NASA Contractor Report 191063

Analysis of the Space Shuttle Main Engine Simulation

J. Alex De Abreu-Garcia and John T. Welch
University of Akron
Akron, Ohio

January 1993

Prepared for
Lewis Research Center
Under Grant NAG3–1031

National Aeronautics and Space Administration

TABLE OF CONTENTS

ANALYSIS OF THE SPACE SHUTTLE MAIN ENGINE SIMULATION

J. Alex DeAbreu-Garcia and John T. Welch
Univeristy of Akron
Akron, Ohio 44325

ABSTRACT

This report analyzes the digital code used to simulate dynamic performance of the Space Shuttle Main Engine. This simulation program is written in Fortran. The purpose of the analysis is to identify a means to achieve faster simulation execution, and to determine if additional hardware would be necessary for speeding up the simulation. The analysis included the use of custom integrators based on the Matrix Stability Region Placement method. In addition to speed of execution the accuracy of computations, the useability of the simulation system, and the maintainability of the program and data files were examined. A revised code implementing the study recommendations was implemented but not verified for accuracy.

1. INTRODUCTION

This is a final report on an analysis of the Space Shuttle Main Engine Program, a digital simulator code written in Fortran. The research was undertaken by the authors in ultimate support of future design studies of a shuttle life-extending Intelligent Control System (ICS). These studies are to be conducted by NASA Lewis Space Research Center.

The primary purpose of the analysis was to define the means to achieve a faster running simulation, and to determine if additional hardware would be necessary for speeding up simulations for the ICS project. In particular, the analysis was to consider the use of custom integrators based on the Matrix Stability Region Placement (MSRP) method.

In addition to speed of execution, other qualities of the software were to be examined. Among these are the accuracy of computations, the useability of the simulation system, and the maintainability of the program and data files.

Accuracy involves control of truncation error of the methods, and roundoff error induced by floating point operations. It also involves the requirement that the user be fully aware of the model that the simulator is implementing.

The useability of the simulation system affects the productivity of ICS designers. The simulation system should support a large number of runs, with reliable tracking, archiving, and retrieval of run results. Setting up simulation runs and reviewing results should be easy to do, and not be subject to hard to detect errors.

Simulation of intelligent control methods for the main shuttle engine is expected to involve considerable manipulation of the shuttle engine control systems. The ICS design studies will take into account new perturbation modes, and may consider variations in shuttle engine design.

Thus the studies will place high maintainability requirements on the simulation code. The SSME Simulation Program must be readable, and organized to permit easy and safe alteration.

Long exposure to the SSME code compels us to acknowledge with special appreciation the work of Ten-Huei Guo of NASA Lewis Space Center, whose extensive interpretive comments made the source intelligible enough for analysis. We have preserved most of his commentary in the report version. A careful reading of the SSME simulation code, as received by NASA Lewis Space Center, would

affirm that our criticisms of the coding do not apply to Dr. Guo's work, or that of NASA Lewis Research Center.

Organization of this Report

The report begins with a summary of results and recommendations. It ends with a suggested implementation and tuning sequence for the simulation.

The next three chapters are devoted to numerical integration methods. Multi-step integration methods and the basis for their analysis are explained in Chapter 3. The next chapter presents the analysis results obtained in the study. Chapter 5 then describes the integration function module of the report version, explaining the numerical and coding techniques implemented.

The remaining chapters cover interpolation methods, energy balance monitoring and control, exponentiation, and simulation output.

The remainder of the report is a set of appendices defining a Fortran 77 code and set of supporting data files. This represents the form of the simulation recommended for ICS design studies. This version is also delivered with the report as ASCII files on diskette. For convenient reference, the appended listing will be called "the report version" throughout this report. The code and run data we received from NASA Lewis, for purposes of study, we will call "the study version".

The research was expected to produce analysis results, rather than a revised code. However, it became obvious on examination of the study version that the line-by-line optimization and the new function modules required could not be adequately defined without a full code revision. The report version is a revised code fully implementing the immediate recommendations, but is not verified to be in accurate running condition, as this report goes to print.

The report version code is itself an important product of the research study. It shows in specific detail how the recommendations may be followed, and can contribute directly to their full implementation. If a majority of the recommendations are accepted, then the report version would certainly be a time-saving source of implementation code, if not the basis for a rapid implementation.

2. SUMMARY OF RESULTS

Study of the SSME Simulation Program revealed many problems that could seriously impede ICS development, if left uncorrected. In the form given to us, the program

- is unacceptably slow in running time.
- produces too much output for reasonable review and handling of results.
- is difficult to set up accurately.
- is of questionable numerical accuracy in some subsystems.
- is difficult to maintain as a testbed simulator for new control methods.

The report version of the system, reprinted in Appendix A, addresses all of these issues in specific terms.

Speedup Results

Slow running time was a primary focus of the study. Lack of significant improvements in this area would have indicated a requirement for hybrid computing hardware. The original proposal anticipated analyis of implementation technique in ADSIM on the ADI AD100 system.

Instead, many opportunities for speedup were found, which can be exploited independently, and therefore work as multiplying factors to produce a very high expected speedup. A negative aspect of this is that the opportunities were so ample and interwoven, that we were hard pressed within the time available to adequately define all of the contributing changes, or to construct more than a conservatively low estimate of the achievable speedup. We predict a speedup in running time of about 20. It could be higher.

With this kind of speedup available, the use of special purpose hardware for the SSME Simulation is not recommended. We believe the additional programming costs and limited access associated with special hardware can be avoided in this case. In fact, with the decrease in bulk of program and data that is demonstrated by the report version, we can recommend that the ICS project use desktop 386 PC Systems or similar workstations to house the SSME simulation, perhaps with management of results data on mainframe systems.

The identification of many available methods of speedup leads us to recommend also a course of action that exploits the most readily available speedup first, and defers the most difficult until it is known to be necessary. Most of the available speedup can be achieved with the following steps, listed by decreasing expected effect:

1. Replacing formatted character output of every output variable by selective output in unformatted binary form. Conversion can be done offline, along with printing and/or plotting. Conversion and output transfer of character strings representing data, and output of headings identifying data, compares with computations in running time. With no changes here, significant improvements in computation time would be severely limited in effect.

2. Replacing Euler integration with a modified version of the multistep Adams-Bashforth Second Order integration formula (MAB2). This integration method, developed by our colleague, Dr. Tom Hartley, permits a step size five times greater than the Euler method used in the study version. An integration module supporting this change is provided in the report version.

3. Replacing interpolation routines implemented with varying levels of efficiency with a single set of routines coded as efficiently as possible. These routines are provided in the report version.

4. Improving code which computes results unnecessarily, code which uses unnecessarily many operations, code which uses divisions where equivalent multiplications are available, and code which makes unnecessary reference to costly special functions. These improvements have been made in the report version.

5. Monitoring the convergence of energy balancing loops more closely and introducing under-relaxation and over-relaxation factors to minimize convergence iterations. A convergence control module is provided in the report version.

6. Confining the use of double precision to the integration process where it is actually required. Using exact integer arithmetic, rather than double precision, for the accurate advancement of time.

7. Replacing high precision, software supported real exponentiation by linear interpolation or by forms better supported in hardware on the host computer system.

All of the above methods are implemented, with every module

undergoing revision, in the report version of the code.

It is the conclusion of the study that, due to changing modes of the model, and its size, a full matrix version of MSRP would not be practical. The number of variables would make both the analysis and implementation impractical.

Step sizes of up to 100 times those achievable with the modified AB2 can be obtained with local versions of MSRP, but at the cost of a more complex and less flexible integration process, requiring additional analysis of the SSME model. The theoretical gain in step size could not be fully realized, since important dynamics of the system would not be followed at such large step sizes. Also there is the fact that evaluation of the SSME model at every time step requires sensitive energy balancing. Since the effect of SSME dynamics on energy balancing at larger time steps is unknown, it seems more prudent to implement the other methods of speedup before undertaking a conversion to local MSRP. The required analysis for local MSRP is continuing beyond the study, in master's thesis work described later.

The report version integration module allows for the progressive extension to higher order multistep integrators, and can accomodate local forms of MSRP, should they prove later prove desirable.

Accuracy results

The time step set in the study version was barely within stability limits for the integration method used. This generally implies that considerable distortion is being introduced by the numerical integration process itself. Reasonable accuracy limits on the time step are definitely exceeded in the valve dynamics module, according to the analysis, even though a higher sampling rate is used in this module.

The analysis shows that the recommended modified AB2 is an integration method better suited to the shuttle engine model, and produces less integration distortion than Euler's method, even when operated at the larger time step recommended for decreased running time.

Two potential floating point roundoff problems are noted, and corrected in the report version. All integration was converted to use a double precision accumulator, to avoid losing significant digits when small integration increments are added to large integrand variables in floating point.

The same floating point roundoff effect causes a significant drift in the time scale as the time step is repeatedly added to

the current time, but double precision is not the best answer to this problem. Instead, the report version keeps time as an integer number of step increments, which stays exact.

A potentially serious source of inaccuracy in the study version was the practice of limiting the input variable to keep it within the input range of an interpolation table. This has the effect of extrapolating the table by a constant value, without warning to the user when it is being done. Since interpolation tables can be readily extended, with little storage cost, there is no justification for this policy. The report version interpolators give a specific complaint and stop the simulation when a range is exceeded. The user can then extend the table by word processing an ASCII file. The simulation can then be rerun without change.

Finally, accuracy of the simulation is potentially enhanced by a new method of storage recommended for tabulated functions. The new method allows more points to be assigned to one function, without incurring a storage penalty on all other functions, and with minimal impact on running time.

Useability Recommendations

A sufficiently fast and accurate simulation program could still fail as a tool for the ICS desigh studies if it were too difficult to operate. The study version had several other major shortcomings of this nature, which are addressed in the report version.

One failing in this area has already been mentioned, an inflexible output system that dumps everything on every run. Adding to this burden on the program's users is the automatically echoed output of every input parameter. This flood of mostly unread data is no virtue. Aside from the running time it costs, uncontrolled amounts of output inhibits simulation activity and makes archiving and retrieval of simulation results unmanageable. Results are consequently lost or never obtained.

The recommended system of selectable binary output, along with a faster running simulation, offers a better approach to data archiving. Simulations can be rerun to repeat and amplify interesting results. The requirement to restart the simulation, which is difficult to support with higher order integrators, can be dropped.

Quality of simulation runs can be improved, when tuning of conditions leading up to archived results can be done without running at full output mode. The report version decouples the selection of input parameters and output configuration,

supporting this mode of operation.

Selected binary data is a much more practical form of archived simulation data than fully annotated hard copy. Results can be viewed and correlated in new ways, deferring hard copy until significant results are observed.

As important as running time and manageable archiving of data are, the opportunities for greater exploitation of NASA Lewis' graphics hardware in the interpretation of simulation results, made possible by a more flexible output system, may contribute as much to the success of the ICS design studies.

Another hurdle to the effective use of the SSME simulation by an ICS design team is the study version's file design for the bulk of the input data. This design mixes run parameters and interpolation data in the file 'dtminp.dat'. Run parameter values appear in this file without identifiers or limiters of any kind. Users are rightly advised to be cautious in altering the file.

We recommend replacing 'dtminp.dat' by two files, one for interpolation tables and another for run parameters. Both files contain identifers and field markings that are not read by the program, but make it easy to find the field to be changed and to be sure that the change is made within the desired field. Since thesee files contain identifying headers, they can serve as documentation for the run parameters and function generation data used in a run. This relieves the program of echoing input data, with identifiers. Examples are provided in Appendix B.

Finally, we consider it a potential useability hazard to depend on Fortran IV, through use of obsolete language features like NAMELIST. The superseding Fortran 77 standard was adopted in 1978, and only the most forgiving of maintained compilers still accepts Fortran IV. Reliance on Fortran IV unnecessarily limits the portability of the program, and will shut off, more and more, opportunities to house it in more productive ways.

Evidence of the above can be offered from our own experience in the study. We were unable to get our copy of the original code to run without eliminating NAMELIST data, although it compiled without complaint. No manual was available, because the compiler we had to use is not actively supported at our facility. We did get around it, but the experience makes the point.

Having the use of the latest in compilers is an advantage soon realized in working with the SSME code. Many artifacts of past changes, which should have been removed when their function became obsolete, were identified in the listings.

The report version is written in Fortran 77. NAMELIST input was absorbed into the run parameter file, where it is almost as convenient to access as it was in the NAMELIST file 'start4.dat'. An added advantage is that there is only one place to set input parameters. The danger in having two entry points for a data item, with one overriding the other, is obvious.

An interim plan to divide the input parameter file into two similar files was abandoned. The idea was to have frequently changed parameters in a smaller file, and infrequently changed parameters in a longer one. It complicates parameter setting to have two possible sites for the value to be changed. With "find" commands in modern word processors, this division is unnecessary.

Maintenance for ICS Design Studies

A serious failing of the study version from the maintenance standpoint was that all executing code was packaged as a single file, a form which forces a complete compilation to accomplish any change in any module. In the report version, source code for each major subroutine is on a separate file. Only changed modules need be compiled on a revision.

The study version included a line number on every line, a format which requires most systems to support the blank space with space characters. Files were cut by about 45% in size by eliminating nine out of every ten line numbers, ending all lines with a line return character following the last nonspace character. Every tenth line number was retained, to allow for reference in the report version back into the study version. These line numbers should also be eliminated when this function is no longer served.

Better overall readability of the code was obtained in the report version by insertion of punctuating white space. Many originators seemed to think it a virtue to pack all of the information into as little space as possible.

Readability was improved considerably in some sections by reduction of spaghetti code sequences to structured code, using nested block IF constructions. This improvement would not be available in Fortran IV. An especially counterproductive style used in some of the study version's "spaghetti" sequences parked GO TO statements far to the right of the line. These statements often escape the attention of readers, misdirecting them as to the effect of the code.

Another change made for the sake of readability was to eliminate selection of a special mode of a subroutine by the value of an integer argument. For the major system component

subroutines, the subroutine was divided into an initialization entry and a time step entry, using different entry names for the two functions. This also eliminates the subroutine having to test to see which mode it is in.

Finally, it should be observed that the study version's output system places an unnecessary burden on the maintenance of the simulation. In the main program of the study version, output variables were renamed and saved again, and new combinations of output variables were computed for the sake of output alone. In some cases, data was saved again on separate files for the expressed purpose of plotting. These represent special needs for simulation output, many probably no longer current, but not removed. Undoubtedly, less obvious extra computations of this type remain in the simulation.

In the recommended output system, no alterations of the simulation program are made to create plot files or to output computed combinations of output variables. Instead, small programs are written for these purposes, all reading the same binary data files, but presenting the data in differing ways offline. These programs have in common the SSME output file format. Some of them, designed to run on machines other than the host of the SSME compiler, might require binary floating point conversions of the data. Each such program builds upon the SSME simulation system, and does not require additional maintenance on it.

A Procedure for Tuning the SSME Simulation

The the report version of the SSME Simulation anticipates the following recommended procedure for tuning the simulation to its highest level of efficiency for the ICS design studies:

1. Run the report version, comparing with previous study version results. Account for differences. Changes made for faster execution should not significantly affect results. Euler's method remains as the default integration method so that these verification runs can be made before changing integration methods.

2. Decrease the step size in the valve dynamics module to recommended values. This is to establish a baseline of runs known to be more accurate, but consistent with previous results and real data.

3. Examine energy balance convergence and adjust relaxation factors, under-relaxing slow converging sections and over-relaxing fast converging sections. Look into relaxation of convergence criteria, and for accuracy problems which might account for residual slow convergence.

4. Attempt to eliminate cubic spline interpolation, and to select the most efficient form of real exponentiation for host system.

5. Switch to recommended integration methods and increase step sizes gradually, monitoring effects on energy balancing convergence and adjusting relaxation factors accordingly. Continue to recommended step sizes and beyond, pulling back to assured accuracy levels.

3. LINEAR MULTISTEP INTEGRATORS: A REVIEW

This chapter gives a brief review of the derivation of and definitions associated with linear multistep methods. Although implicit methods provide more accurate results than explicit methods, they require greater computational times. Therefore, implicit methods are, in general, not used in real-time applications. As a result, only explicit methods will be considered here. It should be pointed out that the material presented in this chapter borrows heavily from the results of references [1] and [2].

In the study version of the SSME simulation numerical integration is done using Euler's method. One of the main reasons being that Euler's method is well understood and simple to implement. However, it will be seen that care must be exercised when choosing the Euler's integration timestep to provide a more accurate simulation. Increased accuracy requires that very small timesteps be used at the expense of a considerably slower running simulation. In this analysis the concept of stability region of an integrator is used to show that it is possible to obtain a faster and more accurate simulation of the SSME should other linear multistep methods be used. Custom designed integrators will also be considered.

This chapter is organized as follows: first, the general linear multistep method is developed. Included in this development are the important notions of order, accuracy, consistency, and zero-stability. Both Euler's method and AB-2 are derived. Then, the concept of stability region of an integrator is discussed. Comparisons are made between the stability region of Euler's integrators and the stability region of AB-2 type integrators. It is shown that AB-2 integrators may result in a more accurate and faster running simulation. To guarantee a stable simulation of those modules whose eigenvalues have large imaginary parts with respect to their real parts, a modified AB-2 integration method is presented [3]. To conclude this analysis, the applicability of the matrix stability region placement method of [4] to the SSME simulation is discussed.

A. LINEAR MULTISTEP METHODS

The general linear multistep method can be written as follows:

$$\sum_{j=0}^{r} a_j x_{k+j} = T \sum_{j=0}^{r} b_j \dot{x}_{k+j} \quad . \tag{1}$$

where $\dot{x}_k$ is the first derivative of x_k, a_j and b_j are constants, a_0 and b_0 are not allowed to be both zero, and, to avoid ambiguity, it is assumed that $a_r=1$.

The method of equation (1) is said to be explicit if $b_r=0$, and implicit if $b_r\neq 0$. For an implicit method the present value of the output is a function of present and past values of the output and the input. For an explicit method only past values of the output and the input are required to determine the present value of the output. Here

we will be concerned only with explicit methods as implicit methods are not suitable for real-time simulation.

Associated with the linear multistep method of equation (1) is the operator:

$$L[x(t),T]=\sum_{j=0}^{r}[a_j x(t+jT)-Tb_j \dot{x}(t+jT)] \quad . \tag{2}$$

where $x(t)$ is an arbitrary continuously differentiable function on some closed interval.

Expanding $x(t+jT)$ and $\dot{x}(t+jT)$ as Taylor series about t, and grouping similar terms yield:

$$L[x(t),T]=C_0 x(t)+C_1 Tx^1(t)+.....+C_q T^q x^q(t)+..... \quad . \tag{3}$$

where C_q : $q=0,1,2,....$ are constants, and $x^q(t)$ refers to the qth derivative of $x(t)$.

The integrator of equation (1) is said to be of order p if the C_i coefficients are such that C_i : $i=0,1,2,....,p$ are all zero and $C_{p+1}\neq 0$. Clearly, knowledge of the C_i's allows the derivation of linear multistep methods of a given order and structure. It turns out that the order of accuracy of the integrator is the same as the order of the operator $L[.]$ of equation (3). Notice that this order is precisely the number of constants C_i which are identically zero. From equation (3) it is possible to determine expressions for the constants C_i as follows:

$$C_0=a_0+a_1+a_2+.....+a_r$$

$$C_1=a_1+2a_2+.....+ra_r-(b_0+b_1+b_2+.....+b_r) \tag{4}$$

$$C_q=\frac{1}{q!}(a_1+2^q a_2+....+r^q a_r-\frac{1}{(q-1)!}(b_1+2^{q-1}b_2+.....+r^{q-1}b_r) \quad q=2,3,.... \quad .$$

B. EULER'S METHOD

Euler's method can be easily derived by letting $r=1$ in equation (1); that is, a one step method. Then equation (1) implies that:

$$a_0 x_k+a_1 x_{k+1}=b_0 T\dot{x}_k+b_1 T\dot{x}_{k+1} \quad . \tag{5}$$

Using the expressions (equation (4)) for the C_i's it follows that:

$C_0=a_0+1$ (recall that $a_r=a_1=1$) $\Rightarrow C_0=0$ if $a_0=-1$

$C_1=1-b_0-b_1 \Rightarrow C_1=0$ if $b_0=1$ (recall that $b_r=b_1=0$ for an explicit method)

$C_2=1/2-b_1 \Rightarrow C_2=1/2$ since $b_1=0$.

Then, substituting these values into equation (5) yields the already familiar Euler's integration method:

$$x_{k+1}=x_k+T\dot{x}_k \Rightarrow H(z)=\frac{T}{Z-1} . \qquad (6)$$

Clearly, Euler's method is only first order accurate as C_0 and C_1 are both zero while C_2 is different from zero.

C. ADAMS-BASHFORTH TWO STEP METHOD (AB-2)

Following the procedure just used to derive Euler's method, an AB-2 integrator can be obtained by letting r=2 in equation (1) and solving equation (4) for the appropriate C_i's; that is,

$$a_0x_k+a_1x_{k+1}+a_2x_{k+2}=Tb_0\dot{x}_k+Tb_1\dot{x}_{k+1}+Tb_2\dot{x}_{k+2} . \qquad (7)$$

$C_0=a_0+a_1+a_2$ (recall that $a_r=a_2=1$)

$C_1=a_1+2a_2-b_0-b_1-b_2$ (recall that $b_r=b_2=0$ for an explicit method)

$C_2=1/2(a_1+4a_2)-(b_1+2b_2)$

$C_3=1/6(a_1+8a_2)-1/2(b_1+4b_2)$

Substituting $a_2=1$ and $b_2=0$ in these equations gives:

$C_0=a_0+a_1+1 \Rightarrow a_0+a_1+1=0$

$C_1=a_1+2-b_0-b_1 \Rightarrow a_1+2-b_0-b_1=0$

$C_2=1/2(a_1+4)-b_1 \Rightarrow (a_1/2)+2-b_1=0 \Rightarrow a_1=-4+2b_1$

$C_3=1/6(a_1+8)-1/2b_1$

Solving the resulting set of equations yields the following values: $a_0=0$, $a_1=-1$, $b_0=-0.5$, $b_1=1.5$, and $C_3=5/12$. Therefore, the two-step Adams-Bashforth (AB-2) integration method can be written as:

$$x_{k+2}=x_{k+1}+T(1.5\dot{x}_{k+1}-0.5\dot{x}_k) \Rightarrow H(z)=\frac{T(1.5z-0.5)}{z^2-z} . \qquad (7)$$

Clearly, AB-2 is second order accurate as C_0, C_1, and C_2 are zero while C_3 is nonzero. This implies that for a given timestep T AB-2 will result in a more accurate simulation than Euler's method.

D. CHARACTERISTICS OF LINEAR MULTISTEP METHODS

Two important concepts associated with linear multistep integrators are the concepts of consistency and zero stability. A

linear multistep method is said to be consistent if it has order greater or equal to one (1). Zero stability is defined via two polynomials. These polynomials are usually referred to as the first and second characteristic polynomials, and are given as:

$\rho(\zeta)=\sum_{j=0}^{n} a_j \zeta^j$, referred to as the first characteristic polynomial

$\sigma(\zeta)=\sum_{j=0}^{n} b_j \zeta^j$, referred to as the second characteristic polynomial

The method is said to be consistent if the first characteristic polynomial always has a root at +1. The method is said to be zero-stable if the roots of the first characteristic polynomial $\rho(\zeta)$ lie on or inside the unit circle with any root on the unit circle being simple. And the method is convergent if and only if it is consistent and zero-stable.

E. STABILITY REGION OF AN INTEGRATOR

Another important concept in the analysis of linear multistep methods is that of stability region. Essentially, the stability region of an integrator is that region where the λT-product should lie in order for the simulation to be stable. Within this region the degree of accuracy of the simulation changes depending on where the λT-product is located. Thus, when designing integrators for a specific application it is important to use the integrator's stability region as a criterion for determining the maximum allowable timestep T. This will not only guarantee a stable simulation but also an accurate one. To determine the stability region of a given integrator it is expedient to work in the frequency domain. Hence, consider the Z-transform of equation (1):

$$\left[\sum_{j=0}^{r} a_j z^j\right] X(z)=\left[T\sum_{j=0}^{r} b_j z^j\right]\dot{X}(z)$$

$$\Rightarrow \rho(z)X(z)=T\sigma(z)\dot{X}(z) \qquad (8)$$

From this equation it follows that the open loop transfer function of the integrator is:

$$X(z)=\frac{T\sigma(z)}{\rho(z)}\dot{X}(z) \qquad (9)$$

Now, the frequency response of the integrator is obtained by replacing z by exponential(jwT); that is,

$$z=e^{j\omega T} \qquad (10)$$

The performance of the integration process on an actual dynamic system is determined using closed loop stability analysis. This closed loop stability analysis is accomplished by introducing the linear test equation:

$$\dot{x}=\lambda x+u \tag{11}$$

$$\Rightarrow \dot{X}(z)=\lambda X(z)+U(z)$$

To close the loop substitute the above linear test equation into equation (8) to yield the closed loop transfer function, viz.

$$\rho(z)X(z)=T\sigma(z)[\lambda X(z)+U(z)]$$

$$\Rightarrow X(z)=\frac{T\sigma(z)}{\rho(z)-\lambda T\sigma(z)}U(z) \tag{12}$$

Clearly, the stability of the integrator is determined by the poles of the closed loop transfer function of equation (12); that is, the root of the equation:

$$\rho(z)-\lambda T\sigma(z)=0 \tag{13}$$

Notice that in equation (13) the product λT can be thought of as the gain of the closed loop system. Thus, a root locus can be plotted. Therefore, it is always possible to determine the region of the λT-plane where the integration is stable by considering the expression:

$$\lambda T=\frac{\rho(z)}{\sigma(z)} \tag{14}$$

Hence, mapping the z-plane unit circle into the λT-plane the stability region of the integrator can be obtained, viz.

$$z^n=e^{jn\theta}=\cos(n\theta)+j\sin(n\theta) \tag{15}$$

The stability regions corresponding to Euler's method (AB-1) and AB-2 method are shown in Figures 1 and 2. For a given integrator the simulation will be stable provided that the λT-product is inside the integrator's stability region. For the simulation to be accurate, the λT-product should be within the region where the "squares" are the least distorted. Notice that although Euler's integrator has a larger stability region, it requires very small timesteps in order to produce an accurate simulation (The λT-product should lie within the squares that are closest to the origin). This is especially the case when the systems being integrated have eigenvalues of large magnitude. This problem is further aggravated by situations where the eigenvalues of the system under consideration are complex with large imaginary parts relative to the real parts. As the stability region of Euler's method touches only the origin of the λT-plane, it is almost impossible to guarantee that the timestep T can be chosen small enough so that the

λT-product lies inside the stability region. In sharp contrast, the stability region of AB-2, although smaller than that of Euler's method, allows the use of larger timesteps with improved accuracy. Moreover, the problem associated with large complex eigenvalues is alleviated due to the fact that AB-2's stability region is asymptotic to a larger portion of the imaginary axis of the λT-plane. Thus, it can be said that although Euler's method is simple to use and implement, care must be exercised when choosing the integration timestep to guarantee a stable and accurate simulation. It should be pointed out that AB-2 integrators do not provide a significant improvement over Euler's method in situations such as those just mentioned. However, using AB-2 it is possible to improve running times by a factor of about five with improved or comparable accuracy.

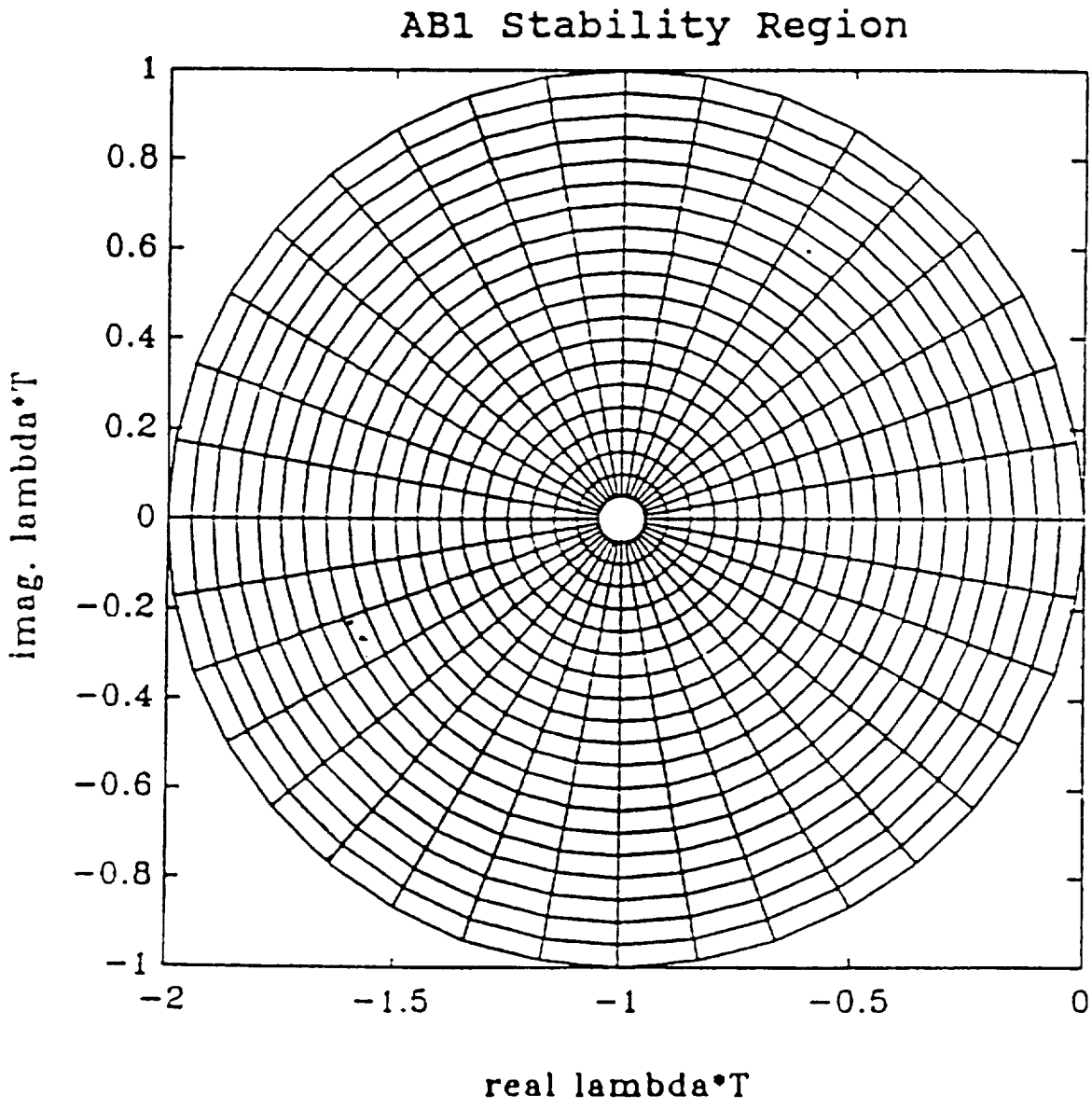


Figure 1. Stability Region of Euler's Method (AB-1)

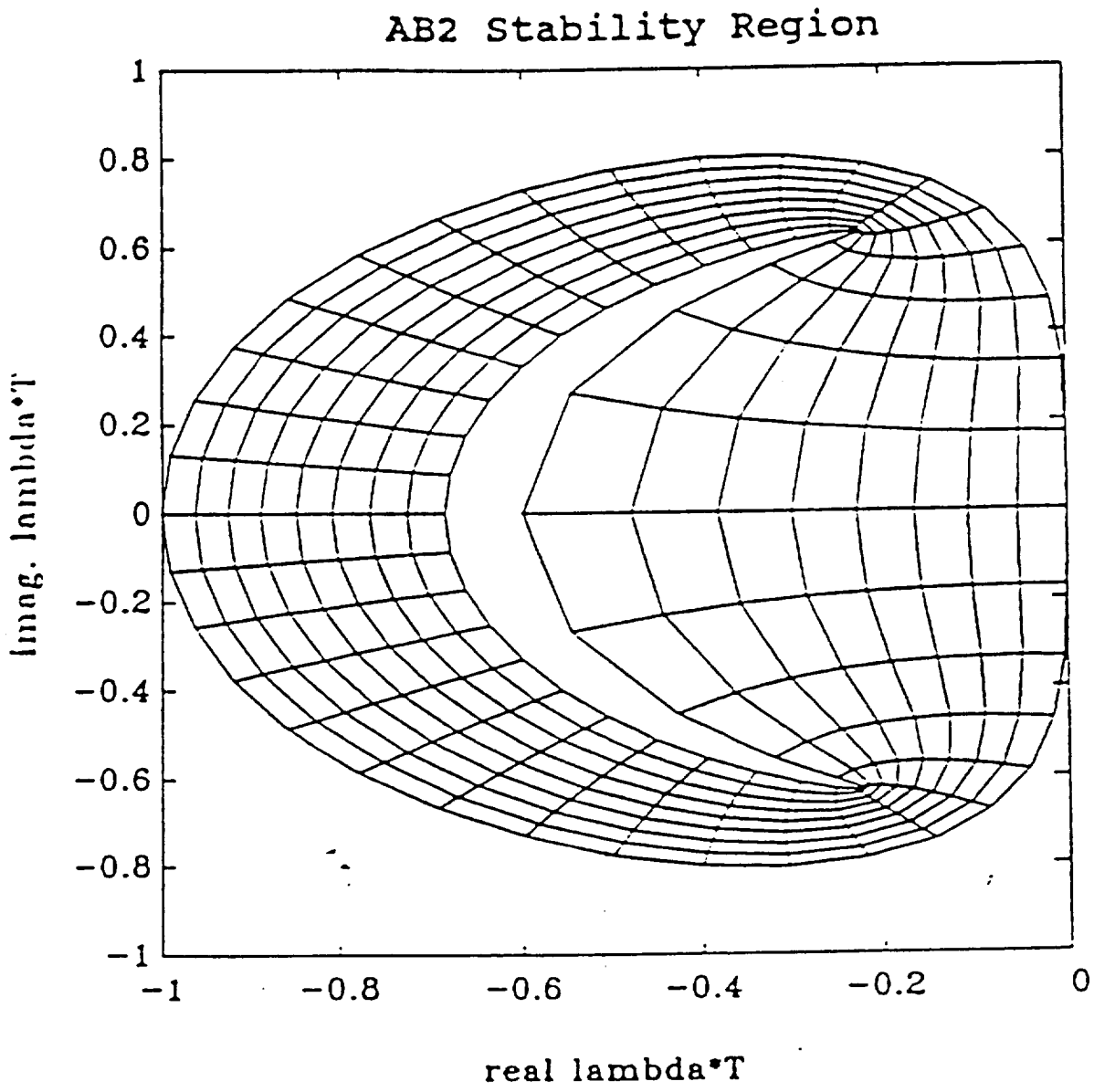


Figure 2. Stability Region of AB-2

F. MODIFIED ADAMS-BASHFORTH TWO STEP METHOD (MAB-2)

To obtain more stable and accurate simulations of systems whose eigenvalues are complex with large imaginary parts relative to their real parts a modified AB-2 method can be used. This modified method can even be used in situations where purely imaginary roots are to be stably integrated [3]. The modification of AB-2 is possible due to the flexibility presented by the C_i's of equation (4). Notice that this flexibility is greatly reduced if it is demanded that the integration method be of maximal accuracy (that is, that as many C_i's as possible be set to zero). However, if accuracy requirements can be relaxed, then the extra freedom that is gained can be used to tailor the integration method to the particular system's dynamics. In this case it is desired to modify the AB-2 method so that the imaginary axis is just enclosed by the stability region of the new integrator.

Recall that AB-2 is second order accurate. This implies that C_0, C_1, and C_2 are all equal to zero while C_3 is different from zero ($C_3=5/12$). Recall further that, for a two step explicit method, the C_i's satisfy the following set of equations:

$$\begin{aligned} C_0 &= a_0+a_1+1 \\ C_1 &= a_1+2-b_0-b_1 \\ C_2 &= 1/2a_1+2-b_1 \\ C_3 &= 1/6a_1+4/3-1/2b_1 \end{aligned} \tag{16}$$

Next, suppose that the order of accuracy is relaxed so that only C_0 and C_1 are required to be zero. Since there are three equations in four unknowns (the equation involving C_3 does not count in this case) it follows that there are two free parameters. Without loss of generality these free parameters can be chosen to be $a_0=\alpha$ and $b_0=\beta$. Since C_0 and C_1 are both zero, equation (16) can be solved for a_1 and b_1 to yield the modified AB-2 integrator (MAB-2), viz.

$$x_{k+2}=(\alpha+1)x_{k+1}-\alpha x_k+T(1-\alpha-\beta)[\dot{x}_{k+1}+\beta/(1-\alpha-\beta)\dot{x}_k)] \ . \tag{17}$$

The transfer function of the MAB-2 integrator (equation (17)) is:

$$H(z)=\frac{T(1-\alpha-\beta)[z+\beta/(1-\alpha-\beta)]}{z^2-(1+\alpha)z+\alpha} \ . \tag{18}$$

Notice that equation (17) is indeed the transfer function of an integrator. This is seen once it is realized that the poles of equation (18) are located at $z=1$ and $z=\alpha$. It should be kept in mind that the pole at $z=\alpha$ must lie inside the unit circle to maintain open loop stability. Therefore, α is allowed to vary between +1 and -1. Notice as well that the zero of equation (18) lies at $z=\beta/(1-\alpha-\beta)$. The flexibility offered by the arbitrary location of this zero allows an entire family of integrators to be generated for different pole locations. It turns out that setting α to zero and β to -0.6 results

in an integrator whose stability region closely approximates that of AB-2. The only difference between the stability regions of MAB-2 and AB-2 is that the stability region of MAB-2 just encloses the imaginary axis of the λT-plane. Thus, allowing systems with complex poles and purely imaginary poles to be stably simulated. The actual integrator and corresponding transfer function are given as:

$$x_{k+2}=x_{k+1}+T(1.6\dot{x}_{k+1}-0.6\dot{x}_k) \Rightarrow H(z)=\frac{T(1.6z-0.6)}{z^2-z} \quad . \tag{19}$$

The difference between the stability region of AB-2 and the stability region of MAB-2 can be clearly seen from Figure 3. Although these stability regions are almost identical, there are several important points to be noticed. The stability region of MAB-2 is slightly shifted to the right of the λT-plane and thus it encloses the jw-axis of this plane. The portion of the stability region where the λT-product gives an accurate and stable simulation is slightly distorted relative to the corresponding region for an AB-2 integrator. This implies that MAB-2 is not as accurate as AB-2. This should not come as a surprise as MAB-2 was derived assuming that accuracy could be traded off with the ability to stably integrate systems with either purely imaginary eigenvalues or eigenvalues with large imaginary parts. However, in actuality MAB-2 is expected to produce as good a simulation as AB-2. The reason for this being that the level of distortion of the stable and accurate region of MAB-2 is not too severe. This can be easily seen by considering the C_i coefficients. Recall that these coefficients determine the order of accuracy of the integration method. Some simple arithmetic gives that for MAB-2 these coefficients are 0,0,-0.1, and 11/30 corresponding to C_0, C_1, C_2, and C_3 respectively. A comparison of these values with those of AB-2 is given in Table 1 below.

Table 1. Order of Accuracy of AB-1, AB-2, and MAB-2

C_i Coefficient $a_r=1$ $b_r=0$	Integration Method		
	AB-1	AB-2	MAB-2
$C_0=a_0+a_1+a_2$	0	0	0
$C_1=a_1+2a_2-b_0-b_1-b_2$	0	0	0
$C_2=1/2a_1+2a_2-b_1-2b_2$	1/2	0	-1/10
$C_3=1/6a_1+8/6a_2-1/2b_1-2b_2$	*	25/60	22/60
Order of Accuracy	first	second	first

Notice that although MAB-2 is first order accurate, the difference between the C_i's for MAB-2 and the C_i's for AB-2 is rather small. Thus, MAB-2 should perform almost as well as AB-2.

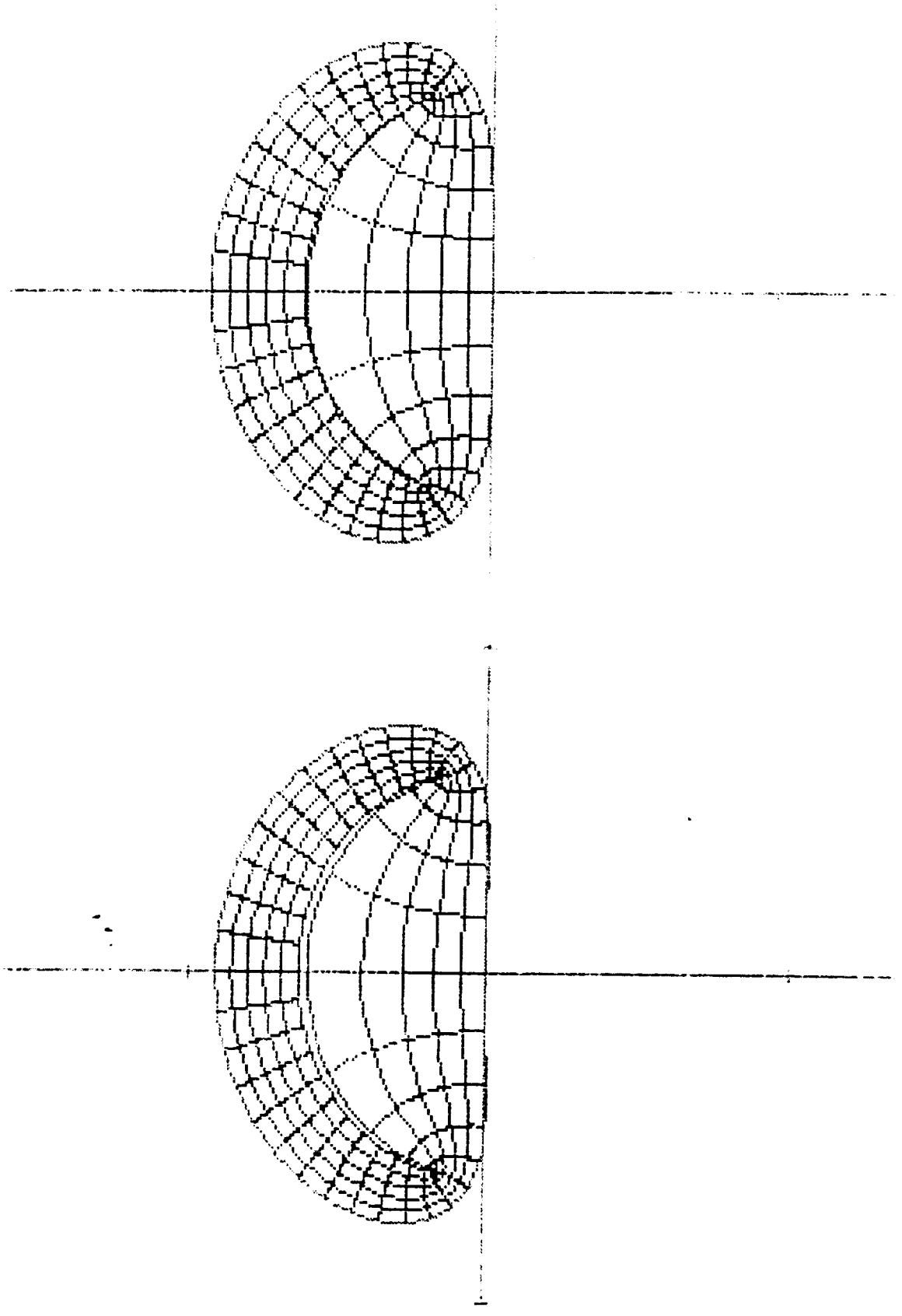


Figure 3. Stability Regions of AB-2 (top) and MAB-2 (bottom)

G. TWO STEP MATRIX INTEGRATORS (MSRP-2)

In this section the two step matrix integrator (MSRP-2) presented in [4] is reviewed. This method is the generalization of the method of [5] to vector systems of differential equations. It is shown that this type of integrators can be very useful in the real-time simulation of stiff systems. The material presented in this section is essentially that of [4] however.

When performing numerical integration it is important to maintain the overall system transient response while the integration operator maintains the character of an integrator. This is normally done by deriving integrators that reproduce the transient response better as the timestep goes to zero [5]. However, for a specific system, the SRP method derives an integrator that improves the transient response for a given nonzero timestep.

The SRP method has the minimum number of coefficients necessary to maintain both the transient response and the integration property for a scalar system, while also allowing arbitrary choice of timestep [5]. MSRP-2 extends the SRP method to vector systems.

The linear MSRP-2 integrator can be written as:

$$x_{k+2}=-A_1 x_{k+1}-A_0 x_k+T(B_1 \dot{x}_{k+1}+B_0 \dot{x}_k) \quad . \tag{20}$$

where T is the integration timestep, kT is the time, x is the n-dimensional state vector at time t, and A_i and B_i : i=0,1 are the n×n regression coefficient matrices.

As in the general linear multistep method, the performance of MSRP-2 is evaluated using a linear test equation of the form:

$$\dot{x}_k=Jx_k+Gu_k \quad . \tag{21}$$

where, u is an m-vector of input functions, and J and G are the system and input matrices, respectively.

To close the loop substitute equation (21) into equation (20) to yield a closed-loop discrete-time linear system, viz.

$$x_{k+2}=-A_1 x_{k+1}-A_0 x_k+T(B_1 Jx_{k+1}+B_0 Jx_k)+T(B_1 Gu_{k+1}+B_0 Gu_k) \quad . \tag{22}$$

Notice that this system is 2n-dimensional and thus has 2n eigenvalues. The n principal roots are those corresponding to the mappings of the eigenvalues of J. The remaining n roots are due to the fact that the first order differential equation (21) is being replaced by a second order difference equation (21). These roots are spurious roots and act as noise sources in the simulation. It is clear that the location of the 2n eigenvalues of the system of (22) determines the stability of the integration process. Taking the z-transform of equation (22) yields:

$$[z^2 I+(A_1-TB_1 J)z+(A_0-TB_0 J)]X(z)=[B_1 z+B_0]TGU(z) \quad . \tag{23}$$

The transfer function matrix (TFM) $H(z)$ of the system of (23) is:

$$H(z)=[z^2I+(A_1-TB_1J)z+(A_0-TB_0J)]^{-1}[B_1z+B_0]TG \ . \qquad (24)$$

Recall that the regression coefficients (A_i, B_i : i=0,1) are square n×n matrices. Therefore, there are $4n^2$ unknowns to be determined which place the 2n eigenvalues of equation (24). It is clear that the poles of the resulting closed-loop system can be arbitrarily placed in an infinite number of locations. The problem is how to determine the regression coefficients A_i and B_i : i=0,1 so that the system transient response is maintained while still having equation (20) perform as an integrator.

To solve the first problem, place the n principal roots at the exact z-plane mapping of the eigenvalues of J, that is, the usual discrete-time system matrix e^{JT}. To guarantee the accuracy of the method, place the n spurious roots as close to the origin in the z-plane as possible [4]-[5]. This pole assignment requires that A_i and B_i : i=0,1 satisfy the following matrix equations:

$$A_1-TB_1J=-e^{JT} \ . \qquad (25a)$$

$$A_0-TB_0J=0, \text{ where 0 is the n×n null matrix.} \qquad (25b)$$

Note that there are $2n^2$ equations and $4n^2$ unknowns. Therefore, more constraints are required. The remaining $2n^2$ constraints are obtained by forcing equation (20) to act as an integrator. Recall that for a linear multistep method, and thus MSRP2, to be convergent it is necessary and sufficient that the method be consistent and zero-stable. Equivalently, the steady state gain of the discrete-time system should be that of the continuous-time system, and an open-loop integrator pole should be at Z=+1, respectively. Algebraically, these constraints can be written, again respectively, as (see [4]):

$$A_1-B_0-B_1=-2I, \text{ where I is the n×n identity matrix.} \qquad (26a)$$

$$I+A_0+A_1=0 \ . \qquad (26b)$$

It is shown in [5] that these constraints imply that the integrator is first order accurate.

Equations (25a)-(26b) provide $4n^2$ equations to be solved for the $4n^2$ unknowns of the A_i's and B_i's. Provided that J is invertible (for the case when J has zero eigenvalues refer to [6]), this set of equations can be solved using linear algebra techniques as follows:

$$\begin{bmatrix} I & : & I \\ \cdots\cdots & : & \cdots\cdots \\ -(JT)^{-1} & : & I-(JT)^{-1} \end{bmatrix} \begin{bmatrix} A_0 \\ \cdots \\ A_1 \end{bmatrix} = \begin{bmatrix} -I \\ \cdots\cdots\cdots \\ e^{JT}(JT)^{-1}-2I \end{bmatrix} \ . \qquad (27a)$$

Postmultiplying the first row of this equation by $(JT)^{-1}$ and adding the result to the second row yields:

$$\begin{bmatrix} \vdots \\ I : I \\ \vdots \\ 0 : I \\ \vdots \end{bmatrix} \begin{bmatrix} A_0 \\ \vdots \\ A_1 \end{bmatrix} = \begin{bmatrix} -I \\ \cdots\cdots\cdots\cdots\cdots \\ e^{JT}(JT)^{-1} - 2I - (JT)^{-1} \end{bmatrix} . \tag{27b}$$

From this equation it is seen that A_1 and A_2 are given as:

$$A_1 = e^{JT}(JT)^{-1} - (JT)^{-1} - 2I . \tag{28a}$$

$$A_0 = -e^{JT}(JT)^{-1} + (JT)^{-1} + I . \tag{28b}$$

While the B_i's follow directly from the A_i's as:

$$B_1 = [A_1 + e^{JT}](JT)^{-1} . \tag{29c}$$

$$B_0 = A_0 (JT)^{-1} . \tag{29d}$$

The above expressions for the A_i's and B_i's can be easily computed using a software package such as PC-Matlab [7] by inputting T and J.

Using equations (25a)-(26b) in conjunction with the regression coefficients, a state space representation for H(z) can be written as:

$$\begin{aligned} x(k+1) &= \begin{bmatrix} \vdots & \\ 0 : & 0 \\ \cdots & \cdots \\ I : & e^{JT} \\ \vdots & \end{bmatrix} x(k) + \begin{bmatrix} TB_0 G \\ \cdots \\ TB_1 G \end{bmatrix} U(k) \quad \Rightarrow \quad x(k+1) = Ax(k) + Bu(k) \\ y(k) &= \begin{bmatrix} \vdots & \\ 0 : & I \\ \vdots & \end{bmatrix} x(k) = x_2(k) = x_k \quad \Rightarrow \quad y(k) = Cx(k) = x_2(k) = x_k . \end{aligned} \tag{30}$$

Notice that this is the well known observable canonical form. Thus, using MSRP-2 it is always possible to recover the state of the system being integrated as this representation is guaranteed to be observable.

The integration method just presented can also be implemented using recursive identification of J and G to adaptively determine the regression coefficients. However, this variant of the method will not be considered here as it is believed that due to the nature of the SSME simulation this method will not provide any significant speedup. The main reason being the computational burden associated with the identification and updating required per time step. Moreover, at present there are no numerically reliable techniques for updating the matrix exponential on-line.

H. COMPUTATIONAL ASPECTS OF LINEAR MULTISTEP METHODS

In this section some computational aspects of linear multistep methods are considered. It is important to realize, however, that

while Euler's method, AB-2, and MAB-2 are localized integrators, in MSRP-2 the integrators are no longer localized, that is, one per system state. As a consequence, the computational burden associated with matrix integration increases rapidly with the number of states (approximately 4n-squared multiplies in addition to the derivative function evaluation). Therefore, it is necessary that techniques for reducing the number of computations be considered. Also, a detailed analysis of the computational burden associated with MSRP integrators is in order. These are the major topics of this section.

Before proceeding further it should be mentioned that comparisons among integrators and numerical details of the different algorithms being used will be done using the number of floating point operations (flops) of each algorithm. A flop is approximately the amount of work involved in a floating point add, a floating point multiply, and the required subscripting [8]. In mathematical terms, this is equivalent to the amount of work associated with the following statement:

$$s:=s+a_{ik}b_{kj} \quad . \tag{31}$$

which in terms of the well-known Fortran computer language can be written as:

```
S=S+A(I,K)*B(K,J) .
```
(32)

H.1. COMPUTATIONAL COST ASSOCIATED WITH AB-1, AB-2, AND MSRP-2

Regarding the real-time simulation methods considered here, the bulk of the computations results from products of the type Ax, where A can be either an n×n or an n×m matrix and x is either an n-dimensional or an m-dimensional vector, accordingly. It is not hard to see that when A is an n×n matrix, the product Ax requires n^2 flops. On the other hand, when A is n×m this product can be performed in nm flops. The amount of work associated with multiplying a given vector by a scalar quantity requires approximately n flops. At this time, it is important to point out that these flop counts are simply rough approximations that are used by computer theorists in an effort to acknowledge the countless operations that take place during program execution (paging, subscripting, etc.). The approximate number of flops required by AB-1, AB-2 (MAB-2), and MSRP-2 for every timestep is given in Table 2.

It is clear from this table that while Euler's method and AB-2 (MAB-2) cost about the same, MSRP-2 requires approximately 5 times more computations than either one of them. Therefore, to break even, the timestep for MSRP-2 must be at least 5 times that of either Euler's method or AB-2 (MAB-2). Letting the timestep for either one of the latter methods be the normal timestep, the speedup factor is:

$$\text{speedup} \cong \frac{T_{MSRP-2}}{5T_{NORMAL}} \quad . \tag{33}$$

Therefore, MSRP-2 can be made more numerically efficient than either Euler's method or AB-2 (MAB-2) provided that the timestep for MSRP-2 is chosen appropriately. This is clearly illustrated by the results obtained in the simulation of the valve dynamics module using these integration methods (more about this later).

Table 2. Computational cost associated with AB-1, AB-2, and MSRP-2

Required Operations (real-time)	Number of Flops	Total Number of Flop[s
Linear test equation		
$\dot{x}=Jx+Gu$	n^2+nm	n^2+nm
Euler's (AB-1)		
$x_{k+1}=x_k+T\dot{x}_k$	n	n^2+n+m
AB-2 (MAB-2)		
$x_{k+2}=x_{k+1}+T(1.5\dot{x}_{k+1}-0.5\dot{x}_k)$	2n	$n^2+2n+nm$
MSRP-2		
$x_{k+2}=-A_1 x_{k+1}-A_0 x_k+T(B_1\dot{x}_{k+1}+B_0\dot{x}_k)$	$4n^2$	$5n^2+nm$

It should be pointed out that the comparison established in Table 2 above is based on the assumption that a linear system is being integrated. For nonlinear systems the computational burden is usually much larger. Thus, MSRP-2 is expected to provide better results with a timestep less than five times the timestep of either Euler's method or AB-2 (MAB-2).

H.2. IMPLEMENTATION OF MSRP-2

In Table I above the approximate flop count for MSRP2 is given assuming that the integration is performed using the original system. However, in an effort to reduce the number of computations in real-time, it is always possible to first transform the coordinates of the original system. This transformation can be done off-line and the results stored. Then the integration process can be carried out on the resulting system. Of special interest here are transformations which yield diagonal or Jordan, Schur, and Hessenberg forms of the original system matrix J. The diagonal form gives essentially the classical linear multistep method, that is, one integrator per system state. It is important to keep in mind that, although this may seem

appropriate, computing the diagonal form of a matrix is, in general, not a numerically reliable process. This is especially the case when the matrix being diagonalized has repeated eigenvalues [8]. In contrast, The well-known Schur and Hessenberg decompositions of a matrix are easily obtained via orthogonal transformations. Since orthogonal matrices are perfectly conditioned, these decompositions are considered to be very stable and numerically robust [8]. Furthermore, both Schur and Hessenberg decompositions of a matrix result in quasi-triangular forms. Thus, it is possible to reduce the number of on-line computations during the integration process by using these matrix decompositions. These approaches are considered next in what follows.

The first approach consists of transforming the initial system to either diagonal or Jordan co-ordinates, then performing the integration process. Recall that, in general a diagonal form results when the system poles are all distinct, while a Jordan form results when the system has multiple poles. As the regression coefficients of MSRP-2 are a function of $(JT)^{-1}$ and e^{JT}, and as these functions of J have a triangular structure whenever J is in Jordan form, and finally, as the Schur and Hessenberg forms of J are also triangular matrices, the case when J is in Jordan form is considered a part of the approach in which the A_i's and B_i's are triangular matrices.

The foregoing indicates that localized integrators can only be obtained provided J is diagonalizable. Recall that diagonalizing a matrix involves determining its eigenstructure. Further, recall that J is, in general, an unsymmetric matrix. In the usual situation, the standard procedure to diagonalize an unsymmetric matrix involves three steps. First, the matrix is reduced to upper Hessenberg form using Householder transformations. Then the Q-R algorithm is used to produce the upper real Schur form of the matrix resulting from step one to yield $H=Q'JQ$ (Q' is the transpose of Q). These two steps require about $15n^3$ flops [8] (this includes the computation of both Q and H). Finally, to obtain the diagonal form of J, a block diagonalization method requiring approximately n^3 extra flops is used. Therefore, the diagonalization of J can be accomplished in about $16n^3$ flops.

Having determined the eigenstructure of J, a transformation matrix whose columns are the eigenvectors of J can be formed and its inverse computed off-line and stored. The latter computation is usually done via the singular value decomposition method, thus requiring about $7n^3$ flops [8]-[9]. Then this transformation matrix is used to transform the system of (21) into diagonal coordinates. This process is given in what follows, assuming that the original system is that of equation (21); that is,

$$\dot{x}_k = Jx_k + Gu_k \quad . \tag{34}$$

First, let P be the matrix whose columns are the eigenvectors of J. Then it follows that $P^{-1}JP=\Lambda$, where Λ is a diagonal matrix with the eigenvalues of J along its main diagonal. Next, use P as a coordinate transformation matrix to obtain localized integrators, viz.

$$x_k=Pz_k \Rightarrow \begin{cases} z_k=P^{-1}x_k \\ \dot{z}_k=P^{-1}\dot{x}_k \end{cases} \qquad (35)$$

At this point it only remains to carry out the integration process, using one integrator per system state, as follows:

$$z_{i\,k+2}=-a_{i\,1}z_{i\,k+1}-a_{i\,0}z_{i\,k}+T[b_{i\,1}\dot{z}_{i\,k+1}+b_{i\,0}\dot{z}_{i\,k}] \qquad (36)$$

for $i=1,2,3,\ldots,n$.

As a last step, use P to determine the state vector, x, in the original set of coordinates by transforming the state vector z obtained from the integration process, viz.

$$x_{k+2}=Pz_{k+2} \qquad (37)$$

This completes the first approach of diagonalization and integration. It is worth mentioning at this point that determining the a_i's and b_i's requires approximately 4n flops. However, these computations are done prior to the actual run and the results are stored. Therefore, the total number of off-line computations is $23n^3+4n$ flops. Also, notice that, when J is diagonal, the total number of flops required per timestep is $4n^2+(4+m)n$. Of this total, the function evaluations of equation (34) constitute the bulk of the computations, that is, n^2+nm flops. The coordinate transformations of equation (35) require n^2 flops each for z and its derivative. Finally, the computations of equation (36) represent a total of 4n flops, while the transformation of equation (37) requires n^2 extra flops.

The second approach, or triangular approach, consists on reducing the original system matrix J to an upper quasi-triangular form. There are several methods to do this. One of these methods involves using a sequence of Householder transformations to reduce the system matrix J to its upper Hessenberg form. Equivalently, determine U and H such that H=U'JU, where U (orthogonal) is a product of Householder matrices and H is upper Hessenberg, that is, $h_{i\,j}=0$ whenever $i>j+1$. This process requires $(7/3)n^3$ flops [8]. A second method is to compute the Schur decomposition of J, that is, determine an orthogonal matrix U such that H=U'JU, where each $H_{i\,i}$ is either a scalar or a 2×2 matrix having complex conjugate eigenvalues. This decomposition can be done using the Q-R algorithm in approximately $15n^3$ flops [8]. At this point it is important to mention that these two processes are very numerically stable.

The third and last approach considers the Jordan reduction of J. As mentioned before, the Jordan form of a matrix can be obtained in approximately $16n^3$ flops. However, it should be emphasized that this reduction is, in general, ill-conditioned. A common aspect among these three methods is that they all yield upper quasi-triangular integrator coefficients. Thus, the total number of on-line computations

decreases as compared to the number of computations required for the unreduced system. However, there might be an increase in the total number of off-line computations. This is seen by considering the amount of work involved in determining the A_i's and B_i's. A detailed description of this process is given next as follows.

As just mentioned, the latter three methods being considered here for reducing the number of computations of the integration process all result in upper quasi-triangular matrices. Therefore, without loss of generality, the number of flops required to compute each integrator coefficient is given next assuming that these coefficients are strictly upper triangular matrices.

For simplicity, the expressions for the A_i's and B_i's for MSRP-2 are repeated here as a function of H the corresponding form of J, viz.

$$A_1 = e^{HT}(HT)^{-1} - (HT)^{-1} - 2I$$

$$A_0 = -e^{HT}(HT)^{-1} + (HT)^{-1} + I$$

$$B_1 = e^{HT}(HT)^{-1} + A_1(HT)^{-1} \qquad (38)$$

$$B_0 = A_0(HT)^{-1} \ .$$

1. **Computation of e^{HT}:** The matrix exponential is usually computed using Pade approximations. Although the number of flops for this algorithm is a function of $||H||_\infty$, a typical number is somewhere between $8n^3$ and $10n^3$ flops. However, since H is upper triangular, only about $6n^3$ flops are required [8].

2. **Computation of $(HT)^{-1}$:** This matrix inversion can be easily done via the singular value decomposition of H. From [9], this algorithm takes about $5n^3$ flops for an upper triangular matrix.

3. **Computation of matrix products:** All the required matrix products involve only upper triangular matrices. Therefore, these products require only $(1/6)n^3$ flops each [8].

From the foregoing it is seen that once that e^{HT} and $(HT)^{-1}$ have been determined and the results stored, the integrator coefficients can be computed in approximately $(1/2)n^3$ flops. Notice that in this process the product $e^{HT}(HT)^{-1}$ is computed only once and then stored.

The approximate flop counts just given are only for those operations which can be done off-line. Therefore, it still remains to consider the number of on-line computations required per timestep. These computations include the function evaluations of equation (34), the state vector coordinate transformations of equations (35)-(36), and the integration process of equation (20) with the coefficients replaced by the matrices of equation (38). As in the case when J is diagonalizable, the function evaluations and coordinate transformations require $4n^2+nm$ flops. Each of the products in the integration process requires $(1/6)n^2$ flops, for a total of $(2/3)n^2$ flops. Therefore, when the integrator coefficients are upper

triangular, the total number of on-line computations can be approximated to $(14/3)n^2+nm$ flops. Notice that this flop count compares very favorably with that of the case when J is diagonalized. At the same time, however, this flop count is very close to the one obtained when the original system is used in the integration. That is, $5n^2+nm$ flops. For ease of comparison, the results just given in the paragraphs above are compiled in Tables 3 and 4 below. The quantities inside the round brackets in these tables indicate the number of times a particular operation is done.

Table 3. Off-line computational cost associated with MSRP-2

Required Operations	System Matrix J (Computational Cost in Flops)				
	Original	Diagonal	Jordan	Hessen.	Schur
Coordinate X-formation	0	$16n^3$	$16n^3$	$7n^3/3$	$15n^3$
Matrix Inversion	$7n^3$	$7n^3$	(2) $12n^3$	$5n^3$	$5n^3$
Matrix Exponential	$10n^3$	0	$6n^3$	$6n^3$	$6n^3$
Matrix Product	$3n^3$	$4n$	$1n^3/2$	$1n^3/2$	$1n^3/2$
Off-line Computation	$20n^3$	$23n^3+4n$	$69n^3/2$	$83n^3/6$	$53n^3/2$

Table 4. On-line computational cost associated with MSRP-2

Required Operations	System Matrix J (Computational Cost in Flops)				
	Original	Diagonal	Jordan	Hessen.	Schur
Function Evaluations	$nm+n^2$	$nm+n^2$	$nm+n^2$	$nm+n^2$	$nm+n^2$
Coordinate X-formation	0	(3) $3n^2$	(3) $3n^2$	(3) $3n^2$	(3) $3n^2$
Integration Process	$4n^2$	$4n$	$2n^2/3$	$2n^2/3$	$2n^2/3$
On-line Computation	$nm+5n^2$	$nm+4n+4n^2$	$nm+14n^2/3$	$nm+14n^2/3$	$nm+14n^2/3$

From these tables it is clear that transforming the original system to upper Hessenberg form prior to performing the integration process requires the least number of off-line computations, while the number of on-line computations compares well with that of the case when the original J matrix is used. This, coupled with the fact that the Hessenberg decomposition of a matrix can be obtained via orthogonal transformations, makes this approach highly desirable. Moreover, the excellent numerical properties of this decomposition guarantees the reliability of the computations.

Notice that although diagonalizing J requires the least on-line computational effort, it should be kept in mind that, in general, not all matrices are diagonalizable and that this process can be numerically ill-conditioned. However, matrix diagonalization is a common practice in actual applications.

Associated with specific hardware and software implementations there are several aspects of MSRP that should be emphasized. As MSRP-2 only involves adds and multiplies, and as most hardware and software packages support these two basic arithmetic operations, MSRP-2 allows real-time simulation on a wide variety of computer systems and software packages. Therefore, the usual constraints associated with the real-time simulation of physical systems are no longer encountered when using MSRP-2 integrators. It should be realized, however, that although MSRP-2 allows timesteps much larger than would be normally possible, the hardware which is attached to the simulator may restrict the stepsize. Hence, the only constraining factors in real-time simulation using MSRP-2 are those due to hardware constraints.

MSRP-2 can be implemented in three steps. The first step consists of obtaining the computation time for the given system. The second step involves choosing the desired timestep T. And the third step designing MSRP-2 by solving for the integrator coefficients using the appropriate formulas.

4. INTEGRATION METHODS IN THE SSME SIMULATION

In the SSME simulation numerical integration is done using Euler's method. This method is very attractive because of its simplicity and ease of implementation. However, care must be exercised in selecting the appropriate integration timestep. This is important since the stability and accuracy of the simulation are a direct function of the integration timestep. It is shown here that a faster and more accurate simulation of the SSME can be obtained by replacing Euler's integrators by a two step method. More specifically, the Valve Dynamics and the Oxidizer Turbopump modules are used as case studies to demonstrate that it is possible to speed up the SSME simulation without requiring additional hardware.

This chapter is organized as follows. First a brief description of the valve dynamics module is given. Then, the results obtained from simulating this module using Euler's method, and AB-2 and MSRP-2 integrators are presented. This is followed by a brief review of the oxidizer turbopump module along with some recommendations to provide a faster simulation.

A. VALVE DYNAMICS MODULE: CASE STUDY I

To throttle the SSME there are a total of five control valves. These valves are:

MFV - Main Fuel Valve
MOV - Main Oxidizer Valve
FPV - Fuel Preburner Valve
OPV - Oxidizer Preburner Valve, and
CCV - Coolant Control Valve

These five valves reside in the valve dynamics module of the SSME simulation. Figure 4 shows the block diagram for valve position control, and Figure 5 gives the fortran code for the main integration loop of the valve dynamics module. From these figures it is seen that position control of each valve is described by a 6th order system. Moreover, the valve position control system is linear within the integration timestep as the nonlinearities (stiction, backlash, and wind-up) introduced by valve linkage remain constant within the integration loop. Notice that multirate sampling is used as the outer timestep $DT=2\times10^{-4}$ is added to 10^{-5} and divided by $Delta=2\times10^{-5}$. Some simple arithmetic shows that the timestep being used within this module is ten times smaller than the external timestep. The main integration loop is run 5 times, one for each valve. Therefore, in actuality this module represents a 30th order system (6 states times 5 valves). Since each main integration loop is run 10 times, the simulation basically sees 300 integrations per outer timestep. It is clear that the computational demands of this particular module can considerably slow down the SSME simulation. In a situation such as this there are two possible solutions to the problem. One solution is to use additional hardware. The other solution is to provide integrators which would allow the use of larger timesteps and thereby reduce simulation time. The latter is the purpose of this study.

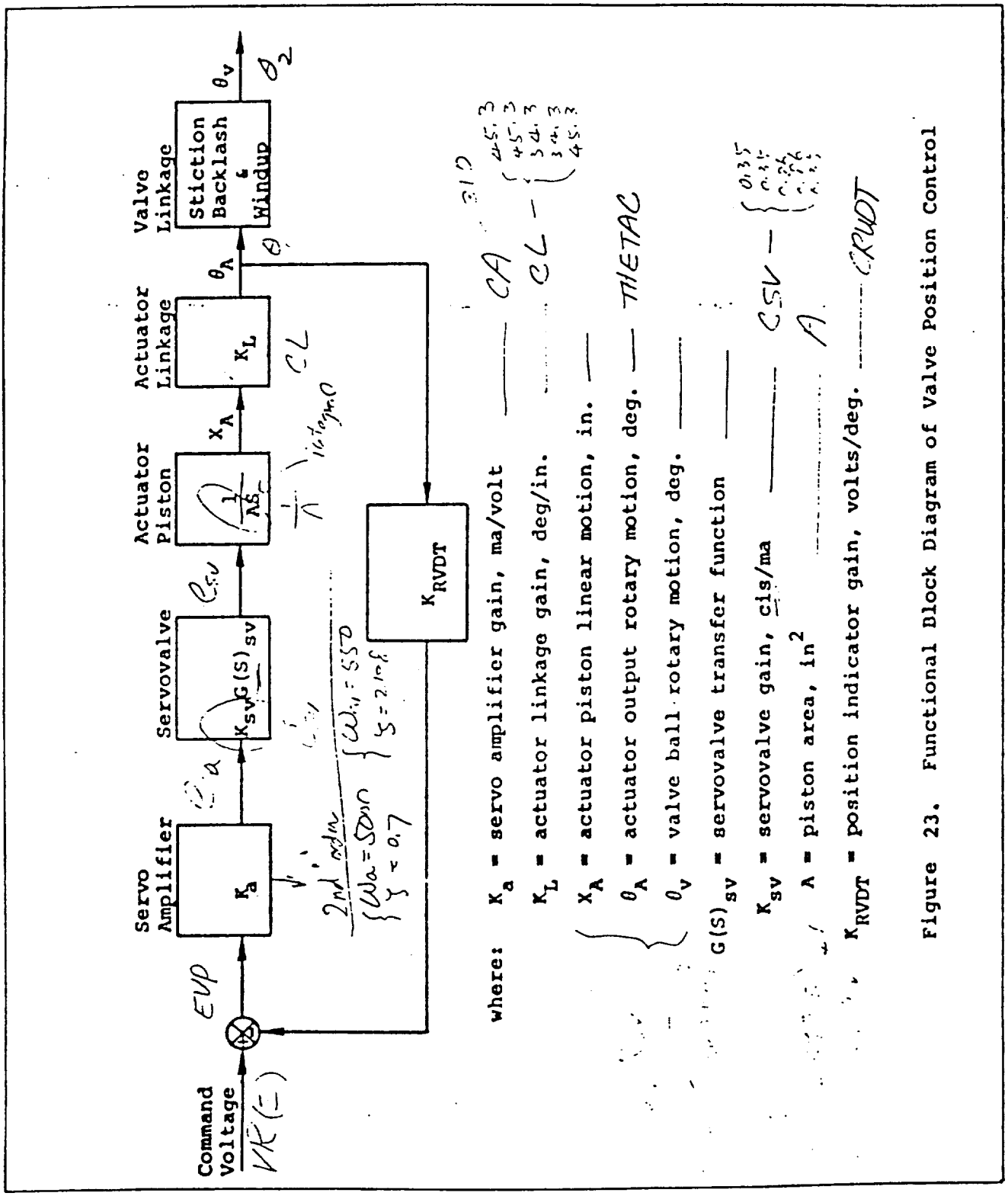


Figure 23. Functional Block Diagram of Valve Position Control

FORM R 131-H-16 REV. 2-78

Figure 4. Functional block diagram of valve position control

```
C                                                    DT = 2x10^-4
                                                     Delta = 2x10^-5  => loop = 10.5
      LOOP=(DT+0.00001)/DELTA
      DO 300 I=1,5
        IF(TPA.GT.0.0.AND.TIME.GE.TPA) GO TO 218
C
C                     MAIN INTEGRATION LOOP
C
      DO 210 J=1,LOOP
C
          EVP        = THETAC(I)*CRVDT-VR(I)
C
          DDESA      = CA*WA**2*EVP-WA**2*ESAC(I)-2.0*SA*WA*DESA(I)
          DDESV      = WSV**2*ESA(I)-WSV**2*ESV(I)-2.0*SSV*WSV*DESV(I)
          DTHETA(I) = CL(I)*CSV(I)/A(I)*ESV(I)
          DVR        = THETA(I)*CRVDT*CRS/TRS-VR(I)/TRS
C
C                        INTEGRATORS
C
          ESAC(I)  = ESAC(I)  + ALIM(DESA(I))    * DELTA
          ESA(I)   = ESAC(I)
          DESA(I)  = DESA(I)  + ALIM(DDESA)      * DELTA
          ESV(I)   = ESV(I)   + ALIM(DESV(I))    * DELTA
          DESV(I)  = DESV(I)  + ALIM(DDESV)      * DELTA
          THETA(I) = THETA(I) + ALIM(DTHETA(I)) * DELTA
          VR(I)    = VR(I)    + ALIM(DVR)        * DELTA
C
C            LIMITS ON INPUT AMPLIFIER ARE +-23 VOLTS.
C
          IF  (ESA(I).GT.23.)        ESA(I)=23.
          IF  (ESA(I).LT.-23.)       ESA(I)=-23.
          IF  (ABS(ESA(I)).LT.0.25) ESA(I)=0.0
C
C            LIMITS ON SERVO AMPLIFIER ARE +-20 VOLTS.
C
          IF  (ESV(I).GT.20.)        ESV(I)=20.
          IF  (ESV(I).LT.-20.)       ESV(I)=-20.
C
          THETA(I)=AMAX1(0.0, AMIN1(THETA(I),THETMAX(I)))
C
  210   CONTINUE
C
C                     END OF INTEGRATION LOOP
C
  218   CONTINUE
C
C   BACKLASH IS DEFINED AS THE AMOUNT OF ACTUATOR OUTPUT SHAFT TRAVEL
C   REQUIRED TO REVERSE DIRECTION OF VALVE BALL MOTION UNDER CONDITION
C   OF ZERO LINKAGE WINDUP AND TORQUE LOADING.  THE VALUES USED HERE I
C   HALF OF THE AMOUNT OF THE TOTAL TRAVELING, WELL HALF ON EACH SIDE.
C
        FAC=1.0
        IF(I.EQ.3.AND.(THETA(I).GT.33.8.AND.THETA(I).LT.84.0)) FAC=0.0
        IF(I.EQ.4.AND.(THETA(I).GT.38.3.AND.THETA(I).LT.75.8)) FAC=0.0
        IF(ABS(THETA(I)-THETA1(I)).GE.(THETBL(I)*FAC)) GO TO 212
```

Figure 5. Valve dynamics main integration loop fortran code

An integration analysis of the valve dynamics module requires that a mathematical model describing the valve position control system be determined. It turns out that a state space model of this system can be obtained directly from the main integration loop of Figure 5, viz.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -w_a^2 & -2s_a w_a & 0 & 0 & 0 & -c_a w_a^2 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ w_{sv}^2 & 0 & -w_{sv}^2 & -2s_{sv} w_{sv} & 0 & 0 \\ 0 & 0 & \frac{[C_1 C_{sv}](I)}{A(I)} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{C_{rvdt} C_{rs}}{T_{rs}} & \frac{-1}{T_{rs}} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} 0 \\ c_a w_a^2 c_{rvdt} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \theta_c(I)$$

Using the notation of Figure 5, the states x_1 through x_6 correspond to the actual states as: x_1=ESAC, x_2=DESA, x_3=ESV, x_4=DESV, x_5=THETA, and x_6=VR.

Although there are five different valves, it was found that three of them are described by the same set of data while the other two are described by another set of data. As a result, only two different systems were considered. However, it turned out that both these systems have the same set of eigenvalues. These eigenvalues are:

$$\lambda_1, \lambda_2 = -3499.7 \pm j3570.4$$

$$\lambda_3, \lambda_4 = -48.9 \pm j136.5$$

$$\lambda_5 = -2184$$

$$\lambda_6 = 1058.5$$

Clearly, this system is very stiff. Therefore, to obtain a stable and accurate simulation a very small timestep must be used. Notice as well that the complex poles have very large imaginary parts. This can make the selection of a suitable integration technique a difficult task.

Recall that for a simulation to be stable it is required that the λT-product be inside the stability region of the integrator being used. In addition, the simulation will be accurate if the λT-product is inside the region where there is minimum distortion. Therefore, for Euler's method the largest timestep that could be used to obtain an accurate and stable simulation of the valve dynamics module is $T=2\times10^{-5}$. A larger timestep would not guarantee accurate results. For a timestep $T=3\times10^{-4}$ the integrator would be operating on the verge of instability, with the slightest roundoff error making the simulation unstable. This timestep would also give very inaccurate results. This

is clearly seen in Figures 6 through 9, which show the results of simulating the system under consideration using Euler's method with different timesteps. The exact solution was obtained using Euler's method with a timestep $T=10^{-5}$. Notice that when a timestep $T=2\times10^{-4}$ is used, the simulation is not very accurate, with the initial transient being almost lost. Therefore, if the transient is important in the simulation, Euler's method should not be used with a timestep larger than $T=2\times10^{-5}$. However, should either an AB-2 or an MSRP-2 integrator be used to simulate this system, timesteps of $T=10^{-4}$ and $T=10^{-2}$ could be used, respectively. Although these timesteps would not approximate the transient too well, the results will be more accurate. Figure 10 shows the corresponding allowable timesteps for AB-2. From Figure 11 it is seen that if AB-2 is used with a timestep of 2×10^{-4} the simulation will be unstable. However, for a timestep $T=10^{-4}$ AB-2 performs quite well, this can be seen in Figures 12 through 14. Thereby given an improvement of a factor of 5. Recall that, for a linear system, MSRP-2 requires five times more computations than either AB-2 or AB-1 and thus its timestep must be five times as large. It turns out that MSRP-2 gives very accurate and stable results even for timesteps as large as $T=10^{-2}$. Figures 15 through 17 give the response using MSRP-2 with $T=2\times10^{-3}$ while Figures 18 through 20 give the results of using MSRP-2 with $T=10^{-2}$. Clearly, MSRP-2 offers the best simulation results with the largest possible timestep for this particular module.

Notice that this module has some of the largest eigenvalues present in the SSME simulation. In addition, their magnitudes differ by several orders of magnitude. Thus, this system is very stiff. It is well known that the real-time simulation of stiff systems poses some of the most challenging problems to the control engineer. Hence, this module could be thought of as typifying some of the problems associated with the SSME simulation.

