = 1 \cdots n_{out}$$

$$\sum_{j=1}^{n_{out}} \sigma_j = 1$$

We now get no error in counting the degrees of freedom.

What we have done is combine variables and equations into the concept of a state. We extended out thinking beyond just equating variables; the equation characterizing the state must be dealt with when making the states equal. We recognize it should only be generated once.

The notion of a state carries over to the holdup within vessel which we need when we develop dynamic models. A holdup is a state and an amount {kg_mole} while a stream is a state and a flowrate {kg_mole/s}. We can equate the state of a holdup to that of the stream leaving the vessel when developing a dynamic model. Thinking in this manner should reduce some subtle modeling errors arising from writing redundant equations as we just did above.

This lesson is sufficiently important that we should make it into a modeling rule.

Stream modeling rule: A stream should always have its state as a single concept within it. Its *state* contains all its intensive variables and all the equations that we can write among them. A stream is then a state and a total flowrate.

Totally recycling component problem

Another diabolical example of writing too many equations occurs when modeling a refrigeration cycle where the refrigerant totally recycles in the process. None is lost nor is any added to the cycle. In this problem we write exactly the right set of equations for each of the unit models. It is the wiring together of them that gets us into trouble. Fig. 18 is a diagram for a refrigeration process.

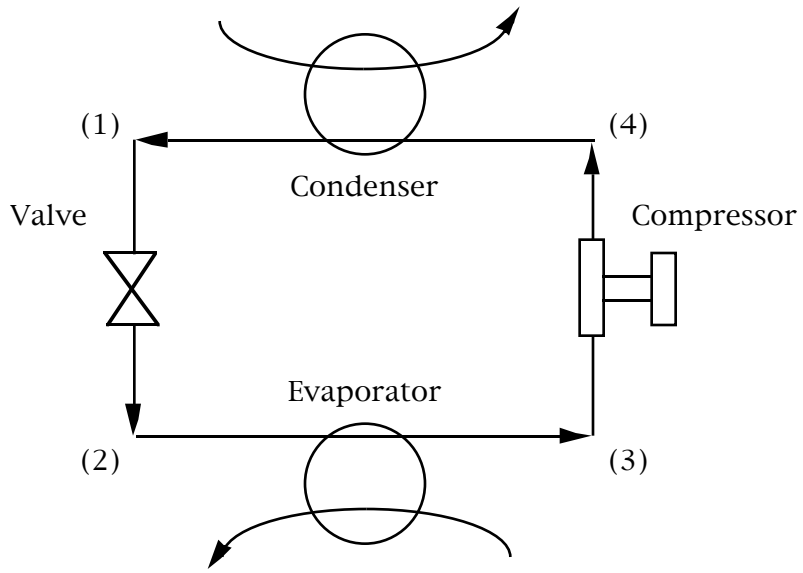


Fig. 18. Refrigeration process

The equations we write include

$$F_{cond,in} = F_{cond,out} \overset{1}{(=)} F_{valve,in} = F_{valve,out} \overset{2}{(=)}$$

$$F_{evap,in} = F_{evap,out} \overset{3}{(=)} F_{comp,in} = F_{comp,out} \overset{4}{(=)} F_{cond,in}$$

There are four material balances, one for each unit, and four connection equations to wire the model together. These lead to a statement that the first flow is equal to the last which is, unfortunately, the first flow again. We have one too many equations here.

So how do we cure this one? We must not write one of these equations. We can suppress one of the connection equations which is hardly an elegant answer but an expedient one.

One might argue that this problem does not occur for a traditional flowsheeting system. Unfortunately it does occur, but it is easily not noticed. A conventional flowsheeting system would tell the modeler who put this flowsheet together that it has a recycle stream in it, which it does. The way to solve a flowsheet with a recycle is to "tear" the recycle by guessing one of the streams, say the one marked with a (1) in Fig. 18. What will happen is that, no matter what flow we guess, that flow satisfies the model. Our problem in this case is not that the model does not solve but rather that we will fail to recognize the flow is a degree of freedom which we are free to fix.

Totally recycling species in a flowsheet will always give rise to this problem, one where the configuration of the model from well-posed parts leads to a redundant material balance equation. If we totally recycle "pressure" or "heat," we can encounter a similar problem.

Homework 5

1. Describe two other examples of processes where one or more components are completely recycled. Discuss the solving of models for these processes using a conventional flowsheeting system.

2. Develop a model for a mixer where the streams are described by a state and a total flowrate.

3. Extend the model for a stream by adding mass fractions and total mass flow. Assume the existence of molecular weights for each of the components. Test out your model on paper by using it to replace the stream model in the stream splitter example earlier. Do you have this model right (probably not)?

Avoiding divides by zero

The form in which one writes an equation can affect its behavior when one is solving it numerically. Consider, for example, solving the following simple equation numerically

$$f(T) = 10 - e^{3/T} = 0$$

for T using Newton's method. Analytically the solution is

$$3/T = \ln 10$$

or

$$T = 3/\ln 10 = 1.303$$

Applying Newton's method

$$\left[\frac{\partial f}{\partial T}\right]_T \Delta T = e^{3/T}(-3/T^2) = -f(T) = -(10 - e^{3/T})$$

or

$$\Delta T = (10 - e^{3/T})T^2 / 3e^{3/T}$$

and guessing 0 < T < 2.115 will give convergence. Otherwise T goes negative and diverges to negative infinity. The plot of f(T) vs T in Fig. 19 makes this behavior evident.

Let us make the following minor change to the form of the problem statement. Defining *x* as

$$x = 3/T$$

we set up the following equivalent problem to solve

$$f_1(x,T) = 10 - e^x = 0$$

$$f_2(x,T) = Tx - 3 = 0$$

which solves from any initial guess for *x* and *T*.

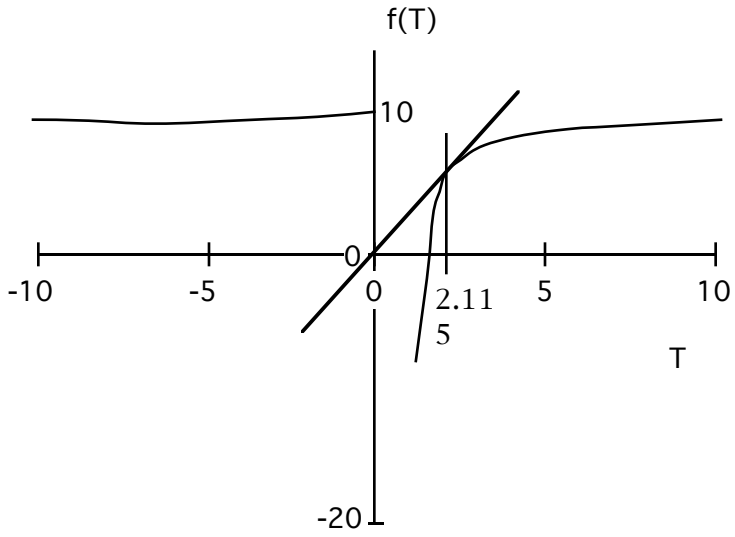


Fig. 19. Plot of $10 - e^{3/T}$ *vs*. *T*

What happened? The second form avoids the divide by zero that is in the first form and which causes the "pole" at T=0 that appears in Fig. 19 - i.e., the

discontinuity in the function *f(t)*. A plot of the transformed functions $f_1(x,T)$ and $f_2(x,T)$ vs x and T shows monotone behavior and is easy for the Newton method to solve.

