Diagnostic Reasoner (DR)

Software User's Manual

Version 1.0

Table of Contents

Notices:

Note: DR cannot run on a target system “out of the box”, there are several tables that must be developed before running on your target system. They are described in more detail later in the document. The user is responsible for creating these tables and ensuring they are correct. Users must understand that if the input tables are not correct, the DR output (system failure modes) will be incorrect and cannot be relied upon.

Disclaimers:

PURPOSE

In various technologies, assessment of the system's health is critical for decision-making and performance. A consistent methodology that recognizes and interprets faulty behavior would give great insight into this investigation. The Diagnostic Reasoner's utilization of a matrix, that describes relationships between tests and failure modes, characterizes the problem in a compact way. Its applicability to many systems allows this procedure to be easily integrated into the current system of interest while the source code remains unchanged.

SOFTWARE

Diagnostic Reasoner (DR) is a Core Flight System (or Core Flight Software) (cFS) based application, developed by the Autonomy Operating System project at NASA. cFS is open-source, reusable flight software, also developed by NASA and targeted to space missions. DR runs within the cFS framework, and does model-based diagnosis of a target system. That is, given sensor data from a system, it will monitor the system to detect any failures (i.e., fault detection), and if a failure has occurred, it will determine which failure did occur (i.e., fault isolation).

LIMITATIONS AND ASSUMPTIONS

DR was not developed to the NASA software standards required to run on and diagnose safety-critical systems. DR is research-level software, Class E and non-safety critical, in the NASA terms defined by its software procedural requirements document NPR 7150.2. An appropriate use of DR is by research groups doing work on autonomous systems, or research on verification of autonomous systems. It also is only advisory: DR will report what failures it believes has occurred, and does not have a capability to issue commands or take action. DR has only been run on tethered aircraft; NASA requires small aircraft running research-class software to be tethered. It is not intended for operational uses or operational autonomous flight.

DR is released for use in non-critical research systems and demonstration projects. DR has gone through some internal software engineering procedures, such as requirements verification and internal code reviews, so it is hoped that the software runs as expected, but these were not done to the level required for higher classifications by NPR 7150.2.

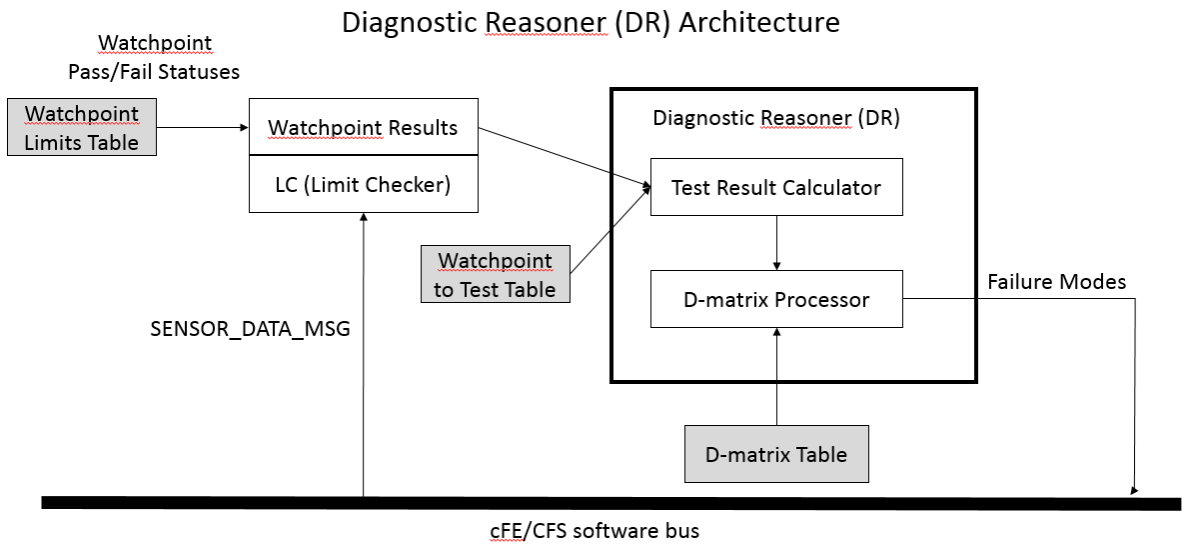
DR has software dependencies on cFS and on the generic cFS app, Limit Checker (LC). These are required to be installed before DR may be installed and used. These are described later in this document.

