STEP PROJECT REPORT

ABSTRACT

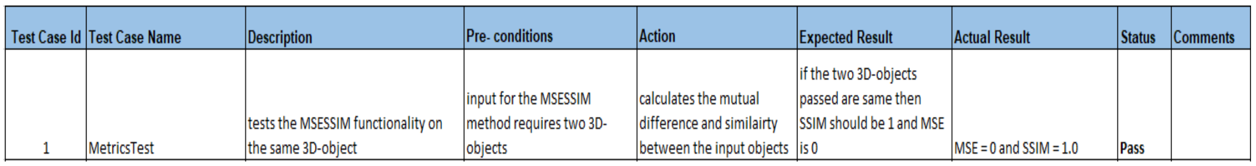
The goal of this STEP project is to create a testing framework to test the MATLAB code of computational imaging of microscopy. The computational imaging process in microscopy includes different stages: data collection from a microscope, data pre-processing, image reconstruction, and image post-processing. This report contains the details of testing framework created to test the project.

INTRODUCTION

Scientific applications perform numerical simulations of different natural phenomena in the filed of computational physics and chemistry, mathematics, informatics, mechanics, bioinformatics, etc. They are primarily developed for the scientific research community and usually they are used for simulating some I/O and data extensive experiments [Segal 2005]. The performance of such simulation experiments requires powerful supercomputers, high performance, or Grid computing [Vecchiola et al. 2009].

A scientific application is a software applications which turns the object into mathematical models by simulating activities from the real world [Ziff Davis Publishing Holdings 1995]. Scientific applications are different from commercial application, especially in the process of testing and evaluation. They are developed for the specific research community and not evaluated by customers. Testing is usually performed by comparing the obtained results with theory or performed physical experiments. Also, scientists can decide for the correctness of the results visually, by comparing images.

In this paper we include the software engineering practices in scientific application testing. For each functional requirement of the scientific application we design test cases, before testing the functionality. A test case definition consists of a unique identifier, name, goal, preconditions, execution environment, expected and actual results, status (passed/failed) and history of changes as shown below:



Test Case Id	Test Case Name	Description	Pre- conditions	Action	Expected Result	Actual Result	Status	Comments
1	MetricsTest	tests the MSESSIM functionality on the same 3D-object	input for the MSESSIM method requires two 3D-objects	calculates the mutual difference and similairty between the input objects	if the two 3D-objects passed are same then SSIM should be 1 and MSE is 0	MSE = 0 and SSIM = 1.0	**Pass**	

Table 1: An Example of Test Case Definition

APPLICATION

A testing framework will be created for the scientific application called 3D-SIM (3D Structured Illumination Microscopy). Its goal is to make the collected microscopy images more clearer using computational imaging and obtain desired images with increased information content. To achieve this, we model the microscopy data to a simulated object and then apply algorithms and do a performance analysis.

