To the Graduate Council:

I am submitting herewith a thesis written by George M. Crews entitled "HAPN: A Hydraulic Analysis of Piping Networks Program." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Engineering Science.

Osama Soliman

Osama Soliman, Major Professor

We have read this thesis
and recommend its acceptance:

Carl J. Remenyi

R. A. Kenn

Accepted for the Council:

C. W. Minkel

Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Requests for permission for extensive quotation from or reproduction of this thesis in whole or in parts may be granted by the copyright holder.

Signature George M Crews

Date April 21, 1989

HAPN: A HYDRAULIC ANALYSIS OF PIPING NETWORKS PROGRAM

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

George M. Crews

May 1989

ABSTRACT

In the past, design engineers were grateful just to have a computer program that could correctly execute an algorithm. Now not only do they require programs that increase their analytical capability, they seek programs that enhance productivity and certitude as well. As a first step toward meeting these needs in the area of hydraulic analysis of piping systems, program HAPN has been developed. An unusual technical feature of the program is that the hydraulic resistance of tees (which is a complicated function of flow) can be analyzed.

The program is written in a modern and portable language (C) and targeted for popular and user-friendly hardware environments (PCs). Structured data and programming techniques were used to produce a program that is easily enhanced.

Input and output has been streamlined as much as practicable, both from the user's and the programmer's perspective. A generic metalanguage processor, called MEL, was designed to handle input and output. (MEL can be used by other analysis programs once the proper "dictionary" has been defined.) HAPN has an extensive database of reference data (such as pipe sizes and hydraulic resistance data) and can automatically handle units conversions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF PLATES

I. INTRODUCTION

Thesis Purpose

A common task for mechanical or civil engineers is the design of internal fluid flow systems such as chemical process piping, hydraulic systems, water distribution systems, and fire-sprinkler piping. A useful tool for completing such assignments would be a hydraulic network analysis program. Unfortunately, there do not seem to be many programs of this type readily available. And the ones accessible are often only for single branched systems, don't include the components needed, or are otherwise difficult to use. One of the goals of this thesis has been to develop a general purpose and convenient to use program, called HAPN, that calculates the pressure drop and flow rate of fluids in piping systems. HAPN was written to be immediately useful to design engineers (as an analysis tool) and to serve as the foundation for a comprehensive piping network design program. The program can currently handle multibranched systems and contains an extensive database of hydraulic resistances, piping data, and conversion factors. An unusual feature of HAPN is that the hydraulic resistances of tees (which are complicated functions of flow) can be analyzed.

Moreover, it was recognized that there is more to engineering analysis programming than just correctly coding a numerical algorithm. Firstly, not only do practicing design engineers require

programs which increase their analytical capability, they need programs that increase their productivity and certitude as well. Thus, technical software will become more oriented to the exigencies of the occasional user of such software. Secondly, experience has shown that software maintenance and enhancement is a very difficult task and thus more attention needs to be devoted to this area during program design and development.

It is the author's feeling that modern engineering analysis programs will soon serve as foundations for a new genre of engineering design programs. It is not too difficult to envision a design program as including an expert system, tutorial, economic optimizer, and a user-friendly interface in addition to "number crunching" abilities. It is important that these added features be developed in a standardized way. Software tools must continuously be developed and modern programming strategies and methodologies followed. In addition, advances in engineering technology need to be incorporated into these programs as rapidly, and with as little extra effort, as possible. Engineering analysis programs are evolving into a combined effort of engineering and computer science.

The Engineering Science and Mechanics Department at the University of Tennessee, Knoxville is noteworthy in its interdisciplinary approach to instruction in several different fields of engineering. Therefore, graduate students in this department, such as the author, are well situated to tackle the problem of user-

friendly technical software. Cohorts Jon B. Woods and Albert M. Hines, for example, are both writing enhancements that include pre- and post- (data) processors for in-house finite-element analysis programs. What was unusual in this author's thesis was that HAPN was specifically designed to be very accommodating to such data processing efforts.

It is also hoped that the program will prove to be integrable into future computer system and technical application software architecture advances (such as the emerging MIT X-Windows "standard"). To the degree the project has been successful, this program can serve as a template for future analysis programming efforts.

Finally, the author notes that he has more than a little experience both as a piping engineer and as a computer programmer. He has written three hydraulic analysis programs in the past, one in BASIC for the old Tektronics 4150, then one in APPLESOFT for the Apple II, and a still later one in FORTRAN for the IBM PC. Although the design and code for HAPN is completely new, the program is really sort of a generalization and culmination of these earlier more practical experiences. This is probably why he feels so confident about the efficacy of the program's design and potential practicality.

Programming Philosophy

More emphasis was placed on software engineering techniques and nonnumerical design for HAPN than typically found in most other engineering analysis programs. HAPN is not just a set of engineering formulae reduced to algorithms with a few input and output statements added front and back. For example, the program was designed to be very "structured" both logically and datawise so that additional features or modifications may be made in a modular manner. The interested reader may examine this structure by referencing the source code contained on the HAPN Source Diskette included at the back of this thesis. (Start by reading Appendix E.) Source code has been heavily commented in part to bring out its structure.

This does not mean, however, that the reader would find the program's design contained (for instance) Yourdon/DeMarco style data-flow diagrams, structure charts and data dictionaries, or that all data "objects" had encapsulation, inheritance, destructors, etc. (Orr, 1989). The paradigm used in the design of HAPN was much more flexible and less formal than that. It has been the author's experience that simple strict adherence to a well-defined methodology does not guarantee (or even strongly influence) the success of a software project. The making of a devout programmer is more spiritual than ritual.

Instead, the general philosophy has been to make as few unnecessary assumptions about data and function as possible, and when they are necessary, to make them as flexible as possible. All program capabilities have been functionalized and accessible strictly through function calls. Where possible, data has been grouped into structures so that its details become less burdensome. Common computer software design methodologies and "safe" programming practices have been adhered to where appropriate. And of course, the user interface has been isolated from the analysis or technical parts of the program.

Consequences of this programming philosophy have been:

1. A substantial amount of time (approximately 50%) needed to be devoted to program design before coding even began. The most important components of the program were determined to be the various data structures, and these were defined first. (For a list of these data structures, see the `hapnincl.h` file on the HAPN Source Diskette.) In order to ensure the adequacy and consistency of these structures, the main program modules were all detailed functionwise and expressed in a structured English-like (but slightly ambiguous and thus easy to write) pseudo code. Even though the pseudo code became obsolete as soon as actual coding in the target language began, it served a very useful and important purpose as a template for the actual code modules. (For those interested, see the `hapn.pc` file on the HAPN Source Diskette for a listing of the pseudo code developed for this program.)

2. HAPN was written in a modern, portable, and extremely popular language (C) and initially has been targeted for a readily available, user-friendly hardware environment (PCs). It is the author's opinion that C has several nonnumerical advantages over FORTRAN, the traditional language for engineering analysis programming. The new ANSI C is a very popular and standardized language, it allows complex data structures to be created, it has more operators and program control constructs, it allows indirection (pointers), and it more easily interfaces to existing GUI (graphical user interface) software. (To be fair, the author should mention that C still has some shortcomings. It is a more difficult language to learn and use, dynamic array dimensioning is not straightforward (Press, 1988), it's still a relatively new language and thus does not have the numerical library that FORTRAN does, and C does not allow operator overloading for data structures nor does it have an exponentiation operator.) The choice of personal computers as the target hardware environment reflects the trend of design engineers away from the computer room and toward the desk top.

3. The current version of HAPN interfaces directly with the user, but in the future will have to interface with its controlling design program. The solution to this anticipated dilemma was to design and implement a generic metalanguage (pre- and post-) processor, called MEL, to handle all input and output syntax and grammar. (For more information on MEL, see Appendix A.) To ensure the acceptability of such a solution, a generic interactive

user interface was conceptually designed as a sequence of terminal display screens. This interface (or GUI design program if you will) would, when made functional, combine a metalanguage editor, on-line documentation, tutorial, expert system, and run-time system shell. (To see a display of these screens, type out the file `screens.doc` on the HAPN Source Diskette.) The author then convinced himself that MEL could indeed interface and efficiently be used in such a GUI environment. (For the current version of HAPN, a standard word processor or text editor must be used to edit MEL files.)

4. Required user input was streamlined as much as practicable. Output is given in a form thought to be most useful to the piping designer.

Program Features

It is hoped that someday HAPN will become a commercial program. However, from a marketing standpoint, this would probably require adding a lot more "bells and whistles" to the program. Such additional efforts are too much work for a master's level thesis. Instead, what the author has tried to do is to build a solid foundation upon which a marketable program can be produced. (HAPN represents well over five man-months of full time work. It could easily be the case that a "full-featured" program would take at least three, perhaps six, times the effort.)

Considerable attention was put into designing a program that could most easily evolve into such a product.

Current program features are listed below, while characteristics required of a commercial program are listed in the Conclusions and Future Enhancements Section. Core program specifics are:

1. The pressure drop and mass flow rate of incompressible newtonian fluids (or compressible fluids at moderate pressure drops and Mach numbers) are calculated using the Darcy equation (Mott, 1979).

2. For multiple-branched systems, the simultaneous linear continuity equations and nonlinear pressure loss equations are solved using the "Linear Theory" method (Jeppson, 1976). The big advantage of this method is that it does not require initial flow guesses from the user. Node and loop equations are automatically generated by the program based on the input geometry (using breath-first traversals of an internally constructed network graph for the loop equations).

3. The hydraulic resistances of common piping components have been made part of the program database (Crane Co., 1978, Ward-Smith, 1980, Idelchik, 1986). This allows the user to include these components by merely referencing a key-word rather than entering a manually calculated number. (This is important since hydraulic resistance formulae are often complicated and functions of the unknown flow.)

4. Input to the program is from a user specified text file in an English-like key-word, free-field format (MEL).

5. Input and output from HAPN is either in default units or units specified by the user. Conversion is automatically made to the desired units.

II. TECHNICAL BACKGROUND INFORMATION

Introduction

One of the purposes of this section is to provide a practically oriented review of the engineering equations upon which HAPN is based. This overview is not designed to be pedagogical, but instead is intended to help potential users determine if they have sufficient academic background to run the program. (Fortunately, hydraulic analysis is not a difficult subject. For a simple yet fairly complete introduction to this topic, see Mott, 1979.)

Perhaps even more importantly, another purpose of this section is to present background information that the author feels will help the program user develop or reinforce a little of the "engineering judgement" so often required to properly use an analysis program in a design environment.

Program Structure

In order to use the program successfully it is important to understand, in general, how HAPN works. The following listing describes the overall structure of the program. Notice that it is just the top most module of the pseudo code from the HAPN Source Code

Diskette file `hapn.pc`. (The interested reader may examine the rest of this file to find out more algorithmic details about the program.)

```
hapn()    // module #0
     // hydraulic analysis of piping networks

     initialize() // module #1
          // global program initializations such as
          // reading in database.

     do {

          set_up_next_problem() // module #2
               // input for several networks may be
               // stacked, do initializations for
               // next problem.

          process_MEL_input_file(yet_another_case_flag)
               // module #3
               // standard meta-language input format
               // used, get input for next problem.
               // flag is true iff there is another
               // problem after this one.

          calculate_resistance_factors() // module #4
               // use data and "canned" formulas to
               // calculate all component resistance
               // factors.  (note that k-factors are
               // often a function of flow, so these
               // initial calculations may well be
               // approximate.)  then sum to obtain
               // branch resistances.

          generate_equations() // module #5
               // figure out node and loop equations
               // based on the geometry that was input.

          do {

               solve_equations() // module #6
                    // this routine iteratively solves
                    // the non-linear equations that
                    // were generated.

               recalculate_resistance_factors
                    (convergence_flag)
                    // module #7
```

```
                    // since k-factors are a function
                    // of flow, see how much they have
                    // changed based on newly
                    // calculated flows.  flag returns
                    // true iff all component k-
                    // factors are within user
                    // specified convergence
                    // tolerance.

          } while ( convergence_flag == false )

     calculate_nodal_pressures()  // module #8
          // once all pressure drops have been
          // calculated, nodal pressures may be
          // determined.

     process_MEL_output_file() // module #9
          // standard meta-language output format is
          // used.  write results in this format to
          // file.

} while ( yet_another_case_flag == true )
```

Modified Bernoulli's Equation

At the present time the fundamental equation used by HAPN for calculating component pressure drops is the constant density, single-phase, steady-state, one-dimensional, laminar or turbulent flow form of the modified Bernoulli's equation. In this equation, the pressure drop across an individual piping component has three contributing factors: elevation change, velocity change, and friction. That is:

$$\Delta p_t = \Delta p_z + \Delta p_v + \Delta p_f \qquad (1)$$

where:

Δp_t = total component pressure drop

Δp_z = pressure drop caused by elevation change

Δp_v = pressure drop caused by fluid velocity change

Δp_f = pressure drop caused by friction (viscous dissipation).