Finally, DR has some limitations according to the current algorithm. First, the failure propagation is assumed to be instant. If a particular failure mode takes time to propagate, a user must account for that and not try to diagnose the system during that time. Also, DR cannot currently diagnose failures that have effects upstream.

DR ARCHITECTURE OVERVIEW

DR uses a D-matrix (dependency matrix) approach to diagnosis. A D-matrix approach is a form of model-based diagnosis; a general inference engine is run on a model of the particular system of interest to do diagnosis of that system. In this case, the model is a D-matrix, which is a matrix relating the diagnostic tests for a system and the failure modes of that system. For a more detailed description of this, see the section DR D-MATRIX SOLVING ALGORITHM DESCRIPTION. The D-matrix approach was chosen to relate to work being done under other projects related to verifying D-matrices.

The general DR architecture is shown in the figure below. It shows the modules and data flow through the DR system.



In the above diagram, Limit Checker (LC) is another cFS application, which receives sensor data and checks that data elements have not exceeded predetermined thresholds. DR makes use of that capability of LC to process the sensor data, and reads the LC watchpoint results. It then uses one of its submodules, the Test Result Calculator, to make the simple conversion from watchpoint results into the test results that DR uses. The test results are then sent to the other submodule of DR, the D-matrix Processor, which uses the test results and a D-matrix to determine if any failures have occurred. The failure modes are then sent back to the cFS system for other applications to read.

The information needed to apply DR to specific systems of interest is stored in 3 tables (as indicated in shaded boxes above); each must be defined for DR to run. These tables are how DR is adapted to perform diagnosis on a particular system. First, the sensor values to read must be defined in the LC watchpoint definition table. The threshold limits on the sensor values must also be specified. The second table is the watchpoint to test table, which defines the mapping between the watchpoints to the DR test results. This is primarily mapping the watchpoint indices into the test indices, but there is some other information used to define the tests as well. The third table is the D-matrix table, which contains the D-matrix.

ADAPTING DR TO YOUR SYSTEM

As described above, the DR source code is general, and doesn't change as DR is used on one system vs. another. To adapt DR to your system, you will need to create 3 tables. These are used by DR to include information that is specific to the system being diagnosed. They are: the Watchpoint Limits Table, the Watchpoint to Test Table, and the D-matrix Table. These three tables are all implemented as cFS tables. (See the cFS documentation and help for descriptions of cFS tables in general.)

1. Watchpoint Results Table:

This table is owned by the Limit Checker (LC) application, and is usually found with the LC source code. The sensors and the thresholds to be used for monitoring must be also specified in this table. Note, an overall project may be using LC for other purposes; in that case, there will simply be additional entries in this table for DR.

2. Watchpoint to Test Table:

This table is owned by DR. Here, the watchpoint indices are mapped into test result indices used as inputs to the DR D-matrix processing algorithm.

3. D-matrix Table:

This table is owned by DR. It defines the D-matrix for the system, which maps how the system's failure modes are associated with the tests for them. This mapping must be accurate for a particular system, or DR will give incorrect diagnoses. The D-matrix is described in more detail later in this manual.

USER INSTRUCTIONS

Getting the Code

DR is expected to be released for general source release by the NASA software office. To get DR, start at

http://software.nasa.gov and search for "Diagnostic Reasoner". Follow the instructions on the page to request the software.

Installing and Running DR

Prior to DR installation:

1. Download and set up cFS software

As mentioned before, DR is an app which runs in the Core Flight System (cFS) framework. Before installing DR, the cFS software must be downloaded and set up. The website http://coreflightsystem.org/ includes links to find the software, which has been released on a combination of sourceforge and github.

Setting up a cFS installation and adapting it to your project is a complex procedure, which usually requires help from the cFS community. Guidance can be found under the support tab.

2. Install LC

Note, as described later, DR currently makes use of another standard cFS application called Limit Checker (LC). LC therefore must also be installed if DR is to be used.

Post cFS and LC setup:

3. Install DR

After cFS is installed, DR may be installed by copying its source into a directory where the other cFS apps are installed. By custom, a cFS installation will usually have a subdirectory named "apps", for these. Also commonly DR will be installed into that directory, under its own directory named "dr". Once DR is copied there, it will need to be integrated into the overall cFS installation, as one of the apps. Again, this is often a complex procedure.

Some of the usual required steps are that DR will need to be connected to the overall cFS build system, assigned message IDs according to how the project is managing them, and set to run in the cFS startup script. The cFS community pages will be helpful in providing descriptions of these terms, and how to go about integrating a cFS app in general.

