Project Title: System Verification and Validation Plan for EOMEE

Gabriela Sánchez Díaz

December 11, 2020

1 Revision History

Date	Version	Notes
29-10-2020	1.0	Created VnV first draft

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **v**

3 General Information **1**
3.1 Summary . . . 1
3.2 Objectives . . . 1
3.3 Relevant Documentation . . . 1

4 Plan **2**
4.1 Verification and Validation Team . . . 2
4.2 SRS Verification Plan . . . 2
4.3 Design Verification Plan . . . 2
4.4 Implementation Verification Plan . . . 3
4.5 Automated Testing and Verification Tools . . . 3
4.6 Software Validation Plan . . . 3

5 System Test Description **4**
5.1 Tests for Functional Requirements . . . 4
5.1.1 Input Verification . . . 4
5.1.2 Energies and TDMs Evaluation . . . 6
5.2 Tests for Nonfunctional Requirements . . . 9
5.2.1 Usability verification . . . 9
5.2.2 Performance . . . 10
5.3 Traceability Between Test Cases and Requirements . . . 10

6 Unit Test Description **10**
6.1 Unit Testing Scope . . . 11
6.2 Tests for Functional Requirements . . . 11
6.2.1 Input Module . . . 11
6.2.2 Integrals Module . . . 14
6.2.3 Density Module . . . 15
6.2.4 EOM IP Module . . . 16
6.2.5 EOM EA Module . . . 16
6.2.6 EOM Excitation Module . . . 17
6.2.7 EOM DIP Module . . . 17
6.2.8 EOM DEA Module . . . 18

6.2.9 Solver Module 19
6.2.10 Output Module 20
6.3 Tests for Nonfunctional Requirements 21
6.3.1 Module ? 21
6.3.2 Module ? 22
6.4 Traceability Between Test Cases and Modules 22

7 Appendix **23**
7.1 Symbolic Parameters 23
7.2 Usability Survey Questions 23
7.3 SRS Questionnaire 23

List of Tables

1 Tests to check that incorrect input parameter types are detected. Columns S1 and S2 specify two test cases. The last row, labeled as “Wrong type”, indicates the incorrect type of input that shall raise a type-error exception. 4
2 Tests to check that incorrect input values are detected. Columns S1 to S5 specify the test cases. The last row, labeled as “Wrong ”, indicates the incorrect input value that shall raise a value-error exception. 5
3 Input variables. The reference energy values (last column) are in Hartree. 7
4 Frontier molecular orbital energies in Hartree. 7
5 Traceability for system tests and requirements 11
6 Common parameters used by Input module tests. 12
7 Input module stored parameters. Column 1 specifies the variables, columns C1 and C2 their expected values for each test case. 12
8 Invalid input values. Column 3 specifies the expected raised exception. 13
9 Parameters used by all Output module tests. 20

10 Input parameters for the Output module tests. Column 1 specifies the variables, columns C1 to C3 their values in each test case. #files are the expected number of generated output files. #lines are the expected number of lines printed in the output file. 21

List of Figures

2 Symbols, Abbreviations and Acronyms

symbol	**description**
T	Test
VnV	Verification and Validation
DD	Data Definition
EOMEE	Equation-of-motion for excited states
IM	Instance Model
MO	Molecular orbital(s)
HOMO	Highest Occupied Molecular Orbital
LUMO	Lowest Unoccupied Molecular Orbital
R	Requirement
SRS	Software Requirements Specification
MG	Module Guide
MIS	Module Interface Specification
RDM	Reduced density matrix
TDM	Transition density matrix
2D	Two-Dimensional
4D	Four-Dimensional
$\mathbf{h}$	1-electron integral matrix (a 2D matrix);
	h_{pq} denotes its p,q-th element
$\mathbf{v}$	2-electron integral matrix (a 4D tensor);
	v_{pqrs} is its p,q,r,s-th element
$\boldsymbol{\gamma}$	1-electron reduced density matrix (a 2D matrix);
	γ_{pq} denotes its p,q-th element
$\boldsymbol{\Gamma}$	2-electron reduced density matrix (a 4D tensor)
	Γ_{pqrs} denotes its p,q,r,s-th element
$\otimes$	Kronecker product
idx	Index
ϵ	Energy (in Hartree) of a spinorbital

For more details about symbols, abbreviations and acronyms see Sections

1.2 and 1.4 in **?**.