The relative importance of each of these factors varies, and so its contribution may or may not be included in a given component type. Each of these terms will next be examined individually in a little more detail.

Elevation Pressure Drop

If the outlet of a component is at a different elevation than its inlet, then the elevation change pressure drop term of the modified Bernoulli's equation can be shown to be:

$$\Delta p_z = g \cdot \rho \cdot \Delta z \qquad (2)$$

where:

g = the gravitational constant (in consistent units)

ρ = fluid density

Δz = component elevation change from outlet to inlet.

This term is usually only significant for liquids and only then for pipe lengths that have a significant vertical run. Thus, except for pipe, this term is ignored by HAPN. Note that it is almost

always possible for the vertical length of nonpipe components to be lumped with the next pipe run or accounted for by adding a fictitious vertical pipe spool of large diameter to a branch's list of components.

Velocity Pressure Drop

Next, consider the static pressure drop across a component caused by a change in the fluid's velocity:

$$\Delta p_v = h_v \cdot w^2 \tag{3}$$

where:

h_v = component hydraulic resistance caused by velocity

w = fluid mass flow rate.

and:

$$h_v = \frac{8}{\rho \cdot \pi^2} \cdot \left(\frac{1}{d_2^4} - \frac{1}{d_1^4} \right) \tag{4}$$

d_2 = component's reference outlet diameter

d_1 = component's reference inlet diameter.

As a practical matter, this term of the modified Bernoulli's equation will be insignificant unless the ratio of inlet to outlet diameters differ greatly from unity. Thus, except for reducers and tank nozzles, this term is ignored by HAPN. (To enhance numerical

stability, the slight pressure increase caused by fluid velocity decrease through an enlarger is ignored.)

Darcy's Equation

Finally, consider the pressure drop due to friction, which causes the dissipation of mechanical energy through the component. The form used by HAPN comes from Darcy's equation:

$$\Delta p_f = h_f \cdot w^2 \tag{5}$$

where:

h_f = component hydraulic resistance caused by friction

and:

$$h_f = \frac{8 \cdot k}{\rho \cdot \pi^2 \cdot d^4} \tag{6}$$

k = component resistance coefficient (k-factor)

d = reference pipe diameter.

Note the difference in component hydraulic resistance (h_f) and component resistance coefficient (k). They may sound similar but they are very much different. (For more information on resistance coefficients see the subsection below.) Also note that the value of a component's resistance coefficient will depend on the choice of reference pipe diameter (d) (either inlet or outlet).

Overall Hydraulic Resistance

Look at Equations (2), (3), and (5) carefully. These can be combined with equation (1) to yield:

$$\Delta p_t = h_t \cdot w^2 + g \cdot \rho \cdot \Delta z \qquad (7)$$

where:

h_t = overall component hydraulic resistance

and:

$$h_t = h_v + h_f \qquad (8)$$

Thus, a component's overall hydraulic resistance (h_t) is just the sum of the contributions due to velocity (h_v) and friction (h_f). It is this equation that HAPN uses to calculate the hydraulic resistance of piping components.

Resistance Coefficients

In order for a component's hydraulic resistance (h_t) to be calculated, its resistance coefficient (k) must first be determined. An attractive feature of HAPN is that it can automatically estimate a value for most components found in piping networks. The reader can reference the source code listing for more information on the exact formulae being used. Importantly, the reader will find that a

component's resistance coefficient (**k**) is often a function of flow (**w**) through the component. (For example, the resistance coefficient of pipe is a function of Reynolds number, which is itself a function of flow.) Thus, if flow is not known, a value for a component's resistance coefficient can only be roughly approximated, and HAPN has to use iterative procedures to solve for the component's actual flow rate. (Recall the Program Structure Subsection above.)

Another important fact for the program user to keep in mind is that there is wide (often more than 50 percent!) variation in the experimental results for resistance coefficient data. That is to say, there is much "scatter" in the data from which the empirical component resistance coefficient formulae used by HAPN have been developed. (For a description of the problems encountered for just one typical type of piping component, see Gnielinski, 1986.) Thus, results from HAPN cannot be considered to be "exact" since these results cannot be any more accurate than the approximate formulae used. Currently, HAPN uses formulae from Crane's Technical Paper 410, except for tees, which are based on formulae from Ward-Smith.

Compressible Fluids

It is very often possible to relax the assumption of constant fluid density and use this program with compressible gases. That is, the Darcy equation will often give results for gas flow problems

that are accurate enough for design purposes. As a tool for determining when this equation (and thus this program) can be used for such conditions, included on the HAPN Source Diskette is a separate program called EQTEST. This program allows the user to compare the Darcy equation with the ideal gas adiabatic flow equation and the ideal gas isothermal flow equation for a given specific component. The program is self-documented and will help the user estimate if the error in using Darcy's equation is acceptable for a particular situation. The program contains help screens that provide background information on the adiabatic and isothermal equations and the assumptions upon which they are based.

The author has used EQTEST (or some earlier version) on several occasions and has confirmed and expanded an old rule of thumb for estimating the error in using the Darcy equation for compressible gas flow problems. The expanded rule is: for low Mach numbers, the percent error in calculated pressure drop will often not exceed the percent change in total pressure. For example, if the gas'es Mach number is less than about 0.3, and the absolute pressure at a component's outlet is ten percent lower than at its inlet, then the error in the calculated pressure drop using Darcy's equation will most likely be less than ten percent. It is important to note that the Darcy equation will always underestimate the pressure drop when compared to the other two equations.

Branch Pressure Drops

By using Equation (7), HAPN is able to perform the first step in solving for the pressure drops and mass flow rates in a piping network, which is to determine (or estimate based on an assumed flow) all component hydraulic resistances. An equation can then be written for the overall hydraulic resistance of each branch which is just the sum of all the component resistances on the branch. That is, for a given branch **j** (of a network that has **m** branches) which has $\mathbf{n}_j$ components on it, **m** equations may be written of the form (see Figure 1):

$$\Delta p\, t_j = \sum_{i=1}^{n_j} \left[h_{ij} \cdot w_j^2 + \Delta pz_{ij} \right] \tag{9}$$

where:

$\Delta p\, t_j$ = total pressure drop for branch **j**

h_{ij} = hydraulic resistance of **i**'th component of branch **j**

w_j = flow rate of branch **j** (**j** = 1 .. **m**)

Δpz_{ij} = elevation pressure drop of **i**'th component of branch **j**

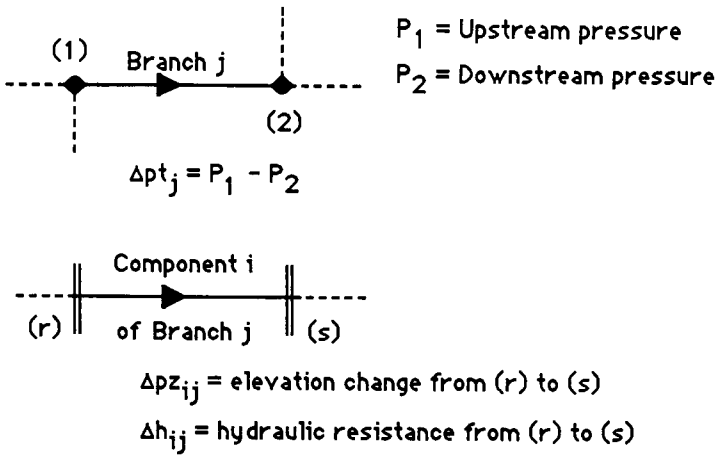


Figure 1. Piping Network Definitions for a Typical Branch j and a Typical Component i on That Branch.

For piping networks that only have a single branch ($\mathbf{m} = 1$), the above equation can be used to solve directly for pressure drop (if flow is known), or iteratively for branch flow (if pressure drop, $\Delta\mathbf{pt}$, is known). The required iterative scheme for this latter case can be summarized in the following steps:

1. Assume an initial branch flow ($\mathbf{w_{old}}$).
2. Calculate and sum all component hydraulic resistances ($\Sigma \mathbf{h_i}$) on the branch using $\mathbf{w_{old}}$ if required.
3. Estimate a new branch flow using the equation:

$$\mathbf{w_{new}} = \sqrt{\frac{\Delta \mathbf{pt} - \Delta \mathbf{p\,z}}{\Sigma \mathbf{h_i}}} \qquad (10)$$

4. Compare this new branch flow with the old branch flow. If the difference is less than a given convergence tolerance, a solution has been found.

5. Else, set $w_{old} = w_{new}$ and go to Step 2.

System Equations

For piping networks that have branched or parallel flow paths (loops), neither branch pressure drops nor branch flows are known. This means that an additional **m** equations are needed so that both flows and pressure drops may be determined. To generate these extra equations, HAPN uses two types of equations, called node mass flow equations and loop pressure drop equations. These extra equations are written solely in terms of, and thus may be solved for, the **m** branch flow rates. HAPN then uses the above branch pressure drop equations to solve for branch pressure drops separately.

Node Equations

Node mass flow equations are generated by observing that when a piping network has tees or junctions (called nodes), then a conservation of mass flow equation may be written for each of the nodes, as follows (see Figure 2):

$$qk = \Sigma qk_{in} - \Sigma qk_{out} \qquad (11)$$

where:

qk = known external flow out of node **k** (may be zero)

Σqk_{in} = sum of all branch flows that flow into node **k**

Σqk_{out} = sum of all branch flows that flow out of node **k**.

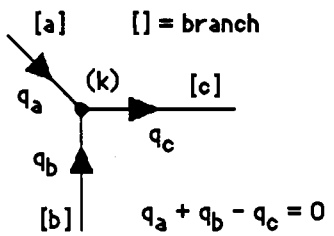


Figure 2. A Typical Nodal Flow Equation.

Notice that these equations are linear in terms of the unknown flows.

Loop Equations

If the pressure drop between any two nodes in the piping network is known, then a conservation of energy, loop pressure drop equation may be written involving all branches (**b**) on a path between the two points (see Figure 3):

$$\Delta p = \sum_{i=1}^{b} [e_i \cdot w_i^2 - \Delta p z_i] \qquad (12)$$

where:

Δp = pressure drop between nodes (0 for a closed loop)

and:

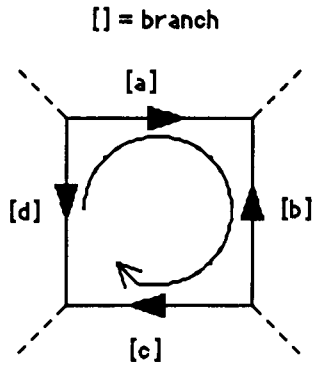
$$e_i = s_i \cdot h_i \qquad (13)$$

e_i = +1 if i'th branch flow is in same direction as known pressure drop direction; -1 if i'th branch flow is in opposite direction as known pressure drop direction

h_i = i'th branch hydraulic resistance

w_i = i'th branch flow

Δpz_i = elevation change pressure drop for branch i (0 for a closed loop).



$$h_a w_a^2 - h_b w_b^2 + h_c w_c^2 - h_d w_d^2 = 0$$

Figure 3. A Typical Loop Pressure Drop Equation.

