# Computer Code AVSYS for Calculation of Fusion Plant Availability

Z. Musicki and C.W. Maynard

November 1985

UWFDM-663

Presented at 11th Symposium on Fusion Engineering, November 18-22, 1985, Austin, TX.

# Computer Code AVSYS for Calculation of Fusion Plant Availability

Z. Musicki and C.W. Maynard

Fusion Technology Institute
University of Wisconsin
1500 Engineering Drive
Madison, WI 53706

http://fti.neep.wisc.edu

November 1985

UWFDM-663

Presented at 11th Symposium on Fusion Engineering, November 18-22, 1985, Austin, TX.

# COMPUTER CODE AVSYS FOR CALCULATION OF FUSION PLANT AVAILABILITY

Z. Musicki and C.W. Maynard
Nuclear Engineering Department, 1500 Johnson Drive
University of Wisconsin-Madison, Madison, WI 53706-1687

## I. Introduction

This paper describes the computer program AVSYS and the recent changes implemented in order to improve its usefulness. The incentive for writing this program comes from the perceived need for availability analysis in fusion engineering (see related paper [1]). Of course, the utility of the code is not confined to the fusion area, but can include the availability/reliability analysis of any system.

AVSYS is a Monte Carlo simulation code, whose basic principles and features have been described in [2] and [3]; sample applications to the MARS conceptual design were included in both references. The code is currently one of the four being examined for use in fusion (two of the others being deterministic and the third also a Monte Carlo code). These programs have been recently compared [4] with the conclusion that none are to be strongly favored over the others. This paper will describe recent improvements to AVSYS in the areas of speed of execution, variance reduction and versatility of application.

In general, there are two basic methods of computing system availability: deterministic and stochastic (i.e., Monte Carlo). In the deterministic methods, the solution is presented in the form of closed form mathematical expressions. These expressions tend to be complicated even for relatively simple cases; however, the execution time of such programs is much shorter than that of a Monte Carlo program. The Monte Carlo methods simulate system transitions (failure, repair, etc.) and options that are possible at each transition by comparing certain probability parameters to random numbers. These methods are flexible in that complex and more realistic situations can be modeled. The disadvantages are increased computation time and the related problem of variance reduction (i.e., many histories have to be run in order to achieve acceptable statistical variance of the result).

In order to minimize the computer running time, most Monte Carlo programs that the authors are aware of simulate Markovian transitions of the system [5,6] with the price being paid in increased inflexibility of application of the program. Transition times (failure, repair) of the whole system are simulated assuming constant failure rates and repair rates of its components (the system being the whole power plant for instance). In AVSYS, the program looks at each component individually and decides if and when it fails or is repaired. There is also no constraint in principle, on the type of distribution (in time) of component failure and repair (the older version of AVSYS did assume constant failure rates and constant repair times).

AVSYS has been run on the NMFECC Cray computers at Livermore; a MARS problem with 85 subsystems, 41 gates, 200 time steps and 100 histories runs in about a minute of Cray-1 time (the C or D machine). The Cray X-MP/2 (the E machine at the NMFECC) should execute a little faster due to a shorter clock period (9.5 ns vs. 12.5 ns). We also ran the same problem on the IBM-PC in about 1.5 hr.

The version of AVSYS described in [2] and [3] was able to handle multiply operating units of the same subsystem including m out of n operation, spares, deferred repair and scheduled maintenance. The assumptions incorporated were of parallel repair facilities and constant failure rates and repair times, with a single failure mode for each subsystem. It was set up to handle 100 subsystems and 100 gates of each kind (these parameters could easily be changed in the program). The assumptions could be changed relatively easily.

In order to use the program, the user had to input the reliability/repair information for each subsystem, as well as the system configuration information (number of identical subsystem units operating, number of spares, etc. as well as the connection of subsystems, AND and OR gates in the success tree of the system).

## II. Variance Reduction

A biasing scheme has been implemented in AVSYS in order to reduce the variance. This scheme is described in [3]. The component failure rates are multiplied by a user-chosen factor in order to artificially increase the number of failures. There is a corresponding decrease in the weight given each failure in order to unbias the downtime estimator. For a component i with a failure rate $\lambda_i$, a new failure rate is instituted:

$$\lambda_i^* = k\lambda_i$$

where k is some pre-specified constant.

