1990 NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

JOHN F. KENNEDY SPACE CENTER
UNIVERSITY OF CENTRAL FLORIDA

p20

MODELING OF FLOW SYSTEMS FOR IMPLEMENTATION UNDER KATE

PREPARED BY: Dr. Jonathan E. Whitlow

ACADEMIC RANK: Assistant Professor

UNIVERSITY AND DEPARTMENT: Florida Institute of Technology
Department of Chemical Engineering

NASA/KSC

DIVISION: Data Systems

BRANCH: Technical and Information Systems

NASA COLLEAGUE: Ms. Carrie Belton

DATE: August 30, 1990

CONTRACT NUMBER: University of Central Florida
NASA-NGT-60002 Supplement: 4

ABSTRACT

The modeling of flow systems is a task currently being investigated at Kennedy Space Center in parallel with the development of the KATE artificial intelligence system used for monitoring diagnosis and control. This report focuses on various aspects of the modeling issues with particular emphasis on a water system scheduled for demonstration within the KATE environment in September of this year. LISP procedures were written to solve the continuity equations for three internal pressure nodes using Newton's method for simultaneous nonlinear equations.

SUMMARY

KATE is a model-based expert system being developed at Kennedy Space Center for the monitoring, diagnosis and control of launch operations. This work focused primarily on modeling issues associated with the construction of the knowledge-base for a water tanking demonstration system.

In order to accurately predict expected values of various sensor readings in the water system for various flow configurations, the model used for simulation by KATE had to be modified. The original model of the flow network used a single mass continuity equation which required an iterative solution for an internal pressure node at each increment of time. In order to sufficiently model the process for certain combinations of valve positions, three mass continuity equations were needed. As a consequence, the values of three unknown internal pressure nodes were then required for solution of the resulting nonlinear equations at each time step.

Various methods were examined for the solution of the three simultaneous nonlinear equations. Newton's method was ultimately chosen for implementation in the KATE knowledge-base due to its speed of convergence. In general, Newton's method requires an initial starting value for the solution vector which is sufficiently close to the actual solution. Testing of the method indicated that initial guesses provided within physical constraints of the system, converged. The concern over guaranteed convergence resulted in the more stable Steepest Descent minimization method to also be examined. The time required for convergence of this method however was more than an order of magnitude greater than Newton's method. Both algorithms were encoded in LISP for use by KATE.

Additional LISP code had to be included in the procedure written to perform Newton's method in order to prevent the generation of error messages. The source of the error messages came from discontinuity in the modeling equations which would result from various valve positions. The discontinuities resulted in a singular matrix being formed in the program and hence the resulting errors. The method used to avoid the error generation in effect looked at all possible valve combinations and flow conditions and dealt with each as necessary.

Although the model complexity was increased in this work over previous work, there are still various simplifications included in the model which may cause severe inaccuracies during simulation under certain conditions. A discussion of these issues, as well as a discussion on modeling of more complex cryogenic systems is also included in this report.

TABLE OF CONTENTS

I. INTRODUCTION

Investigations in the use of Artificial Intelligence to aid in launch operations began at Kennedy Space Center in the early 1980's. The first implementation of AI at KSC was an expert system developed for the monitoring of liquid oxygen. LES or Liquid oxygen Expert System was the predecessor to the present system being developed at KSC (KATE). KATE or Knowledge-based Autonomous Test Engineer has been under development since 1985. KATE is a frame-based expert system which has been developed within the Common LISP programming environment.

Present work at KSC is focused on the continued development of the KATE system to ultimately achieve autonomous launching capabilities. This development effort, which is being supported by Boeing, requires a series of demonstrations on various hardware systems. In 1989, a demonstration of KATE's ability to monitor, diagnose and control was given on a scaled down version of the shuttle environmental control system. This year, a similar demonstration will be given on a water tanking system which is modeled after the liquid oxygen loading system. For 1991 another demonstration will be given this time on a liquid nitrogen loading system.

With each successive demonstration, the complexity of the task increases. For KATE to operate on any process, a knowledge-base must be created from which a simulation can be run. It is through the running of this simulation, that allows KATE to perform monitoring, diagnosis and control tasks. Hence the model of the process becomes a critical component of the overall system.

The objective of this work was to address various modeling issues which are of concern in the knowledge-base construction. In particular, the ALO water model was the primary focus of the effort. This report however, also includes observations and recommendations with respect to other models which are under development within the KATE environment.

II. PROCESS MODELING

All of the demonstrations for KATE under the ALO project will be based on flow processes. For the systems under consideration a single component is being transported without chemical reaction, hence, individual component mass continuity equations are not required. A general mass continuity equation for a given can be written for any pure, non-reacting flow system as follows:

(2-1) Mass In - Mass Out = Mass Accumulation

2.1 The ALO-H_2O Model

For the conditions in which the ALO - water demonstration will be operated, water can be considered an incompressible fluid. In addition, the assumption of isothermal operation can be made, since ambient temperatures are present throughout the system. As a consequence, the density is constant in time and space and the general balance can be written in terms of flow rates since the mass flow rate is the product of density and volumetric flow rate. Furthermore if we write the continuity equation around a section of pipe filled with liquid we can consider the process to be effectively at steady state at any given instant of time. Hence the continuity equation can be simplified to:

(2-2) Flow In = Flow Out

The process flow diagram for this system, which can be generated by KATE, is shown in Figure 1. Indicated on the Figure, are three distinct sections in which the equation for continuity of mass is applied. The three modeling equations around these points are:

Section 1:
(2-3A) Pump Flow = Recycle Flow + Platform Flow

Section 2:
(2-3B) Platform Flow = Drain Flow + Vehicle Flow

Section 3:
(2-3C) Vehicle Flow = V-Tank Flow + Engine Flow

where:
Pump Flow = The total flow from both pumps.
Recycle Flow = The flow recirculated to the storage tank.
Platform Flow = The flow lifted to the elevated platform.
Drain Flow = The flow out the drain valve.
Vehicle Flow = The total flow going to the vehicle.
V-Tank Flow = The flow to the vehicle tank.