For a closed loop, the two nodes are the same node and the pressure drop will therefore be zero. Assume a loop pressure drop direction that is either clockwise or counterclockwise but be consistent for the entire loop. For an open loop, the two nodes are

distinct and the pressure drop will be the difference in their pressures. By convention the pressure drop direction for open loops is always from higher pressure to lower pressure. Notice that these equations are nonlinear in terms of the unknowns (i.e. flows squared).

Unlike the node equations, the form of the loop equations is fairly flexible. (There is usually more than one way to get from one point to another in a network.) Thus, there is the problem of making sure each loop equation is independent and of finding out just how many independent equations the network contains. As it turns out, independence is guaranteed as long as at least one branch on each loop appears in no other loop equation. HAPN uses this fact to generate the loop equations. The exact procedure is fairly complicated, not very enlightening, and will not be presented here. (Although the reader is encouraged to consult the program source code, where the algorithm is explained in detail.) The trick is to make a breath-first search of the network graph until the desired starting and ending nodes are encountered. With certain exceptions, the ending node will be the starting node if its pressure is not known, else it is some other node with known pressure.

Solving the System Equations

These node and loop equations form a set of simultaneous nonlinear equations with branch mass flows as the unknowns. A

node equation can be written for each node in the piping network. It will be found, however, that if all of the pressure drop equations are closed loop equations, then one of the node equations is redundant (dependent) and thus must not be included in the set of simultaneous equations. On the other hand, if one or more of the pressure drop equations are open loop equations, then each of the mass flow equations is independent and thus must be included in the set of simultaneous equations. In any event, the total number of independent equations will always equal the number of branches in the network.

Since the set of simultaneous node and loop equations are nonlinear in terms of the unknown flows, their solution is difficult. HAPN iteratively solves these equations by guessing flows, using these guessed flows to linearize the loop equations, and solving the resulting set of simultaneous linear equations for new flow guesses. This is called the Linear Theory method (Jeppson, 1976).

The loop equations are linearized as follows:

$$\Delta p = \sum_{i=1}^{b} [e_i \cdot g_i \cdot w_i - \Delta pz_i] \tag{14}$$

where:

g_i = guessed value (old value) for flow in branch i

w_i = calculated value (new value) for flow in branch i.

The first guess for flow is always one (a very attractive feature). Subsequent new guesses for even iterations are just the calculated values. For odd iterations (except the first and third) new guesses are the average of the previous two calculated values. (The reason for this strategy is to prevent the flows from oscillating about the actual solution instead of converging.) Iterations, or new guesses for the flow rates, are terminated when the differences between guesses become small.

Finally, recall that it is also necessary to iteratively calculate hydraulic resistances since many of these were calculated based on an assumed flow. The author has found that it is not efficient to recalculate hydraulic resistances between these iterative solutions for branch flows. It's better to have an outer iterative loop just for hydraulic resistance.

Tees

Tees are a complicating factor in hydraulic analysis. That is because their resistance coefficients are a function of the flow splits and flow directions between the various branches comprising the tee. The usual practice has been to avoid algorithmic difficulties by ignoring this effect and assuming a "conservative" value for the resistance coefficients that depends only on if the flow changes direction through the tee or not. HAPN on the other hand models

tees by considering flow split as well as direction. Figure 4 illustrates why this would seem to be the appropriate thing to do.

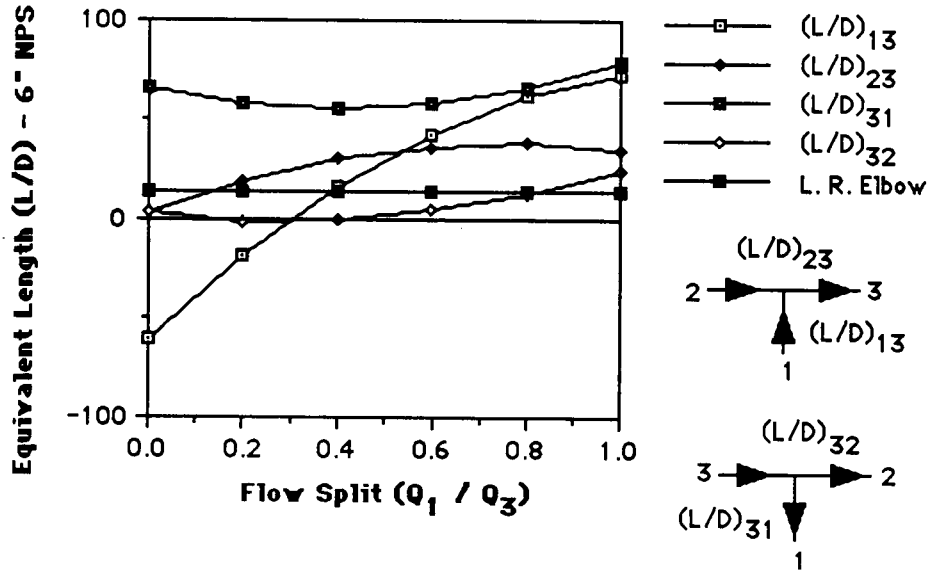


Figure 4. Hydraulic Resistances of a Tee.

This figure gives the hydraulic resistance between the ends of tees in terms of the resistance coefficient of an equivalent length of straight pipe (measured in units of internal pipe diameters) for two different flow patterns. Notice how much their values change as a function of the percent flow going through branch 1. Also notice that for a certain flow pattern and range of flow split that the resistance may even be negative! That is, the main may try and siphon flow from the lateral. Finally, note that these resistances are not insignificant by comparing them to the resistance of a long

radius elbow (another common piping component). It would seem unlikely that a single value for resistance coefficient for tees would be appropriate.

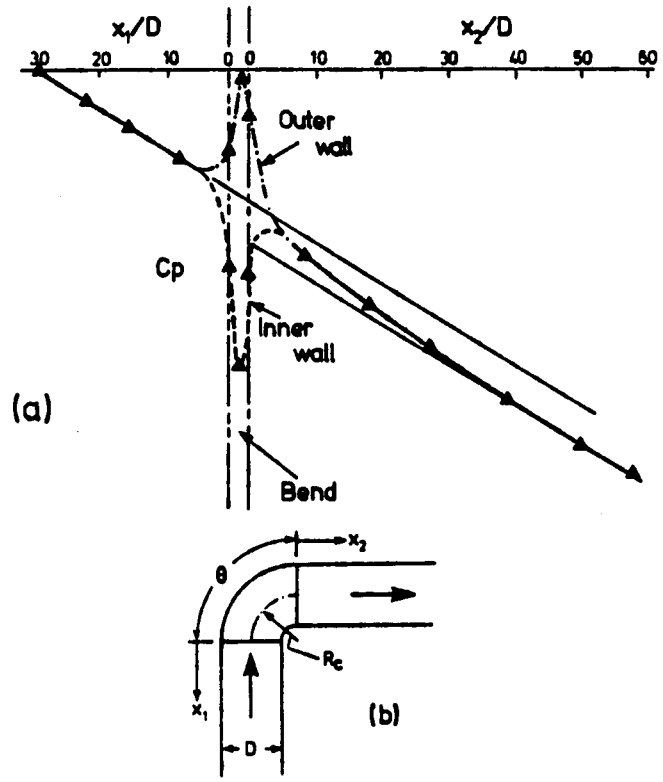
However, not only are tees an additional burden on the programming effort, they may introduce numerical instabilities as well, especially if tees dominate the overall resistance in the piping network. If the user encounters this problem the only solution is to remove the tees from the model and substitute a "conservative" resistance factor directly to each branch.

Component Interaction

If components are placed "close" to one another in a piping system, they may affect each other's hydraulic resistance (Ward-Smith, 1980). As an example of why this interaction can be expected, see Figure 5 at the end of this section, which has been reproduced from the above mentioned text. Notice that the "full effect" of the bend component does not manifest itself for 40 or more pipe diameters downstream of the bend. It's hard to say exactly what the overall effect would be on another component placed just downstream (or upstream) of the bend.

Component interaction has obvious significance for the program user. Since HAPN has no way of allowing for the effects of interaction, program results can only be considered to approximate actual system conditions.

Figure 5 also illustrates another significant and related point. Note the pressure distributions within the length of the bend component itself. Since HAPN only calculates overall average pressure distributions, these detailed distributions are assumed not to be important. Sometimes this is not a good assumption, especially for fluids near their vapor pressure.



Typical pressure distributions in the neighbourhood of a bend, $\theta = 90°$; $R_c/D = 1.85$; $(Re) = 2 \times 10^5$: (a) pressure distribution (After Ito (1960).) (b) geometry.

Figure 5. Pressure Drop Along a Bend Component. (From Internal Fluid Flow, A. J. Ward-Smith, Page 487.)

III. DESCRIPTION OF PROGRAM

Introduction

HAPN is a hydraulic analysis program for piping networks. It can calculate the steady-state pressure drop and mass flow rate of fluids in piping systems of complex geometry, and is currently designed to be an analysis tool for the practicing mechanical or civil engineer. This section describes some of the general features of the program, while more specific information on how to use it can be found in the following sections.

HAPN has the capability of estimating flow rates and pressure drops in either single branch, multiple branched, or looped piping networks. It considers the hydraulic resistance of pipes, fittings, and valves, as well as including the effects of elevation changes, external flows, pumps, and external pressures. All incompressible fluid and many compressible fluid problems can be handled.

HAPN uses a very convenient input and output processor called MEL. MEL allows all common piping components and parameters to be referenced simply by name or identifying label. Additionally, the user may specify parameters in over 150 different types of units.

Hardware and Software Requirements

HAPN will compile and run on personal computers or mainframes under any operating system that has an ANSI standard C compiler. The program was developed on an IBM PC with 640K RAM, an 8087 math chip, and PC-DOS 3.2 using Microsoft C 5.1. The program also runs under VAX VMS without modification. Currently, it will not run under any UNIX (VAX or SUN) at the University of Tennessee, Knoxville since the author has found that none of these as yet have ANSI compatible C compilers! (This is a temporary shortcoming certainly.) A text editor is also required for preparation of user input.

Input and Output

Input and output for the program are from text files that will generically be called user files. By giving different file names to different user files, data and results for many different problems can be stored on the same disk. (The program user indicates which files to use as command line parameters during program start-up.)

An unusual feature of HAPN is that the format or structure of both input and output user files are the same. This format structure, called MEL, was chosen because it is user convenient and (with appropriate choice of terms) almost completely self-documenting. For more information on MEL, see Appendix A.

Reference Data

Estimates for the hydraulic resistance of standard piping components are automatically calculated by the program. Standard piping components include bends and elbows, enlargers, exits and entrances, mitres, nozzles, pipes, reducers, tees, and several types of valves. (See the Technical Background Section or HAPN source code for more information on the hydraulic resistance formulae used.) The user may also input the precomputed hydraulic resistance of a special component directly.

In addition to units conversion factors and component hydraulic resistance estimates, HAPN will automatically calculate internal pipe diameters given nominal pipe sizes and schedule numbers.

Caveats

As described in the Technical Background Section, the most surprising aspect of hydraulic network analysis may well be the approximate nature of its resistance data. A substantial amount of the hydraulic resistance of a component manifests itself upstream and downstream (up to 60 pipe diameters or more!) of the component proper. Since piping components are usually placed more closely together than this, interactions become very

important. Thus, the user cannot expect too much accuracy from this program, even though it uses the "best" available data.

In its present form (as an analysis program) HAPN is not intended to be used by someone unfamiliar with either applied fluid mechanics or computers. It has been the author's repeated experience that design engineers tend to misuse analysis programs and he would warn the neophyte that although this program tries to be user-friendly, it is not easy-to-use in the sense of being fool-proof. Users must make sure they are modeling their problems and interpreting their results correctly. Reading and understanding the Technical Background Section is a recommended starting point. The references in the Bibliography are also highly recommended.