Consider a second example.

$$f(x) = \ln(x) - 5 = 0$$

Here the function is defined only for x strictly greater than zero. If we form the Newton equations for this function, we get:

$$\left[\frac{\partial f}{\partial x}\right]_x \Delta x = \frac{1}{x}\Delta x = -f(x) = -(\ln(x) - 5)$$

The partial derivative has a divide by zero when x reaches zero. If one expects x to solve at values near to zero, this function is dangerous numerically. Consider the following transformation.

Let

$$y = \ln(x)$$

and solve the following two equations instead of the original one

$$x - e^y = 0$$

$$y - 5 = 0$$

These two equations are well behaved and are easy to solve using the Newton's method.

We derive from these simple examples a rule we use throughout all of our models when we wish to solve them numerically.

Modeling rule: Avoid the divide operator in both the model equations and in the forming of the partial derivatives needed by the Newton method if the divisor has any chance of approaching zero.

Homework 6

1. Convert the following functions so that there is no divide operator in the functions and their Newton equations.

a. $$f_1(x,y,z) = \frac{\exp(\frac{x}{y^2 - 5})}{z} + 7 = 0$$

b. $f_2(n,p) = (\sqrt{n} - 47)/p = 0$

c. $f_3(z,t) = \frac{5\ln(1/z)}{t^2} + \frac{2z}{t} + \frac{3z^2}{t^{1.5}} = 0$

Determining number of degrees of freedom

There are two theorems that deal with determining the degrees of freedom when equilibrium is assumed. One is the phase rule that tells us how many degrees of freedom there can be among the intensive variables characterizing the various phases. There is another less well know theorem by Duhem (his other theorem) that tells us the degrees of freedom when we also know the quantity of the material with which we start. Thus it deals with extensive variables, too. The theorem says (Prigogine and Defay, 1954)

> *"Whatever the number of phases, of components or of chemical reactions, the equilibrium state of a closed system, for which we know the total initial masses of each component, is completely determined by two independent variables."*

An intuitive proof is to collect a kilogram of the material, fix the temperature and volume and let it sit for an infinite amount of time until it comes to equilibrium. We know the original amounts of each of the components in the material. The final temperature and volume are the two independent variables we set. Equilibrium could be any number of phases and after the result of any number of reactions.

Equilibrium stream

A direct corollary of this theorem is that an *equilibrium* stream has precisely $n_{comp}+2$ degrees of freedom. Thus if a stream is assumed to be an equilibrium stream, we completely fix it if we specify the amounts of the species in it (originally) and its temperature (or, better, its enthalpy) and its pressure.

We first characterize the state of a stream using the following variables:

y_i, i= 1, 2, .. n_{comp}	mole fractions for each component
T	temperature
P	pressure
H	enthalpy per mole
S	entropy per mole
G	Gibbs free energy per mole

and then add the flow variables

f_i, i= 1, 2, .. n_{comp}	molar flows for each component
$Ftot$	total molar flow

What this corollary says is that only $n_{comp}+2$ of these variables are independent. This allows us to think in the following manner.

> Declare a stream to have $n_{comp}+2$ degrees of freedom and assume that *any* variable which describes it is then available.

If we know the composition, temperature and pressure of a stream, we know (in principle) all its intensive molar properties. We can make $(n_{comp}-1)+2$ specifications here. To add the extensive properties, we need only specify the total flow at which point we can compute all of its extensive properties (total enthalpy flow {kJ/s}, total entropy flow {kJ/(s*K)}, etc.). The total degrees of freedom is again $n_{comp}+2$

Equilibrium flash unit

We can use these ideas to fix the degrees of freedom for a flash unit. We do our accounting in Table 4.

Table 4. Determining the degrees of freedom for an equilibrium flash unit

Item	Net new equations	Net new variables
3 streams		$3(n_{comp}+2)$
One material balance/component	n_{comp}	
Heat balance with heat input Q_{in}	*1*	*1*
Equilibrium: equal fugacities for each component in product streams, equal temperature and pressure	$n_{comp}+2$	
Specification of entire feed	$n_{comp}+2$	
Net degrees of freedom (subtract net new equations from net new variables)		*2*

This accounting, if correct, says we have two degrees of freedom for the flash unit. We can specify its temperature and pressure or its pressure and the heat input, for example. The former is called an isothermal flash calculation while the latter is a variation of an adiabatic flash calculation (an adiabatic flash computation assumes the heat input is exactly zero).

Simple stream splitter

Let's repeat for a stream splitter. Table 5 organizes our calculations. The result indicates that we can specify the feed and all but one of the splits. This result agrees with our intuition.

We can use the final line in a table such as this to summarize the effect of including the type of concept the table represents.

Stream mixer

Table 6 summarizes the degrees of freedom analysis for a stream mixer.

Flowsheet

The sequential modular mind-set

If we have analyzed all the units that are used to construct a complex model, can we then get the degrees of freedom right for the overall model? The answer is that often we can. We shall use a complex flowsheet as our example. The ideas extend to other types of complex models.

Table 5. Determining degrees of freedom for a stream splitter

Item	Net new equations	Net new variables
$n_{out}+1$ streams		$(n_{out}+1)*(n_{comp}+2)$
States are all equal (T, P, and all compositions)	$n_{out}*(n_{comp}+2)$	
Removal of all but one equation summing compositions to unity	$-n_{out}$	
Split on flows (n_{out} new split variables and n_{out} equations)	n_{out}	n_{out}
Splits add to unity	1	
Specification of entire feed	$n_{comp}+2$	
Net degrees of freedom (subtract net new equations from net new variables)		$n_{out}-1$

To handle a flowsheet, we can adopt a sequential modular mind-set to our thinking in order to develop the degrees of freedom correctly. We will discover the degrees of freedom first for all the parts with the following restriction

- Discover the degrees of freedom for a unit where the feeds to the unit are fixed.
- Wire together the flowsheet.

- Count the number of totally recycling components (neither enter nor leave the flowsheet)
- The degrees of freedom are those for each of the units (assuming the feeds were fixed in doing the counting) plus those required to fix the feeds to the entire flowsheet less one for each recycling component.

Table 6. Determining degrees of freedom for a stream mixer

Item	Net new equations	Net new variables
$n_{in}+1$ streams		$(n_{in}+1)*$ $(n_{comp}+2)$
Specification of all feeds	$n_{in}*$ $(n_{comp}+2)$	
One material balance/component	n_{comp}	
Energy balance	*1*	
Specify outlet pressure	*1*	
Net degrees of freedom (subtract net new equations from net new variables)		*0*

Let us apply these ideas to the configuration in Fig. 20. The structure is that of a five tray distillation column. We shall use the table format as we did for the previous examples (see Table 7).

For a column we argue intuitively that we are able to specify the following:

- Total number of trays
- Feed tray location
- Feed
- Reflux ratio
- Distillate top product flow
- Pressure of the condenser

Two of these are fixed for the column in Fig. 20, namely, the total number of trays and the feed tray location. That leaves us with three specifications plus that required to set the feed. Our intuition and this analysis agree.