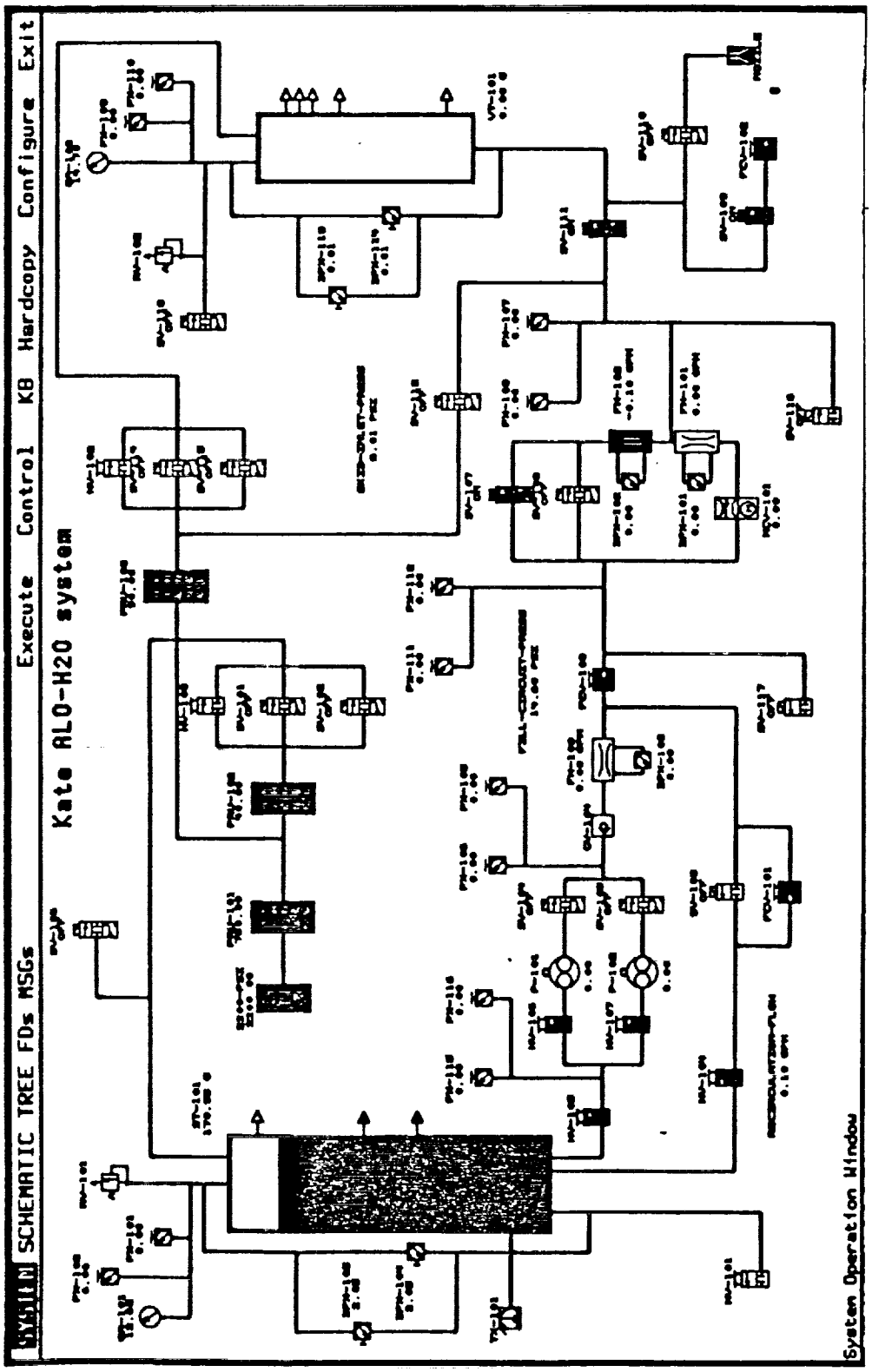

FIGURE 1. Process Diagram for ALO-H_2O Model

Engine Flow = The flow to the engine nozzle and drain.

The flow rate of water through a pipe, valve or fitting can be determined from the equation:

(2-4) $$Q = C_V(\Delta P)^{1/2}$$

where:
Q = the volumetric flow rate.
ΔP = the pressure drop over the hardware.
C_V = the flow coefficient for the hardware.

For a valve the flow coefficient is typically determined from manufacturer's data. The value of C_V is defined as the flow of water at 60 degrees Fahrenheit in gallons per minute at a pressure drop of 1 pound per square inch across the valve. For a pipe or fitting the value of C_V can be determined from the equation:

(2-5) $$C_V = \frac{29.9\ d^2}{(K)^{1/2}}$$

where:
d = the inside diameter of the pipe or fitting in inches.
K = the coefficient for resistance.

Fluid velocity through a pipe, valve or fitting is obtained at the expense of static head. The coefficient for resistance K is the proportionality constant which relates head loss to velocity through the relationship:

(2-6) $$h_L = \frac{Kv^2}{2g}$$

where:
h_L = the head loss through the hardware.
v = the fluid velocity.
g = acceleration of gravity.

Values of K for fittings and hence C_V through the use of equation 2-5 are given in various handbooks. For a given fitting or valve of fixed size, K is constant. For flow through a straight pipe the value of K can be determined from the equation:

(2-7) $$K = f(L/D)$$

where:
f = the friction factor
L/D = the length to diameter ratio for the pipe.

The friction factor is a function of the Reynolds number

which can be obtained from published nomographs or iteratively through equations such as the von Karman-Nikuradse formula:[1]

$$(2\text{-}8) \qquad [f]^{-1/2} = 4.0 \log (Re[f]^{1/2}) - 0.4 \quad (Re > 4000)$$

For a given section of the process a single flow coefficient (or admittance) can be obtained by considering the individual flow coefficients for the valves, fittings and sections of pipe which make up the section as a series of resistances to flow. The admittance is in effect a reciprocal resistance. The effective admittance for n resistances in series can be obtained from the relationship:

$$(2\text{-}9) \qquad A = \frac{1}{[(1/C_{v1})^2 + (1/C_{v2})^2 + \ldots\ldots + (1/C_{vn})^2]^{1/2}}$$

Rewriting equations 3-3A through 3-3C in terms of differential pressures and effective admittances leads to:

$$(2\text{-}10A) \qquad A1[PP - p1]^{1/2} = A2[p1 - STP]^{1/2} + A3[p1 - (HP + p2)]^{1/2}$$

$$(2\text{-}10B) \qquad A3[p1 - (HP + p2)]^{1/2} = A4[(p2 + EP) - ATM]^{1/2} + A5[p2 - p3]^{1/2}$$

$$(2\text{-}10C) \qquad A5[p2 - p3]^{1/2} = A6[p3 - VTP]^{1/2} + A7[p3 - ATM]^{1/2}$$

where:

PP = the pump discharge pressure
STP = the storage tank pressure
VTP = the vehicle tank pressure
HP = the head pressure loss from the elevation change
EP = the head pressure gain from the elevation change
ATM = the barometric pressure
p1 = pressure at point 1
p2 = pressure at point 2
p3 = pressure at point 3
A1 - A7 = admittance values for each branch

At each of the three points, the flow branches into two streams and an unknown pressure node is developed. For a given instant of time all of the parameters except the internal pressures can either be calculated directly (e.g. STP) or assumed constant (e.g. A1). Hence we are faced with the simultaneous solution of three nonlinear equations.

2.2 Modeling of Cryogenic Fluids

The modeling of flow systems involving cryogenic fluids, involves much more complexity. If it could be assured that the fluid was only in the liquid phase, the physical properties and

flow characteristics are less understood then those of water. In reality however, some vaporization of the fluid will occur. This gives rise for the need to model the heat transfer in the system as well as mass continuity. The degree of heat transfer will effect the stream quality and hence the thermodynamics of phase equilibrium will also need to be accounted for.

To sufficiently model this process will require a large effort. Equation 2-1 can no longer be simplified to get equation 2-2 as we are now dealing with a two phase compressible fluid. There have been two approaches taken to model the two phase flow problem: (1) a homogeneous approach in which the two phases are treated as a single phase with averaged fluid properties, and (2) a separated-flow model, in which the two phases are considered artificially segregated.[2] Lockhart and Martinelli [3] developed a semiempirical correlation segregated flow model for flow in horizontal tubes. A modification of this correlation has been shown to give good accuracy for cryogenic nitrogen. [4]

The Lockhart and Martinelli correlation calculates a two phase pressure drop by determining a correction factor ϕ^2_L which is applied to the liquid phase pressure drop. The correlation for adiabatic two phase pressure drop data is:

$$(2\text{-}11) \qquad (\Delta p / \Delta L)_{TP} = \phi^2_L \, (\Delta p / \Delta L)_L$$

where:

$(\Delta p / \Delta L)_{TP}$ = pressure drop per length for two phase flow
$(\Delta p / \Delta L)_L$ = pressure drop per length for liquid flow

The correction factor ϕ_L is determined from the relationship:

$$(2\text{-}12) \qquad \phi_L = (X^2 + CX + 1)^{1/2} / X$$

where C is an empirical constant ranging from 5 (when both phases are in laminar flow) to 20 when both phases are turbulent. The parameter X is given by:

$$(2\text{-}13) \qquad X^2 = \frac{C_L \, (Re_G)^m}{C_G \, (Re_L)^n} \, \frac{\rho_G}{\rho_L} \left(\frac{1 - x}{x}\right)^2$$

where:

C_L and n are empirical constants for the liquid phase
C_G and m are empirical constants for the gas phase
Re_G = gas phase Reynolds number
Re_L = liquid phase Reynolds number
x = the vapor mass fraction (quality)
ρ_L = the liquid phase density
ρ_G = the gas phase density

Since in reality, heat transfer will be occurring in the

system, a modified differential version of equation 2-11 must be coupled with an energy balance and numerically integrated along the length of the tube to get the total frictional pressure drop. This process is complicated by the fact that the physical properties and hence Reynolds numbers will be changing along the tube length. Also the change in the state variables will result in variations in the fluid quality.

The change in fluid quality results in a change in the bulk fluid velocity and consequently a change in fluid momentum. Hence a momentum pressure drop must be added to the contribution from the frictional pressure drop to calculate a total pressure drop. The equation to calculate the momentum pressure drop is:

$$(2\text{-}14) \qquad p_m = \phi_m \, (M_L + M_G)^2 \, / \, g_c \, \rho_L \, A^2$$

where the correction factor O_m for the momentum pressure drop is given by:

$$(2\text{-}15) \qquad \frac{(1 - x_2)^2}{R_{L,2}} - \frac{(1 - x_1)^2}{R_{L,1}} + \left(\frac{x_2^2}{1 - R_{L,2}} - \frac{x_1^2}{1 - R_{L,1}} \right) \left(\frac{\rho_L}{\rho_G} \right)$$

R_L is the volume fraction of liquid phase where the subscript 1 indicates inlet conditions and the subscript 2 indicates outlet conditions. R_L is a function of the Lockhart-Martinelli parameter X:

$$(2\text{-}16) \qquad R_L = X \, / \, (X^2 + CX + 1)^{1/2}$$

III. SOLUTIONS FOR SYSTEMS OF NONLINEAR EQUATIONS

As discussed above, improved modeling of the ALO- H_2O system requires the numerical solution of simultaneous nonlinear equations. Two distinct groups of methods are the most commonly employed for this task: functional iteration and minimizing methods. Both of the aforementioned methods were considered for this study and LISP functions were written for implementation in the KATE knowledge-base. A brief description of the rational behind these schemes is presented here.