Most importantly, the prudent user will never consider any program to be error free, especially when applying it to new classes of problems. Always make enough manual approximate calculations to demonstrate the adequacy of any results.

Current Limitations and Known Bugs

It is important to know the range of applicability of HAPN. That is, for what kinds of problems is the program good for and for what kinds is it not appropriate? The following is a list of many of the important limitations of the current version of HAPN. Some of the limitations are a consequence of the algorithms used by the program and some are due to their implementation. (The author

does not wish to sound overly apologetic. He believes HAPN has fewer limitations than most other hydraulic analysis programs currently available.) A list of known bugs appears in the file `bugs.doc` on the HAPN Source Diskette.

1. In order to keep the size of the program within reasonable limits so that it may be run on personal computers of modest memory and speed, maximum program capacities are as follows:

Item	Max. Number
Branches	100
Components	1000
Nodes	100
Tees	100

These capacities may be increased by simple modification of the source code, recompilation, and relinking if the computer on which the program is running has adequate memory. However, be advised that, as with all simultaneous solution solvers, execution times can become very long as the number of unknown branch flows increase. For personal computers, a math coprocessor is recommended.

2. The program uses Darcy's equation to calculate pressure drop. This means the user cannot expect accurate results for all types of liquids and gases. The fluid is assumed to be constant density, steady-state, and newtonian. Thus for example, two-phase

flow is not allowed and it is up to the user to ensure that cavitation will not occur anywhere in the system. As a practical matter, if the fluid is a gas (i.e., not constant density) adequate results are still obtainable as long as the pressure drop across any branch is no more than about 10% of the total pressure at the branch inlet. See the Technical Background Section for more on this subject.

3. Most of the resistance coefficients used for piping components are calculated from empirically derived formulae. Therefore, they have a limited range of accuracy. (See also 10.)

4. The resistance coefficient of a valve depends greatly on construction details that vary from vendor to vendor. HAPN uses typical approximations for valves of the specified type. Results will be more accurate if specific vendor information is input directly.

5. Except for reducers and nozzles, HAPN ignores the effect of velocity change in the fluid on static pressure. That is, the velocity term in the modified Bernoulli's equation is not considered. The contribution of this term to the total system pressure drop is usually very minor.

6. Except for pipe, HAPN ignores the effect on static pressure of gravity. That is, all other components are assumed to be horizontal. (If required, acceptable accuracy should be obtainable by including a fictitious vertical run of pipe of large diameter.)

7. The equation HAPN generates to model pumps will exactly fit the actual pump curve only at the two points input by the user. Thus, the user must examine the program's output to verify that

the pump's predicted operating point lies sufficiently close to the actual pump curve.

8. Standard pipe data are available only for pipe sizes in the range of 0.5- to 36-inch NPS, schedule 5S to XXS or 160.

9. The Linear Theory Method for solving parallel-flow piping network problems has the reputation for reliable convergence. However, this reputation is not entirely deserved. Experience has shown convergence problems are possible in cases where pressure drop is not approximately a function of flow squared (e.g., the user specifies some special component) or, more commonly, where tees dominate the network's overall hydraulic resistance. Again, see the Technical Background Section for more information. Convergence problems caused by round-off error are also possible if a tolerance is picked that is too small.

10. A considerable portion of the hydraulic resistance of a component manifests itself upstream and downstream of the component proper. Thus, components spaced closely together will interact with each other from the standpoint of overall resistance to flow. This interaction can be very complicated and may be canceling or reinforcing, major or minor. Unfortunately, there is almost no experimental data on the subject. HAPN ignores this effect, even though it is often substantial. (It is the author's opinion this is what accounts for the distressing spread in the results in the literature of experiments on component resistances.)

11. HAPN is a large program and thus (by definition!) contains undetected bugs. Users must confirm its reliability for their intended applications.

12. User input and output file names must be compatible with the operating system the program is being run under.

IV. PREPARATION OF INPUT

Introduction

When the program is run, the first parameter on the command line is the user input file name. It is from this file that the program gets all of its input information. Thus, the job of preparing input consists of creating (or modifying) a user input file. The user will need to use a text editor (word processor) to accomplish this task. Specific examples of program input, with explanations, are included in Appendix F.

Initial Preparations

Before a user input file can be created, the piping system must be modeled. The following is a list of the steps involved:

1. Prepare a sketch of the piping system.
    a. Show how branches are connected together.
    b. Assign an arbitrary flow direction to each branch.
    c. Show all components on each branch.
    d. Show all external pressures and flows.
2. Make a list of required component data.
    a. Describe component parameters (pipe lengths, bend angles, etc.).

b. List all pipe materials of construction.

c. Find head curves for all pumps.

3. Make a list of fluid properties.

   a. Mark where fluid properties change.

   b. List fluid viscosities and densities.

4. Assign numbers.

   a. Assign a number to each branch.

   b. Assign a number to each node and tee.

Input Data Dictionary

After modeling the piping system, the next step is to prepare a user input file that conforms to MEL rules and its input data dictionary. MEL is explained in Appendix A, and the input data dictionary is explained below and in Appendix C.

The input data dictionary defines the meaning (semantics) of MEL descriptors. For example, consider the descriptor:

```
pipe, length = 1.25 (mi), dy = 13. (ft), id = 48. (in);
```

This descriptor means that the next piping component on the current branch is a 1.25 mile length of pipe that rises 13 feet in elevation and has an internal diameter of 48 inches. Appendix C contains a list of all input descriptors and a brief explanation of

their meaning. Appendix F, which gives illustrative example problems, also greatly clarifies how these descriptors are used.

Running the Program

In order to run HAPN successfully, its source code (which consists of all of the files that end with ".h" or ".c" on the HAPN Source Diskette) must first be compiled and linked. The exact procedure for these steps varies depending on the computer system being used, thus the author can only assume potential users already know how to perform this type of task or know someone who does.

HAPN was written on an IBM XT using PC-DOS 3.2 and Microsoft C 5.1. See the "make" file `hapn.mak` on the HAPN Source Diskette for an example of how to compile and link the program in this environment and as a guide for other environments.

When executing the program, three command line parameters may be specified. The first is the user input file name, the second is the user output file name, and the third is the error file name. Default values are `stdin` (the keyboard), `stdout` (the screen), and `stderr` (usually the screen). For example, to run the program on the IBM XT under DOS with an input file name of `test.dat`, sending output to the printer, and errors to the screen, type:

```
hapn test.dat >prn
```

V. DESCRIPTION OF OUTPUT

Introduction

When the program is run, the second parameter on the command line is the user output file name. It is to this file that all of the program's results are written. The user may need to use a text editor (or word processor) to examine the file and a printer if he or she wants a hardcopy of the results. Specific examples of program output, with explanations, are included in Appendix F.

The general appearance of the output is exactly the same as that for the input, i.e., MEL format is followed. MEL is explained in Appendix A, and the output data dictionary is explained below and in Appendix D.

Recall that the third command line parameter is the error file name. If an error is detected in the input, HAPN will print an appropriate error message to this error file and immediately terminate execution (without reading any more data from the user input file). The message will explain the location and nature of the problem encountered so that the user can make the necessary corrections. If an error occurs after the input file has been read, a descriptive error message explaining the problem will be printed and the program will halt.

Output Data Dictionary

The output data dictionary defines the meaning (semantics) of MEL output descriptors. Appendix D contains a list of these output descriptors and a brief explanation of their meaning. Appendix F, which describes the example problems, also clarifies what these descriptors mean.

Interpreting the Results

Just because the program did not generate any fatal errors does not mean that the output is "correct". The user must check the output to make sure it is meaningful. The following is a typical list of things to check when examining output (answers should all be yes):

1. Does pressure decrease moving downstream on a branch? Do the pressure drops across the components with the greatest resistances appear to be of reasonable magnitude? Are program predicted pump operating points close to actual pump curves?

2. Is the pressure drop around any closed loop zero? Is the pressure drop across any open loop equal to the difference in pressure at the end points?

3. Does the summation of internal and external flow at every node or tee equal zero?

4. Does the fluid pressure always stay above its vapor pressure? Do all pumps have sufficient NPSHA?

5. For compressible gas problems, are branch pressure drops less than about 10% of branch inlet pressures?

VI. CONCLUSIONS AND FUTURE ENHANCEMENTS

In common with most larger programming efforts, many interesting problems (both numerical and nonnumerical) have been solved during the course of this thesis. Some conclusions or observations that have been made are:

1. A program has been written that can calculate the flow rates and pressure drops of fluids in piping networks.

2. The program is oriented toward the needs of design engineers.

3. The program should be easily integrable into a commercial design program.

4. MEL is a potentially useful standard user/application interface.

5. C is a viable technical programming language. Modern software design methodologies may be applied to engineering analysis programs.

6. Accurate models for tees can be included in piping network analysis.

7. Due to the unavoidable scatter in experimental data and component interaction in piping systems, hydraulic analysis is not an exact science.

8. Since extreme accuracy is not obtainable in hydraulic analysis, compressible fluids may often be analyzed using Darcy's equation.

Finally, in order to make the program truly appealing to piping design engineers, some or all of the following improvements will need to be made:

1. Add an engineering design user interface. Ideally, this would integrate a metalanguage editor, on-line documentation, tutorial, and expert system to an existing system-standard user interface (GUI) that is already "user-friendly".

2. Insert algorithms for the calculation of the hydraulic resistance of special components such as heat exchangers, separators, etc.

3. Include the capability of economic optimization by repeated analysis.

4. Incorporate complete compressible fluid analysis capabilities using either the ideal gas isothermal equation, the adiabatic equation, or better yet, direct heat transfer calculations.

5. To enhance robustness, allow the nonlinear simultaneous network flow equations be alternatively solved using the Newton-Raphson method. (This would require initial guesses from the user or expert-system.)

6. Adjust the equation forming algorithms so that pressure-reducing valves and one-way check valves may be modeled accurately.

7. Allow the hydraulic resistances of vendor specific components to become a "canned" part of the program. This would include either tables or formulae.

BIBLIOGRAPHY

BIBLIOGRAPHY

Jon Bentley, Programming Pearls, Murry Hill, NJ, Addison-Wesley Publishing Co., 1986. This book contains clear examples showing how software performance (including technical-numerical programs) can be strongly affected by choice of nonnumerical algorithms used in program design.

Jim Gettys, Network Windowing Using the X Window System, Redwood City, CA, Dr. Dobb's Journal, March, 1989. This article describes the X Window environment and how it might be effectively used by software developers.

Volker Gnielinski, Correlations for the Pressure Drop in Helically Coiled Tubes, New York, NY, International Chemical Engineering, Vol. 26, No. 1, January, 1986. This article laments how different engineering handbooks give different equations for the same piping component. It presents how the author standardized the experimental data for one type of component.

I. E. Idelchik, Handbook of Hydraulic Resistance, Second Edition, Moscow, USSR, Hemisphere Publishing Corp., 1986. A much revised and augmented edition over the 1966 translation. This is a very complete and detailed treatment of hydraulic resistance coefficients for mechanical piping systems. However, the nomenclature and parameter selection used in the handbook takes some getting used to, and thus severely limits its value as a quick reference.

David Jeffery. Object-Oriented Programming in ANSI C, San Francisco, CA, Computer Language Magazine, February, 1989. This article describes the object-oriented paradigm, and how to formalize it in a language like C.

Roland W. Jeppson, Analysis of Flow in Pipe Networks, Ann Arbor, MI, Ann Arbor Science Publishers, Inc., 1976. The text considers the computer analysis of flow and pressure drops in large civil piping networks. It contains a description of the Linear Theory method.

James Kempf, Numerical Software Tools In C, Englewood Cliffs, NJ, Prentice-Hall, Inc., 1987. This book contains a brief introduction to both numerical programming and the C programming language. However, the emphasis of the text is on creating small programs that can be used as building blocks for larger programs. The biggest drawback to the book is how it treats doubly dimensioned arrays statically rather than dynamically.

