Tool Validation Planning Kit

User Guide

MathWorks Consulting Team

3/4/2024

Contents

About MATLAB® validation

In order to comply with the US Food and Drug Administration's Quality System Regulations, any software tool that is used as part of a medical device development process must be validated as fit for its intended use. Such validation of MATLAB, which requires documentation of context of use, risks of failure, and mitigation strategies, is outside the scope of this document. However, a very common risk mitigation strategy for commercial off-the-shelf software is to run a set of validation tests to confirm correct results on the end user's computer. This document describes how to run such a set of tests from The MathWorks. The resulting test report may be used as evidence for a MATLAB software validation plan. In addition, documentation is provided to extend these tests for any additional features that are considered high risk for the end user's particular context of use.

A test suite is provided for many commonly-used features of MATLAB, Signal Processing Toolbox™, Image Processing Toolbox™, Statistics and Machine Learning Toolbox™, Computer Vision Toolbox™, and Deep Learning Toolbox™. Major add-on features for additional toolboxes, warrant additional test cases. The test report is organized into chapters based on these large groups of functionalities.

Some typical contexts of use for MATLAB include

- Data analysis
- Algorithm design
- Automating software implementation via production code generation
- Test and measurement applications such as automated test equipment
- Incorporating MATLAB into a medical device

The contents of this kit are oriented toward the data analysis and algorithm design use cases. Production code generation is a higher risk usage, and requires additional verification activities. Please see our IEC 61508 Certification Kit (http://www.mathworks.com/products/iec-61508/) for further recommendations.

The latter two use cases require additional verification and validation activities beyond what is contained in this kit. In particular, using MATLAB as part of a medical device by installing it or deploying an application with MATLAB® Compiler™ requires compliance with FDA's *Guidance for Industry, FDA Reviewers, and Compliance on Off-The-Shelf Software Use in Medical Devices* (1999).

Installing the Tool Validation Planning Kit

Required Software

To utilize the Tool Validation Planning Kit, MATLAB, and the toolboxes to be validated should be installed. If you are running tests from the Computer Vision Toolbox, or Deep Learning Toolbox, the following add-ons are required:

- Deep Learning Toolbox for VGG-19 Network
- Deep Learning Toolbox for VGG-16 Network
- Deep Learning Toolbox for Inception-ResNet-v2 Network
- Deep Learning Toolbox for MobileNet-v2 Network
- Deep Learning Toolbox for Xception Network
- Deep Learning Toolbox for ResNet-50 Network
- Deep Learning Toolbox for ResNet-18 Network

In addition, a valid license for MATLAB, all toolboxes to be validated, and MATLAB Report Generator is needed.

Installing the test package

To install the Tool Validation Kit, simply double-click “Tool Validation Kit App.mltbx” in MATLAB to start the installation process. This will install both the tool validation kit software and the app. The app can be found in the MATLAB app gallery.

Uninstalling the test package

To uninstall the Tool Validation Planning Kit the user can simply uninstall the toolbox from the MATLAB Add-On Manager.

Executing the MATLAB validation tests

There are two separate methods to run the Tool Validation Planning Kit.

1.) Use the Tool Validation Kit CLI. The getting started guide for the toolbox provides documentation on using the Tool Validation Kit programmatically.

2.) Use the Tool Validation Kit App, which will be present in the MATLAB App gallery. Please refer to the Tool Validation Kit App Guide documentation for more information on using the app.

Validation report discussion

There are two reports that can be generated from the Tool Validation Kit:

1. Standard MATLAB Unit Test Framework report
2. Customized report using MATLAB Report Generator

For more information regarding the standard MATLAB Unit Test report, please refer the documentation on the matlab.unittest.plugins.TestReportPlugin.