3.1 Newton's Method

The specific technique used under the functional iteration category is known as Newton's method. It has the advantage over minimization methods in the speed of convergence of the algorithm. This method is an n dimensional extension of the 1 dimensional Newton-Raphson algorithm. Hence to help illustrate the method, the 1 dimensional case will be considered first.

Consider a function f which is twice continuously differentiable in an interval of interest. Let x^0 be an approximation to the root p (i.e. $f(p) = 0$) of the function such that the first derivative $f'(x^0)$ is not equal to 0. The function $f(x)$ can then be approximated with a Taylor series expansion about the point x^0 as follows:

$$f(x) = f(x^0) + (x - x^0)\, f'(x^0) \tag{3-1}$$

Since $f(p) = 0$ the above equation can be rewritten with $x = p$ as:

$$0 = f(x^0) + (p - x^0)\, f'(x^0) \tag{3-2}$$

Solving for p gives:

$$p = x^0 - f(x^0) \,/\, f'(x^0) \tag{3-3}$$

This gives rise to the Newton-Raphson algorithm which involves generating the sequence p_n defined by:

$$p_n = p_{n-1} - f(p_{n-1}) \,/\, f'(p_{n-1}), \quad (n > 1) \tag{3-4}$$

This sequence is generated in the algorithm until successive values of p_n and p_{n-1} are within a specified tolerance. This method is illustrated graphically in Figure 2. It is seen that at each iteration a new approximation of the root p_n is obtained from the slope of the tangent to the function evaluated at the previous approximation p_{n-1} (i.e. the first derivative).

The extension of this method to n dimensions is relatively straight forward. For n dimensions, there are n functions of the

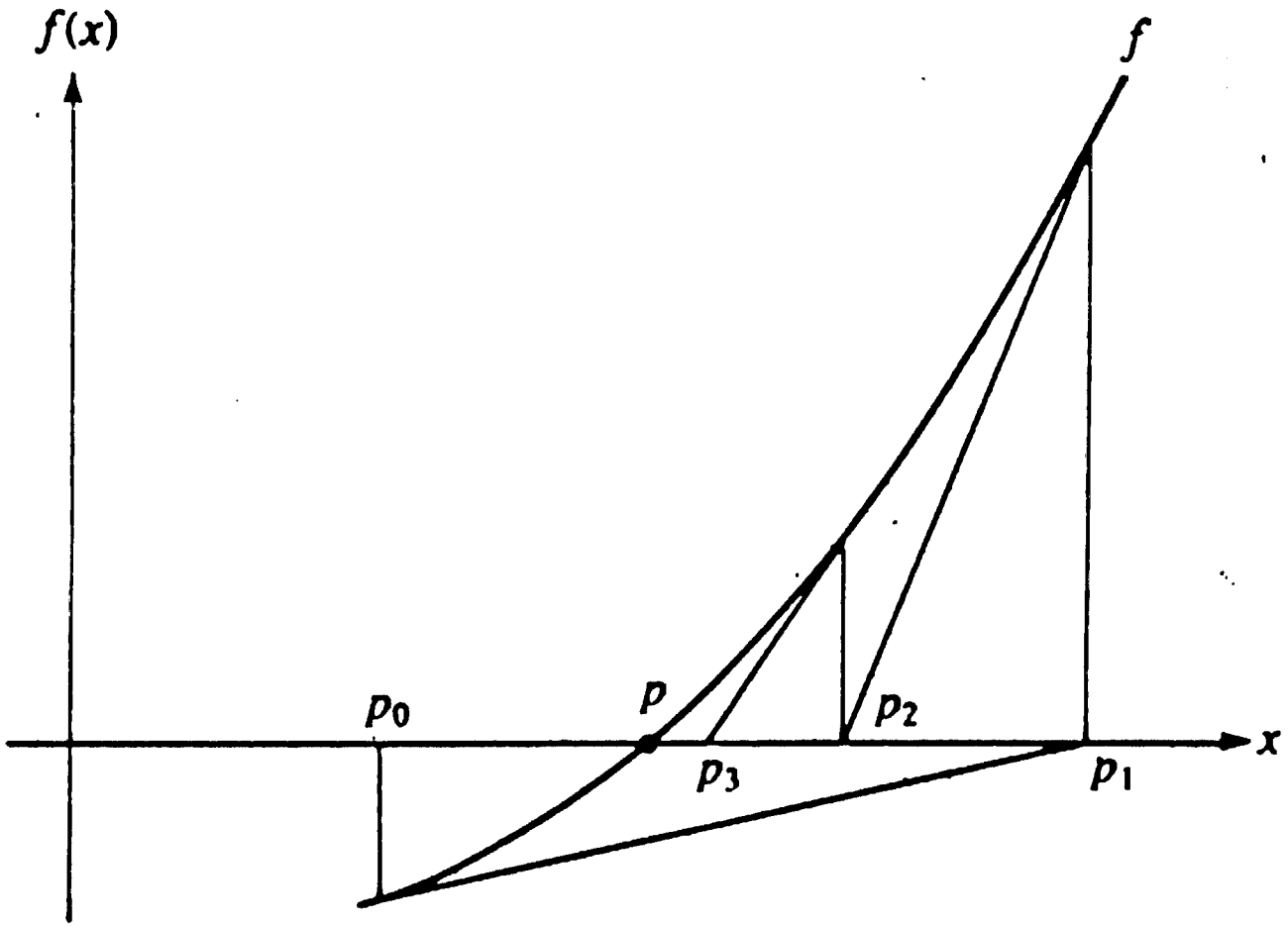

FIGURE 2. Graphical Interpretation of Newton's Method

n independent variables. Analogous to equation 3-1 for the 1 dimensional case, a Taylor's expansion about an approximate solution vector $\mathbf{x}^0 = (x_1^0, x_2^0, \ldots\ldots, x_n^0)^t$ can be truncated after the first degree terms for each of the n functions.

To illustrate this, we can consider the two dimensional case. The results of the Taylor approximations leads to:

$$(3\text{-}5A) \qquad f_1(x_1,x_2) = f_1(x_1^0,x_2^0) + \frac{\partial f_1}{\partial x_1}_0(x_1^0,x_2^0)\,[x_1 - x_1^0] + \frac{\partial f_1}{\partial x_2}_0(x_1^0,x_2^0)\,[x_2 - x_2^0]$$

$$(3\text{-}5B) \qquad f_2(x_1,x_2) = f_2(x_1^0,x_2^0) + \frac{\partial f_2}{\partial x_1}_0(x_1^0,x_2^0)\,[x_1 - x_1^0] + \frac{\partial f_2}{\partial x_2}_0(x_1^0,x_2^0)\,[x_2 - x_2^0]$$

As with equation 3-2 the value of the functions f_1 and f_2 is zero at the roots p_1 and p_2. Rearranging equations 3-5A and 3-5B, replacing x_1 and x_2 with p_1 and p_2 respectively and putting into matrix notation gives:

$$(3\text{-}6) \qquad \mathbf{J}(x_1^0,x_2^0)\ \mathbf{y} = -\mathbf{b}$$

where:

$$\mathbf{J} = \begin{vmatrix} \frac{\partial f_1}{\partial x_1}_0(x_1^0,x_2^0) & , & \frac{\partial f_1}{\partial x_2}_0(x_1^0,x_2^0) \\ \frac{\partial f_2}{\partial x_1}_0(x_1^0,x_2^0) & , & \frac{\partial f_2}{\partial x_2}_0(x_1^0,x_2^0) \end{vmatrix}$$

$$\mathbf{y} = \begin{vmatrix} (p_1 - x_1^0) \\ (p_2 - x_1^0) \end{vmatrix}$$

$$\mathbf{b} = \begin{vmatrix} f_1(x_1^0,x_2^0) \\ f_2(x_1^0,x_2^0) \end{vmatrix}$$

The matrix can be solved for $\mathbf{y}$ by multiplying both sides of equation 3-6 by $\mathbf{J}^{-1}$. Then in a manner analogous to equation 3-4 new guesses are generated for the roots repeatedly until convergence within the desired tolerance is obtained.