Fred Langa, The End of Application Software?, Highston, NJ, BYTE Magazine, February, 1989. This editorial discusses the latest trends in standardization of the user-interface and speculates on application programs becoming objects, and programming becoming scripts manipulating these objects.

Robert L. Mott, Applied Fluid Mechanics, Second Edition, Columbus, OH, C. E. Merrill Publishing Co., 1979. This textbook contains a very understandable, practical, and elementary introduction to the flow of fluids in pipes.

Lindell E. Ormsbee and Don J. Wood, Hydraulic Design Algorithms for Pipe Networks, Journal of Hydraulic Engineering, Vol. 112, No. 12, December, 1986. Although the paper's title is a bit misleading, this is a good starting point for anyone interested in pursuing this thesis' topic from a different perspective. That is, hydraulic economic optimization by direct solution of a special algorithm rather than repeated analysis with varying economic parameters.

Ken Orr, Chris Gane, Edward Yourdon, Peter P. Chen, and Larry L. Constantine, Methodology: The Experts Speak, Highston, NJ, Byte Magazine, April, 1989. Five prominent software engineers briefly describe the methodologies for which they are famous. The article includes a bibliography for further reading.

William H. Press, et. al., Numerical Recipes in C, Cambridge, UK, Cambridge University Press, 1988. Based on an earlier FORTRAN edition, this is a great cookbook giving a wide range

of oven-tested recipes for the numerical gourmet. It shows the correct way to handle multidimensioned arrays (dynamically). A complaint heard is that some of the algorithms are getting a little obsolete due to the rapid advances in numerical techniques currently being made. The simultaneous linear equation solver used by program HAPN is based on one of the ones presented in this text.

A. J. Ward-Smith, Internal Fluid Flow, Oxford, England, Clarendon Press, 1980. This book is a comprehensive theoretical and experimental survey of fluid flow in pipes, components, and systems. The experimental data presented in the book served as a basis for many of the formulae used in HAPN. It is written at an intermediate to advanced level.

ASTM E 380-82 Standard for Metric Practice, Philadelphia, PA, American Society for Testing Materials,1982. This standard contains many useful conversion factors between English and metric units.

Flow of Fluids Through Valves, Fittings, and Pipe, Technical Paper 410, Chicago, IL, Crane Co. 1978. Since the first edition was published in 1957, Technical Paper 410 has probably become the standard reference for the American piping designer. Its format is such that it is an almost ideal ready reference. Whereas the values for resistance coefficients presented by Crane vary somewhat with data presented in other references (such as the Ward-Smith reference), Technical Paper 410 has had wide use for many years, and is of proven utility. Many of the hydraulic resistance formulae presented here are used by HAPN.

APPENDIXES

APPENDIX A

MEL, a Universal Metalanguage Data Processor

Probably the easiest and most reliable way for a design engineer to enter data or examine output from a computer program would be with a standardized, graphical, interactive user interface. This would allow input to be made as "casually" as possible, and output to be presented in formats most readily understandable to human beings. (The reader has only to look at the Apple Macintosh computer for an example of the potential of such an interface.) A standard interface would also allow the engineer-metamorphosed-programmer to worry less about implementation details.

However, few computers currently have such an interface nor is it expected in the near future that such a standardized interface will exist between different (or even upgraded!) hardware. (Engineering programmers, while technically competent about the problem being solved, are usually not experts in interface design nor do they have the time to write a sophisticated interface of their own.)

Additionally, it is the author's opinion that current user-friendly interface designs have shortcomings with respect to technical programs. Often, engineering analysis programs have voluminous input and output data requirements when run, undergo repeated execution with only minor input modifications, need

independent input and output checking and verification, and require customizable and "pretty" hardcopy for design documentation. These features have not really been addressed, and are therefore difficult to satisfy, in current highly interactive interface designs.

Perhaps the most attractive current solution to this interface problem is to have an engineering analysis program behave in a "batch mode" and get its input and output from files where the data is in a structured pseudo English format (called a metalanguage). Data can be made unambiguous and yet readable by people. And since virtually all computer systems have some sort of text editing capability, data may be examined, checked, and modified relatively easily. (It is even possible and probably desirable that an interactive user interface would generate and examine metalanguage files.) Notice that with careful design, any changes required to the interface would have a lessened effect on the analysis program itself.

The following describes a generic or universal metalanguage processor, called MEL, that can be used for engineering analysis program input and output. The idea is to define a shell that surrounds the program and implements a common format, easily readable, and thus potentially very user-friendly method of inputting and outputting data for almost any analysis program.

The general format for MEL is much like function calls in typical programming languages. That is, an input or output

statement consists of a "descriptor" name (somewhat like a function name), followed by an optional parameter list, followed by an end-of-statement symbol. An unusual feature of MEL is that unlike most programming languages, MEL's parameter list is strongly "free-format". Although descriptor and parameter names and meanings (the dictionary) must be unique for each program, the general structure of MEL does not change from program to program.

Perhaps the best way to explain MEL is with an example. Consider Example A-1, at the end of this appendix, which contains a MEL input file listing for the hydraulic analysis of piping networks program HAPN. Does the reader find it understandable? A big improvement over FORTRAN's fixed field input? Now take a look at Example A-2, also at the end of this appendix, which lists MEL output generated by the input in Example A-1. If the reader can understand the input then he or she should have no problem with the output since it's in the same format.

MEL comments are any characters enclosed in "quotes". They may appear anywhere in the input or output stream, and extend over more than one line. Comments are not processed as input (nor are blank lines). Their purpose is to allow a human to make notes to other humans.

Lines of input or output, called descriptors, are separated by semicolons and thus may extend more than one physical line. The descriptor name and all of its parameters are delimited by commas.

Descriptor and parameter names may be abbreviated as long as they remain unambiguously identifiable to MEL.

Usually, defining a parameter value consists of the parameter's name, an equals sign, its value, and (optionally) its units in parentheses. Thus, the order of a parameter list is not important. However, there are lots of exceptions since MEL allows as much flexibility as possible. That is:

1. If a descriptor has only one parameter associated with it, the parameter's name may be omitted, e.g.:

```
title, 'note that strings are enclosed in single
quotes';
```

2. If all parameter names are missing, a default order is assumed. for example, the following are equivalent:

```
branch, number = 100, from_node = 1, to_node = 2;
branch, 100, 1, 2;
```

3. Parameter array values are enclosed in {brackets} separated by commas, starting with the first index given, for example:

```
grid, x = {1, 2, 3}, y = {1, 2, 3};
grid, x(1) = {1}, x(2) = {2, 3}, y(1) = {1, 2, 3};
```

4. If a parameter is only either true or false, giving its name sets it equal to true, e.g.:

```
analyze, newton_method = true;
analyze, newton_method;
```

5. Missing parameters are given the value they were last defined (or initialized) as having. For example, if a bend descriptor's angle parameter is defined as being 90 degrees, then, if the next bend descriptor's bend angle parameter is missing, the bend angle is assumed to be 90 degrees also.

6. Parameters may be given in any consistent units, e.g.:

```
pipe,length = 12;  "assume default is feet"
pipe,length = 144 (in);
```

7. If a parameter's value is not known, it may be given the value "unknown":

```
node, pressure = unknown;
```

8. If no units are given then the current default units are assumed. For more information on units recognized by MEL, see Appendix B.

The MEL include file `melhapn.h` on the HAPN Source Diskette shows how MEL is implemented for program HAPN and describes how to implement a MEL input or output dictionary for any analysis program. By studying this file, a key concept will be seen to be that the programmer must define an input or output data structure that can accept or promulgate any desired descriptor. These structures are of a standard format. It is through these data structures that MEL handles all external program communication. Finally, note that the programmer may specify the internal units of his or her choice. The internal units of HAPN are standard SI.

Example A-1. Example of a MEL Input File.

```
title, 'Example data for program HAPN';

program_options,
    conver_mispelled_gence_tolerance = .1,
    maximum_iterations = 50;

units,
    pipe_length = 'ft',
    diameter = 'in',
    density = 'lbm/ft3',
    viscosity = 'cp',
    flow = 'lbm/h',
    pressure = 'psi';

fluid,
    density = 62.4,
    viscosity = 1.0;

branch, number = 1, from_node = 1, to_node = 2;
    pipe,
        length = 1600.0,
        id = 18,
        absolute_roughness = .0102;
end_of_branch;

branch, number = 2, from_node = 1, to_node = 2;
    pipe,
        length = 2000.0,
        id = 15.0;
end_of_branch;

branch, number = 3, from_node = 1, to_node = 2;
    pipe,
        length = 2400.0,
        id = 18.0;
end_of_branch;

node, number = 1, pressure = 15.0, flow = +1000000.0;

node, number = 2, flow = -1000000.0;
```

Example A-2. Example of a MEL Output File.

```
     program,
          name = 'HAPN - Hydraulic Analysis of Piping
Networks',
          problem_title = 'Example data for program HAPN';
     message,
          text = 'Date: Fri Feb 03 14:20:59 1989';
     message,
          text = 'Input filename: input';
     equations,
          node = 1,
          loop = 2,
          iterations = 8;
     branch,
          number = 1,
          type = 'dependent_branch',
          flow_rate = 424598 (lbm/h),
          flow_change = 0.00345603 (%),
          flow_dp = 0.162763 (psi),
          elevation_dp = 0 (psi);
     branch,
          number = 2,
          type = 'dependent_branch',
          flow_rate = 232618 (lbm/h),
          flow_change = -0.0157368 (%),
          flow_dp = 0.162732 (psi),
          elevation_dp = 0 (psi);
     branch,
          number = 3,
          type = 'dependent_branch',
          flow_rate = 342784 (lbm/h),
          flow_change = 0.00640057 (%),
          flow_dp = 0.162768 (psi),
          elevation_dp = 0 (psi);
     component,
          branch_number = 1,
          component_number = 0,
          resistance = 0.392096 (Pa/kg2/s2),
          change_resistance = -3.1652e-005 (%);
     component,
          branch_number = 2,
          component_number = 0,
          resistance = 1.30611 (Pa/kg2/s2),

    change_resistance = 0.000158481 (%);
```

Example A-2 (Continued)

```
component,
    branch_number = 3,
    component_number = 0,
    resistance = 0.601618 (Pa/kg2/s2),
    change_resistance = -6.12949e-005 (%);
node,
    number = 1,
    pressure = 15 (psi);
node,
    number = 2,
    pressure = 14.8372 (psi);

next;
```

Appendix B

Units

When preparing a user input file, many descriptor parameters will require specification of their units. For example the descriptor:

```
pipe, length = 1.25 (mi), dy = 13. (ft), id = 48. (in);
```

describes a pipe 1.25 miles long, that rises 13 feet along its length, and that has an internal diameter of 48 inches.

If the current default units are not convenient, the program user will need to put in the units wanted in parentheses. (Please also see the "units" descriptor in the MEL-HAPN Input Data Dictionary in Appendix C.)

The following is a list of the units recognized by MEL. MEL can automatically convert from one set of units to another set (say, "m" to "ft"), within a given type (in this case: LENGTH). When choosing units, it is important the user spells them exactly as they appear within the quotation marks.