Although MSRP-2 integrators provide excellent results with a substantial speed improvement, it is only recommended that they be implemented locally within each module. The nature of the SSME simulation coupled with the wide range of operating points do not make this problem amenable to an easy implementation using this type of integrators. However, if AB-2 integrators were used in place of Euler's method, a reasonable speedup could be expected. This would involve only a fraction of the work and complexity should MSRP-2 integrators be used.

Recall that to design an MSRP-2 integrator for a nonlinear system requires that the system be linearized about some operating point. Thus, to replace the integrators of the SSME simulation by MSRP-2 integrators, it would be necessary to linearize the different modules about the operating points of interest. This would involve a tremendous effort as the SSME simulation is highly nonlinear. Although this approach was considered, the time constraints associated with this project did not allow its realization.

A further point worth mentioning is the fact that the eigenvalues of this system are complex with very large imaginary parts. Recall

that this type of eigenvalues can cause instability problems when Euler's method is used. This is due to Euler's stability region being asymptotic to only the origin of the ÑT-plane. Thus, in order to stably and accurately simulate systems with this type of eigenvalues very small timesteps are required. Since the stability region of AB-2 is asymptotic to a larger portion of the imaginary axis of the ÑT-plane, it is expected that AB-2 would perform better in this type of situations. However, if the magnitude of the imaginary parts of the eigenvalues is much larger than their real parts, MAB-2 may be used. That is, the instability problems that may arise due to almost purely complex eigenvalues can be alleviated by replacing the classical AB-2 method by the modified AB-2 method. Keep in mind, however, that the simulation results may not be as accurate as MAB-2 is only first order accurate.

Finally, the concept of stability region in the simulation of nonlinear systems must be used with care. This is due to the fact that for nonlinear systems the margin of the stable region cannot be clearly distinguished from the unstable region. Essentially, there will be an inner region in which the simulation will be stable, an outer region where the simulation will be unstable, and a "fuzzy" region in between these two regions.

B. OXIDIZER TURBOPUMP MODULE: CASE STUDY II

This module presents some of the most challenging integration problems in the SSME simulation. An analytical study of this module was carried out to determine the range of frequencies of the system. Although it is recognized that this is not the best approach to linearizing this type of systems, the intent was to obtain an approximate frequency range of the system's eigenvalues. It is felt that for integration purposes this approximate range suffices. This analytical study was done because the maps describing the nonlinear functions in this module make it very hard to determine an accurate model of the system. Determining an accurate model of the oxidizer turbopump module requires a considerable amount of time and effort; both of which are beyond the scope of this project. Therefore, the 37th order nonlinear model was linearized analytically using the Jacobian method. By considering the maximum and minimum values specified in the maps, a reasonable estimate of the eigenvalues of the actual system could be obtained. It was found that the stable eigenvalues of the linearized system vary from -20 to about -8000. This range agrees well with the actual range of frequencies specified in [10]. The few unstable eigenvalues are believed to be a result of our crude approach to linearization. The numerical values of the eigenvalues of the linearized system are listed in Table 5. Although, it was expected that some lightly damped modes were going to be present in this module, only two sets of complex eigenvalues with a damping ratio of about 0.2 were found. At this time the authors cannot provide any further insight into this last aspect as it was very difficult to determine whether the ducts were actually being modeled inside or outside the module. However, further analysis is being done to determine a linearized model of the entire SSME

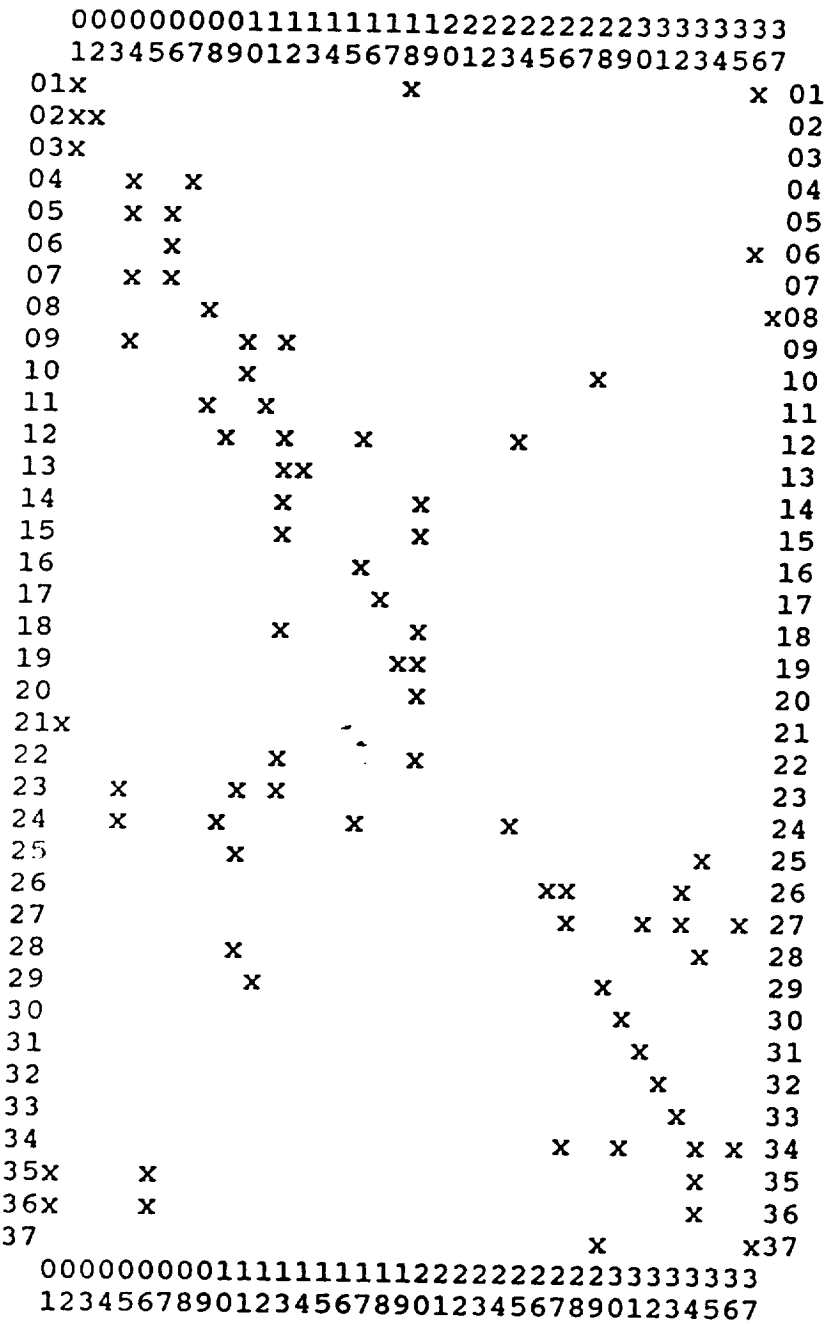
simulation by recording the change in the states due to perturbations introduced into each state. This analysis will provide linearized models and thus more accurate accounts of the eigenvalues of the system at different operating points. Another study being done is the determination of reduced order linear models of the SSME. This study will consider several techniques to obtain the reduced order models. As the transient response is of essence in the SSME simulation, singular perturbation cannot be used as a model reduction technique as it reduces the order by deleting the fast dynamics of the system.

Table 5. Approximate location of the eigenvalues of the oxidizer turbopump linearized model

i	$\lambda_i \times 10^3$
1→12	0
13→18	-0.020
19	-7.9452
20	-0.0116
21	-0.0050
22	-2.5104
23→24	0.0088 ± 0.0373i
25	-0.0393
26	-0.0237
27	-0.0113
28	-0.0344
29	0.0022
30	0.0013
31	-0.0025
32	-0.0023
33→34	-0.0002 ± 0.0010i
35	-0.0147
36	-0.0062
37	-0.0039

From Table 5 it is clear that the oxidizer turbopump module is a stiff system. Therefore, the discussion just given for the valve dynamics module also applies to this system. Notice that this implies that if Euler's method were to be replaced by AB-2, a speedup factor of approximately 5 could be realized. However, if MSRP-2 were used the time savings will be much larger. Since this module and the Fuel.F module are very similar in nature, it is expected that the eigenvalues of Fuel.F be within the same range. These two modules comprise most of the states in the SSME simulation.

The complexity of the linearized system can be observed from the following pictorial representation of the system A matrix, viz.



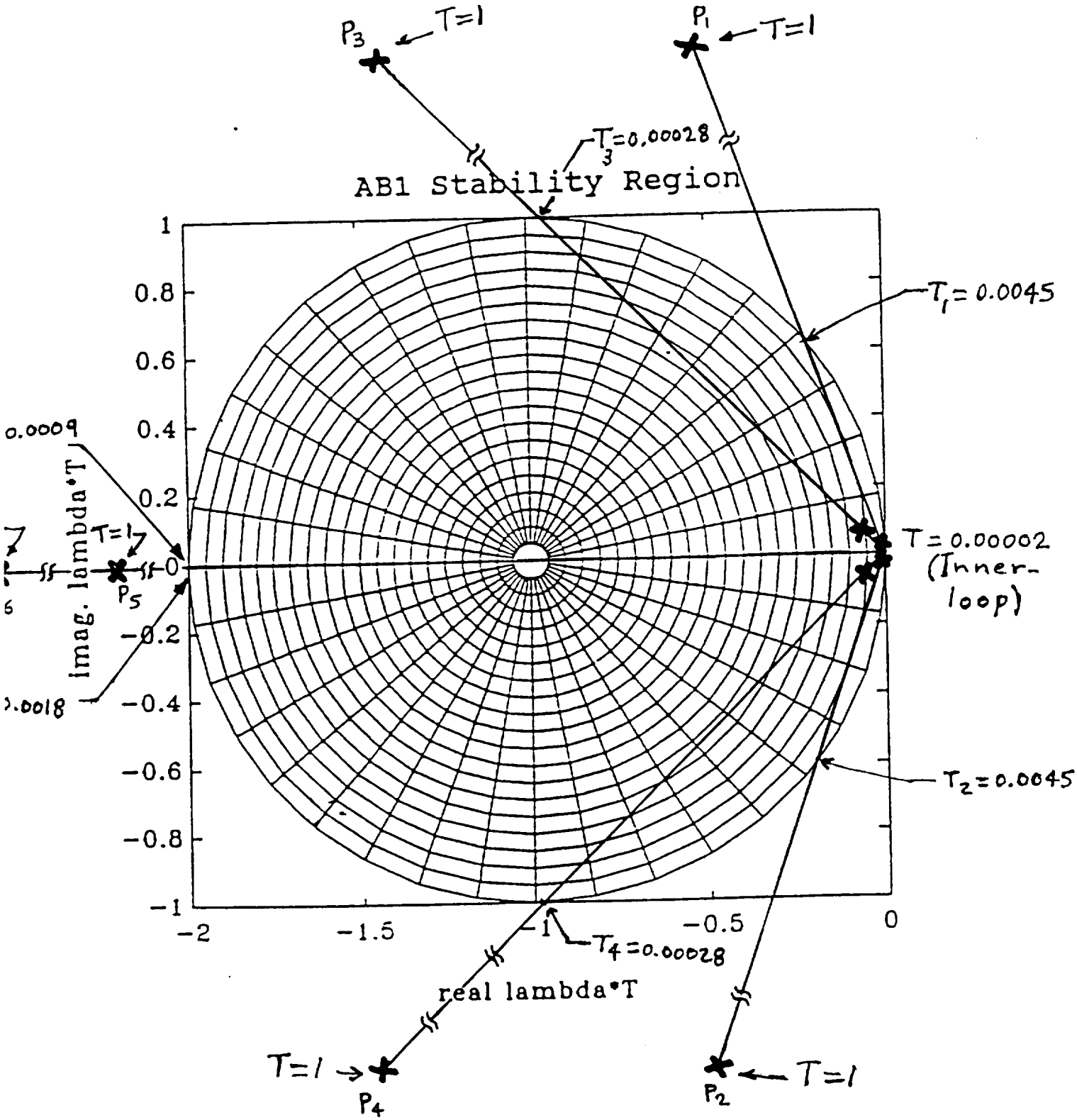
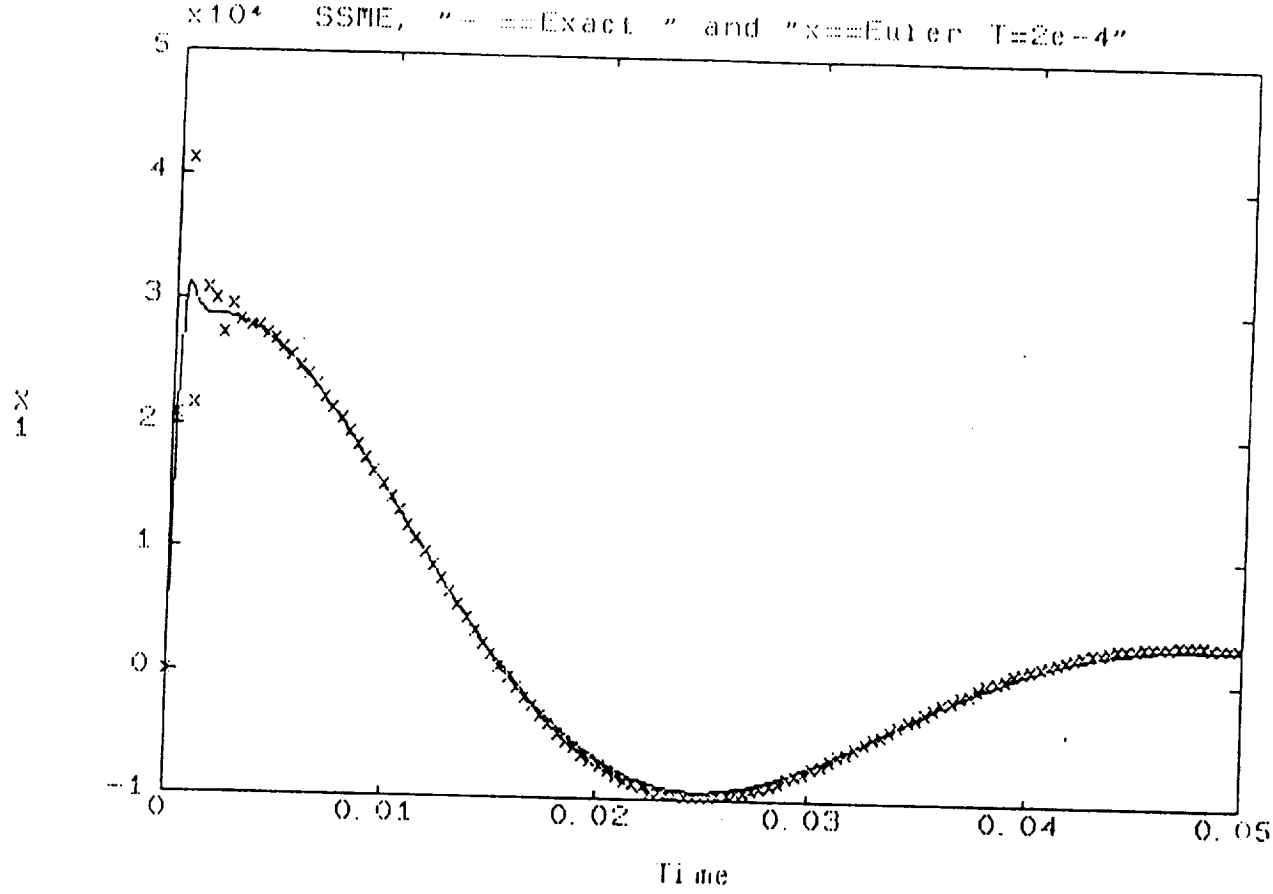


Figure 6. Simulation of the valve dynamics module using Euler's



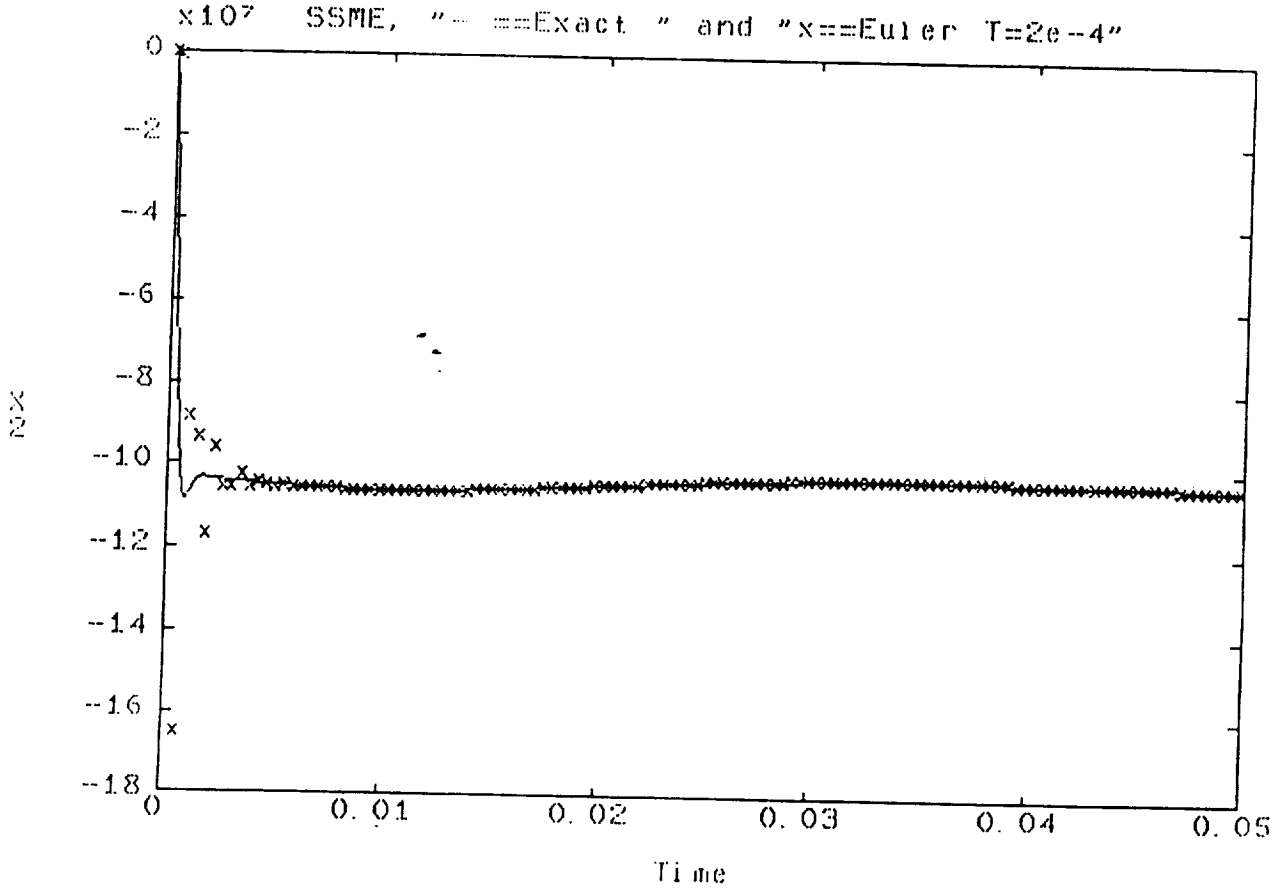
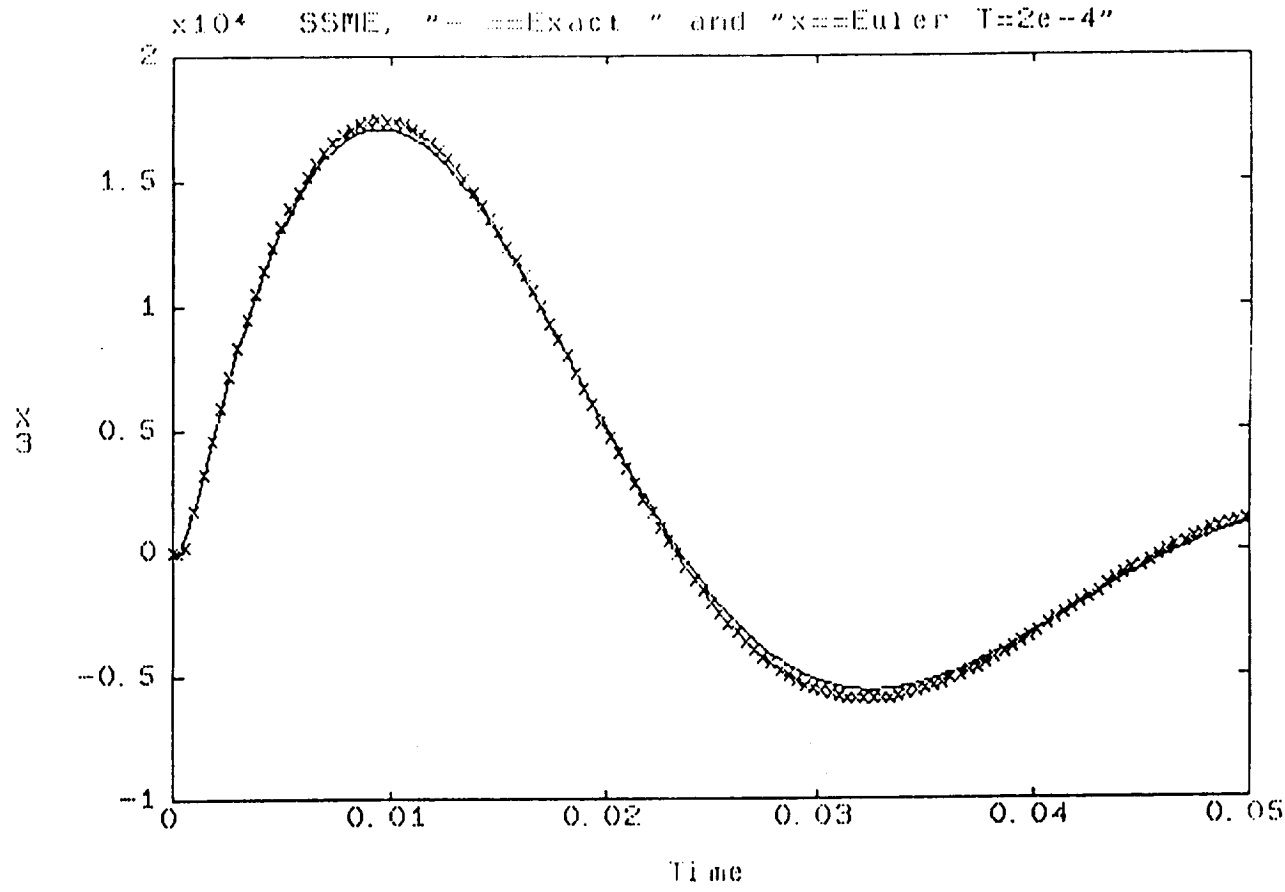


Figure 7. Response of x_1 and x_2 of valve dynamics using Euler's



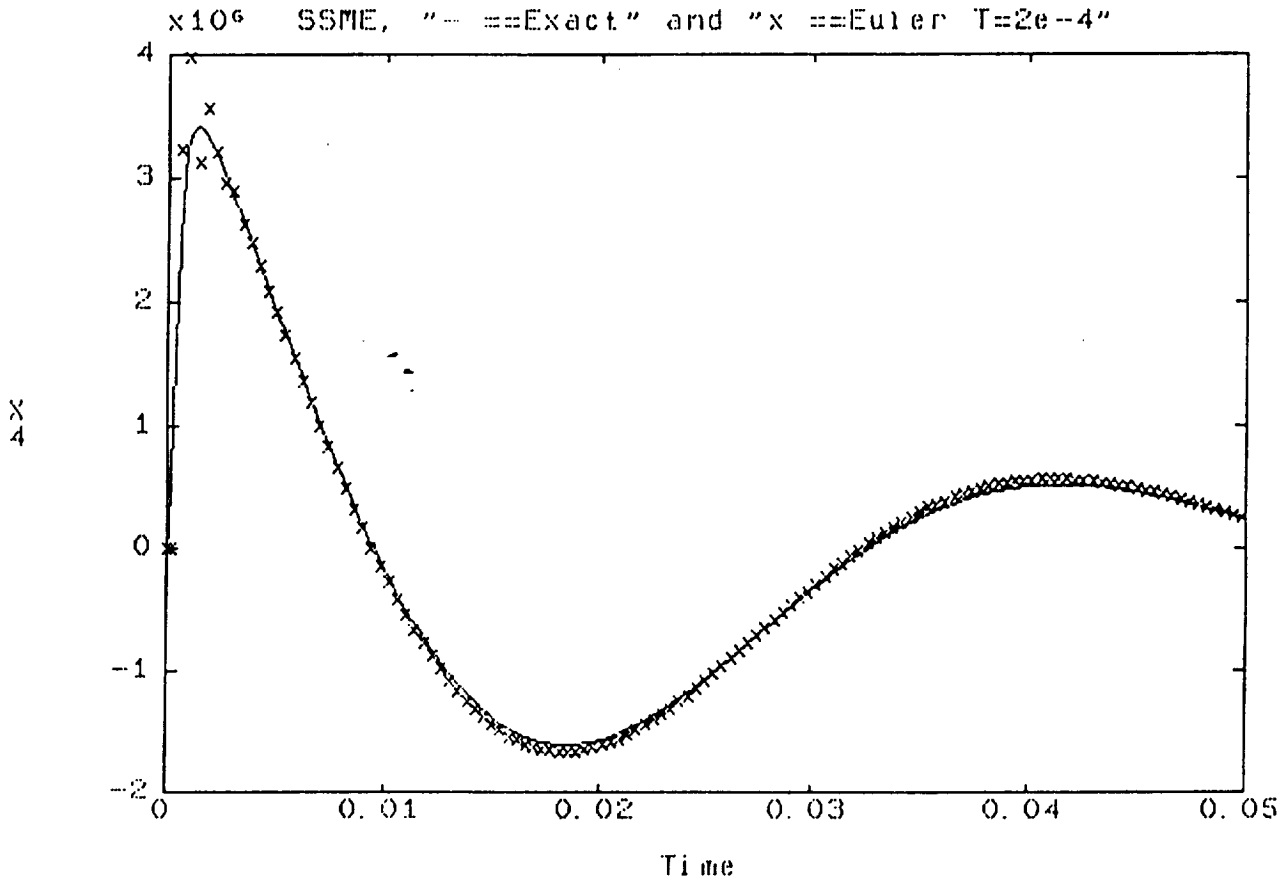


Figure 8. Response of x_3 and x_4 of valve dynamics using Euler's

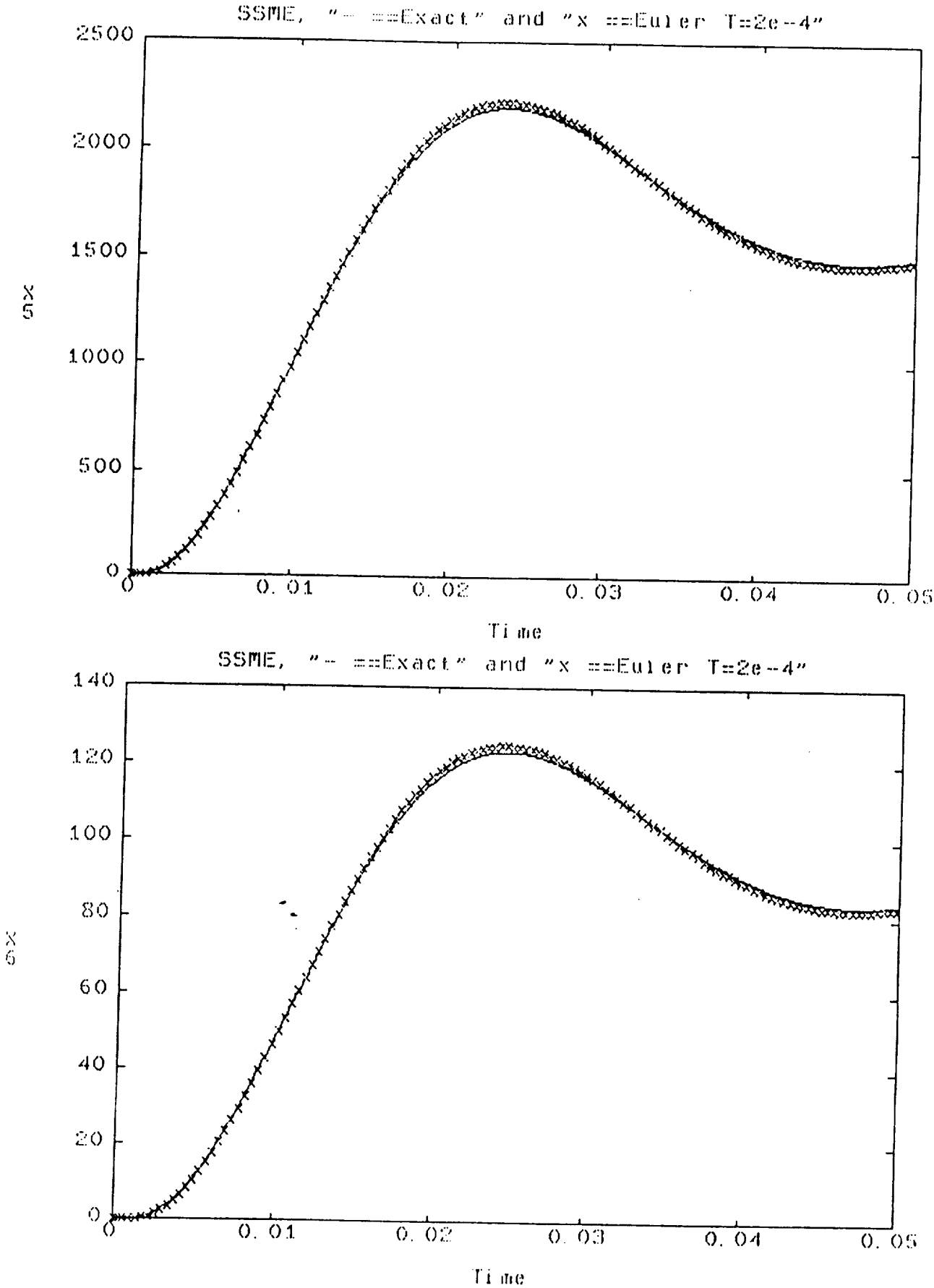


Figure 9. Response of x_5 and x_6 of valve dynamics using Euler's

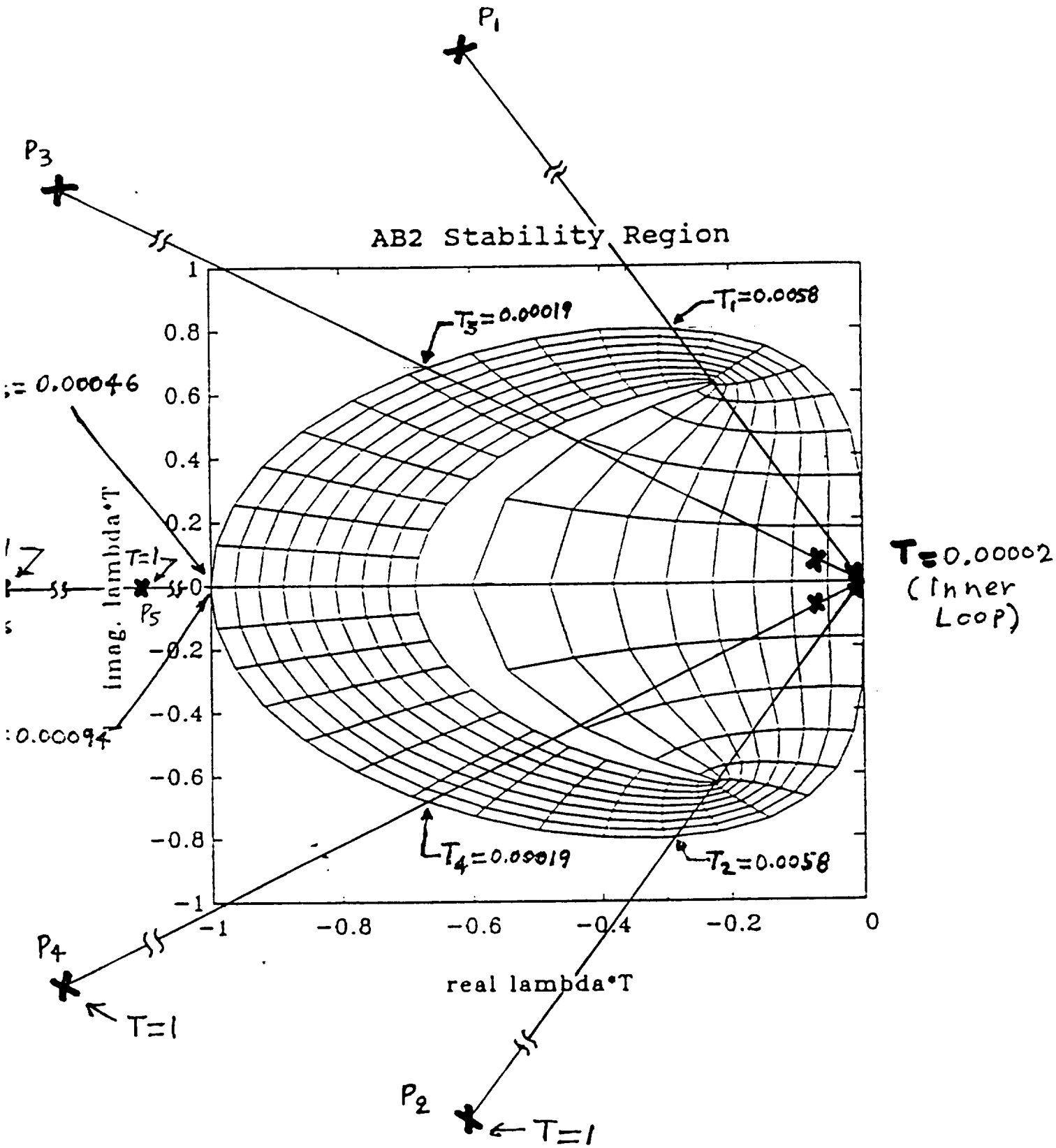


Figure 10. Simulation of valve dynamics module using AB-2

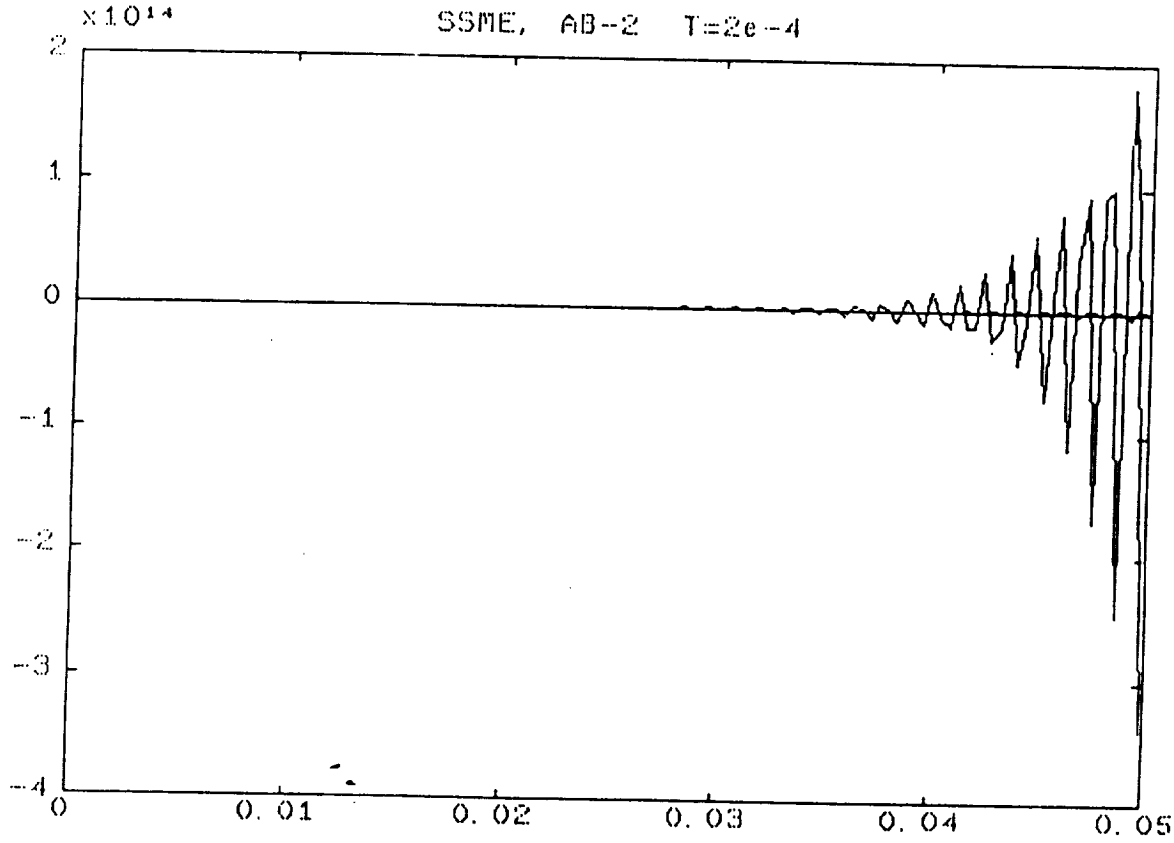
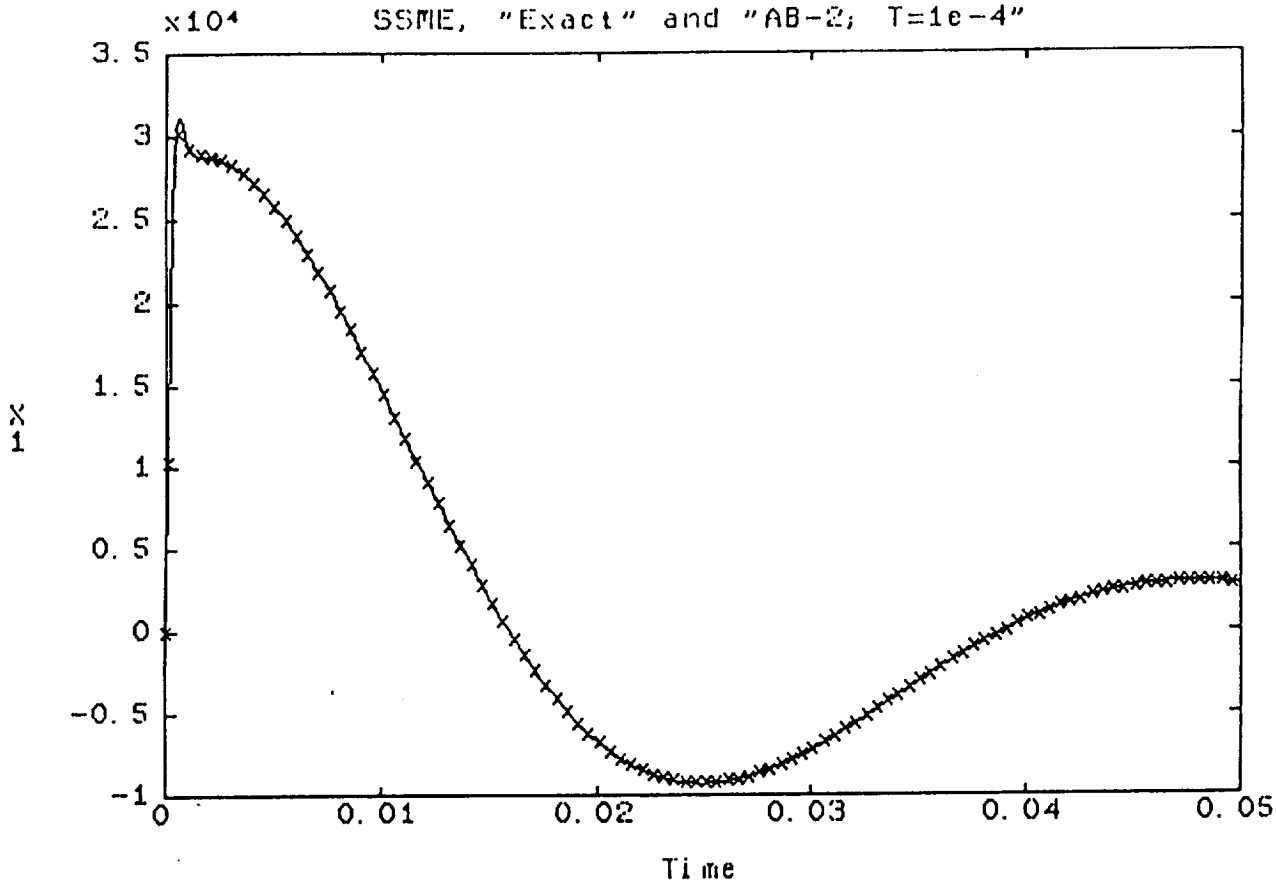


Figure 11. Simulation of valve dynamics module using AB-2 (unstable)



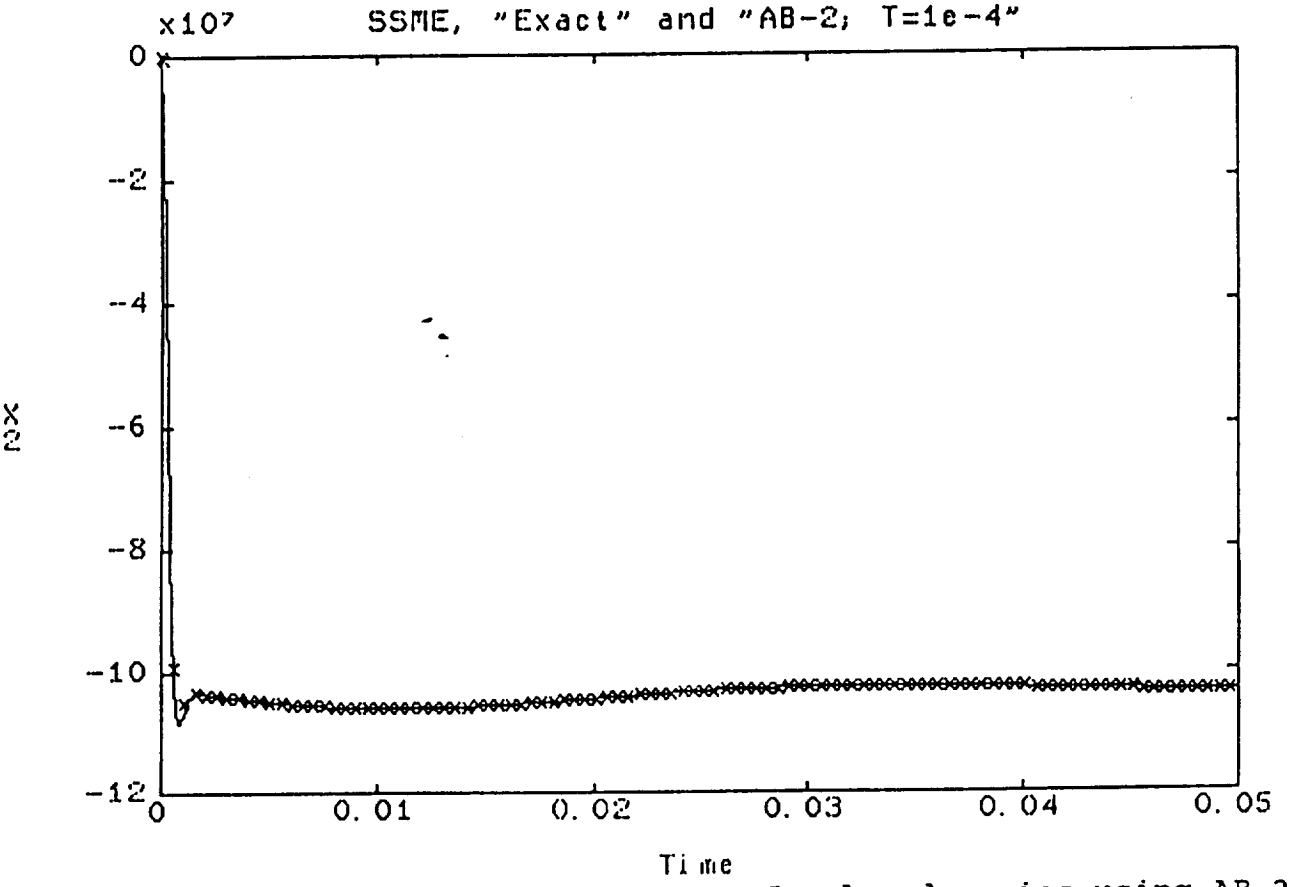
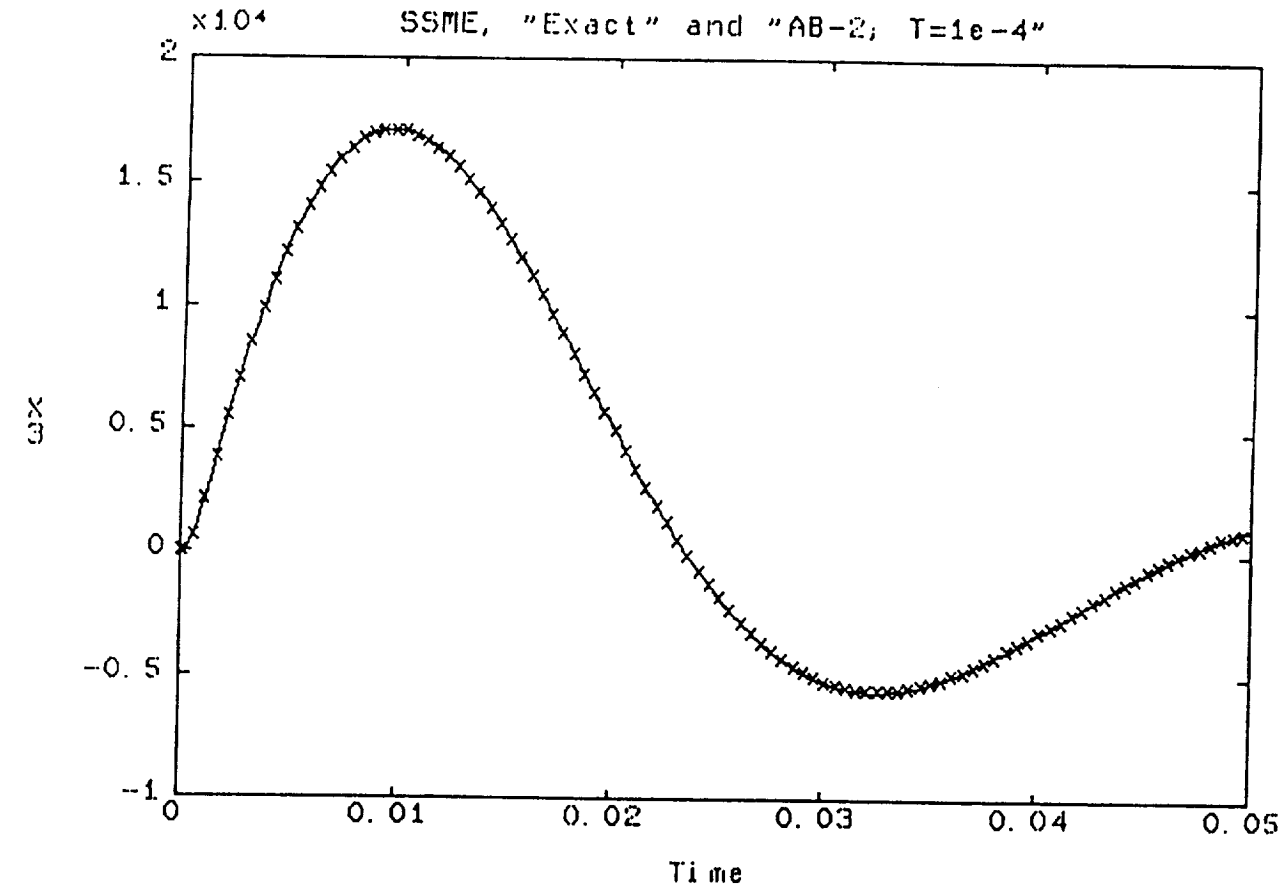


Figure 12. Response of x_1 and x_2 of valve dynamics using AB-2



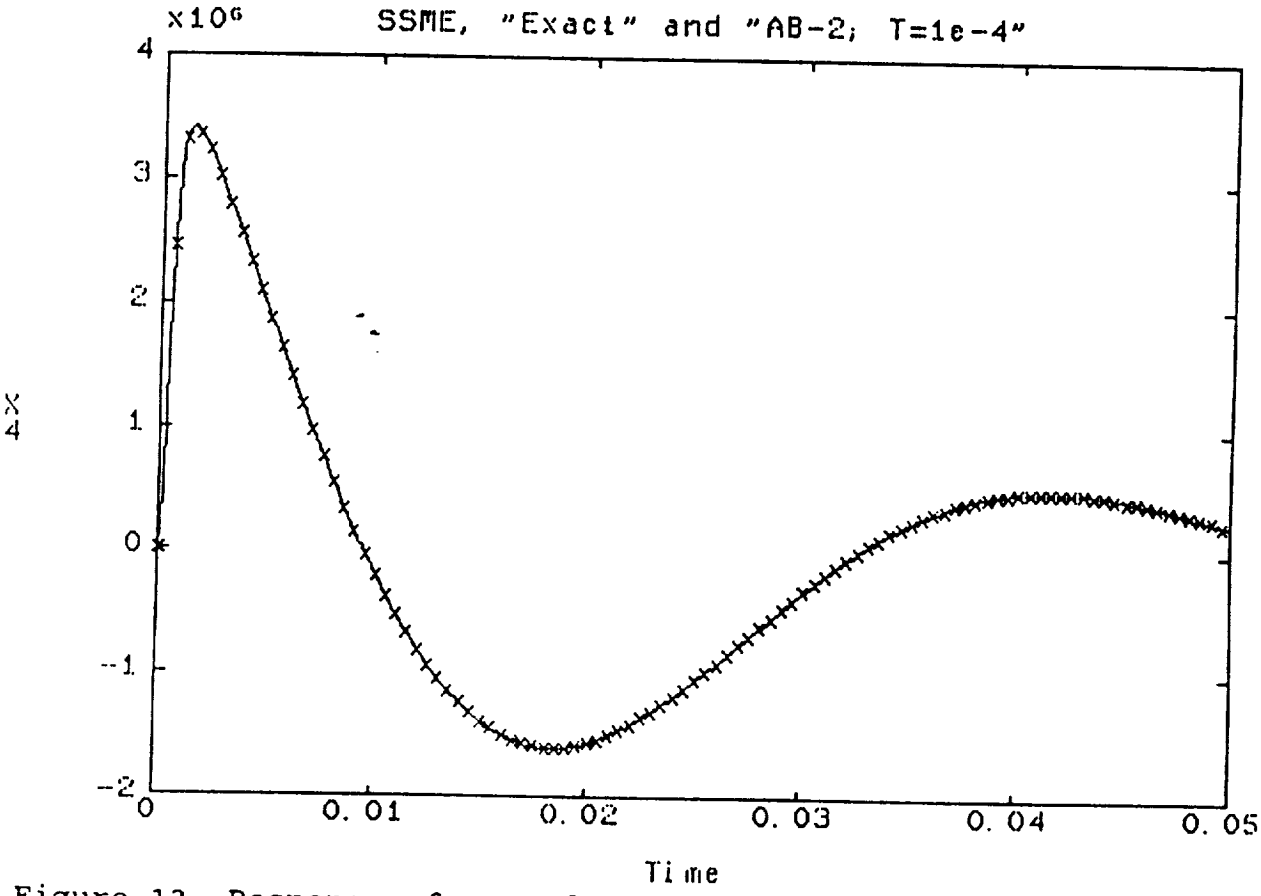


Figure 13. Response of x_3 and x_4 of valve dynamics using AB-2

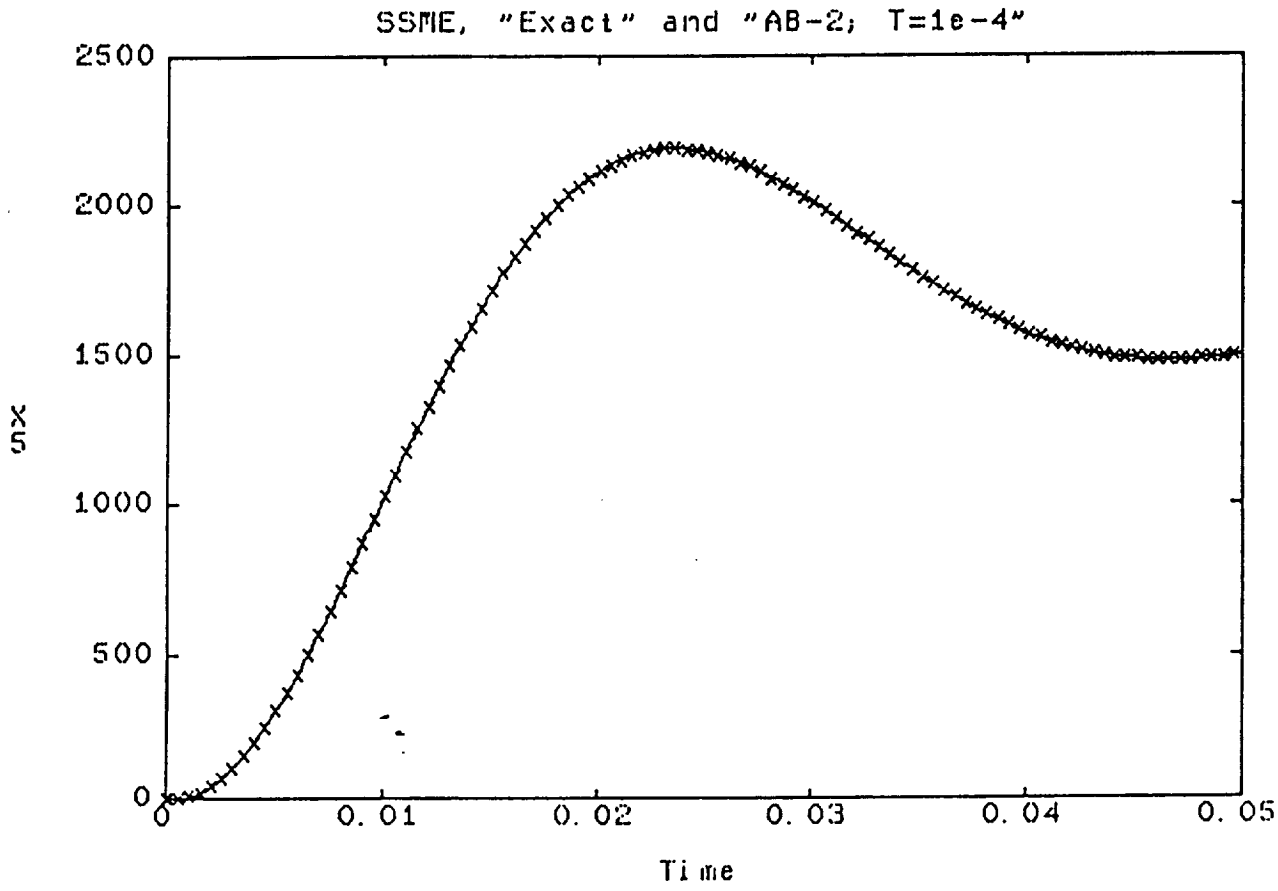


Figure 14. Response of x_s of valve dynamics using AB-2

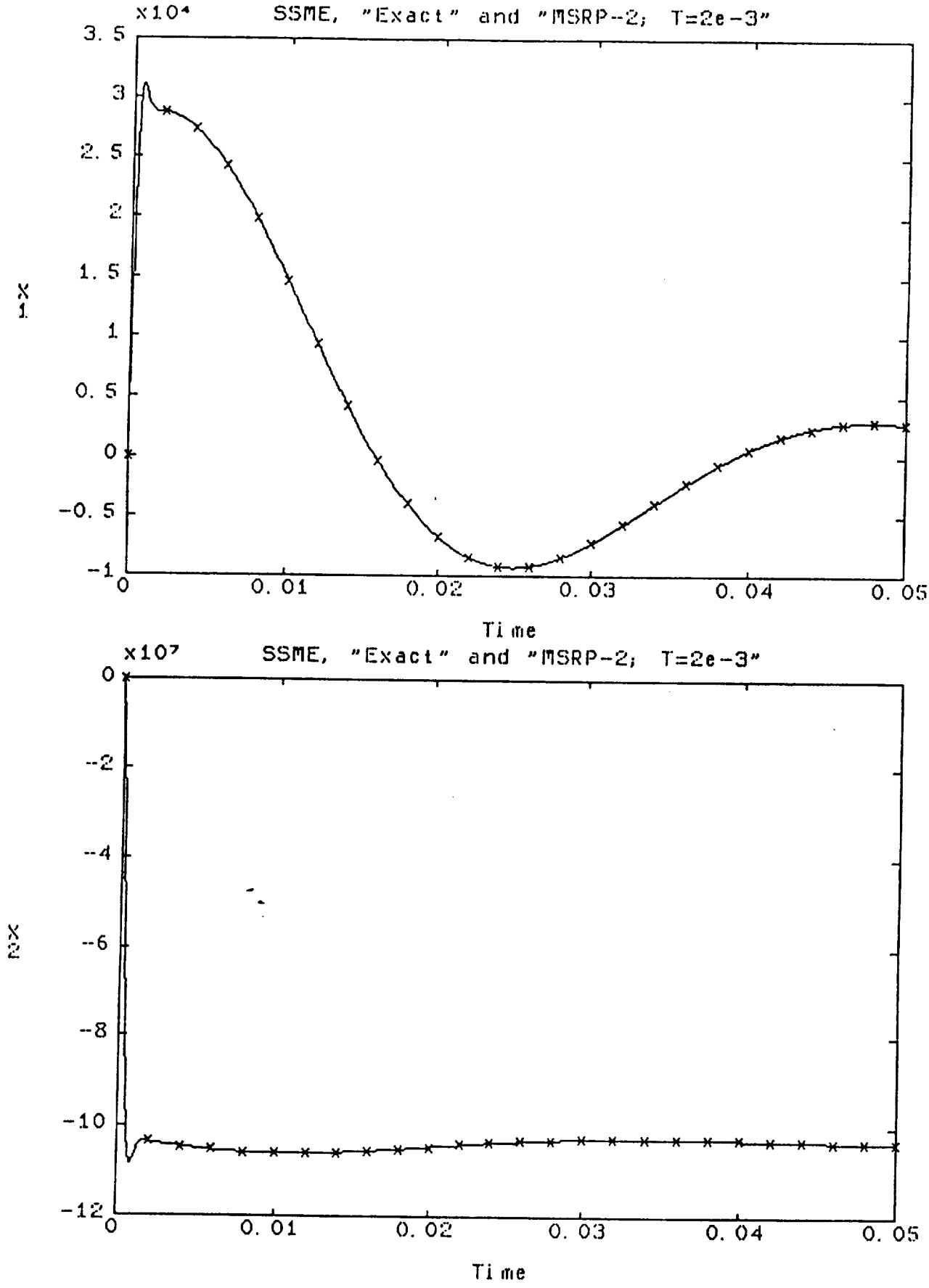
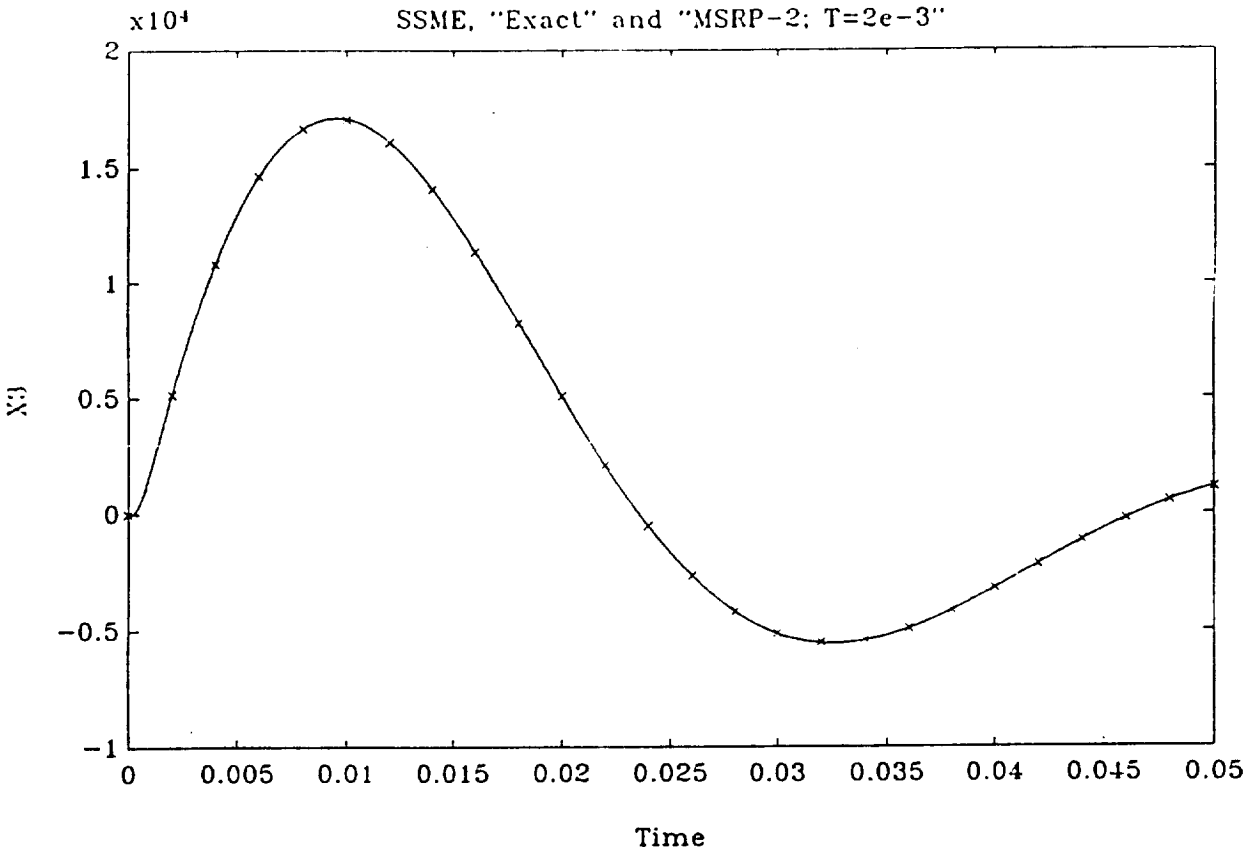


Figure 15. Response of x_1 and x_2 of valve dynamics using MSRP-2



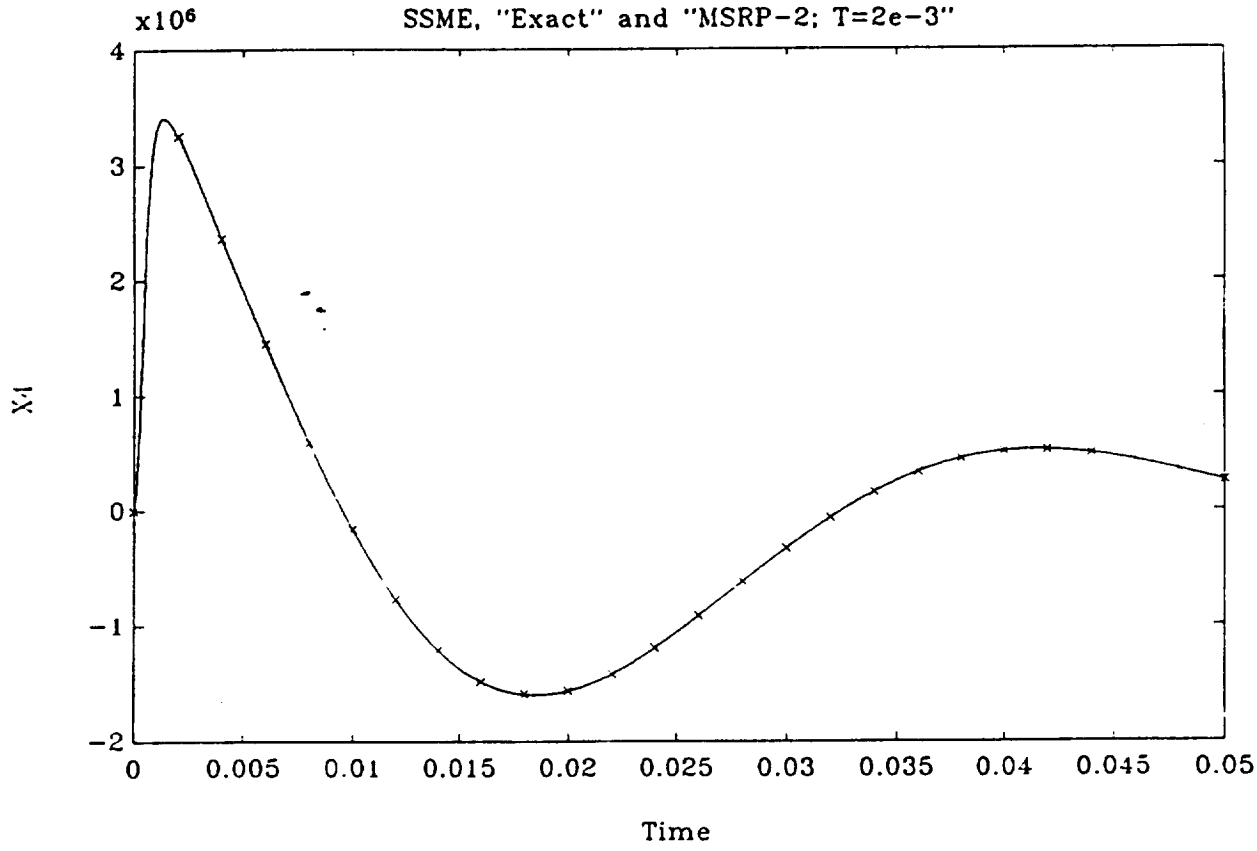
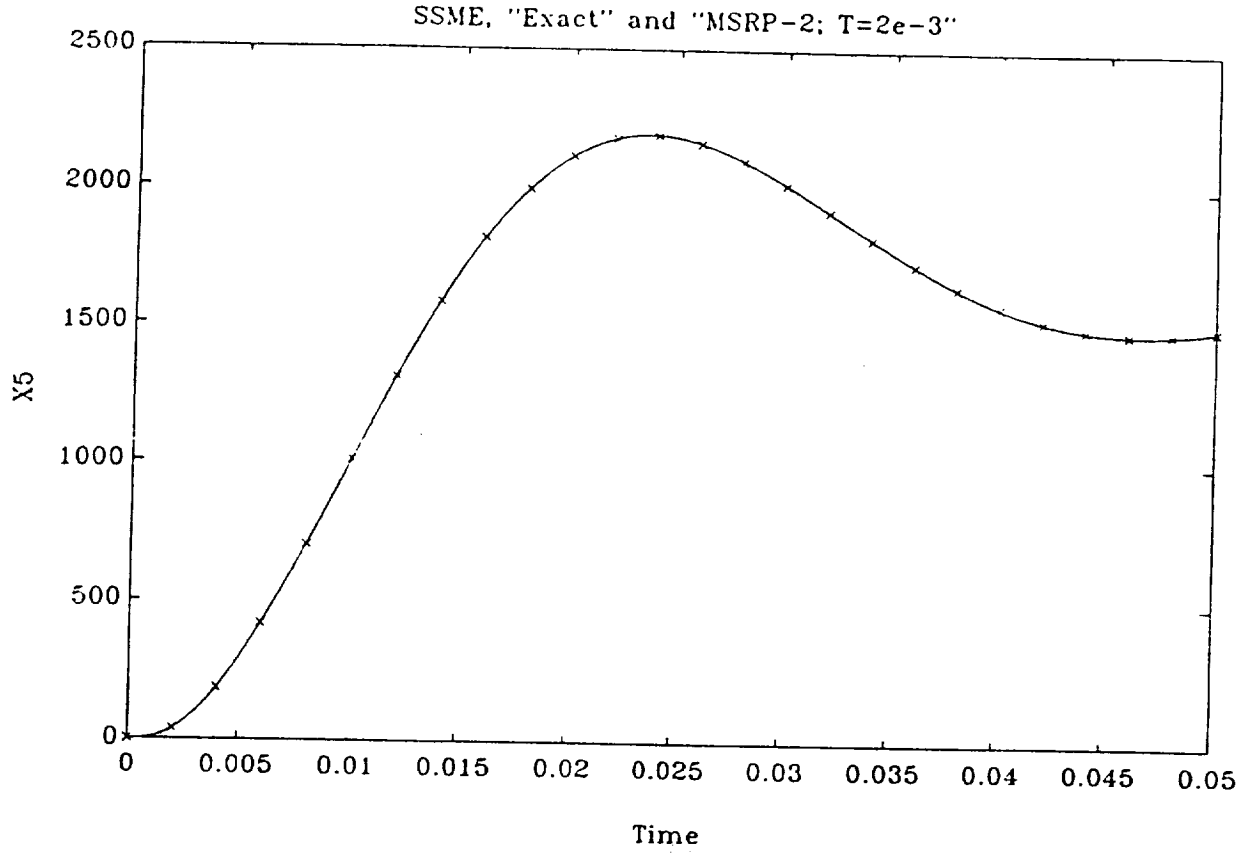


Figure 16. Response of x_3 and x_4 of valve dynamics using MSRP-2



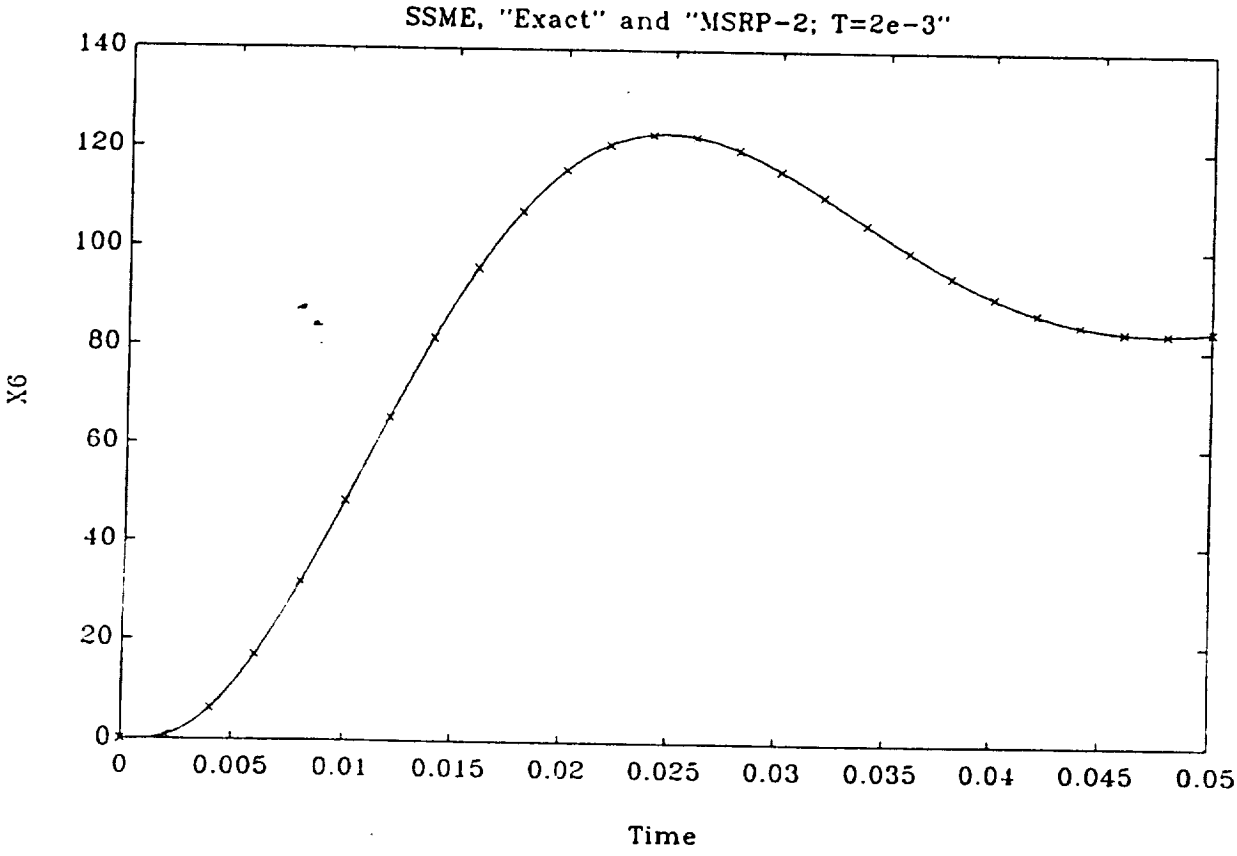
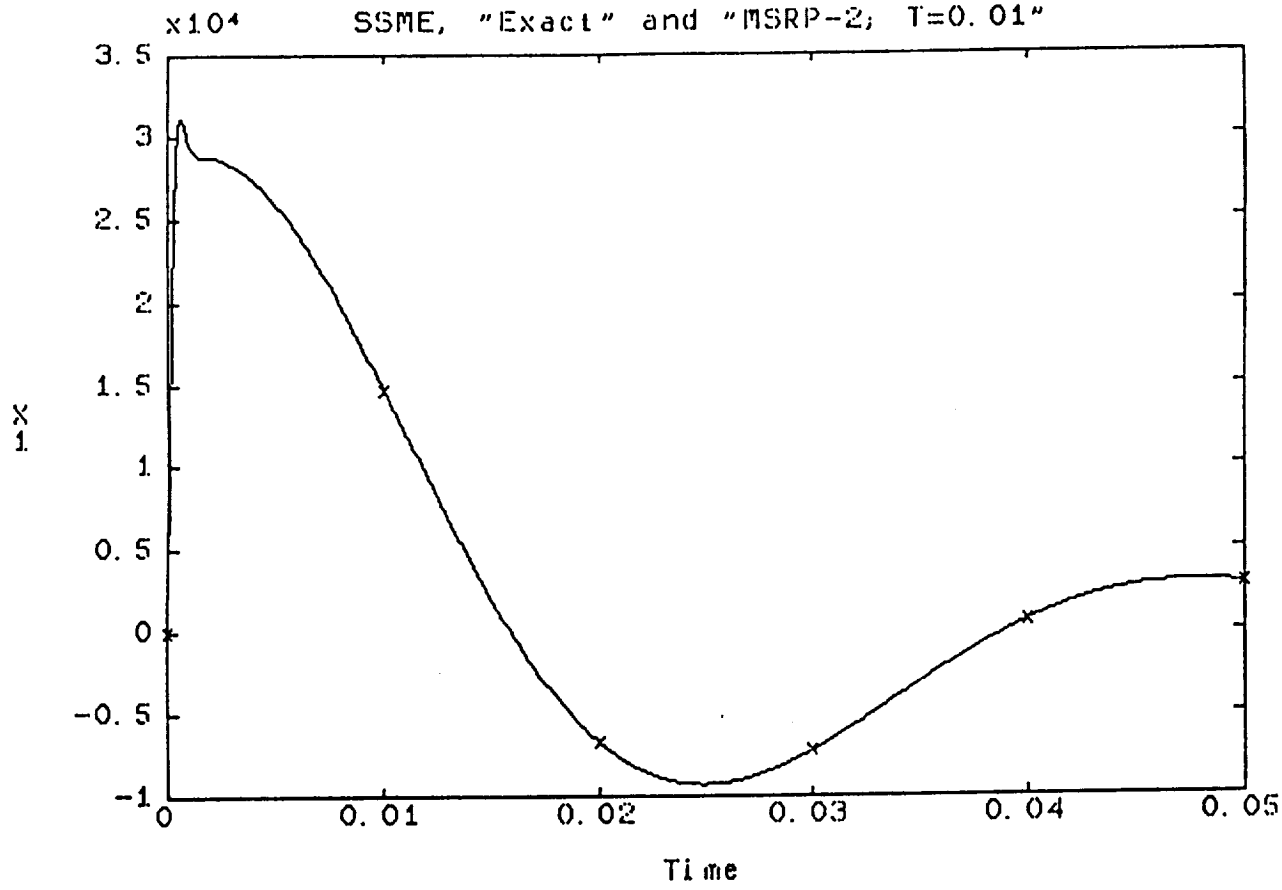