This document describes the procedures to determine whether the specified requirements for EOMEE were satisfied [?]. It is organized as follows: Section 3 gives general information about the software. The project reviewers and verification plans for the documentation and implementation are introduced under Section 4. The system tests (black box tests) are specified in Section 5 including the verification of the functional and nonfunctional requirements. Section 6 describes the unit tests (white box tests). Finally, traceability matrices between system tests and requirements, and between unit tests and modules (see Module Guide [?]) can be found in Subsections 5.3 and 6.4, respectively.

3 General Information

3.1 Summary

The present document presents the validation plan of the package Equation of Motion for Excited States (EOMEE). This program is intended as a research tool for evaluating excited states (including ionized states) and their related spectroscopic properties such as the oscillator strengths. It also strives to assess the effect of the electron correlation on these properties.

3.2 Objectives

The present document's aim is to build confidence in the software by designing tests that verify its correctness (compliance with the requirements). Validation of the inputs and outputs or the code usability will be among the main tests this plan will be focusing on.

3.3 Relevant Documentation

Through this document we will be referring to the requirement specifications for EOMEE [?], in particular to its functional and nonfunctional requirements. Additionally, details about the module design can be found in the Module Guide (MG) [?] and Module Interface Specification (MIS) [?].

4 Plan

4.1 Verification and Validation Team

The review of EOMEE will be conducted by the following parties:

- The Ayers Lab members, in particular Michael Richer, Dr. Paul Ayers and Gabriela Sánchez Díaz.
- Dr. Spencer Smith.
- Members of 2020 CAS741 course; in particular the following responsibilities have been defined:

  Mohamed AbuElAla (Primary Reviewer)

  Seyed Parsa Tayefeh Morsal (SRS Reviewer)

  Ting-Yu Wu (VnV Reviewer)

  Xuanming Yan (MG and MIS Reviewer)

4.2 SRS Verification Plan

The Software Requirement Specifications (SRS) verification should be carried by the reviewers following the indications below:
The SRS document [?] will be provided to Dr. Ayers, Dr. Smith, Mohamed AbuElAla and Gabriela Sánchez along with a questionnaire (to be found in the Appendix section 7.3). The reviewers will read the document and questions and report any problems or inconsistencies as GitHub issues in the project's repository (cas741)

4.3 Design Verification Plan

The modules design (see MG [?] and MIS [?]) will be discussed with the review team members Dr. Paul Ayers and Michael Richer. It will also be presented to/evaluated by Dr. Smith and the members of 2020 CAS741 course. The review process will be guided by the MG and MIS checklists that can be accessed as part of the course materials (**?** and **?**, respectively). Design problems or inconsistencies will be reported as GitHub issues in EOMEE's repository (cas741).

4.4 Implementation Verification Plan

The implementation will be verified through system and unit testing. The following test areas will be defined:

- **Subsection** 5.1.1 Input Verification : includes checks for the input variables types and values.
- **Subsection** 5.1.2 Energies and TDMs Evaluation: tests in which the properties represented by the instance models (IM1-IM6) are evaluated and compared against literature or pseudo-oracle results (see **?** for details of the IMs).
- **Subsection** 5.2.1 Usability Verification Plan: tests with the purpose of identifying potential deficiencies in the user-software interaction.
- **Subsection** 5.2.2 Performance: Assess the numerical stability of the implemented IM1-IM5 [**?**].

[The corresponding sections for the unit tests will be added once that section of the VnV plan is completed —Author]

4.5 Automated Testing and Verification Tools

We will be using Pytest testing framework for tests automation with the plugin pytest-cov to determine the code coverage percentage. Additionally, the following tools will be used to enforce compliance with Python style conventions for code and documentation:

- Pylint, Flake8 and Black as Python linters. The last one is also a code auto-formatter.
- flake8-docstrings as a static analysis for Python docstring conventions.

For continuous integration we will use the Travis-CI tool.

4.6 Software Validation Plan

To assess the validity of EOMEE results, in particular during the system tests, these will be compared against the ones obtained with the package for electronic structure calculations Gaussian[**?**]. Another source for validation can be the NIST Atomic Spectra Database[**?**].

5 System Test Description

5.1 Tests for Functional Requirements

The tests described in the following subsections check functional requirements listed in the subsection 5.1 of the SRS, in particular requirements R1 and R2 about the input variables and R4 and R5 related to the spectroscopic properties evaluated by EOMEE.