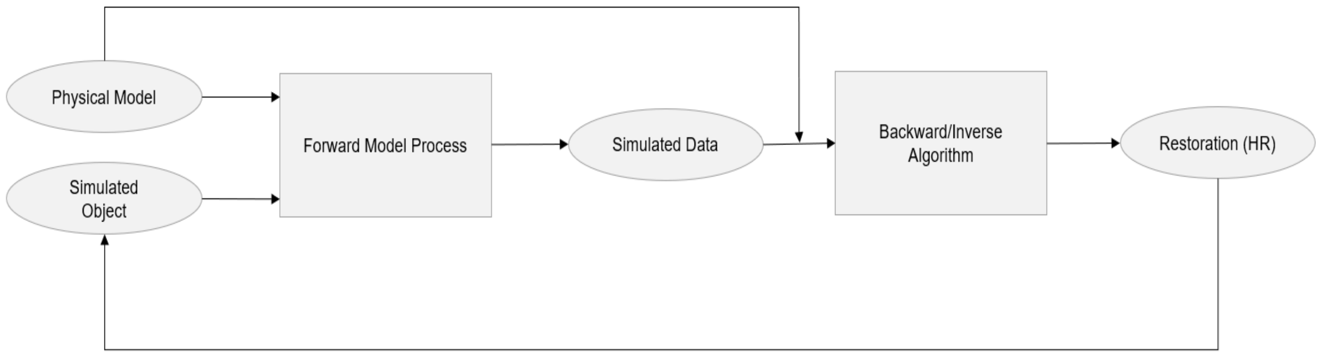


Figure 1: SIM Integration Test Flow

Physical Model creates bead using the settings data for the 3-slit Tunable system.

Simulated Object is the original image from the microscopy that is constructed through computational imaging using the settings data.

Forward Model process takes physical model and simulated object as inputs to produce simulated data as output.

Simulated Data is used to test the model and given as input to the backward algorithm.

Backward/Inverse Algorithm uses the raw data settings and simulated data to generate multiple intermediate images and then generates the final restoration image.

Restoration object is compared with the simulated object and the process is repeated until the desired results are achieved.

Images collected by 3D-SIM are modeled as:

$g = Hf$; where H is the forward model, g is the data of size [X, Y, Z] and f is the object of size [2X, 2Y, 2Z].

Inverse problem: given the data g, restore the object f.

Optimization-based solution: one can solve the above inverse problem by solving the optimization problem:

$f_r = H^{-1}g := \text{argmin}_f (||Hf - Ug|| + R(f))$

where f_r is the restoration, U is the upsampling operator/function, R is the regularization, and H^{-1} is the inverse operator/function.

TESTING FRAMEWORK

A testing framework is created for the application 3D-SIM which creates computational Imaging for 3D- Microscopy. Using this framework, we are going to test whether the model can produce desired images which are better than collected microscopy images.

Before testing the required test cases need to be defined in similar format as mentioned in the Table 1.

For the microscopy imaging process deals with 2D and 3D images, the metrics mean square error (MSE) and the structural similarity index measure (SSIM) can be used in the tests to validate the results.

The testing framework should be created as defined below.

Testing framework: to test the optimization-based solution, one creates some simulated object f_s, applies H to f_s to obtain the simulated data g_s, then uses optimization-based solution to obtain the restoration f_r. One then checks whether f_r is the same as f_s. In other words, the input and output of this testing framework should be:

Input:

- Settings object: containing all the parameters of the inverse problem, i.e. in SIM, the parameters are wavelength, NA, etc.
- Dimensions [X, Y, Z]
- Forward function H
- Restoration function H^{-1}
- Simulation function S to generate simulated data f_s

Output:

- Relative error between the restoration f_r and the simulated object f_s: $||f_r - f_s||/||f_s||$.

The testing framework is independent of the programming language used. For convenience, we will build this testing framework in MATLAB and make sure the MATLAB codes work correctly for the flowchart in Figure 2 below. Then we will translate the MATLAB part of the reconstruction into high-performance languages, such as, Python or Julia on HPC clusters for much better performance. The correctness of the Python or Julia codes on HPC clusters can be guaranteed by the unit-tests using the MATLAB codes' results since the MATLAB codes would have already been tested and verified.