The customized report using MATLAB Report Generator is broken up into 3 chapters:

1. Introduction - includes objectives, scope, and assumptions/exclusions
2. MATLAB Tool Validation Overview – Includes products tested, and tool validation results summary

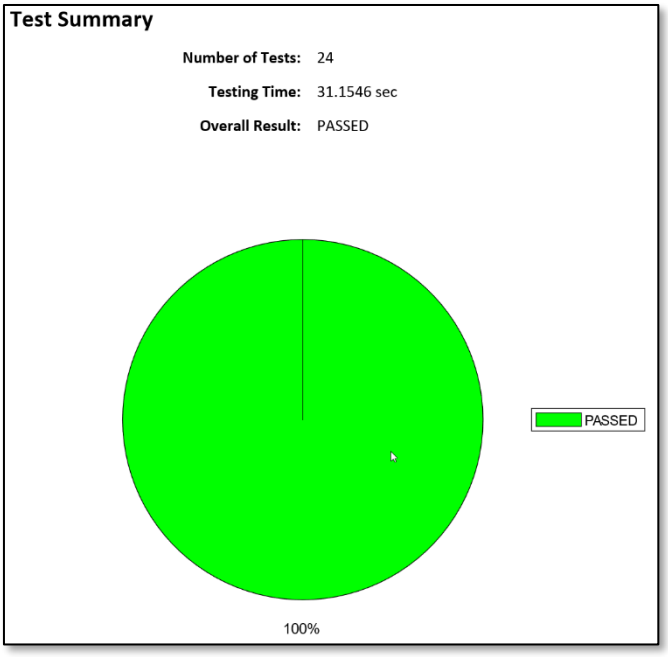
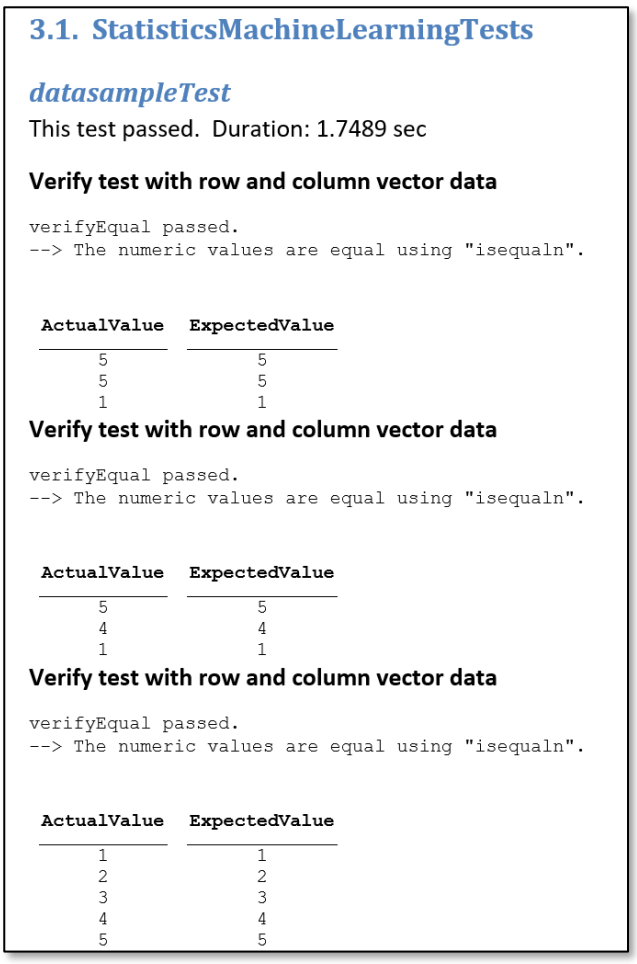


Figure X: Test Results Summary

Figure X shows a pie chart of the number of tests passed/failed, total testing time, and overall results.

3. Tool Validation Test Details – includes test details for all tests run for each test selected



3.1. StatisticsMachineLearningTests

datasampleTest

This test passed. Duration: 1.7489 sec

Verify test with row and column vector data

```
verifyEqual passed.
--> The numeric values are equal using "isequaln".
```

ActualValue	ExpectedValue
5	5
5	5
1	1

Verify test with row and column vector data

```
verifyEqual passed.
--> The numeric values are equal using "isequaln".
```

ActualValue	ExpectedValue
5	5
4	4
1	1

Verify test with row and column vector data

```
verifyEqual passed.
--> The numeric values are equal using "isequaln".
```

ActualValue	ExpectedValue
1	1
2	2
3	3
4	4
5	5

Figure X: Example Test Case Results

The above example shows the detailed results for the test case datasampleTest in the StatisticsMachineLearningTests suite. In addition to the expected results, actual results, and tolerance used, the report can capture a snapshot of any figures that were created during the test and inline that as well.