5.1.1 Input Verification

Input type and value

1. input-type:IT

   Control: Automatic

   Initial State: N/A

   Input: According to Table 1, two sets of inputs (S1 and S2), corresponding to the one-electron integrals ($\mathbf{h}$), two-electron integrals ($\mathbf{v}$), one-electron reduced density matrix ($\boldsymbol{\gamma}$) and two-electron reduced density matrix ($\boldsymbol{\Gamma}$), will be provided:

Input set	**S1**	**S2**
$\mathbf{h}$	tuple	2-idx numpy array
$\mathbf{v}$	4-idx numpy array	4-idx numpy array
$\boldsymbol{\gamma}$	2-idx numpy array	2-idx list
$\boldsymbol{\Gamma}$	4-idx numpy array	4-idx numpy array
Wrong type	tuple variable	list variable

Table 1: Tests to check that incorrect input parameter types are detected. Columns S1 and S2 specify two test cases. The last row, labeled as "Wrong type", indicates the incorrect type of input that shall raise a type-error exception.

   The variables in the table will be defined as:
   tuple: ((1,2), (3,4))

A : [[1.0, 2.0], [3.0, 4.0]]
2-idx numpy array: numpy.asarray(A)
2-idx list: A
4-idx numpy array: numpy.asarray([A, A, A, A])

Output: A type-error will be raised in each case.

Test Case Derivation: N/A

How test will be performed: The test module will feed the inputs, as specified in Table 1, to the EOMEE code and verify that a type-error message is raised.

2. input-value:IV

Control: Automatic

Initial State: N/A

Input: Five input sets (S1 to S5) for the input parameters: one-electron integrals ($\mathbf{h}$), two-electron integrals ($\mathbf{v}$), one-electron reduced density matrix ($\boldsymbol{\gamma}$) and two-electron reduced density matrix ($\boldsymbol{\Gamma}$), as described in Table 2:

Input set	**S1**	**S2**	**S3**	**S4**	**S5**
$\mathbf{h}$	v_{pqrs}	h_{pq}	$A_{(n\times m)}$	$A_{(m\times m)}$	γ_{pq}
$\mathbf{v}$	h_{pq}	v_{pqrs}	$B_{(n\times m\times m\times m)}$	$B_{(n\times n\times n\times n)}$	Γ_{pqrs}
$\boldsymbol{\gamma}$	γ_{pq}	Γ_{pqrs}	$A_{(n\times m)}$	$C_{(n\times n)}$	h_{pq}
$\boldsymbol{\Gamma}$	Γ_{pqrs}	γ_{pq}	$B_{(n\times m\times m\times m)}$	$D_{(n\times n\times n\times n)}$	v_{pqrs}
Wrong	array dimensions		array shape		RDMs constrains

Table 2: Tests to check that incorrect input values are detected. Columns S1 to S5 specify the test cases. The last row, labeled as "Wrong ", indicates the incorrect input value that shall raise a value-error exception.

The variables in Table 2 will be defined as:
h_{pq}: numpy.random.rand(4, 4).astype(float)
v_{pqrs} and $B_{(n\times n\times n\times n)}$: numpy.random.rand(4, 4, 4, 4).astype(float)
γ_{pq} and $C_{n\times n}$: numpy.asarray([[1,0,0,0], [0,0,0,0], [0,0,1,0], [0,0,0,0]])

Γ_{pqrs} and $D_{(n\times n\times n\times n)}$: $\gamma_{pr} \otimes \gamma_{qs} - \gamma_{ps} \otimes \gamma_{qr}$
$A_{(n\times m)}$: numpy.random.rand(4, 3).astype(float)
$B_{(n\times m\times m\times m)}$: numpy.random.rand(4, 3, 3, 3).astype(float)
$A_{(m\times m)}$: numpy.random.rand(6, 6).astype(float)

Output: The EOMEE code must raise a value error for each case specified in Table 2.

Test Case Derivation: Input cases S1 and S2 target incorrect dimensions in the electron integrals or RDMs. S3 verifies that h_{pq} and $\gamma_p q$ are square matrices, while v_{pqrs} and Γ_{pqrs} must be tensors, each with equal numbers of rows and columns. Mismatching number of spin orbitals in input parameters (given by the array shapes) is verified by the test case S4. The last column input set targets the requirement R2 in the SRS relative to the physical constraints of the RDMs.