Upon failure of component i, the downtime credited in each step $\Delta t$ during repair is adjusted by multiplying $\Delta t$ by the weight $w_i^n$ for the n-th failure of component i:

$$w_i^n = w_i^{n-1}\ \lambda_i/\lambda_i^*\ \exp(-d(\lambda_i - \lambda_i^*))$$

$$w_i^o = 1\ ;$$

d is the elapsed time between the end of the previous repair and the beginning of the current failure. Figure 1 explains how the downtime is credited at each failure. Since now only a portion of $\Delta t$ in each time step is credited to downtime at each failure, components and gates may change state inside each $\Delta t$. If there are several units of a subsystem operating together (e.g., m out of n system) then the state of the subsystem is determined by looking at the state of each unit inside $\Delta t$ to compute the state(s) of the subsystem inside this $\Delta t$.

If the weight $w_i^n$ becomes too small, the history is either terminated for this component (meaning it stays up until the end of current history), or it is resumed but with an increased weight. This is the game of Russian roulette. The cutoff weight $w_0$ and the probability R are input by the user. Then if $w_i^n < w_0$, the history of component i is terminated with probability (1 - R) and is resumed with probability R, but with the

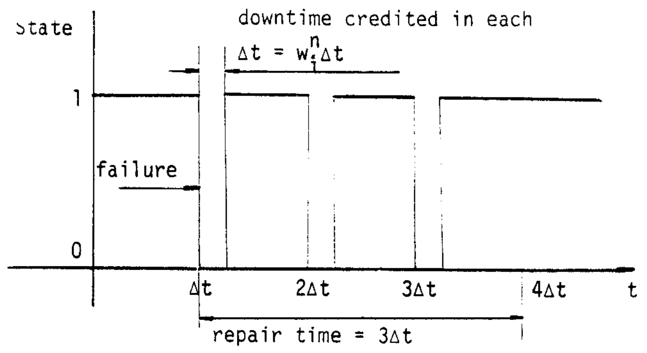
State
downtime credited in each
$\Delta t = w_i^n \Delta t$
1
failure
0
$\Delta t$
$2\Delta t$
$3\Delta t$
$4\Delta t$
t
repair time = $3\Delta t$

Fig. 1. Subsystem downtime credited in each $\Delta t$.

increased weight $w_i^{n+1} = w_i^n/R$, to be acquired at the next failure.

The values of k, $w_0$ and R are determined by trial and error for minimum computation time and variance, $\sigma^2$. In our runs, we generally apply this scheme if $\lambda_i < 10^{-5}$ $hr^{-1}$; our time step is $\Delta t = 50$ hr and there are 200 time steps and 100 histories.

We have run a sample problem with a simple three-input AND gate (input No. 1 consists of 5 identical units, input No. 2 consists of 2 identical units and input No. 3 consists of one unit). The input information is described in Table 1. For 200 time steps of 50 hr each and 100 histories, the problem runs in 6 sec of Cray-1 time with a standard deviation, $\sigma$, of less than 1% for most time steps. Without the biasing scheme in place, this standard deviation is about 10-20% with about the same running time. The sample standard deviation, $\sigma_s$, of the mean values of availability for each time step is 10 times less than $\sigma$ (100 trials). Assuming normal distribution over trials [3], about 99.7% of the population of availabilities for each time step will lie within $\pm 3\sigma$ (standard deviations) of the mean value for a particular time step, so an acceptable precision results from employment of the biasing scheme. Similar results are obtained by running a sample MARS problem [7].

## III. Running Time Reduction; Vectorization and Parallelism

The motivation for attempting to increase the efficiency of the program comes from the fact that any detailed modeling of a reasonably large system will involve thousands of subsystem inputs and logic gates. The simple representation of MARS that we ran had 85 subsystems and 41 gates. That problem took about one minute of Cray time (200 time steps and 100 histories). Usually, one wants to run parametric studies which then involve several runs of the problem.

As a result of our biasing scheme, we've had to add some subroutines in our program with a small increase in running time [3,7] (the program layout is presented in Fig. 2). However, this is more than offset by vectorizing [8] the routine FAIL which does the Monte Carlo simulation of failure by comparing a random number to a component's reliability (in REPAIR, we just add a specified repair time to the component's downtime). Vectorization allows more efficient use of processor time inside DO loops. Before vectorization, about 7 times as much time is spent in FAIL as in an average subroutine (30-60 μs per call). After vectorization FAIL execution time is comparable to that of an average subroutine which cuts program execution time in half.

The program execution time can also be decreased (at least theoretically) by employing the parallel processing capabilities of the Cray X-MP/2 (E machine) recently installed at the NMFECC [9]. This computer has two processors that can process concurrently a single program. The program has to be changed to insure the right timing. However, so far we have not been able to show a significant decrease of computing time; it is difficult to get both processors to work on the same program when there are other users in the system. We are continuing to work on this aspect of speed improvement.