Spreadsheet mind-set

There is an entirely different mind set one can adopt to get the degrees of freedom correct for a complex computation. We can call it the *spreadsheet*

mind-set. In this approach one lists the variables one is willing to specify. Then one adds an equation at a time to the model, with each equation introducing one new variable to the problem which is immediately computed. This is the style of programming one uses when programming a spreadsheet. There is immediate checking that the equation adds information to the problem. Also one gets to look at the numbers produced to see that they make sense.

When one adds an equation that introduces two variables which have not yet been computed, either one must immediately select one to be specified or one deliberately introduces a variable to be guessed and subsequently iterated. We shall not illustrate this mind-set here but simply note it. It works very well on a number of problems, a statement borne out by the effectiveness of spreadsheet modeling.

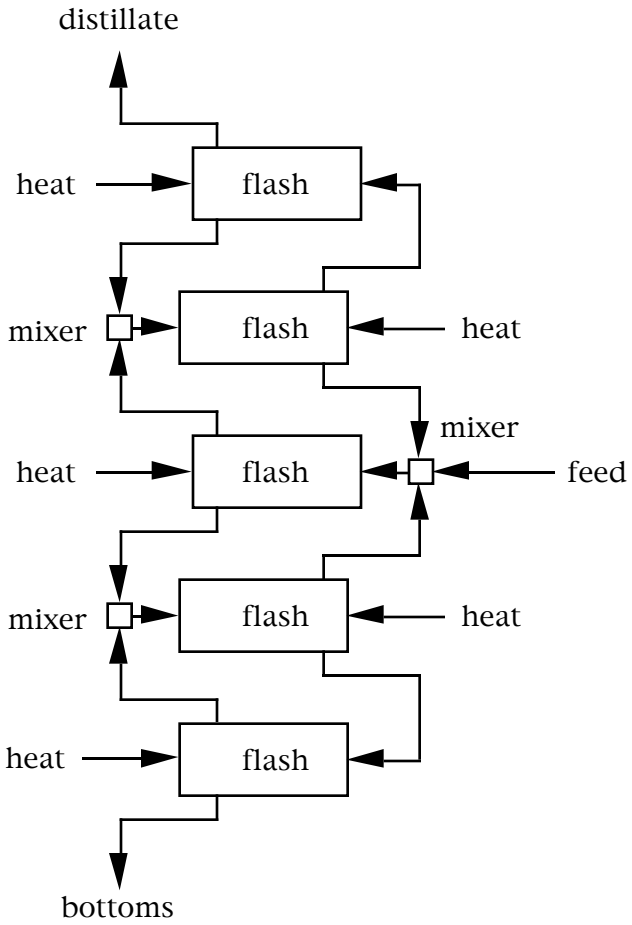


Fig. 20. Distillation column built of flash and mixer units

An aid for selecting the set of fixed variables

Let us suppose we have developed a model and believe it to be a correct one in the sense that it contains no redundant equations nor is it missing any. We wish to solve the model. Assume the model comprises *n* equations in *n+m* variables, where *m* is greater than zero, a situation which is always true for engineering models. A first task is to make the model *square*, i.e., to select a set of *m* variables which are to be fixed when solving. A very effective aid exists which can help one in this activity. Based on the structure of the equations, it is able to report to a modeler which variables are no longer eligible to be fixed. The modeler picks one which is eligible. The eligible set changes as another variable is fixed, requiring that the modeler use the aid again to pick the next one. This aid does not detect poor choices based on the numerical behavior of the model; none-the-less, this aid is very useful. We will discuss how to discover and overcome numerical problems later.

Table 7. Degrees of freedom for the model configured from parts (Fig. 20)

Item	Net new equations	Net new variables
5 flash units @ *2* each		*10*
3 mixer units @ *0* each	*0*	*0*
Column feed		$n_{comp}+2$
Pressure drop from tray to tray (for example, assume all pressures are equal)	*4*	
3 adiabatic trays $(Q_{in}=0)$	*3*	
Net degrees of freedom (subtract net new equations from net new variables)		*(ncomp+2) +3*

To explain the aid, let us consider the set of five equations in six variables whose incidence matrix we display in Fig. 21. We show only the structure for them, represented as an incidence matrix. The meaning is that function f_1 has variables x_1, x_2 and x_5 explicitly mentioned in it. Based on the structure of the equations, our goal is to detect which variables are legal candidates yet to selected to be fixed.

	x_1	x_2	x_3	x_4	x_5	x_6
f_1	x	x			x	
f_2			x	x		
f_3	x	x	x			
f_4			x			
f_5	x					x

Fig. 21. Incidence matrix for small example problem

We start by assigning to each equation a variable which appears in it subject to the restriction that we assign a variable to no more than one equation. Such as assignment we call an *output assignment* for the equations. For a small set of equations we can readily accomplish such an assignment by trial and error by hand. In Fig. 22, we place an asterisk (*) to indicate such as assignment for these equations. The assignment is not unique in general. We only need one of the many that might be possible.

	x_1	x_2	x_3	x_4	x_5	x_6
f_1	x*	x			x	
f_2			x	x*		
f_3	x	x*	x			
f_4			x*			
f_5	x					x*

Fig. 22. An output set assignment for equations

Any variable not yet assigned is one that we can now fix (based only on the structure of the equations). Thus we could select x_5. However, there are others possible. We could trade x_5 for x_1 by assigning x_5 to equation f_1 and unassigning x_1. Then x_1 would be the left over variable.

With x_1 unassigned, we see it could be traded for x_2 by making it the output variable for f_3 and while unassigning x_2. Similarly x_1 could be traded for x_6 using eqn. f_5. We see a path being traced from the incidences for the unassigned variables that move horizontally to an assigned incidence (x_5 to x_1 in f_1), then vertically to an unassigned incidence, then to an assigned incidence, etc., until the path terminates which it will always do on an assigned variable. We can call such a path a Steward path as it was first used by Steward (1962) to aid in developing an output assignment for a set of equations.

The ideas behind the following extremely simple and very fast algorithm should now be evident. It is only a matter of seconds of computer time to analyze several thousand equations.

- Assign output variables to equations
- Find all variables *reachable* from unassigned variables by a Steward path using the following steps

  1) mark unassigned variables
  2) mark equations containing marked non-output variables
  3) mark output variables for marked equations
  4) repeat from (2) until no further changes

Applying the above algorithm to our problems yields the results shown in Fig. 23. Step 1 directs us to mark x_5. Step 2 says we should then mark f_1. Step 3

says to mark x_1. Step 4 directs us back to step 2 which says we should mark f_3 and f_5. Step 3 then says to mark x_2 and x_6. Returning to step 4, we can add no marks and algorithm terminates. We can select any one of the marked variables to fix. Note that variables x_3 and x_4 cannot be selected.

To see why, select x_3 as fixed and attempt then to assign an output set assignment. It will not be possible. If an output set assignment is not possible, the equations are guaranteed to be singular in the remaining variables. Since the decision is based on structural considerations, this type of singularity is called a *structural singularity*.

		x1	***x2***	***x3***	***x4***	***x5***	***x6***
		#3				***#!***	
			#3				***#3***
f_1	#2	**x***	**x**			**x**	
f_2				**x**	**x***		
f_3	#2	**x**	**x***	**x**			
f_4				**x***			
f_5	#2	**x**					**x***

Fig. 23. Markings to find eligible variables to be fixed. Variables x_3 and x_4 are not eligible.

