Project Title: System Verification and Validation Plan for SPDFM

S. Shayan Mousavi M.

October 29, 2020

1 Revision History

Date	Version	Notes
Oct 17 2020	1.0	First Draft

Contents

List of Tables

List of Figures

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
VnV	verification and validation
SPDFM	Surface Plasmon Dynamics Finite Method

The complete table of symbols, abbreviations and acronyms can be found in the SRS document of the software.

This document provides the information on validation and verification plans implemented for the SPDFM software. In this regard,the general approaches and plans are initially discussed and afterwards specific test cases and approaches for validation and verification of functional and nonfunctional requirements (can be found in SRS) are reviewed. VnV plans here are a combination of manual (assigned to a member of the VnV team to assess) and automated testing approaches to evaluate the correctness of the information (whether input or output) or satisfaction of a goal in SPDFM.

3 General Information

3.1 Summary

The SPDFM software, which its VnV plan is discussed in this document, is a software for calculating plasmon-enhanced electric field and electric current in a meshed geometry. This software should be able to setup an optical source (given the related parameters) and study how electric field and current densities in the dielectric are affected. The calculations in this software are based on the newly established theory of surface plasmon oscillations, nonlocal hydrodynamic theory of surface plasmons Hiremath et al. (2012).

3.2 Objectives

This document tries to address the most important areas of the SPDFM software that can act like bottle necks of the system and make sure these areas function properly. In this regard, following chapters will discuss how these key aspects which include, setting up a light source, having properly meshed geometry, well defined dielectric environment, and theoretical formulation used. These areas are reflected within the functional and nonfunctional system requirements in Chapter 5 of the SRS document. In this software some python libraries are implemented such as FEniCS and Meshio which their validity is accepted will not be checked here.

3.3 Relevant Documentation

The relevant documentation for SPDFM, including the problem statement (Mousavi (2020a)), SRS (Mousavi (2020b)), SRS checklist (Smith (2020)),

VnV report, MG, and MIS can be found in the devoted GitHub repository to this software. For theoretical aspects Hiremath et al. (2012), Monk et al. (2003), and Maier (2007) are the major important resources used in this software.

4 Plan

4.1 Verification and Validation Team

The VnV team in this work includes S. Shayan Mousavi M., responsible for reviewing all the documentations, providing test cases and their execution, verifying the theoretical aspects and their implementation; Dr. Spencer Smiths (CAS741 instructor), and S. Parsa Tayefeh Morsal (domain expert), responsible for reviewing the design of the software, all documentations, and the documenting style itself; Naveen Ganesh Muralidharan (secondary reviewer) responsible for reviewing SRS document; Siddharth (Sid) Shinde (secondary reviewer) responsible for reviewing VnV document; Gabriela Sánchez Díaz (secondary reviewer) responsible for reviewing MG and MIS documents. The theoretical aspects and finite element method implemented in SPDFM is verified by Dr. Gianluigi Botton (supervisor), and Dr. Alexander Pofelski (field expert contributor).

4.2 SRS Verification Plan

The SPDFM shall be verified in the following ways:

Initial reviews from assigned members of the VnV team (Dr. Spencer Smith, S. Parsa Tayefeh Morsal, Naveen Ganesh Muralidharan, and Shayan Mousavi). In this regard, the document shall be manually reviewed using the SRS checklist (Smith (2020)) upon its initial version.

Secondary review by the author (Shayan Mousavi). The SRS document shall be reviewed after receiving initial reviews, and completion of VnV document.

Final review by the author (Shayan Mousavi) and the instructor (Dr. Spencer Smith). The document shall be manually reviewed according to the SRS checklist (Smith (2020)) after MG and MIS development.

Review of theoretical aspects by the field experts (Dr. Gianluigi Botton, Dr. Alexander Pofelski, and Shayan Mousavi). Theories used in the SRS document shall be reviewed manually with respect to the governing relations in the realm of plasmonic physics (Maier (2007), Hiremath et al. (2012), Monk et al. (2003)).

Feedback received from interested contributors through issue tracker in GitHub platform will also be used to improve this document.

4.3 Design Verification Plan

The design shall be verified by ensuring that key aspects in SPDFM are, as listed in Section 3.2. In this regard, the system functional requirements shall be tested initially, as outlined in 5.1, and in following the nonfunctional requirements will be review, as outlined in 5.2.