Figure 17. Response of x_5 and x_6 of valve dynamics using MSRP-2



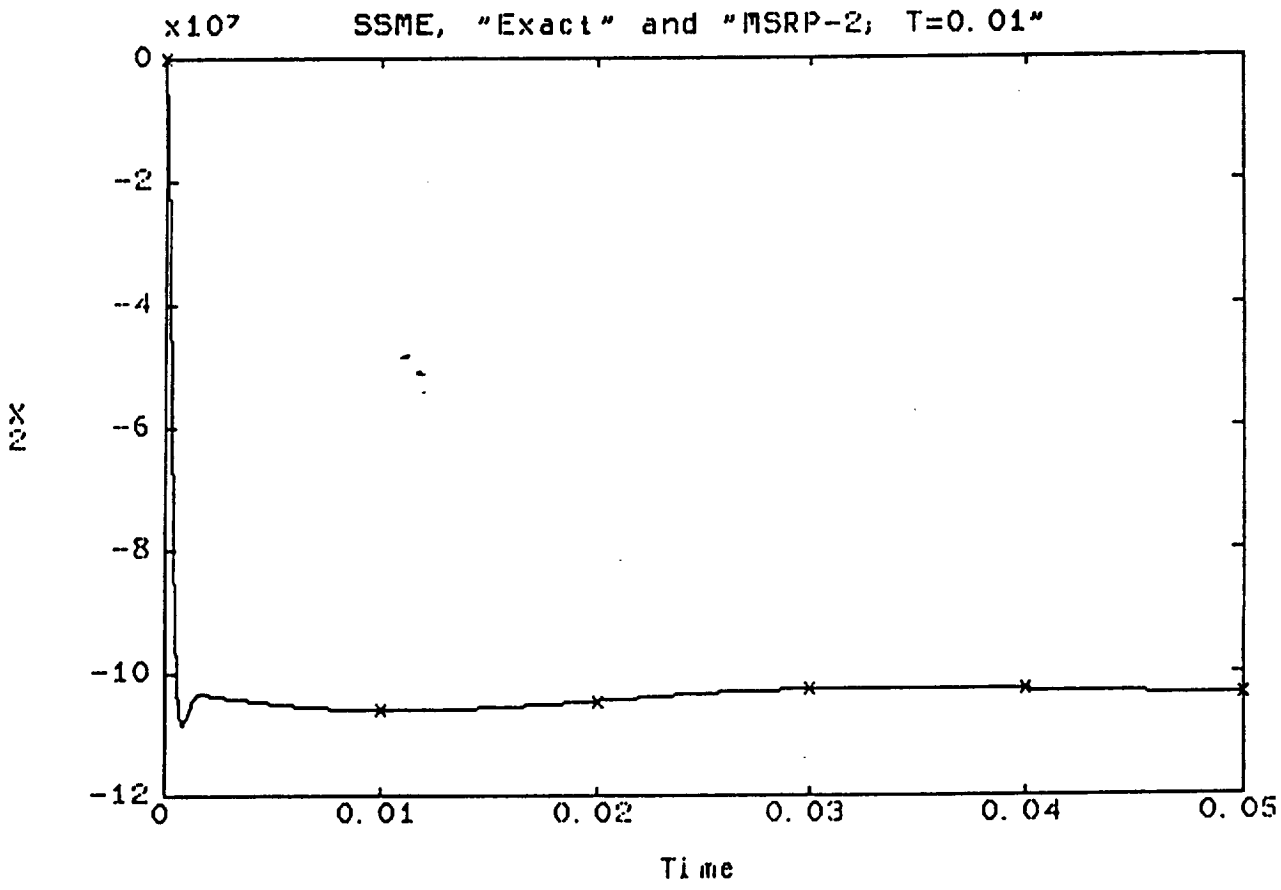
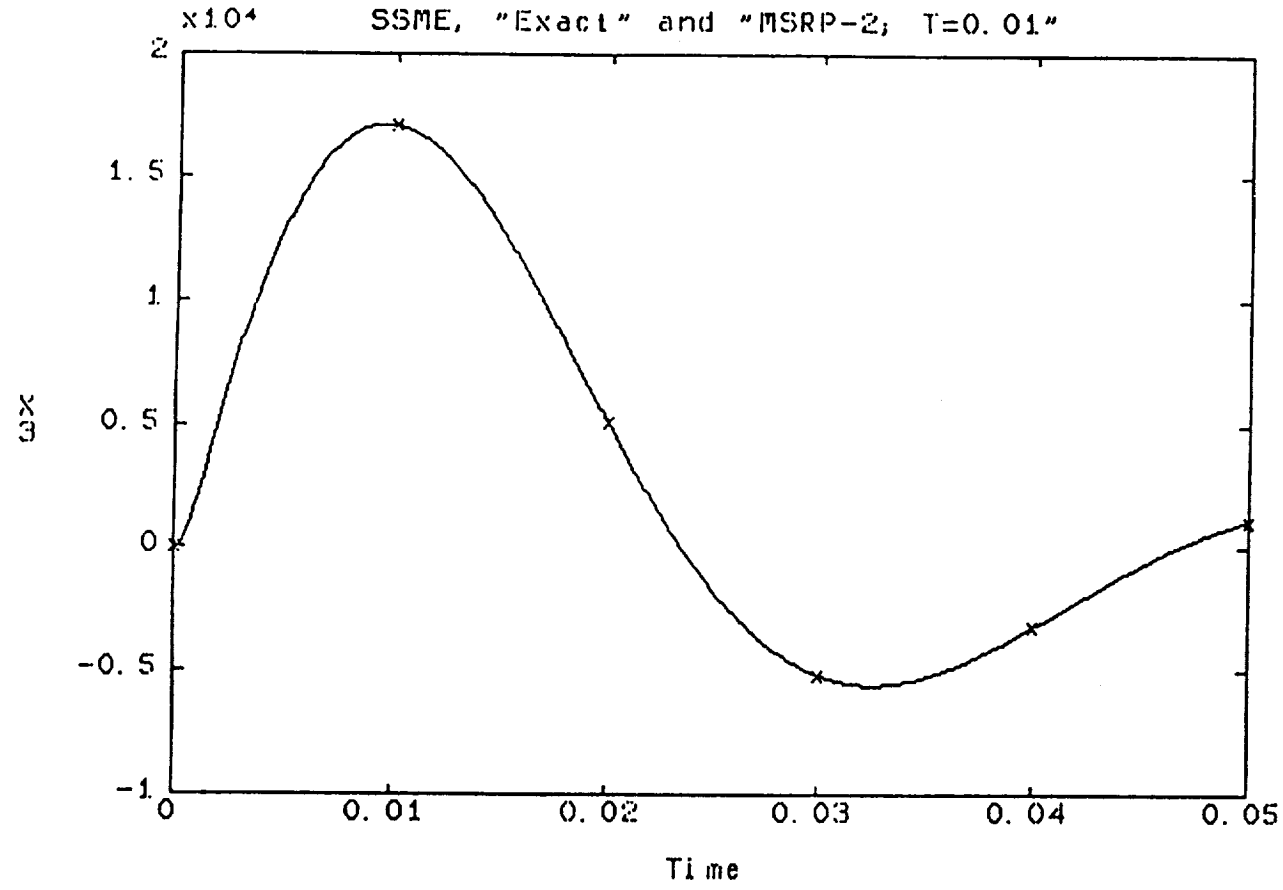


Figure 18. Response of x_1 and x_2 of valve dynamics using MSRP-2



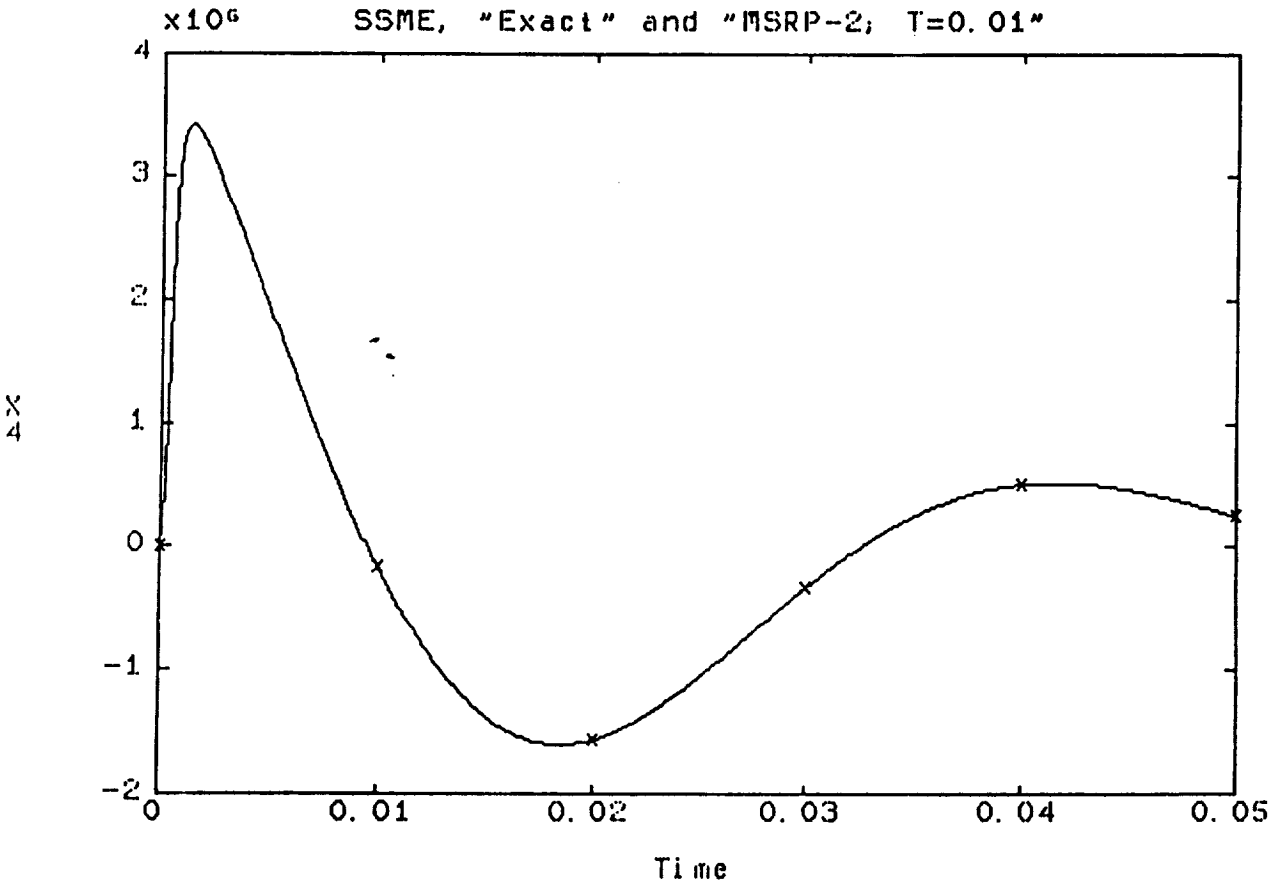
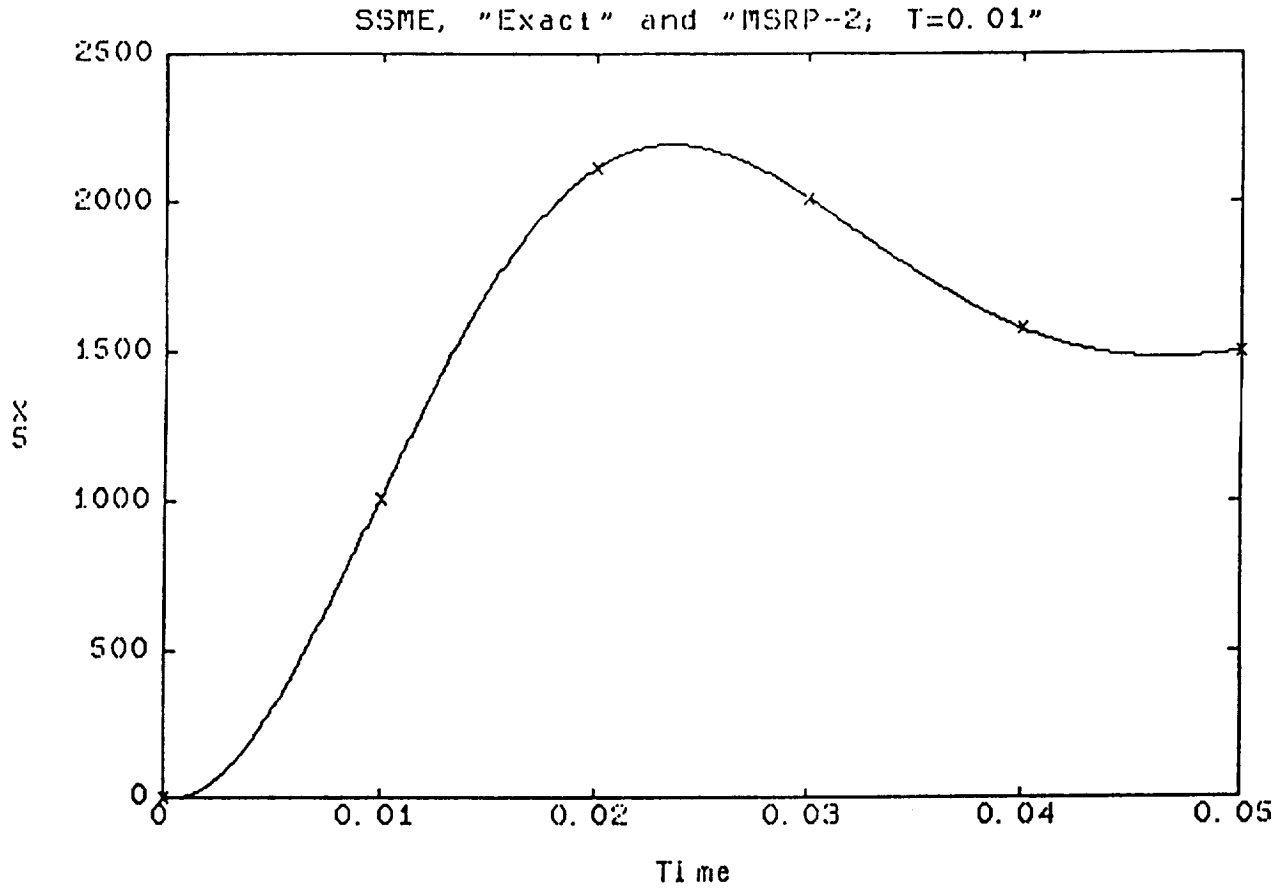


Figure 19. Response of x_3 and x_4 of valve dynamics using MSRP-2



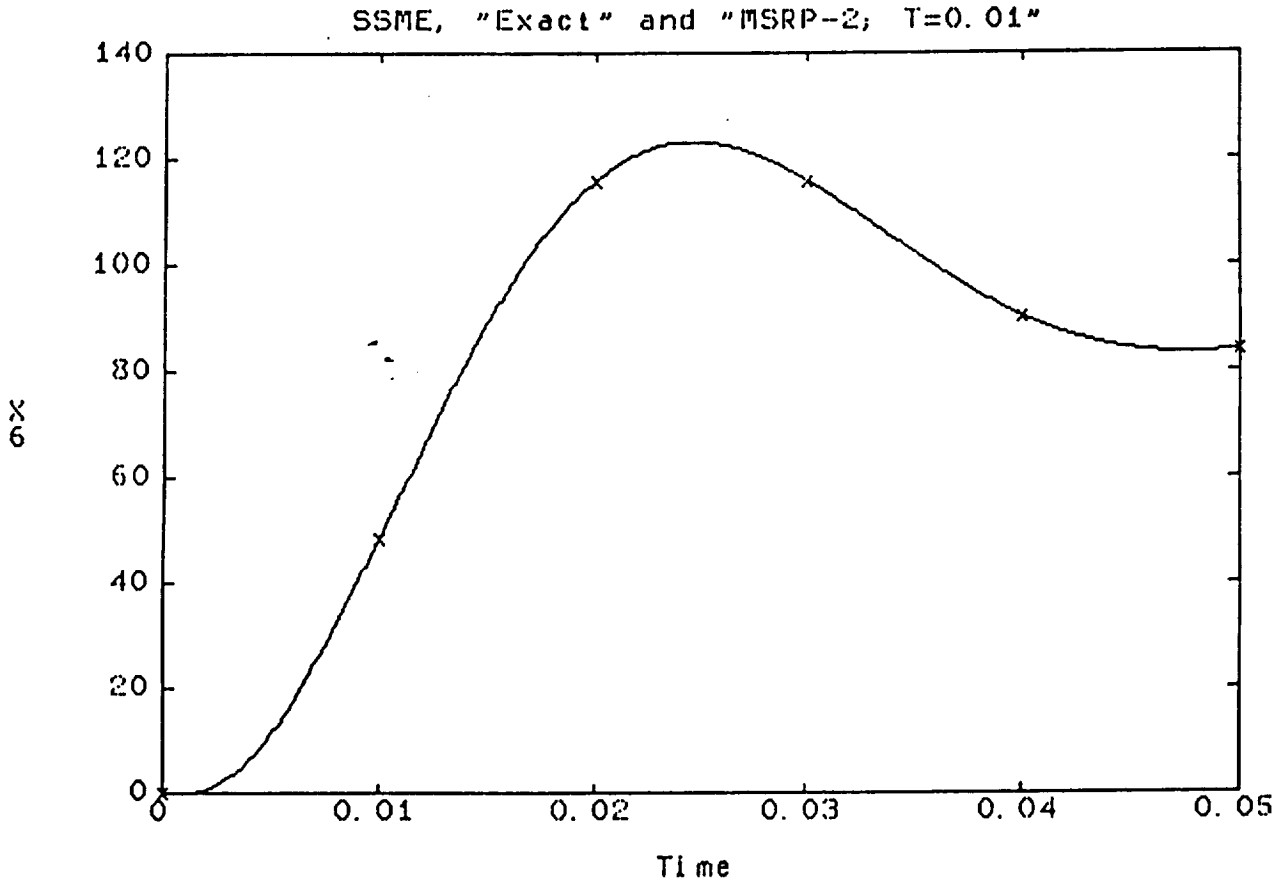


Figure 20. Response of x_5 and x_6 of valve dynamics using MSRP-2

5. IMPLEMENTING TNTEGRATION IN THE SSME SIMULATION

The previous two chapters explain our recommendations of the modified AB2 integration method, over Euler's method. Local MSRP methods, which could be used for much larger step sizes, were also explained. This chapter deals with the practical implementation of the SSME integration methods. It ends with a description of the report version integration module 'integ.for'. A log showing all references to the integration module is included in Appendix C. This log is also available as the file 'integ.log' on the report version diskette.

Numerical Accuracy of an Integration Step

The SSME study version simulation addressed the numerical accuracy of each Euler integration step

$$x = x + xdot * dt$$

by selecting DOUBLE PRECISION as the type for dt. The code generated by Fortran does the following steps:

1. Convert xdot to double precision.
2. Multiply xdot * dt.
3. Convert x to a double precision temporary.
4. Add in double precision.
5. Truncate the sum back to single precision for storage.

Since x and xdot are carried in single precision, extra digits beyond single precision that are generated by the product are not actually significant to the result. The useful part of the above sequence is the way it preserves the significant digits of the product by doing the addition in double precision. Floating point addition of the relatively small increment tends to lose significant digits of the increment, as the increment's mantissa is shifted for binary point alignment with the variable's mantissa. Double precision addition uses a longer register, limiting this loss to extreme cases.

In the report version integration module, all integrators use a method which preserves the significance of the time step increment, and also minimizes floating point conversions. The technique is generally known as "double precision accumulation". The method maintains a double precision variable as an accumulator for x. The time step dt is of single precision REAL type, so that computations of the increment, such as the product x * xdot in Euler's method, are done in single precision. The increment is then converted and the addition is done in double

precision. The double precision accumulator prevents the truncation of significant digits from the time step increment, and x does not have to be re-converted to double precision on the next step. The output value of x returned to the simulation is of single precision type, so that its use in other computations in the simulation does not generate conversions and double precision operations.

Limiting Integration Outputs

When the outputs of integrators are limited, as they frequently are in the SSME simulation, some difficulties arise with both double precision accumulation and multistep integration technique. These complications should not be considered a reason for not implementing the methods, however, because in the SSME simulation, so much computation is required on every time step that almost any method which allows a larger time step is justified, regardless of complications.

If the output of an integrator is changed by a limit, there is the issue of whether or not to adjust the integration process to this change. The study version's Euler method simply replaces the integrated value of x with the limited value, in effect, restarting the integration process. Nothing is lost because Euler's method carries no information from time step to the next time step.

When the output of a multistep method is changed by a limit, the past values are compromised, if not invalidated. Several options are available:

- ignore the discrepancy, using the past values unmodified.
- adjust the past values, taking into account the difference between the unlimited and limited outputs of the integrator.
- discard the past values and use a starting integrator while the output value is on the limit, and for restarting the multi-step method when the output leaves the limit.

The third option is recommended in general for SSME, using Euler's open formula as the starting integrator, for the following reasons:

- it is a fast method, compared to the second option.
- as a starting formula, the Euler's method would only determine the output of the integrator when the integrated value is coming away from its limit. With the low order multi-step methods recommended, these periods are limited to

one time step.

- using the linear approximation of Euler's to restart is at least as good as doing a linear extrapolated adjustment of past values, in the second option above.

The use of Euler's method as a restarting formula is not necessarily recommended for local MSRP implementation, because of the larger step size. A higher order restarting method may be required, but the question was not investigated.

Limiting the output also introduces complications in the double precision accumulation method. To apply the limit to the double precision accumulator, it is not appropriate to reset the accumulator unconditionally to either the output or the limit. This would render the accumulator single precision, despite its double precison type. Instead, it is necessary to know when the output is limited, and to replace the accumulator only then.

In double precision accumulation, efficiency demands that the limits be converted to double precision when the integrator is initialized, and kept in that form, so that continual reconversion of the limit is not required every time it replaces the accumulator. The report version handles limits in that manner.

It is possible that unlimited integrators are more appropriate for some limiting situations in SSME. In this method of limiting, the limits are applied after the integration and are not made known to the integration routine. This would be the case where an integrated process is unaffected by the limit except through feedback into the driving rate. A characteristic of this situation would be that the variable is not expected to come off the limit as soon as the rate changes direction, but only when the underlying unlimited integrator's value would cross the limiting value. It is beyond the scope of the study to distinguish these kinds of limiting situations from others, so all limiting situations are modeled in the report version code by limited integrators.

In situations where operation on the limit is not expected, a limit should not be silently imposed on an integrator, just to stay within bounds of tabulated data. In the report version, the corresponding unlimited integrator is used, and the interpolation routine stops the simulation with an identifying message, if the interpolated table's range is exceeded.

The Report Version Integration Module

The report version integration module 'integ.for' implements

the recommended modes of operation, and also provides a recoded form of the study version's Euler method integration. This makes it practical to operate the simulation with all other forms of speedup in place, but with no change in integration policy or step size, as called for in the earlier recommendations for tuning the SSME simulation. The integration method may then be changed to the modified AB2 with minor recommenting and recompilation of the 'integ.for' source code alone, followed by a relinking step.

A number of integration routines are required to handle the different situations encountered in the SSME simulation. The following short glossary covers the terminology used in the report and in the report version comments to distinguish integration routines:

primary integrator - an integrator used outside of any energy balancing loop. It produces a final value for the time step.

trial integrator - an integrator used in an energy balancing loop. It produces a trial value which is accepted as the final step value when the balancing loop has converged.

unlimited integrator - an integrator whose accumulator is controlled only by initialization and rate input.

limited integrator - an integrator whose accumulator is subject to limits given to it at initialization. The behavior of limited integrators was described above.

flow integrator - an unlimited integrator with the SSME gas flow rate calculation built in. In 'integ.for', it comes in primary and trial flavors.

method - the integration method, such as Euler's method, Adams-Bashforth Second Order, or MSRP.

The integration module consists of the following routines:

- initiation subroutines for limited and unlimited integrators of any method, and of either primary or trial, or flow types.

- primary integrators of limited and unlimited forms, and using Euler and modified AB2 methods. Comments in the source code of these routines are edited to switch integration methods throughout the simulation.

- trial integrators of unlimited form, using Euler and modified AB2 methods. These routines are selected by recommenting

whenever the primary integrators are.

- flow integrators of primary or trial types. These call the primary and trial integrators, and are not changed when switching integration methods.

- step update functions each method of the trial integrators. These are recommented to select the one corresponding to the trial integrator.

The flow integrators replace study version flow integrators based on solving an equation representing the implicit, or closed form of Euler's formula. This method computed the flow rate in double precision, even carrying out a square root to this accuracy. The method is costly in computing time, especially since the flow integration occurs frequently inside of energy balancing loops.

The effort of this calculation is not justified. The Euler formula is only first order, so the double precision results obtained represent only a linear approximation to a curve being followed to single precision accuracy. Implicit formulas are normally used in corrector formulas, where they represent one or more corrective iterations at the same time step.

Double precision in the flow rate calculation may have been adopted in the face of balancing convergence problems. If so, it is of no more than accidental value as a solution to those problems. A more detailed monitoring system and relaxation controls for balancing convergence is recommended instead.

The integration module hides past multi-step values, double precision accumulators, and trial accumulators from the rest of the program. Only the integration routines have access to them. The calling routines refer to these values by a unique number assigned by the programmer to the integrand value. A major reason for the log file is to have a compact record of the number assignments.

Space can be saved in the report version by having different sets of arrays supporting the hidden values for the different types of integrators, rather than one set of arrays for all. The hidden values are defined in the integrator module's labelled common block 'integ' in the file 'integ.com'.

6. INTERPOLATION IN THE SSME SIMULATION

Interpolation of values from tabulated data is a major consumer of computer time in the SSME simulation. Linear interpolation of unequally spaced data points is the prevalent form of interpolation. It is used for function generation, with (x,y) data read from the file 'dtminp.dat' in initialization sections of each module. Two dimensional linear interpolation on temperature and pressure surfaces was used for gas properties. Cubic spline interpolation was used in one such table.

Linear interpolation routines in the study version were not coded for optimum efficiency, and there was a large adverse effect on running time.

With unequally spaced data points, it is necessary on each interpolation to identify the interval

(x_i, x_{i-1}) of the independent variable enclosing

the input value x. The study version's 'fgen' does a linear search from the low end of the table, comparing x to values x_i in increasing order. This is suitable for short tables, or for tables in which most of the simulation time is spent within the few lowest intervals. Otherwise the expected number of comparisons, half the size of the table, is excessive.

From one time step to the next, one would expect the input variable to be in the same interval as it was previously, most of the time. Accordingly, a generally better strategy is to test the endpoints of the previous interval, and if exceeded, search in that direction for the enclosing interval. The report version of 'fgen' does that, favoring the upward direction by testing the upper side of the interval first. Two comparisons are required when the input variable stays within the same interval, two are required when it moves up an interval, and three are necessary when it moves down an interval, for an average close to two.

Another serious fault with the study version function generation is the failure to precompute coefficients for the interpolation polynomial. The study version interpolation, given the interval index i, is of the form

$$f(x) = y_{i-1} + \frac{(y_i - y_{i-1}) * (x - x_{i-1})}{(x_i - x_{i-1} + 10^{-20})}$$

For an inner loop operation of a major simulation, this coding is truly ugly. Three additions and the division can be done once,

when the function values are loaded. Instead they are done on every interpolation. The extra constant term guards against division by zero when x values are repeated in the table. Such an occurrence makes no sense in a function table. The program is spending simulation time to quiet the operating system's complaint about an error in the function data!

In the report version, the corresponding interpolation is of the form

$$f(x) = a_i + b_i * x$$

and the loading routine complains if x_i is too close to x_{i-1}.

The function generation module assigns each function table a unique number, to be used on interpolation calls. The number is a key to the previous interval, the starting point for the interval search. In the report version, this number locates precomputed coefficient values a_i and b_i as well.

In the study version, tables were stored in a two dimensional array. The function number was a row index in the array. There were 15 values per row, so functions were limited to that many points. Functions shorter than 15 points did not fully occupy their rows. In the report version, a one dimensional arrays are used for x and y values. The function number locates the starting point and size of the table in the large array. This method places no constraint on the size of a function table, allowing more accurate approximations to be formulated where necessary, by using more than 15 data points. The method also avoids wasting memory where functions have fewer than 15 points.

In the SSME simulation interpolations within the same table can be used to obtain current values for several variables within the same time step. Therefore the function number, which identifies only the table, cannot be used to recall the previous interval expected to enclose the input value. Instead, a unique value is assigned to the variable within the routine which calls the interpolation module. The interpolation module provides array space for the index of the previous interval, and calling routine passes the variable's index to the interpolation module.

New functions are added to the simulation by placing (x,y) points in the file 'ssme.dat', adding a call to the function loading routine 'fgset', and adding interpolation function references to 'fgen'. All one dimensional linear interpolations should be treated this way, to take advantage of the optimal coding of fgen. The function TLIMIT in the control module 'cntrol.for' was redefined as an fgen reference.

Interpolation of Functions of Two Variables

Several forms of interpolation are used on gas property tables defining temperatures and pressures as functions of two variables. Data points were unequally spaced in both dimensions.

The study version interpolation routines used the search method recommended above to find the enclosing interval in each dimension. Running time was excessive, however, because interpolation coefficients were not precomputed.

The study version two-variable interpolations differed from the function generation routines in that the data tables were incorporated into the simulator as arrays initialized by Fortran DATA statements. There are no compelling reasons to prefer this method over loading from a data file. The DATA statement method does the conversion to binary at compile time, rather than during initialization, and input from a data file becomes input from the load module. With virtual memory the load module data is not actually loaded into memory until needed. But in the SSME simulation all of it is needed during initialization, for the precomputing of interpolation coefficients. It is doubtful that the saving in initialization time would be noticed.

In the event there is a simulation requirement to manipulate gas properties, such as investigation of the effect of impurities, gas property tables should be moved to data files, and an open, accessible format for editing the tables should be adopted, such as in 'ssme.dat.

Cubic Spline Interpolation

Cubic spline interpolation is used in the two dimensional interpolation routine 'O2PROP' (SSM52510). There was no explanation available as to why cubic spline interpolation was required for this table, when linear interpolation was sufficient for hydrogen and OXPROP oxygen properties. Perhaps it was adopted in an attempt to get convergence within energy balancing iteration of OPRIME, which calls O2PROP.

In O2PROP, two-way cubic interpolation is attempted, in the same manner that two-way linear interpolation is done in other modules. Spline interpolation is done on the two adjacent columns of the table which bound the row input value, then the spline interpolation routine is called to interpolate along the row. This procedure does not actually achieve cubic spline interpolation smoothing, however. To do that, additional row values would be required. Two are not enough to define the smooth curve which is desired.

For the report version, the erroneous application of the spline routine is replaced by linear interpolation. Apparently, true two-way spline interpolation is not actually required for energy balancing convergence, or whatever purpose it was intended to serve.

For the sake of speedup, the elimination of cubic spline interpolation from O2PROP should be attempted. The report version coding shows the large initialization time and coefficient space required. Better observations of the convergence problems, and relaxation adjustments will make energy balancing more robust, and will probably make spline interpolation unnecessary.

Without addressing further the need for cubic spline interpolation, which was outside the scope of the study, it was possible to assess the efficiency of cubic spline implementation in the study version, and to recommend an improved version for use where it may be required. In keeping with the modularization policy of the report version, cubic spline routines were added to the interpolation module 'fgen.for', so that cubic spline interpolation could be tried in other table lookup situations within the simulation, with a minimum of programming effort.

The deficiency of the study version's cubic spline implementation is that it does not carry precomputation of interpolation coefficients far enough. Ideally, once the index of the interval enclosing the input value has been determined, an interpolated value on the cubic arc for the interval can be computed with three multiplications and three additions, as

$$f(x) = a_i + x * (b_i + x * (c_i + x * d_i)).$$

The study version of the spline interpolation precomputes second derivative constants, instead of cubic polynomial coefficients, and the interpolation routine working with this precomputed data requires 7 additions or subtractions, 12 multiplications and a division. This is at least four times the cost of the report version's cubic polynomial evaluation.

True two-way cubic spline interpolation is very costly, compared to the interpolations used in the simulation, not only because many points are needed for interpolation in the row direction, but also because the precomputation of cubic segment coefficients is not possible.

The linear interpolation module

The report version module 'fgen.for' contains all routines of one and two dimensional linear interpolation, and cubic spline interpolation. Interval finding, precomputing of interpolation

coefficients and the interpolating routines are included. The file 'fgen.com' defines the labelled COMMON block used by the interpolation routines.

The study version's FGEN routine was replaced by a subroutine 'fgset' for loading function values and precomputing interpolation coefficients, and a separate linear interpolation function 'fgen'. Two-way linear interpolation is also provided in a split package, with precomputing handled in the subroutine 'xyset', and interpolation by the function 'xylint'. One interval searching routine, 'intval' serves for all interpolation with unequally spaced independent variable data.

A maintenance aid file 'fgen.log' records the references to the function generation routines 'fgset' and 'fgen'. A copy is included on the report diskette.

7. ENERGY BALANCE CONVERGENCE DIAGNOSIS AND CONTROL

Two energy balance iterations are performed at every time step in the SSME simulation. The fuel flow module contains an iteration loop carrying a sequence of calculations over 12 stages. The oprime module, called by by hotgas during the first 1.5 seconds, iterates over four stages.

Limits on these iterations were set within the study version source code at 30 iterations. Any difficulty in convergence of these energy balancing loops can be very costly in simulation running time. These loops involve flow rate calculations, which were implemented in a time-consuming manner in the study version.

The study version monitors the convergence of these iteration loops by writing out messages, and the last two iteration changes, when the iteration limits are exceeded. The number of iterations is also written as an output. While this monitoring is sufficient to determine when there is a convergence problem, it is not adequate to diagnose what could be wrong, and it provides no corrective tuning action.

The report version contains an energy balance diagnosis and control module, in the file 'change.for'. This module associates a unique number with each energy balance variable, and uses it as an index to store the number of iterations required for that particular variable to converge. Thus slow converging variables can be identified, and the computation of these variables examined for disturbances to convergence.

The warning system for energy balance convergence problems is similar to the study version's, but provides more information. A diagnostic file is written on any run in which the iteration limits are exceeded. The diagnostic file shows the number of iterations and the relative change level achieved, for each energy balance variable. After this information is written, the diagnostic data output is repressed for 100 iterations, so the simulation may continue to run, and information over a longer period of time may be collected.

A second means of monitoring is available, suitable for close inspection of troublesome intervals. The number of iterations for convergence of each variable can be chosen as an output variable, and thus can be collected at the data collection interval, throughout the simulation.

'Change.for' also provides for a commonly applied remedy to multi-variable convergence problems, the relaxation technique. A relaxation factor for each energy balance variable is included in

the run parameters on file 'ssme.run'. This factor can be changed to brake, or accelerate, the influence of newly calculated values on the iterated value. Instead of simply replacing the old value, the relaxation method computes the iterated value by

```
iterated = factor * new + ( 1. - factor ) * old,
```

where `0. < factor < 2.0 .`

The normal default value is 1.0, implementing a simple replacement. For a variable that tends to converge slowly, the factor is decreased, under-relaxing or damping the corrective calculations. Fast converging variables can be over-relaxed by increasing their factors above 1.0. Their convergence can often be accelerated in this way, providing more stable conditions for the slow convergers.

Contents of the 'Change' Module

The source file 'change.for' provides initialization routines for setting the relative change tolerances, and for initializing convergence iteration counts of energy balance variables at each time step. The FUNCTION 'relax' applies the relaxation factors, and counts iterations in which the variable change is above tolerance. A convergence alert routine 'wrchg' writes the diagnostic file 'ssme.cvg' and a message on the monitor screen.

8. REAL EXPONENTIATION

One possibility for speedup in a digital simulation system is to take advantage of the fact that the six to seven significant digits of accuracy normally delivered by software supported special functions may be more than is required. This is probably the case in the SSME simulation. It appears that a significant reduction in running time can be derived by using special means to compute the function

$$f(x) = x^{(0.1*n)},$$

This function is used frequently in the simulation loop, in the form shown, and as the square root function (n = 5). In the report version, a module is devoted to special methods to compute it.

Without hardware support, the function x^y, where y is a real data type, is normally computed by normalizing both x and y to unit intervals, and using economized series or rational approximations for a logarithm of x, and the exponent of (y log x). The logarithm approximation is particularly slow to converge, and many terms are necessary for the 7 place accuracy normally required in Fortran libraries.

Many floating point hardware systems provide hardware support for real exponentiation, and the use of this hardware by the compiler makes special methods of computation unnecessary.

In the report version of the SSME simulation, all exponentiation and square root function calls have been replaced with calls to routines in the module 'xtoy.for'. The primary routine in this module does a two-way linear interpolation of

$$f(x,n) = x^{(0.1*n)},$$

in one of two tables of equally spaced points (x, n). One table covers normalized x values from zero to 0.2, and the other covers normalized x from 0.2 to 1.0. Other routines in the module perform normalizations and call the primary module.

The module as implemented provides maximum speedup over a software supported exponentiation. The maximum relative error is close to 0.5 per cent. The accuracy could be improved, at the cost of more operations per exponentiation, by using more tables, or by going to a higher order interpolation. In cases where exponentiation is hardware supported, the 'xtoy' module routines can be replaced by simple routines using exponentiation expressions which take maximum advantage of the hardware support. No changes need be made in the calling simulation modules, to implement such a customized version.

9. AN OUTPUT SYSTEM FOR THE SSME SIMULATION

The advantages of selected binary output have been described earlier in the results summary of this report. A recommended implementation is provided in the report version of the code, and is described here.

The output system consists of several components:

1. An ASCII output definition file 'output.def' lists all output variable names, and can be edited to select those variables to be collected on a particular run. The selection file is identified by an 80 - character header.

2. An output initialization segment in the main program reads the output definition file, and sets collection index arrays for the run. The segment opens the output file and writes the input parameter header to identify the run set up, and the output selection header to identify the selection. The collection index array and corresponding variable names are written to the output file.

3. A code segment in the main program monitors the simulation loop counter and triggers output collection. The report version uses integer operations, and is simpler than the study version, offering a single collection frequency.

4. All potential output variables are contained in the COMMON block 'outvar', integer variables first.

5. The routine 'writer' views the 'outvar' COMMON block as an integer array and a real array. It reads output variable index numbers and collects the corresponding data from the COMMON block. It writes the collected data to the output file in one binary record, including the simulation time.

6. Offline print and plot programs read headers, variable names, and data from the output files, convert the data as necessary, compute functions of output data as required, and present the output.

An example of an offline output program of item 6 is the program RTable in the report version file 'rtable.for'. This program tabulates selected real data from the simulation run in up to 10 columns. Whenever there is a function of output variables to be plotted, or a new display device to utilize, a small program similar to RTable can be written, instead of altering the simulation program.

10. A ROUGH ESTIMATE OF SPEEDUP

Our time had to be spent on analysis of the SSME model, and the definition of changes needed for a faster and more accurate simulation program. An inspiration for this effort was the following rough model, offers some insight into the problem of how far to go optimizing the SSME simulation.

Assume the running time about equally divided between output operations and computation. Selective output will encourage more runs, but should reduce the output per run by at least a factor of 10. The encouraged extra runs are worth the price, and should not be assessed as penalty.

Doing data conversions offline relieves the simulation host of a huge load. Binary to decimal conversion is iterative, and floating point conversion to characters costs. Of course, this means that conversions will be repeated, as data is viewed, but on personal computers and workstations, this time is free.

If the output processing per datum is reduced by a factor of 10, we have achieved 100 to 1 speedup on the output operations side.

On the computation side, some of the savings are additive, some multiplicative. It is reasonable to assume the integration processing time is no more than one thousandth of the rate computation time on each integration time step. So if integration methods adopted are 10 times more costly than Euler's, if doesn't matter. Five times the step size still essentially means one fifth of the computation.

Given the size of tables, there is probably a factor of three saving in search time, and precomputation of slope data preserves that factor through interpolation computations on the interval. Optimization outside interpolation may be close to that. The saving on flow integration by going to single precision can be counted here. So can replacement of series evaluations for exponentiation. We believe there is an overall factor of three available from all numerical optimization sources.

A large part of the simulation is subject to a third factor, the reduction in the number of energy balancing iterations. Relaxation of energy balancing loops would dramatically reduce large iterations. Smaller ones would require less improvement to register a good factor of improvement. We estimate a factor of two for energy balancing optimization, applying to 1/3 of the simulation. Roughly, we are assuming that energy balancing portions occupy a third and valve dynamics

requires a third. We ignore the initialization and loading time.

In this rough model of computing time, the half devoted to output is subject to a 10 * 10 speedup. The time for energy balancing loops, say one third of computing time, or a sixth of total time, benefits from integration(speedup = 5) and balancing relaxation (speedup = 2). There is also a predominance of flow integration in this part, so we project a speedup of three from numerical optimization and elimination of the double precision closed Euler's scheme. To the nonbalancing part, we predict a numerical speedup closer to two.

According to this rough model, the running time is reduced by a factor of

$$(1/2)(1/100) + (1/5)((1/6)(1/3)(1/2) + (1/3)(1/2)) = 1/22.8$$

output; integration; balance numerics; relaxation; non-balance numerics

Under these assumptions, suppose local MSRP were introduced, with a time step of 20 times the Euler step size. At worst, the computation costs might equal the numeric savings of the report version. We might expect to lose the speedup due to relaxation, due to larger "first guess" errors on energy balancing. The resulting running time factor would be about

$$1/200 + (1/20)(1/6 + 1/3) = 1/33.3$$

This would be a good result, but the resulting simulation would be only half again faster. On the other hand, it is possible that energy balancing iterations would increase significantly at this step size. If balancing iterations were to double the current values, in spite of the relaxation technique, then the resulting time factor could be

$$1/200 + (1/20)(2/6 + 1/3) = 1/26.1,$$

largely negating the gain from MSRP.

Finally, consider what happens if integration and numeric speedup works as expected, but no action is taken on the output system. The first running time factor above becomes

$$(1/2) + (1/5)((1/6)(1/3)(1/2) + (1/3)(1/2)) = 1 / 1.86$$

and there is no prospect of doing much better, since infinite computational speedup results in a factor of 1/2.

REFERENCES

1. J.D. Lambert, "Computational Methods in Ordinary Differential Equations," Wiley, NY, 1973.

2. S.P. Chicatelli, "Graphical Stability Methods for Numerical Integration of Ordinary Differential Equations," M.S. Thesis, The University of Akron, Akron, OH, 1989.

3. D. Keck and T.T. Hartley, "A Comprehensive Study of Linear Two-Step Methods," Proceedings of the Seventeenth Annual Pittsburgh Conference on Modeling and Simulation, April 1986.

4. J.A. De Abreu-Garcia and T.T. Hartley, "Multistep Matrix Integrators for Real-Time Simulation," to appear in "Advances in Control and Dynamic Systems," Vol. XXXVI, Academic Press, 1990.

5. T.T. Hartley and G.O. Beale, "Integration Operator Design for Real-Time Digital Simulation," IEEE Trans. Ind. Elect., Vol. IE-32, No. 4, pp. 393-398, Nov. 1985.

6. J.A. De Abreu-Garcia, T.T. Hartley, and F. Mossayebi, "On Matrix Integrators for Real-Time Simulation," to appear in IEEE Trans. Ind. Elect., Vol. IE-37, April 1990.

7. PC-MATLAB, The Mathworks Inc., Sherborn, MA (617-653-1415).

8. G.H. Golub and C.F. Van Loan, "Matrix Computations," Johns Hopkins Univ. Press, MD, 1983.

9. C.L. Lawson and R.J. Hanson, "Solving Least Squares Problems," Prentice-Hall Inc., Englewood Cliffs, NJ, 1974.

10. J.P. Landauer, "Real-Time Simulation of the Space Shuttle Main Engine on the Simstar Multiprocessor,' Electronic Associates, Inc., 185 Monmouth Parkway, West Long Branch, NJ 07764.

11. R.W. Hornbeck, "Numerical Methods," Quantum Publishers, New York, 1975.

APPENDIX A. REPORT VERSION CODE

We start with the simulation function modules. The interpolation module 'fgen.for' was constructed out of parts of the study version code, bringing all interpolation to the same level of optimization, and adding some features. The integration module 'integ.for' and balancing loop control module 'change.for' are new.

Additional files '*.def' and '*.log' are included on the report version diskette delivered with this report. The '*.def' files are reference prototypes which can be inserted or viewed when coding, as reminders of the form of the reference. The '*.log' files are copies of referencing lines, which aid in the maintenance of programmer assigned index numbers used in all of the function modules

The remaining report version code sections have similarly named counterparts in the study version code. All code is presented here without additional commentary, outside of report page numbering. For reference, we provide the following table of contents for Appendix A:

```
* This module contains all interpolation routines
*
* General one dimensional interpolation:
*
*  For initializations, see blockdata.for
*                                                                    SSM13800
*     PARAMETER ( NCURVE = *, NWORD = *, NCALL = * )
*     COMMON / fcns /
*    +    nstart(NCURVE), npts(NCURVE), last(NCALL),
*    +    xp(NWORD), a(NWORD), b(NWORD), now
*
      FUNCTION fgset( number )
*****************************************************************
* Loads function points from run.dat file.
* Precomputes coefficients a and b for linear interpolation in the
*     form a + b * x.
* The coefficients go into large arrays, with starting
*     point and length defined.
* Lengths are initialized to 0, meaning 'undefined'.
*****************************************************************
      INCLUDE 'units.com'
      INCLUDE 'fgen.com'
*
      IF ( number > NCURVE ) THEN
         PRINT *, 'Function ', number, ' has too many points.'
         STOP
      ELSEIF ( npts( number ) .GT. 0 ) THEN
         PRINT *, 'Function ', number, ' was previously defined.'
         STOP
      ENDIF
      READ(run,'(//2I5, A )' ) num, n, title
      IF ( num .NE. number ) THEN
         PRINT *, 'Function ', num, ' read, expecting ', number.
         STOP
      ENDIF
      nstart( num ) = now
      npts( num ) = n
      nend = now + n - 1
      READ(run,'(6(/2X,G10.0))') ( xp(i), a1(i), i = now, nend )
      DO 10 i = nend, now + 1, -1
        IF ABS( xp(i) - xp(i-1) ) .LT 1.E-4 ) THEN
            PRINT *, 'X difference too small in function ', num
            STOP
        ENDIF
        slope = ( a1(i) - a1(i-1) ) / ( xp(i) - xp(i-1) )
        a0(i) = a1(i) - xp(i) * slope
        a1(i) = slope
   10 CONTINUE
*                    Initializes fgen's previous interval to first interval
      last( num ) = 2
```

```
      now = nend + 1
      END
*
      FUNCTION fgen( nfcn, ncall, x )
*************************************************************
* Linear interpolation of curve(nfcn) at x.
* Stops with error message if curve is undefined or x out of range.
* Supply a unique index for the interpolated variable in ncall.
* Previous interval index is saved under that index.
* Search for the new interval starts with previous interval.
* Coefficients for each interval are precomputed by fgset.
*************************************************************
      INCLUDE 'fgen.com'
C
      IF ( npts(number) = 0) THEN
        PRINT *, 'Invoked function ', number, ' was not loaded.'
        STOP
      END
      i0 = nstart( number )
      CALL intval( last(number), x, npts(number), xp( i0, nerr )
      IF ( nerr .NE. 0 ) THEN
         IF ( nerr .LT. 0 ) THEN
            PRINT *, 'Value too low for function ', number
         ELSE
            PRINT *, 'Value too high for function ', number
         ENDIF
         STOP
      ENDIF
      itop = i0 + last(number)
      fgen = a(itop) + x * b(itop)
      END
*
*  Two-way linear interpolation:
*
      FUNCTION xylint( x, y, nx, px, ny, py, sx, vdy, table, itop, jtop )
*************************************************************
*   Two-way linear interpolation, with precomputed slopes sx and
*   y differences dy.
*************************************************************
      REAL px(nx), py(ny), sx(nx,ny), vdy(*), table(nx, ny)
*
      ilow = itop - 1
      jlow = jtop - 1
      dx = x - xp(ilow)
      p = table( ilow, jlow ) + sx(itop, jlow) * dx
      q = table( ilow, jtop ) + sx(itop, jtop) * dx                    SSM80300
      xylint = p + ( q - p ) * (y - yp(jlow) ) * vdy(jlow)
      END
```

```
*
      SUBROUTINE intval( itop, x, n, array, below, above )
*************************************************************
*  Locates enclosing interval, searching from last one.
*  Interval is specified by index at higher value.
*  Stops if out of range, with below or above message
*************************************************************
      CHARACTER*(*) below, above
      REAL array(*)
*
      IF( x .GT. array(itop) ) THEN
*                                          Search from there up
        DO 10 k = itop + 1, n
           IF x .LE. array(k) THEN
              itop = k
              GO TO 30
           ENDIF
   10   CONTINUE
*                                          Here, x is above array
        itop = npres
        PRINT *, above
        RETURN
      ELSEIF ( x .LT. array(i - 1)  ) THEN
*                                           Search down from there
        DO 20 k = itop - 2, 1, -1
           IF x .GE. array(k) THEN
              itop = k + 1
              RETURN
           ENDIF
   20   CONTINUE
*                                         Here, x is below array
        PRINT *, below
        STOP
      ENDIF
      END
```

```
*
      SUBROUTINE XYset( nx, xp, ny, yp, table, sx, vdy )
**********************************************************************
* Precomputes slopes sx and reciprocal differences vdy for linear
*    two - way interpolation
**********************************************************************
      REAL xp( nx ), yp( ny ), table( nx, ny )
*
      DO 20 j = 2, ny
         DO 10 i = 2, nx
            sx(i,j) = ( table(i,j) - table(i-1,j) ) /
     +                          ( xp(i) - xp(i-1) )
   10    CONTINUE
         vdy(j) = 1.0 / ( yp(j) - yp(j-1) )
   20 CONTINUE
      DO 30 i = 2, nx
         sx(i,ny) = ( table(i,ny) - table(i-1,ny) ) /
     +                                ( xp(i) - xp(i-1) )
   30 CONTINUE
      END
*
      FUNCTION spline( i, x, a, b, c, d )
**********************************************************************
* Evaluates cubic segment for cubic spline interpolation
* with x(i) < x < x(i+1).
**********************************************************************
*
      REAL a(*), b(*), c(*), d(*)
*
      spline = a(i) + x * ( b(i) + x * ( c(i) + x * d(i) ) )
      END
*
      SUBROUTINE splin0( nx, x, y, a, b, c, d )
**********************************************************************
* Precomputes cubic spline segment coefficients a, b, c, d for spline,
* given x, y coordinates.
* Coded with reference to Hornbeck's Numerical Methods
**********************************************************************
      PARAMETER (maxspl = 21)
      REAL rhs(maxspl), dx(maxspl), gpp(maxspl),
     +     vdx(maxspl), x2(maxspl)
      REAL a(*), b(*), c(*), d(*)
*
* Set up right hand side of equations (4.30)
*
      nxm1 = nx - 1
      DO 10 i = 1, nxm1
        rhs(i) = y(i+1) - y(i)
        dx(i) = x(i+1) - x(i)
        vdx(i) = 1. / dx(i)
```

```
   10 CONTINUE
      DO 20 i = nxm1, 2, -1
        rhs(i) = ( rhs(i) - rhs(i-1) ) / dx(i)**2
   20 CONTINUE
*
*  Solve the tri-diagonal system (4.30) for gpp(i), the
*  second derivative divided by 6,
*  by flowchart of Fig. 6.3, where a(i) = c(i) = 1, b(i) = 4
*
      DO 30 I = nxm1, 3, -1
        rhs(I-1) = ( rhs(I-1) - rhs(I) ) * 0.3333333
   30 CONTINUE
      gpp(2) = rhs(3)
      DO 40 I = 3, nxm1
        gpp(I) = rhs(I) - gpp(I-1)
   40 CONTINUE
*
*  Use the natural cubic spline end conditions of (4.31) and (4.32)
*
      gpp(1) = 0.0
      gpp(nx) = 0.0
*
*  Precompute cubic polynomial coefficients from (4.26 )
*
      DO 50 i = 1, nx
        x2(i) = x(i) ** 2
   50 CONTINUE
      DO 60 i = 1, nxm1
        dg = gpp(i+1) - gpp(i)
        dxg = x(i+1) * gpp(i) - x(i) * gpp(i+1)
        a(i) = ( gpp(i) * x2(i+1) + y(i) ) * x(i+1)
     +       - ( gpp(i+1) * x2(i) + y(i+1) ) * x(i)
     +       - dxg * dx(i)
        b(i) = ( 3.0 * ( gpp(i+1) * x2(i) - gpp(i) * x2(i+1) )
     +           + y(i+1) - y(i) ) * vdx(i) - dg * dx(i)
        c(i) = 3.0 * dxg
        d(i) = dg * vdx(i)
   60 CONTINUE
      END
```

```
*
*  The file 'integ.com' contains:
*
*      DOUBLE PRECISION accum, acctry, xlow, xhigh
*      PARAMETER ( integs = * )
*      COMMON / integ / h(0:1),
*     +      nstart( integs ), limted( integs ),
*     +      accum( integs ), acctry( integs ),
*     +      xlow( integs ), xhigh( integs ),
*     +      fm1( integs ), trfm1( integs )
*
       SUBROUTINE unint0( x, npast )
*************************************************************************
*
*  Initializes any unlimited integrator.
*  Loads double precision accumulator.
*
*************************************************************************
*
       INCLUDE 'integ.com'
*
       accum(npast) = x
       nstart(npast) = 0
       END
*
       SUBROUTINE lmint0( x, npast, botm, top )
*************************************************************************
*
*  Initializes any limited integrator.
*  Loads double precision accumulator and double precision limits.
*
*************************************************************************
*
       INCLUDE 'integ.com'
*
       accum(npast) = x
       xlow(npast) = botm
       xhigh(npast) = top
       nstart(npast) = 0
       END
```

```
*
*     FUNCTION pruEul( rate, nh, npast ) selected unlimited integrator
      FUNCTION pruint( rate, nh, npast )
*****************************************************************
*
*  Explicit Euler as the primary unlimited integrator.
*  Does not use past values. Initialized by uninto.
*
*****************************************************************
*
      DOUBLE PRECISION x
      INCLUDE 'integ.com'
*
      x = accum(npast) + rate * h(nh)
      accum(npast) = x
      pruint = x
*     pruEul = x    reactivate to deselect, but compile Euler.
      END
*
      FUNCTION pruAB2( rate, nh, npast )
*     FUNCTION pruint( rate, nh, npast )  AB2 is not currently selected.
*****************************************************************
*
*  Modified Adams-Bashforth 2nd Order as unlimited primary.
*  Uses one past rate (right hand side f) value.
*  Euler is used once for starting value. Initialized by uninto.
*
*****************************************************************
*
      DOUBLE PRECISION x
      INCLUDE 'integ.com'
*
      IF ( nstart( npast) .EQ. 0) THEN
         x = accum(npast) + rate * h(nh)
         nstart(npast) = 1
      ELSE
        x = accum(npast)
        x = accum(npast) + h(nh) *
     +                      ( 1.6 * rate - 0.6 * fm1(npast) )
      ENDIF
      accum(npast) = x
*     pruint = x          reactivate to select AB2
      pruAB2 = x
      fm1(npast) = rate
      END
```

```
*
*     FUNCTION truEul( rate, nh, npast ) selected unlimited trial integrat
      FUNCTION truint( rate, nh, npast )
*****************************************************************
*
*  Unlimited, trial Euler for use within balancing loops.
*  Requires 'step' to accept trial value.
*  Initialized with unint0.
*
*****************************************************************
*
      DOUBLE PRECISION x
      INCLUDE 'integ.com'
*
      x = accum(npast) + rate * h(nh)
      acctry(npast) = x
      truint = x
*     truEul = x    reactivate to deselect, but still compile truEul.
      END
*
      FUNCTION truAB2( rate, nh, npast )
*     FUNCTION truint( rate, nh, npast ) AB2 is not currently selected
*****************************************************************
*
*  Unlimited trial Modified Adams-Bashforth 2nd Order.
*  Requires 'step' to accept trial values and start itself.
*  Initialized with unint0.
*
*****************************************************************
*
      DOUBLE PRECISION x
      INCLUDE 'integ.com'
*
*****************************************************************
*
      IF ( nstart(npast) .EQ. 0 ) THEN
        x = accum(npast) + rate * h(nh)
      ELSE
        x = accum(npast) + h(nh) *
     +                    ( 1.6 * rate - 0.6 * past(npast) )
      ENDIF
      acctry(npast) = x
      trfm1( npast ) = rate
      truAB2 = x
*     truint = x         reactivate to select AB2
      END
```

```
*
*     FUNCTION prlEul( rate, nh, npast ) selected limited integrator
      FUNCTION prlint( rate, nh, npast )
*************************************************************************
*
*  Explicit Euler as the primary limited integrator.
*  Does not use past values. Initialized by lmint0.
*
*************************************************************************
*
      DOUBLE PRECISION x
      INCLUDE 'integ.com'
*
      x = accum(npast) + rate * h(nh)
      IF ( x .LT. xlow(npast) ) THEN
        x = xlow(npast)
      ELSE IF( x .GT. xhigh(npast) ) THEN
        x = xhigh(npast)
      ENDIF
      accum(npast) = x
      prlint = x
*     prlEul = x    reactivate to deselect, but compile Euler.
      END
*
      FUNCTION prlAB2( rate, nh, npast )
*     FUNCTION prlint( rate, nh, npast ) AB2 is not currently selected.
*************************************************************************
*
*  Modified Adams-Bashforth 2nd Order as the primary limited integrator.
*  Uses a past rate. Initialized by lmint0.
*
*************************************************************************
*
      DOUBLE PRECISION x
      INCLUDE 'integ.com'
*
      IF ( nstart(npast) .EQ. 0 ) THEN
        x = accum(npast) + rate * h(nh)
      ELSE
        x = accum(npast) + h(nh) *
     +                   ( 1.6 * rate - 0.6 * fm1(npast) )
      ENDIF
      IF ( x .LT. xlow(npast) ) THEN
        x = xlow(npast)
        nstart(npast) = 0
      ELSEIF( x .GT. xhigh(npast) ) THEN
        x = xhigh(npast)
        nstart(npast) = 0
      ELSE
        fm1(npast) = rate
```

```
        nstart(npast) = 1
      ENDIF
      accum(npast) = x
*     prlint = x    reactivate to select AB2
      prlAB2 = x
      END
*
*     FUNCTION trlEul( rate, nh, npast ) selected limited trial integrator
      FUNCTION trlint( rate, nh, npast )
*****************************************************************
*
*  Explicit Euler as the trial limited integrator.
*  Does not use past values. Initialized by lmint0.
*
*****************************************************************
*
      DOUBLE PRECISION x
      INCLUDE 'integ.com'
*
      x = accum(npast) + rate * h(nh)
      IF ( x .LT. xlow(npast) ) THEN
        x = xlow(npast)
      ELSEIF( x .GT. xhigh(npast) ) THEN
        x = xhigh(npast)
      ENDIF
      acctry(npast) = x
      trlint = x
*     trlEul = x    reactivate to deselect, but compile Euler.
      END
*
      FUNCTION trlAB2( rate, nh, npast )
*     FUNCTION trlint( rate, nh, npast ) AB2 is not currently selected.
*****************************************************************
*
*  Modified Adams-Bashforth 2nd Order as the trial limited integrator.
*  Uses one past rate value. Initialized by lmint0.
*
*****************************************************************
*
      DOUBLE PRECISION x
      INCLUDE 'integ.com'
*
      IF ( nstart(npast) .EQ. 0 ) THEN
        x = accum(npast) + rate * h(nh)
      ELSE
        x = accum(npast) + h(nh) *
     +                       ( 1.6 * rate - 0.6 * fm1(npast) )
      ENDIF
      IF ( x .LT. xlow(npast) ) THEN
        x = xlow(npast)
```

```
          limted(npast) = .TRUE.
        ELSEIF( x .GT. xhigh(npast) ) THEN
          x = xhigh(npast)
          limted(npast) = .TRUE.
          nstart(npast) = 0
        ELSE
          trfm1(npast) = rate
          limted(npast) = .FALSE.
        ENDIF
        accum(npast) = x
*       trlint = x     reactivate when selecting AB2
        trlAB2 = x
        END
*
*       FUNCTION stpEul( npast )  selected as the trial value acceptor
        FUNCTION step( npast )
*************************************************************************
*
*  Euler step. Accepts trial accumulator as step accumulator.
*  Returns accepted value in single precision
*
*************************************************************************
*
        DOUBLE PRECISION x
        INCLUDE 'integ.com'
*
        x = acctry(npast)
        accum(npast) = x
        step = x
*       stpEul = x   reactivate to deselect, but compile stpEul
        END
*
        FUNCTION stpAB2( npast )
*       FUNCTION step( npast )
*************************************************************************
*
*  Modified Adams-Bashforth 2nd Order step accepts trial accumulator and
*  past rate, and starts itself.
*
*************************************************************************
*
        INCLUDE 'integ.com'
*
        x = acctry(npast)
        accum(npast) = x
        stpAB2 = x
*       step = x          reactivate to select as acceptor of trial values
        fm1(npast) = trfm1(npast)
        IF ( limted(npast) ) THEN
           nstart(npast) = 0
```

```
      ELSE
         nstart(npast) = 1
      ENDIF
      END
*
      FUNCTION trflow( W, Z, R, P, npast )
*****************************************************************
*
*  Trial unlimited flow integration, previously
*
*     FUNCTION FLOW( W, Z, R, T, P).
*
*  Trial integration of flow acceleration of form
*
*          dW/dt = ( pressure - resistance * W**2 )/z
*
C   THIS IS THE FUNCTION TO CALCULATE THE FLOW RATE OF A INCOMPRESSIBLE FLOW
C   ON A GIVEN DUCT WITH KNOWN INERTIA (Z), AND NORMALIZED RESISTANCE
C   (R = RESISTANCE/FLOW DENSITY).
C   INPUT PARAMETERS ARE PREVIOUS FLOW RATE (W), PRESSURE DIFFERENCE (P) AND
C   INTEGRAL TIME INTERVAL (T).
C
C     COMPILER (LINK=IBJ$)                                          SSM14600
*     DOUBLE PRECISION Z1,R1,T1,W1,P1,A,B,C
*     REAL W,Z,R,T,P
C
C                WILL COMPUTE NEGETIVE FLOW
C                     W  =  LAST VALUE OF FLOWRATE
C                     Z  =  INERTIA,L/AG
C                     R  =  RESISTANCE
C                     T  =  DELTA TIME
C                     P  =  DELTA PRESSURE
C
C   UNDER THE GIVEN CONDITION THE FLOW ACCELERATION IS GIVEN BY:
C      DDW=(P - R * (W**2))/Z
C   IF W' IS THE NEXT FLOW VALUE AFTER TIME INTERVAL T THEN
C     DDW = (W' - W)/T
C   AND THE W' CAN BE APPROXIMATED BY THE FOLLOWING STEADY STATE CONDITION
C     (W' -W)/T = (P - R * (W'**2))/Z
C   W' IS THE ONLY UNKNOWN IN THE EQUATION.
C
C                DUMMY VARIABLES                                    SSM14700
*     Z1 =DBLE(Z)
*     R1 = ABS(DBLE(R))
*     T1 =DBLE(T)          repeatedly converts an unchanging variable
*     W1 =DBLE(W)
*     P1 =DBLE(P)
*     A = Z1/2.D0/R1/T1   requires two unnecessary divisions
*     B = A * 2.D0*W1
*     C = P1/R1
```

```
C                 COMPUTE
*     FLOW=SNGL(DSIGN(-A+DSQRT(A**2+DABS(B+C)), B+C))                SSM14800
*     IF(R.LT.0.0 ) RETURN
*     FLOW = AMAX1(0.0 ,FLOW)
*     RETURN
*
*    The above use of implicit Euler, without any iterative correction, is
*    not justified, and is implemented in a costly manner.
*
*    The replacement uses the selected trial integrator, in limited and
*    unlimited forms. The simulation step h = dt is always used.
*    The flow rate is computed in single precision, using the previous W.
*    Because of the possible limit, the integrator is
*    initialized by lminto, with a practically infinite upper limit.
*
*    A call to step is required to accept trial values.
*
*************************************************************************
*
      IF ( R .GE. 0.0 ) THEN
         trflow = trlint( (P - R * W**2)/Z, 0, npast )
      ELSE
         trflow = truint( (P - R * W**2)/Z, 0, npast )
      ENDIF
      END
*
      FUNCTION prflow( W, Z, R, P, npast )
*************************************************************************
*
*  Primary flow integration, for use outside of balancing loops.
*  Uses the selected primary integrator in limited and unlimited forms.
*  Initialized by lminto, with a practically infinite upper limit.
*
*************************************************************************
*
      IF ( R .GE. 0.0 ) THEN
         prflow = prlint( (P - R * W**2)/Z, 0, npast )
      ELSE
         prflow = pruint( (P - R * W**2)/Z, 0, npast )
      ENDIF
      END
```

```
*     change.com contains:
*
*      PARAMETER ( maxchg = 16 )
*      COMMON / change / chgtol, rxfact( maxchg ), change( maxchg )
*
       FUNCTION relax( okloop, xnew, xold, ix )
*************************************************************************
*  Applies relaxation factor to monitored variable changes.
*  Counts iterations in which relative change is over tolerance.
*  Returns relaxed change value.
*  For converging values ix, increase rxfact(ix) above 1.0 to speed up
*  convergence. Keep it less than 2.0.
*  For slow converging values ix, try decreasing rxfact(ix) below 1.0.
*************************************************************************
*
       LOGICAL ok, okloop
       INCLUDE 'change.com'
       INCLUDE 'out.com'
*
       diff = xnew - xold
       change(ix) = diff / xold
       ok = ABS( change(ix) ) .LE. chgtol
       okloop = okloop .AND. ok
       IF ( .NOT. ok ) its(ix) = its(ix) + 1
       relax = xold + diff * rxfact(ix)
       END
*
       SUBROUTINE its0
*************************************************************************
*
*    Initialize balancing iteration counts
*
*************************************************************************
*
       INCLUDE 'out.com'
       INCLUDE 'change.com'
*
       DO 10, I = 1, maxchg
         its(I) = 0
    10 CONTINUE
       END
```

```
*
      SUBROUTINE wrchg( iter, nfirst, nlast, heading )
*************************************************************************
*
*   Writes change monitor counts and last changes to event file
*   Allows no more than one full report per 100 iterations, but
*   reports the number of non-converging iterations of any kind
*   not reported.
*
*************************************************************************
*
      CHARACTER*(*) heading
      INCLUDE 'change.com'
      DATA / last, nskipd / -100, 0 /
      SAVE last, nskipd
*
*************************************************************************
*
      IF ( iter - last .GE. 100 ) THEN
        WRITE( event,
     +   '( A,      I6 /     A )'  )
     +    heading, iter, ' item  iterations over  last change'
        DO 10, I = nfirst, nlast
          WRITE( event, '( I3, 10X,   I3,   8X,   E12.4 )' )
     +                    I,      its(I),     change(I)
   10   CONTINUE
        last = iter
        IF ( nskipd .GT. 0 ) THEN
          WRITE( event, '(I4,           A )' )
     +                  nskipd, ' nonconverged iterations skipped'
          nskipd = 0
        ENDIF
      ELSE
        nskipd = nskipd + 1
      ENDIF
      END
```

```
*  The x to the y module, supporting degrees of normilization.
*  Implements x to y through table lookup and linear
*  interpolation
*
      FUNCTION Xn10th( Xnorm, N10th )
*****************************************************************
*  Table lookup of ( normed X ) ** (0.1*N10th), for 0 < N10th < 10,
*  where 0.0 < normed X < 1.0
*****************************************************************
      REAL X10(0:10, 0:18)
      DATA ( X10(0, J), J = 0, 18)  / 19 * 0.0 /
      DATA ( X10(1, J), J = 0, 18) /
     +   0.0, .6780, .7254, .7551, .7770, .7945, .8090, .8216, .8326,
     + .8424, .8514, .8870, .9126, .9333, .9503, .9650, .9780, .9896,
     + 1.0 /
      DATA ( X10(2, J), J = 0, 18) /
     +   0.0, .4596, .5261, .5702, .6037, .6311, .6545, .6750, .6932,
     + .7097, .7248, .7867, .8329, .8708, .9031, .9313, .9564, .9792,
     + 1.0 /
      DATA ( X10(3, J), J = 0, 18) /
     +   0.0, .3113, .3816, .4305, .4690, .5013, .5295, .5545, .5772,
     + .5979, .6171, .6976, .7602, .8126, .8582, .8987, .9354, .9690,
     +   1.0 /
      DATA ( X10(4, J), J = 0, 18) /
     +   0.0, .2108, .2767, .3250, .3643, .3983, .4284, .4555, .4805,
     + .5037, .5254, .6186, .6937, .7582, .8154, .8672, .9147, .9589,
     +   1.0 /
      DATA ( X10(5, J), J = 0, 18) /
     +   0.0, .1427, .2006, .2453, .2830, .3164, .3465, .3742, .4001,
     + .4243, .4473, .5485, .6330, .7075, .7748, .8369, .8946, .9488,
     +   1.0 /
      DATA ( X10(6, J), J = 0, 18) /
     +   0.0, .0965, .1454, .1851, .2199, .2513, .2083, .3075, .3331,
     + .3575, .3808, .4862, .5775, .6601, .7363, .8076, .8748, .9390,
     +   1.0 /
      DATA ( X10(7, J), J = 0, 18) /
     +  0.0, .06519, .1053, .1397, .1708, .1996, .2268, .2526, .2773,
     + .3011, .3242, .4310, .5269, .6158, .6996, .7792, .8555, .9290,
     +   1.0 /
      DATA ( X10(8, J), J = 0, 18) /
     +  0.0, .04401, .0762, .1054, .1326, .1585, .1834, .2075, .2309,
     + .2537, .2760, .3820, .4807, .5746, .6647, .7519, .8366, .9193,
     +   1.0 /
      DATA ( X10(9, J), J = 0, 18) /
     + 0.0,.02969, .05525, .07953, .1030, .1259, .1484, .1704, .1922,
     + .2137, .2350, .3385, .4385, .5360, .6315, .7255, .8181, .9096,
     +   1.0 /
      DATA ( X10(10, J), J = 0, 18) /
     +   0.0,  0.02,  0.04,  0.06,  0.08,  0.10,  0.12,  0.14,  0.16,
     +  0.18,   0.2,   0.3,   0.4,   0.5,   0.6,   0.7,   0.8,   0.9,
```

```
     +   1.0 /
*****************************************************************************
      IF ( Xnorm .GE. 0.2 ) THEN
        U = Xnorm * 0.1
        V = AINT( U )
        I = INT( V ) + 8
      ELSE
        U = Xnorm * 0.02
        V = AINT( U )
        I = V
      ENDIF
      Xn10th = X10( N10th, I ) + (U - V) * X10( N10th, I + 1)
      END
*
      FUNCTION X10th( X, N10th )
*****************************************************************************
*  ( Unnormalized X ) ** (0.1 * N10th), where 0 < N10th < 10
*  Can be optimized further by substituting Xn10th code for both
*  function references to Xn10th, eliminating overhead of call sequence.
*****************************************************************************
      IF ( X .GT. 1.0 ) THEN
        X10th = 1.0 / Xn10th( 1.0 / X, N10th )
      ELSE
        X10th = Xn10th( X, N10th )
      ENDIF
      END
*
      FUNCTION XntoYn( Xnorm, Ynorm )
*****************************************************************************
*  ( normalized X ) ** ( normalized Y ), where
*     0.0 < normalized X, Y < 1.0
*****************************************************************************
      UX = Xnorm * 0.1
      VX = AINT( UX )
      J = VX
      UX = UX - VX
      UY = Ynorm * 0.1
      VY = AINT( UY )
      I = VY
      XnYlo = X10(I, J) + UX * X10(I, J + 1)
      XnYhi = X10(I + 1, J) + UX * X10(I + 1, J + 1)
      XntoYn = XnYlo + (UY - VY) * XnYhi
      END
*
      FUNCTION XtoY( X, Y )
*****************************************************************************
*  ( unnormalized X ) ** ( unnormalized, positive Y )
*****************************************************************************
      IF ( Y .GT. 1.0 ) THEN
        UY = AINT( Y )
```

```
  N = UY
  VY = Y - UY * 10.
  IF ( X .GT. 1.0 ) THEN
    XtoY = 1.0 / XntoYn( 1.0 / X, VY )
  ELSE
    XtoY = XntoYn( X, VY )
  ENDIF
  XtoY = XtoY * X ** N
ELSE
  IF ( X .GT. 1.0 ) THEN
    XtoY = 1.0 / XntoYn( 1.0 / X, Y )
  ELSE
    XtoY = XntoYn( X, Y )
  ENDIF
ENDIF
END
```

```
*   'output.com' contains:
*       PARAMETER ( nints = 16, nreals = 551 )
*       INTEGER inidx, rlidx
*       COMMON /output/  ninsel, nrlsel,
*      +                 inidx( nints ), rlidx( nreals )
*
        SUBROUTINE writeset( time, runid )
************************************************************
*   Reads the set of write variables from the 'select.out' file
*   Saves set numbers and headings for collection of data during
*   the simulation run
*   Opens the output variable file, and writes on it:
*       - an 80-character identifying header from 'ssme.run'
*       - an 80-character identifying header from 'select.out'
*       - the time interval between data records
*       - the numbers of selected integer and real output variables
*       - indexes of selected integer and real output variables
*       - names of selected integer and real output variables
************************************************************
*
        INCLUDE 'units.com'
        INCLUDE 'output.com'
*
        CHARACTER*8 name
        CHARACTER*1 ok
*
        OPEN( UNIT = set, FILE = 'select.out', STATUS = 'OLD',
     +        ERR = 1 )
        GO TO 2
    1   PRINT *, 'The ''select.out'' file was not found.'
        STOP
*
    2 OPEN(UNIT = out, FILE = 'ssme.out', STATUS = 'NEW',
     +        ERR = 3 )
        GO TO 5
    3   PRINT *,
     +    'Is it OK to overwrite the ''ssme.out'' file? (y/n) '
        READ *, ok
        IF( ok .EQ. 'y' .OR. ok .EQ. 'Y' ) THEN
           OPEN(UNIT = out, FILE = 'ssme.out', STATUS = 'OLD' )
        ELSE
           STOP
        ENDIF
*
    5 WRITE( out, '(A)' ) runid
      READ( set, '(A)' ) slctid
      WRITE( out, '(A)' ) slctid
*
      WRITE(out) time
*
```

```
*  Count integer variables and save integer indexes and headings
*
      ninsel = 0
      DO 10, i = 1, nints
        READ( set, '( I1, 1X, A8 )' ) iyes, name
        IF ( iyes .EQ. 1 ) THEN
          ninsel = ninsel + 1
          inidx( ninsel ) = i
          inhead( ninsel ) = name
        ENDIF
   10 CONTINUE
*
*  Write unformatted: # integer outputs, and indexes and headings
*
      WRITE( out ) ninsel
      IF ( ninsel .GT. 0 ) THEN
        WRITE( out ) ( inidx( i ), i = 1, ninsel )
        WRITE( out ) ( inhead( i ), i = 1, ninsel )
      ENDIF
*
*  Count real variables and save real indexes and headings
*
      nrlsel = 0
      DO 20, i = 1, nreals
        READ( set, '( I1, 1X, A8 )' ) iyes, name
        IF ( iyes .EQ. 1 ) THEN
          nrlsel = nrlsel + 1
          rlidx( nrlsel ) = i
          rlhead( nrlsel ) = name
        ENDIF
   20 CONTINUE
*
*  Write unformatted: # real outputs, and indexes and headings
*
      WRITE( out ) nrlsel
      IF ( nrlsel .GT. 0 ) THEN
        WRITE( out ) ( rlidx( i ), i = 1, nrlsel )
        WRITE( out ) ( rlhead( i ), i = 1, nrlsel )
      ENDIF
      END
*
      SUBROUTINE writer( STIME )
*************************************************************
*  Collects and writes selected output data for one output interval
*  The data is written unformatted.
*************************************************************
*
      DIMENSION idata( nints ), rdata( nreals )
*
      INCLUDE 'output.com'
```

```
      INCLUDE 'units.com'
      COMMON /outvar/ ivar( nints ), rvar( nreals )
*
      DO 10 i = 1, ninsel
        idata(i) = ivar( inidx(i) )
   10 CONTINUE
      WRITE( out ) STIME, (idata(i), i = 1, ninsel)
*
      DO 20 i = 1, rlnsel
        rdata(i) = rvar( rlidx(i) )
   20 CONTINUE
      WRITE( out ) (rdata(i), i = 1, nrlsel)
      END
```

The next section presents data COMMON blocks. The BLOCK DATA module initializes labeled COMMOM blocks.

```
      BLOCK DATA
*****************************************************************
*
* Labeled COMMON blocks are defined in files '*.com'
* This module initializes labeled COMMON
* COMMON blocks have been reorganized to the following exent:
*   1) output variables of all modules have been moved to a single
*      labeled COMMON block 'outvars' in 'outcom'
*   2) other variables recognized as not shared between modules were
*      removed from COMMON blocks
*
*****************************************************************
*
      COMMON /FUEL/ fuels(59)
      DATA fuels / 59 * 0. /
      COMMON /hgas/ gasses(55)
      DATA gasses / 55 * 0. /
      COMMON /oxid/ oxids(71)
      DATA oxids / 71 * 0. /
      COMMON /oxidil/ oxdils(13)
      DATA oxdils / 13 * 0. /
      COMMON / contrl / rcons(55), icons(8)
      DATA rcons, icons / 55 * 0., 8 * 0 /
      COMMOM / balc / balcs(73)
      DATA balcs / 73 * 0. /
      COMMOM / valves / vals(115)
      DATA vals / 115 * 0. /
      COMMOM / pogo / pogos(5)
      DATA pogos / 5 * 0. /
      COMMON /outvar / ITS(16), outs(564)
      DATA outs / 564 * 0. /
*
      INCLUDE 'fgen.com'
      DATA /now, (npts(i), i = 1, NCURVE) /1, NCURVE * 0 /
*
      INCLUDE 'integ.com'
      DATA nstart, limted / integs * 0, integs * .FALSE. /
*
      END
*
      SUBROUTINE IniCom
*****************************************************************
*
*  Initializes blank common not otherwise initialized.
*
```

```
*****************************************************************
*
*   out.com
*
      COMMON  DT, STIME, Reals(28)
      DO 10 I = 1, 28
        Reals(I) = 0.
   10 CONTINUE
      END
      COMMON/outvar/
     + ITS(16),                                                      16
     + ABMOV,       ACCV,        AFPOV,       AGC,         AHPV,     5
     + AIN,         AMFV,        AMOV,        AOPOV,       APV,     10
     + ATH,         DDW1,        DDX,         DPFPAS,      DPOP1,   15
     + DPOPAS,      DQHEAT,      DPHGMF,      DPHGMO,      DTPSTH,  20
     + DTPSTL,      DW(13),      DW1(3),      DW2(3),      DW10P,   41
     + DW3P,        DW4P,        DW7P,        DW8P,        DWC,     46
     + DWCOD,       DWCOD1,      DWCOD2,      DWF,         DWFBPV,  51
     + DWFI,        DWIF(6),     DWFNP,       DWFPF,       DWFPO,   61
     + DWFPOI,      DWFPR1,      DWFPR2,      DWFT1,       DWFT2,   66
     + DWG,         DWGAS,       DWGO,        DWGOP,       DWH,     71
     + DWHO,        DWHOP,       DWLO,        DWMC,        DWMCP,   76
     + DWMOV,       DWO,         DWOE2,       DWOE3,       DWFIG(3),83
     + DWFN,        DWNIG(3),    DWFNBP,      DWOI,        DWOIG(3),92
     + DWOIN,       DWOPF,       DWOP1,       DWOP2,       DWOP2C,  97
     + DWOP3,       DWOP3C,      DWOPO,       DWOPOI,      DWOPR,  102
     + DWOPV,       DWOS,        DWOT1,       DWOT1D,      DWOT1I, 107
     + DWOT2,       DWOTJ,       DWP,         DWPFI,       DWPOI,  112
     + DWQNCH,      DWRE,        DWSFS,       ELCOM,       ELENT,  117
     + ELFC,        ELFCM,       ELFFI,       ELFFP,       ELFFPM, 122
     + ELFOP,       ELFOPM,      ELOIN,       EMRC,        EMRCR,  127
     + EMRE,        EMRF,        EMRIG(3),    EMRFPO,      EMROPO, 134
     + EPC,         ERROR,       ETAFT1,      ETAFT2,      ETAOT2, 139
     + FAC,         FCOMP,       FHEOD1,      FHEOD2,      FHEOI2, 144
     + FHEOP1,      FHEOT1,      FR,          GF1,         GF2,    149
     + H(13),       H3,          H10P,        H7P,         H8P,    166
     + HCAVP1,      HCAVP2,      HCAVP3,      HOD2,        HOD3,   171
     + HOI2,        HOS,         HOS1,        HOT1,        HP,     176
     + HT,          OD,                                            178
     + P(13),       P1,          P10P,        P1NPSH,      P2,     195
     + P2NPSH,      P3NPSH,      P4(3),       P7P,         P8P,    202
     + PANS,        PCFPO,       PCIE,        PCIG(3),     PCNS,   209
     + PCOPO,       PCTPERT,     PD,          PFI,         PFP,    214
     + PFP1,        PFPD1,       PFPOI,       PFPOV,       PFS,    219
     + PFS2,        PFIS,        PFT1,        PFT1D,       PG,     224
     + PGCI,        PGCO,        PHES,        PIF(6),      PIL(12) 245
     + PMOV,        POD1,        POD1M,       POD2,        POD3,   250
     + POT1,        POI2,        POINJ,       POINVP,      POJ,    255
     + POP,         POPOI,       POPOV,       POPRG,       POPVDN, 260
     + POS,         PPURG,      PRES,         PRFT2,               264
```