Estimating reasonable values for the fixed variables

The next step in the many required to find a solution for our model is to provide reasonable values for the variables we selected to be fixed. The model is parametric in these values. We could have selected to fix the recovery of the light key in the top product for a 10 tray distillation column to be 99% when, with that number of trays, the column cannot provide a recovery exceeding 80% at total reflux conditions.

To find reasonable values is imperative as otherwise the computations just wander, neither converging nor diverging. We can conjecture that the equations are becoming more and more singular as one gets closer and closer to the solution. As such it might be possible to discover this problem by detecting this singularity, a topic we shall return to shortly. If we can discover such a singularity, then we shall show how we can develop an aid to suggest altering our choice of what is fixed and what is computed to stabilize the calculations. Again, appealing to our intuition, such an aid could direct us to fix the reflux ratio in the column and compute the recovery. This latter computation is much more likely to converge. We would like the aid to be general purpose and not one especially tailored to the solving of distillation column models (where its knowledge is more like that in an expert system for columns).

Getting the equations to solve

Natural specifications to creep up on solution

One way to get reasonable values for the fixed variables is to use what we might call natural specifications for the problem first just to get a solution to the equations. In other words, first pick to fix a set of variables for which we can supply reasonable values from which the problem is likely to converge. For a column, fixing the reflux flow is a much safer specification to make than fixing the recovery of the light key component. After converging the column one can then attempt to place a recovery specification, but, to be safe, it should be only a bit tighter than that found when getting the first solution to the column. If the just slightly tighter recovery specification works, then one knows that one can ask for a recovery specification. If it does not, the failure could be telling you that you are not able (from a numerical point of view) to select product recovery as a fixed variable. Tightening it up even more leads either to a successful solution or a recovery that the column calculation cannot reach (suggesting perhaps that neither can the column).

Again playing detective with such a problem, one might return to specifying the reflux flow. A possible move is to increase it slightly, resolve, examine the recovery, increase it again, resolve and so forth. The recovery will approach a limit beyond which the column cannot go.

Solving the model by parts

Another approach to solving complex models is to break them apart and solve the parts first. Then put the parts together two or three at a time, perhaps, resolve, and continue until the whole model is solved. Suppose we wish to solve a flowsheet containing recycles. We could guess the recycles and solve each unit one at a time in a forward sequence through the flowsheet until one has solved all the units. Then one could add the recycle to the flowsheet and attempt a solution from there.

Another case of solving part and then the whole is a strategy to solve a column. One could linearly interpolate all the component flows from the top to the bottom as well as the temperature profile throughout the column. Then one could solve each tray at the given temperature and pressure for fixed input streams to the tray. Allowing the heat into the tray to be away from zero releases the heat balance for the tray. Once one has solved all the trays, one can attempt to solve the entire column.

Solving abstractions of the model and then the detailed model, perhaps iteratively

Another approach is to model a column assuming constant molar overflow on each tray and constant relative volatilities. Such a model, in our experience, always solves rather quickly if one specifies the reflux flow and the distillate top product flow. Then one can perform tray by tray computations fixing for each tray the input flows, the temperature and the pressure using rigorous physical property calculations. From these computations, one can extract new estimates for the relative volatilities and repeat the entire computational

sequence until the relative volatilities do not change from iteration to iteration.

Because we believe that all these approaches should be possible for a model, we add the following requirement for an equational-based modeling environment, which is a stonger statement than one we made earlier about the environment allowing a modeler to aid in establishing initial conditions for a model. We now see it is a much more involved activity than we implied earlier.

Requirement for equational-based modeling environment: The environment must allow a modeler to creep up on the solution in almost any way desired so the modeler can gain experience in solving the model and can then encode this experience in the final model description.

Types of behavior when solving

Equations which fail to solve display many different types of behavior. They may diverge, they may drift very slowly or they may even display what appears to be random cycling. Chaos is one cause for this last behavior. Consider trying to solve the following deceptively simply looking equation using a numerical approach based on successive substitution.

$$x_{k+1} = 4ax_k(1 - x_k)$$

Table 8 indicates the behavior of this function.

Table 8. Cyclic and chaotic numerical behavior of a simple function

x_0	a	cycle values
0.2	0.6	0.583 (converges)
0.2	0.8	0.513, 0.799
0.2	0.87	0.395, 0.831, 0.487. 0.869
0.2	0.8925	0.810, 0.548, 0.884, 0.366, 0.828, 0.506, 0.892, 0.343, 0.804, 0.561, 0.879, 0.380, 0.841, 0.480, 0.890, 0.347
0.2	0.95	chaos, nothing repeats ever

Using a successive substitution method we guess x_0 and use the above equation to compute x_1, then x_2, and so forth until x_{k+1} equals x_k. Depending on the initial guess made and the value of a, we get very different behavior.

Homework 7

1. Show that the values above do occur. Repeat the behavior for 50 to 100 interations. Try to find a point where the cycle is even longer than 16 points but is not yet chaotic.

Numerical singularity

We offer here a recipe for thinking about numerical problems one might encounter in a model. In each case we have a model that will not solve. The following are in the order each should be checked.

Too many slightly poor pivot choices to reduce fill

In solving a set of nonlinear equations numerically using a Newton based method, the inner problem is to solve a set of linearized equations as we have seen above. To solve large problems involving hundreds to a few tens of thousands of equations, one must use sparse matrix methods in the linear equation solver. For numerical stability, one should strive to select the largest pivot possible in the equations yet to be pivoted while solving. However, one wants to do the least work possible by choosing pivots to minimize *fill* - i.e., minimize the number of new nonzero elements the method creates when doing the elimination process to solve them. A nonzero will ultimately require more additions and multiplications to complete the factorizing. There is always a trade that has to be made. How much smaller will one allow the pivot to be than the largest one available so that one can get less fill in the matrix. These codes contain a tolerance often set to something in the range 0.1 or even smaller meaning that the pivot can be 10 or more times smaller than the largest and still be selected.

Some problems have a diabolical behavior. The following is one. Here we compute the location for node points when developing a two dimensional finite element grid such as when analysing the heat flow in an object with a complex shape. We place *4n* grid point all around the edges. We then join the edge grid points as shown in Fig. 24 and write equations so that the location of the x and y coordinates for a point are the average of those of its four neighboring points. To make that clearer, the equations are

$$x_i = 0.25 \sum_{j \in \text{neighbors}} x_j$$

$$y_i = 0.25 \sum_{j \in \text{neighbors}} y_j$$

These all linear equations have coefficients of 1 and -0.25 (when rearranging the equations to equal zero by subtracting the RHS from the LHS). There is no really small coefficient in the set. Trying to solve a problem with about 20 interior node points leads to a very strange behavior when using our very good sparse linear equation solving code. The code is not the issue; the problem is sort of a worst case actor that destroys anyone's sparse code. It very often picks a pivot of 0.25 rather than a large pivot to reduce fill. It solves and reports back an answer. However, substituting the answer into the equations shows they are nowhere near being satisfied. The residual for an equations can be of the order of 10^6 or more. What went wrong? The continued selection of the slightly smaller pivots just ganged up on the solver. Changing the tolerance so it must pick the largest pivot each time completely cures the problem. The problem is not singular.