How test will be performed: The test cases in Table 2 are passed to the EOMEE code by the testing framework which will also verify that a value-error message is raised.

5.1.2 Energies and TDMs Evaluation

Transition energies

The tests described under this section require the one- and two-electron integrals for the N-electron, M-atom system to be stored in the directory *test/data* as NumPy's binary files (.npy). Unless otherwise stated, for each implemented EOM method (IM1-IM5, SRS subsection 4.2.5), the lowest (positive valued) transition energy will be taken and compared against an "expected" energy value (generally from a Gaussian[**?**] calculation). Because at this point we are not targeting the performance of the methods, only small systems with at most 5 electrons and 10 molecular orbitals (MOs) were included in the test cases bellow. We will use the absolute error ($|E_{IM\#} - E_{expected}|$) to measure the method's accuracy.

1. correct-lowest-transition:LT

   Control: Automatic

   Initial State: N/A

Input: The one- and two-electron integrals corresponding to the systems presented in Table 3, the tolerance value 1.0×10^{-6}.

Table 3 contains the test cases description. All systems have closed-shell electronic configurations except the boron atom (B). The ϵ_{HOMO} and ϵ_{LUMO} values for the atomic and molecular systems are reported in Table 4.

File	IM	**Nbasis**	**Nocc**	**Expected**
B (STO-3G)	1	5	(3, 2)	$-\epsilon_{HOMO}$
B (STO-3G)	2	5	(3, 2)	ϵ_{LUMO}
He (cc-pVDZ)	1	5	(1, 1)	$-\epsilon_{HOMO}$
He (cc-pVDZ)	2	5	(1, 1)	ϵ_{LUMO}
HeH^+ (STO-3G)	1	2	(1, 1)	$-\epsilon_{HOMO}$
HeH^+ (STO-3G)	2	2	(1, 1)	ϵ_{LUMO}
HeH^+ (STO-3G)	3	2	(1, 1)	0.91123209
H_2 (STO-6G)	4	2	(1, 1)	1.83843430

Table 3: Input variables. The reference energy values (last column) are in Hartree.

File	ϵ_{HOMO}	ϵ_{LUMO}
B (STO-3G)	-0.20051823	0.29136562
He (cc-pVDZ)	-0.91414765	1.39744193
HeH^+ (STO-3G)	-1.52378328	-0.26764028

Table 4: Frontier molecular orbital energies in Hartree.

Output: Eigenvalues (and eigenvectors) solution to the IMs.

Test Case Derivation: The one- and two-reduced density matrices required as input by the EOMEE methods will be generated from the number of electrons (Nocc) and MOs (Nbasis). Because the ground state wave function (Ψ_0) for the systems in Table 3 will be modeled by a single Slater determinant (specifically the lowest energy Hartree-Fock Slater determinant), the one- and two-RDMs can be defined as:

$$\gamma_{pq} = \begin{cases} \delta_{pq} & p, q \in \text{occupied MO} \\ 0 & \text{virtual MO} \end{cases}$$

$$\Gamma_{pqrs} = \frac{1}{\sqrt{2}}(\gamma_{pr}\gamma_{qs} - \gamma_{ps}\gamma_{qr})$$

How test will be performed: The testing framework will pass the electron integrals and density matrices to the EOM method (IM column in Table 3). From the solutions, the lowest transition energy will be determined and compared against the ”expected” value. If the absolute error is less or equal than the tolerance criteria, the test will pass.

Transition Density Matrices

1. matrix-symmetry:TS

   Control: Automatic

   Initial State: The EOM method (IM1-IM5) has been solved.

   Input: The following input arrays located in the *test/data* folder:

   - be_sto3g_twoint_genzd_anti.npy
   - 2dm_be_sto3g_genzd_anti.npy
   - be_sto3g_oneint_genzd.npy
   - 1dm_be_sto3g_genzd.npy

   The expansion coefficients matrix and the lowest nonzero excited state index.

   Output: The TDM for the selected state.

   Test Case Derivation: The equations to evaluate the TDMs corresponding to each EOM method can be found in the SRS (subsection 4.2.5, IM6).

   How test will be performed: The test framework will pass the specified inputs to a function that evaluates the EOMEE methods and TDMs. The selected TDM and its transpose will be compared using NumPy's allclose method. The test will pass if the matrix is asymmetric.