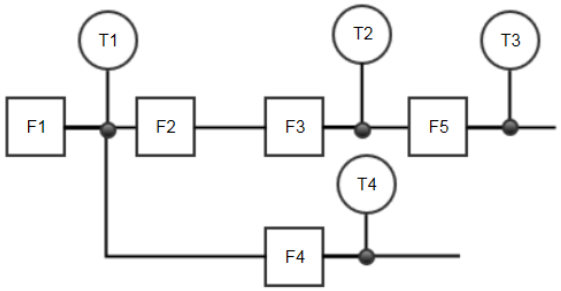
After DR is integrated into the cFS system, it should compile as part of that system, and should start up and run automatically when cFS is started.

DR D-MATRIX SOLVING ALGORITHM DESCRIPTION

Overview

DR uses a dependency-matrix (D-matrix) solving algorithm to perform diagnosis. A D-matrix is used by DR to map tests (which are usually based on sensor data or other means of determining the health of a system) to failure modes. This mapping is of course specific to the particular system of interest. An example system's failure mode propagation diagram and its corresponding D-matrix are shown below:

The failure modes in the diagram are given in squares and labeled F1-F5. The tests are given in circles and labeled T1-T4.



TESTS

FAILURE MODES

	T1	**T2**	**T3**	**T4**
F1	1	1	1	1
F2	0	1	1	0
F3	0	1	1	0
F4	0	0	0	1
F5	0	0	1	0

1: test implicates a failure mode

0: test does not implicate a failure mode

The interpretation of the failure mode propagation diagram is that the failure propagation is left to right. Therefore, T1 passing or failing may only implicate F1 being good or bad/suspect. That is reflected in the D-matrix column for T1, where the only "1" entry is in the row corresponding to F1. The test T2 failing may implicate F1, F2, and / or F3. Similarly, test T3 failing may implicate failure modes F1, F2, F3, and/or F5, and test T4 failing implicates F1 or F4.

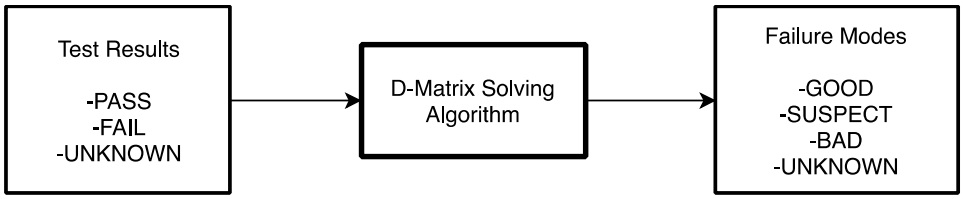
The information in the preceding paragraph is given in D-matrix form below the failure mode propagation diagram. The columns of the D-matrix correspond to the tests and the rows correspond to the failure modes.