So the first thing one should do is examine the equation residuals for the linear equation solver. If the errors are large, try changing the selection criterion for the pivots - make it 0.9999 for example.

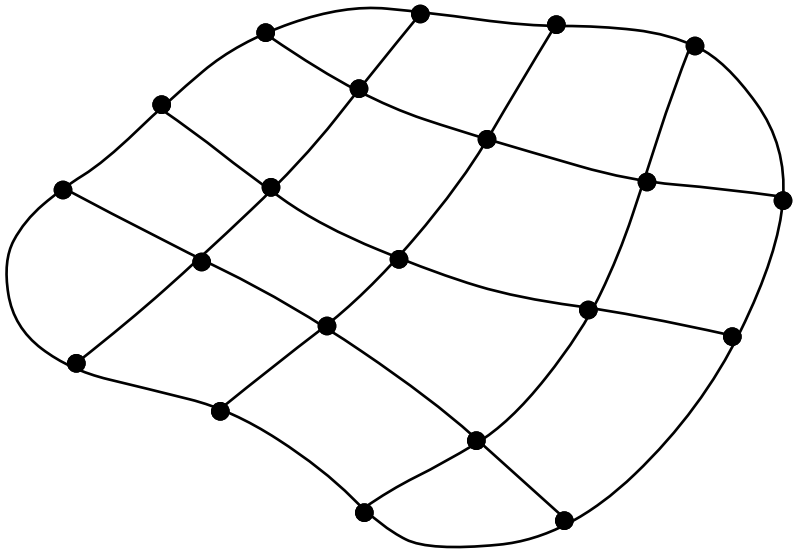


Fig. 24. Computing the locations for the grid points in a two-dimensional finite element mesh

Small pivots when solving

If a problem is ill-conditioned, that ill-conditioning can often manifest itself in the existence of small pivots when attempting to solve the inner linear Newton equations. Fig. 25 illustrates what might happen.

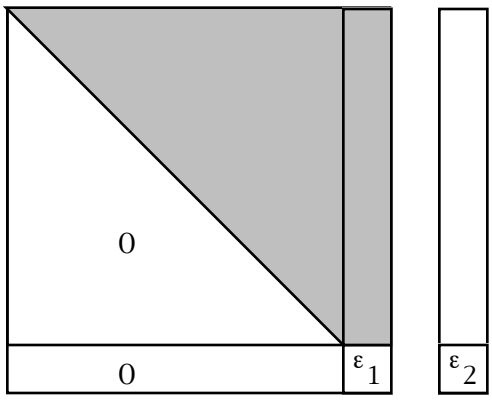


Fig. 25. The last pivot for a set of linear equations during a forward gaussian elimination

In this figure, the last pivot is too small to use. Shown also is the right-hand-side which the method processes along with the coefficient matrix if it is solving a set of linear equations. There exists an ε_2 there. How one interprets what one has depends on its magnitude.

Suppose that ε_1 is of the order of 10^{-14}. Suppose also so is ε_2. If we have used proper scaling, we decide on the magnitude of a number by comparing it to one. These are small numbers, down in the accuracy limits of many computers. The last equation says

$$\varepsilon_1 \Delta x_N = \varepsilon_2 \quad \text{or} \quad \Delta x_N = \frac{\varepsilon_2}{\varepsilon_1}$$

With both numbers being extremely small, one is looking at rather nice random number generator here, albeit one that produces modest sized numbers.

One can show a rather nice interpretation for ε_2. if we do the following:

- remove the last equation from the set
- set Δx_N to zero and solve the previous N-1 equations for the other *N-1* Δx_i, variables.
- substitute these values into the last equation and evaluate its residual

ε_2 is precisely this residual. In other words, if ε_2 is small, this last equation is in fact satisfied. It is locally *dependent* on the other equations.

If it is large, then the last equation is locally *inconsistent* with the other equations. In both cases the extremely small value for ε_1 says the problem is singular. In the one case the equation is dependent and could be redundant while in the latter case it is inconsistent.

How might one cure this problem? Remember that one of our first tasks in preparing to solve these equations was to select a set of variables to be fixed. Let's imagine augmenting the coefficient matrix as we perform the gaussian elimination process with these equations with the columns that correspond to the fixed variables. We carry out the forward elimination on them as we proceed to factor the coefficient matrix in the linear equation solver. Fig. 26 illustrates what we would see when we encounter the small last pivot in the column corresponding to Δx_N.

We can look under one of these columns for a pivot that is of acceptable size. The existence of such a pivot suggests we should trade one of the computed variables for this variable. Let's call this variable Δu_j. The trade is then to:

- move Δu_j from the set of fixed variables to the set of computed variables
- move the variable Δx_N from the set of computed variables to the set of fixed variables.

It turns out that we do not have to consider only Δx_N as the variable we can trade. We can, in fact, exchange Δu_j for any one of the variables Δx_k which contributed to the creation of the small pivot, ε_1. One way to look at a set of linear equations is that it is telling us a combination of the columns that produces the column on the right hand side, as the following illustrates:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \end{bmatrix} x_1 + \begin{bmatrix} a_{12} \\ a_{22} \\ a_{23} \end{bmatrix} x_2 + \begin{bmatrix} a_{13} \\ a_{23} \\ a_{33} \end{bmatrix} x_3 = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

We want to know which linear combination of the previous *N-1* columns equal this last column; it is precisely these that led to its "demise" of the last column and the creation of the small pivot. Look again at the last column Fig. 25. During the forward elimination process we processed it and the RHS in exactly the same manner. It in fact is acting like a RHS to the first *N-1* equations if we ignore the last row.

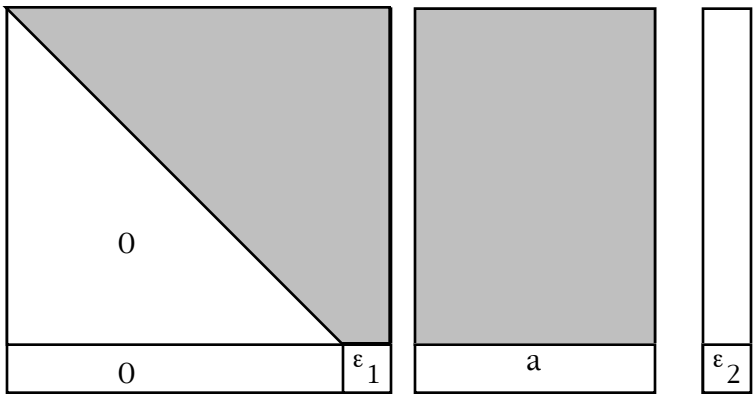


Fig. 26. Result for forward elimination step when columns under the fixed variables are also processed (but prevented from being used in the pivoting)

Back substituting with the existing *N-1xN-1* upper triangular matrix we already have will tell us the solution $\underline{\xi}$ to the following equations; i.e., it will tell us what linear combination of the previous *N-1* columns equals the last column. Let us suppose the solution is

$$\sum_{k=1}^{N-1} \underline{a}_k \xi_k = \underline{a}_N$$

where $\underline{a}_k$ is the k^{th} column of the original coefficient matrix but with the last row removed. Any column whose coefficient ξ_k in the solution is large can be a column we can trade for Δu_i discussed above.