4.4 Implementation Verification Plan

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

The implementation shall be verified in the following ways:

Code Walkthrough by author: Module unit code shall be inspected for functional errors by the author (Shayan Mousavi) after MIS document is developed.

Code Walkthrough by contributors: Module unit code shall be inspected for functional errors by Dr. Alexander Pofelski after MIS document is developed.

System Tests: System tests will be carried out as listed in Section 5. The functional and nonfunctional requirements shall be by those listed in the outline of each test. These tests are conducted either manually or automatically which is outlined for each test individual in Chapter 5.

4.5 Automated Testing and Verification Tools

By predetermining some user inputs, an automated testing approach can be considered. The most of the functional requirement system tests will be executed and validated automatically; these tests are outlined in Section 5.1. The goal of testing is 100% code coverage. However, some system tests () require at least a manual assessment of the test results and are thus removed from the automated scope.

4.6 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS] Metallic Nanoparticle

5 System Test Description

5.1 Tests for Functional Requirements

The subsections below are designed to cover all the functional requirements of the system which are listed in Chapter 5 of the SRS document. coverage of each functional requirement is indicated in each subsection.

5.1.1 Input Verification (and/or Validation) tests

This section covers verification and validation of the user's inputs which are reflected in R2, R4, R5.

Test R 2: Correctness of the light source related input data

1. **Test 1:** validity of **p** vector

   Control: Automatic

   Initial State: N/A

   Input: Incident light polarity: $\mathbf{p} = (a, 0, 0)$
   Direction: $\mathbf{d} = (0, 1, 0)$

Output: An error should be print out saying "Invalid parameter p: p should be a 3D vector of floating numbers"

Test Case Derivation: Physically **p** and **d** must be 3D vectors in $\Re^3$ which are perpendicular to each other.

How test will be performed: Using Pytest library in python Input Module will be fed with the input, **p** and **d** vectors, and the output will be checked (p_d_test.py). Shayan Mousavi is responsible for execution of this test.

2. **Test 2:** validity of **d** vector

Control: Automatic

Initial State: N/A

Input: Incident light polarity: **p** = (1, 0, 0)
Direction: **d** = (0, 0, x)

Output: An error should be print out saying "Invalid parameter d: d should be a 3D vector of floating numbers"

Test Case Derivation: Physically **p** and **d** must be 3D unit vector in $\Re^3$ which are perpendicular to each other.

How test will be performed: Using Pytest library in python Input Module will be fed with the input, **p** and **d** vectors, and the output will be checked (p_d_test.py). Shayan Mousavi is responsible for execution of this test.

3. **Test 3:** validation of **d** vector unity

Control: Automatic

Initial State: N/A

Input: Incident light polarity: **p** = (1, 0, 0)
Direction: **d** = (0, 1, 0.2)

Output: An error should be print out saying "Invalid parameter d: d should be a 3D unit vector of floating numbers"

Test Case Derivation: Physically **p** and **d** must be 3D vectors in $\Re^3$ which are perpendicular to each other.

How test will be performed: Using Pytest library in python Input Module will be fed with the input, **p** and **d** vectors, and the output will be checked (p_d_test.py). Shayan Mousavi is responsible for execution of this test.

4. **Test 4:** validation of **p** and **d** vectors orthogonality

   Control: Automatic

   Initial State: N/A

   Input: Incident light polarity: **p** = (1, 0, 0)
   Direction: **d** = (1, 0, 0)

   Output: An error should be print out saying "Invalid parameters p and d: orthogonality of p and d vectors is not followed"

   Test Case Derivation: Physically **p** and **d** must be 3D vectors in $\Re^3$ which are perpendicular to each other.

   How test will be performed: Using Pytest library in python Input Module will be fed with the input, **p** and **d** vectors, and the output will be checked (p_d_test.py). Shayan Mousavi is responsible for execution of this test.

5. **Test 5:** validation of accepting correct p and d input

   Control: Automatic

   Initial State: N/A

   Input: Incident light polarity: **p** = (1, 0, 0)
   Direction: **d** = (0, 1, 0)

Output: This message should be appear: "Valid polarity and direction vectors" and software should move on

Test Case Derivation: User will be informed of correctness of the input.