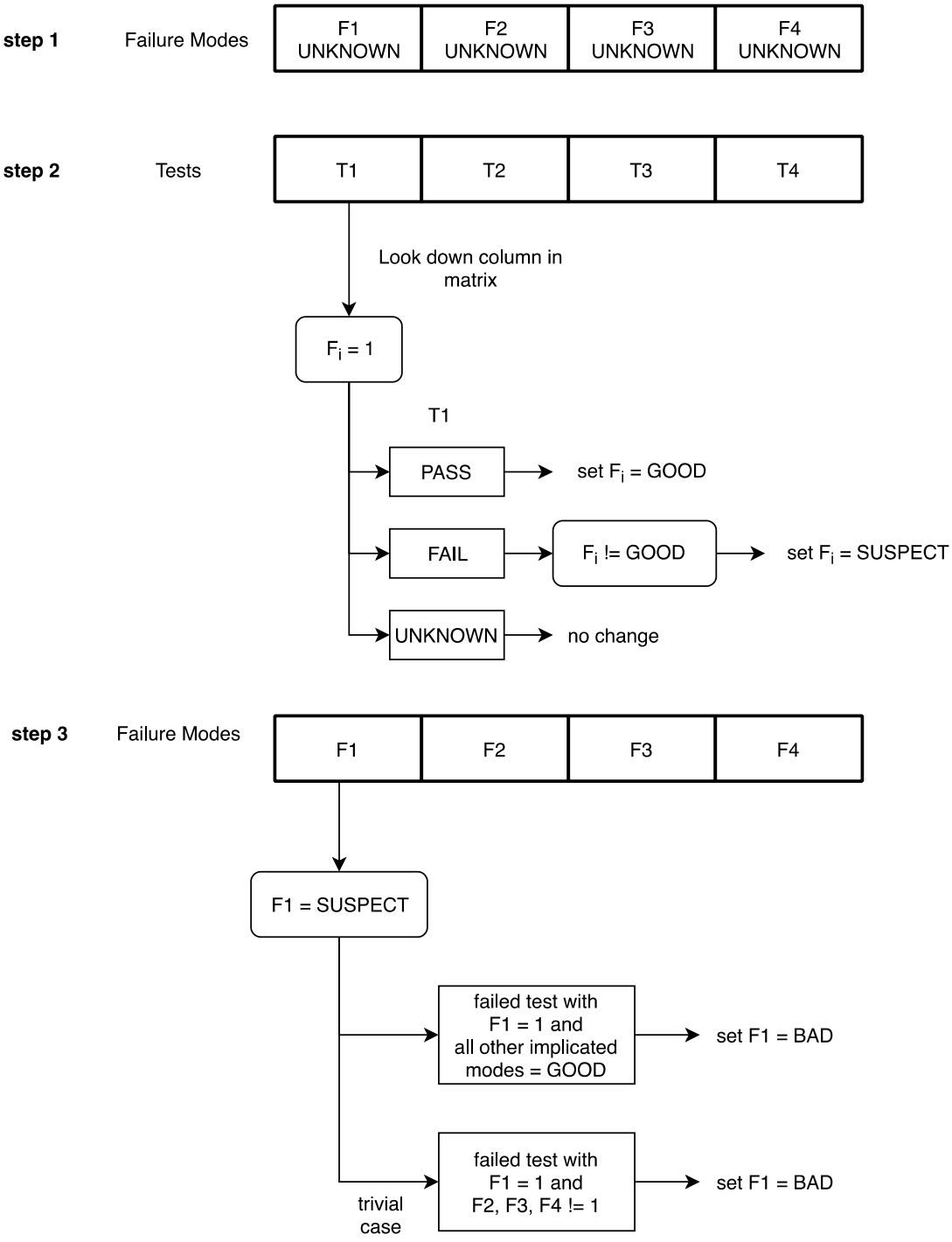
D-matrix solving algorithm

The D-matrix solving algorithm implemented in DR takes the test results as input, and returns which failure modes may have occurred. Each test result may have a value chosen from the enumerated list of "PASS", "FAIL", or "UNKNOWN". Each failure mode may have a value of "GOOD", "SUSPECT", "BAD", or "UNKNOWN".

In words, the D-matrix solving algorithm has the following steps:

1. Initialize the array of failure modes to "UNKNOWN".
2. Step through the array of tests. For each one,
    a. Check which failure modes are implicated by that test (aka the row has a 1 in the D-matrix). For each implicated failure mode, determine the new failure mode value based on value of the test result:
        i. Test result "PASS": Set that failure mode to "GOOD", regardless of the current failure mode value.
        ii. Test result "FAIL": If the failure mode has not already been marked "GOOD" (by an earlier test), then set that failure mode to "SUSPECT".
        iii. Test result "UNKNOWN": Do not set or change the failure mode.
3. Step through the array of failure modes. For each one,
    a. If the failure mode's value is "SUSPECT", check if it may be marked as "BAD":
        i. If there is a "FAIL"-ed test that implicates this failure mode, and all other failure modes associated with that test are marked "GOOD", then this failure mode is marked "BAD".
        ii. If the "FAIL"-ed test only implicates this failure mode, then this is a trivial case of the preceding rule and the failure mode may be marked "BAD".


Test Results
-PASS
-FAIL
-UNKNOWN
D-Matrix Solving Algorithm
Failure Modes
-GOOD
-SUSPECT
-BAD
-UNKNOWN


step 1
Failure Modes
F1 UNKNOWN
F2 UNKNOWN
F3 UNKNOWN
F4 UNKNOWN
step 2
Tests
T1
T2
T3
T4
Look down column in matrix
$F_i = 1$
T1
PASS
set F_i = GOOD
FAIL
F_i != GOOD
set F_i = SUSPECT
UNKNOWN
no change
step 3
Failure Modes
F1
F2
F3
F4
F1 = SUSPECT
failed test with F1 = 1 and all other implicated modes = GOOD
set F1 = BAD
failed test with F1 = 1 and F2, F3, F4 != 1
set F1 = BAD
trivial case

EXAMPLES

Example 1:

Given the D-matrix above, suppose that the 4 tests have the values:

- T1 = "PASS"
- T2 = "PASS"
- T3 = "FAIL"
- T4 = "PASS"

Step 1:

Mark all failure modes as "UNKNOWN":

- F1 = "UNKNOWN"
- F2 = "UNKNOWN"
- F3 = "UNKNOWN"
- F4 = "UNKNOWN"
- F5 = "UNKNOWN"

Step 2:

Process T1's "PASS":
Since it only may implicate F1, we set the failure mode of F1 to “GOOD”.