Ill conditioning

While all small pivots indicate ill-conditioning, not all ill-conditioning shows up as a small pivot, unfortunately. Consider the following set of linear equations

$$\begin{bmatrix} 1 & -2 & 0 & \cdots & & 0 \\ 0 & 1 & -2 & & & \\ 0 & 0 & 1 & -2 & & \\ \vdots & & & \ddots & \ddots & \\ & & & & 1 & -2 \\ 0 & & & \cdots & 0 & 1 \end{bmatrix} \left[\underline{x}_1, \underline{x}_2 \right] = \begin{bmatrix} 0 & \varepsilon \\ 0 & 0 \\ \vdots & 0 \\ & \vdots \\ 0 & \\ \varepsilon & 0 \end{bmatrix}$$

where we show two right-hand-side vectors of equal magnitude. Back substituting on the first column on the right hand side give $\underline{x}_1$ and on the second $\underline{x}_2$ with the result being:

$$\left[\underline{x}_1, \underline{x}_2 \right] = \begin{bmatrix} 2^{N-1}\varepsilon & \varepsilon \\ & 0 \\ \vdots & \vdots \\ 2^2\varepsilon & 0 \\ 2\varepsilon & 0 \\ \varepsilon & 0 \end{bmatrix}$$

The first has a very large magnitude if N is large while the magnitude of the second is unchanged. A problem where the solution for two right-hand-sides of the same magnitude can be have a very different magnitude is called ill-conditioned. If we are solving a set of nonlinear equations using a Newton-based method and if the linearized Newton equations are ill-conditioned, a small change in the RHS can lead to very different sized Newton steps.

For this particular example, we cannot spot the ill-conditioning by looking at the pivots. They are all equal to unity here. In fact the equations fully *precedence order* and are solved one at a time by a backward substitution step . There are none of the usual clues to alert us to a conditioning problem. What can we do here?

Here the problem is not so easy to cure. We are working on an approach that can monitor the condition number as one uses QR factorization rather than the usual L\U factorization. L\U factorization corresponds to standard Gaussian elimination when solving the linear equations. QR factorization produces an upper triangular matrix R which has the same condition number as the original matrix. We have methods based on a paper by G.W. Steward (1992) to find a column to remove from the problem that will improve the condition number the most. We have no real experience with it yet, but the cure we are anticipating is that it will again provide us with an aid that tells us to interchange the roles of a computed and a fixed variable in a manner that makes the problem solvable.

Conditional models

Dynamic modeling

We shall examine dynamic modeling by looking at water vaporizing in a heated tank. We shall proceed as before through the steps of creating a model for this system.

Description of purpose

We wish to model the dynamic behavior of a tank containing boiling water. We shall remove both hot water and steam as products from the tank. The base case has the feed water conditions and flow to the tank fixed. Our goal is to run this vessel at constant pressure and liquid level. We want a simple model capable of illustrating important issues related to dynamic modeling.

Description of underlying principles and assumptions

We shall assume the liquid and vapor phases are in equilibrium. The tank has no heat losses through its walls to its environment.

Criteria to assess its success

The model is a success if we can explain several important issues related to dynamic modeling with it. Of course we would like it to be fast to solve (simple again).

Its defining equations

The equations might be the following.

$$U_T' = \bar{h}_F f - \bar{h}_L l - \bar{h}_V v + Q \tag{1}$$

$$M_T' = f - l - v \tag{2}$$

$$M_T = m_L + m_V \tag{3}$$

$$V_L = \bar{v}_L m_L, \quad V_V = \bar{v}_V m_V \tag{4, 5}$$

$$V_T = V_L + V_V \tag{6}$$

$$U_T = \bar{u}_L m_L + \bar{u}_V m_V \tag{7}$$

$$P(= P^{\text{sat}}_{\text{water}}\{\text{mm Hg}\}) = \exp(18.3036 - \frac{3816.44}{T\{\text{K}\} - 46.13}) \tag{8}$$

$$\overline{v}_L = \overline{v}_L(T) \; (\approx 18 liter \; / \; kg_mole) \tag{9}$$

$$\overline{v}_V = \overline{v}_V(T,P) \; (\approx \frac{RT}{P}) \tag{10}$$

$$\overline{u}_L = \overline{u}_L(T,P), \quad \overline{u}_V = \overline{u}_V(T,P) \tag{11, 12}$$

$$\overline{h}_L = \overline{h}_L(T,P), \quad \overline{h}_V = \overline{h}_V(T,P) \tag{13, 14}$$

$$\overline{h}_F = \overline{h}_F(T_F, P_F) \tag{15}$$

where

f, l, v	feed, liquid and vapor flowrates {kg_mole/s}
$\overline{h}_F, \overline{h}_L, \overline{h}_V$	partial molar enthalpy for feed and for liquid and vapor phases, respectively {kJ/kg_mole}
m_L, m_V	molar amounts of liquid and vapor, respectively {kg_mole}
M_T	total molar holdup in unit {kg_mole}
M_T'	time derivative of total molar holdup in unit {kg_mole/s}
P, P_F	pressure in vessel and of feed respectively {mm Hg}
Q	Heat flow into the system {kJ/s}
T, T_F	temperature in vessel and of feed respectively {K}
$\overline{u}_L, \overline{u}_V$	partial molar internal energy of liquid and vapar phases respectively {kJ/kg_mole}
U_T	total internal energy (holdup) in unit {kJ}
U_T'	time derivative of total internal energy in unit {kJ/s}
$\overline{v}_L, \overline{v}_V$	partial molar volume of liquid and vapar phases respectively {m^3/kg_mole}
V_L, V_V	Volumetric holdup for liquid and vapor phases respectively, {m^3}
V_T	Total volumetric holdup in unit {m^3}

The variables in this problem can be classifed into several kinds.

First there are the *state* variables which are those whose time derivatives appear. In this model the state variables are M_T and U_T. State variables are capacity variables in a dynamic model. They always represent some type of holdup in the system. A way to think about them is that their values are virtually always continuous with time provided we do not allow the inputs or outputs to the system to be modeled using "impulse" functions. If we allow the feed to be an impulse by letting it be a bucket full of water dumped into the system within a split second, the moles in the system will take a discontinuous jump with time. Ruling out such an input (or output), the holdup variables are continuous. Is the choice of holdup variables unique? The answer is that it is not. The number is but not the choice. For example, we could have selected to model the material holdup by selecting the volume of the system to be a state variable. With some variable substitution, modelers often replace the internal energy by temperature, making it the second holdup variable.