C-2

```
     + PROT1,       PROT2,       PT,          PTOL,                              268
     + QB(13),      QBKFLO,      QF,          QINT,        QIN1(13),             297
     + QOUT1(13),   QOUT2(13),                                                   323
     + R1(3),       R2(3),       R3(3),                                         332
     + RBPV,        RCCV,        RCOM,                                          335
     + RFPOV,       RHO(13),     RHO10P,      RHO7P,       RHO8P,                352
     + RHOD1,       RHOD2,       RHOGOX,      RHOHE,       RHOHI,                357
     + RHOI2,       RHOLO,       RHOMOV,      RHOOS,       RHOP,                 362
     + RHOP1,       RHOP3,       RHOREC,      RHOT,        RHOT1,                367
     + RJTPVD,      RMFV,        RMOV,        ROIN,        ROPOV,                372
     + RR(13),      RVALVE,                                                      386
     + SCUT,        SF1,         SF2,         SO1,         SO2,                  391
     + SRATE,       STOPT,       SU(13),       SU10P,                           407
     + SU7P,        SU8P,        SUCOD1,      SUOD1,       SUOD2,                412
     + SUOI2,       SUOIN,       SUOT1,       T(13),                             428
     + TCAVP1,      TCAVP2,      TCAVP3,      TEMP,        TFI,                  433
     + TFIS,        TFP,         TFP1,        TFPM,        TFT1D,                438
     + TFT2D,       TFT2DI,      TGAS,        THETA(5),    THETA1(5),            451
     + THETA2(5),   TOP,         TOPM,         TOT2D,                           459
     + TOT2DI,      TP(5),       TPERT,       TRQFP1,      TRQFP2,               468
     + TRQFT1,      TRQFT2,      TRQOP1,      TRQOP2,      TRQOP3,               473
     + TRQOT1,      TRQOT2,      TSAT,        TSTART,      TSTOP,                478
     + TW,          TW1(13),     TW2(13),                                       505
     + U,           UCFT2,       UCOT2,       UG,          UOD1,                 510
     + UOD2,        UOI2,        UOIN,        UOT1,        UP,                   515
     + VG,          VHEOD1,      VHEOD2,      VHEOI2,      VHEOT1,               520
     + VL,          VOLPV,                                                       522
     + WFPOI,       WGOX,        WHE,         WHECD1,      WHECD2,               527
     + WHEOD1,      WHEOD2,      WHEOI2,      WHEOT1,      WLOX,                 532
     + WOCOM,       WOD1,        WOD2,        WOI2,        WOIN,                 537
     + WOPOI,       WOT1,        WT1BK,       WT2BK,       WTASI,                542
     + WTIGN,       X1,          X2,          X3,          XCCCV,                547
     + XCFPOV,      XCOPOV,      XFPOV,       XGC,         XHPV,                 552
     + XCMFV,       XCMOV,       XMFV,        XMOV,        XOPOV,                557
     + YCCCV,       YCFPOV,      YCMFV,       YCMOV,       YCOPOV,               562
     + ZCOM,        ZOIN                                                         564

    'blank.com':

*                par
      COMMON    DT, STIME,
                          par   par
     +   QIN2(13), AOPTO, RFT1V, PA, RHOO3, PFPD, DWCCV,
*                                              par  par          par
     +   DWOPC, RHOOP2, RHOOP3, RFBV, ROBV, DPR, DPL, PINMC, TCUT
```

```
      'fuel.com':

        COMMON /FUEL/
*           par     par   par:4,5,6,12  par
     +    VOL(13), R(13), AHT1(13), AHT2(13), PFP1R, PFP2R, TFT1
*
     +    PRFT1, DWFT2C, TTI, UCFT1

      'hgas.com'

        COMMON /HGAS/ CFOI,  RFPO, WFPF, WFPO,
*
     +  TFPC, CPFP, GAMFP, EMWFP, RGCFP, WTFP, WFP,
*
     +  WOPOI,  COOI, ROPO, WOPF, WOPO,  TOPC, CPOP,
*
     +  GAMOP, EMWOP, RGCOP, WTOP, WOP,
*
     +  WTOT1, WDUM, TOT1, POT1A,  TOT1D, WTFI, GAMFI,
*
     +  WFIF,  WFIO, RGCFI, WCO, WC, WTC, TC, WCF,  RGCC, GAMC,
*          par    par    par      par
     +   WOPOV, WFPOV, RFPIGB, ROPIGB, RHOOTF,
*
     +  RHOFTF, RHOFI, DWFTF, PX, DWX, PXF, DWXF,
*
     +  DWACV, DWPFS, DWBAF

      'oxid.com':

*
        COMMON /OXID/  DPOP2, DPOP3, PRIMOI,
*
     +  WOI,  RHOOP1, POP1R, POP2R, POP2A,
*                          par       par      par
     +  POP3R,  DDWOS, QO, TOS, TOD2, WOV, H1, H2,
*         par   par
     +  ZFPO, ZOPO, U1, U2, U3, HLPT1, HIOP2, H3I,
*
     +  ULPOT, HLPOTD, TOD1, TOI2, TOD3,
*
     +  UOD1, WOD1, RHOD1, HOD1,
*
     +  UOI2, SUOI2, RHOI2, UOD2, UOT1, WOT1, ZCOM, RCOM,
*
     +  ROIN, ZOIN, RJTPVD, TCAVP3,
*
     +  UCOD1, WCOD1, RHOCD1, HCOD1, PCOD1,
*
```

```
     +  UCOD2, WCOD2, SUCOD2, RHOCD2, HCOD2, PCOD2,
*
     +  DWFPR2, DWFPTV, DWFPTI, PFPOT, POPRG, PFPRG, DWFPVI, DWFPI,
*
     +  DWOPR2, DWOPTV, DWOPTI, DWOPTA, POPOT, DWOPVI, DWOPR
*
      COMMON/OXIDIL/ DWIL(12), RHOP2

    'contrl.com':

*
      COMMON /CONTRL/  TIMEVC, TIMEPR, TIMECP, TIMETR, TIMFME,
*
     + TIMFMC, TIMMRF, TIMFMA, TIMELM,  EMRGC,
*
     + DXFPOV, EPCGC, DXOPOV, XCRV, PCOPOI, TO, DXMFV, XCCV,    DXCCV,
*      par      par     par     par     par     par
     + TPA,    PCMALF, ABCCV,  ABMFV,  ABOPO,  ABFPO,  RHOH,
*
     + DTMC,     FRADS,  ORADS,  RHO0,
*
     + PIPF,   PIPO,  QF0,    Q0,
*               par       par
     + XCMOVC, DTFMRA,   TSMFV, RESET,  XOPLIM,
*       par                    par
     +  PNOISE, PFRNZ, PMRNZ, TOPEN, FRFZ, EMRFZ,
*
     +  POPVNZ, PFPVNZ, PMOVNZ, PMFVNZ, PCCVNZ,
*
     + FZOPV, FZFPV, FZMOV, FZMFV, FZCCV,
*
     + NCF,    NCO,    IFIND,  IOIND,  KOUNTF, KOUNTO,
*
     +  MODETST, IPFLAG

  'valves.com ':

      COMMON/VALVES/
*          out
     +   DTHETA(5), ESAC(5), DESA(5), ESA(5), DESV(5), ESV(5),
*
     +    VS(5), EMF(5), DVM(5), VM(5),
*
     +   THET1L(5), THET2L(5),
*
     +   ISTIC(5), IHYS(5), VR(5), DTHETL(5),
*                    par    par     par     par    par
     +   AD(5), BD(5), CS(5), CPS(5), DSS(5), ESS(5), VPD(5)

  'igni.com':
```

```
      COMMON /IGNI/ TCIG(3), ELFIG(3), DW3(3), TF(3), PFU(3)
```

'balc.com':

```
*
      COMMON/BALC/ AHT14, AHT15, AHT16, AHT112,
*       par     par     par     par     par     par     par
     + CDPFP1, CTQFP1, CDPFP2, CTQFP2, CTQFT1, CDPOP1, CTQOP1,
*       par     par     par     par             par     par     par
     + CDPOP2, CTQOP2, CDPOP3, CTQOP3, CTQOT1, FT2S, AFT2, CTQFT2,
*       par     par     par     par     par     par     par     par     par
     + EOT2S, AOT2, CTQOT2, AFI, EFFCM, ACN, THRSTC, AHTC4, AHTC5,
*       par     par     par     par   par:2,3 par     par     par     par
     + AHTC6, DMOT2, DMFT1, DMFT2, CP(5), ANOT1, BNOT1, CNOT1, AOT1,
*       par
     + BOT1, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13,
*       par     par     par     par     par     par     par     par     par
     + RFCOD, RFMCF, RFMCO, RACV, RBAF, RPFS, RSFS, RFPFI, ROPFI,
*       par   par  par   par     par     par     par     par     par
     + RITN, RMCI, ROS, RFPOI, ROPOI, RFPOL, ROPOL, RFT2C, ROP2C,
*       par   par     par     par     par     par     par
     + ROI, ROCOD, RMOVL, ROP3C, ROT1F, QHT412, TFACT
```

'pogo.com'

```
      COMMON /POGO/  P2, RHOREC, TGAS, XGC, XHPV
```

```
      PROGRAM SSME
*************************************************************************
*
*  This version of the Space Shuttle Main Engine Simulation has
*  been written to provide an efficient vehicle for the study of
*  life-extending modes of operation through knowledge-based control.
C
C  PROGRAM FLOW:
*    Read Run Parameters from rundata file designated in the command line
*    IF restart requested THEN
*       Initialize run data from restart file
*    ELSE
*       Initialize for normal zero start
*    ENDIF
*    REPEAT
C       T <- T + DT
*       IF perturbing THEN
*          add perturbations(T)
*       ENDIF
C       CALL FUELF() FOR FUEL FLOW CALCULATION
C       CALL OXIDF() FOR OXID FLOW CALCULATION
C       CALL HOTGAS() FOR HOT GAS CALCULATION
C       CALL CNTROL() FOR CONTROLLER SIMULATION
C       CALL VALDYM() FOR VALVE DYNAMIC SIMULATION
C       Calculate closed loop variables
*       IF T = output time THEN
*          Write archival and display output
*       ENDIF
*    UNTIL T = stop time
*    IF resume desired THEN
*       Generate resume file
*    ENDIF
C  END OF SIMULATION
C
      INCLUDE 'change.com'
      Call IniCom0
*
*** Removed "no effect" initializations.
*
*
*** Removed commented out calls to ERRSET
C
C   FOLLOWING FILES ARE USED FOR SIMULATION INPUTS AND OUTPUTS
C     DTMINP.DAT:  MAIN INPUT FILE FOR BASIC PARAMETERS
C     NZR100.DAT:  RESTART AT 100% STEADY STATE
C     START4.DAT:  PATCH FILE USING "NAMELIST" READING
C     OUT2.DAT:    MAIN OUTPUT FILE
C     PERTINP.DAT: PERTURBATION SERIES INPUT FILE
C     RESULTS.DAT: QUICK GLANCE OUTPUT FILE FOR PCIE AND MIX-RATIO
C     STATE.DAT:   OUTPUT FILE OF SELECTED IMPORTANT VARIABLES
```

```
C     PLOT2.DAT:   PLOT FILE FOR DEFINED PLOTTER
C
* Assignments of unit numbers are system dependent. Assignments are
* made global by Common /units/
      run = 2
      dat = 3
      prt = 4
      str = 6
      res = 8
      out = 9
      dsp = 1
*
* Input/output files are specified on the command line, when possible.
* For Fortrans giving no access to the command line, use pre-assigned
* file names:
     rundat = 'ssme.run'
     functs = 'ssme.dat'
     output = 'ssme.out'
     pertrb = 'ssme.ptb'
     restart = 'ssme.rst'
     atstop = 'ssme.end'
*
* HP-UX command line:   ssme rundat output [ pertrb ] [ restart ] [resume]
*     CALL getarg( 2, rundat )
*     CALL getarg( 3, output ), etc.
*
      OPEN( UNIT = run, FILE= rundat, STATUS='OLD')
      OPEN( UNIT = dat, FILE= functs, STATUS='OLD')
C
      read( run,'(A // 2X, 6(2X,L12) ) title, restrt, resume, pertb
*               loading blank common:
     +            DT,     DPR,    DPL,    DPUN,   TPUN,  TSTOP,
     +            TPA,    PCMALF, DTVC,   DTPR,   DTCVP, DTTR,      SSM42200
     +            DTFMRE, DTFMC,  DTMRFC, DTFMRA, DTMCX, DTLM
*
      READ( run, '(//2X, 2I12, 4F12.4)' )
     +            NONZRO, MODETST, PCTPERT, TOPEN, TPERT, DTPERT    SSM42240
*
      READ( run, 30) TSTART,TPRINT2,TPLOT2

*  Relaxation factors for energy balance convergence
*
      READ( run, 30) rxfact

   30 FORMAT(//2X,6(G10.4,2X))
*                                    loading BALC common:
      READ(run, 30)
     +   AHT1(4), AHT1(5), AHT1(6), AHT1(12), CDPFP1, CTQFP1,
     +   CDPFP2,  CTQFP2,  CTQFT1,  CDPOP1,   CTQOP1  CDPOP2
     +   CTQOP2,  CDPOP3,  CTQOP3,  CTQOT12,  FT2S,   AFT2,
```

```
     +   CTQFT2,  EOT2S,   AOT2,    CTQOT2,   AFI,    EFFCM,
     +   ACN,     THRSTC,  AHTC4,   AHTC5,    AHTC6,  ABMOV,      SSM42300
     +   ABOPO,   ABFPO,   ABCCV,   ABMFV,    DMOT2,  DMFT1,
     +   DMFT2,   CP(2),   CP(3),   ANOT1,    BNOT1,  CNOT1,
     +   AOT1,    BOT1,    R(1),    R(3),     R(4),   R(5),
     +   ( R(i), i = 6,11 ),
     +   R(12)    R(13),   RFCOD,   RFMCF,    RFMCO,  RACV,
     +   RBAF,    RPFS,    RSFS,    RFPFI,    ROPFI,  RITN,
     +   RMCI,    RFT1V,   ROS,     RFPOI,    ROPOI,  RFPOL,
     +   ROPOL,   RFT2C,   ROP2C,   ROI,      ROCOD,  RMOVL,
     +   ROP3C,   ROT1F,   QHT412,  TFACT
*
      READ(run,30)TLOW,THIGH,(TPRINC(I),I=1,6)
      READ(run,'(//2X,3I12)') NOUTD,IREDAT,ISAVE
      IF pertb THEN
        OPEN( UNIT = prt, FILE = pertrb ,STATUS='OLD')
      ENDIF
      IF restrt THEN
         OPEN( UNIT = str, FILE = restart, STATUS='OLD')
      ENDIF
      IF resume THEN
         OPEN( UNIT = res, FILE= atstop, STATUS='NEW')
      ENDIF
*
* See perturb.for for perturbation code
* All NAMELIST input has been eliminated.
* Instead, edit the input parameter file ssme.run
* All initialization echo output is gone.
* Instead, save a copy of ssme.run as documentation
*
      CALL fgset(24, nerr)
      PA = FGEN(24, 0.0)
      RBPV=1.0E+30
      TTEMP=AMIN1(2.3,TSTOP)                                    SSM43100
      TIME=0.0
      TIMEVC=DTVC
      TIMEPR=DTPR
      TIMECP=DTCVP
      TIMETR=DTTR
      TIMFME=DTFMRE
      TIMFMC=DTFMC
      TIMMRF=DTMRFC
      TIMFMA=DTFMRA
      TIMELM=DTLM                                               SSM43200
C
C   Initialize subsystems
C
      CALL FUELF0
      PRINT *, 'Fuel flow parameters loaded.'
      CALL OXIDF0
```

```
      PRINT *, 'Oxygen flow parameters loaded.'
*     DWFPB=DWFPF+DWOPF                                          SSM43300
      CALL HOTGAS0
      PRINT *, 'Hotgas parameters loaded.'
      CALL CNTROL0
      PRINT *, 'Control parameters loaded.'                      SSM43400
      CALL VALDYM0
      PRINT *, 'Valve dynamics parameters loaded.'
      CALL TRBTRQ(0.,0.,0.,0.,0.,1,1,0.,0.,0.)
      CALL fgset( 26 )
      CALL fgset( 27 )
      CALL fgset( 40 )
*
*** See 'nonzro.for'  for restart code
*
*  Identify output variables and ready file to save output
*
      CALL writeset( runID )
*
*  Simulation loop
*
      istop = tstop / dt
      isec = 1.0 / dt + .1
*
      DO 400 itime = 1, istop
         STIME = float( itime ) * DT
         PA = FGEN(24, 2, STIME)
         CALL FUELF
         CALL OXIDF
*        DWFPB=DWFPF+DWOPF                                       SSM45900
         CALL HOTGAS
         CALL CNTROL
         CALL VALDYM
*
*  Open loop output was removed. Use offline output programs instead.
*
*  At the print interval, a multiple of the solution interval,
*  write selected data, unformattted.
*
       IF ( MOD( itime, iwrite ) .EQ. 0 ) THEN
         CALL writer( STIME )
       ENDIF
*
*  Write one line per second to display.
*
       IF ( MOD( itime, isec ) .EQ. 0 ) THEN
         PRINT *, 'Time = ', STIME, ' seconds.'
       ENDIF
*
  400 CONTINUE
```

```
*
      PRINT *, 'Normal end of simulation'
      END


      SUBROUTINE fuelf0
**********************************************************************
*  fuelf0 initializes variables.
*  A second entry fuelrst does restart initialization and a time step.
*  A third entry fuelf does a simulation time step only.
*  IFCNTRL was eliminated.
**********************************************************************
C
C  SUBROUTINE FUELF(IFCNTL)
C
C  PURPOSE:  COMPUTE FUEL FEED SYSTEM FLOW AND PRESSURE DYNAMICS   SSM15200
C
C  IN THIS SUBROUTINE, THERE ARE 13 NODES DEFINED FOR THE CALCULATION OF
C  FLOW AND OTHER PROPERTIES OF FUEL.
C    1) INLET FROM FUEL TANK TO LOW PRESSURE FUEL PUMP (LPFP)
C    2) DUCT BETWEEN OUTLET OF LPFP AND INLET OF HPFP
C    3) DUCT BETWEEN OUTLET OF HPFP AND MAIN FUEL VALVE (MFV)
C   *4) UPPER NOZZLE COOLING FLOW
C   *5) MAIN COMBUSTION CHAMBER COOLING (BELOW THROAT)
C   *6) MAIN COMBUSTION CHAMBER COOLING (ABOVE THORAT)
C    7) COOLING CONTROL VALVE (CCV) FLOW
C    8) MIXER OF NOZZLE COOLING (4) AND CCV FLOW (7)
C    9) PREBURNER SUPPLY DUCT (DISTRIBUTER NODE)
C   10) COOLING FLOW FROM MFV TO NODE (11)
C   11) DOWN COOLING FLOW TO LOWER END OF NOZZLE
C  *12) COOLING FLOW OF LOWER 15% OF NOZZLE
C   13) COOLING FLOW FROM MFV TO THROAT END
C  THOSE NODES WITH * ARE THE ONES THAT CONTACT WITH COMBUSTION CHAMBER OR
C  NOZZLE AND HAVING HEAT EXCHANGE BETWEEN THEM.
C
C******ARGUMENT******
C  INPUT:  IFCNTL = INITIALIZATION ARGUMENT
C
C******COMMON USAGE******
C  INPUT:
C   VARIABLES                                                  SOURCE
C     TRQFP2,PFI,PINMC,PFP,POP,QIN1,RMFV,RCCV,DWFIG            HOTGAS
C     RMFV,RCCV                                                VALDYM
C     DWFIG                                                    IGN      SSM15300
C
C  OUTPUT:
C   VARIABLES                                                  DESTINATION
C     DWFPF,DWOPF,DWFBPV,DWFT2C,SF2,TW1                        HOTGAS
C     DW(2)                                                    CNTROL
C     P(3),P(10),P(7),P(8),RHO(3),RHO(7),DW(3),DW(7)           EMCO
```

```
C
C  SUBROUTINES CALLED:  PROP0, hyprop
C
*************************************************************************
*
      INTEGER Tstep
      LOGICAL fuelOK
*
      DIMENSION ZH(13),A6(13),ZZ(13),vVOL(13),DWA(13)
      DIMENSION TUBEN(13),PIFP(6),DWIFP(5),QOUT(13)
      DIMENSION ACS(13),ELEN(13),DHYD(13),WW1(13),WW2(13)
      DIMENSION ELENF(6),ZFL(6),ZFC(6),RIF(6)
*
*   A runtime division was replaced by a multiplication.
C
      PARAMETER ( v9336 = 1.0 / 9336.0, v386p4 = 1.0 / 386.4 )
      PARAMETER ( Tstep = 0 )
*
*  Labelled COMMON blocks
*
      INCLUDE 'units.com'
      INCLUDE 'blank.com'
      INCLUDE 'out.com'
      INCLUDE 'fuel.com'
      INCLUDE 'igni.com'
      INCLUDE 'balc.com'
      INCLUDE 'units.com'
*
*     NAMELIST/FUELFD/ELENF,ZFL,ZFC,RIF,DHYD,ELEN,AHT2,WW1,WW2,TW1,TW2
*
*  Namelists have been eliminated, in favor of an editable input
*  parameter file.
C                                                                SSM15900
*  Modern compilers default to dynamic memory allocation of local
*  variables. The following statement requires memory for all local
*  variables to be retained between calls.

      SAVE

C   UNEWF( ) IS TO CALCULATE NEW SPECIFIC INTERNAL ENERGY (SU) FOR A GIVEN
C   NODE WHERE INPUT FLOW (DWIN), INPUT ENTHALPY (HIN), OUTPUT FLOW (DWOUT),
C   NEW PRESSURE (PNEW) AND NEW DENSITY (RHONEW) ARE KNOWN.
*
      UNEWF(I, DWIN, HIN, DWOUT, PNEW1, RHONEW) =
     +  ( RHO(I)*SU(I) + vVOL(I) *
     +    ( QOUT(I) + HIN - DWOUT*PNEW1/( RHONEW * 9336.0 ) ) * DT ) /
     +      ( RHO(I) + vVOL(I)*DWIN*DT )
*
*  Though the Euler form x + rate * dt appears, the function above is not
*  taken to involve time integration in the usual sense. Else it should
```

```
*  be rewritten as the ratio of two trial integrations, as
*
*     UNEWF(I, DWIN, HIN, DWOUT, PNEW1, RHONEW) =
*    +  tryint( RHO(I)*SU(I), vVOL(I) *
*    +     ( QOUT(I) + HIN - DWOUT*PNEW1/( RHONEW * 9336.0 ) ), N  ) /
*    +  tryint( RHO(I),  vVOL(I)*DWIN, N+1 )
C
C   CHGX( ) was replaced by a more detailed monitoring system for
*           balancing iterations
C
*  Statement functions Z1 and Z2 were eliminated, to simplify maintenance.
C                                                                    SSM16000
      rlimit(floor, ceiling, x) = AMAX1( floor, AMIN1( ceiling, x) )
*
*  Replacing rlimit with     IF ( x .LE. floor ) THEN
*                              x = floor
*                            ELSE IF ( x .GT. ceiling ) THEN
*                              x = ceiling
*                            END
*  would be faster, but only by straightline coding, not as a FUNCTION.
*
      recpos(x) = AMAX1(0.,x)
      recneg(x) = AMIN1(0.,x)
*
*****************************************************************
*
*  As a mechanism for bypassing initialization code,
*
*    IF(FLAG.EQ.15.) GO TO 999
*
*  is obscure, and can be unreliable if FLAG is not initialized at load
*  time.  The function was accomplished more directly, and at no runtime
*  cost, by the ENTRY statement.
*
*     In the initialization, the following were eliminated:
*
*     - statement function formal parameters, which created new variables
*       unused by the simulation.
*
*     - initializations overwritten by reading input parameters
*
*     - local variables written before being read
*
      CPH2=0.0
      DDW2=0.0
      DSF1=0.0
      DSF2=0.0
      DWFPFP=0.0
      DWOPFP=0.0
      HI=0.0
```

```
      H10IN=0.0
      H10P=0.0
      H11P=0.0
      H12P=0.0
      H13P=0.0
      H4P=0.0
      H5P=0.0                                                        SSM16800
      H6P=0.0
      H7P=0.0
      H8P=0.0
      H9P=0.0
      PVFP1=0.0
      PVFP2=0.0
      P10P=0.0
      P11P=0.0
      P12P=0.0                                                       SSM17000
      P13P=0.0
      P4P=0.0
      P5P=0.0
      P6P=0.0
      P7P=0.0
      P8P=0.0
      P9P=0.0
      Q1=0.0
      Q2=0.0                                                         SSM17100
      RFS=0.0
      RHOT=0.0
      RITNV=0.0
      RP1=0.0
      RP2=0.0
      DO 991 I=1,13
        QOUT(I)=0.0
        ZH(I)=0.0
  991 CONTINUE
      DO 993 I=1,5
        DWIFP(I)=0.0
  993 CONTINUE
C
C  Read INPUT Parameters
C
      READ(run,30) (ELENF(J),J=1,6), (ZFL(J),J=1,6),
     +             (ZFC(J),J=1,6), (RIF(J),J=1,6)
      READ(run,30) (DHYD(J),J=1,13), (ELEN(J),J=1,13),
     +             (AHT2(J),J=1,13), (WW1(J),J=1,13),
     +             (WW2(J),J=1,13), (TW1(J),J=1,13),
     +             (TW2(J),J=1,13), (P(J),J=1,13),
     +             (T(J),J=1,13), (TUBEN(J),J=1,13)                  SSM17900
      READ(run,30) DWF, ELENT, ZFT1, ZCCV, ZFCOD, ZPFPD
      READ(run,30) ELEFPF, ACSFPF, ELEOPF, ACSOPF, ELENFN
      READ(run,30) HTCON, VOLFP1, VOLFP2, PCFP1, PCFP2
```

```
*     READ(5,30)                                              intentionally skippe
      READ(run,30) TRQF1B, TRQF2B, TDRAGF
      READ(run,'(//2X,3G12.4,I12)') GF1, GF2, PTOL, MAXL
30    FORMAT(//2X,6G12.4)                                              SSM17910
*
* Namelist input and formatted echo of input parameters were eliminated.
*
*     READ(7,FUELFD)                                                   SSM18000
*     WRITE(6,29)(ELENF(J),ZFL(J),ZFC(J),RIF(J),J=1,6) , etc
*
*  This section loads fuel flow interpolated functions.
*  Calls to FGEN( n, 1, x ), which reads the DTMINP file, were
*  replaced by calls to fgset( n ), which reads the editable function
*  file attached to unit 'dat'. Initializations recognized as
*  unnecessary were removed.
*
      CALL fgset( 30 )
      CALL fgset( 22 )
*     STIME=TIME                                                       SSM18300
      CALL fgset( 9 )
      PT = fgen(9, 1, STIME)
*
*  Functions of the same variable use different call numbers unless the
*  interpolation intervals are identical
*
      CALL fgset( 7 )
      HT = fgen(7, 7, STIME)
      CALL fgset( 10 )
      CALL fgset( 51 )
      CALL fgset( 52 )
      CALL fgset( 53 )
      CALL fgset( 54 )
      GAMF = H2GAM0( PT, 40.0, 1 )
      CALL PROP0
C                                                                      SSM18400
      SF2 = 0.01
      QHT412 = 0.464
      RHOT = 2.552E-3
      SUT = HT - PT / ( RHOT * 9336.0 )
*
*  RHOT is reset from the t(u,rho) table, X = throwaway temperature
*
      CALL hyrt(SUT, RHOT, 1, PT, X )
*
*     WRITE(6,37)DWF,PT,HT,RHOT,ELENT       was removed.
*
*  The replacement of mandatory full output by selective unformatted
*  output eliminated the destination for this initialization
*  documentation. DWF and ELENT echo input, and are documented by the
*  'run' unit file. PT, HT, and RHOT appear in the selectable output as
```

```
*  PIF(1), H(1), RHO(1), etc.
C
C      CALCULATION OF CONSTANTS AND INITIALIZATION                    SSM1850C
C
       PIF(1)=PT
       PIFP(1)=PT
       DWIF(1)=DWF
       DWIFP(1)=DWF                                                   SSM1860C
       DO 71 J = 2, 5
          PIF(J)=PIF(J-1)+ELENF(J-1)*RHOT
          PIFP(J)=PIF(J)
          DWIF(J)=DWF
          CALL uninto( PIF(J), J + 27 )
    71 CONTINUE
       DO 72 J = 1, 5
         CALL uninto( DWIF(J), J + 23 )
    72 CONTINUE
       PFS = PIF(5) + ELENF(5)*RHOT
       CALL uninto( PFS, 33 )
       PFSP=PFS
       PIF(6)=PFS
       PIFP(6)=PFS
C
       DO 60 J = 1, 3
         RHO(J) = RHOT
         H(J) = HT                                                    SSM18700
    60 CONTINUE
       DO 70 J = 4,13
         RHO(J)= 1.0875E-4 * P(J) / T(J)
*
*           CALL PROP(SU(J),RHO(J),J,P(J),T(J),4)   was replaced by
*
         CALL hyut( SU(J), RHO(J), 2 * J, P(J), T(J) )
*
*  which adjusts SU(J) based on the p(u,rho) table, and reserves call
*  points 2, ..., 27
*
         H(J) = SU(J) + P(J) / ( RHO(J) * 9336.0 )
C
*         ACS(J) = 3.14*DHYD(J)**2/4.0*TUBEN(J)   replaced by
*
         ACS(J) = 0.7854 * DHYD(J)**2 * TUBEN(J)
*
         VOL(J) = ACS(J) * ELEN(J)
*
*         ZZ(J)=ELEN(J)/ACS(J)/386.4  replaced by
*
         ZZ(J) = ELEN(J) / ( ACS(J) * 386.4 )
         RR(J) = R(J)                                                 SSM18800
*
```

```
*        A6(J)=HTCON/DHYD(J)**1.8        replaced by
*
         A6(J) = HTCON / ( DHYD(J) * X10th( DHYD(J), 8) )
*
*   Interpolation of x**(-1.8) is not necessary as long as the function
*   is used only in initialization.
*
         DW(J) = DWF
         QOUT1(J) = 0.0
         QOUT2(J) = 0.0
         QIN1(J) = 0.0
         QIN2(J) = 0.0
         CALL unint0( TW1(J), J - 1 )
         CALL unint0( TW2(J), J + 9 )
   70 CONTINUE
C
      ZFS=ZZ(2)+1.0/ZFCOD+ZZ(3)+1./ZPFPD
      ZZ(6)=ZFT1                                                    SSM18900
      ZZ(10)=.5*ZZ(11)
      ZZ(11)=.5*ZZ(11)
*
*   Configuration constants set during initialization are documented on
*   an output file on unit 'init'.
*
      WRITE(init,90)(J,ACS(J),VOL(J),A6(J),ZZ(J), J=1,13)
90    FORMAT('  J  ACS         VOL         A6            ZZ'
     *     /(1I3,1P4E11.3))
C
      WRITE(init,260)ZFS
260   FORMAT('0START TRANSIENT ASSUMES PRECHILLED PUMPS AND LIQUID H2',
     *' THROUGH TO PUMP DISCHARGE'/     '  ZFS' /1PE11.3)          SSM19000
*
      DWMC = 0.0
      CALL unint0( DWMC, 36 )
      DWFNBP = 0.0
      CALL unint0( DWFNBP, 37 )
      DWFT1 = 0.0
      DWFPF = 0.0
      DWOPF = 0.0
      undw2c = 0.0
      CALL unint0( 0.0, 23 )
      ZFPF = ELEFPF/(ACSFPF*386.4)
      ZOPF = ELEOPF/(ACSOPF*386.4)
      SF1=0.01                                                      SSM19100
      CALL lmint0( SF1, 1, 0.1, 1.E20 )
      SF2=0.01
      CALL lmint0( SF2, 2, 0.001, 1.E20 )
      TRQFP1=0.0
      TRQFP2=0.0
      PFT1 = 0.0
```

```
      PRFT1 = 1.0
      TFT1=T(6)
      fgen21 = 9340.0 * fgen(21, 2, 0.0)
*
*  Load relative change tolerance for energy balance convergence tests
*
      CALL chg0( PTOL )
*
      IF ( STIME .EQ. 0.0 ) RETURN
C
C       RESTART INITIALIZATION
C
      DWFASI = DWFIG(1) + DWFIG(2) + DWFIG(3)
      RHOP1C = .5 * ( RHO(1) + RHO(2) )
      PHIP1 = DW(2) / (RHOP1C * SF1)
      DP1 = fgen(51, 3, PHIP1) * RHOP1C * SF1**2 * CDPFP1
      DP1P = DP1
      PHIP2 = DW(2) / (RHO(3) * SF2 )
      DP2 = fgen(53, 4, PHIP2) * RHO(3) * SF2**2 * CDPFP2
      DP2P = DP2
      PFSP = PFS
      DO 2010 J=1,6
         PIFP(J)=PIF(J)                                          SSM19400
         DWIFP(J)=DWIF(J)
2010  CONTINUE
      DW2P=DW(2)
      CALL unint0( DW2P, 34 )
      CALL unint0( DW(10), 35 )
      DW10O=DW(10)+DWMC+DWFNBP+DWFASI
      DW10I=DW(3)
      P4P=P(4)
      P5P=P(5)
      P6P=P(6)
      P7P=P(7)
      P8P=P(8)
      P9P=P(9)                                                   SSM19500
      P10P=P(10)
      P11P=P(11)
      P12P=P(12)
      P13P=P(13)
      DO 2015 J=4,13
        SU(J) = H(J) - P(J) / ( RHO(J) * 9336.0 )
2015  CONTINUE
      CALL unint0( RHO(10), 38 )
      CALL unint0( DW(11), 39 )
      CALL unint0( RHO(11), 40 )
      CALL unint0( DW(12), 41 )
      CALL unint0( RHO(12), 42 )
      CALL unint0( DW(4), 43 )
      CALL unint0( RHO(4), 44 )
```

```
      CALL unint0( DW(13), 45 )
      CALL unint0( RHO(13), 46 )
      CALL unint0( DW(5), 47 )
      CALL unint0( RHO(5), 48 )
      CALL unint0( DW(6), 49 )
      CALL unint0( RHO(6), 50 )
      CALL unint0( DW(7), 51 )
      CALL unint0( RHO(7), 52 )
      CALL unint0( DW(8), 53 )
      CALL unint0( RHO(8), 54 )
      CALL unint0( DWFPF, 55 )
      CALL unint0( DWOPF, 56 )
      CALL unint0( RHO(9), 57 )
*
      ENTRY fuelf
*************************************************************************
*
C     TIME TRANSIENT CALCULATIONS                                       SSM19200
C
      DWFASI = DWFIG(1) + DWFIG(2) + DWFIG(3)
C
*      STIME = TIME
C                                                                       SSM19600
C     LOW RATE CALCULATIONS
C
C     FUEL PUMP SPEEDS (RAD/SEC)
C
      IF(SF1 .LT. 11.0 .AND. TRQFT1 .LT. TRQFP1+TRQF1B) THEN
        DSF1 = 0.0
      ELSE
        DSF1 = (TRQFT1-TRQFP1)/GF1
      END IF
*
*   Integral limited below by 0.1
*
      SF1 = prlint( DSF1, delt, 1 )                                     SSM19700
      IF(SF2 .LT. 11.0 .AND. TRQFT2 .LT. TRQFP2+TRQF2B) THEN
        DSF2 = 0.0
      ELSE
        DSF2 = (TRQFT2-TRQFP2-TDRAGF)/GF2
      END IF
*
*   Integral limited below by 0.001
*
      SF2 = prlint( DSF2, delt, 2 )
C
      QIN1(12)=QIN1(4)*QHT412
      QIN1(4)=QIN1(4)-QIN1(12)                                          SSM19800
C
      DO 2050 J=4,13
```

```
          vVOL(J) = 1.0 / VOL(J)
          ZH(J) = A6(J) * HTF(T(J),P(J))
 2050 CONTINUE
C
C    THE AVERAGE FLOW USED TO CALCULATE THE HEAT EXCHANGE IS THE AVERAGE OF
C       1) THE FLOW OF CURRENT NODE
C       2) THE FLOW OF UPSTREAM NODE(S)
C    THIS IS A VERY ROUGH ESTIMATION SINCE IT REALY DEPENDS ON THE TIME
C    INTERVAL OF THE SIMULATION AND THE LENGTH OF EACH SEGMENT.
C
        DWA(4)=ABS(0.5*(DW(12)+DW(4)))
        DWA(5)=ABS(0.5*(DW(13)+DW(5)))
        DWA(6)=ABS(0.5*(DW(5)+DW(6)))
        DWA(7)=ABS(0.5*(DWFNBP+DW(7)))
        DWA(8)=ABS(0.5*(DW(7)+DW(4)+DW(8)))                          SSM19900
        DWA(9)=ABS(0.5*(DW(8)+DWFPF+DWOPF))
        DWA(10)=ABS(0.5*(DW10I+DW10O))
        DWA(11)=ABS(0.5*(DW(10)+DW(11)))
        DWA(12)=ABS(0.5*(DW(11)+DW(12)))
        DWA(13)=ABS(0.5*(DWMC+DW(13)))
C
C    THE FOLLOWING SECTIONS OF PROGRAM CALCULATE THE HEAT EXCHANGE OF THE
C    COOLING FLOW AND THE COMBUSTION CHAMBER.
C    THE EQUATIONS USED IN THE PROGRAM IS DIFFERENT FROM THOSE GIVEN IN THE
C    SSME SPECIFICATION PP.31-32.
C    I BELIEVE THEY ARE THE SIMPLIFIED VERSION OF THE EQUATION.
C    IF THE COOLING EFFECT BECOME IMPORTANT IN THE SIMULATION THEN THE FOLLO
C    SECTION OF CODE WILL HAVE TO BE VERIFIED.
C
C    THERE ARE TWO TYPES OF HEAT FLOW OCCURED IN THE HEAT EXCHANGE
C      1) BY CONDUCTION DESCRIBED AS QOUT1(J) AND QOUT2(J)
C      2) BY CONVECTION DESCRIBED AS FUNCTION OUTPUT QFLUX().
C
C    IN THE FOLLOWING SECTION, THE REPRESENTATIONS ARE
C       1) QOUT1: HEAT TRANSFERED FROM MAIN COMB. CHAMBER AND NOZZLE WALLS
C       2) QFLUX(**1,..): BOILING HEAT FROM MCC AND NOZZLE WALLS
C       3) QOUT2: HEAT TRANSFERED FROM AMBIENT WALLS
C       4) QFLUZ(**1,..): BOILING HEAT FROM AMBIENT WALLS
C       5) QBM: HYDROGEN SATURATION ENTHALPY AT THE PRESSURE
C
* One unnecessary real**real has been eliminated, and the remaining
* real**.55 and real**.8 replaced by equally spaced linear interpolations.

      DO 2060 J=4,13
        IF(J.GT.6.AND.J.NE.12) THEN
          Q1=0.
          QOUT2(J) = ZH(J) * XtoY( T(J)/TW2(J), .55 ) *
     +                  X10th(DWA(J), 8) * (TW2(J)-T(J))
        ELSE                                                         SSM20000
          dwtemp = X10th(DWA(J), 8)
```

```
          QOUT1(J)= ZH(J) * XtoY( T(J)/TW1(J), .55 ) *
     +                      dwtemp * (TW1(J)-T(J))
          Q1=(QOUT1(J)+QFLUX(TW1(J),T(J),P(J),H(J)))*AHT1(J)
          TW1(J) = pruint(
     +    ( QIN1(J) - Q1 ) / ( fgen( 22, 1+J, TW1(J) ) * WW1(J) ),
     +                                                  0 , J - 1 )
          QOUT2(J) = ZH(J) * XtoY( T(J)/TW2(J), .55 ) *
     +                      dwtemp * (TW2(J)-T(J))
        END IF
        QBM = DWA(J)*( H2SATH( P(J) ) - H(J) )
        QB(J) = rlimit( 0., QBM, QFLUX( TW2(J), T(J), P(J), H(J) ) )
        Q2 = ( QOUT2(J) + QB(J) )*AHT2(J)
        TW2(J) = print( TW2(J),
     +   ( QIN2(J) - Q2 ) / ( fgen( 22, 11+J, TW2(J) ) * WW2(J) ),
     +                                                  0, J + 9 )
        QOUT(J) = Q1 + Q2                                          SSM20100
 2060 CONTINUE
C
C  FUEL TANK CHARACTERISTIC AS A FUNCTION OF TIME
C
      PT = fgen(9, 1, STIME)
      HT = fgen(7, 7, STIME)
      H(1) = HT
      PIFP(1) = PT
      SUT= HT - PT / (RHOT * 9336.0)
      CALL hyrt( SUT, RHOT, 1, PT, X )

      RHOFS = RHOT
C                                                                  SSM20200
C     LOW PRES FUEL TURBINE
C
C   THIS IS TO CALCULATE THE PERFORMANCE OF LPFT (FT1).  HOWEVER, THE
C   EQUATION USED HERE TO CALCULATE THE TURBINE PARAMETER (ETAFT1) AND
C   THE SPECIFIC HEAT CONSTANT (CPH2) ARE NOT DESCRIBED IN THE DOCUMENT.
C
      TFT1=T(6)
      ETAFT1 = DMFT1 * SF1 * 33.0965 /
     +     X10th( AMAX1(0.0001, 9270. * TFT1 * (1.0/PRFT1- 1.0) ), 5 )
*
      RITNV = RITN / fgen(30, 25, ETAFT1)**2
      R6 = RITNV + RFT1V + RMCI
*
      CPH2 = fgen(15, 26, TFT1) - 0.0887 +
     +          (0.1241 * PFT1 - 3.732E-5 * PFT1**2) /
     +              ( AMAX1(51.,TFT1) - 50. )                    SSM20300
*
      TRQFT1 = TRBTRQ(SF1,UCFT1,TFT1,PFT1,PRFT1,1,CPH2,DWFT1,
     *              H2GAMA(PFT1,TFT1,7)) * CTQFT1
*
      PFT1 = P(6) - RFT1V / RHO(6) * DW(6) * ABS( DW(6) )
```

```
      PRFT1 = ( PFT1 - RITNV/RHO(6)*DWFT1*ABS( DWFT1 ) ) / PFT1
      PFT1D = PFT1 * PRFT1
      TFT1D = TFT1 - (TRQFT1 * SF1) / ( fgen21 * DWFT1 + 1.0E-06 )
C
C         TURBINE COOLANT FLOW STARTS AT DP=185
C
C   THE EXTRA COOLANT FLOW TO COOL THE HIGH PRESSURE FUEL TURBINE:
C     THE VALVE START TO OPEN AT DP=185 AND BECOME FULLY OPEN AT DP=195.
C
*     DWFT2C = FLOW( DWFT2C, 0.1, RFT2C/RHO(3), P(3)-PFI, 23)       SSM2040C
*     DWFT2C = DWFT2C * rlimit(0.0, 1.0, (P(3) - PFI - 185.) * 0.1 )
*
*  If the rlimit < 1, the effect is to drive DWFT2C down with a time consta
*  dependent on the integration step size, which surely isn't intended.
*  The model is replaced by a variable limit imposed on an unlimited
*  integrator.
*
      undw2c = prflow( undw2c, 0.1, RFT2C/RHO(3), P(3)-PFI, 23)     SSM2040C
      DWFT2C = undw2c * rlimit(0.0, 1.0, (P(3) - PFI - 185.) * 0.1 )
C
C     BEGIN PRESSURE FLOWRATE ITERATION
C
C   THE FUEL SYSTEM USES THE ITERATION METHOD TO CALCULATE THE NEXT STATE
C   OF THE PRESSURE, FLOWRATE AND ENERGY BALANCE.  THE ITERATION IS STOPPED
C   WHEN THE MAXIMUM CHANGE OF THE PRESSURE OF THE FUEL SYSTEM IS LESS THEN
C   THE TOLERANCE ALLOWANCE PTOL.
C
*   Initialize change monitor iteration counts
*
      CALL its0
C
          DO 4000 LOOPS = 1 , MAXL
C                                                                SSM20500
*   Reset change monitor for balance iteration test
*
      fuelOK = .TRUE.

    THIS SECTION IS TO CALCUALTE THE CAVITATION FOR FP1 AND FP2.

C   IN THE FOLLOWING CALCULATIONS, THE FUNCTION PROP( ) IS USED TO FIND
C   OUT THE STATUS OF THE CURRENT HYDROGEN GIVEN KNOWN PARAMETERS.
C   PROP( ) IS A CHARACTERISTIC DATA MAP FOR HYDROGEN - RELATIONS AMONG
      INTERNAL ENERGY, DENSITY, PRESSURE, AND TEMPERATURE.
C
      PVFP1 = H2VP(T(2))
      PVFP2 = H2VP(T(3))
      PFP1R = PVFP1 + PCFP1*SF1**2
      PFP2R = PVFP2 + PCFP2*SF2**2
      RHOP1 = 0.5 * ( RHO(1) + RHO(2) )
      RHOP2 = RHO(3)
```

```
      IF (PFS .GT. PFP1R) THEN
         RHOP1C = RHOP1
      ELSE
C
C  CAVITATION FOR FP1
C
         RHOP1C = RHOP1 * rlimit( 0.25, 1.0,                          SSM20600
     +       0.25 + (PFS-PVFP1)*.75/(PFP1R - PVFP1 + 1.0E-10) )
      END IF
      TRQFP1 = fgen(52, 27, DW2P/(RHOP1C * SF1) ) *
     +                              RHOP1C * SF1**2 * CTQFP1
      TRQFP1 = ABS(TRQFP1)
      IF( PFS2.GT.PFP2R) THEN
         RHOP2C = RHOP2                                               SSM20700
      ELSE
C
C  CAVITATION FOR FP2
C
         RHOP2C = RHOP2 * rlimit(0.05, 1.0,
     +       0.05 + (PFS2-PVFP2) * 0.95 /(PFP2R - PVFP2 + 1.0E-10) )
      END IF
      TRQFP2 = fgen(54, 28, DW2P/(RHOP2C * SF2) ) *
     +                              RHOP2C * SF2**2 * CTQFP2
      TRQFP2 = ABS(TRQFP2)
C
C          FEEDLINE
C
C   THIS IS THE FEEDLINE BETWEEN FUEL TANK AND LPFP
C
      DO 1060 J = 1, 5
        DWIFP(J) = trflow( DWIF(J), ZFL(J), -RIF(J),
     +   PIFP(J) - PIFP(J+1) + ELENF(J) * RHOFS * DDX * v386p4, J+23 )
 1060 CONTINUE
      DO 1070 J = 2, 5
        PIFP(J) = truint( ( DWIFP(J-1) - DWIFP(J) ) / ZFC(J),
     +                                          Tstep, J + 27 )  SSM20800
 1070 CONTINUE
      PNEW = truint(  ( DWIFP(5) - DW2P ) / ZFC(6), Tstep, 33 )
C
*     CHANGE=CHGX(CHANGE,PNEW,PFSP) was replaced by
*
      PFSP = relax( fuelOK, PNEW, PFSP, 1 )
*                                          part of a convergence
*   management system recommended to give more maintenance control
*   over iterative energy balancing.
*
      PIFP(6) = PFSP
C
C          PUMPS
C   LPFP AND HPFP AND FLOW BETWEEN FEEDLINE AND MFV
```

```
C
      IF( RMFV .GT. 1.0E10 ) THEN
        DW2P = 0.
      ELSE
        RP1 = CP(2) * CDPFP1 / RHOP1C                                  SSM20900
        RP2 = CP(3) * CDPFP2 / RHOP2C
        RFS = RFCOD/RHO(2) + RP1 + RP2 + ( RR(3) + RMFV ) / RHO(3)
        DW2P = trflow( DW(2), ZFS, -RFS,
     +    (PFSP - P10P + DP1P + DP2P) + (RP1 + RP2) * DW2P * ABS(DW2P),
     +        34 )
      END IF
      DDW2 = ( DW2P-DW(2) ) / DT
      DW(1) = DW(2)
      IF(DW2P.EQ.0.) THEN                                              SSM21000
        H(2)=H(1)
        H(3)=H(2)
      ELSE
        H(2) = H(1) + TRQFP1 * SF1 / (DW2P * 9336.)
        H(3) = H(2) + TRQFP2 * SF2 / (DW2P * 9336.)
      ENDIF
      DW3P = DW2P - DWFT2C
      pDW3P = recpos(DW3P)
C
      PHIP1 = DW2P / ( RHOP1C * SF1 )
      DW2A = ( DW2P + DW(2) ) * 0.5
      DP1P = fgen(51, 3, PHIP1) * RHOP1C * SF1**2 * CDPFP1
      P(2) = PFS + DP1 - DDW2 * ZZ(2)
C
      PFS2 = P(2) - RFCOD/RHO(2) * DW2A * ABS(DW2A) - DDW2/ZFCOD
C
      PHIP2=DW2P / (RHOP2C * SF2)
      DP2P = fgen(53, 4, PHIP2) * RHOP2C * SF2**2 * CDPFP2             SSM21200
      P(3) = PFS2 + DP2 - DDW2 * ZZ(3)
      SU1 = HT - PFS / ( RHO(1) * 9336. )
      SU2 = H(2) - P(2) / ( RHO(2) * 9336. )
      SU3 = H(3) - P(3) / ( RHO(3) * 9336. )
      CALL hyrt( SU1, RHO(1), 1, PFS, TTI )
      CALL hyrt( SU2, RHO(2), 3, P(2), T(2) )
      CALL hyrt( SU3, RHO(3), 5, P(3), T(3) )
      T(2) = T(1) + T(2) - TTI
      T(3) = T(1) + T(3) - TTI
C                                                                      SSM21300
C     MFV DIFFUSER
C
      HI=H(3)
      IF(DW(3).LT.0.0) HI=H(10)
C
      DW10P = trflow( DW(10), ZZ(10), -RR(10)/RHO(10), P10P-P11P, 35 )
      DWMCP = trflow( DWMC, ZZ(13), -RR(13)/RHO(10), P10P-P13P, 36 )
      DWFNP = trflow( DWFNBP, ZZ(7), -RR(7)/RHO(10), P10P-P7P, 37 )
```

```
C
*   Doing rectifications once instead of three times each.
*
      pDW10P = recpos(DW10P)
      DW10Pn = recneg(DW10P)
      pDWMCP = recpos(DWMCP)
      DWMCPn = recneg(DWMCP)
      pDWFNP = recpos(DWFNP)
      DWFNPn = recneg(DWFNP)
      DW10OP = pDW10P + pDWMCP + pDWFNP + DWFASI - DW3Pn                SSM21400
      DW10IP = pDW3P - DW10Pn - DWMCPn - DWFNPn
      RHO10P = truint( vVOL(10) * (DW10IP - DW10OP), Tstep, 38 )
      H10IN = pDW3P * HI - DW10Pn * H11P - DWMCPn * H13P - DWFNPn * H7P
      SU10P = UNEWF( 10, DW10IP, H10IN, DW10OP, P10P, RHO10P )
      H10P = SU10P + P10P / ( RHO10P * 9336. )
      CALL hypt(SU10P, RHO10P, 10, PNEW, T(10) )                       SSM21500
*
      P10P = relax( fuelOK, PNEW, P10P, 2 )
C
C     DOWNCOMERS
C
      DW11P = trflow( DW(11), ZZ(11), -RR(11)/RHO(11), P11P-P12P, 39 )
      DW11Pp = recpos(DW11P)
      DW11Pn = recneg(DW11P)
      RHO11P = truint( vVOL(11)*(DW10P - DW11P), 40 )
      DW11I = pDW10P - DW11Pn
      HI = pDW10P * H10P - DW11Pn * H12P
      DW11O = DW11Pp - DW10Pn                                          SSM21600
      SU11P = UNEWF( 11, DW11I, HI, DW11O, P11P, RHO11P )
      H11P = SU11P + P11P/(RHO11P * 9336.)
      CALL hypt( SU11P, RHO11P, 11, PNEW, T(11) )

      P11P = relax( fuelOK, PNEW, P11P, 3 )
C
C      LOWER 15% OF NOZZE
C
      DW12P = trflow( DW(12), ZZ(12), -RR(12)/RHO(12), P12P-P4P, 41 ) 1700
      DW12Pp = recpos(DW12P)
      DW12Pn = recneg(DW12P)
      RHO12P = truint( vVOL(12)*(DW11P - DW12P), 42 )
      DW12I = DW11Pp - DW12Pn
      HI = DW11Pp * H11P - DW12Pn * H4P
      DW12O = DW12Pp - DW11Pn
      SU12P = UNEWF( 12, DW12I, HI, DW12O, P12P, RHO12P )
      H12P = SU12P + P12P/(RHO12P * 9336.)
      CALL hypt( SU12P, RHO12P, 12, PNEW, T(12) )
*
      P12P = relax( fuelOK, PNEW, P12P, 4 )
C                                                                      SSM21800
C      NOZZLE REGEN COOLING FLOW
```

```
C
      DW4P = trflow( DW(4), ZZ(4), -RR(4)/RHO(4), P4P-P8P, 43 )
      RHO4P = truint( vVOL(4)*(DW12P - DW4P), 44 )
      DW4Pp = recpos(DW4P)
      DW4Pn = recneg(DW4P)
      DW4I = DW12Pp - DW4Pn
      HI = DW12Pp * H12P - DW4Pn * H8P
      DW4O = DW4Pp - DW12Pn
      SU4P = UNEWF( 4, DW4I, HI, DW4O, P4P, RHO4P )
      H4P = SU4P + P4P/(RHO4P * 9336.)
      CALL hypt( SU4P, RHO4P, 4, PNEW, T(4) )                    SSM2190C
*
      P4P = relax( fuelOK, PNEW, P4P, 5 )
C
C     MCC SUPPLY DUCT
C
      DW13P = trflow( DW(13), ZZ(13), -RR(5)/RHO(13), P13P-P5P, 45 )
      RHO13P = truint( vVOL(13)*(DWMCP - DW13P), 46 )
      DW13Pp = recpos(DW13P)
      DW13Pn = recneg(DW13P)
      DW13I = DWMCPp- DW13Pn
      HI = DWMCPp*H10P - DW13Pn*H5P
      DW13O = DW13Pp - DWMCPn                                    SSM22000
      SU13P = UNEWF( 13, DW13I, HI, DW13O, P13P, RHO13P )
      H13P = SU13P + P13P/(RHO13P * 9336.)
      CALL hypt( SU13P, RHO13P, 13, PNEW, T(13) )
*
      P13P = relax( fuelOK, PNEW, P13P, 6 )
C
C     MAIN COMBUSTOR REGEN COOLING FLOW
C                                                                SSM22100
C      NODE 5 BELOW THROAT
C
      DW5P = trflow( DW(5), ZZ(5), -RR(6)/RHO(5), P5P-P6P, 47 )
      RHO5P = truint( vVOL(5)*(DW13P - DW5P), 48 )
      DW5Pp = recpos(DW5P)
      DW5Pn = recneg(DW5P)
      DW5I = DW13Pp - DW5Pn
      HI= DW13Pp * H13P - DW5Pn * H6P
      DW5O = DW5Pp- DW13Pn
      SU5P = UNEWF( 5, DW5I, HI, DW5O, P5P, RHO5P )
      H5P = SU5P + P5P/(RHO5P * 9336.)
      CALL hypt( SU5P, RHO5P, 5, PNEW, T(5) )                    SSM22200
*
      P5P = relax( fuelOK, PNEW, P5P, 7 )
C
C      NODE 6 ABOVE THROAT
C
      DW6P = trflow( DW(6), ZZ(6), -R6/RHO(6), P6P - PINMC, 49 )
      RHO6P = truint( vVOL(6)*(DW5P - DW6P), 50 )
```

```
      DW6Pp = recpos(DW6P)
      DW6Pn = recneg(DW6P)
      DW6I = DW5Pp- DW6Pn
      HI = DW5Pp*H5P - DW6Pn*H6P
      DW6O = DW6Pp- DW5Pn                                          SSM22300
      SU6P = UNEWF( 6, DW6I, HI, DW6O, P6P, RHO6P )
      H6P = SU6P + P6P/(RHO6P * 9336.)
      CALL hypt( SU6P, RHO6P, 6, PNEW, T(6) )
*
      P6P = relax( fuelOK, PNEW, P6P, 8 )
C
C     CCV INLET DUCT
C
      DW7P = trflow( DW(7), .04, -(RCCV + RR(9) )/RHO(7),
     +                                              P7P - P8P, 51 )
      RHO7P = truint( RHO(7), vVOL(7)*(DWFNP - DW7P), 52 )         SSM22400
      DW7Pp = recpos(DW7P)
      DW7Pn = recneg(DW7P)
      DW7I = pDWFNP- DW7Pn
      HI = pDWFNP*H10P - DW7Pn*H8P
      DW7O = DW7Pp - DWFNPn
      SU7P = UNEWF( 7, DW7I, HI, DW7O, P7P, RHO7P )
      H7P = SU7P + P7P/(RHO7P*9336.)
      CALL hypt( SU7P, RHO7P, 7, PNEW, T(7) )
*
      P7P = relax( fuelOK, PNEW, P7P, 9 )
C
C     MIXER                                                        SSM22500
C
      DW8P = trflow( DW(8), ZZ(8), -RR(8)/ RHO(8), P8P-P9P, 53 )
      RHO8P = truint( vVOL(8)*(DW7P + DW4P - DW8P), 54 )
      DW8Pp = recpos(DW8P)
      DW8Pn = recneg(DW8P)
      DW8I = DW4Pp + DW7Pp - DW8Pn
      HI = DW4Pp*H4P + DW7Pp*H7P - DW8Pn*H9P
      DW8O = DW8Pp - DW7Pn- DW4Pn
      SU8P = UNEWF( 8, DW8I, HI, DW8O, P8P, RHO8P )
      H8P = SU8P + P8P/(RHO8P*9336.)
      CALL hypt( SU8P, RHO8P, 8, PNEW, T(8) )                      SSM22600
*
      P8P = relax( fuelOK, PNEW, P8P, 10 )
C
C           PREBURNER SUPPLY DUCT
C
      DWFPFP = trflow( DWFPF, ZFPF, -RFPFI/RHO(9), P9P - PFP, 55 )
      DWOPFP = trflow( DWOPF, ZOPF, -ROPFI/RHO(9), P9P - POP, 56 )
C
      DW9P = DWFPFP + DWOPFP
      RHO9P = truint( vVOL(9)*(DW8P - DW9P), 57 )                  SSM22700
      DW9Pn = recneg(DW9P)
```

```
      DW9I = DW8Pp- DW9Pn
      HI = DW8Pp*H8P - DW9Pn*H9P
      DW9O = recpos(DW9P) - DW8Pn
      SU9P = UNEWF( 9, DW9I, HI, DW9O, P9P, RHO9P )
      H9P = SU9P + P9P/(RHO9P * 9336.)
      CALL hypt( SU9P, RHO9P, 9, PNEW, T(9) )
*
      P9P = relax( fuelOK, PNEW, P9P, 11 )
C
*     IF( fuelOK.AND.LOOPS.GT.2) GO TO 3500  was replaced by       SSM2280(
      IF( fuelOK ) GO TO 3500
*                                            for a speedup of 3 when
*     balancing keeps up with changing conditions. Output of variable
*     names was eliminated as unnecessary for a simulation monitoring
C     function.
C
 4000 CONTINUE
C
C   There is a convergence failure when the END OF ITERATION LOOP is reache
C
      CALL wrchg( IT, 1, 11, 'Fuelflow convergence failure:' )
*
*  Set stepped values to final trial values:
C
 3500 DO 4100 J = 2, 6
        PIF(J) = step( J + 27 )                                      SSM23000
 4100 CONTINUE
      DO 4110 J = 1, 5
        DWIF(J) = step( J + 23 )
 4110 CONTINUE
C
      PFS=PFSP
      DP1=DP1P
      DP2=DP2P
      DW(2) = step( 34 )
      DW(3)=DW3P
      DWFNBP = step( 37 )
      DWFN=DW10P                                                     SSM23100
      DWMC = step( 36 )
      DW10O=DW10OP
      DW10I=DW10IP
C
      P(4)=P4P
      RHO(4) = step( 44 )
      SU(4)=SU4P
      H(4)=H4P
      DW(4) = step( 43 )
C                                                                    SSM23200
      P(5)=P5P
      RHO(5) = step( 48 )
```

```
      SU(5)=SU5P
      H(5)=H5P
      DW(5) = step( 47 )
C
      P(6)=P6P
      RHO(6) = step( 50 )
      SU(6)=SU6P
      H(6)=H6P                                                    SSM2330
      DW(6) = step( 49 )
      DWFT1=DW6P
C
      P(7)=P7P
      RHO(7) = step( 52 )
      SU(7)=SU7P
      H(7)=H7P
      DW(7) = step( 51 )
C
      P(8)=P8P                                                    SSM23400
      RHO(8) = step( 52 )
      SU(8)=SU8P
      H(8)=H8P
      DW(8) = step( 53 )
C
      P(9)=P9P
      RHO(9) = step( 57 )
      SU(9)=SU9P
      H(9)=H9P
      DW(9)=DW9P                                                  SSM23500
      DWFPF = step( 55 )
      DWOPF = step( 56 )
C
      P(10)=P10P
      RHO(10) = step( 38 )
      SU(10)=SU10P
      H(10)=H10P
      DW(10) = step( 35 )
C
      P(11)=P11P                                                  SSM23600
      RHO(11) = step( 40 )
      SU(11)=SU11P
      H(11)=H11P
      DW(11) = step( 39 )
C
      P(12)=P12P
      RHO(12)=RHO12P
      SU(12)=SU12P
      H(12)=H12P
      DW(12) = step( 41 )                                         SSM23700
C
      P(13)=P13P
```

```
      RHO(13) = step( 46 )
      SU(13)=SU13P
      H(13)=H13P
      DW(13) = step( 45 )
C
C   QF IS FUEL FLOW IN GALLON/MIN
C
      QF=0.25974*DW2P/RHO(2)
      TFP1=T(2)
      PFP1=P(2)                                                    SSM2380(
      PFPD1=P(2)
      PFPD=P(3)
C
      RETURN
      END
C
C    HTF FUNCTION IS TO CALCULATE THE HEAT TRANSFER COEFFICIENT
C    SUCH AS THE ONES IN PAGE 31 OF THE DOCUMENT.
C    I BELIEVE THIS IS THE SIMPLIFIED VERSION OF CALCULATION.
C
      FUNCTION HTF(T,P)
      HTF = 0.1425 + (2.85E-4 + 1.8E-8 * P) * T
      RETURN
      END
```

'hyprop.for':

```
      subroutine prop0
************************************************************************
C
C  SUBROUTINE PROP(SU,SRHO,N,P,T,NN)
C                                                                  SSM76500
C  PURPOSE:  HYDROGEN PROPERTY DATA
C
C   THIS TABLE IS THE ENERGY MAP OF HYDROGEN PROPERTY.  IT IS ABOUT THE
C   RELATIONSHIP AMONG SPECIFIE ENERGY (SU), DENSITY (SRHO), PRESSURE (P)
C   AND TEMPERATURE.  PRESUMELY, FOR ANY GIVEN TWO PARAMETERS, THE OTHER
C   PARAMETERS CAN BE FOUND.
C
C******ARGUMENTS******
C
C  SU   = SPECIFIC INTERNAL ENERCY, BTU/LB
C  SRHO = DENSITY, LB/IN3
C  P    = PRESSURE, PSI
C  T    = TEMPERATURE, DEG R
C  N    = CALLER NODE INDEX
C
*   PROP was divided into four separate entries:
*       prop0 - precomputes slopes for all interpolations
*       hypt  - interpolates P and T, given SU and SRHO
*       hyrt  - interpolates SRHO and T, given SU and P
*       hyut  - interpolates SU and T, given SRHO and P
*
************************************************************************
C
      PARAMETER (NCALL=50, NU=14, NRHO=18, NSAT = 7 )              SSM76700
      PARAMETER ( v1728 = 1728. )
*
*  Hydrogen data, saturation curves:
*
      DIMENSION U(NU), RHO(NRHO), PRES(NU,NRHO), TEMP(NU,NRHO),
     +          RSAT(NSAT), PSAT(NSAT)
*
*  Precomputed slopes: ( DRHO was inverted )
*
      DIMENSION XP(NU,NRHO), TXP(NU,NRHO), vDRHO(NRHO),
     +          UvsP(NU, NRHO), RHOvsP(NU, NRHO),
     +          RSvsU(NSAT), PSvsU(NSAT), vPS(NSAT), DRSAT(NSAT)
*
*  Call point identifiers, initially at lowest interval
*
      DIMENSION ICALL(NCALL), JCALL(NCALL), R(2)
      DATA ICALL / NCALL*2 /, JCALL / NCALL*2 /
*
      DATA RSAT / 4.6, 4.44, 4., 3.33, 2.5, 1.9, 1.25 /
```

```
      DATA PSAT / 5.4, 12.5, 49.9, 121.9, 181., 188., 200. /
*
      DATA                          U   /
     *    79.,    88.,   113.,   147.,   181.,   200.,   225.,   250.,
     *   275.,   300.,   350.,   400.,   600.,  2600./
*
      DATA                          RHO /
     *  .0050,  .0125,  .0250,  .0500,  .1000,  .1250,  .2500,  .5000,
     * 1.0000, 1.2500, 1.9000, 2.5000, 3.3300, 4.0000, 4.4400, 4.6000,
     * 4.9000, 5.5000/
*
*    Pressure( su, rho )
*
      DATA    (PRES(I,01), I=1,14)  /                                    SSM76900
     *   .000,   .000,   .000,   .000,   .000,   .000,   .000,   .000,
     *   .000,   .000,   .000,   .000,  5.953, 25.800/
*
      DATA    (PRES(I,02), I=1,14)  /
     *   .000,   .000,   .000,   .000,  1.103,  1.303,  1.575,  2.229,
     *  3.358,  4.486,  6.733,  8.808, 14.890, 64.580/

                  part of table omitted
*
      DATA    (PRES(I,18), I=1,14)  /
     * 4014.0, 4715.0, 6267.0, 8170.0, 9887.0,10808.0,11949.0,13029.0,
     * 14030.0,14979.0,16704.0,18270.0,20000.0,20000.0/
C
C    TEMPERATURE AS A FUNCTION OF SU AND RHO
C
      DATA    (TEMP(I,01), I=1,14)  /
     *    .00,    .00,    .00,    .00,    .00,    .00,    .00,    .00,
     *    .00,    .00,    .00,    .00, 223.60, 968.20/
*

                  part of table omitted

      DATA    (TEMP(I,18), I=1,14)  /
     *  37.26,  44.66,  60.93,  81.27, 100.50, 110.76, 123.77, 136.08,
     * 147.70, 158.74, 179.14, 197.82, 264.75, 987.65/

      subroutine prop0
**********************************************************************
C
C        INITIALIZE SLOPES
C
      DO 26 J = 1,NRHO
        RHO(J)=RHO(J) * v1728
   26 CONTINUE
      CALL XYset( NU, U, NRHO, RHO, PRES, XP, VDRHO )
      CALL XYset( NU, U, NRHO, RHO, TEMP, TXP, VDRHO )
```

```
      DO 30 J = 2, NRHO
        drho = RHO(J) - RHO(J-1)
        DO 25 I = 2, NU
          UvsP(I,J) = 1.0 / XP(I,J)
          RHOvsP(I,J) =  drho / ( PRES(I,J) - PRES(K,J-1))
   25   CONTINUE
   30 CONTINUE
      DO 40 I = 2, NU
        du = U(I) - U(I-1)
        RSvsU(I) = ( RSAT(I) - RSAT(I-1) ) / ( du * 1728.0 )
        PSvsU(I) = ( PSAT(I) - PSAT(I-1) ) / du
        vPS(I) = 1. / ( PSAT(I) - PSAT(I-1) )
        DRSAT(I) = RSAT(I) - RSAT(I-1)
   40 CONTINUE
      RETURN
*
      ENTRY hypt( SU, SRHO, N, P, T )
*************************************************************************
*
*  Pressure and temperature from energy, density.
*  Added precomputing of pressure saturation slopes.
*
      CALL intval( icall(N), SU, NU, U,
     +         'Hydrogen internal energy is below the table.',
     +         'Hydrogen internal energy exceeds the table.', 0 )
*
      CALL intval( jcall(N), RHO, NRHO, RHO,
     +         'Hydrogen density is below the table.',
     +         'Hydrogen density exceeds the table.', 0 )
*
C           PRESSURE COMPUTATIONS
C
  200 UP1 = SU - U(I-1)
      RHOP1 = ( SRHO - RHO(J) ) * vdrho(J)
      P1 = PRES(I-1,J) + XP(I,J) * UP1
      P2 = PRES(I-1,J+1) + XP(I,J+1) * UP1

      IF(SU.LT.225.) THEN                                          SSM78700
         RHOSAT = RSAT(I-1) + RSvsU(I) * UP1
         R1=RHO(J)
         R2=RHO(J+1)
         IF((SRHO.GT.RHOSAT).AND.(RHOSAT.GT.R1)) THEN
            R1=RHOSAT
            P1 = PSAT(I-1) + PSvsU(I) * UP1
          ELSE IF( (SRHO.LT.RHOSAT) .AND. (RHOSAT.LT.R2) )THEN
            R2=RHOSAT
            P2 = PSAT(I-1) + PSvsU(I) * UP1                        SSM78800
          END IF
          P=P1+(P2-P1)*(SRHO-R1)/(R2-R1)
      ELSE
```

```
            P = P1 + (P2 - P1) * RHOP1
          END IF
*
          T = xylint( SU, SRHO, NU, U, NRHO, RHO, TXP, VDRHO, TEMP,
         +                                          icall(N), icall(N) )
          RETURN                                                    SSM78900
*
          ENTRY hyrt( SU, SRHO, N, P, T )
*****************************************************************
*
C         RHO AND T FROM U AND P        (Uses call number N + 1 )
C
          CALL intval( icall(N), SU, NU, U,
         +             'The hydrogen energy is below the table.',
         +             'The hydrogen energy exceeds the table.', 0 )
*
*  intval could not be used below, because consecutive row elements
*  of PRES are not consecutive in memory. The table escape error stop
*  was added, however.
*
          K = icall(N)
          K=I-1
          J=JCALL(N)
  310     IF(PRES(K,J).LT.P) GO TO 330
  320     IF(PRES(K,J-1).LE.P) GO TO 340
          J=J-1
          GO TO 320                                                 SSM79000
  330     J=J+1
          GO TO 310
*                              Replaced 340 J=MAX0(2,MIN0(J,NRHO)) by
  340     IF ( J .LT. 2 ) THEN
            PRINT *, 'The hydrogen pressure is below the table.'
            STOP
          ELSE IF ( J .GT. NRHO ) THEN
            PRINT *, 'The hydrogen pressure exceeds the table.'
            STOP
          END IF

          IF(K.EQ.I-1) jcall(N) = J
          R(K-I+2) = RHO(J-1) + ( P-PRES(K,J-1) ) * RHOvsP(K,J-1)
          IF(K.EQ.I) GO TO 350
          K=I
          GO TO 310
C                                                                   SSM79100
C     ASSUME RHO LINIER WITH U
C     PROVIDE FOR LIQUID SIDE OF SATURATION LINE ONLY
C
  350     U1=U(I-1)
          U2=U(I)
          IF(I.LT.8 .AND. J.LE.17-I) THEN
```

```
          X = ( P-PSAT(I-1) ) * vPS(I)
          U2 = U1 + X * ( U2 -U1 )
          R(2) = ( RSAT(I-1) + X * DRSAT(I) ) * v1728
        END IF                                                    SSM79200
        SRHO = R(1) + ( R(2)-R(1) ) / (U2-U1) * (SU-U1)
*
*   Now T is found from input SU and interpolated RHO. We use the next call
*   number to remember the previous interval of the above interpolated RHO.
*
        CALL intval( icall(N+1), SRHO, NRHO, RHO,
       +             'The interpolated density is below the table.',
       +             'The interpolated density exceeds the table.', 0 )
*
        T = xylint( SU, SRHO, NU, U, NRHO, RHO, TXP, VDRHO, TEMP,
       +                                    icall(N), icall(N+1) )
        RETURN
*
        ENTRY hyut( SU, SRHO, N, P, T )
*******************************************************************
C
C       GET U AND T FROM RHO AND P  ( Uses call number N + 1 )
C
*   Was optimized by using precomputing slopes and replacing search from
*   table edge with search from point at last call. An error stop is now
*   in effect when density or pressure leaves the tabulated range.
*   The segment was recoded so that K does not change.
*
        CALL intval( icall(N), SRHO, NRHO, RHO,
       +              'The hydrogen density rho is below the table.',
       +              'The hydrogen density rho exceeds the table.', 0 )
*
        K = icall(N)
        CALL intval( jcall(N), P, NU, PRES(1,K-1),
       +              'The hydrogen pressure is below the table.',
       +              'The hydrogen pressure exceeds the table.', 0 )
*
        U1 = U(M-1) + ( P - PRES(M-1,K-1) ) * UvsP(M,K-1)
*
        CALL intval( jcall(N), P, NU, PRES(1,K),
       +              'The hydrogen pressure is below the table.',
       +              'The hydrogen pressure exceeds the table.', 0 )
        M = jcall(N)
*
        U2 = U(M-1) + ( P - PRES(M-1,K) ) * UvsP(M,K)
*
        SU = U1 + ( SRHO-RHO(K-1) )* vdrho(K-1) * (U2 - U1)
*
*   Now T is found from interpolated SU and input RHO. We use the next call
*   number to remember the previous interval of the above interpolated SU.
*
```

```
      CALL intval( icall(N+1), SU, NU, U,
     +             'The interpolated energy is below the table.',
     +             'The interpolated energy exceeds the table.', 0 )
*
      T = xylint( SU, SRHO, NU, U, NRHO, RHO, TXP, VDRHO, TEMP,
     +                  icall(N+1), icall(N) )
      END
```