- F1 = "GOOD"
- F2 = "UNKNOWN"
- F3 = "UNKNOWN"
- F4 = "UNKNOWN"
- F5 = "UNKNOWN"

Process T2's "PASS":
We see that T2 may implicate F1, F2, and F3, so after processing that "PASS"-ed, all three implicated failure modes are now set to “GOOD”.

- F1 = "GOOD"
- F2 = "GOOD"
- F3 = "GOOD"
- F4 = "UNKNOWN"
- F5 = "UNKNOWN"

Process T3's "FAIL":
T3 implicates F1, F2, F3, and F5. However, F1, F2, and F3 all already have the value "GOOD" from previous tests, so they are not affected by T3's "FAIL". F5 will be marked as "SUSPECT" because it had the value "UNKNOWN":

- F1 = "GOOD"

- F2 = "GOOD"
- F3 = "GOOD"
- F4 = "UNKNOWN"
- F5 = "SUSPECT"

Process T4's "PASS":
T4 only implicates F1 and F4, so those are now marked as "GOOD":

- F1 = "GOOD"
- F2 = "GOOD"
- F3 = "GOOD"
- F4 = "GOOD"
- F5 = "SUSPECT"

Step 3:

This step only applies to those failure modes with values “SUSPECT”. F1, F2, F3, and F4, with values marked “GOOD”, are not considered.

F5 has the value "SUSPECT". We can see from the D-matrix that T3 is a "FAIL"-ing test which implicated F5. Looking down the column for T3, we see it also implicates F1, F2, and F3. However, all of those failure modes have value "GOOD". Since F5 is the only "SUSPECT" resulting from T3, we now mark F5 as "BAD":

- F1 = "GOOD"
- F2 = "GOOD"
- F3 = "GOOD"
- F4 = "GOOD"
- F5 = "BAD"

Given this set of test results, these are the final values of failure modes.

Example 2:

Given the D-matrix above, suppose that the 4 tests have the values:

- T1 = "UNKNOWN"
- T2 = "FAIL"
- T3 = "FAIL"
- T4 = "PASS"

Step 1:

Mark all failure modes as "UNKNOWN":

- F1 = "UNKNOWN"

- F2 = "UNKNOWN"
- F3 = "UNKNOWN"
- F4 = "UNKNOWN"
- F5 = "UNKNOWN"

Step 2:

Process T1's "UNKNOWN":
Since it is "UNKNOWN" it has no effect on it's implicated failure modes (in this case only F1):

- F1 = "UNKNOWN"
- F2 = "UNKNOWN"
- F3 = "UNKNOWN"
- F4 = "UNKNOWN"
- F5 = "UNKNOWN"

Process T2's "FAIL":
T2 may implicate F1, F2, and F3, and since they are all "UNKNOWN", they are set to "SUSPECT":

- F1 = "SUSPECT"
- F2 = "SUSPECT"
- F3 = "SUSPECT"
- F4 = "UNKNOWN"
- F5 = "UNKNOWN"

Process T3's "FAIL":
T3 may implicate F1, F2, F3, and F5. Since the first three are already "SUSPECT", they remain so, and F5 is additionally marked "SUSPECT":

- F1 = "SUSPECT"
- F2 = "SUSPECT"
- F3 = "SUSPECT"
- F4 = "UNKNOWN"
- F5 = "SUSPECT"

Process T4's "PASS":
T4 may implicate F1 and F4. This "PASS" therefore exonerates F1, and the value for F1 is changed to "GOOD". F4 is also marked "GOOD":

- F1 = "GOOD"
- F2 = "SUSPECT"
- F3 = "SUSPECT"
- F4 = "GOOD"
- F5 = "SUSPECT"

Step 3:

This step only applies to those failure modes with values “SUSPECT”. F1 and F4 have values of "GOOD", so they are not considered.