LENGTH:

"m"	meter
"angstrom"	
"ast-unit"	astronomical unit
"chain"	(US survey ft)
"fathom"	(US survey ft)
"fermi"	

"ft"	foot
"survey-ft"	US survey ft
"in"	inch
"light-year"	
"micro-in"	microinch
"micron"	
"mil"	
"naut-mi"	nautical mile
"mi"	statute mile (US survey foot)
"parsec"	
"pica"	(printer's)
"point"	(printer's)
"rod"	(US survey foot)
"yd"	yard

ACCELERATION:

"m/s2"	meter / second squared
"ft/s2"	foot / second squared
"g"	standard free-fall
"in/s2"	inch / second squared

ANGLE:

"rad"	radian
"deg"	degree
"min"	minute
"sec"	second
"grad"	gradian

AREA:

"m2"	meter squared
"acre"	(US survey foot)
"ft2"	foot squared
"ht"	hectare
"in2"	inches squared
"mi2"	mile squared (US statute)
"yd2"	yard squared

TORQUE:

"N*m"	newton-meter
"dyne*cm"	dyne-centimeter
"kgf*m"	kilogram(force)-meter

"ozf*in"	ounce(force)-inch
"lbf*in"	pound(force)-inch
"lbf*ft"	pound(force)-foot

DENSITY:

"kg/m3"	kilogram / cubic meter
"g/cm3"	gram / cubic centimeter
"ozm/gal"	ounce(mass) / gallon (US liquid)
"ozm/in3"	ounce(mass) / cubic inch
"lbm/ft3"	pound(mass) / cubic foot
"lbm/in3"	pound(mass) / cubic inch
"lbm/gal"	pound(mass) / gallon
"slug/ft3"	slug / cubic foot

ENERGY & WORK:

"J"	joule
"Btu"	(Inter. Table)
"cal"	thermochemical
"ev"	electronvolt
"erg"	
"ft*lbf"	foot-pound(force)
"ft*poundal"	foot-poundal
"kW*h"	kilowatt-hour
"therm"	
"ton-TNT"	(nuclear equiv. TNT)
"W*h"	watt-hour

POWER / UNIT AREA:

"W/m2"	watt / square meter
"Btu/ft2/s"	Btu / square foot / second
"Btu/ft2/h"	Btu / square foot / hour
"cal/cm2/s"	(thermochemical)
"W/cm2"	watt / square centimeter
"W/in2"	watt / square inch

FORCE:

"N"	newton
"dyne"	
"kgf"	kilogram(force)
"kip"	
"ozf"	ounce(force)

"lbf"	pound(force)
"poundal"	

THERMAL CONDUCTIVITY:

"W/m/K"	watt / meter / degree Kelvin
"Btu/h/ft/F"	Btu / hour / foot / degree F
"Btu*in/h/ft2/F"	
"Btu*in/s/ft2/F"	

HEAT CAPACITY:

"J/kg/K"	Joule / kilogram / degree K
"Btu/lbm/F"	Btu / pound(mass) / degree F

MASS:

"kg"	kilogram
"gr"	grain
"gm"	gram
"kgf*s2/m"	kilogram(force)-sq. sec. / meter
"ozm"	ounce-mass
"lbm"	pound-mass
"slug"	

MASS / UNIT LENGTH:

"kg/m"	kilogram / meter
"denier"	
"lbm/ft"	pound(mass) / foot
"lbm/in"	pound(mass) / inch
"tex"	

MASS / UNIT TIME:

"kg/s"	kilogram / second
"lbm/h"	pound(mass) / hour
"lbm/min"	pound(mass) / minute
"lbm/s"	pound(mass) / second

POWER:

"W"	watt
"Btu/h"	Btu / hour
"Btu/s"	Btu / second
"cal/s"	calorie / second
"erg/s"	erg / second
"ft*lbf/h"	foot-pound(force) / hour

"ft*lbf/min"	foot-pound(force) / minute
"ft*lbf/s"	foot-pound(force) / second
"hp"	(550 ft*lbf/s)
"boiler-hp"	
"elec-hp"	(electric)
"ton-refrig"	(12000 Btu/hr)

PRESSURE OR STRESS:

"Pa"	Pascal
"std-atm"	standard atmosphere
"bar"	
"ftH2O"	ft of water 39.2F
"inHg"	in of mercury 32F
"inH2O"	in of water 39.2F
"kgf/m2"	kilogram(force) / square meter
"kip/in2"	kip / square inch
"millibar"	
"torr"	mmHg 0C
"poundal/ft2"	poundal / square foot
"lbf/ft2"	pound(force) / square foot
"psi"	pound(force) / square inch

VELOCITY:

"m/s"	meter / second
"ft/h"	foot / hour
"ft/min"	foot / minute
"ft/s"	foot / second
"in/s"	inch / second
"km/h"	kilometer / hour
"knot"	(international)
"mi/h"	(international)

DYNAMIC VISCOSITY:

"Pa*s"	pascal-second
"cp"	centipoise
"poundal*s/ft2"	poundal-second / square foot
"lbm/ft/h"	pound(mass) / foot / hour
"lbm/ft/s"	pound(mass) / foot / second
"lbf*s/ft2"	pound(mass)-second / square
foot	

"lbf*s/in2"	pound(mass)-second / square inch
"slug/ft/s"	slug / foot / second

KINEMATIC VISCOSITY:

"m2/s"	square meter / second
"cs"	centistokes
"ft2/s"	square foot / second

VOLUME:

"m3"	meter cubed
"bbl"	barrel (oil 42 gal)
"board-ft"	board foot
"bushel"	(US)
"cup"	
"oz"	fluid ounce (US)
"ft3"	foot cubed
"gal"	gallon (US)
"in3"	cubic inch
"l"	liter
"pint"	(US)
"qt"	quart (US)
"tbl"	tablespoon
"tsp"	teaspoon
"yd3"	cubic yard

VOLUMETRIC FLOW:

"m3/s"	cubic meter / sec
"cfm"	cubic feet / minute
"cfs"	cubic feet / second
"in3/min"	cubic inch / minute
"gpm"	gallons / minute

APPENDIX C

MEL-HAPN Input Dictionary

The following defines a MEL input dictionary for program HAPN - Hydraulic Analysis of Piping Networks. Descriptors are defined followed by brief explanations. In the following, `nnn` is a number, `'sss'` is a string (may start with a digit and contain blanks), `ccc` is a variable name (must start with a letter and no blanks). Initial default units are shown in parentheses. However, once units are changed, they become the new default units.

Note that not all descriptors or parameter options have as yet been implemented in HAPN. Those that have not yet been implemented are marked beginning with an asterisk. (The complete dictionary has been defined in anticipation of the "full-featured" commercial version of the program.)

```
bend, radius = nnn1 (in), angle = nnn2 (deg);
```

Specifies a bend component. The `radius` parameter is the radius of center of curvature of the bend. The `angle` parameter is the total angle the bend makes. Default values are `radius` = same as a long radius elbow (1.5 times internal pipe diameter), `angle` = 90 deg.

```
branch, number = nnn1, from_node = nnn2, to_node =
nnn3, flow_guess = nnn4, known_flow_flag = ccc;
```

Specifies the start of a new branch. All of the components that constitute a branch must follow, on succeeding lines, the branch descriptor. (Components should follow in the order in which they occur.) A branch ends with the `end_of_branch` descriptor. An estimate for flow is not required if the linear theory solution method is being used. The `known_flow_flag` variable, if set to `true` by the user, indicates that the `flow_guess` is the branch's actual fixed flow rate.

```
*catalog_item, item = 'sss';
```

Allows the user to input the resistance coefficient (k-factor) of a special component by referencing a catalog item name. `'sss'` is the name of the catalog item.

```
default_data, id = nnn1 (in), nps = nnn2 (in),
schedule_number = ccc1, material = ccc2, absolute_roughness
= nnn3 (in);
```

A pipe component (length of pipe) may not be the first component encountered on a branch. Yet many types of other components have resistance coefficient formulae that depend on a reference internal diameter and/or internal pipe roughness. This descriptor allows this reference data to be entered. For an explanation of parameters, see the pipe descriptor below.

```
end_of_branch;
```

Specifies the end of the current branch. All components on a branch must be declared after its branch descriptor and before this descriptor.

```
enlarger, id = nnn1 (in), nps = nnn2 (in),
schedule_number = ccc, angle = nnn3 (deg);
```

Describes a transition to a larger pipe diameter. The user specifies the new pipe diameter by inputting either `id` or `nps` and `schedule_number`. The `angle` parameter is the average total angle of divergence from the old wall diameter to the new. Default angle is 90 (deg). (An angle of 180 degrees would equal a sudden enlargement.) The length of the component is not considered and must be added separately if important.

```
equivalent_length, ratio = nnn;
```

This descriptor causes the program to calculate the resistance coefficient of the component based on equivalent length ratio. This ratio is an "equivalent" length of pipe divided by its internal diameter. Thus, for example, an equivalent length ratio of 100 would mean a component has the resistance equivalent to 100 pipe diameters. HAPN uses the current value of pipe internal diameter as the reference diameter.

```
fluid, name = 'sss', temperature = nnn1 (F), density =
nnn2 (lbm/ft3), viscosity = nnn3 (cp), thermal_conductivity
= nnn4 (Btu/hr/ft/F), heat_capacity = nnn5 (Btu/lbm/F);
```

Used to specify a piping fluid's properties by either referencing its `name` and `temperature` (not yet implemented) or inputting their values directly. The `thermal_conductivity` and `heat_capacity` data are required only if heat transfer must be calculated (not required for Darcy's equation).

```
mitre_elbow, angle = nnn1 (deg), number_of_mitres =
nnn2;
```

Used to input a mitre bend component. The `angle` parameter is the total bend angle. The distance between mitres is assumed to be equal to the pipe's internal diameter.

```
next;
```

This descriptor allows problems to be stacked for sequential execution. That is, input after this descriptor is treated as a new problem.

```
node, number = nnn1, flow = nnn2 (gpm), pressure = nnn3
(psig);
```

Specifies the external pressure or external flow rate at or into a node. Positive external flow is defined as being into the node.

The default flow rate is zero, the default nodal pressure is unknown (that is, the program will solve for this value).

```
pipe, length = nnn1 (ft), dy = nnn2 (ft), nps = nnn3
(in), schedule_number = ccc1, id = nnn4 (in), material =
ccc2, absolute_roughness = nnn5 (in), heat_transfer_type =
ccc3, wall_temperature = nnn6 (F);
```

Describes a section of straight pipe of constant diameter to the program.

The program recognizes the following pipe materials of construction, all of which have an associated absolute_roughness: riveted_steel, concrete, cast_iron, galvanized_pipe, asphalted_pipe, steel, pvc, drawn_tubing. Thus, if the material parameter is specified, there is no need to specify absolute_roughness and vice versa. Note that absolute_roughness default units are diameters.

The heat_transfer parameter may be either: darcy, isothermal, adiabatic, or calculate. If calculate is chosen, the wall_temperature must also be input. (So far, only darcy has been implemented and is the default.)