'h2gama.for':

```
      FUNCTION H2GAM0( XPRES, XTEMP, N)
************************************************************************
*
* Initializes temp, pressure points, returns value at XPRES, XTEMP
*
C
C  PURPOSE:  COMPUTATION OF HYDROGEN SPECIFIC HEAT RATIO
C            AS A FUNCTION OF PRESSURE AND TEMPERATURE
C            CALCULATES GAMMA FOR PARA HYDROGEN
C
C******ARGUMENTS******
C  INPUT:
C   XPRES  = PRESSURE, PSI
C   XTEMP  = TEMPERATURE, DEG R                                    SSM79600
C   N      = CALLER NODE INDEX
C
C  OUTPUT:
C   H2GAMA = GAMMA
C
*   H2GAMA0 does initialization and lookup ( NN .LE. 0 )
*   The main entry H2GAMA does lookup alone.
************************************************************************
*
      DIMENSION TEMP(20),PRES(25),GAMA(20,25),I1(10),J1(10),XP(20,25), 9700
     1DPRES(25)
*
      INCLUDE 'units.com'
*
************************************************************************
*
* Reads temperature and pressure points, tablulated values.
*
      READ(dat,11)NTEMP,NPRES,(TEMP(I),I=1,NTEMP),(PRES(I),I=1,NPRES)
   11 FORMAT( //2X, 2(2X,I10)/(//2X,6(2X, G10.0)) )                 SSM79750
      READ(dat,12)((GAMA(I,J),I=1,NTEMP),J=1,NPRES)
   12 FORMAT(/(//3X,12(1X,F6.0)     ) )                            SSM79770
*
* Precomputation of slopes for two-way interpolation was moved to the
* interpolation module.
```

```
*
      CALL XYset( ntemp, temp, npres, pres, gama, xp, dpres )
      DO 23 I=1,10
         I1(I)=2
         J1(I)=2                                                    SSM80000
   23 CONTINUE
*
      ENTRY H2GAMA( XPRES, XTEMP, N )
*************************************************************************
*
*  This is the simulation loop entry, with no initialization.
*  Keeps track of last entry location for each caller, identified by N.
*
*  If it is really desired to return H2GAMA = 1.4 temperature is off the
*  table, then intval can be used instead of the special code below, by
*  adding one point to TEMP, and a column to GAMA defining the range where
*  this return is OK.
*
*************************************************************************
*
      I = I1(N)
      IF( xtemp .GT. temp(i) ) THEN                                 SSM80100
*       Search from there up
        DO 10 k = i + 1, ntemp
           IF( xtemp .LE. temp(k) )THEN
              i = k
              GO TO 30
           ENDIF
   10   CONTINUE
*                                Here, xtemp is above the table
        i = npres
        H2gama = 1.4
        RETURN
      ELSEIF ( xtemp .LT. temp(i - 1)  ) THEN
*     Search down from there
        DO 20 k = i - 2, 1, -1
           IF( xtemp .GE. temp(k) )THEN
              i = k + 1
              GO TO 30
           ENDIF
   20   CONTINUE
        PRINT *, 'Temperature is below H2GAMA table.'
        STOP
      ENDIF
*
*  Error traps were added in case input variables escape the table.
*  The table can be readily extended, after all.
*
   30 CALL intval( j1(N), xpres, npres, pres,                       SSM80200
     +             'Pressure is below H2GAMA table.',
```

```
     +              'Pressure is above H2GAMA table.', 0 )
      H2GAMA = xylint( xtemp, xpres, ntemp, temp, npres, pres,
     +                 xp, dpres, gama, i, j1(n) )
*
*  And to support H2GAM0 as a function:
*
      H2GAM0 = H2GAMA
*
      END
```

```
      SUBROUTINE OXIDF0
C*****************************************************************************
C
C   THIS IS ANOTHER MAJOR PART OF THE SIMULATION PROGRAM.  THIS SIMULATES
C   THE OXIDIZER FLOW AND ENERGY BALANCE OF THE SYSTEM.
C   THE SUBROUTINE FLOW IS:
C   1) INITIALIZATION                          ( OXIDF0 )
C   2) READ DATA                               ( OXIDF0 )
C   3) CALCULATE THE PUMP INLET AND OUTLET DUCTS CONDITIONS  ( OXIDF0 )
C   4) CALCULATE THE FLOW RATE OF EACH NODE
C   5) CALCULATE THE PUMP/TURBINE PERFORMANCE
C   6) CALCULATE THE PUMP CAVITATION EFFECT
C   7) CALCULATE THE CHANGES OF THE OXID LINE FROM TANK TO LPOP
C   8) PRIMING FUNCTION OF MOV AND INJECTOR
C   9) CALCULATE THE PUMP SPEEDS
C      10) RETURN
C
C*****************************************************************************
C
      DIMENSION RIL(12),ELENO(12),ZIL(12),ZIC(12), TPR(6)
      DIMENSION N1(3), N4(3), N5(3), N6(3)
      LOGICAL adjOK, XMOVPF, PRIMIF
C
      COMMON/PVCHEK/ POTVP, DWOP1L, DUMME1, DUMME2, INONZ,
     2               POSX,TAU,ABKFLO,FHECD1,FHECD2
     3     ,FLI2,FL2,PCOD1L,PCOD2L,POD1L,POD2L,POI2L,VHECD1,VHECD2
C                                                                      SSM57000
      INCLUDE 'blank.com'
      INCLUDE 'out.com'
      INCLUDE 'contrl.com'                                             SSM57100
      INCLUDE 'igni.com'
      INCLUDE 'oxid.com'
      INCLUDE 'hgas.com'
      INCLUDE 'balc.com'
      INCLUDE 'pogo.com'
*
      COMMON/PURGE/DWGN2,TCUTPR,DWFN2F,DWGN2O
*
*  Often used constant reciprocals:
*
      PARAMETER ( v386p4 = 1. / 386.4, v9336 = 1. / 9336.,
     +            v41p34 = 1. / 41.34, v8p866 = 1. / 8.866,
     +            v77 = 1. / 77. )
      PARAMETER ( TooBig = 1.E50 )
C                                                                      SSM57600
*  Obsoleted printout control
*     DATA TPR   / 10*100.0 /, IPR   / 2 /, FLAG  / 0.0 /
*
      DATA PRIMIF / .TRUE. /, QBKFL2 /0.0/, XMOVPF / .TRUE. /          SSM57800
      DATA TL     /0.0/, TH     /0.0/
```

```
      DATA ROP3IN /0.0/
      DATA    WTOP1 / 14.4 /
*
      recpos(x) = AMAX1( 0.0, x )
      recneg(x) = AMIN1( 0.0, x )
      rlimit(x, floor, ceiling) = AMAX1( floor, AMIN1( ceiling, x ) )
C
C   WTHE() IS THE FUNCTION TO CALCULATE THE HELIUM CONTENT OF A FLOW MIXER.
C
      WTHE(DWHE1, FHE1, FHE2, DWHE2, FHE3, DWHE3, FHE4) =
     + FHE2*( recneg( DWHE1 ) - recpos( DWHE2 ) + recneg( DWHE3 ) ) +
     + FHE1*recpos( DWHE1 ) - FHE3*recneg( DWHE2 ) + FHE4*recpos( DWE3 )
C
C   ENINOX() IS THE FUNCTION TO CALCULATE THE ENTHALPY OF AN ACCUMULATOR.
C
      ENINOX(WI, WO, H1, HO, HU) = H1 * recpos( WI )  +
     +     HO * ( recneg( WI ) - recpos( WO ) ) - HU * recneg( WO )SSM57900
*
*   Name this function appropriately
*
        quadr( x ) = 620.15 + x * ( 19.1202 + .149371 * x )
*
*********************************************************************
C
      READ(run,30)POT,ZOS,DUM,RHOOT,ZMOV,ZOP1,ZOP2,TQOT1B,TQOT2B,ZOI
     1,TOS,WOV,TDRAGO
      READ(run,30)GO1,GO2
      READ(run,30)ELOT1,AAOT1,RMOVUG,RMOVD,VOLOP1,VOLOP2,VOLOP3,VOLOT1
     *,WCOD,WOTD
      READ(run,30)PVOP1,PVOP2,PVOP3,PCOP1,PCOP2,PCOP3
      READ(run,30) (ELENO(J),J=1,12),(ZIL(J),J=1,12),(ZIC(J),J=1,12),
     1(RIL(J),J=1,12)
      READ(run,30) AOS,AHE,XLO,XL1,XL2,VTOT,PHES,GAMHE
   30 FORMAT(//2X,6G12.4)
      READ(run,'(//2X,3I12)') INONZ,IRHOP2,ICAVMD
      READ(run,30)TCUT2,TCUT3,BASEAR,BOS,TDCOM,RL1,RPV,ZJTPV,ROP3IN,
     1ELJPV,BOJ,BOPVDN,WOCOM,WOIN,DTPV,TCLPV,ABK2,THEADD,VOLOD1,
     2VOLOI2,VOLOD2,VOLCD1,VOLCD2,TCAV1,TCAV2,TIMCAV,GLOPC,TL1,TH1,
     3TL2,TH2
      DUCTS=1.
      ELENT=0.0
      FAC = 8.0
      THE = 180.                                                    SSM58100
      HOI2AD = -41.
C
      DUMME1= WOCOM
      DUMME2= WOIN
C
*      STIME=TIME
      CALL fgset( 5 )
```

```
      HOS = FGEN(5, 8, STIME)
      CALL fgset( 37 )                                                  SSM58600
      POT = FGEN(37, 9, STIME)
      PIL(1) = POT
      DO 90 J= 2,8
        PIL(J)=PIL(J-1)+ELENO(J-1)*RHOOT
        DWIL(J)=0.0
   90 CONTINUE
      DWIL(1)=0.0
      POS = PIL(8)+ELENO(8)*RHOOT
      POSX=POS
      POD1=POS
      POD2=POS                                                          SSM58700
      POD3=POS
      POI2=POS
      POINJ=PA
      SO1= 1.0E-10
      SO2 = 1.0E-10
      so1sq = SO1 ** 2
      so2sq = SO2 ** 2
      WOI=0.0
      DWOI=0.0
      DWOP1=0.0
      DWOP2=0.0
      DWOP3 = 0.0                                                       SSM58800
      DWCOD=0.
      DWMOV=0.0
      DWOT1=0.0
      DWOP2C=0.0
      DWOP3C=0.
      DDX=386.4
      DPOP1=0.0
      DPOP2=0.0
      DPOP3=0.0
      TRQOP1 = 0.0                                                      SSM58900
      TRQOP2 = 1.0E-10
*                    prior function loads created unused variables
      CALL fgset( 44 )
      CALL fgset( 45 )
      CALL fgset( 46 )
      CALL fgset( 47 )
      CALL fgset( 48 )
      CALL fgset( 49 )
      CALL fgset( 50 )
      CALL fgset( 58 )
      ELCOM = fgen(58, 29, 0.)
      CALL fgset( 59 )
      RCOM = FGEN(59, 30, 0.)
      CALL fgset( 60 )                                                  SSM59000
      ZCOM = FGEN(60, 31, 0.)
```

```
      CALL fgset( 61 )
      ELOIN = FGEN(61, 32, 0.)
      CALL fgset( 62 )
      ROIN = FGEN(62, 33, 0.)
      CALL fgset( 63 )
      ZOIN = FGEN(63, 34, 0.)
      CALL fgset( 64 )
      APV = FGEN(64, 35, 0.)
      CALL fgset( 65 )
      DWOE2 = FGEN(65, 36, 0.0)
      CALL fgset( 66 )
      CALL fgset( 67 )
      CALL fgset( 68 )
      CALL fgset( 69 )
      CALL fgset( 70 )
      CALL fgset( 71 )
      CALL fgset( 72 )
      CALL fgset( 73 )
      CALL fgset( 74 )
      CALL fgset( 75 )
      CALL fgset( 76 )
      CALL fgset( 77 )
      CALL fgset( 78 )
      CALL fgset( 79 )
      CALL fgset( 81 )
      CALL fgset( 82 )
C     CALL O2DFPE(HOS,POS,RHOOS,TOS,4,-1,1)
      CALL O2PROP(HOS,RHOOS,1,POS,TOS,-1)
      CALL O2PROP(HOS,RHOOS,1,POS,TOS,3)
      TOD2=180.
      RHOP1=RHOOS
      RHOOP1=RHOOS
      RHOP2=RHOOS
      RHOOP2=RHOOS
      RHOP3=RHOOS                                                  SSM59300
      RHOOP3=RHOOS
      RHOD1=RHOOS
      RHOD2=RHOOS
      RHOT1=RHOOS
      RHOI2     = RHOOS
      RHOCD2    = RHOOS
      fgset ( 3 )                                                  SSM58000
      PRIMOI = FGEN(3, 37, 0.0)
*      ZOT1 = ELOT1 / ( AAOT1 * 386.4)
      hZOT1 = ELOT1 / ( AAOT1 * 772.8)
      ROT1N=ANOT1                                                  SSM59400
      COV=1.0
      HCAVP1=1.
      HCAVP2=1.
      HCAVP3=1.
```

```
      TCAVP1=1.
      TCAVP2=1.
      TCAVP3=1.
      WHEOD1=0.0
      FHEOD1=0.0
      HOS1=HOS                                                    SSM59500
      WHECD1=0.0
      FHECD1=0.0
      WHECD2=0.0
      FHECD2=0.0
      WHEOI2=0.0
      FHEOI2=0.0
      WHEOD2=0.0
      FHEOD2=0.0
      WHEOT1=0.0
      FHEOT1=0.0                                                  SSM59600
      DWOTJ=0.
      DWOE3=0.
      DWOPV=0.
      RJTPVD=3.05E-6
      DWOIN=0.
      POINVP=15.
      VOLPV=0.
      QBKFLO=0.
      HLPT1=HOS
      HIOP2=HOS                                                   SSM59700
      H3I=HOS
      H3=HOS
      HOD2=HOS
      HOD1=HOS
      HOD3=HOS
      HOT1=HOS
      HOI2=HOS
      HLPOTD=HOS
      CALL POGO0
      WRITE(init,100)(PIL(J),J=1,8)
  100 FORMAT( ' PIL(1)     PIL(2)     PIL(3)     PIL(4)     PIL(5)
     1  PIL(6)     PIL(7)     PIL(8)' /1P8E11.3)
*
*  Initialization was extended to include the following:
C
*     IF (FLAG .GT. 0.0) GO TO 1160       obsoleted method of
*      FLAG      = 100.0                  triggering restart adjustment
*
      IF ( DWOP1 .GE. 0. ) HOP = HOS
      VOLOI2 = VOLOI2 + VOLCD1 + VOLCD2

*     IF (TIME .GT. 0.001) GO TO 1160   Apparently an error. It calls for
*   adjustment to occur only if start time < .001
*
```

```
      IF ( STIME .LE. 0.001 ) RETURN
*
      POT        = FGEN (37, 9, STIME)
      HOS        = FGEN (5, 8, STIME)
      ELCOM      = FGEN (58, 29, WOCOM)                                SSM59900
      ELOIN      = FGEN (61, 32, VOLPV)
      IF (TCUT2 .LT. STIME)  DWOE2 = 0.0
      IF (TCUT3 .LT. STIME)  DWOE3 = 0.0
      RHOLO      = -1.0
      RHOHI      = -1.0
      ICOUNT     = 0
C                                                                     SSM59800
C   THE FOLLOWING "ERROR CHECKING" (FROM HERE TO STATEMENT 1100)
C   IS TO FIND THE CONSISTANCY AMONG THE GIVEN INITIAL CONDITIONS
C   OF THE OXID LINE INPUTS, POS (OXID INPUT PRESSURE), SU (INTERNAL
C   SPECIFIC ENERGY), AND RHOOS (OXID LINE INPUT DENSITY).  IF THE CALCULATE
C   VALUE OF PRESSURE FROM SU AND RHOOS IS NOT TOO FAR OFF FROM "POS" THEN
C   RHOOS IS ADJUSTED TO MATCH THE CONDITION.
*
*   The adjustment loop described above is executed when restarting
*   at nonzero time. Its omission from the restart procedure may account
*   for the restart transients mentioned elsewhere. Reporting on the
*   restart convergence is handled by the change monitor.
*
      CALL chg0( .0001, 1 )
      adjOK = .TRUE.
      DO 10, I = 1, 30
        POS = POT + RHOOS * DDX * (ELCOM + ELOIN) *v386p4             SSM60000
        SUt = HOS - POS / (9336.0 * RHOOS)
        CALL OXPROP(PANS, HOX, RHOOS, RHOLI2, RHOGI2,
     +                              HGI2, HLI2, FLI2, SUt, N4)
        CALL chgmnt( adjOK, PANS, POS, 1 )
        IF ( adjOK ) GO TO 1100
        IF (PANS .LE. POS) THEN                                       SSM60100
          RHOLO = RHOOS
          IF (RHOHI .LE. 0.0 ) THEN
             RHOOS = 1.002* RHOLO
          ELSE
             RHOOS = (RHOLO + RHOHI)* 0.5
          ENDIF
        ELSE
          RHOHI = RHOOS
          IF (RHOLO .LE. 0.0) THEN
             RHOOS = 0.998* RHOHI
          ELSE
             RHOOS = (RHOLO + RHOHI)* 0.5
          ENDIF
        ENDIF
   10 CONTINUE                                                        SSM60200
      CALL wrchg ( init, 1, 'OXIDF initial adjustment error' )
```

```
*
*  The following re-initializations are necessary because of the restart
*  adjustment.
*
 1100 CONTINUE
      POJ       = POT + RHOOS * DDX * ELCOM * v386p4
      POPVDN    = POJ + RHOOS * DDX * ELJPV * v386p4
      SUOIN     = HOS - POS / (9336.0 * RHOOS)
      UOIN      = SUOIN * WOIN
      HOD1      = HOS                                              SSM60300
      SUOD1     = SUOIN
      RHOD1     = RHOOS
      WOD1      = VOLOD1 * RHOOS
      UOD1      = SUOIN * WOD1
      POD1      = POS
      POD1L     = POS
      HOI2      = HOS
      SUOI2     = SUOIN
      RHOI2     = RHOOS
      WOI2      = VOLOI2 * RHOOS                                   SSM60400
      UOI2      = SUOIN * WOI2
      POI2      = POS
      POI2L     = POS
      HOD2      = HOS
      SUOD2     = SUOIN
      RHOD2     = RHOOS
      WOD2      = VOLOD2 * RHOOS
      UOD2      = SUOIN * WOD2
      POD2      = POS
      POD2L     = POS                                              SSM60500
      HOT1      = HOS
      SUOT1     = SUOIN
      RHOT1     = RHOOS
      WOT1      = VOLOT1 * RHOOS
      UOT1      = SUOIN * WOT1
      POT1      = POS
      POT1L     = POS
      HOD3      = HOS
      POD3      = POS
      RHOCD2    = RHOOS                                            SSM60600
*
      CALL unint0( UOD1, 58 )
      CALL unint0( WOD1, 59 )
      CALL unint0( WHEOD1, 60 )
      CALL unint0( UOI2, 61 )
      CALL unint0( WOI2, 62 )
      CALL unint0( WHEOI2, 63 )
      CALL unint0( UOD2, 64 )
      CALL unint0( WOD2, 65 )
      CALL unint0( WHEOD2, 66 )
```

```
      CALL unint0( UOT1, 67 )
      CALL unint0( WOT1, 68 )
      CALL unint0( WHEOT1, 69 )
      CALL lmint0( DWOP2, 74, 0.0, TooBig )
      CALL lmint0( DWOT1, 75, 0.0, TooBig )
      CALL lmint0( DWOP2, 76, 0.0, TooBig )
      CALL lmint0( HCAVP1, 77, 0.0, 1.0 )
      CALL lmint0( TCAVP1, 78, 0.0, 1.0 )
      CALL lmint0( HCAVP2, 79, 0.0, TooBig )
      CALL lmint0( TCAVP2, 80, 0.0, TooBig )
      CALL lmint0( HCAVP3, 81, 0.0, TooBig )
      CALL lmint0( TCAVP3, 82, 0.0, TooBig )
      CALL lmint0( WOCOM, 83, 0.0, TooBig )
      CALL lmint0( DWOPV, 84, 0.0, TooBig )
      CALL unint0( POPVDN, 85 )
      CALL lmint0( VOLPV, 86, 0.0, TooBig )
      CALL unint0( UOIN, 87 )
      CALL unint0( WOIN, 88 )
      CALL unint0( POSX, 89 )
      CALL unint0( HOS1, 90 )
      CALL lmint0( WOI, 91, 0.0, TooBig )
      CALL lmint0( WOV, 91, - TooBig, 0.001 )
      CALL lmint0( SO1, 93, 1.0E-10, TooBig )
      CALL lmint0( SO2, 94, 1.0E-10, TooBig )
*
* 1160 CONTINUE            for reference to original
      RETURN
*
      ENTRY OXIDF
*************************************************************************
C
C   THE CALCULATION OF OXID FLOW IS IN GENERAL THE BALANCE OF ENERGY FLOW
C   AND THE BALANCE OF MATERIAL.  IN THIS SECTION AND ALSO FOLLOWING
C   SECTIONS, THE NOTATIONS ARE AS FOLLOWS:
C   Pxyz:    PRESSURE OF DUCT xyz
C   RHOxyz: DENSITY OF DUCT xyz
C   Hxyz:    ENTHALPY OF DUCT xyz
C   SUxyz:   SPECIFIC HEAT OF DUCT xyz
C   Uxyz:    TOTAL INTERNAL ENERGY OF DUCT xyz
C   DWxyz:   FLOW OF DUCT xyz
C   Wxyz:    MASS OF DUCT xyz
C   WHExyz: HELIUM WEIGHT OF DUCT xyz (BECAUSE OF POGO SYSTEM, HELIUM DOES
C                  APPEAR IN THE OXIDIZER SUPPLY DUCTS)
C   VHExyz: HELIUM VOLUMN OF DUCT xyz
C   FHExyz: FRACTION OF HELIUM INDISE DUCT xyz
C
C      WHERE xyz CAN BE ONE OF THE FOLLOWING:
C   OS:      OXIDIZER SUPPLY
C   OD1:     LPOP OUTLET DUCT (COD IS ALSO USED FOR FLOW)
C   OI2:     HPOP INLET DUCT
```

```
C   OD2:    HPOP OUTLET DUCT
C   OT1:    LPOT INLET DUCT
C
C   BECAUSE OF THE HIGH OPRATION TEMPERATURE AND ENERGY, THE OXIDIZER CAN BE
C   WORKING UNDER TWO PHASE CONDITIONS.  FUNCTION OXPROP() IS CALLED TO
C   CALCULATE THE CONDITIONS OF GAS AND LIQUID PHASES OF OXIDIZER.
C*********************************************************************
C
C     LPOP DISCHARGE DUCT
C
C   LPOP DISCHARGE DUCT "OD1" IS A MERGE POINT OF BOTH LPOP AND LPOT.
C   THE INPUT FLOWS ARE DWOP1 AND DWOT1.  AND THE OUTPUT FLOW IS DWCOD.
C   ABKFLO IS THE ENERGY BACK FLOW RATIO TO THE UPSTREAM OF THE PUMP.
C
      IF(DWOP1.LE.0.)HOP=HOD1
      HXD1=HOD1
      IF (DWCOD .LT. 0.0) HXD1 = HOI2
      UOD1 = pruint( DWOP1*HOP + (1. - ABKFLO)*TRQOP1*SO1*v9336
     1 + DWOT1*HOT1 - DWCOD*HXD1 - TRQOT1*SO1*v9336, 0, 58)          SSM60700
      WOD1 = pruint( DWOP1 + DWOT1 - DWCOD, 0, 59 )
      SUOD1 = UOD1 / WOD1
      WHEOD1 = pruint(
     +  WTHE(DWOP1, 0.0, FHEOD1, DWCOD, FHEOI2, DWOT1, FHEOT1), 0, 60 )
      FHEOD1 = WHEOD1 / WOD1
*     VHEOD1 = .99*VHEOD1 + .01*WHEOD1 * 4632.*THE/AMAX1(POD1L-3.,
      VHEOD1 = .99*VHEOD1 + 46.32 * WHEOD1 * THE /
     +  rlimit( POD1L-3., POD1L+3., POD1)
      RHOD1 = WOD1 / AMAX1(1., VOLOD1 - VHEOD1)
      RHOP1 = WOD1 / VOLOD1
      POD1L = POD1                                                    SSM60800
      CALL OXPROP(POD1,HOD1,RHOD1,RHOLD1,RHOGD1,HGD1,HLD1,FLD1,SUOD1,N1)
      HOD1 = SUOD1 + POD1 / (RHOD1 * 9336.)
C
C        HPOP INLET DUCT
C
C   THIS IS THE DUCT RIGHT BEFORE THE HIGH PRESSURE OXID PUMP.
C   THE INPUTS TO THIS DUCT ARE:
C   DWCOD:  FROM LPOP OUTLET DUCT
C   DWOP2C: BYPASS FLOW FROM DOWNSTREAM OF HPOP BOOSTER STAGE
C   DWOP3C: BYPASS FLOW FROM DOWNSTREAM OF HPOP
C   DWGOP:  OXID FLOW FROM POGO SYSTEM IN GAS PHASE (CAN BE + OR -)
C   THE OUTPUTS OF THIS DUCT ARE:
C   DWO:    OXID FLOW TO POGO SYSTEM IN LIQUID PHASE (CAN BE + OR -)
C   DWOP2:  HPOP FLOW
C
      UOI2 = pruint( DWCOD*HOD1 + DWOP2C*HOD3 + DWOP3C*HOD2
     1 - DWO*HOI2 - DWOP2*HOI2 + 75.0*DWGOP + 1.25*(530.-TGAS)*DWHOP
     2 + QBKFL2*WOI2 / AMAX1(10.0, WT2BK), 0, 61 )
      WOI2 = pruint(
     +       DWCOD + DWOP2C + DWOP3C - DWO - DWOP2 + DWGOP, 0, 62 ) SSM60900
```

```
      SUOI2 = UOI2 / WOI2
      WHEOI2 = pruint(
     +  WTHE (DWCOD, FHEOD1, FHEOI2, DWOP2, FHEOD2, DWHOP, 1.0), 0, 63 )
      FHEOI2 = WHEOI2 / WOI2
      VHEOI2 = .99*VHEOI2 + 46.32 * WHEOI2 * THEADD /
     +   rlimit( POI2L - 3., POI2L + 3., POI2))
      RHOI2 = WOI2 / AMAX1(1., VOLOI2 - VHEOI2)
      FHEOI2 = WHEOI2 / WOI2
      POI2L=POI2
      CALL OXPROP(POI2,HOI2,RHOI2,RHOLI2,RHOGI2,HGI2,HLI2,FLI2,    SSM61000
     1 SUOI2,N4)
      HOI2 = SUOI2 + POI2 / (RHOI2 * 9336.)
C
C       HPOP DISCHARGE DUCT TO MOV
C
C    THIS DUCT IS THE CONNECTION BETWEEN HPOP AND MOV.
C    THE INPUTS TO THIS DUCT ARE:
C    DWOP2:  HPOP FLOW
C    THE OUTPUTS FROM THIS DUCT ARE:
C    DWOP3:  OXID FLOW TO HPOP BOOSTER STAGE
C    DWOP3C: BYPASS FLOW BACK TO HPOP INLET DUCT
C    DWMOV:  OXID FLOW TO MAIN OXID VALVE
C    DWOT1I: FLOW TO HPOT INLET DUCT
C
      UOD2 pruint( DWOP2*HOI2 + TRQOP2*SO2*v9336 -
     +             (DWOP3 + DWOP3C + DWMOV + DWOT1I)*HOD2 -
     +              QBKFL2*WOI2/AMAX1(10., WT2BK), 0, 64 )
      WOD2 = pruint( DWOP2 - DWOP3 - DWOP3C - DWMOV - DWOT1I, 0, 65 )
      SUOD2 = UOD2 / WOD2
      WHEOD2 = pruint(
     +   WTHE(DWOP2,FHEOI2,FHEOD2,DWOT1I,FHEOT1,-DWMOV,FHEOD2), 0, 66 ) 100
      FHEOD2 = WHEOD2 / WOD2
      VHEOD2 = .995*VHEOD2 + 23.16 * WHEOD2 *THE /
     +         rlimit( POD2L - 3., POD2L + 3., POD2)
      RHOD2 = WOD2 / AMAX1(1., VOLOD2 - VHEOD2)
      vRHOD2 = 1. / RHOD2
      RHOP2=WOD2/VOLOD2
      IF (IRHOP2 .EQ. 1) RHOP2= WOI2/VOLOI2
      POD2L=POD2
      CALL OXPROP(POD2,HOD2,RHOD2,RHOLD2,RHOGD2,HGD2,HLD2,FLD2,
     1 SUOD2,N5)                                                    SSM61200
      HOD2 = SUOD2 + POD2 /(RHOD2 * 9336.)
C
C       LPOP TURBINE SUPPLY DUCT
C
C
C    THIS IS A SIMPLE ONE INPUT ONE OUTPUT DUCT FROM HPOP OUTLET TO LPOT INLE
C    INPUT TO THE DUCT:
C    DWOT1I: FROM HPOP OUTLET
C    OUTPUT FROM THE DUCT:
```

```
C    DWOT1:  TO HPOT INLET
C
       UOT1 = pruint( DWOT1I*HOD2 - DWOT1*HOT1, 0, 67 )
       WOT1 = pruint( DWOT1I - DWOT1, 0, 68 )
       SUOT1=UOT1/WOT1
       WHEOT1 = pruint(
      +  WTHE(DWOT1I,FHEOD2,FHEOT1,DWOT1,FHEOD1,0.0,0.0), 0, 69 )
       FHEOT1=WHEOT1/WOT1
       VHEOT1 = .99*VHEOT1 + 46.32*WHEOT1*THE /
      +     rlimit( POT1L - 3., POT1L + 3., POT1 )                    SSM61300
       RHOT1 = WOT1 / AMAX1(1., VOLOT1 - VHEOT1)
       RHOT1A=WOT1/VOLOT1
       POT1L=POT1
       CALL OXPROP(POT1,HOT1,RHOT1,RHOLT1,RHOGT1,HGT1,HLT1,FLT1,
      1 SUOT1,N6)
       HOT1 = SUOT1 + POT1 / (RHOT1 * 9336.)
C
C       HPBP DISCHARGE
C                                                                     SSM61400
C    THE HPBP DISCHARGE DUCT HAS:
C    INPUTS TO THE DUCT:
C    DWOP3:  HPOP BOOSTER PUMP FLOW
C    OUTPUTS FROM THE DUCT:
C    DWOP2C: BYPASS FLOW BACK TO HPOP INLET DUCT
C    DWOPOV: OXID SUPPLY TO OXID PREBURNER OXID CONTROL VALVE
C    DWFPOV: OXID SUPPLY TO FUEL PREBURNER OXID CONTROL VALVE
C
C    HOWEVER, FOLLOWING EQUATION USES AN EMPERICAL EQUATION TO ESTIMATE
C    THE NEXT STATE OF SPECIFIC ENTHALPY.  THE EXECT MEANING OF THE EQUATION
C    IS UNKNOWN.  BUT IT SEEMS TO CONSIDER THE TIME DELAY OF THE DUCT AND
C    USE DT/0.2 AS A SMOOTH FACTOR.
C
*      HOD3=HOD3+DT/.2*(TRQOP3*SO2/(AMAX1(.3*SQRT(AMAX1(100.,DPOP3)),
*     1 DWOP3)*9336.)-HOD3+HOD2)
       HOD3 = pruint( 5.*( TRQOP3*SO2/
      +  ( AMAX1( .3*X10th( AMAX1(100., DPOP3), 5 ), DWOP3 )*9336.) -
      + HOD3 + HOD2 ), 0, 70 )
       POD3 = POD2 + DPOP3 - DWOP3 * ABS(DWOP3) * ROP3IN / RHOP2
C      CALL O2DFPE(HOD3,POD3,RHOP3,TOD3,3,1,1)
       CALL O2PROP(HOD3,RHOP3,1,POD3,TOD3,3)
C
C       FLOW RATES
C
C    ALL THE OXID FLOW CALCULATED EXCEPT OPOV, FPOV AND POGO SYSTEM.
C
       DWOP1L=DWOP1
       CALL unint0( HOD3, 70 )
       CALL lmint0( DWOP1, 71, 0.0, TooBig )
       DWOP1 = prflow( DWOP1, ZOP1 + ZOIN, -.4*ROCOD/RHOP1 - ROIN,
      +     POPVDN + ELOIN*RHOOS*DDX*v386p4 + DPOP1 - POD1, 71 )      SSM61500
```

```
      rorhoi = - ROCOD / RHOI2
      CALL lmint0( DWCOD, 72, 0.0, TooBig )
      DWCOD = prflow( DWCOD, ZOCOD, rorhoi, POD1 - POI2, 72 )
      IF ( DWOP2.GE.0.) THEN
      CALL lmint0( DWOP2, 73, 0.0, TooBig )
        DWOP2 = prflow( DWOP2, ZOP2, rorhoi, POI2 + DPOP2 - POD2, 73 )
      ELSE
        DWOP2 = prflow( DWOP2, ZOP2, -.3*ROCOD*vRHOD2, POI2,
     +                                              DPOP2 - POD2, 73 )
      ENDIF
      DWOT1I = prflow( DWOT1I, hZOT1, -ROT1F*vRHOD2, POD2 - POT1, 74 )
      DWOT1 = prflow( DWOT1, hZOT1, -ROT1N/RHOT1A, POT1 - POD1, 75 )
      ROIX = ROI * PRIMOI
      DWMOV = prflow( DWMOV, ZMOV,
     +    - (COV*RMOV + RMOVL + ROIX)*vRHOD2, POD2-PCIE, 76 )        SSM61600
      DWOP3 = DWOPO + DWFPO + DWOP2C
      DWOP2C = X10th( recpos( (POD3 - POI2)*RHOP3/ROP2C ), 5 )
      DWOP3C = X10th( recpos( (POD2 - POI2)*RHOD2/ROP3C ), 5 )
      PHIOT1 = SO1 / (DWOT1+1.E-10)
*     ROT1N=ANOT1+BNOT1*PHIOT1-CNOT1*PHIOT1**2
      ROT1N = ANOT1 + PHIOT1*( BNOT1 - CNOT1*PHIOT1 )
      PROT1=POD1/AMAX1(POT1,.01)
      DWO1A = recpos( DWOP1 )
C
C     PUMP/TURBINE PERFORMANCE
C                                                                    SSM61700
C   IN THIS PUMP/TURBINE SECTION, THE NOTATIONS ARE:
C   FLOCOE: FLOW COEFFICIENT
C   DPxyz:  PRESSURE CHANGE ACROSS THE PUMP/TURBINE
C   TRQxyz: TORQUE REQUIRED
C   CDPxyz: PRESSURE RAISE COEFFICIENT (CONSTANT, PUMP CHARACTERISTICS)
C   CTQxyz: TORQUE COEFFICIENT (CONSTANT, PUMP CHARACTERISTICS)
C   HCAVxyz: CAVITATION FACTOR FOR PRESSURE RAISE
C   TCAVxyz: CAVITATION FACTOR FOR TORQUE
C   WHERE xyz IS ONE OF THE FOLLOWING
C   OP1 (OR P1): LOW PRESSURE OXID PUMP
C   OP2 (OR P2): HIGH PRESSURE OXID PUMP
C   OP3 (OR P3): HIGH PRESSURE OXID PUMP BOOSTER STAGE
C                                                    was 1/3 optimized
      rhoso1 = RHOP1 * SO1
      FLOCOE = AMAX1(-39.5, DWOP1 / rhoso1)
      rhoso = rhoso1 * SO1
      DPOP1 = FGEN(45, 38, FLOCOE) * rhoso * CDPOP1 * HCAVP1
      TRQOP1 = FGEN(46, 39, FLOCOE) * rhoso * CTQOP1 * TCAVP1
      rhoso2 = RHOP2 * SO2
      FLOCOE = DWOP2 / rhoso2
      rhoso = rhoso2 * SO1
      DPOP2 = FGEN(47, 40, FLOCOE) * rhoso * CDPOP2 * HCAVP2
      TRQOP2 = FGEN(48, 41, FLOCOE) * rhoso * CTQOP2 * TCAVP2
      rhoso3 = RHOP3 * SO2
```

```
      FLOCOE = DWOP3 / rhoso3
      rhoso = rhoso3 * SO2
      DPOP2 = FGEN(49, 42, FLOCOE) * rhoso * CDPOP3 * HCAVP3
      TRQOP2 = FGEN(50, 43, FLOCOE) * rhoso * CTQOP3 * TCAVP3
*
*  replaces typical
*      DPOP3=FGEN(49,2,DWOP3/(RHOP3*SO2) )*RHOP3*SO2**2*CDPOP3*HCAVP3
*      TRQOP3=FGEN(50,2,DWOP3/(RHOP3*SO2) )*RHOP3*SO2**2*CTQOP3*TCAVP3
*
*  Eliminated formatted output here
*
      IF ( GLOPC .GT. 0. ) THEN
        DDX = 386.4 * GLOPC * PCIE
      ELSE
        DDX = 386.4
      ENDIF
C                                                                    SSM62000
C     PUMP CAVITATION DESCRIPTIONS
C
C    PUMP CAVITATION IS RATHER COMPLICATED IN NATURE.  SINCE I DON'T HAVE THE
C    DOCUMENT OR NBS TABLE ON HAND, I CAN ONLY FOLLOW THE CODE AND TRY TO
C    EXPLAIN WHAT IS MEANT BY THE PROGRAM.
C
C    THE FIRST SECTION OF THE PROGRAM SEEMS TO FIND THE CAVITATION STATUS
C    OF LPOP AT LOW OR NO FLOW CONDITIONS.  THIS IS THE SITUATION WHEN THE
C    ENGIN IS SHUTTING DOWN AND THE OXID PREVALVE IS CLOSING.
C    (PREVALVE IS THE VALVE BEFORE THE LPOP INPUT LINE TO CONTROL THE FLOW
C     OF OXIDIZER FROM THE OXID TANK FOR AN INDIVIDUAL ENGIN.)
C
C    TIMCAV:          AN INPUT VARIABLE, CAVITATION HAPPENS ONLY AFTER TIMCAV
C    POINVP: OXID INPUT LINE VAPOR PRESSURE
C    P1NPSH: NET POSITIVE SUCTION HEAD (NPSH) VARIABLE OF P1
C    ICAVMD: FLAG TO SELECT THE METHOD OF CALCUALTING CAVITATION AT ZERO
C            OXID FLOW OF P1 WHEN PREVALVE IS CLOSING
C
C
      IF ( STIME .GT. TIMCAV) THEN
        P1NPSH = (POS - POINVP)/(12.0*RHOOS)
        FLOC1 = AMIN1(.8, DWO1A/rhoso1*v41p34)
        X1 = P1NPSH/(1. + so1sq)
        IF ( FLOC1 .LT. 0.3) THEN
          IF (ICAVMD .EQ. 1) THEN
            DUMY= AMAX1(0.,APV)
            dubas1 = AMIN1(1. , DUMY/BASEAR)
            dubas2 = 1. - dubas1
            ZERHFL= dubas1*FGEN(66, 44, X1) + dubas2 * FGEN(81, 56, X1) 2100
            ZERTFL= dubas1*FGEN(67, 45, X1) + dubas2 * FGEN(82, 57, X1)
          ELSE
            ZERHFL= FGEN(66,44,X1)
            ZERTFL= FGEN(67,45,X1)
```

```
            ENDIF
C
C    THE FOLLOWING CONDITION IS WHEN THE DWOP1 IS LESS THAN 0.0 (FLOW BACK)
C    AND THE CAVITATION IS COLLAPSING IN DIRECT PROPORTIONAL TO FLOW.
C
            IF (DWOP1 .LT. 0.0) THEN
              accel = - DWOP1 / (RHOP1 * WTOP1)
              HCAVP1 = prlint( accel, 0, 77 )                          SSM62200
              TCAVP1 = prlint( accel, 0, 78 )
C
C    INTERPOLATION OF CAVITATION FACTOR FOR FLOW COEF BETWEEN 0.0 AND 0.3 OF
C    TCAV1 IS THE TIME CONSTANT TO GENERATE (OR COLLAPSE) CAVITATION.
C
            ELSE
              fact2 = FLOC1 * 3.33333
              fact1 = 1. - fact2
              HCAVP1 = prlint( ( fact1*ZERHFL +
     +                 fact2*FGEN(72, 46, X1) - HCAVP1) / TCAV1 , 0, 77 )
              TCAVP1 = prlint( ( fact1*ZERTFL
     +                 fact2*FGEN(74, 47, X1) - TCAVP1 ) / TCAV1 , 0, 78 )
            ENDIF
C
C    INTERPOLATION OF CAVITATION FACTOR FOR FLOW COEF BETWEEN 0.3 AND 0.8 OF
C
          ELSE
            fact1 = 1.6 - 2. * FLOC1
            fact2 = 1. - fact1
            HCAVP1 = prlint( ( fact1*FGEN(72, 46, X1) +
     +           fact2*FGEN(73, 48, X1) - HCAVP1 ) / TCAV1, 0, 77 )  SSM62300
            TCAVP1 = prlint( ( fact1*FGEN(74, 47, X1) +
     +           fact2*FGEN(75, 49, X1) - TCAVP1 ) / TCAV1, 0, 78 )
          ENDIF
          FLOC2 = rlimit( .2, .8, DWOP2 / rhoso2 * v8p866)
C
          POI2VP = quadr( HOI2 )
          P2NPSH = (POI2 - POI2VP) / (12.0*RHOI2)
C
          X2 = P2NPSH / ( so2sq + 1.)
          IF( FLOC2 .LT. 0.5) THEN
C
C    INTERPOLATION OF CAVITATION FACTOR FOR FLOW COEF BETWEEN 0.0 AND 0.5 OF
C
            fact1 = 1.66667 - 3.33333 * FLOC2
            fact2 = 1. - fact1
            HCAVP2 = prlint( ( fact1 * FGEN(68, 50, X2) +
     +           fact2 * FGEN(76, 51, X2) - HCAVP2 ) / TCAV2, 0, 79 )
            TCAVP2 = prlint( ( fact1 * FGEN(69, 52, X2) +
     +           fact2 * FGEN(78, 53, X2) - TCAVP2 ) / TCAV2, 0, 80 )
C
C    INTERPOLATION OF CAVITATION FACTOR FOR FLOW COEF BETWEEN 0.5 AND 0.8 OF
```

```
C
      ELSE                                                          SSM62400
        fact1 = 2.66667 - 3.33333 * FLOC2
        fact2 = 1. - fact1
        HCAVP2 = prlint( ( fact1 * FGEN(76, 51, X2) +
     +          fact2 * FGEN(77, 54, X2) - HCAVP2 ) / TCAV2, 0, 79 )
        TCAVP2 = prlint( ( fact1 * FGEN(78, 53, X2) +
     +          fact2 * FGEN(79, 55, X2) - TCAVP2 ) / TCAV2, 0, 80 )
      ENDIF
*
*     POD2VP=620.15+19.1202*HOD2+.149371*HOD2**2
*     P3NPSH=(POD2-POD2VP)/(12.*RHOD2)                              SSM62500
*     X3=P3NPSH/(1+SO2**2)                    it matters
*
        POD2VP = quadr( HOD2 )
        P3NPSH = (POD2 - POD2VP)*.833333*vRHOD2                      SSM62500
        X3 = P3NPSH / (1. + so2sq)
*
        HCAVP3 = prlint( (FGEN(70, 58, X3) - HCAVP3)/TCAV2, 0, 81 )
        TCAVP3 = prlint( ( FGEN(71, 59, X3) - TCAVP3 )/TCAV2, 0, 82 )
*
*  Cavitation formatted output was eliminated. Output variables are
*  included in the regular output list
*
      ENDIF
C                                                                   SSM62600
C     OX INLET LINE DESCRIPTION
C
C    OXID TANK SUPPLIES THREE ENGINS.  DWOE2 AND DWOE3 ARE THE FLOWS FROM
C    TANK TO ENGIN #2 AND ENGIN #3 RESPECTIVELY.
C    THE NOTATIONS USED IN THIS SECTION ARE:
C    TDCOM:   TIME TO DETACH OXID TANK
C    WOCOM:   OXID WEIGHT IN THE COMMON DUCT (FOR ALL THREE ENGINS)
C    POT:     TANK PRESSURE AS FUNCTION OF TIME
C    ELCOM:   ELEVATION OF OXIDIZER IN COMMON DUCT (VERTICAL FEEDING)
C    RCOM:    RESISTANCE OF OXID COMMON DUCT
C    ZCOM:    INERTIA OF OXID COMMON DUCT
C    ELOIN:   ELEVATION OF OXID IN INLET DUCT (BETWEEN PREVALVE AND LPOP)
C    ROIN:    RESISTANCE OF OXID INLET DUCT (BETWEEN PREVALVE AND LPOP)
C    ZOIN:    INERTIA OF OXID INLET DUCT (BETWEEN PREVALVE AND LPOP)
C    RJTPVD: THE RESISTANCE BETWEEN JUNCTION AND DOWN SIDE OF PREVALVE
C
      IF (STIME .GE. TDCOM ) THEN
        WOCOM = prlint( -DWOTJ, 0, 83 )
      ENDIF
      POT = FGEN(37, 9, STIME)
      ELCOM = FGEN(58, 29 ,WOCOM)
      RCOM = FGEN(59, 30, WOCOM)
      ZCOM = FGEN(60, 31, WOCOM)
      ELOIN = FGEN(61, 32, VOLPV)                                   SSM62700
```

```
      ROIN = FGEN(62, 33, VOLPV)
      ZOIN = FGEN(63, 34, VOLPV)
      APV = recpos( FGEN (64, 35 ,(STIME - TCUT - TCLPV) / DTPV) )
      RJTPVD = RL1 + RPV / AMAX1(1.E-10, APV)**2
C
C   THIS SECTION CALCULATE THE BACK FLOW OF PUMPS.  THE EQUATIONS USED TO
C   DESCRIBE THE BACKFLOW AND MINIMUM WEIGHT OF PUMP INPUT LINES ARE NOT
C   IN THE DOCUMENT.
C   QBKFLO: ENERGY BACK FLOW RATE TO THE UPSTREAM OF THE PUMP, BTU/SEC
C   WT1BK, WT2BK: DEFINITION NOT CLEAR, DIMENSION = LB
C
C
      FLCOE1 = DWO1A / rhoso1
      EFF = recpos( FLCOE1 * (.03066 - .0003709*FLCOE1) )
      flcoe = ( 1. - FLCOE1*v41p34 )
      QBKFLO = recpos( TRQOP1*SO1*v9336*(1.-EFF)*flcoe )
      WT1BK = AMIN1(800., 33. + 2.74E+06 *  P1NPSH / so1sq,
     1               1310. - 1.14E+06 * P1NPSH / so1sq) * flcoe    SSM62800
      IF ( VOLPV .GE. 10. )
     +    WT1BK = AMIN1( WOIN,WT1BK)
      ABKFLO = QBKFLO*9336./( AMAX1(1., TRQOP1)*SO1)
      FLCOE2 = recpos( DWOP2/(RHOP2*SO2) )
      EFF2 = recpos( FLCOE2 * ( .1537 - .008667 * FLCOE2 ) )
      QBKFL2 = ABK2 *
     +     recpos( TRQOP2*SO2*v9336*(1.-EFF2)*(1.-FLCOE2*v8p866) )
      WT2BK = AMIN1(270., so2sq*(2.583E-4 - 2.91E-5*FLCOE2) )
C
C   THIS PART CALCULATES THE ENERGY BALANCE AND FLOW BETWEEN OXID TANK AND
C   LPOP.  THE LAYOUT OF THE HARDWARE OF THE SYSTEM IS:
C
C
C   TANK ------------> JUNCTION -------------> PREVALVE ------------> LPOP
C                  TO OTHER
C                   ENGIN
C  (TOP)  COMMON DUCT                  EL1                        OIN        (DOW
C
C  FLOW:   DWOTJ                                 DWOPV                 DWOP1
C
C  PRESSURE:
C   POT                       POJ                                POPVDN     POS
C
      HOTNK = FGEN(5, 8, STIME)
      POTVP = quadr( HOTNK )
      fact1 = AMIN1(1.0, 3.33333 * APV )
      POTVP = fact1 * POTVP + (1. - fact1) * POINVP
      DWOTJ = DWOPV + DWOE2 + DWOE3                                SSM62900
      IF (WOCOM .LE. 0.0) DWOTJ = 0.0
C
C   OXID FLOWS TO ENGIN #2 AND ENGIN #3 ARE TIME SCHEDULED
C
```

```
      DWOE2L=DWOE2
      DWOE2=FGEN(65, 36, STIME-TCUT2)
      DDWOE2=(DWOE2-DWOE2L)/DT
      DWOE3L=DWOE3
      DWOE3=FGEN(65, 60, STIME-TCUT3)
      DDWOE3=(DWOE3-DWOE3L)/DT
      DWOPVL=DWOPV
*
*  Originally DWOPV step was computed at some expense, then discarded
*  if WOCOM .LE. 0.
*
      rhdd4 = RHOOS * DDX * v386p4
      IF ( WOCOM .LE. 0.0) THEN
        DWOPV = 0.0
      ELSE
        CALL prflow( DWOPV, ZCOM+ZJTPV, -RJTPVD,
     +              POT + (ELCOM + ELJPV)*rhdd4 - POPVDN -
     +              RCOM*DWOTJ*ABS(DWOTJ) - (DDWOE2+DDWOE3) * ZCOM, 84 ) 3000
      ENDIF
      DDWOTJ = DDWOE2 + DDWOE3 + (DWOPV - DWOPVL) / DT
      POJ = POT + ELCOM*rhdd4 - RCOM*DWOTJ**2 - ZCOM*DDWOTJ
C
C   IN THE FOLLOWING CALCULATION, THE CONSTANT "BOPVDN" USED FOR CALCULATING
C   POPVDN IN THE CASE OF FLOW CHANGE IN THE INPUT LINE IS SUSPECIOUSLY
C   THE VALUE OF "BOPVDN" GIVEN IN THE FILE "START4.DAT" IS 5.0 (PSI/LB).
C   WHILE AN EQUIVALENT CASE IN OPOV LINE, "CFACT"=400000. (PSI/LB).
C   I BELIEVE THE VALUE SHOULD BE CHANGED ESPECIALLY IN THE TRANSIENT STUDY.
C
C   CONDITION FOR OUTPUT_FLOW > INPUT_FLOW WHERE EMPTY SPACE CAN BE GENERAGE
C   WITH THE PRESSURE EQUAL TO VAPOR PRESSURE OF THE OXID LINE.
C   VOLPV:  EMPTY SPACE VOLUMN FILLED BY OXID VAPOR IN PREVALVE SIDE
C
*  In the original, if DWOP1 .EQ. DWOPV, UOIN and WOIN get integrated twice.
*
      pDWOPV = recpos( DWOPV )
      IF ( DWOP1 .GT. 0.0 ) THEN
        IF ( DWOP1 .GE. DWOPV ) THEN
          CALL unint0( POPVDN, 85 )
          POPVDN = pruint( BOPVDN*(DWOPV-DWOP1), 0, 85 )

*         POPVDN = AMAX1(POTVP, POPVDN ) variable lower limit is implemented
*    by re-initializing the integrator.
*
          IF ( POPVDN .LT. POTVP ) THEN
            POPVDN = POTVP
            CALL unint0( POTVP, 85 )
          ENDIF
          IF ( POPVDN .LE. POTVP + .001) THEN
            VOLPV = pruint( ( DWOP1 - pDWOPV )/RHOOS, 0, 86 )
          ENDIF
```

```
*
*  No change in VOLPV if POPVDN .GT. POTVP + .001 ?
*
           UOIN = pruint( pDWOPV*FGEN(5, 8, STIME) -
     +                DWOP1*HOS + QBKFLO*WOIN/AMAX1(77.,WT1BK), 0, 87 )M63100
           WOIN = pruint( pDWOPV - DWOP1, 0, 88 )
C
C    CONDITION FOR INPUT_FLOW > OUTPUT_FLOW WHERE EMPTY SPACE (IF EXISTS)
C    ARE BEING FILLED.  OR THE PRESSURE MAY RAISE IF THERE IS NO EMPTY SPACE.
C
         ELSE
           VOLPV = prlint( (DWOP1 - DWOPV)/RHOOS, 0, 86 )
           IF( VOLPV .GE. .001) THEN
             POPVDN = POTVP
           ELSE
             POPVDN = pruint( BOPVDN*(DWOPV- DWOP1), 0, 85 )
             IF ( POPVDN .LT. POTVP ) THEN
               POPVDN = POTVP
               CALL unint0( POTVP, 85 )
             ENDIF
           ENDIF
           IF( DWOPV .GT.0.) THEN
             HOPV = FGEN(5, 8, STIME)
           ELSE
             HOPV = HOS                                              SSM63200
           ENDIF
           UOIN = pruint( DWOPV*HOPV - DWOP1*HOP +
     +              QBKFLO*WOIN/AMAX1(77., WT1BK), 0, 87 )
           WOIN = pruint( DWOPV - DWOP1, , 0, 88 )
         ENDIF
C
C    BACK FLOW FROM THE PUMP, FILLING UP THE EMPTY SPACE
C
       ELSE
         IF ( VOLPV .GT. 0.001 ) THEN
           VOLPV = prlint( (DWOP1 - pDWOPV)/RHOOS, 0, 86)
           POPVDN = POTVP
           CALL unint0( POTVP, 86 )
           UOIN = pruint( pDWOPV*FGEN(5, 8, STIME)
     +          - DWOP1*HOD1+QBKFLO*WOIN/AMAX1(77., WT1BK), 0, 87 )
           WOIN = pruint( pDWOPV-DWOP1, 0, 88 )
C
C    BACK FLOW FROM THE PUMP, WITHOUT EMPTY SPACE IN THE LINE
C
         ELSE                                                        SSM63300
           POPVDN = pruint( BOPVDN*(DWOPV - DWOP1), 0, 85 )
           IF ( POPVDN .LT. POTVP ) THEN
             POPVDN = POTVP
             CALL unint0( POTVP, 85 )
           ENDIF
```

```
      IF( DWOPV .GT. 0.0) THEN
        UOIN = pruint( DWOPV*FGEN(5,8,STIME) - DWOP1*HOD1 +
     +                 QBKFLO*WOIN/AMAX1(77., WT1BK), 0, 87 )
      ELSE
        UOIN = pruint( DWOPV*HOS - DWOP1*HOD1 +
     +                 QBKFLO*WOIN/AMAX1(77., WT1BK), 0, 87 )
      END
      WOIN = pruint( DWOPV-DWOP1, 0, 88 )
    ENDIF
   ENDIF
   POS = POPVDN + ELOIN*rhdd4 - ROIN*DWOP1*ABS(DWOP1) -
  +          ZOIN*(DWOP1 - DWOP1L)/DT                                  SSM63400
   IF (POS .LT. POINVP)  POS = POINVP
   POSX = pruint( TAU*(POS - POSX), 0, 89 )
   SUOIN = UOIN/WOIN
   HOS = SUOIN + POS/(RHOOS*9336.)
C
C  FOLLOWING IS AN ESTIMATION OF THE SPECIFIC ENTHALPY OF THE "OIN" LINE.
C  IT SEEMS THIS IS ONLY TRUE FOR A STEADY STATE FLOW.
C  BASED ON THE EQUATION BELOW, IT SUGGESTS THAT THE FILLED UP WEIGHT OF TH
C  "OIN" LINE IS 77LB.
C
   IF ( DWOP1 .GT. 0. ) THEN
     HOS1 = pruint( DWOP1*v77*(HOS-HOS1), 0, 90 )
   ELSE
     HOS1 = pruint( -DWOP1*v77*(HOD1 - HOS1), 0, 90 )
   ENDIF
   POINVP =  quadr( HOS1 )
   TRQOT1 =(AOT1 - BOT1*PHIOT1)*DWOT1*ABS(DWOT1)*CTQOT1
C
C  POGO SYSTEM BALANCE IS CALLED AS PART OF THE OXID FLOW.
C
   CALL POGOS
C                                                                        SSM63500
C     PRIME MAIN CHAMBER
C
C  PRIMING OF MOV AND OXID INJECTOR IS DEFINED AS SIMPLE FUNCTIONS OF
C  FILLED UP RATION OF THE PARTICULAR SPACE.
C  WOV:    WEIGHT TO BE FILLED, INITIAL VALUE = -2.0, FILLED = 0.0
C  WOI:    INJECTOR OXID WEIGHT, INIT VALUE = 0.0, FILLED = 46.2
C
   IF( XMOVPF .AND. WOV.GT.-.1) THEN
     COV = 1. + 10. * WOV
     IF(WOV.GT.-.001) THEN
       WRITE(event,'(A,E12.4)' ) ' MOV BUBBLE PRIMED AT ', STIME
       XMOVPF = .FALSE.                                                  SSM63600
     END IF
   ELSE IF ( WOV .LT. -.001 ) THEN
     COV=0.025
   ELSE
```

```
        COV=1.0
      END IF
C
      WOI = prlint( DWMOV*COV - DWOI, 0, 91 )
      WOV = prlint( DWMOV*(1.0 - COV), 0, 92 )
      rhomov = VOLOD2 / WOD2
      PRIMOI=FGEN(3, 37, WOI)
C
      IF( PRIMIF .AND.WOI .GT. 40. ) THEN
        WRITE( event,'(A,E12.4)' ) ' MC PRIMED AT ', STIME          SSM63700
        PRIMIF = .FALSE.
      END IF
C
      DWOI=DWMOV*PRIMOI*COV
      POINJ=PCIE+ROI*DWOI**2 * rhomov
      PMOV=POD2
*
*      IF(TIME.GE.TL1.AND.TIME.LE.TH1)WRITE(6,1410)TIME,WOCOM,ELCOM, etc
C
C    IN FUEL FLOW SUBROUTINE, QF IS TOTAL FUEL FLOW IN GALLON/MIN.
C    QO IN THE FLOWWING EQUATION DOESN'T SEEM TO REPRESENT SIMILAR QUANTITY.
C
*      QO=(1.0-0.002789*(TOS-160.))*(17.5339+6.19891*(DWOTPR+DWMOV+DWOP3
*     1   -DWOP2C)+0.208812*SO2)*AMIN1(1.0,(DWOTPR+DWMOV+DWOP3-DWOP2C)*
*     2   .1)
*
      dwsum = DWOTPR + DWMOV + DWOP3 - DWOP2C
      QO = ( 1.0 - 0.002789*(TOS - 160.) ) *
     +          ( 17.5339 + 6.19891*dwsum + 0.208812*SO2 ) *
     +               AMIN1(1.0, dwsum*.1)
C
C     OXIDIZER PUMP SPEEDS
C
C    IN THE FOLLOWING CALCULATIONS:
C    TQxyzB: BREAKAWAY TORQUE OF PUMP xyz
C
      IF (SO1 .LT. 11.0) THEN
        IF (TRQOT1-TRQOP1 .LT. TQOT1B ) THEN
          DSO1=0.0
        ELSE                                                        SSM63900
          DSO1=(TRQOT1-TRQOP1)/GO1
        ENDIF
      ELSE
          DSO1=(TRQOT1-TRQOP1)/GO1
      ENDIF
      SO1 = prlint( DSO1, 0, 93 )
      so1sq = SO1 ** 2
      trqsum = TRQOT2 - TRQOP2 - TRQOP3
      IF ( SO2 .LT. 11.0) THEN
        IF( trqsum .LT. TQOT2B) THEN 2021,2025,2025
```

```
    DSO2=0.0
  ELSE
    DSO2 = (trqsum - TDRAGO)/GO2
  ENDIF
ELSE
  DSO2 = (trqsum - TDRAGO)/GO2
ENDIF
SO2 = prlint( DSO2, 0, 94 )
so2sq = SO2 ** 2
RETURN
END
```

```
*
      SUBROUTINE oxprp0 (P, H, RHO, RHOL, RHOG, HG, HL, FL, SUIN, icall)
**********************************************************************
C
C  PURPOSE:  CALCULATE LIQUID AND SATURATED OXYGEN PROPERTIES
C
C******ARGUMENTS******
C INPUT:
C  RHO  = DENSITY, LB/IN3
C  SUIN = SPECIFIC INTERNAL ENERGY, BTU/LB                          SSM64100
C  NX   = CALLER NODE INDEX ARRAY
C
C OUTPUT:
C  P    = PRESSURE, PSI
C  H    = SPECIFIC ENTHALGY, BTU/LB
C
C IF SATURATED, ADDITIONAL OUTPUT IS:
C  FL   = LIQUID MASS FRACTION
C  RHOL = SATURATED LIQUID DENSITY, LB/IN3
C  RHOG = SATURATED GAS DENSITY, LB/IN3                             SSM64200
C  SLIQ = SATURATED LIQUID SPECIFIC INTERNAL ENERGY, BTU/LBM
C  SGAS = SATURATED GAS SPECIFIC INTERNAL ENERGY, BTU/LBM
C  HL   = SATURATED LIQUID SPECIFIC ENTHALPY, BTU/LBM
C  HG   = SATURATED GAS SPECIFIC ENTHALPY, BTU/LBM
C
*  The entry OXPROP bypasses initialization.
*
**********************************************************************
      PARAMETER (NSAT = 74, NSU = 60, NRHO = 60, NCALL = 10 )
*
      DIMENSION  A(60,60) , PSC(74) , SLC(74) , SGC(74) , RLC(74) ,
     *          RGC(74)   , SU(60)   , RH(60)   , RS(60)            SSM64300
*
*  Precomputed slopes:
*
      REAL SUvsRS(NSU), AvsSU( NSU, NRHO ), vdRHO(NRHO),
     +   RLvsP( NSAT ), RGvsP( NSAT ), SLvsP( NSAT ), SGvsP( NSAT )
*
*  Previous call intervals:
*
      INTEGER isu(NCALL), irho(NCALL), ipres(NCALL)
*
      DATA PSC /
     *   1.10,   1.33,   1.61,   1.92,   2.29,   2.71,   3.19,
     *   3.74,   4.36,   5.06,   5.85,   6.73,   7.71,   8.80,
     *  10.01,  11.34,  12.81,  14.42,  16.18,  18.11,  20.20,
     *  22.47,  24.94,  27.60,  30.47,  33.56,  36.88,  40.43,
     *  44.24,  48.31,  52.65,  57.28,  62.19,  67.41,  72.95,
     *  78.81,  85.01,  91.56,  98.47, 105.75, 113.42, 121.48,     SSM64500
     * 129.95, 138.84, 148.16, 157.93, 168.15, 178.83, 190.00,
```

```
     * 201.66, 213.83, 226.52, 239.74, 253.50, 267.82, 282.71,
     * 298.19, 314.26, 330.95, 348.27, 366.23, 384.86, 404.16,
     * 424.16, 444.87, 466.32, 488.53, 511.52, 535.32, 559.97,
     * 585.49, 611.92, 639.30, 731.38/
*
      DATA SLC /
     * -71.21, -70.41, -69.62, -68.82, -68.02, -67.22, -66.42,
     * -65.62, -64.82, -64.02, -63.22, -62.42, -61.61, -60.81,
     * -60.00, -59.20, -58.39, -57.58, -56.77, -55.96, -55.15,
     * -54.34, -53.52, -52.70, -51.89, -51.07, -50.24, -49.42,      SSM64600
     * -48.59, -47.77, -46.94, -46.10, -45.27, -44.43, -43.59,
     * -42.74, -41.90, -41.04, -40.19, -39.33, -38.47, -37.60,
     * -36.72, -35.85, -34.96, -34.07, -33.18, -32.27, -31.37,
     * -30.45, -29.52, -28.59, -27.64, -26.68, -25.71, -24.73,
     * -23.73, -22.72, -21.69, -20.65, -19.59, -18.51, -17.41,
     * -16.29, -15.13, -13.95, -12.73, -11.46, -10.14,  -8.76,
     *  -7.29,  -5.72,  -3.98,   4.95/
*
      DATA SGC /
     *  19.68,  19.97,  20.27,  20.57,  20.86,  21.15,  21.43,
     *  21.72,  22.00,  22.28,  22.56,  22.83,  23.10,  23.37,      SSM64700
     *  23.63,  23.89,  24.14,  24.39,  24.64,  24.88,  25.12,
     *  25.35,  25.58,  25.80,  26.02,  26.23,  26.43,  26.63,
     *  26.82,  27.01,  27.19,  27.36,  27.53,  27.69,  27.84,
     *  27.98,  28.12,  23.25,  28.37,  28.48,  28.59,  28.68,
     *  28.76,  28.84,  28.90,  28.96,  29.00,  29.03,  29.05,
     *  29.06,  29.05,  29.03,  29.00,  28.95,  28.88,  28.80,
     *  28.69,  28.57,  28.42,  28.26,  28.06,  27.84,  27.59,
     *  27.30,  26.97,  26.60,  26.17,  25.68,  25.11,  24.45,
     *  23.66,  22.69,  21.51,   4.95/
*
      DATA RLC /                                                    SSM64800
     * .04450, .04431, .04413, .04394, .04376, .04357, .04338,
     * .04319, .04300, .04281, .04262, .04243, .04224, .04204,
     * .04185, .04165, .04145, .04125, .04105, .04085, .04065,
     * .04045, .04024, .04003, .03983, .03962, .03940, .03919,
     * .03897, .03876, .03854, .03832, .03809, .03787, .03764,
     * .03741, .03717, .03694, .03670, .03645, .03621, .03596,
     * .03571, .03545, .03519, .03492, .03465, .03438, .03410,
     * .03381, .03352, .03322, .03292, .03261, .03230, .03197,
     * .03163, .03129, .03093, .03056, .03018, .02979, .02938,
     * .02895, .02850, .02803, .02753, .02700, .02644, .02582,      SSM64900
     * .02515, .02440, .02353, .01576/
*
      DATA RGC /
     * .00001, .00002, .00002, .00002, .00003, .00003, .00004,
     * .00005, .00005, .00006, .00007, .00008, .00009, .00010,
     * .00011, .00013, .00014, .00016, .00018, .00020, .00022,
     * .00024, .00026, .00029, .00032, .00035, .00038, .00041,
     * .00045, .00049, .00053, .00057, .00062, .00067, .00072,
```

```
     * .00077, .00083, .00089, .00096, .00102, .00110, .00117,
     * .00125, .00134, .00142, .00152, .00162, .00172, .00183,
     * .00195, .00207, .00220, .00233, .00248, .00263, .00279,      SSM65000
     * .00296, .00313, .00332, .00353, .00374, .00397, .00422,
     * .00448, .00476, .00507, .00540, .00577, .00617, .00663,
     * .00715, .00776, .00847, .01576/
*
      DATA SU  /
     * -64.00, -63.75, -63.50, -63.25, -63.00, -62.75, -62.50,
     * -62.25, -62.00, -61.75, -61.50, -61.25, -61.00, -60.75,
     * -60.50, -60.25, -60.00, -59.50, -59.00, -58.50, -58.00,
     * -57.50, -57.00, -56.50, -56.00, -55.50, -55.00, -54.50,
     * -54.00, -53.50, -53.00, -52.50, -52.00, -51.50, -51.00,
     * -50.50, -50.00, -49.00, -48.00, -47.00, -46.00, -45.00,      SSM65100
     * -44.00, -43.00, -42.00, -41.00, -40.00, -38.00, -36.00,
     * -34.00, -32.00, -30.00, -28.00, -26.00, -24.00, -22.00,
     * -20.00, -18.00, -16.00, -14.00/
*
      DATA RS  /
     * .04281, .04275, .04269, .04263, .04257, .04251, .04245,
     * .04239, .04233, .04227, .04221, .04215, .04209, .04203,
     * .04197, .04191, .04185, .04172, .04160, .04148, .04136,
     * .04123, .04111, .04099, .04086, .04074, .04061, .04049,
     * .04036, .04024, .04011, .03998, .03985, .03973, .03960,
     * .03947, .03934, .03908, .03882, .03855, .03829, .03802,      SSM65200
     * .03775, .03748, .03720, .03692, .03664, .03607, .03549,
     * .03490, .03429, .03367, .03304, .03239, .03172, .03104,
     * .03033, .02960, .02884, .02805/
*
      DATA RH  /
     * .70000, .75000, .80000, .85000, .90000, .95000,1.00000,
     *1.00002,1.00004,1.00006,1.00008,1.00010,1.00012,1.00016,
     *1.00020,1.00025,1.00030,1.00035,1.00040,1.00050,1.00060,
     *1.00080,1.00100,1.00120,1.00160,1.00200,1.00240,1.00300,
     *1.00360,1.00420,1.00480,1.00600,1.00840,1.00960,1.01080,
     *1.01200,1.01440,1.01680,1.01920,1.02280,1.02640,1.03000,      SSM65300
     *1.03360,1.03720,1.04080,1.04440,1.04800,1.05160,1.05520,
     *1.05880,1.06240,1.06600,1.06960,1.07320,1.07680,1.08040,
     *1.08400,1.08760,1.09180,1.09540/
*
      DATA (A(I, 1), I=1,60) /
     *      .37,      .39,      .41,      .43,      .45,      .47,
     *      .49,      .51,      .54,      .56,      .58,      .61,
     *      .64,      .66,      .69,      .72,      .75,      .82,
     *      .88,      .95,     1.03,     1.11,     1.19,     1.28,
     *     1.37,     1.47,     1.57,     1.68,     1.80,     1.92,
     *     2.04,     2.18,     2.32,     2.46,     2.62,     2.77,  SSM65400
     *     2.94,     3.30,     3.68,     4.10,     4.55,     5.04,
     *     5.56,     6.13,     6.73,     7.37,     8.05,     9.55,
     *    11.24,    13.11,    15.18,    17.45,    19.93,    22.61,
```

```
     *    25.48,    28.56,     7.66,    13.38,    14.53,    18.15/
*
      DATA (A(I, 2), I=1,60) /
     *      .40,      .42,      .44,      .46,      .48,      .50,
     *      .52,      .55,      .57,      .60,      .63,      .65,
     *      .68,      .71,      .74,      .77,      .81,      .87,
     *      .95,     1.02,     1.10,     1.19,     1.27,     1.37,

                most of table omitted

     * 10000.00, 10000.00, 10000.00, 10000.00, 10000.00, 10000.00,
     * 10000.00, 10000.00, 10000.00, 10000.00, 10000.00, 10000.00,
     * 10000.00, 10000.00, 10000.00, 10000.00, 10000.00,
10000.00,SSM71900
     * 10000.00, 10000.00, 10000.00, 10000.00, 10000.00, 10000.00,
     *  9815.73,  8760.81,  7130.06,  6397.53,  5773.35,  5155.16,
     *  4661.49,  4221.40,  3813.73,  3432.45,  3054.53,  2743.86/
*
*************************************************************************
*
*  Unnecessary initializations were removed.
*
*  Precomputing slopes:
*
      DO 10 I = 2, NSU
        SUvsRS(I) = ( SU(I) - SU(I-1) ) / ( RS(I) - RS(I-1) )
   10 CONTINUE
      DO 20 I = 2, NSAT
         VDPRES = 1.0 / ( PRES(I) - PRES(I-1) )
         RLvsP(I) = ( RLC(I) - RLC(I-1) ) * VDPRES
         RGvsP(I) = ( RGC(I) - RGC(I-1) ) * VDPRES
         SLvsP(I) = ( SLC(I) - SLC(I-1) ) * VDPRES
         SGvsP(I) = ( SGC(I) - SGC(I-1) ) * VDPRES
   20 CONTINUE
      CALL xyset( NSU, SU, NRHO, RH, A, AvsSU, vdRHO )

      ENTRY OXPROP (P, H, RHO, RHOL, RHOG, HG, HL, FL, SUIN, icall)
*************************************************************************
*
*  This entry bypasses initialization
*
*  Interpolation with TABLX and TABLXY was replaced by interpolation
*  with the reorganized interpolation module routines. Extrapolation
*  to the table boundary value was therefore changed to an error stop
*  if an input value escapes its table. Also interpolation is based on
*  precomputed slopes.
*************************************************************************
      CALL intval( isu(icall), SUIN, NSU, SU,
     +     'Oxygen energy is below the table.',
     +     'Oxygen energy exceeds the table.', 0 )
```

```
      iu = isu(icall)
      R = RHO / xlint( SUIN, NSU, SU, RS, SUvsRS, iu)
*
      CALL intval( irho(icall), R, NRHO, RH,
     +     'Oxygen density is below the table.',
     +     'Oxygen density exceeds the table.', 0 )
      ir = irho(icall)
      P = xylint( SUIN, R, NSU, SU, NRHO, RH, AvsSU, vdRHO, A, iu, ir )
*
      H = SUIN + P / (9336.0 * RHO)                                    SSM72200
      IF (R .GE. 1.0)  THEN
         FL = 1.0
      ELSE
         CALL intval( ipres(icall), P, NSAT, PSC,
     +        'Oxygen pressure is below the table.',
     +        'Oxygen pressure exceeds the table.', 0 )
         ip = ipres(icall)
         RHOL = xlint( P, NSAT, PSC, RLC, RLvsP, ip )
         RHOG = xlint( P, NSAT, PSC, RGC, RGvsP, ip )
         SLIQ = xlint( P, NSAT, PSC, SLC, SLvsP, ip )
         SGAS = xlint( P, NSAT, PSC, SGC, SGvsP, ip )
         FL = (SUIN - SGAS) / (SLIQ - SGAS)
         HL = SLIQ + P / (9336.0 * RHOL)
         HG = SGAS + P / (9336.0 * RHOG)                               SSM72300
      END IF
      RETURN
      END
```