## IV. Running Time Reduction: A Method of Sampling Transition Times

The method of simulation of subsystem/component failure in our program has been to compare its time step reliability, $\exp(-\lambda_i \Delta t)$, to a random number in (0,1). This is very inefficient, because in order to generate one failure, many time steps with no failure have to be sampled even with a biasing scheme in place. For example, for $\lambda_i^* = 10^{-3}$ $hr^{-1}$ and $\Delta t = 50$ hr, a failure will occur approximately once for every 20 steps sampled, on the average. It would be more efficient to sample time to failure of each component/subsystem since then, each sampling will lead to a failure and we can generate as many failures as we want for good statistical properties of our result. Apparently, this method has been used in a reliability Monte Carlo program [10]. The time to next failure of component i, $t_{fi}$ is generated by:

$$t_{fi} = - \ln \xi/\lambda_i$$

where $\xi$ is a random number in (0,1) and $\lambda_i$ is the ith subsystem failure rate (assumed constant over time). For time-dependent failures:

$$\xi = \exp\left(-\int_{t_1}^{t_1+t_f} \lambda_i(t)\, dt\right)$$

Table 1. Input Variables for Sample Problem

| Subsystem Number | # Operating Units | # Actively Redundant | # Spares | Failure Rate, $hr^{-1}$ | Repair Time, hr |
|---|---|---|---|---|---|
| 1 | 5 | 0 | 0 | 1. E-5 | 100. |
| 2 | 2 | 0 | 0 | 3.3E-6 | 500. |
| 3 | 1 | 0 | 0 | 1.4E-6 | 200. |

| k | $w_0$ | R | |
|---|---|---|---|
| 100. | 0.001 | 0.002 | for all three subsystems |

where $t_1$ is the time at the beginning of the next "up" period after the end of the current repair. An iterative procedure has to be implemented to find $t_f$; this shouldn't be too time consuming provided a limited (say 10) number of time intervals (inside which $\lambda_i$ is constant) is allowed.

Bookkeeping procedures have to be implemented in order to keep track of transition times for each component and how each transition influences the whole system and the other components and their transition times. This method is not as flexible as far as ease of modeling the "real world" is considered, but so far we haven't encountered any insurmountable difficulties.

## V. Program Changes: Failure Modes

The next few sections will describe the changes made in the code that relax the previously made assumptions and constraints and make it more realistic.

One of the changes is the possibility of including more than one failure mode for each subsystem. Each failure mode will have its own failure rate and repair time. This is accomplished by adding extra dimensions to the arrays holding the values of failure rates and repair times (up to 10 failure modes are allowed). An extra random number is generated at failure in order to choose the failure mode.

## VI. Time Dependent Failures and Wear-out

Failure rates can vary with time instead of being constant as is often assumed. Some types of components have failure rates that follow the bathtub curve in which there are three distinct periods in the life of a component: burn-in (decreasing failure rates), random failures (constant failure rates) and wear-out (increasing failure rates) [11]. Some other types of components do not exhibit the constant failure rates at all.

The input stream to the program can specify up to 10 cutoff times at which a component's failure rate changes and assumes a constant, specified value until the next cutoff time. The program automatically updates the failure rate at the specified time. The lifetime of a component can also be specified. Upon failure, the component is repaired to "as good as old" condition, unless the failure occurs within mttf (mean time to failure) of the component's lifetime; in that case it is replaced, i.e. repaired to "as good as new" condition.

## VII. Component Dependent (Common Mode) Failures

A component may fail because some other component has failed. In the input, we can specify the component i and its failure mode l that induces failure in component j with probability $P_{ilj}$. Repair time is also input for component j. Upon failure of component i, a random number in (0,1) is compared to $P_{ilj}$ to determine if component j fails too.

## VIII. Summary

The current improvements in the availability simulation computer program AVSYS are presented. These are