Extending the MATLAB validation tests

The Tool Validation Planning Kit also contains a sample template to illustrate how to create additional test directories, suites and cases. The following sections will detail the necessary steps to add these extra tests.

Basic design description

The Tool Validation Planning Kit already contains multiple test suites. These Suites are contained in the Base, Signal Processing, Computer Vision, Deep Learning, Image Processing, and Stats and Machine Learning sub-folders.

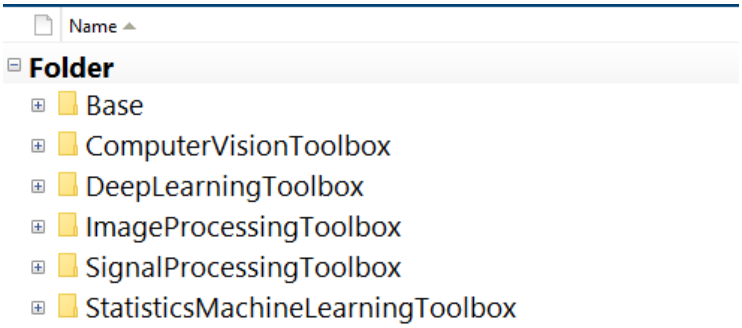


Figure X Test Suite Folders

Each of these folders contain .m files which implement the tests within that test suite. For example, if you proceed further down folder structure into the Base folder you will notice multiple test files.

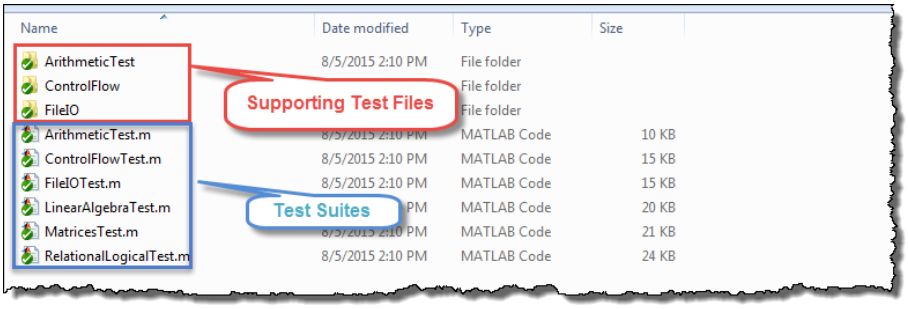


Figure X Folder Structure of Base Folder

In Figure 10 the folders ArithmeticTest, ControlFlow, and FileIO contain supporting files for the corresponding .m file. The Tool Validation Planning Kit will only look at the top level folders to find test cases. The next layer of subfolders can be used to place supporting files and will not impact the generation of the report.

Test framework template

Included in the kit is a file named TemplateTest.m. This file provides an example how to create an additional test suite and test cases. It is derived from the MATLAB unittest class, which is fully documented (run >> doc('unit testing framework') to learn more). However, you can take advantage of this framework with only limited changes to the template test suite file. To create a new test suite, complete the following steps.

1. Copy and Rename TemplateTest.m into its own subfolder like the folders of Base and SignalProcessingToolbox.
2. After renaming TemplateTest.m to the new file name make sure to change the .m file in the following places to match the new file name.

```
classdef TemplateTest < matlab.unittest.TestCase
    % TEMPLATETEST is an example validation test case for some features
    % of the MATLAB language.  This sample is intended to show how the user
    % create additional tests to be executed.
    %
    % Copyright 2022 The MathWorks, Inc.

    %% Properties
    properties
       %This section can be used to define test properties
    end

    %% Class and Method Setup
    methods(TestClassSetup)
        %This section can be used to define actions to execute before all
        %tests.
    end %methods

    methods(TestMethodSetup)
        %This section can be used to define actions to execute before each
        %test.
    end %methods

    methods(TestMethodTeardown)
        %This section can be used to define actions to execute after each
        %test.
    end %methods

    methods(TestClassTeardown)
        %This section can be used to define actions to execute after all
        %tests in this class have been completed.
    end %methods