'o2prime.for':

```
      SUBROUTINE O2PRM0
*****************************************************************
C
C  SUBROUTINE O2PROP(SU,SRHO,N,P,T,NN)                               SSM52500
C
C  PURPOSE:  OXIGEN PROPERTY DATA
C
C   THIS SUBROUTINE IS THE OXIDIZER PROPERTY TABLE LOOK-UP. (NOT HYDROGEN)
C   THERE ARE ACTUALLY TWO TABLES.
C   FIRST ONE IS THE TWO DIMENTIONAL CURVES: PRESSURE(U, RHO)
C   SECOND ONE IS THE TWO DIMENTIONAL CURVES: TEMPERATURE(U, RHO)
C   SPECIFIC ENTHALPY U RANGE FROM -57.5 TO 110.0
C   DENSITY RHO RANGES FROM 0.08 TO 80.0
C   THE PROGRAM ONLY DO INTERPOLATION AND WILL BE KICKED OUT IF THE RANGE IS
C   OUT OF REACH.
C
C******ARGUMENTS******
C  NN = 0  INITIALIZATION
C
C  CONDITION       INPUT      OUTPUT
C  NN = 1 OR 2     SU,SRHO    P,T
```

```
C  NN = 3          SU,P        SRHO,T
C  NN = 4          SRHO,P      SU,T                                        SSM52600
C
C  SU   = SPECIFIC ENTHALPY, BTU/LB
C  SRHO = DENSITY, LB/IN3
C  P    = PRESSURE, PSI
C  T    = TEMPERATURE, DEG R
C  N    = CALLER NODE INDEX
C
      PARAMETER (NCALL=20, NU=15, NRHO=21, NSAT=NU)                        SSM52700
      PARAMETER (NP=NSAT*NRHO, NT=NRHO-4)
C
      DIMENSION U(NU),RHO(NRHO), PRES(NU,NRHO), TEMP(NU,NRHO)
      REAL UvsP(NU,NRHO), TXP(NU,NRHO), vDRHO(NRHO), H(NU,NRHO),
     + apo2(NRHO, NU), bpo2(NRHO, NU), cpo2(NRHO, NU), dpo2(NRHO, NU),
     + ato2(NRHO, NU), bto2(NRHO, NU), cto2(NRHO, NU), dto2(NRHO, NU),
     + JSAT(NSAT)
      DIMENSION I1(NCALL), J1(NCALL), J2(NCALL), R(2)
*
      SAVE
C
      DATA I1 /NCALL*2/, J1 /NCALL*2/, J2 /NCALL*2/
C
*      DATA G / NP*0.0 /                                                   SSM52800
*      DATA H / NP*0.0 /
*      DATA C / NT*1.0 /
*
      DATA JSAT / 19, 18, 17, 16, 15, 14, 12,
     $            11, 11, 10,  9,  8,  8,  7, 7 /
C
      DATA                          U   /
     *  -57.5,  -55.1,  -51.0,  -46.8,  -39.9,  -29.8,  -20.0,  -10.0,
     *     .0,   10.0,   30.0,   50.0,   70.0,   90.0,  110.0/
      DATA                          RHO /
     *  .0800,  .2000,  .4000,  .8000,  1.200,   2.00,   4.00,
8.0,52900
     * 16.000, 32.000;  40.00,  50.00,  55.00, 58.430, 63.400, 66.590,
     * 68.450, 70.20,   71.29,  75.00,  80.00 /
      DATA    (PRES(I,01), I=1,15)  /

                  most of tables omitted

     * 161.90, 165.40, 171.40, 177.50, 187.80, 203.40, 218.80, 234.60,
     * 200.00, 200.00, 200.00, 200.00, 200.00, 200.00, 200.00/
      DATA    (TEMP(I,20), I=1,15)  /
     * 152.00, 154.00, 162.00, 164.00, 174.00, 192.00, 210.00, 220.00,
     * 200.00, 200.00, 200.00, 200.00, 200.00, 200.00, 200.00/
      DATA    (TEMP(I,21), I=1,15)  /
     * 131.00, 137.00, 143.00, 150.00, 165.00, 180.00, 200.00, 200.00,
     * 200.00, 200.00, 200.00, 200.00, 200.00, 200.00, 200.00/
```

```
C
*********************************************************************
C
C         INITIALIZE SLOPES
C
      DO 26 J = 1, NRHO
        RHO(J)=RHO(J)/1728.
        DO 25 I = 2, NU
          UvsP(I,J) = ( U(I) - U(I-1) ) / ( PRES(I,J) - PRES(I-1,J) )
          TXP(I,J) = (TEMP(I,J) - TEMP(I-1,J)) / (U(I) - U(I-1))
   25   CONTINUE                                                    SSM54300
        IF(J.NE.NRHO) THEN
           DRHO = RHO(J+1)/1728.- RHO(J)
           vDRHO(I) = 1. / DRHO
        ENDIF
   26 CONTINUE
*
*  This overcomplicated precomputation replaced by a more efficient,
*  more straightforward, and generally reusable cubic spline system.
C
C               GET U CURVATURE, H AND
C           RHO CURVATURE, G, FOR SPLINE INTERPOLATION
C
      DO 30 J = 1, 21
        CALL splin0( 15, RHO, PRES(1,J),
     +               apo2(1,J), bpo2(1,J), cpo2(1,J), dpo2(1,J) )
        CALL splin0( 15, RHO, TEMP(1,J),
     +               ato2(1,J), bto2(1,J), cto2(1,J), dto2(1,J) )
   30 CONTINUE
*
*  Removed echo of O2 tables
*
      RETURN
*
      ENTRY o2pt( SU, SRHO, N, P, T )
*********************************************************************
C
C      NN=1,    PRESSURE COMPUTATIONS                               SSM55500
C
*********************************************************************
      I = I1(N)
      CALL intval( I, SU, 21, U,
     +   " Specific enthalpy SU is below the O2 enthalpy table",
     +   " Specific enthalpy SU is above the O2 enthalpy table", 0 )
  140 J = J1(N)
      CALL intval( J, SRHO, 21, RHO,
     +   " Density RHO is below the O2 density table",
     +   " Density RHO is above the O2 density table", 0 )
*
      P1 = spline( I-1, SRHO
```

```
     +      apo2(1,I-1), bpo2(1,I-1), cpo2(1,I-1), dpo2(1,I-1) )
       P2 = spline( I, SRHO
     +             apo2(1,I), bpo2(1,I), cpo2(1,I), dpo2(1,I) )
*
       RHOP1=(SRHO - RHO(J)) * vDRHO(J)
C
       U1 = U(I-1)
       U2 = U(I)
       JS =JSAT(I)
       JS1=JSAT(I-1)
       IF((J.EQ.JS.AND.JS.NE.JS1) .OR. (I.EQ.7.AND.J.EQ.JS+1)) THEN
         USAT=U2+(U1-U2)*(SRHO-RHO(JS))/(RHO(JS1)-RHO(JS))
         PSAT=PRES(I-1,JS1)+(PRES(I,JS)-PRES(I-1,JS1))/(U2-U1)
     $             *(SU-U1)
         IF( SU.GT.USAT ) THEN                                          SSM55700
           P1=PSAT
           U1=USAT
*          H1=0.0
         ELSE
           P2=PSAT
           U2=USAT
*          H2=0.0
         END IF
       END IF                                                           SSM55800
C
*      P=SPLINE(P1,P2,H1,H2,SU-U1,U2-SU)
C
*  The above spline fit of pressure parallel to the enthalpy axis is
*  misapplied. Only two points have been found on the surface at the
*  interpolated cross section. To fit a spline, it would be necessary
*  to find more than four cross section points. It might be desirable to
*  find four points and interpolate with a cubic through them. Here, we
*  substitute a linear interpolation in SU.
*
       UP1 = SU - U(I-1)
       P = P1 + UP1*vdu(I)*( P2 - P1 )
*
       T1 = TEMP(I-1,J) + TXP(I,J) * UP1
       T2 = TEMP(I-1,J+1) + TXP(I,J+1) *UP1
       T = T1 + (T2 - T1) * RHOP1
   900 RETURN
*
       ENTRY rto2( SU, SRHO, N, P, T )
*****************************************************************
C
C    NN=3, RHO AND T FROM U AND P                                    SSM55900
C
*****************************************************************
   300 CONTINUE
       K=I-1
```

```
      J=J2(N)
  310 IF(PRES(K,J).LT.P) GO TO 330
  320 IF(PRES(K,J-1).LE.P) GO TO 340
      J=J-1
      GO TO 320
  330 J=J+1
      GO TO 310
*
*     Should be recoded to halt with message, rather than limit J
*     If rows and columns of PRES were reversed, intval could be
*      used.
*
  340 J=MAX0(2,MIN0(J,NRHO))                                        SSM56000
      IF(K.EQ.I-1) J2(N)=J
*
*     R(K-I+2)=RHO(J-1)+(P-PRES(K,J-1))/(PRES(K,J)-PRES(K,J-1))
*    $  *DRHO(J-1)         should use precomputed slope
*
      R(K-I+2)=RHO(J-1)+(P-PRES(K,J-1)) * RvsP(K,J)
      IF(K.EQ.I) GO TO 350
      K=I
      GO TO 310
C   ASSUME RHO LINIER WITH 1/U
  350 SRHO=R(1)+(U(I)-U(I)*U(I-1)/SU)*(R(2)-R(1))/(U(I)-U(I-1))
      GO TO 140
*
      ENTRY uto2( SU, SRHO, N, P, T )
*********************************************************************
C                                                                     SSM56100
C     GET U AND T FROM RHO AND P
C
*********************************************************************
 1000 DO 1010 K=1,NRHO
      IF(RHO(K).GT.SRHO) GO TO 1030
 1010 CONTINUE
      K = NRHO
 1030 IT=1
      K=MAX0(2,K)
 1035 DO 1040 M=1,NU
      IF(PRES(M,K-1).GT.P) GO TO 1050                               SSM56200
 1040 CONTINUE
      M = NU
 1050 IF(IT.NE.1) GO TO 1060
*             Should be recoded to halt with message, not limit M
      M=MAX0(2,M)
*
*     U1=U(M-1)+(P-PRES(M-1,K-1))/(PRES(M,K-1)-PRES(M-1,K-1))*(U(M)-
*    1   U(M-1))
*
      U1=U(M-1)+(P-PRES(M-1,K-1)) * UvsP(M,K)
```

```
      IT=2
      K=K+1
      GO TO 1035
 1060 M=MAX0(2,M)                                                    SSM56300
*
*     U2=U(M-1)+(P-PRES(M-1,K-1))/(PRES(M,K-1)-PRES(M-1,K-1))*(U(M)-
*    1   U(M-1))
*
      U2 = U(M-1) + (P-PRES(M-1,K-1)) * UvsP(M,K)
*     SU=U1+(RHO(K-1)-RHO(K-1)*RHO(K-2)/SRHO)/(RHO(K-1)-RHO(K-2))*
*    $  (U2-U1)
C     SU=U1+(SRHO-RHO(K-2))/(RHO(K-1)-RHO(K-2))*(U2-U1)
*
      SU = U1 + (SRHO-RHO(K-2)) * vdrho(K-1)*(U2-U1)
*
      UP1 = SU - U(I-1)
      T1 = TEMP(I-1,J) + TXP(I,J) * UP1
      T2 = TEMP(I-1,J+1) + TXP(I,J+1) *UP1
      T = T1 + (T2 - T1) * RHOP1
      RETURN
      END
```

```
      SUBROUTINE HOTGAS0
************************************************************************
C
C  PURPOSE:  COMPUTE ENGINE HOT GAS PRESSURE AND FLOW DYNAMICS
C
C******ARGUMENT******
C  IHCNTL = INITIALIZATION ARGUMENT  was eliminated by use of two entries
C
C******COMMON USAGE******                                          SSM25000
C  INPUT:
C   VARIABLE                                                  SOURCE
C     POD3,DWOI,PMOV,POINJ,RHOMOV,RHOOP3,SO2                  OXIDF
C     DWFPF,DWOPF,DWFT1,DWFBPV,DWFT2C,SF2,TW1                 FUELF
C     RFPOV,ROPOV                                             VALDYM
C
C  OUTPUT:
C   VARIABLE                                                  DESTINATION
C     DWOPO,DWFPO,TRQOT2,PCIE                                 OXIDF
C     DWOPO,DWFPO,PFPOV,POPOV,PFPOI,POPOI                     EMCO      SSM25100
C     PCIE                                                    CNTROL
C     PFP,POP,PFI,PINMC,TRQFT2,QIN1                           FUELF
C
C  SUBROUTINES CALLED:  IGN
C
************************************************************************
*
      LOGICAL FFPTV, FFPVI, FFPTI, FFPOI, FFPOT, FFPTA
      LOGICAL OPOIPF, OPOVPF, wtason, wtigon
      LOGICAL FOPTV, FOPVI, FOPTI, FOPOI, FOPTA, FOPOT, FPIG
      INTEGER Tstep
C
      PARAMETER ( NPREO = 11, NFSO = 6 , NFSO1 = NFSO - 1 )
      PARAMETER ( NPREF=11, NFSF=7,
     +            NPREF1 = NPREF - 1, NFSF1 = NFSF - 1 )
      PARAMETER ( TooBig = 1.E20, Tstep = 0 )
C                                                                 SSM25200
      REAL sawop(4), sawfp(4),
     +     sdwfp( NPREF1, NFSF1 ), vdfs( NPREF1 ),
     +     sdwofp( NPREF1, NFSO1 ), vdfso2( NFSO1 )
      DIMENSION DWOIG(3),DWFIG(3)
      DIMENSION DW2(3),DW1(3),DW3(3)
      DIMENSION WFPTAB(4), AFPTAB(4), WOPTAB(4), AOPTAB(4)
      DIMENSION FPRO(NPREO), FSOTAB(NFSO), DWOTAB(NPREO,NFSO)     SSM25300
      DIMENSION FPRF(NPREF), FSFTAB(NFSF), DWFTAB(NPREF,NFSF)
      DIMENSION JSF(2), JSO(2), ZFIG(3), RFIG(3)
C
      INCLUDE 'blank.com'
      INCLUDE 'hgas.com'
      INCLUDE 'oxid.com'
      INCLUDE 'igni.com'
```

```
      INCLUDE 'balc.com'
      INCLUDE 'fuel.com'
      INCLUDE 'contrl.com'
      INCLUDE 'out.com'
      INCLUDE 'units.com'
C
      DATA TINIT / 460.0    /
      DATA AFPR1 / 0.0142   /
      DATA AFPR2 / 0.04323  /
      DATA AFPTAB/ 0.100, 0.0013, 0.0013, 0.0     /                  SSM26100
      DATA AOPTAB/ 0.1513, 0.00166, 0.00166, 0.0  /
      DATA AOPRG / 0.0039   /
      DATA ASFPC / 0.53899  /
      DATA ASOPC / 0.53899  /
      DATA PHES  / 750.0    /
      DATA PRINLO/ 100.0    /
      DATA RHES  / 50000.   /
      DATA TAUH  / 0.01     /
      DATA TFPDH / 250.0    /
      DATA TOPDH / 250.0    /                                        SSM26200
      DATA TPRG  / 520.0    /
      DATA TPRC  / 250.0    /
      DATA WFPTAB/ 0.0, 2.2, 3.2, 3.3  /
      DATA WOPTAB/ 0.0, 0.8, 2.0, 2.1  /
      DATA WTASI / 0.06643  /
      DATA WTIGN / 0.0154   /
      DATA RHGFM / .0340E-4 /
      DATA RHGOM / .1060E-4 /
      DATA IGOFU / 1        /
      DATA IGOOX / 1        /                                        SSM26300
      DATA FSFTAB / 0.0, 946.9, 2470., 4117., 5352., 6999., 8192. /
      DATA FPRF   /1.0,1.204,1.307,1.411,1.515,1.619,1.7,1.8,2.0,2.4,3./
      DATA FSOTAB / 0.0, 3000., 4000., 5000., 6000., 7000. /
      DATA FPRO   /1.0, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 2.0, 2.4, 3./
      DATA DWFTAB/
     * 0.0,2.740,3.015,3.140,3.204,3.225,3.235,3.240,3.240,3.240,3.240,
     1 0.0,2.528,2.844,3.016,3.104,3.143,3.158,3.175,3.195,3.200,3.200,
     2 0.0,2.258,2.607,2.822,2.939,3.009,3.040,3.075,3.120,3.160,3.160,
     3 0.0,2.068,2.409,2.636,2.776,2.867,2.920,2.970,3.034,3.095,3.100,
     4 0.0,1.988,2.311,2.529,2.670,2.771,2.823,2.876,2.960,3.041,3.060, SSM2
     5 0.0,1.937,2.221,2.421,2.564,2.658,2.715,2.780,2.870,2.965,3.000,
     6 0.0,1.930,2.193,2.382,2.507,2.599,2.655,2.710,2.795,2.900,2.960/
*
      DATA DWOTAB/
     * 0.0,.8780,.9560,1.020,1.046,1.055,1.060,1.060,1.060,1.060,1.060,
     1 0.0,.7433,.8229,.8924,.9418,.9737,.9905,.9989,1.018,1.025,1.025,
     2 0.0,.7156,.7888,.8538,.9086,.9459,.9681,.9818,1.002,1.013,1.013,
     3 0.0,.6989,.7656,.8263,.8785,.9191,.9471,.9662,.9865,1.000,1.000,
     4 0.0,.6906,.7522,.8090,.8574,.8973,.9280,.9498,.9740,.9900,.9900,
     5 0.0,.6843,.7466,.7968,.8414,.8805,.9124,.9344,.9600,.9780,.9800/
```

```
*
      DATA CDWFT2 / 1.0 /                                                SSM26500
      DATA CDWOT2 / 1.0 /
*
      recpos(x) = AMAX1( 0.0, x )
      recneg(x) = AMIN1( 0.0, x )
      rlimit( floor, ceiling, x ) = AMAX1( floor, AMIN1( ceiling, x ) )
*
      SAVE
*
*************************************************************************
*
*     Precompute slopes for DATA tables
*
      CALL sxset( sawop, 4, WOPTAB, AOPTAB )
      CALL sxset( sawfp, 4, WFPTAB, AFPTAB )
      CALL xyset( NPREF, FPREF, NFSF, FSF2, DWFTAB, sdwfp, vdfs )
      CALL xyset( NPREO, FPRO, NFSO, FSO2, DWOTAB, sdwofp, vdfso2 )
*
C INITIALIZE LOCAL VARIABLES AND ARRAYS WHICH AREN'T ASSIGNED VALUES
C THIS IS A NECESSARY REQUIREMENT FOR SUCCESSFUL EXECUTION ON THE IBM
C
      A=0.0
      AFPOI=0.0
      AFPTA=0.0
      AFPTI=0.0
      AFPTV=0.0
      AFPVI=0.0
      AOPT2=0.0
      AREAF=0.0
      AR4=0.0
      CFACT=0.0
      DIAT=0.0
      DPOPAS=0.0
      DWACV=0.0
      DWBAF=0.0
      DWFPBI=0.0
      DWFPR1=0.0
      DWPFI=0.0
      DWPFS=0.0
      EW=0.0
      FDWFT2=0.0
      FDWOT2=0.0
      FOPTV= .TRUE.
      FSF2=0.0                                                           SSM27400
      FSO2=0.0
      GAM6=0.0
      IF1=0.0                                                            SSM27500
      IF2=0.0
      ISV=0.0
```

```
      OPOVPF = .TRUE.
      PFPBFB=0.0
      RC=0.0
                    initializations omitted

      WOPTA=0.0
      WOPTI=0.0
      WOPTV=0.0                                                        SSM28400
      WOPVI=0.0
      wtason = .TRUE.
      wtigon = .TRUE.
      ZFPOI=0.0
 9999 CONTINUE
C
C                  DATA INPUT
      READ(run,30)ZFPO,VOLFP,ZOPO,VOLOP,AOPTO,VOLOT1,VOLFI,VOLC,AR4,AR5
     1,AR6,EMC5,EMC6,EW,DIAT,RC,WOPOV,WFPOV,RFPIGB,ROPIGB
   30 FORMAT(//2X,6g12.4)
      READ(run,30)AHTMCF,TKMCF,AHTMCO,TKMCO,VOLMCF,VOLMCO,VOLFTD,VOLOTD
     *,RFPVUG,ROPVUG                                                   SSM28600
      READ(run,30)CDWFT2,CDWOT2,DPRNT,PRINLO,PRINHI,PHES
*
*           Added here to eliminate a simulation loop division:
      vVOLC = 1.0 / VOLC
      ROPO=0.0
      CALL IGN0(1,PFPOV,PFPOI,RHO03,P(9),RHO(9),PFP)
      CALL IGN0(2,POPOV,POPOI,RHO03,P(9),RHO(9),POP)
      CALL IGN0(3,PMOV,POINJ,RHOMOV,P(9),RHO(9),PCIE)
      WOPOI = 0.0                                                      SSM28900
      EMC7=1.0
      AR7=1.0
      WFPOI=0.0
      ELFFP=0.0
      CALL fgset(15)
      CPH2 = fgen(15, 61, TINIT)
      CALL fgset(1)
      COOI = FGEN(1, 62, WOPOI)
      CALL fgset(2)
      CFOI = FGEN(2, 63, WFPOI)
      CALL fgset(4)
      SC4 = FGEN(4, 64, AR4)
      CALL fgset(8)
      BC4 = FGEN(8, 65, AR4)
      CALL fgset(20)
      TFPC = FGEN(20, 66, ELFFP)                                       SSM29000
      CALL fgset(21)
      CPFP = FGEN(21, 67, ELFFP)
      CALL fgset(13)
      GAMFP = FGEN(13, 68, ELFFP)
      CALL fgset(12)
```

```
      EMWFP = FGEN(12, 69, ELFFP/(1.-ELFFP))
      CALL fgset(25)
      EMUC = FGEN(25, 70, ELFFP)
      CALL fgset(23)
      EKC = FGEN(23, 71, ELFFP)
      CALL fgset(28)
      ETAOT2 = FGEN(28, 72, 0.0)
      CALL fgset(29)
      ETAFT2 = FGEN(29, 73, 0.0)
      CALL fgset(41)
      CFG = FGEN(41, 74, 0.)
      CS=CSTAR(0,0.,0.)
C                                                                     SSM29100
C                INITIALIZATION
C
      DWFPO=0.0
      DWFPOI=0.0
      RHOO3=RHOOP3
      PFP=PA
      WTFP=PFP*VOLFP/9270.0
      ELFFPM=0.0
      DWFT2=0.0
      WFPO=0.0                                                        SSM29200
      WFP=WTFP/TINIT
      WFPF=WFP
      WOPOI=0.0
      DWOPO=0.0
      DWOPOI=0.0
      POP=PA
      WTOP=POP*VOLOP/9270.0
      WOP = WTOP/TINIT
      WOPF=WOP
      WOPO=0.0                                                        SSM29300
      ELFOP=0.0
      ELFOPM=0.0
      DWOT2=0.0
      TRQOT2=0.0
      DWOT1=0.0
      POT1=PA
      PROT1 = 1.0
      PRFT2 = .999
      PROT2 = 0.999
      WTOT1=POT1*VOLOT1/9270.0                                        SSM29400
      WOT1=WTOT1/TINIT
      TOT1=TINIT
      TOP=TINIT
      TFP=TINIT
      TRQOT1=0.0
      TOT1D=TINIT
      PFI=PA
```

```
PFT2D=PFI
POT2D=PFI
WTFI = PFI*VOLFI/9270.0                                          SSN29500
TFI=TINIT
WFI=WTFI/TFI
WFIF=WFI
WFIO=0.0
GAMFI=1.4
RGCFI=9270.0
DWC=0.0
SC4 = FGEN(4, 64, AR4)
BC4 = FGEN(8, 65, AR4)
AA2 = EW * .2                                                    SSM29600
A3=0.8-AA2
A5 = X10th( DIAT/RC, 1 )
PCIE=PA
PCNS=PA
TC=TINIT
WTC=PCIE*VOLC/9270.0
WCO=0.0
WC=WTC/TC
WCF=WC
ELFCM=0.0                                                        SSM29700
ELFC=0.0
GAMC=FGEN(13, 75, ELFCM)
GAM6=FGEN(13, 76, 0.0)
GAM4=GAM6
RGCC=18540.0/FGEN(12, 77, ELFC/(1.-ELFC))
RGC6=18540.0/FGEN(12, 78, 0.0)
WOPGN2=0.
WFPGN2=0.
RGN2=0.
RGC4=RGC6                                                        SSM29800
WCN=0.
ENFC=0.
DWOI=0.0
DWFI=0.0
PFPOV=POD3
PFPOI=PA
POPOV=POD3
POPOI=PA
CFOV=1.0
RHOFTF=RHO(6)                                                    SSM29900
RHOOTF=RHO(6)
PINMC=PA
PFIS=PA
RHOFI=RHO(6)
PPURG=50.0
DWSFS=0.
DWFTF=0.
```

```
DWX=0.
FFPTV = .TRUE.
FFPVI = .TRUE.                                                  SSM30000
FFPTI = .TRUE.
FFPOI = .TRUE.                                                  SSM30020
FFPTA = .TRUE.
FFPOT = .TRUE.
DWFPTV = 0.0
DWFPVI = 0.0
DWFPSG = 0.0
WFPTV = 0.0576
WFPVI = 0.5356
WFPTI = 0.00376                                                 SSM30100
WFPOT = 0.0128
WFPTA = 0.0136
ZFPOI = 2.000 E-03
RFPOTV =  68.77
RFPOVT =  29.02
RFPOVI = -0.885
RFPOTI =-141.97
RFPOTA =-470.16
POPRG  = PA
PFPRG  = PA                                                     SSM30200
AFPTV = 4.34 E-03
AFPVT = 6.677E-03
AFPVI = 3.82 E-02
AFPTI = 3.019E-03
AFPOI = 2.62 E-01
AFPTA = 1.659E-03
AFPRS = 1.761 E-02
RHOFTV = RHOOP3
RHOFVI = RHOOP3
RHOFTI = RHOOP3                                                 SSM30300
RHOFOI = RHOOP3
RHOFTA = RHOOP3
CFACT  = 400000.
FOPTV = .TRUE.
FOPVI = .TRUE.
FOPTI  = .TRUE.
FOPOI  = .TRUE.
FOPTA  = .TRUE.
FOPOT  = .TRUE.
DWOPTV  = 0.0                                                   SSM30400
DWOPVI  = 0.0
DWOPTI  = 0.0
DWOPTA  = 0.0
WOPTV  = 0.0587
WOPVI  = 0.5150
WOPTI  = 0.00376
WOPOT  = 0.01372
```

```
      WOPTA  = 0.01393
      ZOPOI  = 2.00  E-03
      ROPOTV = 110.895                                              SSM30500
      ROPOVT =  46.853
      ROPOVI =  -1.393
      ROPOTI = -100.20
      ROPOTA =-781.830
      POPOT = 700.0
      POPOV = 700.0
      AOPTV  = 3.420 E-03
      AOPVT =  5.260E-03
      AOPVI  = 3.048 E-02
      AOPTI  = 3.590 E-03                                           SSM30600
      AOPTA  = 1.290 E-03
      AOPR1  = 2.553 E-03
      AOPR2  = 2.502 E-03
      RHOOTV = RHOOP3
      RHOOVI = RHOOP3
      RHOOTI = RHOOP3
      RHOOOI = RHOOP3
      RHOOTA = RHOOP3
      DWXF=0.
*
*  Initiialization moved from simulation loop.
*
      RFLEAK = 1.0 / (772.8 * (FGEN(18, 79, 20.) * ABFPO / 100.) ** 2)
      WFPOIZ = FGEN (2, 80, 0.999) + 0.01
      RHOFGN    = RHOOP3
      vTP46 = 1. / ( 4632. * TPRC )
      CALL unint0( DWFPO, 95 )
      CALL unint0( DWFPOI, 96 )
      CALL unint0( WFPOI, 97 )
      CALL unint0( WFPTV, 98 )
      CALL unint0( DWFPTV, 99 )
      CALL unint0( WFPVI, 100 )
      CALL unint0( DWFPVI, 101 )
      CALL unint0( WFPTI, 102 )
      CALL unint0( DWFPTI, 103 )
      CALL unint0( WFPOT, 104 )
      CALL uninto( WFPOI, 105 )
      CALL unint0( DWFPOI, 106 )
      CALL unint0( WFPTA, 107 )
      CALL unint0( DWOIG(1), 108 )
      CALL unint0( PFPOV, 109 )
      CALL unint0( PFPOT, 110 )
      CALL unint0( PFPOI, 111 )
      CALL lmint0( WFPF, 112, 0.0, TooBig )
      CALL lmint0( WFPO, 113, 0.0, TooBig )
      CALL lmint0( WTFP, 114, 0.0, TooBig )
      CALL lmint0( DWFIG(1), 115, 0.0, TooBig )
```

```
      CALL lmint0( WOPOV, 116, 0.05, TooBig )
      CALL unint0( WOPOI, 117 )
      CALL lmint0( DWOPO, 118, 0.0, TooBig )
      CALL unint0( WTIGN, 119 )
      CALL unint0( WTASI, 120 )
      CALL unint0( WOPTV, 121 )
      CALL lmint0( DWOPTV, 122, 0.0, TooBig )
      CALL unint0( WOPVI, 123 )
      CALL lmint0( DWOPVI, 124, 0.0, TooBig )
      CALL unint0( WOPOI, 128 )
      CALL unint0( WOPTI, 125 )
      CALL lmint0( DWOPTI, 126, 0.0, TooBig )
      CALL unint0( WOPOT, 127 )
      dwmax = .002 / DT
      CALL unint0( DWOPOI, 129 )
      CALL unint0( POPOI, 134 )
      CALL unint0( WOPTA, 130 )
      CALL lmint0( DWOPTA, 130, 0.0, TooBig )
      CALL unint0( POPRG, 131 )
      CALL unint0( POPOV, 132 )
      CALL unint0( POPOT, 133 )
      CALL lmint0( WOPF, 135, 0.0, TooBig )
      CALL lmint0( WOPO, 136, 0.0, TooBig )
      CALL lmint0( WTOP, 137 )
      CALL unint0( TFT2DI, 138 )
      CALL lmint0( WTFI, 139, 0.0, TooBig )
      CALL lmint0( WFIF, 140, 0.0, TooBig )
      CALL lmint0( WFIO, 141, 0.0, TooBig )
      CALL lmint0( WTC1, 142, 0.0, TooBig )
      CALL lmint0( WCO, 143, 0.0, TooBig )
      CALL lmint0( WCF, 144, 0.0, TooBig )
      CALL lmint0( WCN, 145, 0.0, TooBig )
*
      ENTRY HOTGAS
**********************************************************************
C
C                FUEL PREBURNER INJECTION FLOWS
C
**********************************************************************
C   IN THE FOLLOWING, IGOFU IS USED TO INDICATE THE STATE OF THE PREBURNER
C   INJECTION FLOWS, THEY ARE:
C   IGOFU=1:         INITIALIZATION
C   IGOFU=2:         BEFORE THE INJECTOR AND MFV ARE PRIMED
C   IGOFU=3:         MAIN STAGE, INJECTOR AND MFV ARE PRIMED
C   IGOFU=4:         MAIN STAGE TO PURGE TRANSITION
C   IGOFU=5:         FLOW LINES PURGE, POWER CUT
C
C   VALVE AND INJECTOR PRIMING IS THE DYNAMIC THAT THE INJECTOR STARTS
C   EMPTY AND REQUIRES TO BE FILLED BEFORE FULL INJECTION CAN HAPPEN.
C          CFOV IS THE FACTOR OF TRANSFER FOR FPOV
```

```
C           CFOI IS THE FACTOR OF TRANSFER FOR OXID INJECTOR
C   THEY ARE FUNCTIONS OF THE PERCENTAGE OF FILL OF THE EMPTY SPACE.
*****************************************************************
*
*  The convoluted logic of the restart conditions was commented, for
*  the sake of maintenance. Single precision time is good enough for
*  state transition tests.
C
C           TRANSIENT CALCULATION SECTION
C
      GO TO (1200, 1240, 1260, 1280, 1320), IGOFU                    SSM30800
*
*   Test restart time once ( IGOFU is changed ).
*
 1200 IF ( STIME .GT. 2.5) THEN
        CFOV = 1.0
        WFPOI = WFPOIZ
        CFOI = 1.0
        WFPOV = 0.05                                                 SSM30900
        FPIG = .TRUE.
*                              Bypass OPRIME
        IGOFU = 3
        GO TO 1260
      ELSE IF ( STIME .LT. 1.5) THEN
        CFOV = 0.025
*                              OPRIME until time = 1.5
        IGOFU = 2
      END IF
C     *          *          *          PRIME FPB OXIDIZER INJECTOR
C
 1240 CALL OPRIME(POD3, PFP-DPFPAS, RFPOV, DWFPOI, DWFPO)
*
*     RHOFGN = PFPOV / (1159.0 * 200.0)  was replaced by
      RHOFGN = PFPOV * 4.314064E-7
*                  When time reaches 1.5,
*                   -     stop OPRIME, start mainstage
      IF( TIME .LT. 1.5 ) GO TO 1280
*                  if 1.5 < restart time < 2.5,
*                        do OPRIME once, then start mainstage
      IGOFU = 3                                                      SSM31000
      GO TO 1340
C     *          *          *          MAINSTAGE CALCULATIONS       SSM31300
C
 1260 RFPO = (RFPOL + RFPOV + RFPOI) / RHOOP3
      DWFPO = prflow( DWFPO, ZFPO, RFPO, POD3 - PFP + DPFPAS, 95 )
      DWFPOI = prflow( DWFPOI, ZFPOI,
     +                    RFPOI/RHOOP3, PFPOI-PFP+DPFPAS, 96 )
*
*     IF (XFPOV .GT. 20. .OR. PFPOV .GT. PHES) GO TO 1280 , that is
      IF ( XPOV .LE. 20. .AND. PFPOV .LE. PHES ) THEN
```

```
*                    stop integrating DWFPO and DWFPOI
         TDWFPO = 0.
         IGOFU = 4
       END IF
C
C   DPFPAS IS THE SUCTION PRESSURE DUE TO BERNOULLI EFFECT OF
C   THE FUEL FLOW INSIDE THE FUEL PREBURNER
C
*1280 DPFPAS = ASFPC * (DWFPF / 258.0) ** 2 / (PFP / (9272.0 * T(9)))
*                     is replaced by
 1280 DPFPAS = ASFPC * DWFPF**2 * T(9) / PFP * 0.1392945
*
       PFPOI = PFP + RFPOI * DWFPO**2 / RHOOP3 - DPFPAS
       PFPOV = (POD3 / RFPOV + PFPOI / RFLEAK) * (RFPOV * RFLEAK
SSM31400
     1   / (RFPOV + RFLEAK))
       IF (TIME.GT.TCUTPR .AND. PFPRG.LT.100.)  PFPRG = AMIN1(PHES,PFPOV)
       IF (IGOFU .LT. 4) GO TO 1340
       IF (TIME .GT. TCUTPR .AND. PFPOI .LT. PHES) THEN
C
C                       CUT OXID LINES AND START HELIUM PURGE
         IGOFU    = 5
       ELSE
         DWFPOI = recpos( -DWOIG(1) )
C
C!!!!! I CHANGE THE SECOND EQUATION TO THE ONE FOLLOWS.  THIS IS THE SITUATI
C!!!!! WHEN THE FUEL LINE IS PURGED BY THE HELIUM.
C      WFPOI = WFPOI - recneg( DWFPOI) * DT
*
         WFPOI = pruint( - DWFPOI, Tstep, 97 )
         GO TO 1340
       END IF
C
C   THIS IS TO PURGE THE OXID CONTENT IN LINE FPTV (BETWEEN PFPOT AND PFPOV)
C
*    PFPTV was changed from a real variable, tested for 0.0, to a
*    logical value. Similar logical variables were introduced throughout.
*    Limiting the integrated value to positive values was an unnecessary
*    operation, and was dropped.
*
 1320 IF ( FFPTV ) THEN                                              SSM31500
        WFPTV = pruint( - ABS(DWFPTV), Tstep, 98 )
        FFPTV = WFPTV .GT. 0.0
*
*        DWFPTV = FLOW(DWFPTV, ZFPOI, RFPOTV/RHOOP3, DT, PFPOT - PFPOV)
*        IF (PFPOT .LT. PFPOV)   DWFPTV = 0. - FLOW(DWFPTV,
*     1  ZFPOI, RFPOVT/RHOOP3, DT, PFPOV - PFPOT)
*        RHOFTV = PFPOT / 4632. / TPRC
*         DWFPTV = GFLOW (PFPOT, PFPOV, TPRC, 4632., 1.66) * AFPTV
*      IF (PFPOT .LT. PFPOV)   DWFPTV = 0. - GFLOW (PFPOV, PFPOT, TPRC,
```

```
*     1  4632., 1.66) * AFPVT                                          SSM31600
*
*     The above sequence was replaced. It does an integration step, then
*     if the wrong sign was used, does it again. The replacement code
*     looks first, then does the integration step once.
*     Also, since the second integration does not depend on RHOFTV,
*     there is no need to test twice. Similar replacements are made
*     below without comment.
*
          IF (PFPOT .LT. PFPOV) THEN
            DWFPTV = 0. -
     +       prflow(DWFPTV, ZFPOI, RFPOVT/RHOOP3, PFPOV - PFPOT, 99 )
            DWFPTV = 0. -
     +       GFLOW (PFPOV, PFPOT, TPRC, 4632., 1.66) * AFPVT          SSM31600
          ELSE
            DWFPTV =
     +       prflow( DWFPTV, ZFPOI, RFPOVT/RHOOP3, PFPOT - PFPOV, 99 )
            DWFPTV = GFLOW (PFPOT, PFPOV, TPRC, 4632., 1.66) * AFPTV
          END IF
          RHOFTV = PFPOT * vTP46
        END IF
C
C    THIS IS TO PURGE THE OXID CONTENT IN LINE FPVI (BETWEEN PFPOI AND PFPOV)
C
        IF ( FFPVI ) THEN
           WFPVI = pruint( - DWFPVI, Tstep, 100 )
           FFPVI = WFPVI .GT. 0.0
           DWFPVI = prflow( DWFPVI, ZFPOI,
     +                          RFPOVI/RHOOP3, PFPOV - PFPOI, 101 )
        ELSE
           RHOFVI = PFPOV * vTP46
           WFPVI = GFLOW (PFPOV, PFPOI, TPRC, 4632., 1.66) * AFPVI
        END IF
C
C    THIS IS TO PURGE THE OXID CONTENT IN LINE FPTI (BETWEEN PFPOT AND PFPOI)
C
        IF ( FFPTI .OR. FFPOT ) THEN
           IF ( .NOT. FFPOT ) WFPTI = pruint( - DWFPTI, Tstep, 102 )
           FFPTI = WFPTI .GT. 0.0                                     SSM31700
           DWFPTI = prflow(DWFPTI, ZFPOI,
     +                        RFPOTI/RHOOP3, PFPOT - PFPOI, 103 )
        ELSE
           RHOFTI    = PFPOT * vTP46
           DWFPTI    = GFLOW (PFPOT, PFPOI, TPRC, 4632., 1.66) * AFPTI
        END IF
        IF ( FFPOT ) THEN
          WFPOT = pruint( - (DWFPTV + DWFPTI+ DWOIG(1) ), Tstep, 104 )
          FFPOT = WFPOT .GT. 0.0
        END IF
        IF ( FFPOI ) THEN
```

```
      WFPOI = pruint( - DWFPOI, Tstep, 105 )
      FFPOI = WFPOI .GT. 0.0                                          SSM31800
      WDFPOI = DWFPOI
      DWFPOI = prflow( DWFPOI, ZFPOI, -RFPOI/RHOOP3,
     +                                    PFPOI-PFP+DPFPAS, 106 )
      DWFPOI = WDFPOI - AMIN1(WDFPOI - DWFPOI, 0.002)
      IF (DWFPOI.GT.WDFPOI )
     +       DWFPOI=WDFPOI + AMIN1(DWFPOI - WDFPOI ,.002)
*
*       AREAF = TABLX (4, ISV, WFPOI, WFPTAB, AFPTAB) replaced with
*                       interpolator using precomputed slopes
*
      AREAF = xlint( WFPOI, 4, WFPTAB, AFPTAB, sawfp, ISV )
*
        DWPFI = GFLOW (PFPOI, PFP - DPFPAS, TFPDH, 4632., 1.66) * AREAF
      ELSE
        DWFPOI =  0.0
        DWPFI = GFLOW (PFPOI, PFP - DPFPAS, TPRC, 4632., 1.66) * AFPOI
      END IF
C
C   PURGE IGNITOR INJECTOR LINE
C
      IF (FFPTA .OR. FFPOT ) THEN                                     SSM31900
         IF ( .NOT. FFPOT ) WFPTA = pruint( - DWOIG(1), Tstep, 107 )
         FFPTA = WFPTA .GT. 0.0
         DWOIG(1) = prflow(DWOIG(1), ZFPOI, RFPOTA / RHOOP3,
     +                                        PFPOT - PFP, 108 )
      ELSE
         RHOFTA = PFPOT * vTP46
         DWOIG(1) = GFLOW(PFPOT, PFP, TPRC, 4632., 1.66) * AFPTA
      END IF
      DPFPAS = ASFPC * DWFPF**2 * T(9) / PFP * 0.1392945
*     DPFPAS   = ASFPC * (DWFPF / 258.) **2 / (PFP / 9272 / T(9))
      DWFPR1 = recpos( GFLOW(PHES, PFPOV, TPRG, 4632.0, 1.66) * AFPR1 )
      PFPOV = pruint(
     +  DWFPR1 + CFACT *( DWFPTV/ RHOFTV - DWFPVI / RHOFVI ) *
     +       PFPOV * vTP46, Tstep, 109 )                            SSM32000
*     PFPRG    = PFPOV + .9422 * 4632 * TPRG / PFPOV * DWFPR1 ** 2
      PFPRG = PFPOV + 4364.3 * TPRG / PFPOV * DWFPR1 ** 2
      DWFPR2 = recpos( GFLOW(POPRG, PFPOT, TPRG, 4632.0, 1.66) * AFPR2 )
      PFPOT = pruint( CFACT * ( DWFPR2 -
     + ( DWOIG(1)/RHOFTA + DWFPTI/RHOFTI + DWFPTV/RHOFTV ) *
     +   PFPOT * vTP46 ) ), Tstep, 110 )
      PFPOI = pruint( CFACT * (
     + ( DWFPTI / RHOFTI + DWFPVI / RHOFVI - DWFPOI / RHOFOI ) *
     + ( PFPOI * vTP46 ) - DWPFI), Tstep, 111 )
C
C    END OF PURGE PROCESS
C
C     FUEL PREBURNER COMBUSTION
```

```
C
 1340 CONTINUE
C
C    IN THE FOLLOWING:
C    DWFPF:   FUEL PREBURNER FUEL FLOW
C    DWFPBI:  FUEL PREBURNER TOTAL INLET FLOW
C    WFPF:    FUEL PREBURNER TOTAL FUEL
C    WFPO:    FUEL PREBURNER TOTAL OXIDIZER
C    ELFFPM:  FUEL PREBURNER OXIDIZER FRACTION
C    ELFFP:   FUEL PREBURNER OXIDIZER FLOW FRACTION
C    FPIG:    IGNITOR INDICATOR, = .TRUE. FOR FUEL PREBURNER IGNITED
C    TFPC:    FUEL PREBURNER COMBUSTION TEMPERATURE
C    CPFP:    SPECIFIC HEAT INSIDE FUEL PREBURNER
C    GAMFP:   RATIO OF SPECIFIC HEAT (GAMMA) OF FUEL PREBURNER
C    EMWFP:   MOLECULAR WEIGHT IN FUEL PREBURNER
C    RGCFP:   GAS CONSTANT IN FUEL PREBURNER (PSI-IN**3/LB)
C    TFP:     FUEL PREBURNER TEMPERATURE
C    PFP:     FUEL PREBURNER PRESSURE
C    WFP:     FUEL PREBURNER TOTAL MASS
C    WTFP:    WEIGHT TIMES TEMPERATURE WITHIN FUEL PREBURNER
C
      DWFPFA = recpos( DWFPF )                                        SSM32100
      posdwo = recpos( DWOIG(1) )
      DWFPBI = DWFPFA + DWFPOI + DWFIG(1) + posdwo
C
C    DURING THE PURGE FFPOI = .FALSE.
C
      IF ( .NOT. FFPOI ) DWFPBI = DWFPFA + DWPFI + DWFIG(1) + posdwo
*
*     WFPF=AMAX1(0.0,WFPF+(DWFPFA+DWFIG(1)-(1.0-ELFFPM)*DWFT2)*DT)
*
      WFPF = prlint( DWFPFA + DWFIG(1) - (1.0 - ELFFPM)*DWFT2, 0, 112 )
      WFPO = prlint( DWFPOI + posdwo - ELFFPM * DWFT2, 0, 113 )
      ELFFPM = WFPO / (WFPO + WFPF + 1.0E-6)
      ELFFP = ( DWFPOI + posdwo ) /
     +        ( DWFPOI + DWFPFA + DWFIG(1) + posdwo + 1.0E-06 )
      IF( .NOT. FPIG ) THEN
        TFPC = T(9) + 20.                                             SSM32200
        IF( ELFFP .GE. 0.20 .AND. STIME .GT. 0.3 ) THEN
          FPIG = .TRUE.
          WRITE(6,*)' FPB IGN AT',TIME
        END IF
      ELSE
        TFPC = (1.0927E-05 * PFP + 0.91985) * FGEN(20, 66,ELFFP) + T(9)
      END IF
      CPFP = FGEN(21, 81, ELFFPM)
C
C    SPECIAL CONSIDERATION FOR SMALL O/F RATIO
C
      IF (ELFFPM .GT. 0.1) A = recpos(1.0 - 10.0*ELFFPM)              SSM32300
```

```
      GAMFP = A * H2GAMA(PFP,TFP,1,2) + (1.0 - A)*FGEN(13, 68,ELFFPM)
      CPH2 = FGEN(15, 61,TFP) - 0.0887 + FPF * (0.1241 - 3.732E-5*PFP) /
     +     ( AMAX1(51.,TFP) - 50.)
      CPFP = A*CPH2 + (1.0 - A)*CPFP
      GO TO 1380
 1360 GAMFP = FGEN (13, 68, ELFFPM)
 1380 CONTINUE
      EMWFP = FGEN(12, 69, ELFFPM/(1.-ELFFPM))
      RGCFP = 18540.0 / EMWFP
      TFP = AMAX1(10.0, WTFP / (WFP+1.0E-12))                          SSM32400
      PFP = RGCFP * WTFP / VOLFP
      PFP = AMAX1(0.01,PFP)
      WTFP = prlint( GAMFP*(DWFPBI*TFPC - DWFT2*TFP), 0, 114 )
C
C   WFPGN2 IS NEVER DEFINED EXCEPT INITIALIZED TO 0.0 AT THE BEGINING
C
      WFP=WFPF+WFPO+WFPGN2
C
C                FUEL PREBURNER IGNITOR
C
      PFPBFB = P(9) - RFPIGB / RHO(9) * DWFPF**2
      IF (IGOFU .EQ. 5) THEN
        DWFIG(1) =
     +  prflow( DWFIG(1), ZFIG(1), RFIG(1) / RHO(9), PFPBFB - PFP, 115 )
      ELSE                                                             SSM32500
        CALL IGN (1, PFPOV, PFPOI, RHOFGN, PFPBFB, RHO(9), PFP, 2)
        PFPOT = P4(1)
        POPOT = P4(2)
        DWFPTV = -DW1(1)
        DWFPTI = DW2(1)
      END IF
C
C      HIGH PRESSURE FUEL TURBINE PERFORMANCE
C
C   TURBINE FLOW EFFICIENCY FDWFT2 IS THE FUNCTION OF EFFECTIVE PRESSURE
C   RATIO (PREFT2) AND NORMALIZED TURBINE SPEED (FSF2).  A LOOK-UP TABLE
C   IS GIVEN AS DWFTAB WITH FPRF ARRAY AS X-AXIS AND FSFTAB ARRAY AS Y-AXIS.
C   THE ACTUAL EQUATIONS USED FOR SIMULATION IS NOT IN THE DOCUMENT RL00001.
C
      GAMP1 = GAMFP + 1.                                               SSM32600
      GAMM1 = GAMFP - 1.
      PRFT2 = AMAX1 (1.0, PFP/PFT2D )
*
*   X ** Y may require evaluation of a slowly converging series for the
*   logarithm of X, and another series for the exponentiation, after the
*   multiplication by Y. Replacement routines involving two - way linear
*   interpolation in a table of equal intervals were defined.
*
*     PREFT2 = (1.-.166667*GAMP1/GAMM1*
*    +                (1. - PRFT2**(-GAMM1/GAMFP)))**(-3.5)
```

```
*
      preft2 = 1. - .166667 * GAMP1 /
     +                   (1. - 1. / XtoY( PRFT2, GAMM1/GAMFP ) )
      PREFT2 = 1.0 / ( preft2 ** 3 * X10th(preft2, 5) )

      TCRFT2 = ( GAMFP/GAMP1 * RGCFP * TFP * 5.15917E-6 )
*                                        (2.*32.174/12./1019.5**2)
*    The square root is taken immediately, since it is used more than
*    once. Since the X ** Y interpolation routines use a table of
*     X ** (.1 * n ) values, the square root can be obtained faster
*     as X ** .5, than by the square root function.
*
*      tcsqrt = X10th( TCRFT2 )
*
      FSF2 = SF2 / tcsqrt * 9.5493
*                                (60./2*PI)
C
*      FDWFT2=TABLXY(NPREF,IF1,PREFT2,FPRF,NFSF,IF2,FSF2,FSFTAB,DWFTAB)
*    was replaced by an interpolation using precomputed slopes.
*
      FDWFT2 = xylint( PREFT2, FSF2, NPREF, FPRF, NFSF, FSF2,
     +                              sdwfp, vdfs FSFTAB, DWFTAB, IF1, IF2 )
*
*      A table of EPSF vs GAMFP is needed here. It would be used for
*      EPSO vs GAMOP as well.
*
*       EPSF  = .739594/GAMFP*(2./GAMP1)**(-GAMFP/GAMM1)
      EPSF  = .739594 / ( GAMFP * XtoY( 2./GAMP1, GAMFP/GAMM1 ) )
      DWFT2 = FDWFT2*CDWFT2*PFP/(14.7*tcsqrt*EPSF)
      IF (STIME.LT.0.1) THEN
         TRQFT2=0.0
      ELSE
         TRQFT2=TRBTRQ(SF2,UCFT2,TFP,PFP,1./PRFT2,2,CPFP,DWFT2,GAMFP)
     1        *CTQFT2
      END
      IF( DWFT2.LE..0) THEN
        TFT2D = TFP
      ELSE
        TFT2D=TFP-rlimit( 0.0, 300.0, (TRQFT2 * SF2)/
     +                           (9340.0*CPFP*DWFT2 + 1.0E-6) )
      ENDIF
      hpfi = 0.5 * PFI
      PFT2D = hpfi + SQRT( hpfi**2 + RHGFM*RGCFP*TFT2D*DWFT2**2)
C                                                                      SSM32800
C                OXID  PREBURNER INJECTION FLOWS
C
C
C*************************************************************************
C   IN THE FOLLOWING, IGOFU IS USED TO INDICATE THE STATE OF THE PREBURNER
C   INJECTION FLOWS, THEY ARE:
```

```
C    IGOOX=1:          INITIALIZATION
C    IGOOX=2:          BEFORE THE INJECTOR AND MOV ARE PRIMED
C    IGOOX=3:          MAIN STAGE, INJECTOR AND MOV ARE PRIMED
C    IGOOX=4:          MAIN STAGE TO PURGE TRANSITION
C    IGOOX=5:          FLOW LINES PURGE, POWER CUT
C
C    VALVE AND INJECTOR PRIMING IS THE DYNAMIC THAT THE INJECTOR STARTS
C    EMPTY AND REQUIRES TO BE FILLED BEFORE FULL INJECTION CAN HAPPEN.
C             COOV IS THE FACTOR OF TRANSFER FOR OPOV
C             COOI IS THE FACTOR OF TRANSFER FOR OXIDIZER INJECTOR
C    THEY ARE FUNCTIONS OF THE PERCENTAGE OF FILL OF THE EMPTY SPACE.
C*************************************************************************
C
       IF (WTIGN .GT. 0.0)  RHOIGN    = RHOOP3
       GO TO (1500, 1540, 1560, 1580, 1680), IGOOX
 1500 ROLEAK = 1.0 / (772.8 * (FGEN(17, 82, 20.) * ABOPO / 100.) ** 2)
       WOPOIZ  = FGEN(1, 83, 0.999) + 0.01
       IF (STIME .GT. 2.5)          GO TO 1520
       COOV       = 0.025
       IGOOX      = 2
       GO TO 1540                                                    SSM32900
 1520 COOV       = 1.0
       WOPOI      = WOPOIZ
       COOI       = 1.0
       WOPOV      = 0.05
       IGOOX      = 3
              GO TO 1560
C      *           *           *           PRIME OPB OXIDIZER INJECTOR
C
*1540 WOPOV = AMIN1 (0.05, WOPOV + (1.0 - COOV) * DWOPO * DT)
*
 1540 WOPOV = prlint( (1.0 - COOV) * DWOPO, Tstep, 116 )
       CALL pruint( WOPOI, DWOPO * COOV - DWOPOI, Tstep, 117 )       SSM33000
       COOI = FGEN(1, 62, WOPOI)
C      IF (WOPOV .GT. 0.0)  COOV = 1.0
C
       IF( OPOVPF .AND. WOPOV.GT.-.05) THEN
         COOV = 1. + WOPOV * 20.
         IF(WOPOV.GT.-.001) THEN
           PRINT *,' OPOV BUBBLE PRIMED AT',TIME
           OPOVPF = .FALSE.
           COOV=1.
         END IF                                                      SSM33100
       END IF
C
       IF( OPOIPF .AND. WOPOI.GT.1.75) THEN
         PRINT *,' OPB INJECTOR PRIMED AT',TIME
         OPOIPF = .FALSE.
       END IF
C
```

```
      ROPO      = (ROPOL + ROPOV * COOV + ROPOI * COOI * COOV) / RHOOP3
      DWOPO = prFLOW( DWOPO, ZOPO, ROPO, POD3 - POP + DPOPAS, 118 )
      DWOPOI    = COOI * COOV * DWOPO                                  SSM33200
*     RHOIGN    = POPOV / (1159.0 * 200.0)
      RHOIGN = PFPOV * 4.314064E-7

      IF (WOPOI .LT. WOPOIZ)     GO TO 1580
      WOPOI     = WOPOIZ
      IGOOX     = 3
C     *          *          *          MAINSTAGE CALCULATIONS
C
 1560 ROPO = (ROPOL + ROPOV + ROPOI) / RHOOP3
      IF (ROPOV .LT. 1.0E+10) THEN                                    SSM33300
         DWOPO = prflow( DWOPO, ZOPO, ROPO, POD3 - POP + DPOPAS, 118 )
         DWOPOI = DWOPO
      ELSE
         DWOPO     = 0.0
         DWOPOI    = 0.0
         IGOOX     = 4
      END IF
*1580 DPOPAS    = ASOPC * (DWOPF / 120.)**2 / (POP  / (9272. * T(9)))
 1580 DPOPAS = ASOPC * (DWOPF**2 * T(9) / POP * 0.1392945

      POPOI = POP + ROPOI * DWOPOI**2 / RHOOP3 - DPOPAS
      POPOV = (POD3 / ROPOV + POPOI / ROLEAK) * (ROPOV * ROLEAK)
     1  / (ROPOV + ROLEAK)
      IF (IGOOX .LT. 4) GO TO 1800
C     *          *          *          COMPUTE OPB ASPIRATION BACKFLOW
C                                                                     SSM33400
      DWOPOI = recpos( -DWOIG(2) * RHOOP3 / RHOIGN )
      WOPOI = pruint( DWOIG(2), Tstep, 117 )
*
* Changed next two tests to be independent of DT and consistent with
* previous approach.
*
      IF ( wtigon ) THEN
         WTIGN = pruint( DWOIG(2), Tstep, 119 )
         IF ( WTIGN .GT. 0.0) THEN
            WRITE (init, 1590) STIME
 1590 FORMAT (1H0, 10X,
     + ' *  *  *  *  BACKFLOW REACHED ORIFICE BLOCK AT TIME = ',
     +  F8.3, ' SECONDS  *  *  *  * ' )
            wtigon = .FALSE.
            GO TO 1640
         END IF
      END IF
 1600 RHOIGN = PCIG(2) / (9272.016 * T(9))
      IF( wtason ) THEN
        WTASI = pruint( DW1(2) * RHOOP3 / RHOIGN, Tstep, 120 )
        IF (WTASI .LT. 0.0) THEN
```

```
            WRITE (6,1610) TIME
 1610 FORMAT (1H0, 10X,
     +   ' *  *  *  *  BACKFLOW REACHED OPOV AT TIME = ',
     +            F8.3, ' SECONDS  *  *  *  * ' )
            wtason = .FALSE.
          ENDIF
        ENDIF
 1640 IF (TIME .LT. TCUTPR)      GO TO 1800
        POPRG    = AMIN1 (PHES, POP)
        IF (POPOI .GT. PHES)      GO TO 1800
        IGOOX    = 5
C
C     ******     COMPUTE  PURGE  FLOW  AND  PRESSURE
C
C     OXIDIZER PREBURNER OXIDIZER LINES HAVE THE SAME SET UP AS THOSE OF
C     FUEL PREBURNER.  SAME PURGE SEQUENCE APPLIED.
C
*   The code was restored to read like 1320 above, where a logical
*   variable shuts off the integration when its state change function
*   is completed.
*
 1680 IF ( FOPTV ) THEN                                          SSM33600
        WOPTV = pruint( - ABS(DWOPTV), Tstep, 121)
        FOPTV = WOPTV .GT. 0.
        IF (POPOT .LT. POPOV) THEN
          DWOPTV = 0. - prflow (DWOPTV, ZOPOI, ROPOVT / RHOOP3,
     +                          POPOV - POPOT, 122 )
        ELSE
          DWOPTV = prflow(DWOPTV, ZOPOI, ROPOTV/RHOOP3,
     +                          POPOT - POPOV, 122 )
        ENDIF
      ELSE
        RHOOTV = POPOT * vTP46
        IF (POPOT .LT. POPOV) THEN
           DWOPTV = 0. -
     +        GFLOW (POPOV, POPOT, TPRC, 4632., 1.66) * AOPVT
        ELSE
           DWOPTV = GFLOW (POPOT, POPOV, TPRC, 4632., 1.66) * AOPTV
        ENDIF
      ENDIF
      IF ( FOPVI ) THEN
        WOPVI = pruint( - DWOPVI, Tstep, 123)
        FOPVI = WOPVI .GT. 0.0
        IF ( FOPVI ) THEN
          DWOPVI =
     +     prflow( DWOPVI, ZOPOI, ROPOVI/RHOOP3, POPOV - POPOI, 124 )
        ELSE
          RHOOVI = POPOV * vTP46
          DWOPVI = GFLOW(POPOV, POPOI, TPRC, 4632., 1.66) * AOPVI
        ENDIF
```

```
      ENDIF
      IF ( FOPTI ) THEN
        WOPTI = pruint( DWOPTI, Tstep, 125 )
        FOPTI = WOPTI .GT. 0.0                                        SSN33800
        IF ( FOPTI .OR. FOPOT ) THEN
          DWOPTI = prflow(DWOPTI, ZOPOI, ROPOTI/RHOOP3,
     +                                      POPOT - POPOI, 126 )
        ELSE
          RHOOTI = POPOT * vTP46
          DWOPTI = GFLOW(POPOT, POPOI, TPRC, 4632., 1.66) * AOPTI
        END IF
      END IF
 1730 IF ( FOPOT ) THEN
        WOPOT = pruint( -(DWOPTV + DWOPTI+ DWOPTA), Tstep, 127 )
        FOPOT = WOPOT .GT. 0.0
      END IF
      IF ( FOPOI ) THEN
        WOPOI = pruint( - DWOPOI, Tstep, 128 )
        FOPOI = WOPOI .GT. 0.0                                        SSM33900
*
*  Limit integration increase or decrease to .002, obscurely.
*
*     WDOPOI = DWOPOI
*     DWOPOI = FLOW(DWOPOI, ZOPOI, ROPOI/RHOOP3,DT, POPOI-POP+DPOPAS)
*     DWOPOI = WDOPOI - AMIN1 (WDOPOI - DWOPOI, 0.002)
*     IF (DWOPOI.GT.WDOPOI ) DWOPOI=WDOPOI +AMIN1(DWOPOI-WDOPOI ,.002)
*
*  Now modelled as an unlimited integration of a limited flow rate.
*
      dwrate = rlimit( -dwmax, dwmax,
     +            POPOI - POP + DPOPAS - ROPOI/RHOOP3*DWOPOI**2/ZOPOI )
      DWOPOI = pruint( dwrate, Tstep, 129 )
*
        AREAO = xlint( WOPOI, ISO, WOPTAB, AOPTAB, sawop, 4)
        DWPOI = GFLOW (POPOI, POP - DPOPAS, TOPDH, 4632., 1.66) * AREAO
      ELSE
        RHOOOI = POPOI * vTP46
        DWOPOI = GFLOW(POPOI, POP - DPOPAS, TPRC, 4632., 1.66) * AREAO 4000
      END IF
      IF ( FOPTA ) THEN
        WOPTA = pruint( - DWOPTA, Tstep, 130)
        FOPTA = WOPTA .GT. 0.0
        IF ( FOPTA ) THEN
          DWOPTA = prflow( DWOPTA, ZOPOI, ROPOTA / RHOOP3,
     +                                          POPOT - POP, 130 )
        ELSE
          RHOOTA = POPOT * vTP46
          DWOPTA = GFLOW (POPOT, POP, TPRC, 4632., 1.66) * AOPTA
        ENDIF
      ENDIF
```

```
      DPOPAS = ASOPC * DWOPF**2 / POP * T(9) * 0.1392945
      DWFPSG = GFLOW(PHES, POPRG, TPRG, 4632., 1.66) * AFPRS
      POPRG = pruint(
     +         48849. * ( DWFPSG - DWFPR2 - DWOPR - DWOPR2),
     +                                                  Tstep, 131 )
      POPRG = AMIN1( PHES - .01, POPRG )                          SSM34100
      DWOPR = GFLOW (POPRG, POPOV, TPRG, 4632.0, 1.66) * AOPR1
      POPOV = pruint( CFACT * ( DWOPR +
     +  (DWOPTV/RHOOTV - DWOPVI/RHOOVI ) * POPOV * vTP46, Tstep, 132 )
      DWOPR2 = GFLOW (POPRG, POPOT, TPRG, 4632.0, 1.66) * AOPR2
      POPOT = pruint( CFACT * (DWOPR2 -
     + (DWOPTA/RHOOTA + DWOPTI/RHOOTI + DWOPTV/RHOOTV) *
     +                                    (POPOT * vTP46), Tstep, 133 )
      POPOI = pruint( CFACT * ((DWOPTI / RHOOTI + DWOPVI / RHOOVI
     +      - DWOPOI / RHOOOI) * (POPOI * vTP46 - DWPOI), Tstep, 134 )
C
C   END OF OXIDIZER PREBURNER PURGE
C
 1800 CONTINUE                                                    SSM34200
C
C                OXID PREBURNER IGNITOR
      POPBFB = P(9) - ROPIGB / RHO(9) * DWOPF**2
      CALL IGN (2, POPOV, POPOI, RHOIGN, POPBFB, RHO(9), POP)
C
C                OXID PREBURNER COMBUSTOR
C
C   IN THE FOLLOWING:
C   DWOPF:  OXID PREBURNER FUEL FLOW
C   WOPF:   OXID PREBURNER TOTAL FUEL
C   WOPO:   OXID PREBURNER TOTAL OXIDIZER
C   ELFOPM: OXID PREBURNER OXIDIZER FRACTION
C   ELFOP:  OXID PREBURNER OXIDIZER FLOW FRACTION
C   TOPC:   OXID PREBURNER COMBUSTION TEMPERATURE
C   CPOP:   SPECIFIC HEAT INSIDE OXID PREBURNER
C   GAMOP:  RATIO OF SPECIFIC HEAT (GAMMA) OF OXID PREBURNER
C   EMWOP:  MOLECULAR WEIGHT IN OXID PREBURNER
C   RGCOP:  GAS CONSTANT IN OXID PREBURNER (PSI-IN**3/LB)
C   TOP:    OXID PREBURNER TEMPERATURE
C   POP:    OXID PREBURNER PRESSURE
C   WOP:    OXID PREBURNER TOTAL MASS
C   WTOP:   WEIGHT TIMES TEMPERATURE WITHIN OXID PREBURNER
C
      DWOPFA = recpos(DWOPF)
      WOPF = prlint( DWOPFA + DWFIG(2) - (1.0 - ELFOPM) * DWOT2,
     +                                                    0, 135 )
      posig = recpos( DWOIG(2) )
      WOPO = prlint( DWOPOI + posig - ELFOPM*DWOT2, 0, 136 )
      ELFOPM = WOPO / (WOPO + WOPF + 1.0E-6)                      SSM34300
      ELFOP = ( DWOPOI + posig ) /
     +   (DWOPOI + DWOPFA + DWFIG(2) + posig + 1.0E-06)
```

```
      TOPC = FGEN(20, 66, ELFOP)+T(9)
      CPOP = FGEN(21, 67, ELFOPM)
C
C   SPECIAL CONSIDERATION WHEN O/F RATIO IS REALLY SMALL
C
      IF(ELFOPM.LE.0.1) THEN
        A = recpos(1.0 - 10.*ELFOPM)
        GAMOP = A * H2GAMA(POP, TOP, 2, 2) +
     +                  (1.0 - A)*FGEN(13, 68,ELFOPM)
        CPH2 = FGEN(15, 61,TOP) - 0.0887 +
     +     POP * (0.1241 - 3.732E-5*POP) / (AMAX1(51.,TOP)-50.)
        CPOP= A * CPH2 + (1.0 - A)*CPOP                          SSM34400
      ELSE
        GAMOP = FGEN(13, 68, ELFOPM)
      ENDIF
      EMWOP = FGEN(12, 69, ELFOPM/(1. - ELFOPM) )
      RGCOP = 18540.0 / EMWOP
      TOP = AMAX1(10.0, WTOP/(WOP + 1.0E-12) )
      POP = RGCOP*WTOP/VOLOP
      POP=AMAX1(0.01,POP)
      WTOP = prlint( GAMOP *
     + (recpos(DWOPFA) + DWOPOI + DWFIG(2) + posig)*TOPC
     +                                          - DWOT2*TOP, 0, 137
)SSM34500
      WOP = WOPF + WOPO + WOPGN2
      IF(POP.LT.PA) DWOPOI=0.
C
C                HIGH PRESSURE OXIDIZER TURBINE
C
C   TURBINE FLOW EFFICIENCY FDWOT2 IS THE FUNCTION OF EFFECTIVE PRESSURE
C   RATIO (PREOT2) AND NORMALIZED TURBINE SPEED (FSO2).  A LOOK-UP TABLE
C   IS GIVEN AS DWOTAB WITH FPRO ARRAY AS X-AXIS AND FSOTAB ARRAY AS Y-AXIS.
C   THE ACTUAL EQUATIONS USED FOR SIMULATION IS NOT IN THE DOCUMENT RL00001.
C
      GAMP1 = GAMOP + 1.
      GAMM1 = GAMOP - 1.
      PROT2 = AMAX1 (1.0, POP/POT2D )
      tmp = 1. - .166667 * GAMP1 / GAMM1 *
     +            (1. - XtoNY( PROT2, -GAMM1/GAMOP) )
      PREOT2 = 1. / ( X10th(tmp, 5) * tmp**3 )
      TCROT2 = GAMOP/GAMP1 * RGCOP * TOP * 5.15917E-6
      tcsqrt = X10th(TCROT2, 5)
      FSO2 = SO2 / tcsqrt * 9.54927                              SSM34600
      FDWOT2 = xylint( PREOT2, NPREO, FPRO, NFSO, FSO2,
     +                       sdwofp, vfso2, FSOTAB, DWOTAB, IO1, IO2)
      EPSO = .739594 / GAMOP * XtoY( .5 * GAMP1, GAMOP/GAMM1 )
      DWOT2 = FDWOT2 * CDWOT2 * POP/(14.7 * tcsqrt(TCROT2) * EPSO)
      TRQOT2=TRBTRQ(SO2,UCOT2,TOP,POP,1./PROT2,4,CPOP,DWOT2,GAMOP)
     1       *CTQOT2
      IF ( DWOT2.LE..0) THEN
```

```
        TOT2D = TOP
       ELSE
        TOT2D = TOP - rlimit( 0.0, 300.0,
     +         TRQOT2 * SO2 / ( 9340.0 * CPOP * DWOT2 + 1.0E-6 ) )
SSM34700
       POT2D = hpfi + X10th( hpfi**2 + RHGOM*RGCOP*TOT2D*DWOT2**2, 5 )
C
C                  HOTGAS MANIFOLD COOLING
C
C   SINCE THE FUEL FLOW OUT OF THE LPFT PASSES THE OUTSIDE OF THE HOTGAS
C   MANIFOLD BEFORE ENTERING THE FUEL INJECTOR, THE HEAT EXCHANGE OF THESE
C   TWO FLOWS HAS TO BE ACCOUNTED FOR.
C           QOTMC:  HEAT FLOW FROM THE OUTLET OF OXID PREBURNER
C           TOTMC:  TEMPERATURE OF FUEL LINE AT OP SIDE
C           QFTMC:  HEAT FLOW FROM THE OUTLET OF FUEL PREBURNER
C           TFTMC:  TEMPERATURE OF FUEL LINE AT FP SIDE
C           TFT2DI: TEMPERATURE OF HOT GAS AT INJECTOR END, FP SIDE
C           TOT2DI: TEMPERATURE OF HOT GAS AT INJECTOR END, OP SIDE
C           RHOFTF: FUEL DENSITY OF FUEL LINE AT FP SIDE
C           RHOOTF: FUEL DENSITY OF FUEL LINE AT OP SIDE
C           PFIS:   FIS (FUEL INJECTOR SUPPLY) IS THE POINT BEFORE FI
C                   WHICH ALSO SUPPLY FUEL DIRECT TO COMBUSTION CHAMBER
C                   DWACV, DWBAF, AND DWPFS BESIDE DWSFS (TO FI).
C
      IF ( DWFT1.LE.0..OR.DWOT2.LE.0. ) THEN
        QOTMC=0.
      ELSE
        QOTMC = TKMCO*AHTMCO*(TOT2D - TFT1D)
      ENDIF
*
*   Not an integration step:
*
      TOTMC = TFT1D + QOTMC/(2.435*RHOOTF*VOLMCO)*DT
*
      IF(DWFT1.LE.0..OR.DWFT2.LE.0.) THEN
        QFTMC=0.                                                    SSM34800
      ELSE
        QFTMC = TKMCF*AHTMCF*(TFT2D - TFT1D)
      ENDIF
*
*  Not an integration step:
*
      TFTMC = TFT1D + QFTMC/(2.435*RHOFTF*VOLMCF)*DT
*
      TFT2DI = pruint( - QFTMC*RGCFP*TFT2D/(CPFP*PFT2D*VOLFTD),
     +                                  Tstep, 138 )
*
*  Not an integration step:
*
      TOT2DI = TOT2D - QOTMC*RGCOP*TOT2D/(CPOP*POT2D*VOLOTD)*DT
```

```
      RHOOTF = rlimit(4.0E-03, 1.0E-06, PFIS/(TOTMC*9201.6) )
      RHOFTF = rlimit(4.0E-03, 1.0E-06, PFIS/(TFTMC*9201.6) )
      PINMC=PFI+RSFS*ABS(DWSFS)*DWSFS/RHOFI+RFMCF*ABS(DWFTF)*DWFTF/
     *      RHOFTF
C
C    DWFT1 SEPERATED INTO DWFTF AND DWOTF AND THEY MERGE AFAIN AT NODE FI
C    FOR A GIVEN PRESSURE DROP DP, THE STEADY FLOW EQUATION IS:
C    DP = (R1/RHO1)*(DW1)**2
C
      DWFTF = DWFT1 / (1.0 + X10th(RFMCF*RHOOTF/(RFMCO*RHOFTF), 5 ) )
      DWOTF = DWFT1 - DWFTF
      IF(DWFT1.LE.0.) THEN                                          SSM34900
        TFIS = (TOTMC+TFTMC)*.5
        RHOFI = RHO(6)
      ELSE
        TFIS = (DWOTF*TOTMC + DWFTF*TFTMC)/DWFT1
        RHOFI = (DWFTF + DWOTF)/(DWFTF/RHOFTF + DWOTF/RHOOTF)
      ENDIF
      PFIS = PINMC - RFMCF*DWFTF*ABS(DWFTF)/RHOFTF
C
C    DEMON = TOTAL_FLOW / SQRT(RHO) IN STEADY STATE
C
      DEMON = X10th( ABS(PFIS - PFI) / RSFS, 5 ) +
     +          X10th( ABS(PFIS - PCIE), 5 ) *
     + ( 1./X10th(RACV, 5) + 1./X10th(RBAF, 5) + 1./X10th(RPFS, 5) )
      IF(DEMON.EQ.0.0) THEN
                                                                    SSM35000
        DWSFS=0.0
        DWACV=0.0
        DWBAF=0.0
        DWPFS=0.0
      ELSE
        DWSFS = DWFT1 * X10th(ABS(PFIS-PFI)/RSFS, 5) / DEMON
        DWACV = DWFT1 * X10th(ABS(PFIS-PCIE)/RACV, 5) / DEMON
        DWBAF = DWFT1 * X10th(ABS(PFIS-PCIE)/RBAF, 5) / DEMON
        DWPFS = DWFT1 * X10th(ABS(PFIS-PCIE)/RPFS, 5) / DEMON
      ENDIF
      DPHGMF=PFIS-PFT2D                                             SSM35100
      DPHGMO=PFIS-POT2D
C
C                MAIN CHAMBER FUEL (HOT GAS) INJECTOR
C
C    FUEL INJECTOR INPUTS ARE:
C            DWFT2:  OUTPUT FROM FP THROUGH FP TURBINE
C            DWOT2:  OUTPUT FROM OP THROUGH OP TURBINE
C            DWSFS:  FROM NODE FIS
C            DWFT2C: FT2 COOLING WHEN DP >=185 PSI
C            DWFBPV: BLEEDING VALVE FROM NOZZLE COOLING FLOW (NOT USED)
C    FUEL INJECTOR OUTPUTS ARE:
C            DWFI:   EXIT TO MAIN COMBUSTION CHAMBER
C
```

```
      WTFI = prlint( GAMFI *
     + ( DWFT2*TFT2DI + DWOT2*TOT2DI + DWSFS*TFIS + DWFBPV*T(4) +
     +   DWFT2C*T(3) - DWFI*TFI ), 0, 139 )
      WFIF = prlint( DWFBPV + DWSFS + DWOT2*(1.0 - ELFOPM) +
     +  DWFT2*(1.0 - ELFFPM) + DWFT2C - DWFI*(1.0 - ELFFI) ), 0, 140 )
      WFIO = prlint( DWOT2*ELFOPM + DWFT2*ELFFPM - DWFI*ELFFI, 0, 141 )
*
      WFI = WFIO + WFIF
      TFI = AMAX1(10.0, WTFI/(WFI + 1.0E-12) )                        SSM35200
      PFI = RGCFI*WTFI/VOLFI
      PFI=AMAX1(PA,PFI)
      IF(DWFT2.LE.0..AND.DWOT2.LE.0.) THEN
        GAMFI = GAM6
        RGCFI=RGC6                                                    SSM35300
      ELSE
        GAMFI=(GAMOP*DWOT2 + GAM6*DWSFS+(DWFBPV+DWFT2C)*GAM4+
     1  DWFT2*GAMFP)/(DWOT2+DWSFS+DWFT2+DWFBPV+DWFT2C+1.0E-06)
        GAMFI = rlimit(1.01, 10.0, GAMFI)
        RGCFI=(RGCOP*DWOT2+RGC6*DWSFS+RGCFP*DWFT2+RGC4*(DWFBPV+
     *      DWFT2C))/(DWOT2+DWSFS+DWFT2+DWFBPV+DWFT2C+1.0E-6)
        RGCFI = rlimit(100.0, 20000.0, RGCFI)
      ENDIF
      DWFIX = GFLOW(PFI,PCIE,TFI,RGCFI,GAMFI)*AFI
      DWFI= DWFI + 0.05*(DWFIX - DWFI)
      DWFI = recpos( DWFI )
      ELFFI = WFIO/(WFI + 1.0E-6)
C
C
C                MAIN CHAMBER IGNITOR
C
      CALL IGN(3,PMOV,POINJ,RHOMOV,P(10),RHO(10),PCIE)
C
C                MAIN CHAMBER COMBUSTOR                                SSM35400
C
C   IN THE MAIN COMBUSTION CHAMBER, FOLLOWING NOTATION ARE USED:
C   ELFC:    OXID FLOW / TOTAL FLOW IN COMBUSTION CHAMBER
C   ELFCM:   OXID / TOTAL MASS IN COMBUSTION CHAMBER
C   TCC:     COMBUSTION TEMPERATURE IN THE CHAMBER
C   GAMC:    GAMMA IN COMBUSTION CHAMBER
C   WTC:     WEIGHT TIME TEMPERATURE IN COMBUSTION CHAMBER
C   WCO:     OXID WEIGHT IN COMBUSTION CHAMBER
C   WCF:     FUEL WEIGHT IN COMBUSTION CHAMBER
C   WCN:     NITROGEN WEIGHT IN COMBUSTION CHAMBER
C   DWC:     EXIT FLOW OF COMBUSTION CHAMBER
C   DWGN2:   NITROGEN PURGE DURING SHUT DOWN (NOT USED)
C   TC:      TEMPERATURE IN COMBUSTION CHAMBER
C   WC:      TOTAL MASS IN COMBUSTION CHAMBER
C   EMRC:    O/F RATIO
C   PCIE:    CHAMBER PRESSURE AT INJECTOR END
C   PCNS:    CHAMBER PRESSURE AT NOZZLE END
```

```
C    EMWC:    EQUIVALENT MOLECULAR WIGHT IN COMBUSTION CHAMBER
C    CPC:     SPECIFIC HEAT RATIO IN COMBUSTION CHAMBER
C    EMUC:    VISCOSITY WITHIN COMBUSTION CHAMBER
C    EKC:     THERMAL CONDUCTIVITY WITHIN COMBUSTION CHAMBER
C    PRDC:    PRANDTL NUMBER IN COMBUSTION CHAMBER
C    REYC:    REYNOLDS NUMBER IN COMBUSTION CHAMBER
C
      dsum = DWOI + DWFI + DWACV + DWFIG(3) + DWBAF + DWPFS
      ELFC = (DWOI + DWFI*ELFFI) / ( dsum + 1.0E-6 )
      ELFCM = WCO/(WC+1.0E-6)
      TCC = FGEN(20, 84, ELFC) + T(9)
      GAMC = FGEN(13, 85, ELFCM)
      WTC1 = prlint( GAMC*( dsum * TCC - DWC * TC), 0, 142 )
      WCO = prlint( DWOI + DWFI*ELFFI - DWC*ELFCM, 0, 143 )          SSM35500
      WCF = prlint( DWACV + DWBAF + DWPFS + (1.0 - ELFFI) * DWFI
     +                        - (1.0 - ELFCM)*DWC + DWFIG(3), 0, 144 )
      IF ( PA/PCNS .LE.
     +    XtoNY( 2./(GAMC + 1.0), GAMC/(GAMC - 1.0) ) ) THEN
        DWC = PCNS * ACN * 32.2 / CSTAR(1, EMRC, PCNS)
      ELSE
        DWC = GFLOW(PCNS,PA,TC,RGCC,GAMC) * ACN
      ENDIF
      WCN = prlint( DWGN2*RGN2 - DWC*ENFC, 0, 145 )
      IF( WCN.GT.0.0)
         ENFC = WCN/(WC + 1.0E-06)
      DWC = recpos( DWC )
*
*  Unit 6 formatted output was removed
*
      WC = WCO + WCF + WCN
      TC = AMAX1(10.0, WTC/( WC + 1.0E-12) )
*                                  Added for 'SIGMAi ='s
      vTC = 1.0 / ( 2.0 * TC )

      EMRC = WCO / ( WCF + 1.0E-12 )
      PCIE = RGCC * WTC * vVOLC
      PCIE = AMAX1( PCIE, PA )                                       SSM35800
      PCNS = PCIE * EFFCM
*                     Order was changed to calculate EMWC and CPC once,
                          and to test ENFC once.
      IF ( ENFC.GT.0.)
         EMWC = ( (WCO + WCF) * FGEN(12, 86, EMRC) + WCN * 28.0) / WC
         CPC = ( (WCO + WCF) * FGEN(21, 87, ELFCM) + WCN * 0.274 ) / WC
      ELSE
         EMWC = FGEN(12, 86, EMRC)
         CPC = FGEN(21, 87, ELFCM)
      END IF
      RGCC = 18540.0 / EMWC
      EMUC = FGEN(25, 88, ELFCM)
      EKC = FGEN(23, 89, ELFCM)
```

```
      GAMAC = (GAMC - 1.0) * 0.5
      PRDC = (CPC * EMUC) / EKC                                        SSM35900
      A4C = ABS(DWC) / ACN
      REYC = CPC * X10th(EMUC, 2) / X10th(PRDC, 6)
C
C                FIXED NOZZLE HOT GAS SIDE HEAT TRANSFER
C
C   THE HEAT TRANSFER EQUATIONS USED FOR THE FOLLOWING CODES ARE DESCRIBED
C   IN THE DOCUMENT 29-30 WITH A SLIGHT MODIFICATION.
C   SIMILAR EXPLANATIONS CAN BE FOUND IN HILL'S BOOK OF
C   "MECHANIC AND THERMODYNAMICS OF PROPULSION".
C   !!!!!   HOWEVER, THERE IS A MAJOR PROBLEM   !!!!!
C   THE NODE 12 (DOWN-COMER) HEAT TRANSFER IS NOT CALCULATED HERE.
C   AND, THE FOLLOWING CODE INCLUDES NODE 7 HEAT TRANSFER WHICH DOES NOT
C   CONTACT WITH THE NOZZLE.
C   IN THE FOLLOWING:
C           EMC#:   MACH NUMBER AT SPECIFIC POINT
C           DIAT:   THROAT DIAMETER
C           QIN1(): HEAT FLOW FROM HOT GAS TO WALL
C
      EMC4 = SC4 * GAMC + BC4
      EMFTR4 = 1.0 + GAMAC * EMC4**2
      SIGMA4 = 1.0 / ( XtoY( TW1(4) * vTC * EMFTR4 + 0.5, A3 ) *
     +                 XtoY( EMFTR4, AA2 ) )
*
*     HTCC=0.026/DIAT**0.2*REYC*A4C**0.8*A5  x ** 2  is much cheaper
*                                            than x ** 0.2
      HTCC = 0.026 / DIAT**2 * REYC * X10th(A4C, 8) * A5
      HTCC4 = HTCC * X10th(AR4, 9) * SIGMA4
      QIN1(4) = HTCC4 * AHTC4 * ( TC - TW1(4) )                        SSM36000
      IF( PA .GT. 1.0 ) THEN
*
*        IF(PCIE.LE.700.)QIN1(4)=QIN1(4)*AMIN1(5.6,9.68-.0236*PCIE+
*       1  1.6E-05*PCIE**2)
*        IF(PCIE.LE.350.)QIN1(4)=QIN1(4)*AMIN1(2.,
*       1 AMAX1(1.,1.+(350.-PCIE)/100.))   Tests PCIE in the wrong
*                                          order and wastes results.
         IF( PCIE.LE.350. ) THEN
     +      QIN1(4) = QIN1(4) * rlimit( 1., 2., 1. + (350.-PCIE)*.01 )
         ELSE IF( PCIE.LE.700.) THEN
            QIN1(4) = QIN1(4) *
     +           AMIN1( 5.6, 9.68 - PCIE * ( .0236 - 1.6E-05 * PCIE ) )
         END IF
      END IF
C
C                MAIN CHAMBER HOT GAS SIDE HEAT TRANSFER
*
  130 EMFTR5 = 1.0 + GAMAC * EMC5**2
      EMFTR6 = 1.0 + GAMAC * EMC6**2
      EMFTR7 = 1.0 + GAMAC * EMC7**2                                   SSM36100
```

```
      SIGMA5 = 1.0 / ( XtoY( TW1(5) * vTC * EMFTR5 + 0.5, A3 ) *
     +                 XtoY( EMFTR5, AA2 )
      SIGMA6 = 1.0 / ( XtoY( TW1(6) * vTC * EMFTR6 + 0.5, A3 ) *
     +                 XtoY( EMFTR6, AA2 )
      SIGMA7 = 1.0 / ( XtoY( TW1(7) * vTC * EMFTR7 + 0.5, A3 ) *
     +                 XtoY( EMFTR7, AA2 )
      HTCC5 = HTCC * X10th(AR5, 9) * SIGMA5
      HTCC6 = HTCC * X10th(AR6, 9) * SIGMA6
      HTCC7 = HTCC * X10th(AR7, 9) * SIGMA7
      QIN1(5) = HTCC5 * AHTC5 * ( TC - TW1(5) )
      QIN1(6) = HTCC6 * AHTC6 * ( TC - TW1(6) )
C     QIN1(7) = HTCC7 * AHTC7 * ( TC - TW1(7) )
      RETURN                                                        SSM36200
      END
```