Hence for the n dimensional case, the n functions require n partial derivatives to be evaluated. The result of this differentiation is placed in an n x n matrix known as the

Jacobian matrix. The n elements in the first row of this matrix contains the results of differentiating the first function with respect to each of the n variables. Similarly the nth row elements contain the differentiation of the nth function with respect to the n variables. Although the n equations can be solved by inverting the Jacobian matrix, in practice an iterative technique would be used [3].

3.2 Steepest Descent Method

The specific method investigated under the category of minimization techniques is known as the method od steepest descent. Although the minimization methods will generally converge slower than the iterative methods they will generally converge with any initial approximation. The method of steepest descent determines a local minimum for a multivariable function $G(\mathbf{x})$ defined by:

$$(3\text{-}7) \qquad G(x_1, x_2, \ldots, x_n) = [f_i(x_1, x_2, \ldots, x_n)]^2$$

where each function $f_i(x_1, x_2, \ldots, x_n) = 0$. An exact solution at $\mathbf{x} = (x_1, x_2, \ldots, x_n)^t$ is determined when the function G is zero.

The algorithm for finding the solution can be summarized as follows:

1. Evaluate G at an initial approximation $\mathbf{x}^0 = (x_1^0, x_2^0, \ldots, x_n^0)^t$

2. Determine a direction from $\mathbf{x}^0$ that results in a decrease in the value of G.

3. Decide the amount which should be moved in this direction and call the new value $\mathbf{x}^1$.

4. Repeat steps 1 through 3 with $\mathbf{x}^0$ replaced by $\mathbf{x}^1$.

From calculus, the Extreme Value Theorem implies that a single variable differentiable function has a minimum when the derivative is zero. This can be extended to a multivariable function in that a minimum exists at $\mathbf{x}$ when the gradient is zero. Hence the solution vector $\mathbf{x}$ occurs where:

$$(3\text{-}8) \qquad G(\mathbf{x}) = \left[\frac{\partial G}{\partial x_1}(\mathbf{x}), \frac{\partial G}{\partial x_2}(\mathbf{x}), \ldots\ldots \frac{\partial G}{\partial x_n}(\mathbf{x})\right]^t = 0$$

Explanation of the logic involved in the determination of steps 2 and 3 above is detailed and of little value to this report since this method was not actually implemented. Further details of this method are given by Burden and Faires [3].

IV. RESULTS AND DISCUSSION

The source code for the solution of the modeling equations is given in the Appendix. Comments have been included in the code to aid in any future modifications. The LISP routine had to be capable of handling not only the normal situation of forward flow (i.e when the pump is on) but the back flow condition when the pump is off. In addition various valve configurations leading to zero admittance values had to be addressed. As a consequence, much of the code written was aimed at dealing with these "abnormal" situations can occur.

A example of the problem encountered with particular values of zero admittance can be shown if say both A4 and A5 in equation 2-10b are equal to zero. This indicates there is no flow in either the fourth or fifth legs in the second section of pipe. Hence the flow in the third leg must also be zero and either A3 must be zero or the differential pressure in the leg must be zero. In either case the original treatment will cause problems in the Jacobian matrix. Having A3 also zero would cause **J** to become singular, while a zero differential pressure would cause the element $J_{2,2}$ to go to infinity. Comments within the code try and address the logic for each of the cases encountered and hence no further discussion will be given on them.

The program developed to solve the three internal pressure nodes by Newton's method was loaded into the KATE system and a simulation run. Although all possible combinations of events which could occur were not tested, the system worked well on those that were tested. Figure 3 shows a plot of the values for p1, p2 and p3 during a simulation run. Six distinct changes were made in the system in order to observe the response by the program. These changes are labeled A through F on Figure 3.

When the system was first started p1 was at approximately 52 psi as a single pump was operating. As the vehicle tank was filled a slight decrease in p1 was observed while p2 and p3 showed increases. This behavior is expected as p1 drops some since the pressure in the storage tank decreases, while p2 and p3 increase due to the filling, and subsequent head pressure, of the vehicle tank. At point A the drain valve is opened. This is equivalent of changing A4 to a non-zero value. Again, the results are expected as little effect is observed on p1 and p3, however p2 has an immediate drop in pressure.

At point B the valve is shut again and the system responds accordingly. At point C the second pump is turned, resulting in pressure increases throughout the system. Point D signifies the start of back flow as the pumps were stopped. As expected p1 has an immediate drop in pressure, while p2 and p3 slowly decline. As shown on the plot, p3 is greater than p2, which is greater than

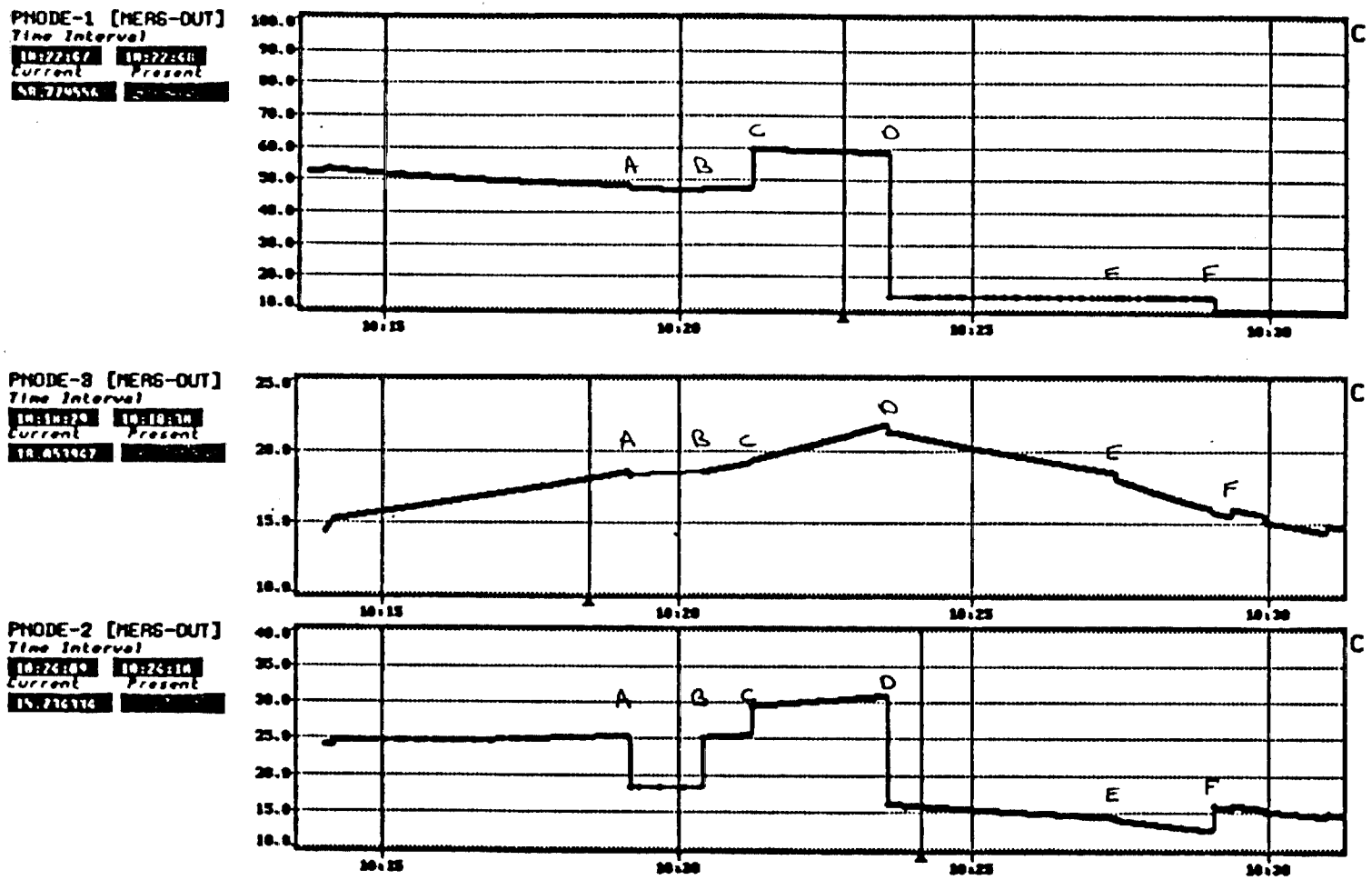

FIGURE 3. Data Plots for P1, P2 & P3 During the Simulation Run

p1 as required for back flow. At point E, the valves to the engine section (i.e. A7) were opened. Since flow is now draining through the engine as well as the transfer lines, the rate of pressure decrease in p2 and p3 should become greater as is observed.

The last change made to the system (point F) was to set A3 to zero. At this point, p1 equalizes to the storage tank pressure as expected, while a sudden increase is observed in p2 and p3. Again this is expected since less area is available for flow.