How test will be performed: Using Pytest library in python Input Module will be fed with the input, **p** and **d** vectors, and the output will be checked (p_d_test.py). Shayan Mousavi is responsible for execution of this test.

6. **Test 6:** validation of wavelength input

   Control: Automatic

   Initial State: for having the code moving forward and accepting the wavelength value, an acceptable set of **p**, and **d** data should be already given to the software.

   $\mathbf{p} = (1,0,0)$
   $\mathbf{d} = (0,1,0)$

   Input: wavelength = 100

   Output: An error should be print out saying "Out of range wavelength input: Input wavelength should be in nm and between 200 nm to 6000 nm"

   Test Case Derivation: SPDFM only accepts wavelength values for the light source between 200 nm to 6000 nm as the surface plasmonic behaviour of most of the known materials is observed in this energy range (Table 1, Section 4.2.6, SRS document).

   How test will be performed: Using Pytest library in python Input Module will be fed with the inputs, **p**, **d**, and the wavelength value then the output will be checked (wavelength_test.py). Shayan Mousavi is responsible for execution of this test.

7. **Test 7:** validation of wavelength input

Control: Automatic

Initial State: for having the code moving forward and accepting the wavelength value, an acceptable set of **p**, and **d** data should be already given to the software.

$$\mathbf{p} = (1,0,0)$$
$$\mathbf{d} = (0,1,0)$$

Input: wavelength = 6200 Output: An error should be print out saying "Out of range wavelength input: Input wavelength should be in nm and between 200 nm to 6000 nm"

Test Case Derivation: SPDFM only accepts wavelength values for the light source between 200 nm to 6000 nm as the surface plasmonic behaviour of most of the known materials is observed in this energy range (Table 1, Section 4.2.6, SRS document).

How test will be performed: Using Pytest library in python Input Module will be fed with the inputs, **p**, **d**, and the wavelength value then the output will be checked (wavelength_test.py). Shayan Mousavi is responsible for execution of this test.

8. **Test 8:** validation of wavelength input

Control: Automatic

Initial State: for having the code moving forward and accepting the wavelength value, an acceptable set of **p**, and **d** data should be already given to the software.

$$\mathbf{p} = (1,0,0)$$
$$\mathbf{d} = (0,1,0)$$

Input: wavelength = 700

Output: A confirmation massage should be showed up: "The input wavelength is valid" and software must move on.

Test Case Derivation: SPDFM only accepts wavelength values for the light source between 200 nm to 6000 nm as the surface plasmonic behaviour of most of the known materials is observed in this energy range (Table 1, Section 4.2.6, SRS document).

How test will be performed: Using Pytest library in python Input Module will be fed with the inputs, **p**, **d**, and the wavelength value then the output will be checked (wavelength_test.py). Shayan Mousavi is responsible for execution of this test.

Test R 4: Correctness of the materials properties data entry

1. **Test 9:**

   Control: Automatic

   Initial State:

   Input:

   Output: "FEniCS is engaged" if the the input data are valid otherwise specifying the issue[The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

5.1.2 Output Verification (and/or Validation) tests

Test R 1: Software providing the user with information

1. **Test 1:** Instruction print out

   Control: Manual

   Initial State: N/A

   Input: N/A

   Output: "To initiate SPDFM software simulations user need to provide the software with following information:
   p = (p1, p2, p3) : 3D light source polarity vector, {p1, p2, p3} in R;
   d = (d1, d2, d3) : 3D light source propagation direction unit vector, {d1, d2, d3} in R;
   p and d must be perpendicular;

Wavelength = wl : light source spatial wavelength (nm), wl in R;
Frequency = freq : light source angular frequency (THz), freq in R;
Time step = delt_t : time step (fs), delta_t in R;
Final time = t_final : (fs), t_final in R;
Fermi velocity in the medium = v_f : (m/s), v_f in R;
Dielectric information;
Mesh file : address of the .gmsh file."

Test Case Derivation: As soon as the software is executed, the above paragraph should be shown to notify the user of the

How test will be performed: Shayan Mousavi is responsible to check and verify functionality of this feature.

Test R 2: Correctness of the light source related data entry

Test R 3: Correctness of the light source electric field calculation

1. **Test 11:** calculation of the light source electric field

   Control: Automated

   Initial State: **p**, **d**, and wavelength are previously given to the software.
   p=(1,0,0)
   d=(0,1,0)
   wavelength=700 nm
   The location vector, r, and time, t, will be given as below.