We need to comment on the use of internal energy in our dynamic energy balance while the right-hand-side is written in terms of enthalpies. A correct energy balance has to be in terms of internal energy, not enthalpies. The first law says that energy is conserved. The difference in the heat brought in and the work done by the system is accounted for by a change in the internal energy in the system, not by the change in the enthalpy of the system. The use of enthalpies on the RHS is also correct. A stream entering the system brings in its internal energy. It also has to "push" its way into the system, doing "PV" (pressure-volume) work. Enthalpy accounts for both these terms. Unfortunately, we must observe that *most text books and many articles in the literature do the energy balance incorrectly*. It is a good approximation to do an enthalpy balance for liquids as $\Delta H=\Delta U+\Delta(PV)$ (if you remember your thermodynamics) and ΔPV is very small relative to ΔU for liquids. It is a major error to do it for vapor holdups

There are also a number of variables we might classify as being physical property variables: $\overline{h}_F, \overline{h}_L, \overline{h}_V, \overline{v}_L, \overline{v}_V$. These are provided by a physical property package and are a function of the temperature, pressure and composition (if the system were a multicomponent one). We have very deliberately represented these variables as (partial) molar quantities. The bar over them is usually reserved for the molar property of a component *in a multicomponent mixture*. As an example, the partial molar volume of water in a water and ethanol mixture is the contribution to the molar volume of the mixture $\{m^3/kg_mole\}$ that is made by the water in the mixture. It would be less than 18 $\{liters/kg_mole\}$ for the water because adding 1 liter of water to 1 liter of ethanol results in something less than 2 liters of mixture. They do not mix ideally. When the material is a pure component, one can, but does not have to, remove the overbar to indicate a pure component. As the two are equal, we leave it there to guarantee there is no misunderstanding about what it is.

Why do we use partial molar quantities? We do it because then the equations we have written have absolutely no approximations due to the physical properties in them. If we later chose to approximate how to compute these partial molar quantities, so be it. But there is no good reason to put that approximation into the model at this point. In other words, we chose not to mix approximations to estimate physical properties with the other modeling approximations. We believe that it is much less confusing to do it as a later

step. In fact we believe it so strongly that we shall make this idea a rule of modeling.

> Rule of modeling: Use partial molar quantities when writing a model. If you intend to use approximate calculations to estimate partial molar quantities, do it as a separate later step in the modeling.

Other variables in the model are frequently termed *independent* variables. We shall use our intuition here to pick these out, but, in a moment, we shall do it properly. Independent variables are those we manipulate by opening and closing valves, by turning on and off pumps, or by turning on and off heaters. They are also variables which we choose to provide as inputs to the problem. Here we classify Q, f, l and v as potential control variables and T_F and P_F are variables we are likely to provide as inputs.

If our model is correct we can compute the remaining variables using the algebraic equations we wrote; we term them *dependent algebraic* variables.

Let's now analyze the model to see if it and our intuition in classifying the variables agree. We have written 15 equations in 24 variables. If we are correct, then we have a model with 9 degrees of freedom. We intuitively selected 6 earlier to be independent variables. Why are we off by three?

We can explain two of the three because we did not yet account for the equations that relate the state variables to their respective time derivatives. These equations are the numerical integration formulas that will be provided by the numerical integration package. For example, let us assume that we are going to integrate these equations using Euler's method:

$$M_T(k+1) = M_T(k) + \Delta t(k) * M_T'(k) \tag{16}$$

$$U_T(k+1) = U_T(k) + \Delta t(k) * U_T'(k) \tag{17}$$

where

$$\Delta t(k) = t(k+1) - t(k)$$

and $t(k)$ is the k^{th} time point. These equations state that we can compute the state variables at the $k+1^{st}$ time point using both their values and their time derivatives at the k^{th} time point.

(Euler's method is an *explicit* method. The values estimated for the states depend only on the values for variables at the previous time. Euler's *implicit* method (often called the trapezoidal method) computes the states at the $k+1^{st}$ time step in terms of the average of the derivatives at the $k+1^{st}$ and k^{th} time step. In other words, the values implicitly involve themselves as we compute the time derivatives at the $k+1^{st}$ time step in terms of the states at that time. The equations are:

$$M_T(k+1) = M_T(k) + \Delta t(k) * \frac{M_T'(k+1) + M_T'(k)}{2}$$

$$U_T(k+1) = U_T(k) + \Delta t(k) * \frac{U_T'(k+1) + U_T'(k)}{2}$$

Explicit methods are faster but fail to solve stiff systems unless we use extremely small time steps. Implicit methods require iterative solution at each time step. They solve stiff problems stably even while using large time steps.)

We are still off by one degree of freedom. After some frustration, counting and recounting the equations and variables, it occurs to us that we forgot one specification for the problem. We are using a vessel with a fixed total volume. Therefore, V_T is fixed. (It was, after all, the reason we related the moles of holdup to the volume - so we could place this specification.) It should be in set of independent variables. We now have 24 variables, 17 equations and 7 independent variables. The model looks as if it might be correct. Until we actually do some computations successfully with it, we cannot be sure we have it right, however.

Solving

We now need to consider how we would solve this model. To march forward in time, we proceed as follows.

Initial steady-state (not necessarily the initial condition)

We somehow have to establish initial values for the state variables (i.e., the holdups) in the system. Let's assume we are at steady-state initially. At this condition the time derivatives for the variables are all zero. We fix the variables M_T' and U_T' to zero.

We have no idea the amounts of material in the vessel nor its internal energy. What we are willing to state is that we want it to be 50% liquid and 50% vapor and we want the initial pressure to be 1 atm. We use the model equations to compute this steady-state. It is question of picking enough variables to give us 15 equations in 15 unknowns for the user model equations (excluding the integration package equations which are used to relate the variables at the next time step to those in the previous time step(s)). There are 24 variables; we need to fix 9. The two state variable derivatives are fixed, as we said, at zero. We specify another 7 by specifying the feed completely *(f, T_F,P_F)*, the total volume of the unit and the volume of liquid *(V_T, V_L)*, the pressure *(P)* and liquid product flow *(l)*. If we have it right, we can now compute an initial steady-state condition, obtaining values for the temperature, the heat input, etc..

Initial condition for simulation vs. time

This steady-state may not be the desired initial state. Often in dynamic simulation, we would like to "jolt" the system and watch its time response. If we jolt the system, it will not be in a steady-state to start. We need to compute

starting values for the time derivatives of the states which are different from zero. Many articles perceive this computation to be difficult. They suggest resetting the setpoints on some very fast control loops and altering the setpoints. However, it is really straight forward to carry out the needed computation to get our initial point provided we think about it correctly.

Suppose we would like to make a step change in the inlet flow to see the time response to it. If we change the inlet flow by stepping it, the states will remain continuous, so we can let them retain the values just computed from the initial steady-state. We use the 15 model equations. We fix the states leaving us with 7 variables to fix to make the system square. This time the variables selected are those we intend to use as the independent variables for the simulation. We listed these already: the total feed *(f, T_F,P_F)*, the heat input *(Q)*, the outlet flows *(l and v)*, the volume of the unit *(V_T)*. All but the feed flow, *f*, will retain their values at the just computed steady-state condition. We make the step change in the feed flow and compute values for remaining variables including the non-zero time derivatives at the initial condition.

Marching forward in time

We are now ready to march forward with time. Fig. 27 illustrates a way to think about the approach. It is mixture of an equation-solving and a procedural approach. We give control to the integration package. It decides on the times and the values for the state variables at which it wants the model to compute the time derivatives for the states. At each step, it computes error estimates to control the size of step it should take in time. The smaller the step, the more accurate the trajectories are (unless round-off error takes over) but also the more time it takes for the simulation. If the problem is stiff (has some extremely rapidly *decaying* states relative to the time span over which one wants the simulation), one will use implicit numerical methods such as those based on Gear's ideas. We use the LSODE routine which are available from Stanford.