V. CONCLUDING REMARKS

Although a liquid water transfer system is a relatively easy system to model, various assumptions are being used to allow for real-time implementation. For example, the use of an effective flow coefficient for each section of the transfer hardware was used in the model. Although the values used can be readily determined for a given set of conditions they will change as conditions change since the resistance coefficient depends on the Reynolds number which in turn depends on the flow.

The initial values being used in the knowledge-base are approximated assuming highly turbulent flow. In the highly turbulent regime the friction factor does in fact become constant and hence fairly accurate simulation values could be obtained. When the pumps are turned off however and the water drains back into the storage tank the flow regime will not be highly turbulent and I anticipate substantial error to be present if the flow coefficients are not adjusted.

As an example of the magnitude of the error which could be expected let's consider water flowing through a 10 foot section of 1 inch smooth pipe in highly turbulent flow. The lowest Reynolds number for highly turbulent flow in this size pipe is 8×10^5. This corresponds to a friction factor of 0.023. Using equation 3-7 with a L/D ratio of 120 gives a resistance coefficient of 2.76 and a flow coefficient of 18.00. If the flow is reduced to by one tenth during say a pump failure, the Reynolds number will be dropped proportionately. The new value of the friction factor will be .025 resulting in a new resistance coefficient of 3.0 and a new flow coefficient of 17.26. The increase in the resistance translates into an 8.8% error in the calculation of pressure drop from the model if a correction is not made.

As the complexity of the systems to be modeled increases, more simplifications will be required in order to allow real-time simulations to be carried out. With cryogenic systems, the simplifications may prove to cause substantial error in the predictive ability of the model. It is my belief that before the KATE system can adequately deal with systems of increased complexity, a methodology for verifying and updating model parameters needs to be developed. In some instances, this could be done by comparing predicted values from the simulation to sensor data within tolerance limits. If drift in the predictive abilities is observed over time, the model parameters could be modified to correct for it. Monitoring of the model base would probably require the use of a distributed processor to analyze trends in the data.

In addition to updating model parameters, it may also be necessary to have separate modeling equations under different conditions. For example, the modeling of line chill down requires a much higher degree of model sophistication due to the unsteady state heat transfer involved than the modeling after a thermal steady state has been reached.

VI. APPENDIX

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This variable is used to account for the elevation head above the nozzle bleed and drain
;; A 1.5 foot elevation was assumed
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defvar nozzle-head 0.65)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to check if two numbers are essentially equal.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun almost-equal (x y)
  (< (abs (- x y)) .000001))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to count the number of zero admittances.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun count-zeros (list)
  (do* ((cnt 0)
        (mlist list (cdr mlist)))
       ((null mlist) cnt)
    (when (zerop (car mlist))(incf cnt))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to determine if any complex numbers were encountered in
;;   which case the solve routine will exit with an error message.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun cmplx-chk (list)
  (do* ((cnt 0)
        (mlist list (cdr mlist)))
       ((null mlist) cnt)
    (when (complexp (car mlist))(incf cnt))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to compute the square root of the pressure difference in the first leg.
;; Logic is included to avoid dividing by zero in the Jacobian matrix by returning the symbol DELP0
;;   when the differential pressure is zero.
;; Logic is also included to avoid the return of complex numbers.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun pvar1 (p1 ppump)
  (cond ((> ppump p1)
         (sqrt (- ppump p1)))
        ((almost-equal p1 ppump)
         'delp0)
        (t (- (sqrt (abs (- ppump p1)))))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to compute the square root of the pressure difference in the second leg.
;; Logic is included to avoid dividing by zero in the Jacobian matrix by returning the symbol DELP0
;;   when the differential pressure is zero.
;; Logic is also included to avoid the return of complex numbers.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun pvar2 (p1 st-press)
  (cond ((> p1 st-press)
         (sqrt (- p1 st-press)))
        ((almost-equal p1 st-press)
         'delp0)
        (t (- (sqrt (abs (- p1 st-press)))))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to compute the square root of the pressure difference in the third leg.
;; Logic is included to avoid dividing by zero in the Jacobian matrix by returning the symbol DELP0
;;   when the differential pressure is zero.
;; Logic is also included to avoid the return of complex numbers.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun pvar3 (p1 p2 head-press)
  (cond ((> p1 (+ head-press p2))
         (sqrt (- p1 head-press p2)))
        ((almost-equal p1 (+ head-press p2))
         'delp0)
        (t (- (sqrt (abs (- (+ p2 head-press) p1)))))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to compute the square root of the pressure difference in the fourth leg.
;; Logic is included to avoid dividing by zero in the Jacobian matrix by returning the symbol DELP0
;;   when the differential pressure is zero.
;; Also if p2 is less than atmospheric pressure or the lines are not filled the symbol delp0
;;   is returned to indicate no true flow.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun pvar4 (p2 in-sys)
  (cond
    ((not (> in-sys drain-capacity)) 'delp0)
    ((> (+ p2 elevation-press) ambient-atm)
     (sqrt (- (+ p2 elevation-press)  ambient-atm)))
    ((almost-equal p2 ambient-atm)
     'delp0)
    (t 'delp0)))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to compute the square root of the pressure difference in the fifth leg.
;; Logic is included to avoid dividing by zero in the Jacobian matrix by returning the symbol DELP0
;;   when the differential pressure is zero.
;; Logic is also included to avoid the return of complex numbers.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun pvar5 (p2 p3)
  (cond
    ((> p2 p3)
     (sqrt (- p2 p3)))
    ((almost-equal p3 p2)
     'delp0)
    (t (- (sqrt (abs (- p2 p3)))))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to compute the square root of the pressure difference in the sixth leg.
;; Logic is included to avoid dividing by zero in the Jacobian matrix by returning the symbol DELP0
;;   when the differential pressure is zero.
;; Logic is also included to avoid the return of complex numbers.
;; Also if the lines are not filled the symbol delp0 is returned to indicate no true flow.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun pvar6 (p3 vt-press in-sys)
  (cond
    ((not (> in-sys line-capacity)) 'delp0)
    ((>  p3 vt-press)
    (sqrt (- p3 vt-press)))
    ((almost-equal p3 vt-press)
     'delp0)
    (t (- (sqrt (abs (- p3 vt-press)))))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to compute the square root of the pressure difference in the seventh leg.
;; Logic is included to avoid dividing by zero in the Jacobian matrix by returning the symbol DELP0
;;   when the differential pressure is zero.
;; Also if p3 is less than atmospheric pressure or the lines are not filled the symbol delp0
;;   is returned to indicate no true flow.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun pvar7 (p3 in-sys)
  (cond
    ((not (> in-sys line-capacity)) 'delp0)
    ((> (+ p3 nozzle-head) ambient-atm)
     (sqrt (- (+ p3 nozzle-head) ambient-atm)))
    ((almost-equal p3 ambient-atm)
     'delp0)
    (t 'delp0)))