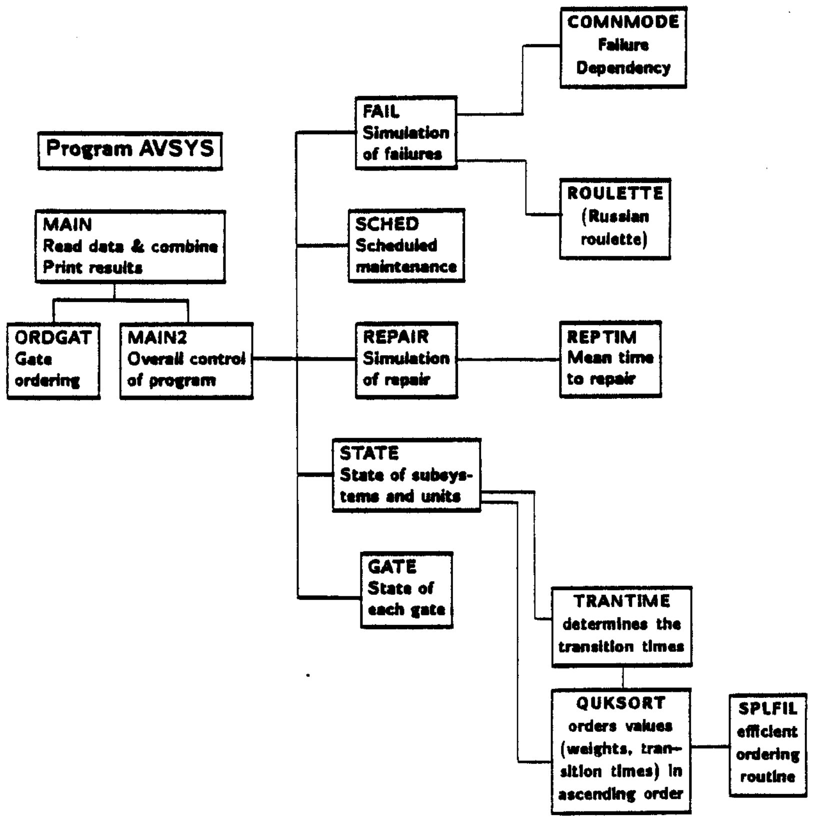
Program AVSYS
MAIN Read data & combine Print results
ORDGAT Gate ordering
MAIN2 Overall control of program
FAIL Simulation of failures
SCHED Scheduled maintenance
REPAIR Simulation of repair
STATE State of subsystems and units
GATE State of each gate
COMNMODE Failure Dependency
ROULETTE (Russian roulette)
REPTIM Mean time to repair
TRANTIME determines the transition times
QUKSORT orders values (weights, transition times) in ascending order
SPLFIL efficient ordering routine

Fig. 2. Layout of program AVSYS

in the areas of variance reduction, speed of execution and more realistic modeling.

## IX. Acknowledgement

This work has been sponsored by the U.S. Department of Energy.

## X. References

[1] Z. Musicki, C.W. Maynard, Y. Watanabe, A. Bennethum, K. Gruetzmacher, "The Fusion Engineering Data Base," presented at the 11th Symposium on Fusion Energy, Nov. 18-22, 1985, Austin, TX.

[2] Z. Musicki, C.W. Maynard, "The Availability Analysis of Fusion Power Plants as Applied To MARS," Nuclear Technology/Fusion, Vol. 4, No. 3, pp. 284-289, Sept. 1983 (Proceedings of the 5th Topical Meeting on the Technology of Fusion Energy, Knoxville, TN, 1983).

[3] Z. Musicki, "Availability Analysis of Fusion Plants Employing a Monte Carlo Simulation Computer Code," Ph.D. thesis, 1984, University of Wisconsin-Madison.

[4] S.L. Thomson, A. Dabiri, D.C. Keeton, B.W. Riemer, L.M. Waganer, "Availability Program Phase I Report," ORNL/FEDC-84/10, May 1985.

[5] G.E. Apostolakis, "Mathematical Methods of Probabilistic Safety Analysis," UCLA-ENG-7464, Sept. 1974.

[6] Ernest J. Henley, Hiromitsu Kumamoto, Reliability Engineering and Risk Assessment, Prentice Hall, 1981.

[7] Z. Musicki, "Recent Modifications in Availability Program AVSYS," University of Wisconsin Fusion Technology Institute Report UWFDM-634, Nov. 1985.

[8] "CFT, The Cray-1 Fortran Compiler," National Magnetic Fusion Energy Computer Center, Lawrence Livermore National Laboratory, Nov. 1984.

[9] "MPDOC, CTSS Multiprocessor Support," National Magnetic Fusion Energy Computer Center, Lawrence Livermore National Laboratory, Nov. 1984.

[10] Satish J. Kamat, Michael W. Riley, "Determination of Reliability Using Event-Based Monte Carlo Simulation," IEEE Transactions on Reliability, Vol. R-24, No. 1, April 1975.

[11] Z. Musicki, C.W. Maynard, "A Preliminary Fusion Availability Data Base," University of Wisconsin Fusion Technology Institute Report UWFDM-532, Feb. 1984.