For F2's "SUSPECT", we can see from the D-matrix that T2 was the first "FAIL"-ed test that implicated F2. However, T2 also implicates F1 and F3. F1 is "GOOD", so it has no effect on determining if F2 should switch to "BAD". However, F3 is also "SUSPECT", so we cannot move F2 to "BAD" on the basis of T2. We also see from the D-matrix that T3 was also a "FAIL"-ed test that implicated F2. T3 also implicates F1, F3, and F5. F1 is "GOOD", but F3 and F5 are also "SUSPECT". So, F2 does not switch to "BAD" on the basis of T3's "FAIL"-ed test. There are no more tests that implicated F2, so F2 remains "SUSPECT".

For F3's "SUSPECT", we can see from the D-matrix that T2 was the first "FAIL"-ed test that implicated F3. However, T2 also implicates F1 and F2. F1 is "GOOD", so it has no effect on determining if F3 should switch to "BAD". However, F2 is also "SUSPECT", so we cannot move F3 to "BAD" on the basis of T2. We also see from the D-matrix that T3 was also a "FAIL"-ed test that implicated F3. T3 also implicates F1, F2, and F5. F1 is "GOOD", but F2 and F5 are also "SUSPECT". So, F3 does not switch to "BAD" on the basis of T3's "FAIL"-ed test. There are no more tests that implicated F3, so F3 remains "SUSPECT".

For F5's "SUSPECT", we can see from the D-matrix that T3 was the first "FAIL"-ed test that implicated F5. However, T3 also implicates F1, F2, and F3. F1 is "GOOD", so it has no effect on determining if F5 should switch to "BAD". However, F2 is also "SUSPECT", so we cannot move F5 to "BAD" on the basis of T3. There are no more tests that implicated F5, so F5 remains "SUSPECT".

- F1 = "GOOD"
- F2 = "SUSPECT"
- F3 = "SUSPECT"
- F4 = "GOOD"
- F5 = "SUSPECT"

Given this set of test results, these are the final values of failure modes.

DR TIMING RESULTS

These are the DR timing results:

- DR is fairly fast, less than 0.5 ms for a 50x50 D-matrix
- DR's algorithm looks polynomial-time with the size of the D-matrix

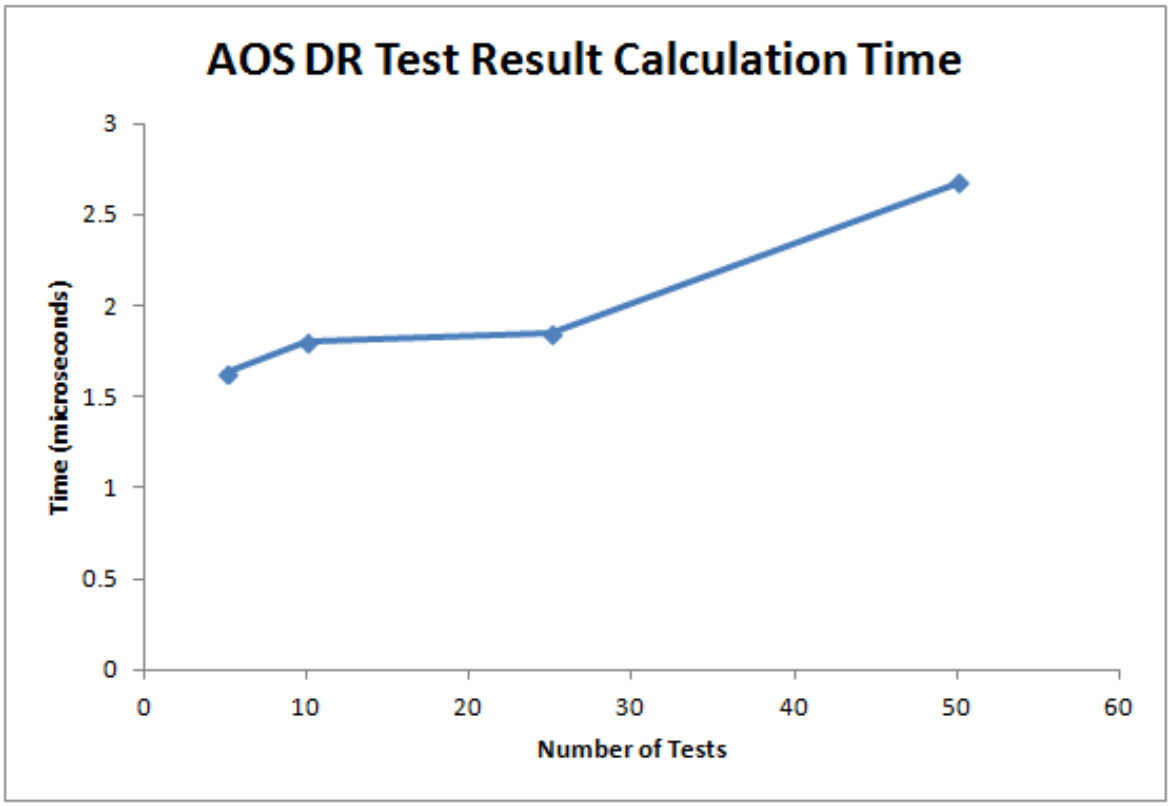
Bottom line: We plan to use D-matrices on the order of 50x50 so it should work.