Notice that none of the other piping components are assumed to have any significant vertical length. If the elevation change of any of these other components is significant, input a "fictional" pipe length for them using this component.

```
*pressure_valve, pressure = nnn (psig);
```

A pressure-reducing or pressure-sustaining value may be specified.

```
     program_options, solution_method = ccc1,
convergence_tolerance = nnn1 (%), maximum_iterations = nnn2,
output_format = ccc2, output_file = 'sss1';
```

This descriptor tells the program how to obtain a solution. Allowable solution methods are: `linear_theory` and `newton_raphson`. (Only `linear_theory` has been implemented so far and is the default.) Allowable output formats are: `print` (regular output), `mel_short` (MEL output without parameter identifiers to specified file), and `mel_long` (MEL output with parameter identifiers to specified file).

```
     pump, curve = 'sss', impeller_size = nnn1 (in), flow1 =
nnn2 (gpm), head1 = nnn3 (ft), flow2 = nnn4 (gpm), head2 =
nnn5 (ft);
```

Describes a centrifugal pump to the program. Two methods are available for inputting pump characteristics. The name of a previously stored pump curve and its impeller size may be described (but this feature has not yet been implemented). Else, two points from the pump curve may be input, and the program will figure its own pump equation coefficients.

```
     reducer, id = nnn1 (in), nps = nnn2 (in),
schedule_number = ccc, angle = nnn3 (deg);
```

Describes a reduction in pipe size. The parameter list is the same as described for an enlarger descriptor.

```
resistance_coefficient, k_factor = nnn;
```

Allows the user to input the resistance coefficient or k-factor of any special component directly. See the technical background section for more information on resistance coefficients.

```
tank_nozzle, type = ccc, transition_radius = nnn (in);
```

Takes care of the hydraulic losses associated with tank entrances or exits. Allowable types are: `entrance`, `exit`, and `flush_exit`. The `transition_radius` parameter, required only for `flush_exit`, is the average fillet radius of the transition from the inside tank wall to the inside wall of the nozzle.

```
tee, type = ccc, node_number = nnn1, main1 = nnn2,
main2 = nnn3, lateral = nnn4, fillet_radius = nnn5 (in),
angle = nnn6 (deg);
```

The hydraulic analysis of a piping network is complicated by the fact that the resistance of a tee is strongly affected by the flow split between its various branches. (For example, flow mostly through a main can actually siphon flow from its lateral.) A further complication is that the boundary of a model of a piping system is

often at a tee. The type parameter may be: `thru` (for straight flow through a nonnode tee), `side_out` (for turning flow through a non-node tee), or `normal` (the default for a node where three branches meet). For `thru` or `side-out` tees, no other information is required. The `main1`, `main2`, and `lateral` parameters tell the program how the tee is physically oriented. (A tee is composed of a main, where two branches meet straight-on, and a lateral.) The `fillet_radius` parameter gives the radius of curvature between the main and lateral, and the `angle` parameter gives the angle between the main1 branch and the lateral.

```
title, string = 'sss';
```

This descriptor allows problem output to be easily identified.

```
units, file = 'sss1', pipe_lengths = 'sss2', diameters
= 'sss3', radii = 'sss4', angles = 'sss5', densities =
'sss6', viscosities = 'sss7', temperatures = 'sss8',
heat_transfer_rates = 'sss9', flow_coefficients = 'sss10';
```

Allows the user to select default units. It changes the units associated with the parameters of all other descriptors that follow it. `'sss1'` is a file name containing the conversion factors (not yet implemented). Else, the appropriate units may be entered directly. Units not appearing are either given the values specified in `'sss1'`, or the units given in the above parameter descriptions. Note that `pipe_length` units are the same as `absolute_roughness` units.

```
    *user_defined_component, function_name = ccc,
arguements = nnn1 ...;
```

This descriptor allows the user to indirectly call and pass arguements to a user written function. The function must return a value the program can interpret as a resistance coefficient (k-factor). The program assumes the function prototype to be:

```
    double function_name(struct function_data x, double
arg1, ...);
```

The structure function_data contains useful data such as branch flow rate, fluid density, and current reference pipe diameter. See HAPN source code for further details.

```
    valve, type = ccc, cv = nnn2 (gpm/psi)
```

Describes a valve component to the program. If the flow coefficient of the valve (`cv`) is known, it should be specified. If it is not known, input the valve's `type` and the program will estimate a resistance coefficient for the user. Allowable valve types are: `angle, ball, butterfly, swing_check, gate, globe, plug,` and `stop_check`.

APPENDIX D

MEL-HAPN Output Dictionary

The following defines a MEL output dictionary for program HAPN:

```
branch, number = nnn1, flow = nnn2 (gpm);
```

This descriptor outputs calculated branch flow rates.

```
component, branch_number = nnn1, component_number =
nnn2, hydraulic_resistance = nnn3 (psid/gpm2), elevation_dp
= nnn4 (psid);
```

This descriptor outputs calculated component hydraulic resistances and elevation pressure drops. The first component encountered on the branch during input is given the component number zero, and subsequent components on the branch are given sequential integer numbers. Note that a hydraulic resistance is not the same thing as a k-factor.

```
equations, node = nnn1, loop = nnn2, convergence_error
= nnn3 (%), iterations = nnn4;
```

Informs the user of the number of linear node and nonlinear loop equations that the program generated. It also outputs the

number of iterations and the convergence error resulting from the numerical solution of the equations.

```
message, code = nnn, text = 'sss';
```

Informs the user of a program message such as a warning or error. Each message has an associated message code.

```
next;
```

Separates output between stacked runs.

```
node, number = nnn1, pressure = nnn2 (psid),
external_flow = nnn3 (gpm);
```

This descriptor outputs calculated nodal external pressures and flows.

```
program, name = 'sss';
```

Identifies the name of the program that generated the output.

```
tee, number = nnn1, pressure = nnn2 (psig), hr12 = nnn6
(psid/gpm2), hr13 = nnn7 (psid/gpm2), hr23 = nnn8
(psid/gpm2);
```

A `thru` or `side_out` tee is output as an ordinary piping component. However, a `normal` tee doubles as a node and as a complicated source of pressure drop. Parameters `hr12`, `hr13`, `hr23`

are the hydraulic resistances between branches `main1` and `main2`, `main1` and the `lateral`, and `main2` and the `lateral` respectively.

```
title, string = 'sss';
```

Echo of title string just read from the input file.

APPENDIX E

Description of HAPN Source Diskette

The diskette included at the back of this thesis contains source code and other related data for program HAPN - a hydraulic analysis of piping networks program. The diskette is MS-DOS format and thus readable by IBM PC compatible computers. The following files are included on the diskette:

readme.doc
hapn.mak
melhapn.h
melio.c
utility.h
utility.c
hapnincl.h
hapn1.c
hapn2.c
hapn2_1.c
hapn3.c
hapn4.c
hapn5.c
nsptbl.dat
eqtest.c
hapn.pc
screens.doc
bugs.doc
exprob1.in
exprob1.out
exprob2.in
exprob2.out

In the event the diskette is missing, a copy of it may be obtained from the Engineering Science and Mechanics Department Office.

APPENDIX F

Example Problems

This appendix gives illustrative example problems that describe how HAPN works. Input and output files for these problems are included on the HAPN Source Diskette. The first example describes a simple single branched piping system and the second problem describes a typical multibranched network. Sketches for these problems are included at the end of this appendix.

Example Problem One

As shown on Sketch F-1, this problem describes fluid being pumped from one tank to another. Notice that both tanks are open to air so that atmospheric pressure cancels. Besides piping component resistance, the pump must overcome a twenty-five foot change in elevation minus the five foot height differential in tank fluid levels. Input data for this problem is as follows:

```
title, 'Example Problem Number One';

program_options,
    convergence_tolerance = 0.1,
    maximum_iterations = 50;

units,
```

```
    pipe_length = 'ft',
    diameter = 'in',
    angle = 'deg',
    density = 'lbm/ft3',
    viscosity = 'cp',
    flow = 'lbm/h',
    pressure = 'psi';

fluid, den = 62.4, vis = 1.0;  "water"

node, 1, pressure = 8.6667;  "20 ft of water"

default_data, nps = 1.5, sch = '10S', material =
'steel';

branch, 1, from = 1, to = 2;
    tank_nozzle, 'exit';
    valve, 'gate';
    pipe, len = 50, dy = -20;
    tee, 'side_out';
    pump, flow1 =  25000, head1 = 110,
          flow2 =  50000, head2 =  80;
    valve, cv = 100 (gpm/psi);
    pipe, len = 120, dy = 45;
    bend;
    resistance_coefficient, 5;
    bend;
    valve, 'gate';
    tank_nozzle, type = 'entrance';
end_of_branch;

node, 2, pressure = 6.5;   "15 ft of water"
```

The above data was saved as a text file called "exprob1.in". HAPN was then executed and given this file name as the first command line parameter. The following output was generated:

```
program,
    name = 'HAPN - Hydraulic Analysis of Piping
Networks',
    problem_title = 'Example Problem Number One';
message,
    text = 'Date: Sun Mar 26 17:55:40 1989';
message,
    text = 'Input filename: exprob1.in';
equations,
    node = 0,
    loop = 0,
    iterations = 7;
```

```
branch,
     number = 1,
     type = 'independent_branch',
     flow_rate = 40024.9 (lbm/h),
     flow_change = -0.0955007 (%),
     flow_dp = 32.2474 (psi),
     elevation_dp = -30.0807 (psi);
component,
     branch_number = 1,
     component_number = 0,
     type = 'tank_nozzle',
     resistance = 243.414 (Pa*s2/kg2),
     change_resistance = 0 (%),
     pressure_drop = 0.897871 (psi);
component,
     branch_number = 1,
     component_number = 1,
     type = 'valve',
     resistance = 38.8697 (Pa*s2/kg2),
     change_resistance = 0 (%),
     pressure_drop = 0.143377 (psi);
component,
     branch_number = 1,
     component_number = 2,
     type = 'pipe',
     resistance = 1897.9 (Pa*s2/kg2),
     change_resistance = 0.000210167 (%),
     pressure_drop = 7.00071 (psi);
component,
     branch_number = 1,
     component_number = 3,
     type = 'tee',
     resistance = 291.522 (Pa*s2/kg2),
     change_resistance = 0 (%),
     pressure_drop = 1.07533 (psi);
component,
     branch_number = 1,
     component_number = 4,
     type = 'pump',
     resistance = 0 (Pa*s2/kg2),
     change_resistance = 0 (%),
     pressure_drop = 0 (psi);
component,
     branch_number = 1,
     component_number = 5,
     type = 'valve',
     resistance = 173.54 (Pa*s2/kg2),
     change_resistance = 0 (%),
     pressure_drop = 0.640128 (psi);
component,
```

```
    branch_number = 1,
    component_number = 6,
    type = 'pipe',
    resistance = 4554.97 (Pa*s2/kg2),
    change_resistance = 0.000210167 (%),
    pressure_drop = 16.8017 (psi);
component,
    branch_number = 1,
    component_number = 7,
    type = 'bend',
    resistance = 48.1519 (Pa*s2/kg2),
    change_resistance = 0 (%),
    pressure_drop = 0.177616 (psi);
component,
    branch_number = 1,
    component_number = 8,
    type = 'resistance_coefficient',
    resistance = 1217.07 (Pa*s2/kg2),
    change_resistance = 0 (%),
    pressure_drop = 4.48936 (psi);
component,
    branch_number = 1,
    component_number = 9,
    type = 'bend',
    resistance = 48.1519 (Pa*s2/kg2),
    change_resistance = 0 (%),
    pressure_drop = 0.177616 (psi);
component,
    branch_number = 1,
    component_number = 10,
    type = 'valve',
    resistance = 38.8697 (Pa*s2/kg2),
    change_resistance = 0 (%),
    pressure_drop = 0.143377 (psi);
component,
    branch_number = 1,
    component_number = 11,
    type = 'tank_nozzle',
    resistance = 189.863 (Pa*s2/kg2),
    change_resistance = 0 (%),
    pressure_drop = 0.70034 (psi);
node,
    number = 1,
    pressure = 8.6667 (psi);
node,
    number = 2,
    pressure = 6.5 (psi);
next;
```

Example Problem Two

This problem is a simple hydraulic network as illustrated in Sketch F-2. Input data for this problem is as follows:

```
title, 'Example Problem Number Two';

program_options, convergence_tolerance = 0.1,
     maximum_iterations = 50;

fluid, den = 62.4 (lbm/ft3), vis = 1.0 (cp);
                                        "water @ 68 F"

branch, number = 1, from_node = 1, to_node = 2;
     pipe, 1600 (ft), id = 18 (in),
     absolute_roughness = 0.0102 (in);
end;
branch, 2, 2, 3; pipe, 2000, id = 15; end;
branch, 3, 1, 3; pipe, 2400, id = 18; end;
branch, 4, 2, 4; pipe, 1800, id = 12; end;
branch, 5, 4, 3; pipe, 1900; end;
branch, 6, 5, 4; pipe, 1300, id = 10, abs = 0.0090;
end;
branch, 7, 5, 6; pipe, 1700, id = 15, abs = 0.0102;
end;
branch, 8, 6, 3; pipe, 2000, id = 18, abs = 0.0090;
end;
branch, 9, 5, 7; pipe, 1200, id = 24, abs = 0.0102;
end;
branch,10, 7, 6; pipe, 1800, id = 15; end;

node, number = 1, flow = 10.0e5 (lbm/h),
     pressure = 14.7 (psi);
node, 2, -1.5e5;
node, 3, -4.5e5;
node, 4, -2.5e5;
node, 5,  7.5e5;
node, 6, -4.0e5;
node, 7, -5.0e5;
```