'ignt.for':

```
      SUBROUTINE IGN0(I, POU, POD, RHOO, PFU, RHOF, PC)
*****************************************************************
C                                                                  SSM37700
C  PURPOSE:  SIMULATE THE IGNITION SYSTEM TRANSIENTS
C
C******ARGUMENTS******
C  I  = CHAMBER INDEX
C   1 = FPB
C   2 = OPB                                          C   3 = MCC
C
C  A1 = OXIDIZER VALVE PRESSURE,PSI
C  A2 = OXIDIZER INJECTOR MANIFOLD PRESSURE,PSI                    SSM37800
C  A3 = OXIDIZER VALVE DENSITY,LB/IN3
C  A4 = FUEL SUPPLY PRESSURE,PSI
C  A5 = FUEL SUPPLY DENSITY, LB/IN3
C  A6 = IGNITOR CHAMBER DISCHARGE PRESSURE, PSI
C  A7 = NOT USED
C  N  = INITIALIZATION FLAG  inactivated
C
C   IGNITOR ITSELF IS A SMALL COMBUSTION CHAMBER INSIDE (?) THE
C   COMBUSTION CHAMBER.
C   FUEL FEED LINES FOR IGNITORS ARE JUST LIKE REGULAR PIPE LINES
C   WITH SMALLER SIZE.  THE CALCULATION OF FUEL FEED LINE IS SIMPLE
C   FOR A GIVEN CONDITION.
C   OXID FEED LINES FOR IGNITORS ARE THE BYPASS LINES AFTER THE CONTROL
C   OXID VALVES.  THE FOLLOWING CALCULATION IS MAINLY DEALING WITH
C   THE FLOW AND PRESSURE BALANCE OF THE IGNITOR LINES AND MAIN OXID LINES.
C   THE SCHEMATIC OF THE OXID LINES FOR A COMBUSTOR IS:
C

               diagram omitted
C
C******COMMON USAGE******
C INPUT:
C  VARIABLE                                          SOURCE         SSM37900
C   T(3),T(9)                                        FUELF
C
C OUTPUT:
C  VARIABLE                                          DESTINATION
C   DWFIG,DWOIG,DW1,DW2,P4,PCIG                      HOTGAS
C   DWFIG                                            FUELF
C
C
C               SUBSCRIPT (1)=FUEL PB,(2)=OXID PB,(3)=MAIN CHAMBER SSM38000
C
*****************************************************************
C               Eliminated superfluous arrays:
```

```
*     DIMENSION POU(3), POD(3), RHOO(3), PC(3), RHOF(3)
*
      REAL  AN(3), TAUC(3), ZFIG(3),
     +      RFIG(3), P4(3), EL1(3), EL2(3), EL3(3),
     +      R1I(3), R2I(3), R3I(3)                                    SSM38100
C
      INCLUDE 'blank.com'
      INCLUDE 'out.com'
      INCLUDE 'igni.com'
      INCLUDE 'units.com'
*
      INTEGER Tstep
      PARAMETER( Tstep = 0 )
C
      IF ( I .EQ.1) THEN
        DO 11 J=1,3
          READ(run,12) R1I(J), R2I(J), R3I(J), AN(J), TAUC(J),
     +                 ZFIG(J), RFIG(J), EL1(J), EL2(J), EL3(J)      SSM38400
          R1(J) = R1I(J)*0.0412
          R2(J) = R2I(J)*0.0412
          R3(J) = R3I(J)*0.0412
   11   CONTINUE
      ENDIF
C
      P4(I)=PA
      PCIG(I)=PA
      DWOIG(I)=1.0E-30
      DW3(I)=DWOIG(I)
      DW2(I)=1.0E-30
      DW1(I)=2.0E-30
      DWFIG(I)=1.0E-25
      DWNIG(I)=1.0E-25
      ELFIG(I)=0.0                                                   SSM38700
*
      CALL unint0( DWOIG(I), 213 + I )
      CALL unint0( DW1(I), 216 + I )
      CALL unint0( DW1(I), 219 + I )
      CALL unint0( DWFIG(I), 222 + I )
      CALL unint0( PCIG(I), 225 + I )
*
      GAM=1.4
      R=9270.0
      TC=460.0
      XINT=1.0
      RETURN
*
      ENTRY IGN(I, POU, POD, RHOO, PFU, RHOF, PC)
*****************************************************************
      POU(I)=A1
      POD(I)=A2
```

```
      RHOO(I)=A3
      PFU(I)=A4
      RHOF(I)=A5
      PC(I)=A6                                                    SSM38300
      TF(1)=T(9)
      TF(2)=T(9)
      TF(3)=T(3)
C
C     THIS IS TO CALCULATE THE PRESSURE CHANGE OF THE SEPARATION NODE (P4)
C     AS THE RESULT OF THE CHANGES OF NEIGHBOR NODES POD, POU AND PCIG WITHIN
C     THE TIME FRAME DT.  THIS IS A TRANSIENT CALCULATION, FOR A STEADY STATE
C     CONDITION (DT = LARGE) AND THE LAST TERM DROPS OUT.
C     IN THE FOLLOWING:
C             E = DELTA_POU / DELTA_DW1
C             F = DELTA_PCIG / DELTA_DW2
C             G = DELTA_POD / DELTA_DW3
C
*       E=2.0*R1(I)/RHOO(I)*ABS(DW1(I))+EL1(I)/DT
*       F=2.0*R2(I)/RHOO(I)*ABS(DW2(I))+EL2(I)/DT
*       G=2.0*R3(I)/RHOO(I)*ABS(DWOIG(I))+EL3(I)/DT                 SSM38800
*       A=(-P4(I)+POD(I)+R2(I)/RHOO(I)*ABS(DW2(I))*DW2(I))/F
*       B=(-P4(I)+PCIG(I)+R3(I)/RHOO(I)*ABS(DWOIG(I))*DWOIG(I))/G
*       C=(POU(I)-P4(I)-R1(I)/RHOO(I)*ABS(DW1(I))*DW1(I))/E
*       D=1.0/E+1.0/F+1.0/G
*       DP4=(A+B+C)/D       this is d(P4)/dt * DT
*
       vRHOO = 1.0 / RHOO
       rvt = 2.0 * vRHOO * DT
       abdw = ABS( DW1(I) )
       vedt = 1.0 / ( rvt * R1(I) * abdw + ELI(I) )
       cvdt = ( POU - P4(I) - R1(I)*vRHOO * abdw * DW1(I) ) * vedt
       abdw = ABS( DW2(I) )
       vfdt = 1.0 / ( rvt * R2(I) * abdw + EL2(I) )
       avdt =( -P4(I) + POD + R2(I)*vRHOO * abdw * DW2(I) ) * vfdt
       abdw = ABS( DWOIG(I) )
       vgdt = 1.0 / ( rvt * R3(I) * abdw  + EL3(I) )                SSM38800
       bvdt = ( -P4(I) + PCIG(I) + R3(I)*vRHOO * abdw * DWOIG(I) ) * gdt
       dvdt = vedt + vfdt + vgdt
       p4rate = ( avdt + bvdt + cvdt ) / ( dvdt * DT )
*
       DWOIG(I) = prflow( DWOIG(I), EL3(I), -R3(I)*vRHOO,
     +                                      P4(I) - PCIG(I), 213 + I )
       DW1(I) = prflow( DW1(I), EL1(I), -R1(I)*vRHOO,
     +                                      POU - P4(I), 216 + I )
       DW2(I) = DW1(I) - DWOIG(I)                                   SSM38900
*
       P4(I) = pruint( p4rate, Tstep, 219 + I )
*
       RF = RFIG(I) / RHOF
       DWFIG(I) = prflow( DWFIG(I), ZFIG(I), RF, PFU -PCIG(I), 222+I )
```

```
*
*     ELFIG(I)=AMAX1(0.0,AMIN1(1.0,DWOIG(I)/(DWOIG(I)+DWFIG(I)+1.E-25)))
*
      dwigs = DWOIG(I) + DWFIG(I)
      IF ( ABS( dwigs ) .LT. 1.E-25 ) THEN
        ELFIG(I) = 1.0
      ELSE
        ELFIG(I) = AMAX1( 0.0, AMIN1(1.0, DWOIG(I)/dwigs) )
      ENDIF
      GAM = fgen(13, 120 + I, ELFIG(I) )
*
*   Example of limiting table input to avoid extending table,
*   instead of looking up  ABS(DWOIG(I))/(ABS(DWFIG(I))+1.E-6))
*
      IF ( ABS( DWFIG(I) .LT. 1.0E-10 ) THEN
        EMRIG(I) = 1.0
        R = 1000.
      ELSE
        EMRIG(I) = DWOIG(I) / DWFIG(I)
        dwigs = ABS( EMRIG(I) )
        IF ( dwigs .GT. 9999. ) THEN
          R = 600.
        ELSE
          R = 18540.0 / fgen(12, 123 + I, dwigs )
        ENDIF
      ENDIF
*
      TCIG(I) = AMAX1( 40.0, fgen(20, 126 + I, ELFIG(I))+TF(I) )
      DWNIG(I) = DWNIG(I) + 0.3 *
     + ( GFLOW(PCIG(I), PC, TCIG(I), R, GAM) * AN(I) - DWNIG(I) )    SSM39000
*
      PCIG(I) = pruint( (DWFIG(I) + DWOIG(I) - DWNIG(I))/TAUC(I),
     +                                                  Tstep, 225 + I )
*
      EMRIG(I) = DWOIG(I) / (DWFIG(I)+1.0E-10)     replaced by above
*
      END
```

'gflow.for':

```
      FUNCTION GFLOW(PU,PD,TU,R,G)
************************************************************
C
C  PURPOSE:  COMPUTE ISENTROPIC IDEAL GAS FLOW FOR CHOKED
C            AND UNCHOKED ORIFICES
C                                                                SSM14900
C******ARGUMENTS******
C  INPUT:
C   PU    = UPSTREAM PRESSURE, PSI
C   PD    = DOWNSTREAM PRESSURE, PSI
C   TU    = UPSTREAM TEMPERATURE, DEG R
C   R     = GAS CONSTANT, IN-LBF/(LBM-DEG RF)
C   G     = GAMMA, RATIO OF SPECIFIC HEAT
C
C  OUTPUT:
C   GFLOW = MASS FLOW RATE/AREA LB/IN2-SEC                       SSM15000
C
************************************************************
*
      slow1(gee) = 2.0 * xtoy( gee + 1.0, - gee / (gee - 1.0) )
*
      slow2(pr,gee) = xtoy( pr, 2.0/gee ) - xtoy( pr, (gee + 1.0)/g )
*
      test =  772.8 * G / (R * TU * (G-1.0) )
      IF ( test .LT. 0.0 ) THEN
        GFLOW = 0.0
      ELSE
        IF ( PU .LT. PD ) THEN
          P1 = - PD
          ratio = PU / PD
        ELSEIF ( PU .eq. PD ) THEN
          P1 = 0.0
          ratio = 1.0
        ELSE
          P1 = PU
          ratio = PD / PU
        ENDIF
        PR =  AMAX1( slow1(G), ratio )
        if ( PR .LT. 0.0 ) THEN
          GFLOW = 0.0
        ELSE
          test2 = slow2(PR)
          IF ( test2 .LT. 0.0 ) THEN
            GFLOW = 0.0
          ELSE
            GFLOW = P1 * X10th( test * test2, 5 )
          ENDIF
```

```
        ENDIF
      ENDIF
      END

      SUBROUTINE CNTRL0
*****************************************************************
C   THIS IS THE PROGRAM SIMULATE THE DIGITAL CONTROL FUNCTIONS OF SSME.
C   IT IS APPARENT THAT THE DATA WERE READ AND WRITTEN EVERY 0.02 SECOND.
C   D/A DELAY AND INSTRUMENT DELAYS ARE ACCOUNTED FOR BY TIME CONSTANTS
C   TIMEVC, TIMECP, TIMEPR, AND OTHERS.
C   THE PURPOSE OF THIS SIMULATION IS TO READ MEASUREMENT OUTPUT AND
C   CALCULATE THE NECESSARY OUTPUT AS DEFINED BY THE CONTROL SCHEME.
C
C   ONLY INPUTS THAT CAN BE USED TO CALCULATE CONTROL SIGNALS ARE
C   THOSE ACTUALLY MEASURED AND USED.
C*****************************************************************
C
C******ARGUMENT******
C  ICCNTL = INITIALIZATION FLAG  eliminated in favor of initialization
C                                routine CNTRL0 and simulation loop entry
C******COMMON USAGE******
C  INPUT:
C   VARIABLE                                  SOURCE
C    DW(1),DW(2),P(3),T(2),RHO(2)             FUELF
C    PCIE,DWOPO,DWFPO                         HOTGAS                    SSM01500
C    DWMOV                                    OXIDF
C    XOPOV                                    VALDYM
C
C  OUTPUT:
C   VARIABLE                                  DESTINATION
C    XCFPOV,XCOPOV,XCMOV,XCMPV,XCCCV          VALDYM
C
C  SUBROUTINES CALLED:  EMCO
C
*****************************************************************
C
      LOGICAL RESET, first
      DIMENSION D(5),PIN(5),DEN(5),PD(5),XP(5),I1(5)
      DIMENSION AT(5),BT(5)
      DIMENSION STHETA(5)
C
      INCLUDE 'blank.com'
      INCLUDE 'out.com'
      INCLUDE 'contrl.com'
      INCLUDE 'hgas.com'
      INCLUDE 'oxid.com'
      INCLUDE 'balc.com'
      INCLUDE 'valves.com'

      COMMON/PURGE/DWGN2,TCUTPR,DWGN2F,DWGN2O                          SSM02000
```

```
C
*  Computation constants
*
      PARAMETER ( PI = 3.141596, tpcnst = 8./( 386.4 * PI ** 2 ),
     +            piov2 = PI * .5 , twopi = PI * 2. )
      PARAMETER ( v1728 = 1./1728., v41p12 = 1./41.42,
     +            v14p06 = 1./14.06, v29p95 = 100. / 2995. )
      PARAMETER ( v29p95 = 100. / 2995. )
*
      DATA XOPOMS/ 100.0   /
      DATA I1/11,14,16,32,36/
C
C INITIALIZE LOCAL VARIABLES AND ARRAYS NOT ASSIGNED VALUES
C THIS IS NECESSARY FOR SUCCESSFUL EXECUTION ON THE IBM              SSM02500
* Unnecessary initialization was eliminated. It created unused
* variables, due to confusion between 'O' and '0' characters.
*
*     IF(FLAG.EQ.15.) GO TO 9999
      C2=0.0
      EEMR=0.0                                                      SSM03000

                        initializations omitted
      YCOPVL=0.0
      DO 9998 I=1,5
       D(I)=0.0
       DEN(I)=0.0
       PD(I)=0.0                                                    SSM04300
       PIN(I)=0.0
       STHETA(I)=0.0
       XP(I)=0.0
 9998 CONTINUE
      IPBCO=0
C
C*****************************************************************************
C   THIS IS THE INPUT READING FOR MOST CONTROL PARAMETERS.
C   THESE PARAMETERS DETERMINE THE SCHEDULE, LEVEL VS. TIME, OF:
C           1) THRUST REQUESTS,
C           2) MIX-RATION REQUESTS,
C           3) OPEN-LOOP OPOV SETTINGS,
C           4) CLOSED-LOOP GAINS, K, KP AND KI, OF THRUST CONTROL (OPOV),
C           5) OPEN-LOOP FPOV SETTINGS,
C           6) CLOSED-LOOP GAINS, K, KP AND KI, OF MIX-RATION CONTROL (FPOV)
C           7) MAIN OXID VALVE (MOV),
C           8) MAIN FUEL VALVE (MFV),
C           9) COOLANT CONTROL BYPASS VALVE (CCV),
C*****************************************************************************
C
   30 FORMAT(1X,6F12.0)                                             SSM04400
      READ(run, 30)TIDMO,TOPOV,TMOVST,TMOVRA,TBPV,XMOVST,DTF,FRZ
     1, DBEF,DBMR,TPI,DTPS,DTOPFS,DTVFS,DTMCR,TCUT,TCLF,F1,DF1,F2,FC,TVC
```

```
     2, FCO,TUT,DUM,OPOVCP,OPCR1,OPCR2,OPOVOP,RO1,RO2,RO3,OSTEP,CFG1
     3, CFG2,FPOVOP,RF1,RF2,RF3,DUM1,FSTEP,FPOVCP,FPCR1,FPCR2,OMVCR
     4, TSMOV,DMOVO,XMOVM,DMOVUT,TMS,DFMOV,FMOVC,CCV1,DCCV1,TCCV1,DCCVC
     5, DCCV2,CCV2,CCVM,DFCCV,FCCVC,TFRMFV,DMFV1,XMFVM,FMVM,CMFV,DFMFV
     6, P1MFV,DP1MFV,DTMFVR,DP2MFV,DTPURG,PCFACT,TMRC,OPCR0,OPOVCM
     7, OMVCRO,DTCOMF,DTCOCV,FA1,FB1,FC1,FD1,DCFG,CFGC,CCVC,DTPNC,DFRC
     8, CCV3,DCCV3,DCCV0,XCCCV0,DXCFPO,FPCR0,FPOVCM,OPOVPB,CFGMS,T1FPV
     9, T2FPV,FPVDX,T1OPV,T2OPV,OPVDX                                  SSM04500
      TCUTPR=TCUT+DTPURG                                               SSM05000
C
*   load function interpolation tables
*
      CALL fgset( 6 )
      CALL fgset( 17 )
      CALL fgset( 18 )
      CALL fgset( 19 )
      CALL fgset( 31 )
      CALL fgset( 33 )
      CALL fgset( 55 )
      YCMFV = FGEN(55, 90, 0. )
      CALL fgset( 56 )
      CALL fgset( 57 )
*  Added for TLIMIT
      CALL fgset( 84)
      XOPOV=0.0                                                        SSM05100
      XFPOV=0.0
      XMOV=0.0
      ROPOV=1.0E+12
      RFPOV=1.0E+12
      RMOV=1.0E+12
      EMRGC=0.0
      EMRFPO=0.0
      EMROPO=0.0
      DXFPOV=0.0
      DXOPOV=0.0                                                       SSM05200
      PCOPO=5.0
      PCFPO=0.0
      FR=60000.0
      XCOPOV=0.0
      XCFPOV=0.0
      XCMFV=0.0
      XCMOV = 0.0
      XCMOVC=XCMOV
      XCCCV=100.                                                       SSM05300
      YCFPOV=0.0
      YCOPOV=0.0
      YCMOV=0.0
      YCCCV=100.0
      TFT2D=TFP
      TOT2D=TOP
```

```
      EPCGC=0.0
      EPC=0.0
      TCUTFP=1000.0                                                  SSM05400
      TCUTOP=1000.0
      TCUTCV=1000.0
      TCUTFV=1000.0
      TCUTOV=1000.0
      TCUTFV=1000.0
      TMPL=1000.0
      DPRI=DPR
      DPLI=DPL
*     DTI=DT
      IFIND=0                                                        SSM05500
      IOIND=0
      PIPF=-1.
      PIPO=-1.
      NCF=0
      NCO=0
      ICUT=1                                                         SSM05600
      RESET = .FALSE.
      NMALF = 0
      STEPO    = 0.0
*
*  Resetting of TIMExx values in EMCO(1), now EMCO0, was eliminated.
*  Initializing timing values to STIME works for a restart as well.
*
      TIMECP = STIME
      TIMEPR = STIME
      TIMETR = STIME
      TIMEVC = STIME
      TSMFV=AMIN1(TSMFV+TPA,TSMFV+TCUT)
      CALL EMCO0
      inkdt = 300000. * DT
      df1dt = DF1 * DTMC
      dfrcdt = DFRC * DTMC
      frbias = 1. + .01 * PFRNZ
      osdt = DTMC * OSLAM
      op0dt = DTMC * OPCR0
      op1dt = DTMC * OPCR1
      op2dt = DTMC * OPCR2
      fp0dt = DTMC * FPCR0
      fp1dt = DTMC * FPCR1
      fp2dt = DTMC * FPCR2
      omvdt = OMVCR * DTMC
      omvodt = OMVCRO * DTMC
      omvfdt = OMVCRF * DTMC
      ydt1 = .32 - DTMC
      ydt2 = DTMC / (.32 + DTMC )
      dmovdt = DMOVO * DTMC
      dcc3dt = DCCV3 * DTMC
```

```
      dcc2dt = DCCV2 * DTMC
      dccvdt = DCCVC * DTMC
      dcc1dt = DCCV1 * DTMC
      dcc0dt = DCCV0 * DTMC
      dp1dt = DP1MFV * DTMC
      dp2dt = DP2MFV * DTMC
      IF ( ipflag .EQ. 1 ) THEN
        fzopvb = FZOPV
        fzycf = FZFPV
        fzycm = FZMOV
        fzycc = FZCCV
        fzycmf = FZMFV
      ELSE
        fzopvb = FZOPV * ( 1. + .01 * POPVNZ )
        IF ( fzopvb .GE. 100.0 ) fzopvb = FZOPV
        fzycf = FZFPV * ( 1. + .01 * PFPVNZ )
        IF ( fzycf .GE. 100.0 ) fzycf = FZFPV
        fzycm = FZMOV * ( 1. + .01 * PMOVNZ )
        IF ( fzcm .GE. 100.0 ) fzycm = FZMOV
        fzycc = FZCCV * ( 1. + .01 * PCCVNZ )
        IF ( fzycc .GE. 100.0 ) fzycc = FZCCV
        fzycmf = FZMFV * ( 1. + .01 * PMFVNZ )
        IF ( fzycmf .GE. 100.0 ) fzycmf = FZMFV
      ENDIF
      dtm100 = 100. * DTMC
      dtm200 = 200. * DTMC
      CALL unint0( FRADS, 146 )
      CALL unint0( ORADS, 147 )
*
      ENTRY CNTROL
C*************************************************************************
C   CONTROLLER SIMULATION BEGINS HERE
C           PIN( )   : INPUT PRESSURE TO THE VALVE
C           D( )     : FLOW RATE TO THE VALVE
C           DEN( )   : DENSITY OF FLOW MEDIA
C           PD( )    : DOWN STREAM PRESSURE
C           VPD( )   : VALVE PORT DIAMETER
C           XP( )    : % OPENNING OF VALVE ANGLE
C   READER SHOULD REFER TO THE SSME DOCUMENT FOR FURTHER DETAILS ON
C           VALVE CHARACTERISTICS.
C*************************************************************************
C
C*****      NMALF IS THE FLAG OF PURGE REQUEST WHEN MALFUNCTIONS DETECTED  *
C
      IF ( NMALF .GT. 0 ) THEN
        NMALF = 1                                                  SSM08600
        TCUTPR = AMIN1( TCUTPR, TPA + DTPURG )
*       DT=DTI                                  change in DT disallowed
        IF ( STIME .LE. TPA + DTPNC ) RETURN
        CALL EMCO(2)
```

```
          EMRE = ( DWMOV + DWOPO + DWFPO ) / ( DW(1) + 1.0E-10 )
          RETURN
        ENDIF
C
        PIN(1)=PFPOV
        PIN(2)=POPOV                                                    SSM05700
        PIN(3)=PMOV
        PIN(4)=P(3)
        PIN(5)=P(7)
        D(1)=DWFPO
        D(2)=DWOPO
        D(3)=DWMOV
        D(4)=DW(3)
        D(5)=DW(7)
        DEN(1)=RHOOP3
        DEN(2)=RHOOP3                                                   SSM05800
        DEN(3)=RHOOP2
        DEN(4)=RHO(3)
        DEN(5)=RHO(7)
        PD(1)=PFPOI
        PD(2)=POPOI
        PD(3)=POINJ
        PD(4)=P(10)
        PD(5)=P(8)
        XP(1)=XFPOV
        XP(2)=XOPOV                                                     SSM05900
        XP(3)=XMOV
        XP(4)=XMFV
        XP(5)=XCCV
*       STIME = TIME
C
C*******  CALCULATE HYDRAULIC AND SLIDING FRICTIONAL TORQUE TP( )  *********
C    ALTHOUGH THE VALUE TP( ) SHOULD BE USED IN CALCULATING THE WIND-UPS
C    AND STICTIONS, THIS VALUE IS REALLY NEVER USED IN THIS SIMULATION.
C    THE WIND-UPS AND STICTIONS ARE SET TO BE CONSTANTS IN THE VALVE
C    DYNAMIC SIMULATION SUBROUTINE VALDYN( ).
C
*   The calculation of TP is left in, on the theory that the values can
*   be monitored or later used in the simulation.
*
        DO 250 I=1,5
          IF( DTHETA(I) .EQ. 0.D0 ) THEN
*           STHETA(I) = DTHETA(I) replaced to avoid unnecessary conversion
            STHETA(I) = 0.
            THK = 0.0
            TP(I)=0.
          ELSE
            THK = FGEN(I1(I), 90 + I, XP(I)*.01 )                       SSM06000
*           TP(I) =
*     *      SIGN(AT(I)+(BT(I)+0.8*ESS(I))*(PIN(I)-8./386.4/DEN(I)*(D(I)/
```

```
*     *       PI/VPD(I)**2)**2)+0.8*DSS(I)+CPS(I)+CS(I)*(PIN(I)-PD(I)),
*     *       STHETA(I))+THK*D(I)**2/DEN(I)/VPD(I)/1728.
            TP(I) = SIGN( AT(I) + ( BT(I) + 0.8*ESS(I) ) *
     +          ( PIN(I) - tpcnst/DEN(I)*( D(I) / VPD(I)**2 )**2 ) +
     +                  0.8*DSS(I) + CPS(I) + CS(I)*( PIN(I) - PD(I) ),
     +                    STHETA(I) ) + THK*D(I)**2/(DEN(I)*VPD(I)*1728.)
          ENDIF
  250 CONTINUE
C
C*********************************************************************
C   I HAVE ABSOLUTELY NO IDEA OF WHAT THIS SECTION OF CODES IS DOING.
C       PFFM AND POFM ARE PRESSURES OF MFV AND MOV (?)
C       DENF AND DENO ARE CALCULATED DENSITIES
C       FRADS AND ORADS ARE ACCUMULATED VOLUMNS
C       SIN() OF FRADS AND ORADS MEANS ???????????
C   SINCE PIPF IS THE PREVIOUS VALUE OF SFRADS (DT=0.0002), THE FOLLOWING
C   CONDITION SEEMS TO INDICATE THE CHANGE OF SIGN FROM - TO +.
C       IF(IFIND.NE.2.OR. SFRADS.LT.0.) GO TO xx
C       IF(IFIND.NE.2.OR. PIPF.GT.0.) GO TO xx
C*********************************************************************
C   HOWEVER, NONE OF THE VARIABLES GENERATED IN THIS SECTION ARE USED
C   ANYWHERE IN THE SIMULATION OF THE SSME.
C*********************************************************************
C
*  Unfortunately for simulation speedup,  the code of this section does
*  affect the simulation. The KOUNTF and KOUNTO set NCF and NCO,
*  affecting EMRF.
*
      PFFM=P(3)-DW(2)*ABS(DW(2))/RHO(2)*2.87137E-06
*
*     DENF=((-1.4013E-03+3.09257E-06*PFFM)*T(2)**2+(6.5220E-02
*    *     -2.14666E-04*PFFM)*T(2)+3.8956+4.27390E-03*PFFM)/1728. replaced b
*
      DENF = v1728*( 3.8956 + 4.27390E-03*PFFM + T(2) *
     +           ( 6.5220E-02 - 2.14666E-04*PFFM +
     +                (-1.4013E-03 + 3.09257E-06*PFFM)*T(2) ) )
*
      FRADS = pruint( DW(2)*v41p12/DENF, 0, 146 )
      POFM = POD2 -
     + (DWOP2 - DWOT1)*ABS(DWOP2 - DWOT1)/RHOOP2*1.543474E-05        SSM06100
      DENO = ( (-0.16603+5.6683E-06*POFM)*TOD2 + 98.5 -1.8237E-04*POFM )
     +           * v1728
      ORADS = pruint( (DWMOV+DWOPO+DWFPO)/DENO*v14p06, 0, 147 )
      IF(FRADS.GT.1.0E+20) THEN
        FRADS = 0.
        unint0( 0., 146 )
      ENDIF
      IF(ORADS.GT.1.0E+20) THEN
        ORADS = 0.
        unint0( 0., 147 )
```

```
      ENDIF
*
*     SFRADS=SIN(FRADS)     sin and arcsin series had been used, where
*     SORADS=SIN(ORADS)     only a mod function is required.
      sfrads = MOD( FRADS, twopi )
      sorads = MOD( ORADS, twopi )
*
*   The selection sequence based on IFIND and IOIND was rewritten
*   for clarity and to eliminate repeated tests.
*
      GO TO ( 40, 50 ) IFIND
      GO TO 60
*
*        case IFIND is 1
*
   40 IF ( SFRADS .LT. PI .AND. PIFP .LE. 0. ) THEN
*
*   Statements of the form kountx = kountx + realexpression
*   force the unnecessary conversion of kountx to REAL and
*   reconversion of the sum to integer. They are replaced by
*   a sequence requiring the minimum single conversion.
*
*       KOUNTF=KOUNTF+300000.*ASIN(SFRADS)*DENF*41.12/DW(2) replaced by
        ink = 12.336E+6 * MOD( FRADS, piov2 ) * DENF / DW(2)
        KOUNTF = KOUNTF + ink
*
        IFIND=2
      ENDIF
      GO TO 60
*
*        case IFIND is 2
*
   50 IF ( SFRADS .GE. 0. .AND. PIFP .LE. 0. ) THEN
        ink = 12.336E+6 * ABS( MOD( PIPF, piov2) ) * DENF / DW(2)
        KOUNTF = KOUNTF + ink
        IFIND=3                                                    SSM06200
      ENDIF
*
*  The simulation loop invariant inkdt = 300000. * DT is computed in CNTRL0
*
      KOUNTF = KOUNTF + inkdt
*
   60 dwsum = DWMOV + DWOPO + DWFPO
      GO TO( 70, 80 ) IOIND
      GO TO 95
*
*        case IOIND is 1
*
   70 IF ( SORADS .LT. PI .AND. PIFO .LE. 0. ) THEN
        ink = 12.336E+6 * MOD( ORADS, piov2 ) * DENO /
```

```
     +          ( dwsum + 1.0E-06 )
           KOUNTO = KOUNTO + ink
           IOIND=2
         ENDIF
         GO TO 95
*
*          case IOIND is 2
*
   80  IF ( SORADS .LT. PI .AND. PIFO .LE. 0. ) THEN
           ink = 12.336E+6 * ABS( MOD( PIPO, piov2 ) )* DENO /
     +           ( dwsum + 1.0E-06 )
           KOUNTO = KOUNTO + ink
           IOIND = 3
         ENDIF
         KOUNTF = KOUNTF + inkdt
*
   95  PIPF = SFRADS
         PIPO = SORADS                                              SSM06400
         DPL = DPLI
         DPR = DPRI
         F = FCOMP
         EMRE = dwsum / (DW(1) + 1.0E-10)
C
C****       RESET INTEGRAL OF ERROR FOR PI CONTROLLER AT MAINSTAGE  ******
C
         IF ( STIME .GE. TMS .AND. .NOT. RESET) THEN
           RESET = .TRUE.
           EMROPO = YCOPOV - STEPO
           EPCGC=0.
*          PCOPOI=0.        Ineffective assignment replaced by
           PCOPOP = 0.
*
           PCOPOI= -VPG*EPC                                         SSM06500
           EMRFPO   = YCFPOV - STEPF
           EMRGC=0.
         ENDIF
C
C*************************************************************************
C   THIS SECTION DESCRIBES THE SCHEDULE OF THE SHUTDOWN PROCESSS
C   IT INCLUDES THE VALVE CLOSING, AND LINE PURGE SEQUENCES.
C*************************************************************************
C
         IF ( TPI .GT. 0.0 .AND. STIME .GE. TPI ) THEN
C
C*****       SIMULATION OF THE PNEUMATICAL CUT-OFF OF THE POWER  *****
C
           IF ( STIME .LT. TPI + DTPS) RETURN                       SSM09200
           IF( STIME .LT. tpm ) THEN
*
*  Eliminated unnecessary save and restore of TIMExx.
```

```
*
      tpm = TPI + DTMCR
      TPA = TPI + DTPS
          CALL EMCO(2)
          EMRE = ( DWMOV+DWOPO+DWFPO)/(DW(1)+1.0E-10)
          RETURN
        ENDIF
*
*  This section makes obscure timing interval adjustments that
*  should be explained. Perhaps it compensates for roundoff errors
*  accumulated in time advance. The first one is partially interpreted
*  below. Relief from roundoff time advance roundoff error should be
*  obtained by going to a frequency based, integer time advance. With all
*  important time intervals as multiples of the sampling interval.
*
        IF ( TIMFMA .LE. tpm) THEN
          TLMC = TIMFMA - DTFMRA
*                                          normally TLMC = TIME - DT
          IF ( TIMECP .LT. TLMC ) THEN
     +      TIMECP = DTMC + TIMECP
*                                        advances controller interval
          TIMECP=TIME+TIMECP-TLMC
*                          essentially TIMECP = TIMECP + DT, with
*                          perhaps some accuracy dependent effect
*
          IF(TIMEPR.LT.TLMC) TIMEPR=DTMC+TIMEPR
          TIMEPR=TIME+TIMEPR-TLMC
          IF(TIMEVC.LT.TLMC) TIMEVC=DTMC+TIMEVC                    SSM09400
          IF(TIMFME.LT.TLMC) TIMFME=DTMC+TIMFME
          TIMFME=TIME+TIMFME-TLMC
          IF(TIMFMC.LT.TLMC) TIMFMC=DTMC+TIMFMC
          TIMFMC=TIME+TIMFMC-TLMC
          IF(TIMETR.LT.TLMC) TIMETR=TIMETR+DTMC
          TIMETR=TIME+TIMETR-TLMC
          IF(TIMMRF.LT.TLMC) TIMMRF=TIMMRF+DTMC
          TIMMRF=TIME+TIMMRF-TLMC
          IFIND=0
          KOUNTF=0                                                 SSM09500
          IOIND=0
          KOUNTO=0
          TIMEVC=TIME+TIMEVC-TLMC
          TIMFMA=TIME+DTFMRA
        ENDIF
5015    IF(TIMEVC.GT.TPI+DTOPFS) GO TO 5020
          TIMEVC=TIMEVC+DTMC
          GO TO 5015
5020    IF( STIME .GE. TPI + DTVFS ) THEN
          TPI=0.
          tpm = DTMCR
          TPA=0.                                                   SSM09600
```

```
          TPA = DTPS
          GO TO 1008
        ENDIF
 5030   IF ( STIME .LT. TPI + DTOPFS ) THEN
          CALL EMCO(2)
          EMRE=(DWMOV+DWOPO+DWFPO)/(DW(1)+1.0E-10)
          IF(TIMEVC.LT.STIME) TIMEVC=TIMEVC+DTMC
          GO TO 1008
        ENDIF
 5040   XX=XCOPOV
        CALL EMCO(2)                                                    SSM09800
        XCOPOV=XX
        IF(TIMEVC.GT.TIME) GO TO 1008
        TIMEVC=TIMEVC+DTMC
        XCOPOV=YCOPOV
        GO TO 1008
      ENDIF

      IF ( TPA .GT. 0.0 .AND. STIME .GE. TPA ) THEN
C
*   Initiates purge request from malfunction detected
C
        NMALF = 1                                                       SSM08600
        TCUTPR = AMIN1( TCUTPR, TPA + DTPURG )
*       DT=DTI    Change in DT was disallowed, no restoration necessary.
        IF ( STIME .LE. TPA + DTPNC ) RETURN
        CALL EMCO(2)
        EMRE = ( DWMOV + DWOPO + DWFPO ) / ( DW(1) + 1.0E-10 )
        RETURN
      ENDIF
*
      GO TO (1002,1004,1006,1008),ICUT
*
*     Case ICUT is 1
*
*1002 IF(TIME.GT.TCUT.AND.FR.LT.1500.) ICUT=2
*     IF(TIME.GT.TCUT.AND.IPBCO.EQ.1) ICUT=2     Time vs TCUT once.
*     IF (TCUT.LT.TMS.AND.TIME.GT.TCUT)ICUT=2
 1002 IF ( STIME .GT. TCUT .AND.
     +       ( FR .LT. 1500. .OR. IPBCO .EQ. 1 .OR. TCUT .LT. TMS ) )
     +  ICUT = 2
*                 Delay transition state one DT
      GO TO 1008                                                        SSM06600
*
*    Case ICUT is 2: transition state goes immediately into state 3 or 4
*
 1004 TCUTFP = STIME
      TCUTOP = STIME
      TCUTOV = STIME
      ICUT = 3
```

```
*
*     Case ICUT is 3: wait until XCOPOV .LE. OPOVPB, then ___
*
 1006 IF(XCOPOV.GT.OPOVPB) GO TO 1008
      ICUT=4
      TCUTFV = STIME + DTCOMF
      TCUTCV = STIME + DTCOCV
      TMPL = STIME
*
*     Case ICUT is 4: all transitions completed
*
 1008 CONTINUE                                                    SSM06700
C
C****** FOLLOWING LINE CHECK WHETHER IT IS TIME TO ACTIVATE THE CONTROLLER
C****** THE CONTROLLER SAMPLING AND CYCLE TIME IS DTMC (=0.02)
*       DTMC is an input parameter, not a constant 0.02
C
*     IF(TIME.LT.TIMECP.AND.ICCNTL.EQ.2)  GO TO 4000  The ICCNTL test is
*   unnecessary, provided TIMECP is initialized to the starting time.
*
      IF ( STIME .LT. TIMECP ) GO TO 4000
C
C     CALCULATION OF CONTROL REFERENCE VALUES
C
C****** SAVE THE PREVIOUS CALCULATED VALUE WHICH WILL BE SENT OUT
C****** AT NEXT VALVE-COMMAND-OUTPUT TIME
C
      YCOPVL=YCOPOV
      YCFPVL=YCFPOV
      YCOMVL=YCMOV
      YCFMVL=YCMFV
      YCCVL=YCCCV
C
C                THRUST REFERENCE
C
      IF ( STIME .LE. TCUT ) THEN
        IF ( STIME .LE. TMOVRA ) THEN                               SSM06800
          IF ( STIME .LE. TCLF ) THEN
            FR = F1
          ELSE
            FR = AMAX1(FRZ, FR - df1dt)
          ENDIF
        ELSE
          IF( STIME .GT. TVC ) THEN
            FR = FGEN(33, 96, STIME)
          ELSE
            FR = AMIN1( F2 + DTF*(STIME - TMOVRA), FC )
          ENDIF
        ENDIF
      ELSE
```

```
*  1015 DT=DTI  restored DT after reset to 0.0001. But unless time runs
*               backward, each restoration is immediately undone by the
*               next statement.
*
*      IF(TIME.GT.TCUT+0.5) DT=0.0001 was deleted. It reset the time step
*    of the entire simulation to a constant, overriding the input parameter
*    DT. This disrupts all multistep integrators, and cannot be permitted.
*    The philosophy is questionable, allowing the simulated controller to
*    interfere with the simulation itself. There is no need to supply a
*    local version of DT for the controller to manipulate, since it doesn't
*    use DT. Increasing the time step DT with better integrators, speeding
*    up the simulation, and going to selective output should eliminate the
*    need for changing DT after TCUT.
*
        FR = AMAX1(FCO, FR - dfrcdt )
      ENDIF
C                                                                      SSM06900
C     THRUST CONTROL OF OXID PREBURNER OXID VALVE
C
C     THRUST AND PREBURNER OVERTEMP ERRORS
C
C*****      THE FOLLOWING LINE IS USED TO SIMULATE A PERTURBATION ON THE
C*****      REFERENCE INPUT FR (CHAMBER PRESSURE) REQUEST DURING THE CLOSED-
C*****      CONTROL.  PFRNZ IS THE % NOISE APPLIED TO THE REFERENCE INPUT FR
*    PFRNZ is a constant bias set by the main program perturbation section
*    but initialized to zero in CNTROL.
*
*     FR = FR*(1. + PFRNZ/100.)  was moved to CNTRL0
*
      FR = FR * frbias
C
C*****      TOPEN IS A PARAMETER ADDED TO OPEN THE CONTROL LOOP TO SIMULATE
C*****      OPEN LOOP CONDITION OF THE SSME MAIN STAGE OPERATION.
C*****      IT TURNS OUT THAT THE SSME IS A STABLE SYSTEM AS SIMULATED
C      Does the last comment mean open loop simulation is no longer needed?
*      if so TOPEN and its activated sections should be removed. Assuming
*      that open loop tests will continue,  open loop operations were
*      were consolidated, with one test per iteration for open loop
*      conditions. Control through IPFLAG, described below, was implemented.
*
      IF ( STIME .LT. TOPEN .OR. IPFLAG .EQ. 0 ) THEN
        EPC = FR - PCIE
        EPCGCL = EPCGC
C
C*****      PCGC IS THE GAIN FACTOR DETERMINE HOW MUCH THE POSITION ERROR IS
C*****      USED FOR PI CONTROLLER.
C
        PCGC = rlimit( GMIN, 1., GPC1 + GPC2*FR )
        EPCGC = EPC * PCGC
C
```

```
C*****  THE NEXT LINE SHOULDN'T BE HERE.  IT IS ABOUT THE MIX-RATIO CONTROL
C*****       AND AN EXECT SAME LINE IS IN SSM07570.  THIS IS THE DEAD-BAND OF
C*****       CONTROLLER.  I ADDED THE LINE AFTER THAT FOR CHECKING EPCGC.
C
C      IF(ABS(EMRGC).LT.DBMR) EMRGC=0.0                                  SSM07000
*
         IF(ABS(EPCGC).LT.DBEF) EPCGC=0.0
C
C*****       FOR T < TCLF              OPEN-LOOP, NO CONTROL
C*****       FOR TCLF < T < TUT        POSITION CONTROL ONLY, KI=0
C*****       FOR T > TUT               PI CONTROLLER APPLIES
C
         IF ( STIME .LE. TUT) THEN
           VIG=0.0
           VPG=0.01444
         ELSE
           VIG=VIGMS
           VPG=VPGMS
         ENDIF
*
         OPOVIP = FGEN(56, 97, STIME + DTMC)
         IF ( STIME .LT. TMS )
     +      EMROPO=OPOVIP                                               SSM07100
C
         IF ( STIME .LT. TCLF) THEN
           PCOPO = 0.0
         ELSE
           PCOPOP = VPG*EPCGC
           PCOPOI = AMIN1( XOMAX, 0.5*(EPCGC + EPCGCL)*VIG*DTMC )
           PCOPO = rlimit( -100., 100., PCOPOP + PCOPOI )
         ENDIF
C
C      OXID PREBURNER OXID VALVE POSITION
C
         IF ( STIME .GE. TCUTOP ) THEN                                   SSM07200
C
C*****       CUT-OFF SCHEDULE OF OPOV  ********
C    OSLAM was added as a input parameter, and invariant multiplies
*    removed from simulation loop
*
           IF ( STIME .GT. TCUTOP + DTSLAM) THEN
             YCOPOV = AMAX1(.00001, YCOPOV - osdt)
           ELSE IF ( XCOPOV .GT. OPOVCM) THEN
             YCOPOV = AMAX1(.00001, YCOPOV - op0dt)
           ELSE IF ( XCOPOV .GT. OPOVCP) THEN
             YCOPOV = AMAX1(.00001, YCOPOV - op1dt)
           ELSE
             YCOPOV = AMAX1(.00001, YCOPOV - op2dt)
           ENDIF                                                         SSM07300
C
```

```
C*****      CALCULATE OPOV COMMANDS AT DIFFERENT STAGE
C
        ELSE
          IF ( STIME .LT. TUT ) THEN
            IF( STIME .GT. TCLF) THEN
              RO = RO3
            ELSE IF ( YCOPOV .LT. OPOVOP ) THEN
              RO = RO1
            ELSE
              RO = RO2
            ENDIF
            IF ( STIME .LE. TOPOV ) THEN
              YCOPOV = 0.0
            ELSE
              YCOPOV = AMIN1( EMROPO + PCOPO, YCOPOV + RO*DTMC )
            ENDIF
          ELSE
            STEPO = OSTEP
*
C     *         *           OPOV COMMAND LIMIT                          SSM07400
*     Function TLIMIT was replaced by a call to fgen, which
*     does linear interpolation in an optimal manner.
*     TLIMIT data was reconstructed and added to SSME.DAT file.
*     TLIMIT was optimized, but extrapolated beyond the table.
*     fgen halts the simulation and identifies the table
*     if the input escapes the interpolation table.
*
*      EFFPL = 100.0 * FR / 2995.0 + 1.5 * XOPOMS - 97.5
*      YCOPOV = TLIMIT (FR, XOPLIM, XOPOMS, YCOPOV, NMR)
*
            EFFPL = v29.95 * FR + 1.5 * XOPOMS - 97.5
            YCOPOV = rlimit(.00001, 100.0, EMROPO + PCOPO + STEPO )
            YCOPOV = fgen( 84, 114, EFFPL )
*
            IF (YCOPOV .LT. XOPLIM) THEN
              NMR = 1                                                   SSM10400
            ELSE
              NMR = 0
              YCOPOV = XOPLIM
            ENDIF
          ENDIF
        ENDIF
C

C                MRC REFERENCE
C
        IF ( TMRS .GE. 1.0 .AND. STIME .GE. TMRS ) THEN
          EMRCR = 5.5
        ELSE
          EMRCR=6.026
```

```
        ENDIF
C
C     MR CONTROL USING FUEL PREBURNER OXID VALVE                        SSM07500
C
C*****      THE FOLLOWING LINE IS USED TO SIMULATE A PERTURBATION ON THE
C*****      REFERENCE INPUT EMRCR (MIX-RATIO) REQUEST DURING THE CLOSED-LOOP
C*****      CONTROL.  THE MIX-RATIO (EMRF) TO BE CONTROLLED IS NOT THE ACTUA
C   MIXRATIO IN THE COMBUSTION CHAMBER (NOT MEASUREABLE), IT IS THE
C   ESTIMATED MIX-RATIO BASED ON CHAMBER PRESSURE AND FUEL FLOW RATE.
C
        EMRCR = EMRCR * pmrnzb
C
C   THE DEFINITION OF THE ERROR IS DIFFERENT FROM THE COMMON DEFINITION.
C   HERE:   ERROR= - (REFERENCE - OUTPUT/MEASUREMENT)
C   WHICH IS BECAUSE OF THE NEGATIVE GAIN OF FPOV VS. MIX-RATIO.
C
        IF ( TIME .LT. TMRC ) THEN
          EEMR=0.0
          EMRGCL=EMRGC
        ELSE
          EEMR=EMRF-EMRCR
          EMRGCL=EMRGC
        ENDIF
        GCEMR = rlimit( GMRMIN, GMRMAX, GMR1 + GMR2 * FR )
        EMRGC=EEMR*GCEMR
        IF ( ABS(EMRGC) .LT. DBMR ) EMRGC=0.0
C
C*****      THE ADJUSTMENT OF FPOV IS IN PROPORTIONAL TO THE OPOV ADJUSTMENT
C*****      THIS RATIO IS A FUNCTION OF TIME
C
        IF( STIME .GE. TMS ) THEN
          PCFPO = CFGMS * PCOPO
        ELSE
          PCFPO = PCOPO * rlimit( CFG1, CFG2,
     +                            DCFG*(STIME - TCLF) + CFGC )
        ENDIF
C
C*****      IF TIME < TMRC OPEN LOOP WITH SCHEDULED VALVE POSITION  *******
C*****              > TMRC PI CONTROL                               *******
C
        IF ( STIME .GE. TMRC ) THEN                                   SSM07600
C
C*****      THE INTEGRAL GAIN IN THIS CASE IS 40.0 AND KP = VPM (=7.0) *****
C                                              20. = 40. / 2.
          EMRFPO = rlimit(.00001, 100., EMRFPO +
     +             DTMC*20.*(EMRGC + EMRGCL) + VPM*(EMRGC - EMRGCL) )
C
C*****      START-UP SCHEDULING OF FPOV  **********
C
        ELSE
```

```
            IF ( STIME .LE. -0.1 ) THEN
              EMRFPO = 0.0
            ELSE
              FPOVIP = FGEN(57, 98, STIME + DTMC)
              IF ( STIME .GT. TCLF) THEN
                RO = RF3                                                SSM07700
              ELSE IF ( YCFPOV .LT. FPOVOP) THEN
                RO = RF1
              ELSE
                RO=RF2
              ENDIF
              EMRFPO = AMIN1( FPOVIP, EMRFPO + RO*DTMC )
            ENDIF
          ENDIF
*
          IF (TIME .LE. TCFG)  PCFPO = 0.0
          STEPF    = FSTEP
          IF (TIME .LT. TUT) THEN
            STEPF = 0.0
            IF ( STIME .LT. TCUTFP ) THEN
              YCFPOV = AMIN1(EMRFPO + PCFPO, YCFPOV + RO*DTMC)
C
C*****        FPOV CUT-OFF PROCESS  **********
C
            ELSE IF ( XCFPOV - XCOPOV .GE. DXCFPO
     +                       .OR. XCOPOV .LE. 10. ) THEN                SSM07800
              IF ( XCFPOV .GT. FPOVCM ) THEN
                YCFPOV = AMAX1( .00001, YCFPOV - fp0dt )
              ELSE IF ( XCFPOV .GT. FPOVCP) THEN
                YCFPOV = AMAX1(.00001, YCFPOV - fp1dt)
              ELSE
                YCFPOV = AMAX1(.00001, YCFPOV - fp2dt)
              ENDIF
            ENDIF
          ELSE IF ( STIME .LT. TCUTFP ) THEN
            IF ( STIME .LE. TMPL .OR. TMPL .GE. TMRC) THEN
               YCFPOV = rlimit(.00001, 100.0, EMRFPO + PCFPO + STEPF) SSM07900
            ENDIF
          ENDIF
          IF (TIME .LT. T1FPV)  YCFPOV = AMIN1 (FPOVMX, YCFPOV)
C
C                 MOV POSITION
C
C*****        MOV OPENING IS TIME SCHEDULED BEFORE THE MAIN STAGE
C*****        AFTER THAT, IT IS PROPORTIONAL TO THRUST REQUEST (FR)
C
 1300     XCMOVL = XCMOVC
          IF ( STIME .LT. TCUTOV ) THEN
            IF ( STIME .GE. TMOVRA ) THEN
              IF ( STIME .GT. TMS ) THEN
```

```
              XCMOVC = AMIN1(100., DFMOV*FR + FMOVC)
            ELSE
              XCMOVC = AMIN1(XMOVM, XMOVST + (STIME - TMOVRA)*DMOVUT) M08100
            ENDIF
          ELSE IF ( STIME .LT. TMPL) THEN
            IF ( XCFPOV - XCOPOV .LT. DXCFPO ) THEN
              XCMOVC = XCMOVC + omv0dt
            ELSE
            XCMOVC = AMAX1(.00001, XCMOVC - omv0dt)
            ENDIF
          ELSE IF ( STIME .GT. TIMOV .AND. STIME .LT. T2MOV ) THEN
            XCMOVC = XMOVPT
          ELSE
            IF ( T2MOV .GE. 1. .AND. STIME .GE. T2MOV) omvdt = omvfdt M08000
            XCMOVC = AMAX1(.00001, XCMOVC - omvdt )
          ENDIF
*
*         YCMOV=(YCMOV*(2.0*.16-DTMC)+DTMC*(XCMOVL+XCMOVC))/(2.0*.16+DTMC)
*             Is this correct? Was average xcmov intended? Invariants were
*             removed.
          YCMOV = YCMOV * ydt1 + (XCMOVL + XCMOVC) * ydt2
*
        ELSE IF ( STIME .GT. TSMOV) THEN
          XCMOVC = AMIN1(XMOVST, XCMOVC + dmovdt)
          IF ( SF2 .GE. SF2R ) XMOVST = XMOV2
          YCMOV = YCMOV * ydt1 + (XCMOVL + XCMOVC) * ydt2
        ENDIF
C
C                 COOLANT CONTROL VALVE
C
C*****      THE CCV OPENING IS MAINLY TIME SCHEDULED
C*****      HOWEVER, WHEN T > 2.8 AND FR > 1500, CCV IS PROPORTION TO FR
C   THE VALVE POSITION IS FULLY OPEN DURING THE MAIN STAGE OPERATION
C
C     IF(TIME.LT.TCUT+.5) GO TO 1800
C     YCCCV=AMAX1(75.,YCCCV-DCCV1*DTMC)
C     GO TO 1800
C
        IF ( STIME .LE. TCUT ) THEN
          IF ( FR .GE. 1500. ) THEN
C
C  CCV NOT RATE LIMITED HERE - THRUST SCHEDULED                          SSM08400
C
            IF ( STIME .GT. 2.8 ) THEN
              YCCCV = AMIN1( DFCCV*FR + FCCVC, 100. )
            ENDIF
*
          ELSE IF ( STIME .LT. TCUTCV) THEN
            IF ( STIME .GT .TCCV1 ) THEN
              IF ( STIME .GE. TMOVRA) THEN
```

```
              YCCCV = AMAX1(CCV3, YCCCV - dcc3dt)
            ELSE
              YCCCV = AMAX1(YCCCV - dcc2dt, CCV2)
            ENDIF
          ELSE
            YCCCV = AMIN1(YCCCV + dccvdt, CCVC)
          ENDIF
        ELSE IF ( STIME .LT. TMPL + DTMFVR ) THEN
          YCCCV = AMIN1 (CCV1, YCCCV + dcc1dt)
        ELSE
          YCCCV = AMAX1 (.00001, YCCCV - dcc0dt)
        ENDIF
      ELSE IF ( ICUT .EQ. 1 ) THEN
C
C   CCV NOT RATE LIMITED HERE - THRUST SCHEDULED                          SSM08400
C
        IF ( STIME .GT. 2.8 ) THEN
          YCCCV = AMIN1( DFCCV*FR + FCCVC, 100. )
        ENDIF
*
      ELSE IF ( STIME .LT. TCUTCV ) THEN
        IF (XCFPOV - XCOPOV .GE. DXCFPO .AND. XOPOV .GT. OPOVPB) THEN
          YCCCV = AMAX1 (XCCCV0, YCCCV - dcc0dt)
        ENDIF
      ELSE  IF ( STIME .LT. TMPL + DTMFVR ) THEN
        YCCCV = AMIN1 (CCV1, YCCCV + dcc1dt)
      ELSE
        YCCCV = AMAX1 (.00001, YCCCV - dcc0dt)
      ENDIF
C
C                MAIN FUEL VALVE
C
C*****      MFV OPENING IS MAINLY TIME SCHEDULED AT THE BEGINING  **********
C*****      WHEN T > TFRMFV, THE OPENING IS PROPORTIONAL TO FR     **********
C   HOWEVER, THE VALVE IS FULLY OPEN AT THE MAIN STAGE.
C
      IF ( TCUT .GE. 5.0 .OR. STIME .LE. TCUT ) THEN
        IF( STIME .GT. TCUTFV ) THEN
C
C*****      SHUT-OFF PROCESS  ************
C
          IF( STIME .LE. TMPL + DTMFVR) THEN
            YCMFV = AMAX1(P1MFV, YCMFV - dp1dt )
          ELSE
            YCMFV = recpos( YCMFV - dp2dt)
          ENDIF
        ELSE IF ( STIME.GT.TFRMFV) THEN
          YCMFV = rlimit( FMVM, XMFVM, CMFV + DFMFV * FR )
        ELSE
          YCMFV = FGEN(55, 90, STIME + DTMC)
```

```
          ENDIF
        ENDIF
      ELSE
*
*   Open loop operation
*
        EPC = 0.
        EPCGCL=0.
C
C*****      FOLLOWING FOUR LINES ARE ADDED TO SIMULATE PERTURBATION ON THE V
C*****      OPENINGS AT THE MAIN STAGE.
C*****              IPFLAG = 0 FOR CLOSED-LOOP CONTROL
C*****                       1 FOR OPEN-LOOP NO PERTURBATIONS
C*****                       2 FOR OPEN-LOOP AND PERTURBATIONS ON VALVE OPEN
C    ipflag = 2 had not been implemented. Since it involves changing the
*               constant setting of the frozen control valve, it was
*               implemented in CNTROL0.
*
*       YCOPOV = FZOPV * (1 + POPVNZ/100)
*       IF(YCOPOV.GE.100.0)YCOPOV=FZOPV  was replaced by
*
        YCOPOP = fzopvb
*                              and invariant operations moved to CNTROL0.
        EEMR=0.0
        EMRGCL=0.
*
C*****      FOLLOWING FOUR LINES ARE ADDED TO SIMULATE PERTURBATION ON THE V
C*****      OPENINGS AT THE MAIN STAGE.
C
*     IF (TIME.LT.TOPEN)GO TO 1208
*       YCFPOV = FZFPV*(1 + PFPVNZ/100)     was replaced,
*     IF(YCFPOV.GE.100.0)YCFPOV=FZFPV       moving invariant operations to
*1208 CONTINUE                              CNTRL0.
C
        YCFPOV = fzycf
C
C*****      FOLLOWING FOUR LINES ARE ADDED TO SIMULATE PERTURBATION ON THE V
C*****      OPENINGS AT THE MAIN STAGE.
C
*     IF(TIME.LT.TOPEN) GO TO 1488
*     YCMOV = FZMOV * (1 + PMOVNZ/100)
*     IF(YCMOV.GE.100.0)YCMOV=FZMOV
*1488 CONTINUE
*
        YCMOV = fzycm
C
C*****      FOLLOWING FOUR LINES ARE ADDED TO SIMULATE PERTURBATION ON THE V
C*****      OPENINGS AT THE MAIN STAGE.
C
*     IF(TIME.LT.TOPEN) GO TO 1888
```

```
*      YCCCV = FZCCV * (1 + PCCVNZ/100)
*      IF(YCCCV.GE.100.0)YCCCV=FZCCV
*1888 CONTINUE
*
         YCCCV = fzycc
C
C*****       FOLLOWING FOUR LINES ARE ADDED TO SIMULATE PERTURBATION ON THE V
C*****       OPENINGS AT THE MAIN STAGE.
C
*      IF(TIME.LT.TOPEN) GO TO 1588
*      YCMFV = FZMFV * (1 + PMFVNZ/100)
*      IF(YCMFV.GE.100.0)YCMFV=FZMFV
*1588 CONTINUE
*
         YCMFV = fzycmf
       ENDIF
*
*      IF(ICCNTL.EQ.2) TIMECP=TIMECP+DTMC
*
       TIMECP = TIMECP + DTMC
C
C*****       THESE VALUES ARE TO FREEZE THE OPENINGS OF CONTROL VALVES  ****
*      Simulation loop invariant perturbations added.
C
       IF ( IPFLAG .EQ. 0 .AND. STIME + DTMC .GE. TOPEN) THEN
         FZOPV = YCOPOV
         fzopvb = FZOPV * ( 1. + .01 * POPVNZ )
         IF ( fzopvb .GE. 100.0 ) fzopvb = FZOPV
         FZFPV = YCFPOV
         fzycf = FZFPV * ( 1. + .01 * PFPVNZ )
         IF ( fzycf .GE. 100.0 ) fzycf = FZFPV
         FZMOV = YCMOV
         fzycm = FZMOV * ( 1. + .01 * PMOVNZ )
         IF ( fzcm .GE. 100.0 ) fzycm = FZMOV
         FZMFV = YCMFV
         fzycmf = FZMFV * ( 1. + .01 * PMFVNZ )
         IF ( fzycmf .GE. 100.0 ) fzycmf = FZMFV
         FZCCV = YCCCV
         fzycc = FZCCV * ( 1. + .01 * PCCVNZ )
         IF ( fzycc .GE. 100.0 ) fzycc = FZCCV
         IPFLAG = 1
       ENDIF
C
C*****       FOLLOWING IS THE SCHEDULE OF HANDLING DIFFERENT INSTRUMENTATION
C*****       TIME DELAYS DURING THE SAMPLING TIME PERIOD DTMC (=0.02) OF
C*****       CONTROL LOOP.
C
 4000 IF ( STIME .GE. TIMEPR .OR. first ) THEN
         PCF = PCFACT * PCIE
         IF ( .NOT. first ) TIMEPR=TIMEPR+DTMC
```

```
      ENDIF
      IF ( STIME .GE. TIMETR .OR. first ) THEN                          SSM08700
        TO=TOD2
        IF( .NOT. first ) TIMETR=TIMETR+DTMC
      ENDIF
C
C*****      EXECUTION OF VALVE OPENING COMMANDS  ***********
C*****      NOTICE THE OUTPUT COMMANDS ARE USING THE OLD (YCxxVL) VALUES STO
C*****      CURRENT CALCUALTED VALUES ARE TO BE SENT OUT IN NEXT TIME INTERV
C   THE RATE OF CHANGE IS LIMITED TO 200% OPENING/SEC.
*  Method changed to eliminate a multiply and divide per value.
C
      IF ( STIME .GE. TIMEVC .OR. first ) THEN
        IF (TIME .GT. TCLF .AND. TIME .LT. TCUT) THEN
          XRFPV = dtm100
        ELSE
          XRFPV = dtm200
        ENDIF
*
        yxdiff = YCFPVL - XCFPOV
        abdiff = ABS( yxdiff )
        IF ( abdiff .LT. XRFPV ) THEN
          XCFPOV = XCFPOV + SIGN( abdiff, yxdiff )
        ELSE
          XCFPOV = XCFPOV + SIGN( XRFPV, yxdiff )
        ENDIF
*
        yxdiff = YCOPVL - XCOPOV
        abdiff = ABS( yxdiff )
        IF ( abdiff .LT. dtm200 ) THEN
          XCOPOV = XCOPOV + SIGN( abdiff, yxdiff )
        ELSE
          XCOPOV = XCOPOV + SIGN( dtm200, yxdiff )
        ENDIF
*
        yxdiff = YCOMVL - XCMOV
        abdiff = ABS( yxdiff )
        IF ( abdiff .LT. dtm200 ) THEN
          XCMOV = XCMOV + SIGN( abdiff, yxdiff )
        ELSE
          XCMOV = XCMOV + SIGN( dtm200, yxdiff )                        SSM08800
        ENDIF
*
        yxdiff = YCCVL - XCCCV
        abdiff = ABS( yxdiff )
        IF ( abdiff .LT. dtm200 ) THEN
          XCCCV = XCCCV + SIGN( abdiff, yxdiff )
        ELSE
          XCCCV = XCCCV + SIGN( dtm200, yxdiff )
        ENDIF
```

```
*
        yxdiff = YCFMVL - XCMFV
        abdiff = ABS( yxdiff )
        IF ( abdiff .LT. dtm200 ) THEN
          XCMFV = XCMFV + SIGN( abdiff, yxdiff )
        ELSE
          XCMFV = XCMFV + SIGN( dtm200, yxdiff )
        ENDIF
      ENDIF
*
      IF(ICCNTL.EQ.2) TIMEVC=TIMEVC+DTMC
*
      IF ( STIME .GE. TIMFMA) THEN
        IF(IFIND.EQ.0) IFIND=1
        IF(IOIND.EQ.0) IOIND=1
        TIMFMA=TIMFMA + DTMC                                          SSM08900
      ENDIF
      IF ( STIME .GE. TIMFME ) THEN
        TIMFME = TIMFME + DTMC
        IF( IFIND .EQ. 3) THEN
          NCF=KOUNTF
          KOUNTF=0
          IFIND=0
        ENDIF
        IF( IOIND .EQ. 3 ) THEN
          NCO=KOUNTO
          KOUNTO=0
          IOIND=0
        ENDIF
      ENDIF                                                           SSM09000
*4050 IF(TIME.LT.TIMFMC) GO TO 4060
*     TIMFMC=TIMFMC+DTMC
C
C***** QF0, Q0, RHOH, AND RHO0 ARE NEVER USED EXCEPT PRINT-OUTS ******
C
*     QF0=67.11*300000./(NCF+1.0E-5)     Computations for output only
*     Q0=22.95*300000./(NCO+1.0E-5)      are to be done in offline
*     RHOH=DENF                          plotting programs, not in the
*     RHO0=DENO                          simulation
*
 4060 IF(TIME.LT.TIMMRF) RETURN
C
C*****      EMRF IS THE CALCULATED MIX-RATION FROM MEASURED VARIABLES OF
C*****  CHAMBER PRESSURE AND FUEL FLOW RATE
C
      IF ( STIME .GE. TMRC) THEN
        IF (NMR .EQ. 1) THEN
          EMRF  = PCNS / (C2 * DW(2)) - 1.0                           SSM09100
        ELSE
          EMRF = FR / (C2 * DW(2)) - 1.0
```

```
        ENDIF
      ENDIF
*
*  Here is where the KOUNT variables affect the simulation. There is
*  an effect only when TIME .GE. TMRC
*
      IF ( TIME .LT. TMRC) THEN
        EMRF = 6.0
      ELSE IF ( NCO.EQ.0 .OR. NCF.EQ.0) THEN
        EMRF = 0.
      ENDIF
      F = ( FA1*PCF + FB1*EMRF*PCF +FC1*EMRF +FD1 ) * TFACT
      F = AINT(F/512.)*512.
      TIMMRF = TIMMRF + DTMC
      FCOMP = F
      END                                                          SSM09900
*
*  The function TLIMIT was removed. Its interpolation is done by
*  fgen. Its data was reconstructed from the interpolation
*  coefficients below.
C
C  FUNCTION TLIMIT(FR,OXPLIM,XOPOMS,YCOPOV,NMR0
C
C  PURPOSE:  COMPUTE THE OPOV COMMAND LIMITS USING FLIGHT 38 - MERGE
C
C******ARGUMENTS******
C  INPUT:
C   FR     = COMMAND PC
C   XOPLIM = MAXIMUM OPOV POSITION ALLOWED
C   XOPOMS = VALUE OF OPOV POSITION AT RPL (NOMINALLY 65%)          SSM10000
C   YCOPOV = CONTROLLER COMMAND VALUE OF XOPOV BEFORE LIMITING
C   NMR    = FLAG TO SELECT METHOD OF MIXTURE RATIO COMPUTATION
C
C  OUTPUT:
C   TLIMIT = LIMITED YCOPOV
C
**********************************************************************
C
*     DIMENSION EPLTAB(11), A1TAB(11), A0TAB(11)
C
*                                                                    SSM10100
*     DATA EPLTAB/  0.0, 70.0, 75.0, 80.0, 85.0, 90.0, 95.0, 100.0,
*    1       105.0,110.0,1000./
*     DATA A1TAB /  0.0, .100, .118, .218, .340, .414, .428, .447,
*    1        .689,1.564,3.722/
*     DATA A0TAB /  0.0,49.55,48.29,40.79,31.03,24.74,23.48,21.71,
*    1       -2.57,-94.4,-331.78/
*     DATA M     / 8 /
*
*    The y value is available from the interpolation coefficients
```

```
*     a0(i), a1(i) as y(i-1) = a0(i) + a1(i) * x(i-1)
*     where x = EPLTAB and y = TLIMIT:
*
*     TLIMIT = 49.55, 56.55, 57.14, 58.23, 59.93, 62.0, 64.18,
*              66.33, 69.82, 77.64, 3390.
*
*     TLIMIT vs. EPLTAB was put in 'ssme.dat', as fcn # 84, after
*                       fcn #57
```