   Input: R = $\{\mathbf{r} = (0.1 * r_x, 0, 0) | \forall r_x \in \aleph, r_x \in [0, 10]\}$
   T = $\{t | \forall t \in \aleph, t \in (0, 10]\}$

   Output: For all the elements in sets R and T the output for the $\|E_i - E_i^t\|$ should be equal zero.

   Test Case Derivation: In this test E_i is the electric field of the incident beam calculated by the software and E_i^t is the electric field calculated in the test function following the equation $E_i^t = cos(k\ \mathbf{d}.\mathbf{r} - \omega t) - i\ sin(k\ \mathbf{d}.\mathbf{r} - \omega t)$

   How test will be performed: Using Pytest library in python. Codes can be found in **E_field_test.py** in the src folder. Shayan Mousavi is responsible for execution of this test.

2. **Test 12:** Light source electric field evolution

   Control: Manual

   Initial State: **p**, **d**, and wavelength are previously given to the software.
   p=(1,0,0)
   d=(0,1,0)
   wavelength=700 nm
   The location vector, r, and time, t, will be given as below.

   Input: $\mathrm{R} = \{\mathbf{r} = (0.1 * r_x, 0, 0) | \forall r_x \in \aleph, r_x \in [0, 10]\}$
   $\mathrm{T} = \{t | \forall t \in \aleph, t \in (0, 10]\}$

   Output: for t=0, plot of evolution of electric field by R. for r=(0, 0, 0), plot of evolution of electric field in time.

   Test Case Derivation: In this test $E_i = cos(k\ \mathbf{d}.\mathbf{r} - \omega t) - i\ sin(k\ \mathbf{d}.\mathbf{r} - \omega t)$ is plotted with respect to time and space and the examiner is responsible to see if the behaviour of the function is as expected or not.

   How test will be performed: Using Pytest library in python. Codes can be found in **E_field_plot_test.py** in the src folder. Shayan Mousavi is responsible for execution of this test.

5.1.3 Area of Testing2

...

5.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. —SS]

[Tests related to usability could include conducting a usability test and survey. —SS]

5.2.1 Area of Testing1

Title for Test

1. test-id1

   Type:
   Initial State:
   Input/Condition:
   Output/Result:
   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.
   Initial State:
   Input:
   Output:
   How test will be performed:

5.2.2 Area of Testing1

Title for Test

1. test-id1

   Type:
   Initial State:
   Input/Condition:
   Output/Result:
   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.2.3 Area of Testing2

...

5.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

6 Unit Test Description

[Reference your MIS and explain your overall philosophy for test case selection. —SS] [This section should not be filled in until after the MIS has been completed. —SS]

6.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

6.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

6.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

6.2.2 Module 2

...

6.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

6.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

6.3.2 Module ?

...

6.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Kirankumar R Hiremath, Lin Zschiedrich, and Frank Schmidt. Numerical solution of nonlocal hydrodynamic drude model for arbitrary shaped nano-plasmonic structures using nédélec finite elements. *Journal of Computational Physics*, 231(17):5890–5896, 2012.

Stefan Alexander Maier. *Plasmonics: fundamentals and applications*. Springer Science & Business Media, 2007.

Peter Monk et al. *Finite element methods for Maxwell's equations*. Oxford University Press, 2003.

S. Shayan Mousavi. Spdfm/docs/problemstatement at master · shmouses/spdfm · github. https://github.com/shmouses/SPDFM/tree/master/docs/ProblemStatement, 2020a. (Accessed on 10/29/2020).

S. Shayan Mousavi. Spdfm/docs/srs at master · shmouses/spdfm · github. https://github.com/shmouses/SPDFM/tree/master/docs/SRS, 2020b. (Accessed on 10/29/2020).

Spencer Smith. Blankprojecttemplate/docs/srs/srs-checklist.pdf · master · w. spencer smith / cas741 · gitlab. https://gitlab.cas.mcmaster.ca/smiths/cas741/-/blob/master/BlankProjectTemplate/docs/SRS/SRS-Checklist.pdf, Sept 2020. (Accessed on 10/29/2020).

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

7.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]