Output generated from this data is as follows:

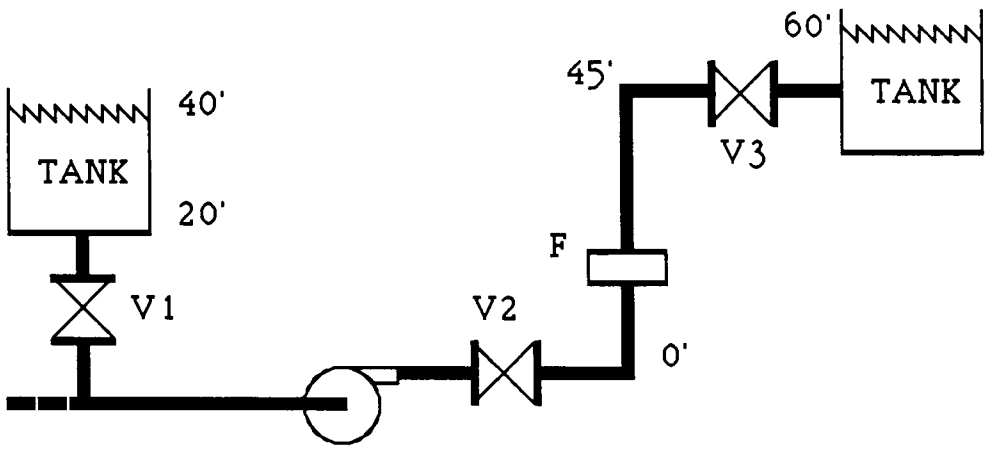
```
program,
    name = 'HAPN - Hydraulic Analysis of Piping
Networks',
    problem_title = 'Example Problem Number Two';
message,
    text = 'Date: Thu Mar 30 13:45:50 1989';
message,
    text = 'Input filename: exprob2.in';
equations,
    node = 6,
    loop = 4,
    iterations = 8;
branch,
    number = 1,
    type = 'dependent_branch',
    flow_rate = 492163 (lbm/h),
    flow_change = -0.00342964 (%),
    flow_dp = 0.215605 (psi),
    elevation_dp = 0 (psi);
branch,
    number = 2,
    type = 'dependent_branch',
    flow_rate = 204512 (lbm/h),
    flow_change = -0.0127366 (%),
    flow_dp = 0.127713 (psi),
    elevation_dp = 0 (psi);
branch,
    number = 3,
    type = 'dependent_branch',
    flow_rate = 507837 (lbm/h),
    flow_change = 0.00332401 (%),
    flow_dp = 0.343354 (psi),
    elevation_dp = 0 (psi);
branch,
    number = 4,
    type = 'dependent_branch',
    flow_rate = 137651 (lbm/h),
    flow_change = 0.00666305 (%),
    flow_dp = 0.166804 (psi),
    elevation_dp = 0 (psi);
branch,
    number = 5,
    type = 'dependent_branch',
    flow_rate = -61098.3 (lbm/h),
    flow_change = -0.0098157 (%),
    flow_dp = -0.0390595 (psi),
```

```
    elevation_dp = 0 (psi);
branch,
    number = 6,
    type = 'dependent_branch',
    flow_rate = 51250.2 (lbm/h),
    flow_change = -0.00619152 (%),
    flow_dp = 0.0469753 (psi),
    elevation_dp = 0 (psi);
branch,
    number = 7,
    type = 'dependent_branch',
    flow_rate = 145340 (lbm/h),
    flow_change = -0.0056335 (%),
    flow_dp = 0.057359 (psi),
    elevation_dp = 0 (psi);
branch,
    number = 8,
    type = 'dependent_branch',
    flow_rate = -201250 (lbm/h),
    flow_change = -0.0015768 (%),
    flow_dp = -0.0494466 (psi),
    elevation_dp = 0 (psi);
branch,
    number = 9,
    type = 'dependent_branch',
    flow_rate = 553410 (lbm/h),
    flow_change = 0.00205305 (%),
    flow_dp = 0.0476153 (psi),
    elevation_dp = 0 (psi);
branch,
    number = 10,
    type = 'dependent_branch',
    flow_rate = 53409.6 (lbm/h),
    flow_change = 0.0212771 (%),
    flow_dp = 0.00975004 (psi),
    elevation_dp = 0 (psi);
component,
    branch_number = 1,
    component_number = 0,
    type = 'pipe',
    resistance = 0.386575 (Pa*s2/kg2),
    change_resistance = 3.18621e-005 (%),
    pressure_drop = 0.215605 (psi);
component,
    branch_number = 2,
    component_number = 0,
    type = 'pipe',
    resistance = 1.32615 (Pa*s2/kg2),
    change_resistance = 0.000138076 (%),
    pressure_drop = 0.127713 (psi);
```

```
component,
    branch_number = 3,
    component_number = 0,
    type = 'pipe',
    resistance = 0.57821 (Pa*s2/kg2),
    change_resistance = -3.03509e-005 (%),
    pressure_drop = 0.343354 (psi);
component,
    branch_number = 4,
    component_number = 0,
    type = 'pipe',
    resistance = 3.82329 (Pa*s2/kg2),
    change_resistance = -5.39281e-005 (%),
    pressure_drop = 0.166804 (psi);
component,
    branch_number = 5,
    component_number = 0,
    type = 'pipe',
    resistance = 4.54423 (Pa*s2/kg2),
    change_resistance = 0.000113642 (%),
    pressure_drop = -0.0390595 (psi);
component,
    branch_number = 6,
    component_number = 0,
    type = 'pipe',
    resistance = 7.76732 (Pa*s2/kg2),
    change_resistance = 6.84622e-005 (%),
    pressure_drop = 0.0469753 (psi);
component,
    branch_number = 7,
    component_number = 0,
    type = 'pipe',
    resistance = 1.17929 (Pa*s2/kg2),
    change_resistance = 5.05282e-005 (%),
    pressure_drop = 0.057359 (psi);
component,
    branch_number = 8,
    component_number = 0,
    type = 'pipe',
    resistance = 0.53022 (Pa*s2/kg2),
    change_resistance = 1.53621e-005 (%),
    pressure_drop = -0.0494466 (psi);
component,
    branch_number = 9,
    component_number = 0,
    type = 'pipe',
    resistance = 0.067522 (Pa*s2/kg2),
    change_resistance = -1.50373e-005 (%),
    pressure_drop = 0.0476153 (psi);
component,
```

```
    branch_number = 10,
    component_number = 0,
    type = 'pipe',
    resistance = 1.48443 (Pa*s2/kg2),
    change_resistance = -0.000274906 (%),
    pressure_drop = 0.00975004 (psi);
node,
    number = 1,
    pressure = 14.7 (psi);
node,
    number = 2,
    pressure = 14.4844 (psi);
node,
    number = 3,
    pressure = 14.3567 (psi);
node,
    number = 4,
    pressure = 14.3176 (psi);
node,
    number = 5,
    pressure = 14.3646 (psi);
node,
    number = 6,
    pressure = 14.3072 (psi);
node,
    number = 7,
    pressure = 14.317 (psi);
next;
```

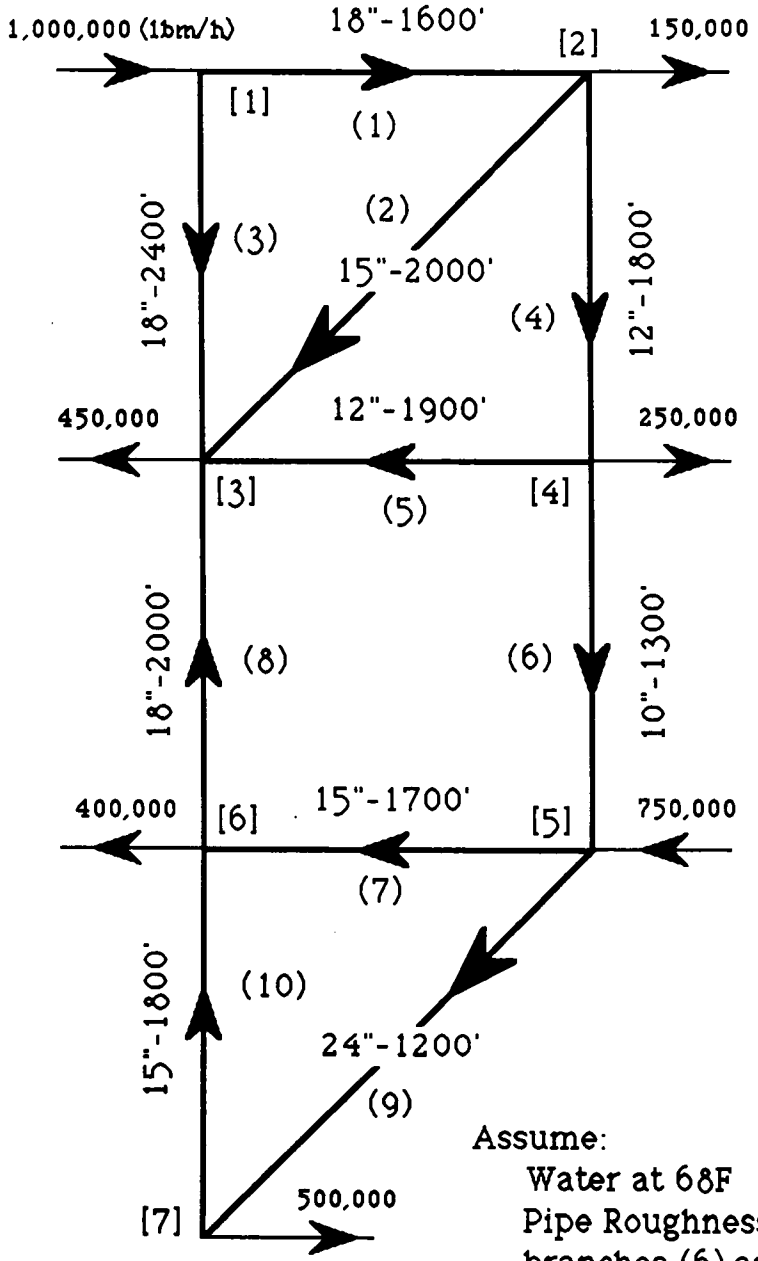
Sketch F-1. Sketch for Example Problem One



Data:

Pipe = 1.5" NPS, Sch. 10S, Steel
Suction Line Length = 50'
Discharge Line Length = 120'
Fluid = Water at 68F
V2 = Ball Valve (C_V = 100)
V1, V3 = Gate Valves
F = Flow Meter (K = 5.0)
Pump = Develops 110 Ft. of Head at 50 GPM
Develops 80 Ft. of Head at 100 GPM.

Sketch F-2. Sketch for Example Problem Two



Assume:
Water at 68F
Pipe Roughness = 0.012" except branches (6) and (8) whose roughness = 0.009"

VITA

My full name is George Michael Crews and I was born in Lake Providence, Louisiana on July 23, 1950. My family moved to Las Vegas, Nevada in 1962 and I lived there until graduating from the University of Nevada, Las Vegas in 1972, receiving a Bachelor of Science degree in General Engineering.

Over the years, I have worked as a mechanical engineer or software developer for several companies around the United States. These include the Ralph M. Parsons Company, Monsanto Company, Union Carbide Incorporated (Nuclear Division), EG&G Idaho Incorporated, Pixel Software, EG&G Services Incorporated, and Lockwood Greene Engineers.

In addition to doing practical work, I have always had an interest in extending my education and have attended California State University at Los Angeles, the University of Missouri, the University of Idaho, and the University of Tennessee.

I am a registered professional engineer in mechanical engineering in the states of Kentucky and Nevada.