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function for the material balance around section 1.
;;   [Flow from pump - Recirculation Flow - Flow to the elevated platform = 0]
;; Logic is included to check to make sure the functions PVAR1, PVAR2 & PVAR3 return a number.
;;   Return of the symbol DELP0 from these functions occurs when a pressure differential of zero is
;;   encountered. If DELP0 is returned the contribution from the leg is set equal to zero.
;; Logic is also included to have the function F1 return zero if any two admittances passed to the
;;   function are zero. This avoids discontinuity in the material balance.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun f1 (p1 p2 ppump st-press head-press a1 a2 a3)
  (cond
    ((or (and (zerop a1)(zerop a2))
         (and (zerop a1)(zerop a3))
         (and (zerop a2)(zerop a3)))0)
    (t
     (- (cond ((numberp (pvar1 p1 ppump))
               (* a1 (pvar1 p1 ppump)))
              (t 0))
        (+ (cond ((numberp (pvar2 p1 st-press))
                  (* a2 (pvar2 p1 st-press)))
                 (t 0))
           (cond ((numberp (pvar3 p1 p2 head-press))
                  (* a3 (pvar3 p1 p2 head-press)))
                 (t 0)))))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function for the material balance around section 2.
;;   [Flow to the elevated platform - Flow toward the vehicle - Flow out the drain = 0]
;; Logic is included to check to make sure the functions PVAR3, PVAR4 & PVAR5 return a number.
;;   Return of the symbol DELP0 from these functions occurs when a pressure differential of zero is
;;   encountered. If DELP0 is returned the contribution from the leg is set equal to zero.
;; Logic is also included to have the function F2 return zero if any two admittances passed to the
;;   function are zero or the functions PVAR4 and PVAR5 return DELP0. This avoids discontinuity in
;;   the material balance.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun f2 (p1 p2 p3 head-press a3 a4 a5 in-sys)
  (cond
    ((or (and (zerop a3)(zerop a4))
         (and (zerop a3)(zerop a5))
         (and (zerop a4)(zerop a5))
         (and (not (numberp (pvar4 p2 in-sys)))(not (numberp (pvar5 p2 p3)))))0)
    (t
     (- (cond ((numberp (pvar3 p1 p2 head-press))
               (* a3 (pvar3 p1 p2 head-press)))
              (t 0))
        (+ (cond ((numberp (pvar4 p2 in-sys))
                  (* a4 (pvar4 p2 in-sys)))
                 (t 0))
           (cond ((numberp (pvar5 p2 p3))
                  (* a5 (pvar5 p2 p3)))
                 (t 0)))))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function for the material balance around section 3.
;;   [Flow toward the vehicle - Flow to the vehicle tank - Flow to the engine nozzle and bleed = 0]
;; Logic is included to check to make sure the functions PVAR5, PVAR6 & PVAR7 return a number.
;;   Return of the symbol DELP0 from these functions occurs when a pressure differential of zero is
;;   encountered. If DELP0 is returned the contribution from the leg is set equal to zero.
;; Logic is also included to have the the function F3 return zero if admittances a5 and a7 are passed
;;   to the function as zero or the functions PVAR6 and PVAR7 return DELP0. This avoids discontinuity
;;   in the material balance.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun f3 (p2 p3 vt-press a5 a6 a7 in-sys)
  (cond
   ((or (and (zerop a5)(zerop a7))
        (and (not (numberp (pvar6 p3 vt-press in-sys)))(not (numberp (pvar7 p3 in-sys)))))0)
   (t
    (- (cond ((numberp (pvar5 p2 p3))
              (* a5 (pvar5 p2 p3)))
             (t 0))
       (+ (cond ((numberp (pvar6 p3 vt-press in-sys))
                 (* a6 (pvar6 p3 vt-press in-sys)))
                (t 0))
          (cond ((numberp (pvar7 p3 in-sys))
                 (* a7 (pvar7 p3 in-sys)))
                (t 0)))))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to compute the element in row 1 and column 1 of the Jacobian matrix. This function
;;   represents the partial derivative of the function F1 with respect to the variable P1.
;; Logic is included to check to make sure the functions PVAR1, PVAR2 & PVAR3 return a number.
;;   Return of the symbol DELP0 from these functions occurs when a pressure differential of zero is
;;   encountered. If DELP0 is returned the contribution from the leg is set equal to zero.
;; Logic is also included to have the the function J11 return one if any two admittances are passed to
;;   the function as zero or the flow in the leg is zero. This prevents the Jacobian matrix from
;;   becoming singular.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun j11 (p1 p2 ppump st-press head-press a1 a2 a3)
  (cond
   ((or
     (and (zerop a1)(zerop a2))
     (and (zerop a2)(zerop a3))
     (and (zerop a1)(zerop a3)))1)
   (t
    (let* ((result
            (+ (cond ((numberp (pvar1 p1 ppump))
                      (/ (* -.5 a1)
                         (abs(pvar1 p1 ppump))))
                     (t 0))
               (cond ((numberp (pvar2 p1 st-press))
                      (/ (* -.5 a2)
                         (abs(pvar2 p1 st-press))))
                     (t 0))
               (cond ((numberp (pvar3 p1 p2 head-press))
                      (/ (* -.5 a3)
                         (abs(pvar3 p1 p2 head-press))))
                     (t 0)))))
      (cond ((zerop result) 1)
            (t result))))))
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to compute the element in row 1 and column 2 of the Jacobian matrix. This function
;;   represents the partial derivative of the function F1 with respect to the variable P2.
;; This function is also used for the element in row 2 and column 1 of the Jacobian matrix and hence
;;   also represents the partial derivative of function F2 with respect to the variable P1.
;; Logic is included to check to make sure the function PVAR3 returns a number.
;;   Return of the symbol DELPO from this function occurs when a pressure differential of zero is
;;   encountered. If DELPO is returned the function is set equal to zero.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun j12 (p1 p2 head-press a3)
  (cond ((numberp (pvar3 p1 p2 head-press))
         (/ (* .5 a3)(abs(pvar3 p1 p2 head-press))))
        (t 0)))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to compute the element in row 2 and column 2 of the Jacobian matrix. This function
;;   represents the partial derivative of the function F2 with respect to the variable P2.
;; Logic is included to check to make sure the functions PVAR3, PVAR4 & PVAR5 return a number.
;;   Return of the symbol DELPO from these functions occurs when a pressure differential of zero is
;;   encountered. If DELPO is returned the contribution from the leg is set equal to zero.
;; Logic is also included to have the the function J22 return one if any two admittances are passed to
;;   the function as zero or there is no flow in the leg. This prevents the Jacobian matrix from
;;   becoming singular.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun j22 (p1 p2 p3 head-press a3 a4 a5 in-sys)
  (cond
    ((or
       (and (zerop a3)(zerop a4))
       (and (zerop a4)(zerop a5))
       (and (zerop a3)(zerop a5))
       (and (not (numberp (pvar4 p2 in-sys)))(not (numberp (pvar5 p2 p3)))))1)
    (t
     (let* ((result
              (+  (cond ((numberp (pvar3 p1 p2 head-press))
                          (/ (* -.5 a3)
                             (abs(pvar3 p1 p2 head-press))))
                         (t 0))
                  (cond ((numberp (pvar4 p2 in-sys))
                          (/ (* -.5 a4)
                             (abs(pvar4 p2 in-sys))))
                         (t 0))
                  (cond ((numberp (pvar5 p2 p3))
                          (/ (* -.5 a5)
                             (abs(pvar5 p2 p3))))
                         (t 0)))))
       (cond ((zerop result) 1)
             (t result))))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to compute the element in row 2 and column 3 of the Jacobian matrix. This function
;;   represents the partial derivative of the function F2 with respect to the variable P3.
;; This function is also used for the element in row 3 and column 2 of the Jacobian matrix and hence
;;   also represents the partial derivative of function F3 with respect to the variable P2.
;; Logic is included to check to make sure the function PVAR5 returns a number.
;;   Return of the symbol DELPO from this function occurs when a pressure differential of zero is
;;   encountered. If DELPO is returned the function is set equal to zero.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun j23 (p2 p3 a5)
  (cond ((numberp (pvar5 p2 p3))
         (/ (* .5 a5)(abs(pvar5 p2 p3))))
        (t 0)))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to compute the element in row 3 and column 3 of the Jacobian matrix. This function
;;   represents the partial derivative of the function F3 with respect to the variable P3.
;; Logic is included to check to make sure the functions PVAR5, PVAR6 & PVAR7 return a number.
;;   Return of the symbol DELPO from these functions occurs when a pressure differential of zero is
;;   encountered. If DELPO is returned, the contribution from the appropriate leg is set equal to zero.
;; Logic is also included to have the the function J33 return one if admittances a5 & a7 are passed to
;;   the function as zero or there is no flow in the leg. This prevents the Jacobian matrix from
;;   becoming singular.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun j33 (p2 p3 vt-press a5 a6 a7 in-sys)
  (cond
    ((or (and (zerop a5)(zerop a7))
         (and (not (numberp (pvar6 p3 vt-press in-sys)))(not (numberp (pvar7 p3 in-sys)))))1)
    (t
     (let* ((result
             (+  (cond ((numberp (pvar5 p2 p3))
                        (/ (* -.5 a5)
                           (abs(pvar5 p2 p3))))
                       (t 0))
                 (cond ((numberp (pvar6 p3 vt-press in-sys))
                        (/ (* -.5 a6)
                           (abs(pvar6 p3 vt-press in-sys))))
                       (t 0))
                 (cond ((numberp (pvar7 p3 in-sys))
                        (/ (* -.5 a7)
                           (abs(pvar7 p3 in-sys))))
                       (t 0)))))
    (cond ((zerop result) 1)
          (t result))))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function which is called by the solve function if any 5 admittance values are equal to zero.
;; The values returned depend on the exact configuration of valves being open and closed (0 admittance)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun get-vals->4 (p1 p2 p3 ppump st-press vt-press head-press a1 a2 a3 a4 a5 a6 a7)
  (setq p3 vt-press)
  (cond
    ((not (zerop a1))
     (setq p1 ppump))
    ((not (zerop a2))
     (setq p1 st-press))
    ((not (zerop a3))
     (setq p1 + head-press (/ (+ p1 p2 - head-press) 2))
     (setq p2 (- p1 head-press)))
    ((not (zerop a4))
     (setq p2 ambient-atm))
    ((not (zerop a5))
     (setq p2 vt-press))
    ((not (zerop a7))
     (setq p3 (/ (* vt-press (square (/ a6 a7)))
                 (+ 1 (square (/ a6 a7)))))))
  (values p1 p2 p3))
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Function which is called by the solve function if any 4 admittance values are equal to zero provided
;;   that one of the zero values for admittance is a4 or a5.
;; The values returned depend on the exact configuration of valves being open and closed (0 admittance)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun get-vals->3 (p1 p2 p3 ppump st-press vt-press head-press a1 a2 a3 a4 a5 a6 a7)
  (cond
    ((zerop a7)
     (setq p3 vt-press))
    (t
     (setq p3 (/ (* vt-press (square (/ a6 a7)))
                 (+ 1 (square (/ a6 a7)))))))
  (cond
    ((and (not (zerop a1))(not (zerop a2)))
     (setq p1 (/ (+ st-press (* ppump (square (/ a1 a2))))
                 (+ 1 (square (/ a1 a2))))))
     ((and (not (zerop a1))(not (zerop a3)))
      (setq p1 ppump)
      (setq p2 (- ppump head-press)))
     ((and (not (zerop a1))(not (zerop a4)))
      (setq p1 ppump)
      (setq p2 ambient-atm))
     ((and (not (zerop a1))(not (zerop a5)))
      (setq p1 ppump)
      (setq p2 vt-press))
     ((and (not (zerop a1))(not (zerop a7)))
      (setq p1 ppump))
     ((and (not (zerop a2))(not (zerop a3)))
      (setq p1 st-press)
      (setq p2 (- st-press head-press)))
     ((and (not (zerop a2))(not (zerop a4)))
      (setq p1 st-press)
      (setq p2 ambient-atm))
     ((and (not (zerop a2))(not (zerop a5)))
      (setq p1 st-press)
      (setq p2 vt-press))
     ((and (not (zerop a2))(not (zerop a7)))
      (setq p1 st-press))
     ((and (not (zerop a3))(not (zerop a4)))
      (setq p1 head-press)
      (setq p2 ambient-atm))
     ((and (not (zerop a3))(not (zerop a5)))
      (setq p1 (+ vt-press head-press))
      (setq p2 vt-press))
     ((and (not (zerop a3))(not (zerop a7)))
      (setq p2 (- p1 head-press)))
     ((and (not (zerop a4))(not (zerop a7)))
      (setq p2 ambient-atm))
     ((and (not (zerop a5))(not (zerop a7)))
      (setq p2 p3)))
    (values p1 p2 p3))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function which is called by the solve function if any 3 admittance values are equal to zero provided
;;   that one of the zero values for admittance is a4 or a5. In addition, none of the following
;;   conditions can exist for this function to be called:
;;                                                              a1, a4 & a7 can not all be zero
;;                                                              a2, a4 & a7 can not all be zero
;;                                                              a2, a5 & a7 can not all be zero
;; The values returned depend on the exact configuration of valves being open and closed (0 admittance)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun get-vals->2 (p1 p2 p3 ppump st-press vt-press head-press a1 a2 a3 a4 a5 a6 a7)
  (cond
    ((zerop a7)
     (setq p3 vt-press))
    (t
     (setq p3 (/ (* vt-press (square (/ a6 a7)))
                 (+ 1 (square (/ a6 a7)))))))
  (cond
    ((and (zerop a1)(zerop a2)(zerop a4))
     (setq p2 p3)
     (setq p1 (+ p2 head-press)))
    ((and (zerop a1)(zerop a2)(zerop a5))
     (setq p1 head-press)
     (setq p2 ambient-atm))
    ((and (zerop a1)(zerop a3)(zerop a4))
     (setq p1 st-press)
     (setq p2 p3))
    ((and (zerop a1)(zerop a3)(zerop a5))
     (setq p1 st-press)
     (setq p2 ambient-atm))
    ((and (zerop a1)(zerop a4)(zerop a5))
     (setq p1 st-press)
     (setq p2 (- st-press head-press)))
    ((and (zerop a1)(zerop a5)(zerop a7))
     (setq p1 st-press)
     (setq p2 (- st-press head-press)))
    ((and (zerop a2)(zerop a3)(zerop a4))
     (setq p1 ppump)
     (setq p2 p3))
    ((and (zerop a2)(zerop a3)(zerop a5))
     (setq p1 st-press)
     (setq p2 ambient-atm))
    ((and (zerop a2)(zerop a4)(zerop a5))
     (setq p1 ppump)
     (setq p2 (- ppump head-press)))
    ((and (zerop a3)(zerop a4)(zerop a5))
     (setq p1 (/ (+ st-press (* ppump (square (/ a1 a2))))
                 (+ 1 (square (/ a1 a2))))))
     ((and (zerop a3)(zerop a4)(zerop a7))
      (setq p1 (/ (* vt-press (square (/ a6 a7)))
                  (+ 1 (square (/ a6 a7)))))
      (setq p2 vt-press))
     ((and (zerop a3)(zerop a5)(zerop a7))
      (setq p1 (/ (+ st-press (* ppump (square (/ a1 a2))))
                  (+ 1 (square (/ a1 a2)))))
      (setq p2 ambient-atm))
     ((and (zerop a4)(zerop a5)(zerop a7))
      (setq p1 ppump)
      (setq p2 (- ppump head-press))))
   (values p1 p2 p3))
```

ORIGINAL PAGE IS
OF POOR QUALITY

```lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function which solves the nonlinear flow equations using Newton's method. This involves iterating
;;   on the unknown internal pressures p1, p2 & p3 until convergence within a tolerance of .01 is
;;   achieved. The solution procedure involves solving for the vector x, where each element of x
;;   represents the difference in the guessed values for p1, p2 & p3 and the next approximation for
;;   each. The equation which is solved is:
;;                                        J x = F
;; The elements of the Jacobian matrix J and the vector F have been defined above. Solution of the
;;   system of equations is accomplished by using the LISP functions from the mathematics package
;;   MATH:DECOMPOSE and MATH:SOLVE.
;; Logic is included to check for conditions which can not be solved directly by these functions and
;;   therefore need special attention. Explanation of these special conditions is included within
;;   the body of the code.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun solve (p1 p2 p3 ppump st-press vt-press head-press a1 a2 a3 a4 a5 a6 a7 it in-sys)
  (cond
    ((and (almost-equal vt-press p2) (almost-equal vt-press p3)      ;; stagnant system
          (almost-equal p1 st-press) (almost-equal ppump st-press)) ;; just return the
     (values p1 p2 p3))                                             ;; previous values

    ((and (almost-equal vt-press ambient-atm) (< p1 (- st-press 1))) ;; initialize system when the
     (setq p1 st-press)                                              ;; function is first called
     (setq p2 ambient-atm)
     (setq p3 ambient-atm)
     (values p1 p2 p3))
    (t
     (cond
       ((and (almost-equal vt-press ambient-atm) (almost-equal p1 st-press) ;; initialize the
             (not (almost-equal ppump st-press)) (zerop it))                ;; recirculating
        (setq p1 (- ppump .1))                                              ;; system after
        (setq p2 ambient-atm)                                               ;; pump is first
        (setq p3 ambient-atm)))                                             ;; turned on
     (cond
       ((and (zerop a1) (almost-equal vt-press ambient-atm)    ;; when backflow occurs
             (not (almost-equal head-press elevation-press))) ;; and the vt-press = atmospheric pressure
        (setq p3 ambient-atm))                                 ;; set p3 to atmospheric pressure

       ((and (zerop a1) (almost-equal vt-press ambient-atm))  ;; when backflow occurs
        (setq p2 ambient-atm)                                  ;; and the upper level piping has drained
        (setq p3 ambient-atm))                                 ;; set p2 to atmospheric pressure

       ((and (not (zerop a1)) (almost-equal vt-press ambient-atm) ;; if the 20 foot riser is just filled
             (almost-equal head-press elevation-press))           ;; p3 is still equal to atmospheric
        (setq p3 ambient-atm))                                    ;; pressure

       ((and (not (zerop a1)) (almost-equal vt-press ambient-atm)) ;; before the 20 foot riser is filled
        (setq p2 ambient-atm)                                      ;; p2 and p3 are equal to atmospheric
        (setq p3 ambient-atm)))                                    ;; pressure

     (cond
       ((almost-equal ppump st-press) ;; if the pump is off set the a1 admittance to zero
        (setq a1 0)))

     (let*
       ((zero-ads (count-zeros (list a1 a2 a3 a4 a5 a7)))) ;; count how many admittances are zero
       (cond
         ((> it 800) ;; Too many iterations exit the solve routine
          (values p1 p2 p3))