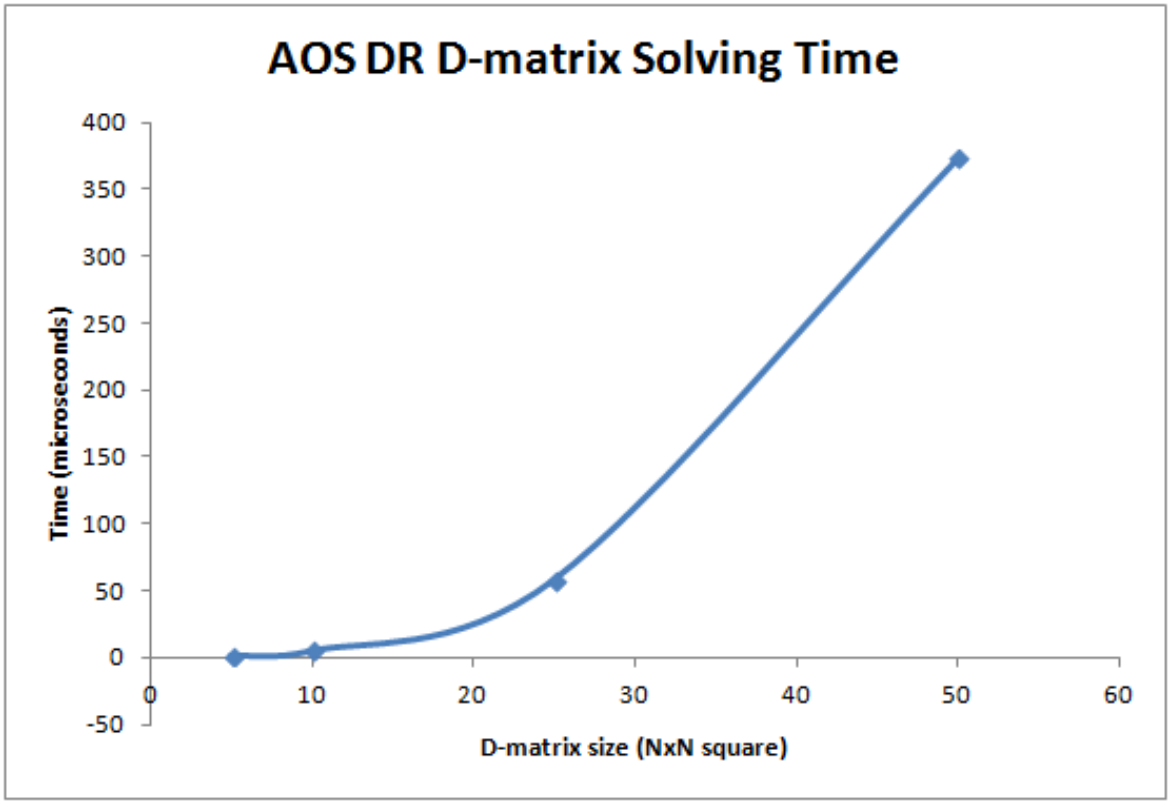
Test was implemented manually:

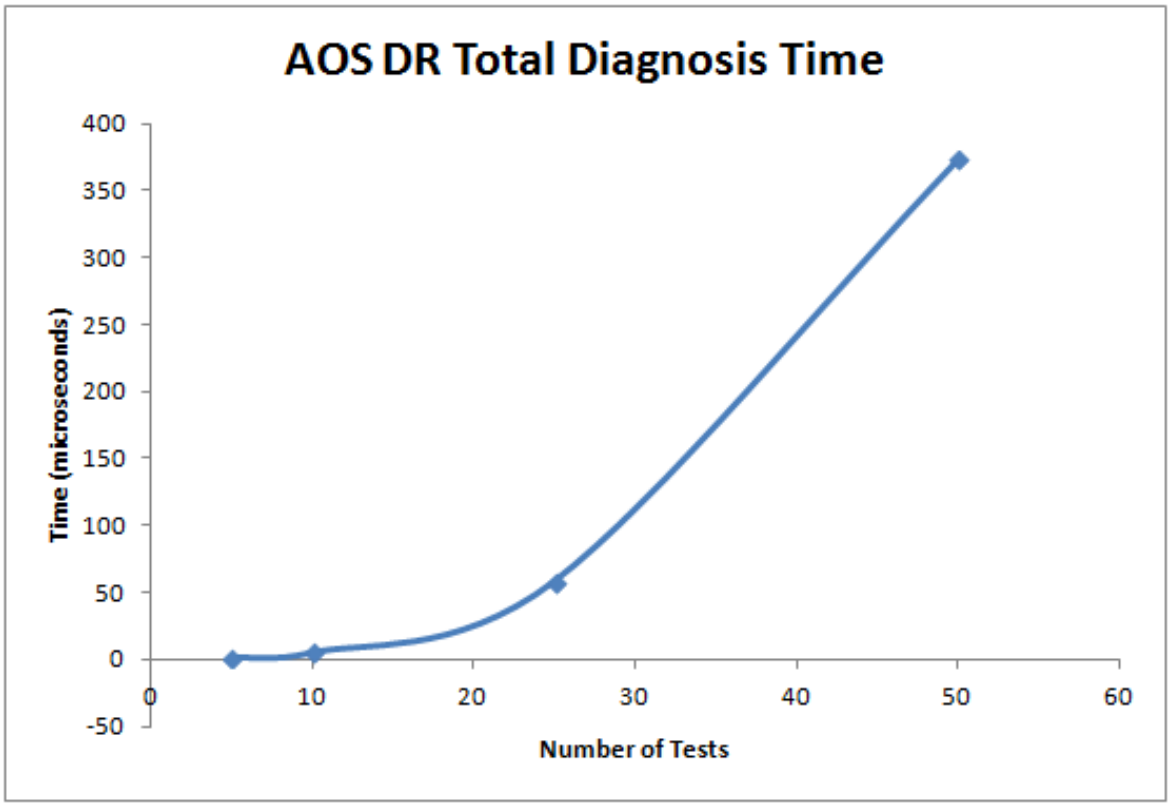
- Instrumenting the DR code
- Creating D-matrices and test result mappings of different sizes
- Running DR and printing time differences to the screen
- Manually copying the results to a spreadsheet.

The test was done on a 2.7 GHz quad-core Intel i7 processor laptop, running a Linux virtual machine.

The D-matrices used were square matrices of the given size, with every matrix entry as "true", to give a worst-case answer. This is not representative of the d-matrices that were generated, so the actual diagnosis time will be less than this measurement.




AOS DR D-matrix Solving Time
Time (microseconds)
400
350
300
250
200
150
100
50
0
-50
0
10
20
30
40
50
60
D-matrix size (NxN square)


AOS DR Total Diagnosis Time
Time (microseconds)
400
350
300
250
200
150
100
50
0
-50
0
10
20
30
40
50
60
Number of Tests

POTENTIAL IMPROVEMENTS

Remove DR's Dependence on LC:

As described previously and illustrated in the DR Architecture diagram, Limit Checker is another cFS application separate from that of Diagnostic Reasoner. DR relies on LC's functionality to process sensor data and assure the elements remain within predefined thresholds. Implementing ways for DR to perform these tasks itself would remove its dependence on LC and eliminate two steps from this system.

The Watchpoint Results Table would no longer be needed to store processed sensor data, as well as the Watchpoint to Test Table, because conversion of results from the LC tables would be unnecessary. As a result, the reduction of tables has the potential to lessen space taken up and better optimize runtime.

Build Additional Kinds of Tests into DR:

Building on DR's potential independence from LC as described above, supplementary tests, as well as those originally performed by LC can be built into DR. Removing the link between LC and DR would allow greater flexibility in DR's functionality and adaptability to the system of interest.

An example of an additional test that could prove useful would be the comparison of two or more values. This expands on LC's previous capability of simply checking whether certain values exceed defined thresholds but not between each other. Tests suitable to the system being studied and problem structure that aren't encompassed by the LC would also prove useful.