```

Figure X Updating test suite Class Name

3. Add Test Cases to the test suite. In the TestTemplate.m file there are two separate example test cases. One test case will pass and one will fail if run. In sampleTestPoint1 we use verifyEqual to check the expected value to the actual value. If these two values do not match the test point will be marked as a failure.

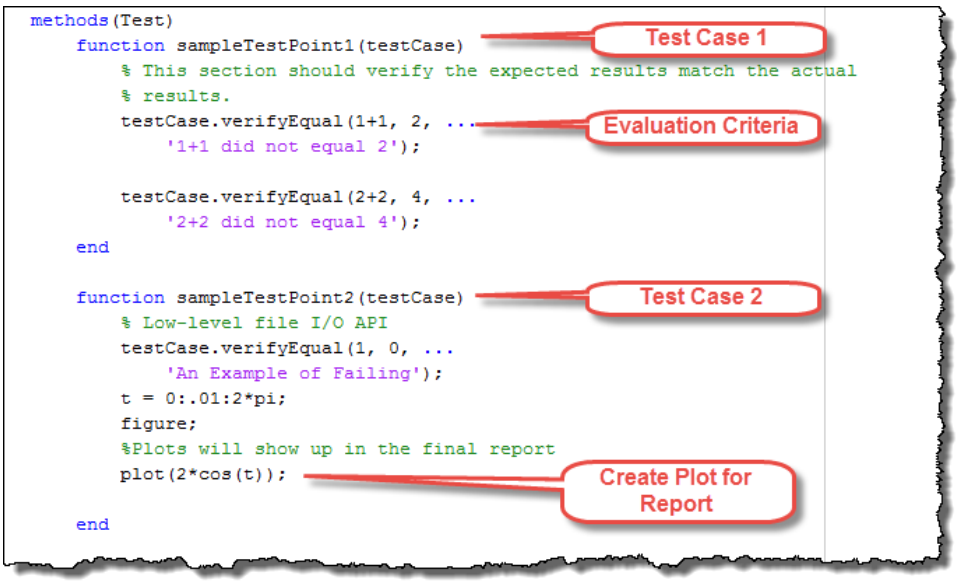


Figure X. Area to add/modify test cases

Note: sampleTestPoint2 demonstrates an example test case that plots an expected result. Anytime that the plot command is used in one of the test cases the corresponding plot will be captured in the resulting report.

4. Once you have created your new test suite(s), change the MATLAB working directory to the new suite directory you previously created. You should try executing your tests from MATLAB to debug any issues before running via the report generator. (Error messages, etc are not reported to the user when the report is running.) Use runtests('<YourTestFileName.m>') and observe the results. If the test results and optional figures appear, you are ready to change the directory back to the installation folder for the kit and run the report again. Your test results should appear in alphabetical order based on the test directory name then test suite file name.

NOTE: An alternative way to create new tests is to use the "createNewTest" function shown below:

```
filename = "AmazingTests.m";
createNewTest(filename);
```

This function will automatically copy the test template to the current folder, and rename the test class file and all class name instances with the input file name. You can provide an optional second input to this function to specify the file path in addition to the file name.

Known limitations

Currently, verifyEqual, verifyTrue, verifyFalse, verifyClass, verifyLessThanOrEqual, verifyLessThan, verifyGreaterThan, verifyGreaterThanOrEqual, verifySize, verifyWarningFree, and verifyError are the only unittest test case evaluation method that is fully supported. Please use this method for all test evaluations.

The MATLAB Validation Report template currently assumes that licenses are available for all of the products under test. To remove a test suite (such as SignalProcessingToolbox), simply use the TVK App or TVK CLI to select the appropriate tests to run.