         ((> zero-ads 5) ;; if all 6 admittances are zero set p3 to vt-press and return
          (setq p3 vt-press)
          (values p1 p2 p3))
```

```
((> zero-ads 4)                    ;; if 5 admittances are zero determine what to return from
 (multiple-value-setq (p1 p2 p3)   ;; solve by calling the function get-vals->4
   (get-vals->4 p1 p2 p3 ppump st-press vt-press head-press a1 a2 a3 a4 a5 a6 a7))
 (values p1 p2 p3))

((and (> zero-ads 3) (or (zerop a4) (zerop a5))) ;; if 4 admittances are zero and a4 or a5
 (multiple-value-setq (p1 p2 p3)                 ;; are zero, call the function get-vals->3
   (get-vals->3 p1 p2 p3 ppump st-press vt-press head-press a1 a2 a3 a4 a5 a6 a7))
 (values p1 p2 p3))

((and (equalp vt-press ambient-atm) (zerop a3))
 (setq p1 (/ (+ st-press (* ppump (square (/ a1 a2))))
             (+ 1 (square (/ a1 a2)))))
 (values p1 p2 p3))

((and (> zero-ads 2) (or (zerop a4) (zerop a5))                ;; if 3 admittances are zero and
      (and (not (zerop a1)) (not (zerop a4)) (not (zerop a7))) ;; a4 or a5 are zero and in
      (and (not (zerop a2)) (not (zerop a4)) (not (zerop a7))) ;; addition (a1, a4 & a7),
      (and (not (zerop a2)) (not (zerop a5)) (not (zerop a7))));; (a2, a4 &a7) & (a2, a5 & a7)
 (multiple-value-setq (p1 p2 p3)                               ;; are not all zero, call the
   (get-vals->2 p1 p2 p3 ppump st-press vt-press head-press a1 a2 a3 a4 a5 a6 a7)) ;; function
 (values p1 p2 p3))                                                               ;; get-vals->2

((and (zerop a1) (zerop a5))                ;; if this point is reached and both a1 & a5 are
 (setq p1 (+ st-press))                     ;; zero return the appropriate values
 (setq p2 (+ ambient-atm))
 (setq p3 (/ (* vt-press (square (/ a6 a7)))
             (+ 1 (square (/ a6 a7)))))
 (values p1 p2 p3))

(t
 (cond                                                          ;; if this is first iteration &
   ((and (zerop a1)(zerop a2)(zerop a3)(zerop a7)(zerop it))    ;; a1, a2, a3, & a7 are all zero,
    (setq p3 (- vt-press 1))                                    ;; adjust the initial guess of p2
    (setq p2 (- p3 1))                                          ;; and p3 to avoid divergence
    (setq it 1))

   ((zerop a1)
    (cond                               ;; if this is the first iteration and a1 is
      ((zerop it)                       ;; zero adjust the initial guesses for the
       (setq p3 (- vt-press .1))        ;; pressures to avoid divergence
       (setq p2 (- p3 .1))
       (setq p1 (- (+ p2 head-press) .1))
       (setq it 1)))
    (cond                               ;; if both a1 & a2 are zero also set a3 to zero
      ((zerop a2)                       ;; in order to avoid discontinuity and adjust
       (setq a3 0)                      ;; the value of p1
       (setq p1 (+ p2 head-press)))
      ((zerop a3)                       ;; if both a1 & a3 are zero also set a2 to zero
       (setq a2 0)                      ;; in order to avoid discontinuity and adjust
       (setq p1 st-press))))            ;; the value of p1

   ((zerop a3)
    (cond
      ((zerop it)                       ;; if this is the first iteration and a3 is
       (setq p1 (- ppump 1))            ;; zero adjust the initial guesses for the
       (setq p3 (- vt-press 1))         ;; pressures to avoid divergence
       (setq p2 (- p3 1))
       (setq it 1)))
    (cond                               ;; if both a2 & a3 are zero also set a1 to zero
      ((zerop a2)                       ;; in order to avoid discontinuity and adjust
       (setq a1 0)                      ;; the value of p1
       (setq p1 ppump))
      ((zerop a4)                       ;; if both a3 & a4 are zero also set a5 to zero
       (setq a5 0)                      ;; in order to avoid discontinuity and adjust p2
       (setq p2 p3))
      ((zerop a5)                       ;; if both a3 & a5 are zero also set a4 to zero
       (setq a4 0)                      ;; in order to avoid discontinuity and adjust p2
       (setq p2 ambient-atm))))
```

```
        ((zerop a5)
         (cond                                  ;; if a5 is zero and this is the first iteration
           ((zerop it)                          ;; adjust the initial guesses for the pressures
            (setq p1 (- ppump 1))               ;; to avoid divergence
            (setq p2 (- p1 1 head-press))
            (setq p3 (- vt-press 1))
            (setq it 1)))
         (cond
           ((zerop a4)                          ;; if both a5 & a4 are zero also set a3 to zero
            (setq a3 0)                         ;; in order to avoid discontinuity and adjust
            (setq p2 (- p1 head-press))))))      ;; the value of p2

      (let*  ;; set up the vector b and matrix a to solve for x in the equation A x = b
        ((b (make-array 3 :initial-contents
                        `(,(f1 p1 p2 ppump st-press head-press a1 a2 a3)
                          ,(f2 p1 p2 p3 head-press a3 a4 a5 in-sys)
                          ,(f3 p2 p3 vt-press a5 a6 a7 in-sys))))

         (a (make-array '(3 3) :initial-contents
                        `((,(j11 p1 p2 ppump st-press head-press a1 a2 a3)
                           ,(j12 p1 p2 head-press a3) 0)
                          (,(j12 p1 p2 head-press a3) ,(j22 p1 p2 p3 head-press a3 a4 a5 in-sys)
                           ,(j23 p2 p3 a5))
                          (0 ,(j23 p2 p3 a5) ,(j33 p2 p3 vt-press a5 a6 a7 in-sys)))))

         (x (multiple-value-bind (c d)(math:decompose a)(math:solve c d b)))

;;       (x (iterate a b x))) ;; This routine only needs to be used for ill-conditioned matrices
          )                   ;; & has been commented out.

        (cond
          ((< (max (abs (aref x 0))(abs (aref x 1))(abs (aref x 2))) .01)
           (values p1 p2 p3))
          (t
           (setq it (+ it 1))            ;; update the iteration number and obtain the next
           (setq p1 (- p1 (aref x 0)))   ;; value for the unknown pressures and then recursively
           (setq p2 (- p2 (aref x 1)))   ;; call solve until the tolerance is reached.
           (setq p3 (- p3 (aref x 2)))
           (solve  p1 p2 p3 ppump st-press vt-press head-press a1 a2 a3 a4 a5 a6 a7 it in-sys)
           )))))))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This functon could be used if ill-conditioned matrices were being encountered. This does not seem
;;   to be the case, however the code has been left for future reference.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun iterate (a b x)
  (let* ((ax (math:multiply-matrices a x))

         (r (make-array 3 :element-type 'double-float
                          :initial-contents
                          `(,(- (aref b 0)(aref ax 0))
                            ,(- (aref b 1)(aref ax 1))
                            ,(- (aref b 2)(aref ax 2)))))

         (y (multiple-value-bind (e f)(math:decompose a)(math:solve e f r)))


         (xx (make-array 3 :initial-contents
                          `(,(+ (aref x 0)(aref y 0))
                            ,(+ (aref x 1)(aref y 1))
                            ,(+ (aref x 2)(aref y 2))))))
    (cond
      ((< (abs (max (- (aref x 0)(aref xx 0))(- (aref x 1)(aref xx 1))(- (aref x 2)(aref xx 2)))) .0001)
       x)
      (t
       (setf (aref x 0)(aref xx 0))
       (setf (aref x 1)(aref xx 1))
       (setf (aref x 2)(aref xx 2))
       (iterate a b x)))))
```

VII. REFERENCES

1. Denn, M. M., Process Fluid Mechanics, Prentice-Hall, Englewood Cliffs, NJ, 1980.

2. Barron, R. F., Cryogenic Systems, Second Edition, Oxford University Press, New York, 1985.

3. Martinelli, R. C., and R. W. Lockhart, "Proposed Correlation of Data for Isothermal Two-Phase Flow in Pipes", Chemical Engineering Progress, 45, 1, 39, 1949.

4. Shen, P. S., and Y. W. Jao, "Pressure Drop of Two-Phase Flow in a Pipeline with Longitudinal Variations in Heat Flux, Advances in Cryogenic Engineering, 15, Plenum Press, New York, 1970.

3. Burden, R. L. and J. D. Faires, Numerical Analysis, Third Edition, Prindle, Weber & Schmidt, Boston, 1985.