'emco.for':

```
      SUBROUTINE EMCO0
************************************************************************
C
C  PURPOSE:  COPUTE ACTUATOR POSITIONS DURING PNEUMATIC SHUTDOWN
C   DURING THE EMERGENCY CUT-OFF THE VALVE COMMANDS HAS A SPECIAL
C   CIRCUIT FOR SEQUENTIALLY SHUT-DOWN (OR OPEN) THE VALVES.
C
C !!!!! THIS PORTION OF PROGRAM HAS NOT BEEN CHECKED OUT BECAUSE OF THE LACK
C !!!!! OF DOCUMENT ON THE ACTUAL SHUT-DOWN PROCEDURE AND DYNAMICS
C
C                                                                    SSM10500
C******ARGUMENT******
C INPUT:
C   N = INITIALIZATION ARGUMENT was eliminated
C
C******COMMON USAGE******
C INPUT:
C   VARIABLES                                          DESTINATION
C    PFPOV,POPOV,DWFPO,DWOPO,PFPOI,POPOI               HOTGAS
C    PMOV,DWMOV,RHOOP3,RHOOP2,POINJ                    OXIDF
C    P(3),P(7),RHO(3),RHO(7),P(10),P(8)                FUELF     SSM10600
C    XFPOV,XOPOV,XMOV,XMFV,XCCV                        VALDYM
C
C  OUTPUT:
C   VARIABLES                                          DESTINATION
C    THETA,DTHETA                                      VALDYM
C
*     DOUBLE PRECISION DTHETA, ESAC, ESA, DESA, ESV, DESV          SSM10700
*     DOUBLE PRECISION TIME
************************************************************************
*
      DIMENSION FRDEL(5),FS(5),TL(5),PCT(5),A1(5),A2(5),A3(5),B1(5),S(5)0800
     1    ,D(5),E(5),CKT1(5),CKL1(5),HKT1(5),HKL1(5),CKT2(5),CKL2(5),
     2    HKT2(5),HKL2(5),CKT3(5),CKL3(5),HKT3(5),HKL3(5),HTO(5,3),W2(3)
     3    ,H2(3),H3(3),ORFMAX(3),ORFMIN(3),TOMAX(3),TOMIN(3),PNFRIC(3)
     4    ,PNLOAD(3),PNORGX(3),PNK(3),WTO(3),Y(5),A0(5),PHID(5),X2DOT(5)
     5    ,FY1(5),FY2(5),X1DOT(5),I1(5),X1(5),DY(5)
     6    ,N1(5),PIN(5)
     7    ,              W(5),DEN(5),PD(5),A4(5)
```

```
      REAL L(5),M(5),MU(5),KT1(5),KL1(5),KT2(5),KL2(5),KT3(5),KL3(5),
     *          KT0(5),K3
      DIMENSION XP(5), vG1(5), XHE(5)
      LOGICAL JK(5)
      REAL dmesq(5), twod(5), sqm(5), slx(5)
*
      PARAMETER ( PI = 3.141597, degprd = 180. / PI,
     +                           vperi = 8.0/(386.1 * PI ) )
*
      INCLUDE 'blank.com'
      INCLUDE 'units.com'
      INCLUDE 'out.com'
      INCLUDE 'contrl.com'
      INCLUDE 'oxid.com'
      INCLUDE 'hgas.com'
*
      DATA I1/11,14,16,32,36/
      DATA JK, FY1, FY2, D1, D2/ 5 * .TRUE., 10*0., -1.085, 1.02/
*****************************************************************
*
      CALL fgset( 34 )                                          SSM11400
      CALL fgset( 35 )
      CALL fgset( 11 )
      CALL fgset( 14 )
      CALL fgset( 16 )
      CALL fgset( 32 )
      CALL fgset( 36 )
      FR = fgen( 36, 108, 100. )
*
      READ(run,10) PHE, PR, FRDEL, FS, TL,
     +             TEMP, PCT,( A1(I), A2(I), I=1,5 ),
     +             A3, B1, S, L, M, D, E,                       SSM11500
     +             MU, CKT1, CKL1, HKT1, HKL1, CKT2,
     +             CKL2, HKT2, HKL2, CKT3, CKL3, HKT3, HKL3,
     +             ( (HTO(I,J), I=1,5 ), J=1,3 ),
     +             W2, H2, H3, ORFMAX, ORFMIN, TOMAX, TOMIN,
     +             PNFRIC, PNLOAD, PNORGX, PNK, WTO,
     +             TSFPOV, TSMOV, TSMFV,
     +             (AT(I),I=1,5), (BT(I),I=1,5),
     3,          (CS(I),I=1,5),(CPS(I),I=1,5),(DSS(I),I=1,5)
     4,          (ESS(I),I=1,5),(VPD(I),I=1,5),(A4(I),I=1,5)
  10  FORMAT (//2X,6g12.4)
C
C   THE FOLLOWING TIME CONSTANT WERE SET TO 0.0 DURING THE CUT-OFF.
C   HOWEVER, DURING THE INITIALIZATION OF THE PROGRAM THESE VALUES SHOULD
C   BE SAVED BEFORE THE EMCO(1) WAS CALLED DURING THE DATA READING.
C
*      TIMEVC=0.0       If there are values to save, then don't
*      TIMEPR=0.0       change them.
*      TIMECP=0.0
```

```
*       TIMETR=0.0
*
C      TPT=TPA
       PRC=PR
       PPB=PHE
C
C     TEMP IS THE TEMPERATURE INDICATOR, 1=COLD, 2=NORMAL, 3=HOT
C
       IF(TEMP.LE.1.0)GAM=SQRT(.86158)                                  SSM11900
       IF(TEMP.EQ.2.0)GAM=SQRT(.83974)
       IF(TEMP.EQ.3.0)GAM=SQRT(.7895)
       IT=TEMP+0.1
       G2=4.*146.2*.62*W2(IT)*H2(IT)/GAM
       W3=(ORFMAX(IT)-ORFMIN(IT))
       K3=4.*146.2*.62*W3*H3(IT)/GAM
       XV=(PPB*.1398-PR*.1398-PNFRIC(IT)-6.0*PNLOAD (IT)-6.0*PNORGX(IT)
      1   *PNK(IT))/(PNK(IT)*6.0)
       IF(XV.LT.0.)XV=1.E-5
       IF(XV.GT.WTO(IT)) XV=WTO(IT)                                     SSM12000
       G3=(K3*(XV-ORFMIN(IT)))/(ORFMAX(IT)-ORFMIN(IT))
       IF(XV.LT.ORFMIN(IT))G3=.000001
       IF(XV.LT.ORFMIN(IT))G2=.000001
       IF(XV.GE.ORFMAX(IT))G3=K3
       GE=SQRT(1./((1./G3)**2+(1./G2)**2))
C
C          1 = FPOV
C          2 = OPOV
C          3 = MOV                                                      SSM12100
C          4 = MFV
C          5 = CCV
C
       DO 720 I=1,5
         KT1(I)=CKT1(I)
         KL1(I)=CKL1(I)
         KT2(I)=CKT2(I)
         KL2(I)=CKL2(I)
         KT3(I)=CKT3(I)
         KL3(I)=CKL3(I)
         IF( TEMP .GE. 2.0 ) THEN                                       SSM12200
           KT1(I)=HKT1(I)
           KL1(I)=HKL1(I)
           KT2(I)=HKT2(I)
           KL2(I)=HKL2(I)
           KT3(I)=HKT3(I)
           KL3(I)=HKL3(I)
         ENDIF
         CG0 = ( D(I)**2 + M(I)**2 + L(I)**2 - E(I)**2 ) /
      +        ( 2.*D(I)*SQRT( M(I)**2 + L(I)**2 ) )
         SG0 = SQRT(1. - CG0**2)                                        SSM12300
         GA0 = ATAN2(SG0,CG0)
```

```
          BE0 = ATAN( M(I) / L(I) )
          A0(I) = PI - GA0 - BE0
          KT0(I) = 146.2*.62*HTO(I,IT)/GAM
          XHE(I)=0.
          IF( XV.LE.TOMIN(IT) ) THEN
            vG1(I) = 1.E6
          ELSEIF ( XV .GT. TOMAX(IT) ) THEN
            vG1(I) = 1.0 / ( KT0(I)*(TOMAX(IT) - TOMIN(IT) ) )
          ELSE
            vG1(I) = 1.0 / ( KT0(I)*(XV - TOMIN(IT) ) )
          ENDIF
  720 CONTINUE
      DO 5 I = 1, 5
        Y(I) = THETA(I) / PCT(I)
        DY(I) = -1.0E-5
        CALL unint0( Y(I), 198 + I )
        X2DOT(I) = DY(I)
        X1DOT(I) = DY(I)
        X1(I) = Y(I)
        dmesq(I) = D(I)**2 + M(I)**2 - E(I)**2
        twod(I) = 2. * D(I)
        sqm(I) = M(I) ** 2
        slx(I) = L(I) + S(I) - X1(I)
    5 CONTINUE
      RETURN
*
      ENTRY EMCO
*************************************************************************
 1000 CONTINUE                                                    SSM12400
      PIN(1)=PFPOV
      PIN(2)=POPOV
      PIN(3)=PMOV
      PIN(4)=P(3)
      PIN(5)=P(7)
      W(1)=DWFPO
      W(2)=DWOPO
      W(3)=DWMOV
      W(4)=DW(3)
      W(5)=DW(7)                                                  SSM12500
      DEN(1)=RHOOP3
      DEN(2)=RHOOP3
      DEN(3)=RHOOP2
      DEN(4)=RHO(3)
      DEN(5)=RHO(7)
      PD(1)=PFPOI
      PD(2)=POPOI
      PD(3)=POINJ
      PD(4)=P(10)
      PD(5)=P(8)                                                  SSM12600
      XP(1) = .01 * XFPOV
```

```
      XP(2) = .01 * XOPOV
      XP(3) = .01 * XMOV
      XP(4) = .01 * XMFV
      XP(5) = .01 * XCCV
      X1DOT(5)=DTHETA(5)
      DO 2000 I=1,5
*
*     IF(I.EQ.1.AND.THETA(2).GT.TSFPOV) GO TO 2000
*     IF(I.EQ.3.AND.THETA(2).GT.TSMOV)  GO TO 2000
*     IF(I.EQ.4.AND.TIME.LE.TSMFV)      GO TO 2000                    SSM12700
*     IF(I.EQ.5.AND.TIME.LE.TSMFV) GO TO  2000  requires too many tests of I
*
        GO TO ( 15, 50, 20, 40, 40 ), I
   15     IF ( THETA(2) .GT. TSFPOV ) GO TO 2000
          GO TO 50
   20     IF ( THETA(2) .GT. TSMOV ) GO TO 2000
          GO TO 50
   40     IF ( STIME.LE.TSMFV ) GO TO 2000
   50   CONTINUE
*
*       IF ( N1(I) .EQ. 0 ) GO TO 422
*         Y(I) = THETA(I) / PCT(I)          Moved to EMC0
*         DY(I) = -1.0E-5
*         X2DOT(I) = DY(I)
*         X1DOT(I) = DY(I)
*         X1(I) = Y(I)
*         N1(I)=1
*
        IF(Y(I) .LE. 0.) GO TO 2000
        X2 = Y(I)                                                      SSM12800
*
*     CG=(D(I)**2+M(I)**2+(L(I)+X2)**2-E(I)**2)/(2.*D(I)*SQRT(M(I)**2+(
*    *   L(I)+X2)**2))  was replaced by
*
        x2l = X2 + L(I)
        x2lsq = x2l ** 2
        vdenBE = 1. / X10th( sqm(I) + x2lsq, 5 )
        CG = ( dmesq(i) + x2lsq ) * vdenBE / twod(I)
*
        SG = X10th( 1. - CG**2, 5 )
        GA = ATAN2(SG, CG)                    special functions analysis
        BE = ATAN( M(I)/x2l )                 not completed on this
        A = PI - GA - BE                      module
        PHI = A - A0(I)
        PHID(I) = degprd * PHI
*
        IF( X2DOT(I) .EQ. 0. ) THEN
          TP = 0.                                                      SSM12900
        ELSE
*
```

```
*        THK=FGEN(I1(I),2,XP(I)/100.)
*        TP(I)=SIGN(AT(I)+(BT(I)+0.8*ESS(I))*(PIN(I)-8./386.1/DEN(I)*(W(I)/
*     *       PI/VPD(I)**2)**2)+0.8*DSS(I)+CPS(I)+CS(I)*(PIN(I)-PD(I)),
*     *       X1DOT(I))+THK*W(I)**2/DEN(I)/VPD(I)/1728. was replaced by
*
           THK = fgen( I1(I), I + 103, xp(I) )
           TP = SIGN( AT(I) + ( BT(I) + 0.8*ESS(I) ) *
     +     ( PIN(I) - vperi/DEN(I)*( W(I)/VPD(I)**2 )**2 ) + 0.8*DSS(I) +
     +      CPS(I) + CS(I)*( PIN(I) - PD(I) ), X1DOT(I) ) +
     +          THK*W(I)**2 / ( DEN(I)*VPD(I)*1728. )
*
           ENDIF
C       FB1=MU(I)*ABS(FY1(I))
C       FB2=MU(I)*ABS(FY2(I))
           BV1 = B1(I)*.5
           BV2 = BV1                                                    SSM13000
*          IF(ABS(X2DOT(I)).LT..9E-4) JK(I)=1
           JK(I) = JK(I) .OR. X2DOT(I) .LT. .9E-4
           IF( JK(I) ) THEN
             FRIC1 = -SIGN( FS(I)*.5, X2DOT(I) )
             FRIC2 = FRIC1
           ELSE
*
*          IF(I.LT.3.OR.I.EQ.5) FR=FGEN(34,2,PHID(I))  Why lookup twice for I =
*          IF(I.GT.2)FR=FGEN(35,2,PHID(I))             replaced by
*
             IF( I .LT.3 ) THEN
               FR = FGEN( 34, 108 + I, PHID(I) )
             ELSE
               FR = FGEN( 35, 108 + I, PHID(I) )
             ENDIF
             FR = FR + FRDEL(I)
             FRIC1 = -SIGN(FR*.5, X1DOT(I) )
             FRIC2 = -SIGN(FR*.5, X2DOT(I) )                            SSM13100
           ENDIF
*
*          SINA=SIN(A)
*          COSA=COS(A)      replaced by
*
           sinBE = M(I) * vdenBE
           cosBE = x21 * vdenBE
           SINA = SG * cosBE + CG * sinBE
           COSA = SG * sinBE + CG * cosBE
*
         U1 = D(I) * SINA
*       U2 = -.5*(E(I)**2-(M(I)-U1)**2)**(-.5)*(-2.*(M(I)-U1)*(-D(I)*COSA))
*                                                          replaced by
           dmu1 = M(I) - U1
           U2 = -.5 / X10th( E(I)**2 - dmu1**2, 5 ) * twod(I) * dmu1 * COSA
*
```

```
      UA = (U1 + U2) / (U1 - U2)
*
      TAND1 = dmu1 / ( slx(I) - D(I)*COSA )
      TAND2 = dmu1 / ( D(I)*COSA + x21 )
      XDHE = -X2DOT(I)
      XHE(I) = S(I) - X2
      SIGN1 = SIGN(1., X2DOT(I) )                                    SSM13200
      Q1 = A1(I) * X1DOT(I)
      Q2 = A2(I)*X2DOT(I) + A4(I)*XDHE
      IF ( XV .LE. TOMIN(I) ) Q1=0.
      IF ( XV .LE. ORFMIN(I) ) Q2=0.
      Q3 = Q2 - Q1
      PR1 = ( KT3(I)*ABS(Q3) + KL3(I) )*Q3 + PR
      C1 = ( KT1(I)*ABS(Q1) + KL1(I) )*(-Q1)
      C2 = ( KT2(I)*ABS(Q2) + KL2(I) )*Q2
      P1 = PR1 - SIGN1*( A1(I)*X1DOT(I)*vG1(I) )**2
      IF( XV .LE. .046) THEN
        P2 = PHE
      ELSE
        P2 = SIGN1*( ( ( A2(I) - A4(I) )*X2DOT(I) )*vGE)**2 + PR1 SSM13300
      ENDIF
      D1 = -TAND1*D(I)*COSA - U1
      vD2 = 1.0 / ( U1 - TAND2*D(I)*COSA )
      FX1 = ( P1 + C1 - PRC )*A1(I) + FRIC1 - BV1*X1DOT(I)
      FX2 = ( TP + FX1*D1 )*vD2
      FY1(I) = -FX1 * TAND1
      FY2(I) = FX2 * TAND2
*
*     VA=(-(A2(I)-A4(I))*((A2(I)-A4(I))/GE)**2+D1/D2*A1(I)*(A1(I)*UA/G1 400
*    X   (I))**2)*SIGN1
*     VB=-BV2-D1/D2*(-BV1*UA)
*     VC=-(PR1+C2)*(A2(I)-A4(I))-PHE*A3(I)+FRIC2-1./D2*(TP(I)+D1*(A1(I)*
*    X   PR1+C1*A1(I)+FRIC1-PRC*A1(I)))+PRC*A2(I)
*
      VA = ( ( A2(I) - A4(I) )**2 * vGE)**2 +
     +           D1*vD2*( A1(I)*( A1(I)*UA*vG1(I) )**2 )*SIGN1       SSM13400
      VB = D1*vD2*BV1*UA - BV2
      VC = FRIC2 - (PR1 + C2)*( A2(I) - A4(I) ) - PHE*A3(I)
     +       - vD2*( TP + D1*( A1(I)*(PR1 + C1 - PRC) + FRIC1 ) )
     +       + PRC*A2(I)
      VRAD = VB**2 - 4.*VA*VC
      IF ( VRAD .GE.0. ) THEN
        X2DOT(I) = ( -VB - SQRT(VRAD) )/(2.*VA)
        DY(I) = X2DOT(I)
        PHIDOT = X2DOT(I) / (U1 - U2) * degprd
        X1DOT(I) = UA * X2DOT(I)                                     SSM13500
        X1(I) = - twod(I)*COSA + S(I) - X2
        THETA(I) = PHID(I)
        DTHETA(I) = PHIDOT
        Y(I) = pruint( DY(I), Tstep, 198 + I )
```

```
          JK(I) = .FALSE.
        ELSE
1500      WRITE( event, 1510) STIME, I
1510      FORMAT(' AT ',F10.4,' VALVE',I3,' MOVING IN WRONG DIRECTION')
        ENDIF
2000  CONTINUE
      RETURN                                                        SSM13600
      END
```

```
      SUBROUTINE POGO0
*****************************************************************
C
C  PURPOSE:  SIMULATE POGO SUPPRESSOR OPERATION
C
C   POGO SUPPRESSOR SYSTEM IS THE "STABILIZER" OF THE OXIDIZER SYSTEM.
C   IT USES THE PRESSURIZED HELIUM SYSTEM TO SUPPLY ENOUGH OXID LINE
C   PRESSURE AT THE BEGINNING OF THE OPERATION.  A BACKFLOW LINE IS ALSO
C   USED TO DRAIN THE OVERFLOW OF THE EXCESS OXIDIZER DURING THE OPEATION.
C
C   POGO SYSTEM HAS THREE DUCTS/VALVES CONNECTED:
C   OI2:             HOPO INPUT DUCT, DOWN SIDE OF THE POGO ACCUMULATOR
C   RIV TUBE:        OVERFLOW TUBE BACK TO OXID TANK
C   DIFFUSER VALVE:  INPUT VALVE FOR EITHER PRESSURIZED HELIUM OR
C                    PRESSURIZED LOX FROM HPOP OUTLET.
C
C******ARGUMENT******
C INPUT:
C  I = INITIALIZATION ARGUMENT inactivated                          SSM80400
C
C******COMMON USAGE******
C INPUT:
C  VARIABLES                                                 SOURCE
C    POI2,POD2,POJ                                           OXIDF
C
C OUTPUT:
C  VARIABLES                                                 DESTINATION
C    DWO,DWGOP,OWHOP                                         OXIDF
C
*****************************************************************
*     DOUBLE PRECISION TIME  replaced by STIME
*
      INCLUDE 'blank.com'
      INCLUDE 'pogo.com'
      INCLUDE 'oxid.com'
*
      PARAMETER ( NPTS = 4, MPTS = 12 )                          SSM81100
      DIMENSION  WGTAB(NPTS), FACTAB(NPTS), PGTAB(MPTS), SATTAB(MPTS),
     +           SATvPG(MPTS), FACvWG(NPTS)
C
      INTEGER Tstep
      PARAMETER ( Tstep = 0, TooBig = 1.E50 )
      PARAMETER ( workc = 12.0 * 778.16 )
*
      DATA WGTAB  /      0.0 ,   0.2 ,   0.3 ,   1.0  /
      DATA FACTAB /      0.0 ,   0.2 ,   0.9 ,   1.0  /             SSM80900
      DATA PGTAB  /    0.022 ,   6.7 ,  20.2 ,  52.6 ,  98.5 , 148.2 ,
     +                 201.7 , 298.2 , 404.2 , 511.5 , 611.9 , 731.38/
      DATA SATTAB /   97.831 , 150.0 , 168.0 , 186.0 , 204.0 , 216.0 ,
     +                 226.0 , 240.0 , 252.0 , 262.0 , 270.0 , 278.24/
```

```
      DATA DTPSTH / 4.0      /
      DATA DTPSTL / 0.0      /
      DATA QINT   / 0.01792  /
      DATA QNCON  / 955.5    /
      DATA QNEXP  / 2.006    /                                        SSM81000
      DATA QSLP   / 0.00491  /
      DATA TGOX   / 650.0    /
      DATA THE    / 520.0    /
      DATA THTHI  / 2.0      /
      DATA THTLO  / 1.4      /
      DATA TLOX   / 180.0    /
*
*  Is Z below a function of the step size? If so, compute it so
*  step size can be changed.
*
      DATA Z      / 0.0005 /
C
      R(X)=1./(772.8*X*X+1.0E-12)
*
      DQHEAT=0.0
      ISAVE = 1
      JSAVE = 1
      TSAT=0.0
      WLOX=0.0                                                       SSM81600
      WORK=0.0
      X=0.0
C
      READ (run,50) ATH, AIN, AHPV, AGC, RGC, RS, RGHS, RREC, ZREC, HP,
     +              PHES, TGC, THEO, RHS, THEC, RGVO, ZCD, RGSL, VTOT,
     +              RHPV, DTPSTL
   50 FORMAT(//2X,6G12.4)                                            SSM81700
*
      P1=POI2
      P2 = POI2
      PD = POI2
*
*     TPRINT    = TPR(1) - 0.1 * DT  replaced by one print interval
*
*    Input echo eliminated.
*
      CALL fgset( 42 )
      CALL fgset( 43 )
*
      DO 60  I = 2, MPTS
        SATvPG(I) = ( SATTAB(I) - SATTAB(I-1) ) /
     +                           ( PGTAB(I) - PGTAB(I-1) )
   60 CONTINUE
      DO 70, I = 2, NPTS
        FACvWG(I) = ( FACTAB(I) - FACTAB(I-1) ) /
     +                            ( WGTAB(I) - WGTAB(I-1) )
```

```
   70 CONTINUE
*
      dwfac2 = 500. * DT
      IF ( dwfac2 .LE. 1.0 ) THEN
        dwfac1 = 1.0 - dwfac2
      ELSE
        PRINT *, "Time step assumed less than 1/500 in pogosup."
        STOP
      END
      tgox5 = 579.5 * TGOX
      rhfac2 = 20. * DT
      rhfac1 = 1. - rhfac2
*
      CALL lmint0( XHPV, 204, 0.0, 100.0 )
      CALL lmint0( XGC, 205, 0.0, 100.0 )
      CALL unint0( UG, 206 )
      CALL lmint0( WLOX, 207, 0.00001, TooBig )
      CALL lmint0( WGOX, 208, 0.00001, TooBig )
      CALL lmint0( WHE, 209, 1.0E-08, TooBig )
      CALL unint0( DWO, 210 )
      CALL unint0( DWGAS, 211 )
      CALL unint0( PD, 212 )
      CALL unint0( DWRE, 213 )
*
*  The following restart section is added to initialization.
*
      vRHOHE = 4636.0 * THE / PHES
      TGAS      = 470.0
      WLOX      = 16.24
      WGOX      = 1225.0 * POI2 / (579.5 * TGAS)
      WHE       = 25.0 * POI2 / (4636.0 * TGAS)
      UG        = TGAS * (0.17811 * WGOX + 0.74824 * WHE)
      RHOREC    = 0.01                                              SSM82000
      IPR       = 2
*
      RETURN
*
      ENTRY POGO
*****************************************************************
C
C     COMPUTE GAS FLOWRATES INTO THE POGO ACCUMULATOR
C
C     *     *     *     COMPUTE VALVE POSITIONS
C
C   VALVE OPENING SCHEDULE TIMES ARE:
C   THEO:   HELIUM VALVE OPENING TIME
C   THEC:   HELIUM VALVE CLOSING TIME
C   TCUT:   SYSTEM POWER CUT DOWN TIME
C   DTPSTL: SYSTEM POWER CUT TO PURGE DELAY
C   DTPSTH: TIME PERIOD OF HELIUM PURGE DURING POWER CUT DOWN
```

```
C    TGC:     LOX VALVE OPENING TIME, THIS WORKS WITH "THEC"
C    XGC:     VALVE OPENING OF LOX FLOW BACK FROM "OD2" DUCT, % OF AREA
C    XHPV:    VALVE OPENING OF PRESSURIZED HELIUM SYSTEM, % OF AREA
C    (NOTE: XGC AND XHPV VALVES OPEN EXCLUSIVELY.)
C
       IF (    STIME .LT. THEO
      +   .OR. ( STIME .GT. THEC .AND. STIME .LT. TCUT + DTPSTL )
      +   .OR. STIME .GT. TCUT + DTPSTH )     THEN                    SSM82100
C
C    TIME PERIODS WHEN HELIUM FLOWS ARE NOT EXISTING, START OR MAIN STAGE
C
           XHPV = prlint( - RHPV, Tstep, 204 )
           IF (XHPV .GT. 0.2)    GO TO 260
           IF (STIME .GT. TGC) THEN
             XGC = prlint( RGC, Tstep, 205 )
           ENDIF
           IF (XGC .GT. 0.01)    GO TO 300
           IF (XHPV .LE. 0.01) THEN
             DWG = 0.0
             DWH = 0.0
             GO TO 400
           ENDIF
         ELSE                                                          SSM82200
C
C    TIME PERIODS WHEN HELIUM FLOW DOMINATE, DURING START OR PURGE
C
           XGC = prlint( - RGC, Tstep, 205 )
           IF (XGC .GT. 0.2) GO TO 300
*
*    Do you really want to skip the integration of XHPV in this case?
*
           XHPV = prlint( RHPV, Tstep, 204 )
         ENDIF
*
C      *      *      *      COMPUTE HELIUM FLOWRATE INTO ACCUMULATOR
C
C    HELIUM FLOW INTO ACCUMULATOR, GOX FLOW MUST BE ZERO (OR ALMOST ZERO).
C    THE FOLLOWING SECTION OF PROGRAM CALCULATE THE HELIUM FLOW BY:
C    1) COMPUTE INITIAL FLOW VALUE WHEN XHPV < 0.2%
C    2) WHEN XHPV > 0.2%, USES THE SMOOTHING FACTOR OF 500*DT=0.1 TO
C       CALCULATE THE NEXT FLOW VALUE: DWH(NEXT)=0.9*DWH(OLD)+0.1*DWH(IN)
C       (500*DT CAN BE CONSIDERED AS THE TIME DELAY OF THE DUCT.)
C    FOLLOWING NOTATIONS ARE USED:
C    PHES:    HELIUM SYSTEM PRESSURE
C    P1:      PRESSURE OF JUNCTION OF HELIUM DUCT AND GOX DUCT
C    P2:      PRESSURE BEFORE THE DIFFUSER
C    PGCO:    PRESSURE AFTER THE GOX VALVE
C    PG:      POGO SYSTEM PRESSURE
C    DWG:     GOX FLOW INTO THE DIFFUSER
C    DWH:     HELIUM FLOW INTO THE DIFFUSER
```

```
C
  260 RAHPV = R(XHPV * AHPV / 100.0)
      IF (XHPV .LT. 0.2) THEN
        DWH = SQRT( ABS(PHES - P1) / ( vRHOHE * RAHPV ) )
      ENDIF
      dwsr = vRHOHE * DWH ** 2
      PHPV = PHES - RHS * dwsr
      P1 = PHPV - RAHPV * dwsr
      IF (P1 .LT. P2)  P1 = P2
      PGCO = P1
      DWG = 0.0
      DWH = dwfac1* DWH +
     +      dwfac2 * AIN * GFLOW(P1, P2, P1*vRHOHE, 1.0, 1.684)      SSM82300
      P2 = PG + RGHS * DWH ** 2 * vRHOHE
      GO TO 400
*
C     *     *     *     COMPUTE GOX FLOW INTO THE ACCUMULATOR
C
C   THIS IS THE FLOW FROM DUCT "OD2" TO THE DIFFUSER IN GAS OXID FORM TO
C   SUPPLY PRESSURE FOR POGO SYSTEM DURING THE MAIN STAGE OPERATION.
C   GAS OXID FLOW INTO ACCUMULATOR, HELIUM FLOW MUST BE ZERO (OR ALMOST ZERO
C   THE FOLLOWING SECTION OF PROGRAM CALCULATE THE HELIUM FLOW BY:
C   1) COMPUTE INITIAL FLOW VALUE WHEN XGC < 0.2%
C   2) WHEN XGC > 0.2%, USES THE SMOOTHING FACTOR OF 500*DT=0.1 TO
C      CALCULATE THE NEXT FLOW VALUE: DWG(NEXT)=0.9*DWG(OLD)+0.1*DWG(IN)
C      (500*DT CAN BE CONSIDERED AS THE TIME DELAY OF THE DUCT.)
C   FOLLOWING NOTATIONS ARE USED:
C   PGCI:   PRESSURE OF INPUT SIDE OF THE RETURING GOX VALVE
C   PGCO:   PRESSURE OF OUTPUT SIDE OF THE RETURING GOX VALVE
C
  300 RAGCV = R(XGC * AGC / 100.0)
      vRHOGO = tgox5 / POD2
      IF (XGC .LT. 0.2) THEN
        DWG = SQRT(ABS(POD2 - P1) / (vRHOGO * RAGCV)  )
      ENDIF
      dwgr = vRHOGO * DWG ** 2
      PGCI = POD2 - RGSL * dwgr                                      SSM82400
      PGCO = PGCI - RAGCV * dwgr
      P1 = PGCO - RGVO * dwgr
      IF (P1 .LT. P2)  P1 = P2
      IF (PGCO .LT. P1)  PGCO = P1
      DWH = 0.0
      DWG = dwfac1 * DWG + dwfac2 *
     +        GFLOW(P1, P2, P1/RHOGOS, 1.0, 1.4) * AIN
      P2 = PG + RGHS * DWG ** 2 * vRHOGO
C
C     COMPUTE PROPERTIES IN THE ACCUMULATOR                          SSM82500
C
C   PROPERTIES IN THE ACCUMULATOR IS DEFINED BY THE FOLLOWING:
C   UG:     TOTAL ENERGY OF THE GAS PHASE IN THE ACCUMULATOR
```

```
C    DQHEAT: HEAT TRANSFERRED BETWEEN LOX AND GAS PHASE
C    WORK:   WORK DONE BY GAS PHASE IN PUSHING LOX (= P * Delta V)
C    DWLOX:  LOX FLOW INTO THE ACCUMULATOR, INCLUDING:
C            DWO:    LOX FLOW FROM DUCT "OI2"
C            DWQNCH: LOX FLOW FROM GAS PHASE DUE TO QUENCHING EFFECT
C            DWLO:   LOX FLOW FROM THE ACCUMULATOR TO OXID TANK
C    DWGOX:  GAS OXID FLOW INTO THE ACCUMULATOR, INCLUDING:
C            DWG:    GAS OXID INPUT FROM "OD2" THROUGH XGC VALVE
C            DWQNCH: GAS FLOW TO LIQUID PHASE DUE TO QUENCHING EFFECT
C            DWGOP:  GAS FLOW TO DUCT "OI2"
C            DWGO:   GAS FLOW TO OXID TANK IN BACK FLOW TUBE
C    DWHE:   HELIUM FLOW INTO THE ACCUMULATOR, INCLUDING:
C            DWH:    HELIUM FLOW FROM HELIUM SUPPLY THROUGH XHPV VALVE
C            DWHOP:  HELIUM FLOW TO DUCT "OI2"
C            DWHO:   HELIUM FLOW TO OXID TANK IN BACK FLOW TUBE
C
  400 DLTUG = 1.24471 * (DWH * THE - (DWHOP + DWHO) * TGAS)
     1  + 0.24017 * (DWG * TGOX - (DWGOP + DWQNCH + DWGO) * TGAS)
     2  +DQHEAT - WORK
      UG = pruint( DLTUG, Tstep, 206 )
      DWLOX = DWO + DWQNCH - DWLO
      WLOX = prlint( DWLOX, Tstep, 207 )
      DWGOX = DWG - DWQNCH - DWGOP - DWGO
      WGOX = prlint( DWGOX, Tstep, 208 )
      DWHE = DWH - DWHOP - DWHO                                    SSM82600
      WHE = prlint( DWHE, Tstep, 209 )
      VL         = WLOX / 0.04061
      VG         = AMAX1(1.0, VTOT - VL)
      TGAS       = UG / (0.17811 * WGOX + 0.74824 * WHE)
      PG         = (579.5 * WGOX + 4636.0 * WHE) * TGAS / VG
      RHOHE      = PG / (4636.0 * TGAS)
      RHOGOX     = PG / (579.5 * TGAS)
C
C     COMPUTE FLOWRATES
C     *     *     *     FLOWRATES AT THE ACCUMULATOR NECK          SSM82700
C
C   THIS SECTION IS TO CALCULATE THE CONDITIONS OF THE LIQUID AND GAS FLOW
C   OF THE POGO SYSTEM.  FOLLOWING NOTATIONS ARE USED:
C   ALIQ:   FRACTION OF THE LOX IN THE DUCT BETWEEN "OI2" AND NECK
C           ALIQ=1 FOR LOX FILLED DUCT, ALIQ=0 FOR GOX FILLED DUCT, AND
C           0<ALIQ<1 FOR PARTIALLY FILLED DUCT
C   FAC:    DIFFUSING FACTOR TO CALCULATE GAS FLOW IN A GAS MIXTURE
C
      ALIQ = AMAX1(1.0E-10, AMIN1(1.0, VL/130.6))
      DWO = prflow(DWO, Z, -RS/(RHOCD2*ALIQ**2), POI2-PG, 210)
*
*  The translation below is accurate, but the intent is doubted.
*
      IF (VL .GE. VTOT - 1.0 .AND. DWO .GE. 0.0) THEN
        CALL unint0( 0., 210 )
```

```
        DWO = 0.0
        PG = POI2
      ENDIF
  500 IF (ALIQ .GE. 0.9999) THEN
        DWGOP     = 0.0
        DWHOP     = 0.0
      ELSE
C
C   THE CONDITION OF PARTIALLY FILLED DUCT BETWEEN "OI2" AND POGO NECK.
C   DWGAS:  TOTAL GAS FLOW (HELIUM AND GOX)
C
        wgx = WGOX/AMAX1(1.0E-12, WGOX+WHE)
        CALL intval( ISAVE, wgx, NPTS, WGTAB,
     +     'Below WG table in pogosup.',
     +     'Above WG table in pogosup.', 0 )
        FAC = xlint( wgx , NPTS, WGTAB, FACTAB, FACvWG, ISAVE )      SSM82800
*
        RHOG = PG / (TGAS * (579.5 * FAC + 4636.0 * (1.0 - FAC)))
        DWGAS = prflow(DWGAS, Z, RS/(RHOG*(1.0-ALIQ)**2), PG-POI2, 211)
        IF (WHE .GE. 1.0E-07) THEN
          DWGOP = FAC * DWGAS
          DWHOP = (1.0 - FAC) * DWGAS
        ELSE
          DWGOP = DWGAS
          DWHOP = 0.0
        ENDIF
      ENDIF
*
C     *     *     *     COMPUTE THE FLOWRATES OUT THE RIV TUBE       SSM82900
C
C   BACK FLOW TUBE CONDITIONS:
C   HP:     LOX HEIGHT IN POGO ACCUMULATOR
C   AREAH:  LOX AREA INSIDE BACK FLOW TUBE (RIV TUBE)
C   DWDUM:  GAS FLOW OF RIV TUBE, (WHEN AREAH < ATH)
C   PD:     PRESSURE OF THE DUCT OF RETURNING LOX, BEFORE CHECK VALVE
C   DWRE:   RECIRCULATING FLOW INTO OXID TANK
C
  700 HP = FGEN(42, 119, VL)
      AREAH = FGEN(43, 120, HP)
      DWDUM = GFLOW(PG, PD, PG*VG/(WHE+WGOX), 1.0,
     1   (1.684*WHE+1.4*WGOX)/(WHE+WGOX)) * AMAX1(0.0, ATH-AREAH)
      IF (WHE .GE. 1.0E-07) THEN
        DWGO = DWDUM * WGOX / (WHE + WGOX)
        DWHO = DWDUM - DWGO
      ELSE
        DWGO = DWDUM
        DWHO = 0.0                                                   SSM83000
      ENDIF
  800 DWLO = SIGN(SQRT(ABS(PG-PD) * RHOCD2 / R(AREAH)), PG-PD)
      PD = pruint( (DWDUM + DWLO - DWRE) / ZCD, Tstep, 212 )
```

```
C
C    CALCULATING RECIRCULATING DUCT DENSITY USING TIME AVERAGE 20*DT(=.004)
C    LOOKS LIKE THE DUCT IS VERY HUGE AND TAKES LONG TIME TO FILL
C
       RHOREC = rhfac1 * RHOREC + rhfac2 *
      1     (ABS(DWLO + DWGO + DWHO) / AMAX1(ABS(DWLO / RHOCD2
      2   + DWGO / RHOGOX + DWHO / RHOHE), 0.00001))
       DWRE = prflow(DWRE, ZREC, -RREC/RHOREC, PD-POJ, 213)
C
C      *      *      *      COMPUTE HEAT TRANSFER AND LOX QUENCHING
C
C    DQHEAT IS THE HEAT FLOW FROM LOX TO GAS.  AS TO WHY IT IS A FUNCTION OF
C    IS UNKNOWN.
C
*      DQHEAT     = (QSLP * AMAX1(DWO, 0.0) + QINT) * (TLOX - TGAS)
*      IF (TIME .GT. THTHI)                                    GO TO 860
*      STIME=TIME                                                      SSM83100
*      DQHEAT     = DQHEAT * AMAX1 (0.0,(STIME-THTLO) / (THTHI-THTLO))
*  860 CONTINUE       sequence replaced by
*
       IF ( STIME .LE. THTHI ) THEN
         DQHEAT = (QSLP * AMAX1(DWO, 0.0) + QINT) * (TLOX - TGAS)
       ELSE
         DQHEAT = DQHEAT * AMAX1 (0.0,(STIME-THTLO) / (THTHI-THTLO))
       ENDIF
C
C            TSAT:    THE SATURATING TEMPERATURE UNDER THE PRESSURE PG.
C
       CALL intval( JSAVE, PG, MPTS, PGTAB,
      +   'The pressure PG is below SATTAB in pogosup.',
      +   'The pressure PG is above SATTAB in pogosup.', 0 )
       TSAT = xlint( PG, MPTS, PGTAB, SATTAB, SATvPG, JSAVE )
*
       DWQNCH = WGOX * QNCON *
      +                   XtoY( AMAX1(1.0, TGAS-TSAT), - QNEXP )
       WORK = -PG * (DWO + DWQNCH - DWLO) / (workc * RHOCD2)
C
C      Eliminated PRINT OUT DATA
C
       END
```

```
   'valdym.for':

      SUBROUTINE VLDYM0
**********************************************************************
C
C  PURPOSE:  COMPUTE VALVE DYNAMICS
C
C   THIS SUBROUTINE SIMULATE THEDYNAMICS OF THE VALVE SERVO SYSTEM AND
C   VALVE MOTION INCLUDING STICTIONS AND BACKLASHES.  THE INPUTS ARE:
C   XCxyzV: POSITION COMMAND TO MOVE VALVE xyz.
C   THETA(): CURRENT VALVE POSITION, DEG
C   DTHETA(): CURRENT VALVE VELOCITY, DEG/SEC
C
C******ARGUMENTS******
C  N = INITIALIZATION FLAG  eliminated
C                                                                  SSM83500
C******COMMON USAGE******
C  INPUT:
C   VARIABLE                                          SOURCE
C    XCFPOV,XCOPOV,XCMOV,XCMFV,XCCCV                  CNTROL
C    THETA,DTHETA                                     EMCO
C
C  OUTPUT:
C   VARIABLE                                          DESTINATION
C    RFPOV,ROPOV                                      HOTGAS
C    RMOV                                             OXIDF        SSM83600
C    RMFV,RCCV                                        FUELF
C    XFPOV,XOPOV,XMOV,XMFV,XCCV                       EMCO
C    XOPOV                                            CNTROL
C
*  Double precision is probably unnecessary with double precision
*  accumulators in the integration routines. It should be restored only
*  on evidence of actual accuracy problems outside of integration.
C
*     DOUBLE PRECISION EVP, DDESA, DDESV, DVR,  DELTA, ALIM, Q
*     DOUBLE PRECISION DTHETA, ESAC, ESA, DESA, ESV, DESV
*     DOUBLE PRECISION TIME
C                                                                  SSM83700
      DIMENSION DPPC(5),THETAC(5),CL(5),CSV(5),A(5)
      DIMENSION THETHY(5), THETST(5), DTHET1(5), DTHET2(5)
      DIMENSION WINDUP(5), THETSK(5), THETBL(5), THETWU(5), THETMAX(5)
      REAL CvA(5), vdppc(5)
C
      INCLUDE 'blank.com'
      INCLUDE 'out.com'
      INCLUDE 'units.com'
      INCLUDE 'contrl.com'
      INCLUDE 'valves.com'
*
```

```
      SAVE
C
C   WINDUP IS ADDED TO THE SIMULATION IN DATA FORMAT
C   THETMAX IS THE MAXIMUM ACTUATOR STROKE LIMIT
C
      DATA WINDUP/0.091, 0.115, 0.1, 0.141, 0.067/
      DATA THETMAX/79.0, 79.0, 84.25, 84.25, 80.0/
C
*     DATA SLOPF /0.0/ added to input parameters
*
*   Please rename this function.
*
      Rename(Z) = 1.0 / (772.8 * Z**2 + 1.0E-12 )
*
*  ALIM biases the magnitude of Q down minutely. IF Q is small enough,
*  it is replaced by 0.D0. It probably had been intended to function
*  differently. As it stands, ALIM was not considered of value and
*  was discarded. The reimplementer should note that double precision
*  outside the integrators has been eliminated.
*
*     ALIM(Q) = DMAX1(0.D0, DABS(Q) - 1.D-20) * DSIGN(1.D0,Q)
C
      rlimit(floor, ceiling, x ) = AMAX1( floor, AMIN1( ceiling, x ) )
*
*****************************************************************************
*     IF(FLAG.EQ.15.)GO TO 9999  is obsolete
*
* Unnecessary initializations were eliminated
*
      READ( run, '(//2X, 6G12.4 )' )
     +    CA, WA, SA, CSV, WSV, SSV, A, CL, CRS, TRS, CLS,
     +    TLS, CMF, TMF, CM, WM, SM, CRVDT, CLVDT,
     +    ( THETHY(I), THETST(I), DPPC(I), I = 1, 5),
*
*  The following input parameters were transferred from NAMELIST input
*
     +   AR1, AR2, TL, TH, XOMAX,                                    SSM84100
     +   OPLEAK, OPX1, OPX2, FPLEAK, FPX1, FPX2, SLOPF,
     +   IBKLASH, IWUSTN,
*
*      DELTA was made an input parameter.
*
     +   DELTA
*
*   Functions of DELTA for Euler and AB2 integrators
*
      h(1) = DELTA
*
      DO 30 I=1,5
        DESA(I)=0.0
```

```
          ESA(I)=0.0
          DESV(I)=0.0
          ESV(I)=0.0
          THETA(1)=DPPC(1)*XCFPOV
          THETA(2)=DPPC(2)*XCOPOV
          THETA(3)=DPPC(3)*XCMOV
          THETA(4)=DPPC(4)*XCMFV
          THETA(5)=DPPC(5)*XCCCV                                    SSM84900
          VS(I)=0.0
          EMF(I)=0.0
          DVM(I)=0.0
          VM(I)=0.0
          THET1L(I)=0.0
          THET2L(I)=0.0
          THETA1(I)=0.0
          THETA2(I)=0.0
*         ISTIC(I)=1
*         IHYS(I)=1                                                 SSM85000
          DTHETL(I)=0.01
          ESAC(I)=0.0
          VR(I)=0.0
          CvA(I) = CL(I) * CSV(I) / A(I)
          vdppc(I) = 1. / DPPC(I)
   30 CONTINUE
      fbklsh = FLOAT(IBKLASH)
      fwustn = FLOAT(IWUSTN)
      LOOP=(DT+0.00001)/DELTA
      twosa = 2. * SA
      twossv = 2. * SSV
      vTRS = 1. / TRS
      crvs = CRVDT * CRS
      vfpx = 1. / (FPX2 - FPX1)
      abfp = ABFPO * .01
      vopx = 1.0 / (OPX2 - OPX1)
      abopf = ABOPO * 0.01
      abmov = ABMOV * 0.01
      abmfv = ABMFV * 0.01
      abccv = ABCCV * 0.01
      RFBV = 1.0E+12
*
C      DELTA=0.0001                                                 SSM85040
C
C   DELTA IS CHANGED TO 0.00002 SECOND TO BETTER SIMULATE THE ANALOG SERVO
C   OF THE VALVE DYNAMIC.  THIS IS NECESSARY WHEN APPLYING STEP INPUTS TO
C   THE VALVE OPENINGS.
C
*     DELTA=0.00002  DELTA was made an input parameter.
*
*  Initialize integrators:
*
```

```
      CALL unint0( DESA(I), 147 + I )
      CALL lmint0( ESA(I), 152 + I, -20.0, 20.0 )
      CALL unint0( DESV(I), 157 + I )
      CALL lmint0( ESA(I), 162 + I, -20.0, 20.0 )
      CALL lmint0( THETA(I), 167 + I, 0.0, THETAMAX(I) )
      CALL unint0( VR(I), 172 + I )
*
      GO TO 310
*
      ENTRY VALDYM
*************************************************************************
C
C                SERVO CALCULATIONS
C
  200 CONTINUE
      THETAC(1)=DPPC(1)*XCFPOV                                        SSM85100
      THETAC(2)=DPPC(2)*XCOPOV
      IF ( STIME .GE. TL .AND. STIME .LE. TH) THETAC(2)=DPPC(2)*1000.
      THETAC(3)=DPPC(3)*XCMOV
      THETAC(4)=DPPC(4)*XCMFV
      THETAC(5)=DPPC(5)*XCCCV
C
C   SCALING FOR BACKLASH, WINDUP, AND STICTION FROM % OPENING TO DEGREE
C
      DO 205 I=1,5
        THETBL(I) = THETHY(I) * DPPC(I) * fbklsh
        THETWU(I) = WINDUP(I) * DPPC(I) * fwustn
        THETSK(I) = THETST(I) * DPPC(I) * fwustn
 205  CONTINUE
      DO 300 I=1,5
        IF( TPA.GT.0.0 .AND. STIME.GE.TPA) THEN
          DO 210 J=1,LOOP
            EVP = THETAC(I)*CRVDT - VR(I)                             SSM85200
*
*           DDESA=CA*WA**2*EVP-WA**2*ESAC(I)-2.0*SA*WA*DESA(I) replaced by
            DDESA = ( ( CA*EVP - ESAC(I) ) * WA - twosa * DESA(I) ) * WA
*
            DDESV = ( ( ESA(I) - ESV(I) ) * WSV - twossv*DESV(I) ) * WSV
            DTHETA(I) = CvA(I) * ESV(I)
*                                          CvA(I) is CL(I)*CSV(I)/A(I)
*           DVR=THETA(I)*CRVDT*CRS/TRS-VR(I)/TRS replaced by
            DVR = ( THETA(I) * crvs - VR(I) ) * vTRS

C
C   THE POWER AMPLIFIER LIMIT SHOULD BE PUT AT THE OUTPUT END NOT IN
C   THE MIDDLE.
C
C       IF(ESA(I).GT.23..AND.DESA(I).GT.0.) DESA(I)=0.0
C       IF(ESA(I).LT.-23..AND.DESA(I).LT.0.) DESA(I)=0.0
*
```

```
*    The statements above illustrate how to limit the internal double
*    precision accumulator of an integrator, by zeroing the input rate.
*    The effect of the commenting out is to substitute an unlimited
*    accumulator, for the limited one, with limits applied to the
*    output of the integration. This is the approach used throughout
*    other modules of the simulation.
*
*      ESAC(I)=ESAC(I)+ALIM(DESA(I))*DELTA                          SSM85300
*      ESA(I)=ESAC(I)
*
*    Without arguments to the contrary, it appears that the integration of
*    of ESA and ESV with old rates must be in error. Accordingly, the
*    higher order integrations are done first.
C
C    LIMITS ON INPUT AMPLIFIER ARE +-23 VOLTS.  AND LIMITS ON SERVO AMPLIFIER
C    ARE +-20 VOLTS.
C
*      IF(ESA(I).GT.23.) ESA(I)=23.
*      IF(ESA(I).LT.-23.) ESA(I)=-23.
*      IF(ABS(ESA(I)).LT.0.25) ESA(I)=0.0   replaced by
*
            DESA(I) = pruint( DDESA, 1, 147 + I )
            ESA(I) = prlint( DESA(I), 1, 152 + I )
            IF ( ABS(ESA(I)) .LT. 0.25) THEN
              ESA(I) = 0.0
              CALL lmint0( 0.0, 152 + I, -20., 20. )
            ENDIF
*
*     DESA(I)=DESA(I)+ALIM(DDESA)*DELTA  was moved ahead of ESA integration
*
C      IF(ESV(I).GT.20..AND.DESV(I).GT.0.0) DESV(I)=0.0
C      IF(ESV(I).LT.-20..AND.DESV(I).LT.0.0) DESV(I)=0.0
*
*      ESV(I)=ESV(I)+ALIM(DESV(I))*DELTA
*      IF(ESV(I).GT.20.) ESV(I)=20.
*      IF(ESV(I).LT.-20.) ESV(I)=-20.
*
            DESV(I) = pruint( DDESV, 1, 157 + I )
            ESV(I) = prlint( DESV(I), 1, 162 + I )
*
*      DESV(I)=DESV(I)+ALIM(DDESV)*DELTA
*      THETA(I)=THETA(I)+ALIM(DTHETA(I))*DELTA
*
            CALL intgrl( THETA(I), DTHETA(I), DELTA, 167 + I )
C
C    ACTUATOR STROKES ARE LIMITED TO THE MAXIMUM THETA
C
            THETA(I) = rlimit( 0.0, THETMAX(I), THETA(I) )
C
C      VS(I)=VS(I)+ALIM(DVS)*DELTA
```

```
C      IF(EMF(I).GT.12.5.AND.DEMF.GT.0.0) DEMF=0.0                    SSM85400
C      IF(EMF(I).LT.-12.5.AND.DEMF.LT.0.0) DEMF=0.0
C      EMF(I)=EMF(I)+ALIM(DEMF)*DELTA
C      VM(I)=VM(I)+ALIM(DVM(I))*DELTA
C      DVM(I)=DVM(I)+ALIM(DDVM)*DELTA
*      VR(I)=VR(I)+ALIM(DVR)*DELTA
*
            VR(I) = pruint( DVR, 1, 172 + I )
  210     CONTINUE
        ENDIF
C
C   NEW PROGRAM
C
C   BACKLASH IS DEFINED AS THE AMOUNT OF ACTUATOR OUTPUT SHAFT TRAVEL
C   REQUIRED TO REVERSE DIRECTION OF VALVE BALL MOTION UNDER CONDITIONS
C   OF ZERO LINKAGE WINDUP AND TORQUE LOADING.  THE VALUES USED HERE IS
C   HALF OF THE AMOUNT OF THE TOTAL TRAVELING, WELL HALF ON EACH SIDE.
C
        IF ( I .EQ. 3 .AND.
     +       THETA(I) .GT. 33.8 .AND. THETA(I) .LT. 84.0 ) THEN
          FAC = 0.0
        ELSE IF( I .EQ. 4 .AND.
     +       THETA(I) .GT. 38.3 .AND. THETA(I) .LT. 75.8 ) THEN
          FAC=0.0
        ELSE
          FAC = 1.0
        ENDIF
        IF ( ABS(THETA(I) - THETA1(I) ) .LT. THETBL(I)*FAC ) THEN
          DTHET1(I) = 0.0
          THET1L(I) = THETA1(I)
        ELSE IF (ABS(DTHETA(I)).LT.1.0E-06) THEN
          DTHET1(I) = 0.0
          IF( THETA1(I) .LE.THETA(I)) THEN
            THETA1(I) = THETA(I) - THETBL(I)*FAC
          ELSE
            THETA1(I) = THETA(I) + THETBL(I)*FAC
          ENDIF
        ELSE
          IF (THETA1(I).LE.THETA(I)) THEN
            THETA1(I) = THETA(I) - THETBL(I)*FAC
          ELSE
            THETA1(I) = THETA(I) + THETBL(I)*FAC
          ENDIF
          DTHET1(I) = DTHETA(I)
        ENDIF
C
C   WINDUP AND STICTION SIMULATION:
C   WINDUP IS DEFINED AS THE AMOUNT OF ACTUATOR OUTPUT SHAFT TRAVEL REQUIRED
C   AFTER LINKAGE BACKLASH HAS BEEN ABSORBED TO INITIATE VALVE MOTION.
C   STICTION IS DEFINED AS THE AMOUNT OF VALVE BALL OVERTRAVEL RESULTING
```

```
C   FROM A CHANGE IN LINKAGE WINDUP DURING THE TRANSITION FROM A STATE OF
C   STATIC FRICTIONAL RESISTANCE TO SLIDING FRICTIONAL RESISTANCE AT
C   START OF BALL MOTION.
C
        IF( ABS(DTHET1(I)) .LT. 1.0E-06) DTHET2(I)=0.0
        IF( ABS(DTHET2(I)) .LT. 1.0E-06 .OR.
     +  ABS(THETA1(I) - THETA2(I)) .LE. (THETWU(I) - THETSK(I)) ) THEN
          IF(ABS(THETA1(I)-THETA2(I)).LE.THETWU(I)) THEN
            DTHET2(I)=0.0
          ELSE
            DTHET2(I) = DTHET1(I)
          ENDIF
          IF ( THETA2(I) .GT. THETA1(I) ) THEN
            THETA2(I) = THETA1(I) + THETWU(I) - THETSK(I)
          ELSE
            THETA2(I) = THETA1(I) - THETWU(I) + THETSK(I)
          ENDIF
        ELSE
          DTHET2(I) = DTHET1(I)
          IF ( THETA2(I) .GT. THETA1(I) ) THEN
            THETA2(I) = THETA1(I) + THETWU(I) - THETSK(I)
          ELSE
            THETA2(I) = THETA1(I) - THETWU(I) + THETSK(I)
          ENDIF
        ENDIF
C
  300 CONTINUE                                                    SSM85800
C
C   THIS SECTION OF CODE USES THETA2(I) AS THE ACTUAL VALVE OPENINGS TO
C   FIND THE VALVE OPENING AREAS AND VALVE RESISTANCES.
C   THE LEAK (FPLEAK AND OPLEAK) PHENOMENA DEFINED IN THE FOLLOWING IS NOT
C   CLEAR.  HOWEVER, BOTH OPLEAK AND FPLEAK ARE SET TO 0.0 IN INPUT DATA.
C
C                FUEL PREBURNER OXIDIZER VALVE
C
*                Continuation point for initialization
  310 CONTINUE
      IF ( SLOPF .GT. .01 ) THEN
        XFPOV = XCFPOV
      ELSE
        XFPOV = rlimit( 0.0, 100.0, THETA2(1)* vdppc(1) )
      ENDIF
      AFPOV = FGEN(18, 99, XFPOV)
     +          + FPLEAK * rlimit( 0.0, abfp, (XFPOV - FPX1) * vfpx )
      RFPOV = Rename(AFPOV)
C
C                OXIDIZER PREBURNER OXIDIZER VALVE                  SSM85900
C
      IF ( SLOPF.GT..01 ) THEN
        XOPOV = XCOPOV
```

```
       ELSE IF( STIME .GE. TL .AND. STIME .LE. TH ) THEN
         XOPOV = XOPOV + 210. * DT
C
C    XOPOV HAS A MAXIMUM INCREASING RATE OF 210%/SEC AND MAXIMUM DECREASING
C    RETE OF 270%/SEC.                    not percents
C
       ELSE
*
*    XOPOV follows THETA2(2)/DPPC(2) perfectly between limits imposed by
*         max rates and fixed limits 0. and 100. Otherwise XOPOV
*         increases at the maximum rate. There is no need to use a higher
*         order integrator here, because at the constant limit rate,
*         Euler's is exact.
*
         XOPOV = AMIN1( 100., XOPOV + 210.*DT,
     +             AMAX1( 0., XOPOV - 270.*DT, THETA2(2)*vdppc(2) ) )
       ENDIF
       XOPOV = AMIN1( XOMAX, XOPOV )
*
       AOPOV = FGEN(17, 100, XOPOV) + OPLEAK *
     +   rlimit( 0., abopf, ( XOPOV - OPX1) * vopx )
       ROPOV = Rename(AOPOV)                                          SSM86000
C
C                 MAIN OXIDIZER VALVE
C
       XMOV = THETA2(3) * vdppc(3)
       AMOV = FGEN(6, 101, XMOV) * abmov
       RMOV = Rename(AMOV)
C
C                 MAIN FUEL VALVE
C
       XMFV = THETA2(4) * vdppc(4)                                    SSM86100
       AMFV = FGEN(31, 102, XMFV) * abmfv
       RMFV = Rename(AMFV)
C
C                 COOLANT CONTROL VALVE
C
       XCCV = THETA2(5) * vdppc(5)
       ACCV  = FGEN(19, 103, XCCV) * abccv
       RCCV = Rename(ACCV)
C
C                 FUEL BLEED VALVE                                   SSM86200
C
C      IF(TIME.LE.TFBV) GO TO 400
C      RFBV=AMIN1(1.0E+12,0.012/(AMIN1(20.0*(STIME-TFBV),1.0)))
C      GO TO 410
*  410 CONTINUE
       RETURN
       END
```

'cstar.for':

```
      FUNCTION CSTAR0(EMR,P)                                          SSM36700
**********************************************************************
C
C  PURPOSE:  INTERPOLATE C* FROM TABLE VS PRESSURE AND MIXTURE RATIO
C
C******ARGUMENTS******
C  INPUT:
C   N     = INITIALIZATION FLAG  inactivated
C   EMR   = MIXTURE RATIO
C   P     = PRESSURE, PSI
C
C  OUTPUT:                                                          SSM36800
C   CSTAR = CHARACTERISTIC VELOCITY, FT/SEC
C
**********************************************************************
      DIMENSION CS(12,9), PR(12)
      REAL MR(9), scspr(12), vdmr(9)
      INCLUDE 'units.com'
*
      READ(run,10)NPR,NMR
      READ(run,11) (PR(I),I=1,NPR),(MR(I),I=1,NMR)
   10 FORMAT(//2X,2I12)
   11 format(//2X,6G12.4)
      READ(run,12)((CS(I,J),I=1,NPR),J=1,NMR)                         SSM36900
   12 FORMAT(2X,12F6.0)
      CALL xyset( NPR, PR, NMR, MR, CS, scspr, vdmr )
      RETURN
      FUNCTION xylint( x, y, nx, xp, ny, yp, sx, vdy, table,
     +                 itop, jtop )
      SUBROUTINE xyset( nx, xp, ny, yp, table, sx, vdy )
*
      ENTRY CSTAR( EMR, P )
**********************************************************************
      CALL intval( I1, EMR, NPR, PR,
     +   'EMR below range of PR in CSTAR.',
     +   'EMR above range of PR in CSTAR.', 0 )
*
      CALL intval( J1, P, MPR, MR,
     +   'P below range of MR in CSTAR.',
     +   'P above range of MR in CSTAR.', 0 )
*
      CSTAR = xylint( EMR, P, NPR, PR, NMR, MR, scspr, vdmr, CS, I1, J1)
      END
```

'qflux.for':

```
C
      FUNCTION H2SATH(P)
*****************************************************************************
C
C  PURPOSE:  CALCULATE HYDROGEN SATURATION ENTHALPY                    SSM24300
C
C******ARGUMENTS******
C  INPUT:
C   P      : PRESSURE, PSI
C  OUTPUT:
C   H2SATH : SATURATION ENTHALPY, BTU/LB
C
C       ENTER PRESSURE,P IN PSIA                                       SSM24400
C       RETURN SATURATED VAPOR ENTHALPY IN BTU/LB (NBS REF H + 200)
C
*****************************************************************************
      DIMENSION H(10),DH(10)
C
      DATA H / 275.,283.8,287.3,287.3,285.3,281.7,276.4,269.5,260.,243./
      DATA DH / .44,.175,0.,-.1,-.18,-.265,-.345,-.475,-.85,-3.  /
C
*****************************************************************************
*
      IF( P .LT. 187.5) THEN
         H2SATH = 0.
      ELSE                                                             SSM24500
*
*        I = INT(P/10.) / 2 + 1
*        H2SATH = H(I) + DH(I)*(P-20*I+20)  was replaced by
*
         H2SATH = H(I) + DH(I) * (P - 20.0 * AINT( P * 0.05 ) )
      END IF
      END
C
      FUNCTION QFLUX(TW,TF,P,HF)                                       SSM24700
*****************************************************************************
C
C  PURPOSE:  CALCULATE HYDROGEN BOILING HEAT FLUX
C
C******ARGUMENTS******                                                  SSM2
C  INPUT:
C   TW    = WALL TEMPERATURE, DEG R
C   TF    = FLUID TEMPERATURE, DEG R
C   P     = FLUID PRESSURE, PSI
C   HF    = FLUID ENTHALPY, BTU/LB
C
C  OUTPUT:
```

```
C    QFLUX = HEAT FLUX TO FLUID, BTU/IN2-SEC
C
C      CALCS. HEAT FLUX FOR NUCLEATE, TRANSITION AND FILM BOILING REGIONS
C      BASED ON DATA FROM R-5598, FIG.26, PG. 90
C
C      THE PARAMETER VALUES USED HERE IS SLIGHTLY DIFFERENT FROM THE ONES
C      GIVEN IN THE DOCUMENT PAGE 32 AND PAGE 33.
C      I ASSUME THAT THIS IS NOT TOO IMPORTANT BECAUSE THAT THE HEAT TRAN
C      IS ONLY A SMALL FACTION OF THE ENERGY FLOW OF THE SYSTEM.
C
*     *  The replacement code assumes that the limits are intended, and
*   avoids repeatedly taking logarithms of limiting constants.
**********************************************************************
*
      rlimit(floor, ceiling, x) = AMAX1( floor, AMIN1( ceiling, x) )
**********************************************************************
*
      HS = H2SATH(P)
      IF ( TW .LT. 80.0 .OR. HF .GT. HS) THEN
         QFLUX = 0.0
      ELSE
*        X=ALOG10(AMAX1(0.01,TW-TF))
*        Y=AMAX1((2.0+4.0*X-2.0*X**2)*AMIN1(1.0,AMAX1(0.0,(100.0-TF)/40.0))
*    *     ,2.1 + X )
*        QFLUX=10.0**(AMAX1(0.01,AMIN1(5.0,Y)))/5.184E+05
*
         test = TW - TF
         IF ( test .GT. 0.01 ) THEN
           X = ALOG10( test )
           Y = AMAX1( 2.1 + X, 0.05 * (1.0 + X * (2.0 - X) ) *
     +                               rlimit( 0.0, 40.0, 100.0 - TF ) )
           QFLUX = 1.0 /
     +               XntoYn( 0.1, rlimit( 0.01, 5.0, Y ) * 1.929E-6 )
         ELSE
           QFLUX = 1.0
         ENDIF                                                    SSM24800
       ENDIF
       END
C
C    THE SETPT( ) FUNCTION IS NOT USED
C
```

'h2vp.for':

```
      FUNCTION H2VP(T)
**********************************************************************
C
C  PURPOSE:  CALCULATE HYDROGEN SATURATION PRESSURE
C
C   IT IS USED FOR CALCULATING CAVITATION OF HYDROGEN PUMPS.
```

```
C    FOR A GIVEN TEMPERATURE, IF THELIQUID HYDROGEN PRESSURE
C    IS LESS THAN THE VAPOR PRESSURE AT CERTAIN POINT THEN BUBBLES
C    WILL START TO GROW.  THIS HAPPENS ESPECIALLY ON THE FOLLOWING
C    EDGE OF THE PUMP BLADE WHERE THE PRESSURE IS LOWEST.
C
C******ARGUMENTS******
C   INPUT:
C    T     = TEMPERATURE, DEG R
C
C   OUTPUT:
C    H2VP = SATURATION PRESSURE, PSI                                    SSM24100
C
*   H2VP can be speeded up considerably by using an fgen linear
*   approximation. One function can be used for all three segments,
*   eliminating the tests for segment boundaries 29.0 and 32.976
*
************************************************************************
*
      DATA CA,        CB,        CC,       CD,          CE        /
     +   2.0062, -50.09708, 1.0044, 1.748495E-2, 1.317E-03    /,
     +     CF,        CG     /
     + -5.926E-5, 3.913E-6  /
*
*     TK=T/1.8  replaced by
      TK = T * .555555
*
      IF( TK. LT. 32.976 ) THEN
         ALOGPV = CA + CB/(TK + CC) + CD * TK
         H2VP = XtoY( 10.0, ALOGPV )
         IF(TK .GT. 29.0) THEN
           tk29 = TK - 29.0
           tk29sq = tk29 ** 2
           H2VP = H2VP + tk29 *
     +           ( tk29sq * (CE + tk29sq * (CF + tk29sq * CG ) ) )  SSM24200
*                          is faster than
*          PVAPOR=PVAPOR+CE*(TK-29.0)**3+CF*(TK-29.0)**5+CG*(TK-29.0)**7)
*
        ELSE
           H2VP = H2VP * 14.696
        END IF
      ELSE
        H2VP = 0.
      END IF
      END

    'oprime.for:

      SUBROUTINE OPRIM0
*   Initializes integrators for OPRIME
************************************************************************
```

```
C
      INTEGER Tstep
      PARAMETER (NVOL=4, MAXN=6, MAXC=5)                                SSM50400
      PARAMETER (MAXIT=30, TOL= .0005)
      PARAMETER ( Tstep = 0, TooBig = 1.E20 )
      LOGICAL FLAG, prpOK
C
      DIMENSION P(MAXN), RHO(MAXN), RHOP(MAXN), U(MAXN), UP(MAXN),
     $          H(MAXN), T(MAXN),   TW(MAXN),   D(MAXN),AHT(MAXN),
     $          W(MAXN), VOL(MAXN), QC(MAXN), QF(MAXN), QB(MAXN),
     $          QOUT(MAXN)                                              SSM50500
      DIMENSION DW(MAXC), DWP(MAXC), R(MAXC), Z(MAXC)
C
*  Keep all local variables between calls
*
      SAVE
*
      DATA RHOV  /9.693E-4/
      DATA RHOVAL/9.693E-4/
      DATA RHOP3 /.03944  /
      DATA RHOSL /.0365   /
      DATA   HSL / -39.1  /                                             SSM50600
      DATA    HV /  39.4  /
      DATA    HL / -50.0  /
      DATA RLINE /6.331E-4/
      DATA ZLINE / .02    /
      DATA XLINE /  2563. /
      DATA WPOVI / -2.2   /
      DATA WPOV  / -2.2   /
      DATA WPOVP /  0.0   /
      DATA FLAG  /  .TRUE.  /
      DATA DWLINE/  0.0   /                                             SSM50700
      DATA DWLINP/  0.0   /
      DATA    P1 /  103.2 /
C
      DATA P /MAXN*14.7 /, RHO /MAXN*4.69E-5/,  RHOP /MAXN*4.69E-5/,
     $     H /MAXN*117.4 /,  U /MAXN*83.6  /,     UP /MAXN*83.6 /,
     $     T /MAXN*540.  /, TW /MAXN*540.  /,     QC /MAXN*0.0/,
     $     QF/MAXN*0.0  /,  QB /MAXN*0.0   /,   QOUT /MAXN*0.0/,
     $     DW/MAXC*0.0  /, DWP /MAXC*0.0   /
C
      DATA D  /0.0,  2*1.40,          1.45,  1.75,  0.  /               SSM50800
      DATA AHT/0.0,  40.25,   30.25, 155.9, 198.,  0.  /
      DATA W  /0.0,  1.13,    .85,   4.4,   5.5,   0.  /
      DATA VOL/0.0,  2*12.35,         29.3,  27.8,  0.  /
C
      DATA R  /0.0,  2*.5E-4, 1.E-4, -78.0E-4 /
      DATA Z /.020,  2*.0065, .0080,   .013   /
C
*  Statement function UNEWF deleted. It is only used once.
```

```
C
*     CHGF(CHG,XNEW,OLD)=AMAX1(CHG,ABS( XNEW-OLD )/(ABS(OLD)+.1    ))
*     replaced by relaxation system.
C
*     OXHPF(RH )=AMAX1(.0149,.0149+.013*(RH -5.E-4)/.0389)  replaced by
      OXHPF(RH )=AMAX1(.0149,.0149 + .3342*(RH - 5.E-4) )
C
**********************************************************************
*
      CALL lmint0( WPOV, 178, -TooBig, .05 )
      CALL unint0( WPOVP, 179 )
      CALL lmint0( P(1), 184, 37., TooBig )
      CALL unint0( DWLINE, 185 )
      CALL unint0( DW(1), 186 )
      DO 10 I = 2, MAXN-1
        CALL unint0( TW(I), 178+I )
        CALL unint0( DW(I), 185+I )
        CALL unint0( RHOP(I), 189+I )
        CALL unint0( RHO(I) * (1. + U(I) ), 193+I )
   10 CONTINUE
      RETURN
*
      ENTRY OPRIME(P0,PC,RVALVE,DWINJ,DWVALV)
**********************************************************************
C   THIS SUBROUTINE IS TO CALCULATE THE PRIME CHARACTERISTICS OF
C   THE FPOV AND OXID INJECTOR FOR FUEL PREBURNER.  THIS SUBROUTINE IS
C   ONLY CALLED BY FUELF SUBROUTINE.  THERE MUST BE A BIG DIFFERENCE BETWEEN
C   THE FPOV AND OPOV SET UP.  THE OPOV USES A SIMPLIFIED VERSION OF THE
C   PRIMING FUNCTION.
C
C   THIS SUBROUTINE USES THE ITERATION TO FIND THE NEW STEADY STATE VALUES
C   FOR A GIVEN CONDITION.
C   THE INJECTOR IS DIVIDED INTO SIX NODES AND THE ITERATION IS TO FIND THE
C   ENERGY BALANCE AMONG THEM.  OF THE SIX NODES, THE FIRST NODE IS THE
C   FPOV VALVE, AND THE LAST NODE IS THE INJECTOR OUTLET.  THESE ARE USED AS
C   THE BOUNDARY CONDITIONS.
C
**********************************************************************
C
C PRIME VALVE BUBBLE
C
      WPOV = prlint( (RHOSL/RHOVAL - 1.) * DWVALV, Tstep, 178)
*
*   WPOV =< .05
*
      IF ( FLAG ) THEN                                                SSM51000
        X = 1. - WPOV/WPOVI
        HT = AMIN1( 100., 100.+(X-.5)*(HV-100.)* 2.5 )
*
*       H(1) = AMIN1(HT,(HV+.900/.100*(HV-HSL))-X*(HV-HSL)/.100) replaced by
```

```
          hvhsl = HV - HSL
          H(1) = AMIN1(HT, HV + hvhsl*(9.0 - 10.*X) )
*
          X = (H(1) - HSL) / hvhsl
          IF( X.LT. 1. ) THEN
            RHOVAL = 1./(X/RHOV + (1. - X)/RHOSL )
          ELSE
            RHOVAL = RHOV * HV / H(1)
          END IF
          IF (WPOV .GT. -.001) THEN                                    SSM51100
*
*   The effect of the reset is hold WPOV output at 1.0, but
*   compute X above with a WPOV of .05 . If this is right, it
*   should be recoded to look intentional.
*
            WPOV = 1.
            CALL lmint0( WPOV, 178, -TooBig, .05 )
            WPOVP = pruint( DWVALV, Tstep, 179 )
            IF( WPOVP .LT. .04 ) THEN
              hlhsl = HL - HSL
              H(1) = HSL + 25. * WPOVP * hlhsl
              RHOVAL = RHOSL + (H(1) - HSL)/hlhsl*(RHOP3 - RHOSL)
            ELSE
              RHOVAL=RHOP3
              H(1)=HL
              FLAG = .FALSE.
            ENDIF                                                      SSM51200
          ENDIF
        ENDIF
C
        H(MAXN)=H(MAXN-1)
        P(MAXN)=PC
        RHOP(1)=RHOVAL
        RHOP(MAXN)=RHOP(MAXN-1)
C
        DO 110 I=2,MAXN-1
*         QC(I)=.0227/D(I)**1.8*OXHPF(RHO(I))*AHT(I)*1.5   replaced by
          QC(I)=.03405/( D(I)* X10th( D(I), 8) ) * OXHPF(RHO(I))*AHT(I) M51300
          TW(I) = pruint( - QOUT(I) /( W(I) * FGEN(22, 113+I, TW(I)) ),
       +                                                  Tstep, 178+I )
  110 CONTINUE
C
C     BEGIN ITERATION LOOP
C
        prpOK = .TRUE.
  200   DO 310 ITER = 1, mxopit
C
          P1 = trlint( XLINE*(DWLINP - DWP(1)*RHOP3/RHOVAL),
       +                                                  Tstep, 184 )
*
```

```
          P(1) = relax( prpOK, P1, P(1), 12 )
*
          DWLINP = trflow( DWLINE, ZLINE, -RLINE/RHOP3, P0 - P(1), 185 )
          IF( RHOVAL .LT. 1.E-3 ) THEN
            DWP(1) = GFLOW(P(1), P(2), P(1)/RHOVAL, 1., 1.4) /
     +                     X10th( 772.8*RVALVE, 5)
          ELSE                                                          SSM51500
            DWP(1) = trflow( DW(1), Z(1), -RVALVE/RHOVAL, P(1)-P(2), 186 )
          ENDIF
C
          DO 300 I = 2, MAXN-1
            J=I-1
            DWNEW = trflow( DW(I), Z(I), -R(I)/RHOP(I), P(I)-P(I+1),
     +                                                      185 + I )
            IF( I.LT.MAXN-1 ) THEN
              DWP(I)=DWNEW
            ELSE                                                        SSM51600
              DWP(I)=.5*( DWP(I)+ DWNEW )
            ENDIF
            RHOP(I) = truint( ( DWP(J) - DWP(I) ) / VOL(I), Tstep, 189+I )
            DWIN = recpos( DWP(J) ) - recpos( DWP(I) )
            HIN = recpos( DWP(J) ) * H(I-1) - recneg( DWP(I) ) * H(I+1)
            DWOUT = recpos( DWP(I) ) - recneg( DWP(J) )
*
*   Replaced the reference to the statement function UNEWF with:
*
            UP(I) = truint( QOUT(I) + HIN -
     +                    DWOUT*P(I)/(RHOP(I)*9336.*VOL(I) ) +
     +                    DWIN/VOL(I), Tstep, 193+I )                 SSM50900
*
            H(I) = UP(I) + P(I)/(RHOP(I) * 9336.)
            CALL o2pt( H(I), RHOP(I), I+4, PNEW, T(I) )
*
            P(1) = relax( prpOK, PNEW, P(I), 11+I )                     SSM51700
*
            DWA = .5*( ABS( DWP(I) ) + ABS( DWP(J) ) )
            QF(I) = QC(I) * XtoY( T(I)/TW(I), 0.47) * X10th( DWA, 8) *
     +              ( TW(I) - T(I) )
            QB(I) = 0.
            HG = AMIN1(28.7 + 2.13*ALOG( P(I) ), 38.6 - .00762*P(I) )
            IF ( H(I) .LT. HG)
     +         QB(I) = AMIN1( DWA*(HG - H(I) ),
     +                    OXBOIL( TW(I), T(I) ) * AHT(I) )
            QOUT(I) = QF(I) + QB(I)
  300     CONTINUE
          RHOP(MAXN)=RHOP(MAXN-1)
C                                                                       SSM51800
          IF ( prpOK ) GO TO 400
  310   CONTINUE
*
```

```
*  Convergence failure if do loop is completed
*
      CALL wrchg( 12, 16, "OPROP convergence failure." )
C
  400 CONTINUE
C
*  Formatted output to unit 6 was deleted.
C
      DO 500 I=2,MAXN-1
        DW(I) = step( 185 + I )                                        SSM52300
        RHO(I) = step( 189 + I )
        U(I) = step( 193 + I )
  500 CONTINUE
C
      DW(1) = step( 186 )
      P1 = step( 184 )
      DWLINE = step( 185 )
      DWINJ = DW(MAXC)
      DWVALV = DW(1)
C
      END                                                              SSM52400
*
      FUNCTION OXBOIL(TW, T)
**********************************************************************
C
C   THIS FUNCTION IS THE BOILING HEAT TRANSFER OF THE OXIDIZER INSIDE THE
C   INJECTOR.  THE OUTPUT IS THE HEAT TRANSFER RATE BTU/IN**2 FROM THE WALL
C   TO OXID.
C
**********************************************************************
C
      PARAMETER ( twlo = 10.**.4217, twhi = 10.**1.135,
     +            oxb01 = 1.929E-6 * 10.** -.29, ten1 = 10.**1.75 )
*
*      X=ALOG10(AMAX1(.01,TW-T))
*      Y=AMAX1(5.15-2.3*(X-1.)**2,1.75+1.02*X)
*      OXBOIL=10.**Y*1.929E-6                  is faster when reduced to
*
      twmt = TW - T
      IF ( twmt .LE. .01 ) THEN
        OXBOIL = oxb01
      ELSEIF ( twlo .LE. twmt .AND. twmt .LE. twhi ) THEN
        OXBOIL = ten1 * XtoY( twmt, 1.02 )
      ELSE
        X = ALOG10( twmt )
        OXBOIL = XtoY( 10., 5.15 - 2.3 * (X - 1)**2 )
      ENDIF
      END
```

'trbtrq.for':

```
      FUNCTION TRBTQ0
************************************************************************
C
C   FUNCTION TRBTRQ(S,UC,T,PI,PRII,NCH,IFLG,CP,DW,G)                    SSM37300
C
C   PURPOSE:  COMPUTE TURBINE TORQUE AS A FUNCITON OF INPUTS
C
C    THE EQUATIONS AND FUNCTION TABLES USED ARE DESCRIBED IN PAGE
C    37-38 (ANALOG SIMULATION) OF THE DOCUMENT.  THE EQUATIONS IN PAGE 24
C    ARE INCOMPLETE.
C
C******ARGUMENTS******
C   INPUT:
C    S       = SPEED, RAD/SEC
C    T       = TEMPERATURE, DEG R
C    PI      = NOT USED            eliminated
C    PRII    = PRESSURE RATIO, OUT/IN
C    NCH     = TURBINE DESIGNATION, 1=LPFT, 2=HPFT, 4=HPOT              SSM37400
C    IFLG    = FLAG=0 FOR INPUT MODE  eliminated
C    CP      = SPECIFIC HEAT AT CONSTANT PRESSURE, BTU/LBM-DEG R
C    DW      = FLOWRATE, LB/SEC
C    G       = GAMMA
C
C   OUTPUT:
C    UC      = ISENTROPIC VELOCITY RATIO
C    TRBTRQ  = TURBINE TORQUE
C
************************************************************************
C
      DIMENSION NFG(4),D(4)
      PARAMETER ( const = 12. * 0.1446 )
      DATA NFG/38,39,0,40/
      DATA D/7.4,10.19,6.0,10.09/
*
      CALL fgset( 38 )
      CALL fgset( 39 )
      CALL fgset( 40 )
*                       Next to make TRBTQ0 a function so the
*                       entry TRBTRQ will be one.
      TRBTQ0 = 0.0
      RETURN
C
      ENTRY TRBTRQ(S, UC, T, PRII, NCH, CP, DW, G)                       SSM37500
************************************************************************
C
      DH = AMAX1( 1.0E-04, CP*T*(1. - XTOY( PRII,(G - 1.)/G ) ) )
      sqrtdh = X10th( DH, 5 )
```

```
UC = D(NCH)*S / ( 5371.2 * sqrtdh )
EF = fgen( NFG(NCH), 129 + NCH, UC)
TRBTRQ = const * EF * DW * sqrtdh * D(NCH)
RETURN
END
```

APPENDIX B: SUPPORTING INPUT DATA FILES

This appendix shows the contents of input data files supporting the report version of the SSME simulation.

'ssme.run' is an edit-and-run file of input parameters defining the run. The top line is a run identifying header, which is reproduced on the output file.

BALDATA FILE FROM RKDYN 10/84 MOD FOR FUEL2/NEW PROP,O2PROP,START DATA

RESTRT	RESUME	PERTB	iwrite		
false	false	false	100		
DT	DPR	DPL	DPUN	TPUN	TSTOP
0.0002	0.01	.00567	15.	40.	4.00
TPA	PCMALF	DTVC	DTPR	DTCVP	DTTR
40.0	0.	.008	.005	.0035	.015
DTFMRE	DTFMC	DTMRFC	DTFMRA	DTMCX	DTLM
.012	.013	.017	.001	.02	.006
NONZRO	MODETST	PCTPERT	TOPEN	TPERT	DTPERT
0	0	0.	116.9999	100.	100.
rxfact(1)	rxfact(2)	rxfact(3)	rxfact(4)	rxfact(5)	rxfact(6)
1.0	1.0	1.0	1.0	1.0	1.0
rxfact(7)	rxfact(8)	rxfact(9)	rxfact(10)	rxfact(11)	rxfact(12)
1.0	1.0	1.0	1.0	1.0	1.0
rxfact(13)	rxfact(14)	rxfact(15)	rxfact(16)		
1.0	1.0	1.0	1.0		
AHT1(4)	AHT1(5)	AHT1(6)	AHT1(12)	CDPFP1	CTQFP1
1.4049E 04	3.8770E 03	2.6510E 03	3.4395E 04	1.0000E 00	1.0000E 00
CDPFP2	CTQFP2	CTQFT1	CDPOP1	CTQOP1	CDPOP2
1.0000E 00	1.0000E 00	8.9498E-01	1.0000E 00	1.0000E 00	1.0000E 00
CTQOP2	CDPOP3	CTQOP3	CTQOT12	FT2S	AFT2
1.0000E 00	1.0000E 00	1.0000E 00	1.0000E 00	8.7225E-01	9.4153E 00
CTQFT2	EOT2S	AOT2	CTQOT2	AFI	EFFCM
1.0072E+00	9.3861E-01	2.9433E+00	1.0000E 00	2.4725E+01	1.0000E+00
ACN	THRSTC	AHTC4	AHTC5	AHTC6	ABMOV
8.1810E+01	1.5626E+02	2.7770E+03	2.0200E+02	2.4600E+02	1.4460E+01

ABOPO	ABFPO	ABCCV	ABMFV	DMOT2	DMFT1
3.7982E-01	2.4034E 00	6.8180E 00	1.5590E 01	1.0090E 01	6.6300E 00
DMFT2	CP(2)	CP(3)	ANOT1	BNOT1	CNOT1
1.0190E+01	1.9595E-06	1.2981E-04	4.1430E-03	1.6814E-04	1.9595E-06
AOT1	BOT1	R(1)	R(3)	R(4)	R(5)
1.0998E 00	1.6443E-01	1.2981E-04	7.0952E-06	7.2660E-05	2.4327E-03
R(6)	R(7)	R(8)	R(9)	R(10)	R(11)
1.3170E-03	3.0090E-04	4.0210E-06	7.7691E-05	5.1824E-05	4.8890E-05
R(12)	R(13)	RFCOD	RFMCF	RFMCO	RACV
1.3370E-04	4.4319E-04	6.7547E-06	5.3910E-05	9.9706E-05	1.7584E-01
RBAF	RPFS	RSFS	RFPFI	ROPFI	RITN
9.4648E-04	1.1741E-02	5.5757E-03	1.3360E-04	5.3980E-04	1.1030E-03
RMCI	RFT1V	ROS	RFPOI	ROPOI	RFPOL
6.8150E-05	2.0600E-04	4.1367E-07	7.9780E-03	5.6486E-02	6.3310E-04
ROPOL	RFT2C	ROP2C	ROI	ROCOD	RMOVL
3.8800E-03	4.9240E-01	7.1738E+00	3.5000E-05	1.1683E-06	3.1148E-05
ROP3C	ROT1F	QHT412	TFACT		
3.9354E+01	2.1890E-04	8.2044E-02	1.0110E+00		
ELENF(1)	ELENF(2)	ELENF(3)	ELENF(4)	ELENF(5)	ELENF(6)
43.0	72.0	82.0	90.0	40.0	0.0
ZFL(1)	ZFL(2)	ZFL(3)	ZFL(4)	ZFL(5)	ZFL(3)
9.159E-4	3.003E-3	3.163E-3	1.928E-3	8.520E-4	0.
ZFC(1)	ZFC(2)	ZFC(3)	ZFC(4)	ZFC(5)	ZFC(6)
0.0	9.615E-4	2.191E-3	1.129E-3	8.944E-4	6.213E-4
RIF(1)	RIF(2)	RIF(3)	RIF(4)	RIF(5)	RIF(6)
1.026E-5	3.379E-5	3.027E-5	2.766E-5	2.092E-5	0.0
DHYD(1)	DEHYD(2)	DEHYD(3)	DEHYD(4)	DEHYD(5)	DEHYD(6)
11.0	6.0	3.51	4.73	2.16	1.53
DEHYD(7)	DEHYD(8)	DEHYD(9)	DEHYD(10)	DEHYD(11)	DEHYD(12)
2.455	3.6	2.637	3.68	2.49	7.17

truncated - full file on diskette

The function generation tables are contained in the editable file 'ssme.dat':

```
No:  24  Pts:  2    PA vs TIME
    0.0        14.7        10.0        14.7
---------- ---------- ---------- ----------
No:  30  Pts:  4    LPFT NORM AREA VS ETAFT1
       0.          1.4         275.       1.065         300.       1.005
---------- ---------- ---------- ---------- ---------- ----------
    1000.         .81
---------- ----------
No:  22  Pts:  3    CPM VS TEMP
       0.         .02        1000.         .12        2000.         .12
---------- ---------- ---------- ---------- ---------- ----------
No:   9  Pts:  4    FUEL TANK PRESSURE VS TIME
       0.          46.         2.3          46.        12.3          46.
---------- ---------- ---------- ---------- ---------- ----------
      40.          46.
---------- ----------
No.   Pts        FUEL TANK ENTHALPY VS TIME
   7    4           0          92        1.65          92        1.95
---- ---- ---------- ---------- ---------- ---------- ----------
        92          10          92
---------- ---------- ----------
No.   Pts        MFV DIFFUSER THERMAL FACTOR VS TIME
  10    2           0.          1.         100.          1.
---- ---- ---------- ---------- ---------- ---------- ----------
No.   Pts        DELTA-P/RHO*N**2 VS DW/RHO*N FOR LPFP
  51   13     -26.1       .0467       -17.4       .0397        -8.7
---- ---- ---------- ---------- ---------- ---------- ----------
 .0358         0.0         .0335        10.         .0380        20.
---------- ---------- ---------- ---------- ---------- ----------
 .0410        27.8900      .04100      32.3600      .0400        37.8
---------- ---------- ---------- ---------- ---------- ----------
 .0370        42.50        .0350       50.0         .0225        55.0
---------- ---------- ---------- ---------- ---------- ----------
 .000         100.         0.
---------- ---------- ----------
No.   Pts        TORQ/RHO*N**2 VS DW/RHO*N FOR LPFP
  52    5      0.          1.080       31.00        1.88         38.17
---- ---- ---------- ---------- ---------- ---------- ----------
   2.102        60.0         2.720       80.00        0.00
---------- ---------- ---------- ---------- ----------
No.   Pts        DELTA-P/RHO*N**2 VS DW/RHO*N FOR HPFP
```

53	13	-11.3	.229	-7.5	.195	-3.8
.1760		0.00	.1630	4.00	.1700	8.00
.1730		10.00	.1730	12.00	.1690	14.00
.1630		16.00	.1480	20.00	.1150	30.00
0.0		100.0	-.100			

No. Pts — TORQ/RHO*N**2 VS DW/RHO*N FOR HPFP

54	10	0.	.300	4.00	1.116	8.00
1.912		10.00	2.319	12.00	2.672	14.00
3.018		16.00	3.178	20.00	3.527	30.00
5.50		100.00	5.500			

No. Pts — MAIN CHAMBER PRIMING FUNCTION

3	6	0.	0.	10.	.02	40.
.05		46.2	1.	70.	1.	80.
1.						

No. Pts — OXID TANK ENTHALPY VS TIME

5	2	0.	-56.56	100.	-56.56	

No. Pts — OXID TANK PRESSURE VS TIME

37	4	0.	55.	2.3	55.	12.3
55.		40.	55.			

OX PREVALVE R VS TIME

44	2	0.0	1.516E-6	10.0	1.516E-6	

DELTA-P/RHO*N**2 VS DW/RHO*N FOR LPOP

45	12	-40.	.0530	0.0	.0408	5.
.0417		10.	.0413	15.	.0398	20.
.0379		25.	.03610	30.0	.03400	35.0
.03160		40.0	.02850	45.0	.0244	50.
0.0195						

TORQ/RHO*N**2 VS DW/RHO*N FOR LPOP

46	12	-40.0	1.7	0.00	1.700	5.00

1.40	10.0	1.00	15.00	1.00	20.0	
1.1250	25.	1.254	30.	1.380	35.	
1.510	40.	1.635	45.	1.764	50.	
1.890						

DELTA-P/RHO*N**2 VS DW/RHO*N FOR HPOP

47	14	0.	.01418	2.0	.01428	3.0
.0143		4.0	.01426	5.00	.01417	6.00
.0140		6.50	.01386	7.0	.01364	7.50
.01329		8.0835	.01266	8.509	0.01194	8.9344
.01109		9.3598	.01025	12.00	0.00504	

TORQ/RHO*N**2 VS DW/RHO*N FOR HPOP

48	14	0.	.1101	2.0	.1156	3.00
.1182		4.0	.1216	5.00	.1270	6.0
.1327		6.50	.1361	7.0	.1388	7.5
.1414		8.0835	.1439	8.509	.1435	8.9344
.1415		9.3598	.1390	12.0	.1052	

DELTA-P/RHO*N**2 VS DW/RHO*N FOR PBP

49	13	0.	.0086	.1	.0087	.2
.00884		.30	.00902	.40	.0093	.50
.00974		.60	.01012	.70	.00987	.80
.00944		.90	.00893	1.0	0.0084	1.1
.00781		1.5	.00545			

file is truncated - full file on diskette

The data for output is selected by editing the 'select.out'

```
Study Run          Identifying header line.
0 ITS( PFSP )
0 ITS( P10P )
0 ITS( P11P )
0 ITS( P12P )
0 ITS( P4P )
0 ITS( P13P )
0 ITS( P5P )
0 ITS( P6P )
    .
    .
    .

1 ABMOV
1 ACCV
1 AFPOV
1 AGC
1 AHPV
1 AIN
1 AMFV
1 AMOV
1 AN
1 AOPOV
1 APV
1 ATH
1 DATA
1 DDW1
    .
    .
    .

1 XMFV
1 XMOV
1 XOPOV
1 YCCCV
1 YCFPOV
1 YCMFV
1 YCMOV
1 YCOPOV
1 ZCOM
1 ZFIG
1 ZOIN
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1993	Final Contractor Report

4. TITLE AND SUBTITLE

Analysis of the Space Shuttle Main Engine Simulation

5. FUNDING NUMBERS

WU-590-21-11
NAG3-1031

6. AUTHOR(S)

J. Alex De Abreu-Garcia and John T. Welch

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Department of Electrical Engineering
University of Akron
Akron, Ohio 44325

8. PERFORMING ORGANIZATION REPORT NUMBER

E-7567

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135-3191

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA CR-191063

11. SUPPLEMENTARY NOTES

Project Manager, Carl Lorenzo, Instrumentation and Control Technology Division, NASA Lewis Research Center, (216) 433-3733.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 15

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This report analyzes the digital code used to simulate dynamic performance of the Space Shuttle Main Engine. This simulation program is written in Fortran. The purpose of the analysis is to identify a means to achieve faster simulation execution, and to determine if additional hardware would be necessary for speeding up the simulation. The analysis included the use of custom integrators based on the Matrix Stability Region Placement method. In addition to speed of execution the accuracy of computations, the useability of the simulation system, and the maintainability of the program and data files were examined. A revised code implementing the study recommendations was implemented but not verified for accuracy.

14. SUBJECT TERMS	15. NUMBER OF PAGES
SSME; Simulation; Multistep integration	260
	16. PRICE CODE: A12

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

NASA