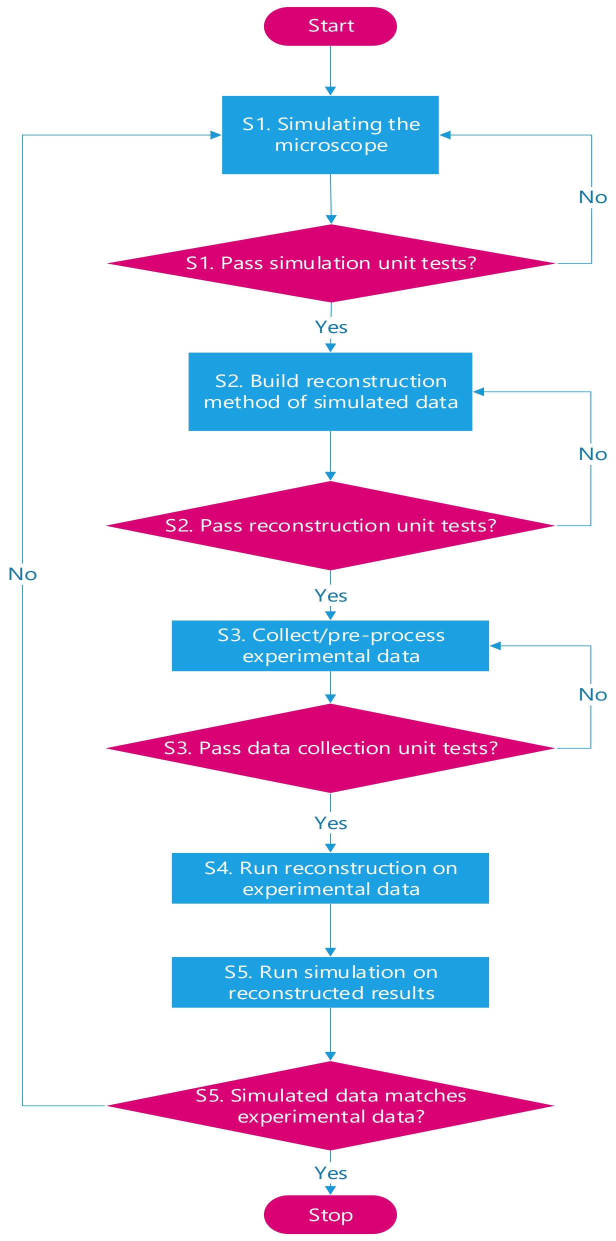

Start
S1. Simulating the microscope
S1. Pass simulation unit tests?
No
Yes
S2. Build reconstruction method of simulated data
S2. Pass reconstruction unit tests?
No
Yes
S3. Collect/pre-process experimental data
S3. Pass data collection unit tests?
No
Yes
S4. Run reconstruction on experimental data
S5. Run simulation on reconstructed results
S5. Simulated data matches experimental data?
No
Yes
Stop

Figure 2: Flowchart of testing framework for computational imaging in microscopy.

EXPERIMENTS

As part of the testing framework for the application 3D-SIM below changes were done in the code:

- **Settings** data structure object is created to include all the parameters of microscopy data that is needed for the inverse algorithm. It is created so that all the parameters can be easily used in different functions and the changes made in any code reflects in the object. Main purpose is other applications can also use the existing code functionality by just passing the Settings object.

- **InverseTest** contains complete functionality of the application which includes the forward model process and restoration process for a spherical object. "**TestInverseTest.m**" is created to test the InverseTest.
- **Metrics Test** calls SphericalShell function and the inbuilt MATLAB function MSESSIM. "**TestMetricsTest.m**" unit test is created to test MetricsTest to validate the MSESSIM function works as expected when a same 3D-object is passed.
- **PatternTunableTest** unit test is created to test the implementation of the pattern function which includes a call to PatternTunable3DNSlits function.
- **ForwardModelTest unit test** is created to test the functionality of ForwardModel function.
- **SimTunableStarLikeMB** contains complete functionality of the application which includes the forward model process and restoration process for star shaped object which is a larger object than spherical object. **HPCSimTunableStarLikeMB.sh** script is created to test SimTunableStarLikeMB on HPC.

Settings data used in unit tests are not the original data values but are simple integer values used to test the code for easy computations, whereas Settings used in InverseTest and Scaling Test are actual microscopy data.

Scaling of the inverse problem is tested on different input objects to see how well the model scales. It is tested on HPC(High Performance Computing) because of the limitations of local test not enough memory space and takes too long to compute. Using HPC repetitive simulations can be tested and the computation is faster with parallel computing. On HPC test the number of iterations value is much higher in Settings than in the local test.