The integration method does not have to know about nor, in principle, care about the algebraic variables, provided their solution is done accurately enough that one does not mess up the error control or the convergence properties of the integration method. Typically this means one must control errors somewhat tighter for solving the user model than the accuracy the modeler has asked of the integration package (say, 10^{-9} vs. 10^{-6} for the integration package).

At each time t(k) the integration package asked for a solution of the model. We specify the independent variables as functions of time so we first determine these at t(k). Then we solve the model equations, getting values for the algebraic variables and the derivatives of the state variables. We return to the integration packages with these latter values. It decides where to evaluate the model next and the cycle continues. Marching forward in time continues until the model meets some termination criterion. It could quit at a prescribed final time or when some variables reaches a prescribed value.

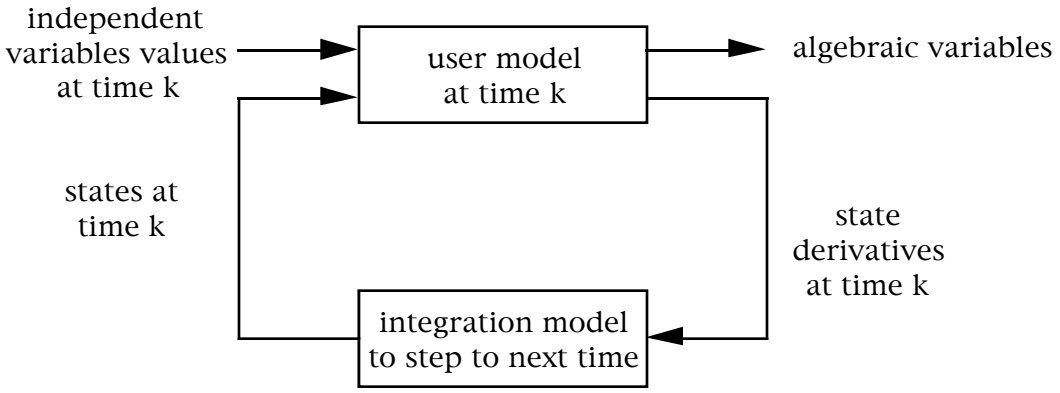


Fig. 27. Coupling a user model to an integration model

Index problems

In the above model, the model equations must be able to compute the time derivatives for the states given the values for the states. Sometimes we choose independent variables in such a way that this computation is singular; i.e., it cannot be done. Often, however, the model really has enough information that the requested computation is possible. To accomplish it requires we (or the system) do more work on the model. If this situation occurs, we say the problem has an index problem. We are in trouble in the above model if we select the pressure to be an independent variable whose value we intend to hold constant. This calculation is a valid one to ask for, in principle. However, the model as formulated will not solve as described above.

To expose one view of this problem, let's examine the incidence matrix (Table 9) for the equations and variables for this model. To keep the matrix manageable, we will "mentally substitute" the definitions for the physical properties into the first eight equations and drop their defining equations (eqns 9 to 15). That means that every time a partial molar quantity appears in an equation, we will indicate that the equations depends on pressure and termperature. We shall also drop the columns for the variables which we know now are to be fixed for the computation: the two states (M_T, U_T), the feed (f, T_F, P_F) and the total volume (V_T).

The first two equations must be used to compute the time derivatives for the state variables (they must be computed, and they appear in no other equations). We note that when these variables are assigned, the variables *l*, *v* and *Q* appear in no other equations; they can only get their values by our choosing them to be among the set of independent variables for the problem We can readily assign the remaining variables to eqns 3 to 8 as shown including assigning *P*. *P* must be calculated - i.e., not fixed - therefore. The exact same analysis used to check earlier models to find a proper set of variables to fix also works here to avoid structurally caused index problems. The trick is to fix the state variables and choose degrees of freedom to make the remaining problems square. One must avoid selecting the time derivatives of the state variables to be among those one fixes.

Table 9. Incidence matrix for dynamic boiling water example. Assigned variables are marked by asterisks before and after the incidence.

eqn \ var	l	v	m_L	m_V	M_T'	P	Q	T	U_T'	V_L	V_V
1	X	X				X	X	X	$*X*$		
2	X	X			$*X*$						
3			$*X*$	X							
4			X			X		X		$*X*$	
5				$*X*$		X		X			X
6										X	$*X*$
7			X	X		X		$*X*$			
8						$*X*$		X			

Overcoming an index problem manually

Suppose we would like to set the pressure equal to a constant value and solve our boiling water example. Our intuition tells us we should be able to back ompute one of the flows out of the system to give us this pressure, e.g., the vapor flow. Is it possible? It is not without doing some manipulation of the problem statement. With the current form for the model equations, we must include pressure among the variables being computed.

An approach is to take the full time derivatives for all but the first two equations. Then add to these equations that the $P'=0$ (note it is P' and not P) along with an initial condition equal to the value desired. The full time derivatives for Eqns 3 and 7 will create equations containing the time derivatives of M_T and U_T in them, respectively. Let's take the time derivative of eqn 3 to see this:

$$M_T' = m_L' + m_V'$$

Thus these equations offer an alternative from which one can compute these time derivatives, very likely leaving eqn 1 or 2 available to compute f. The other extra equation will allow one to compute M_T or U_T directly. Suppose it is M_T. Then one has a way to compute the value for a state variable M_T at any time point as well as its time derivative M_T'. There is no reason to compute it through a numerical integration. The net effect is that one of the state variables disappears from the problem as a state variable.

Another physical way to think about an index problem is that one loses a state variable if one can cure the problem. Consider two tanks with an pipe connecting them that allows flow in either direction. If we leave the valve open, the tank levels become equal. Suppose the valve resistance goes to zero. The levels in the two tanks become totally coupled and they act like one level. The level is a state variable in each tank, but, as the resistance goes to zero, one become algebraically equal to the other and is no longer a state variable.

Homework 8

1. Develop the modeling equations for the above example of water boiling in a tank where you introduce the concept of a state to characterize the intensive variable describing the liquid and vapor phases as well as the feed and product streams. The model you develop should equate the state for the liquid holdup in the tank to the state of the liquid stream leaving the tank as well as the state for the vapor holdup to the state of the vapor stream leaving the tank. Are there any equations associated with the state in this special case of a pure component? (Think of the physical property computations to be done.)

References

Bullock, L. and L.T. Biegler,

Levenberg,

Marquardt,

Prigogine, I. and R. Defay, *Chemical Thermodynamics* (trans. D.H. Everett), pp187-188, Longman, London (1954).

Reid, R.C., J.M. Prausnitz, and T.K. Sherwood, *The Properties of Gases and Liquids, 3rd Ed.*, McGraw-Hill, New York (1977).

Steward, D.V., "On an Approach to Techniques for the Analysis of the Structure of Large Systems of Equations," SIAM Rev., **4**, 321-342 (1962).

Taylor, natural continuation

Westerberg, A.W., and S.W. Director,