Test cases created as part of the testing framework are described in the APPENDIX section.

Note: To run tests on HPC created an HPC account and installed software's Putty, WINSCP and VPN to access the HPC from home network.

GitHub is an open source development platform where one can create their own account and access repositories which are public. The testing framework is hosted in GitHub repository at https://github.com/congUoM/InverseTest.

PYTHON MATLAB INTEGRATION

Tested **MATLAB Integration with Python** before creating the testing framework in Python. First used Google Colab Notebook to test Python calling MATLAB function as it is much easier to share the testing framework with others. Installed required packages and tried in different ways but MATLAB calling was not successful from it.

Later, Installed Python setup under matlab external packages using Anaconda prompt on local system. Using Jupyter Notebook able to start and stop the matlab engine from Python and was able to call simple Matlab function Add and use the results in Python.

Tried to create the Settings data structure in python and passed to Matlab function but it did not work as Matlab does not support Python struct and dataframe objects. Then created parameters as variables in python and then converted to Matlab double types to pass it to Matlab functions.

Was able to call SphericalShell matlab function from Python but unable to plot the 3D image of the output object in the python as '<' is not supported between instances of 'tuple' and 'int’. Tried to Run Forward Model without displaying the results of before functions from Python but it was erroring out in PatternTunable3DNSlits matlab function because of the “Undefined function 'Visibility' for input arguments of type 'double'” error.

Finally concluded that building the Testing Framework in Python is not possible because of data inconsistencies between Matlab and Python and also Matlab result objects are unsupported.

APPENDIX

Test 3 in the Table 2 corresponds to step S1 in Figure 2, Test 4 in the Table 2 is corresponding to step S1, S2, S3 in Figure 2 and Test 6 in the Table 2 corresponds to steps S1, S2, S3, S4, S5 in Figure 2.

Table 2: Test Cases for the testing framework

Test Case Id	**Test Case Name**	**Description**	**Pre-conditions**	**Action**	**Expected Result**	**Actual Result**	**Status**
1	MetricsTest	tests the MSESSIM functionality on the same 3D-object	input for the MSESSIM method requires two 3D-objects	calculates the mutual difference and similarity between the input objects	if the two 3D-objects passed are same then SSIM should be 1 and MSE is 0	MSE = 0 and SSIM = 1.0	**Pass**
2	PatternTunableTest	generates two pattern components of 3-slit tunable simulation and compares the jm pattern with expected value	input is the Settings data structure	calculates the im and jm values	jm(1,2,1,1,1,2) == cos(pi)	jm(1,2,1,1,1,2) == cos(pi)	**Pass**
3	ForwardModelTest	compares the forward model output with expected value using convolution	input is the Settings data structure	generates "g" 5D matrix using Spherical Shell object and	MSE = 0 , SSIM = 1	MSE = 0.0140, SSIM = 0.8389	**Diff**

				the high resolution PSF "h" and modulating patterns "im , jm"			
4	InverseTest	constructs the restoration object and compares the simulated object with restoration object	input is the Settings data structure	generates the restoration object with 10 iterations	MSE = 0 , SSIM = 1	MSE = 0.0055, SSIM = 0.8994	**Diff**
5	InverseTest on HPC	constructs the restoration object and compares the simulated object with restoration object	input is the Settings data structure	generates the restoration object with 200 iterations	MSE = 0 , SSIM = 1	MSE = 3.9908e-05, SSIM = 0.9993	**Diff**
6	HPCSimTunableStarLikeMB	constructs the restoration object for StarLikeSample and compares with Star object, also generates matrix passing restoration image as input to forward model and compares with the previous output of forward model	input is the Settings data structure	generates the restoration object with 10 iterations	MSE = 0 , SSIM = 1	MSE = 0.0025, SSIM = 0.6524	**Diff**
				simulates the restoration object which was obtained after 10 iterations	MSE = 0 , SSIM = 1	MSE = 4.7556e-07, SSIM = 0.9990	**Diff**