5.2 Tests for Nonfunctional Requirements

The tests described in the following subsections verify the nonfunctional requirements listed in the subsection 5.2 of the SRS.

5.2.1 Usability verification

Portability and usability tests

1. installability-test:PT

   Type: Manual

   Initial State: The may or may not be some Python program and modules already installed.

   Input/Condition: Internet connection and some web browser available.

   Output/Result: EOMEE module installed or issue reported in GitHub.

   How test will be performed: The GitHub repository will be cloned by Gabriela Sánchez in different platforms (Ubuntu Focal Fossa, Mac OS Catalina and Windows 10). The installation will follow the instructions in the README.md file located on the repository root folder. Upon installation completion the system and unit tests will be ran.

2. usability-test:UT

   Type: Manual.

   Initial State: The EOMEE code may or not be installed on the user's computer.

   Input/Condition: Members of the test team, the link to the project's GitHub repository and the following example input files for the beryllium atom:

   - be_sto3g_twoint_genzd_anti.npy
   - 2dm_be_sto3g_genzd_anti.npy
   - be_sto3g_oneint_genzd.npy
   - 1dm_be_sto3g_genzd.npy

These files will be located in the *test/data* folder.

Output/Result: Users response to the usability survey found in the Appendix section.

How test will be performed: The users will clone the repository and install the program. Using the provided input files they will choose to evaluate one of the following properties for Be: ionization potential, electron affinity or electron excitation. Additionally, they will determine the TDMs for their selected method. After completing these tasks the test team will fill out the survey.

5.2.2 Performance

Numerical instability

1. illconditioning-test:NIT

   Type: automatic

   Initial State: N/A

   Input/Condition: A matrix orthogonalization methods (strings *symmetric* or *asymmetric*), threshold values in the range $1.0 \times 10^{-4} - 1.0 \times 10^{-9}$, the example input files from the usability-test and the instance models IM1 to IM5 (SRS subsection 4.2.5).

   Output/Result: A graph showing how the lowest nonzero value for instance models IM1-IM5 changes with the tolerance value.

   How test will be performed: The test framework will pass the inputs to EOMEE and a graph will be generated.

5.3 Traceability Between Test Cases and Requirements

Table 5 shows what requirement specifications the functional and nonfunctional system tests address:

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]

	IT	IV	LT	TS	SQ	PT	UT	NIT
R1	X	X						
R2		X						
R3								
R4			X	X				
R5			X	X				
R6								
Reusable								
Usable						X	X	
Portable						X		
Correct			X	X	X			

Table 5: Traceability for system tests and requirements

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

6.2.1 Input Module

The interface of the Input module is described in [?]. It requires as input a file with line-separated instructions and stores the parsed parameters in a data structure (called ParsedParams in the MIS). The tests bellow aim at checking that the input file is parsed correctly (test_parse_inputfile) and that the correct input variables were specified (test_wrong_inputs). Table 6 lists common input variables between these tests. The parameters file1 and file2 correspond to sample input files that should be located under the package's *test/data* folder.

1. test_parse_inputfile

Parameters	Value
file1	test/data/test_input1.in
file2	test/data/test_input2.in
int1	be_sto3g_oneint_spino.npy
int2	be_sto3g_twoint_spino.npy
dm1	be_sto3g_onedm_spino.npy
dm2	be_sto3g_twodm_spino.npy
orthog	symmetric
eom	ip

Table 6: Common parameters used by Input module tests.

Type: Automatic

Initial State: N/A

Input: The files file1 and file2 from Table 6. Each file is a test case (C1 and C2 respectively).

Output: For tests C1 and C2 the output is a data structure storing the parsed input data as variables (ParsedParams). The expected stored values are reported in Table 7. The common variables between the two tests are listed in Table 6 (int1, int2, dm1, dm2, orthog, eom).

	Test Cases	
Parameters	C1	C2
nelec	(2, 2)	4
get_tdm	False	True
roots	None	1
tol	1.0e-7	1.0e-6

Table 7: Input module stored parameters. Column 1 specifies the variables, columns C1 and C2 their expected values for each test case.

Test Case Derivation: While the values for column C2 in Table 6 are parsed from the input file file2, the values for C1 are the defaults assigned by the module when their corresponding parameters aren't included in the input file (file1)

How test will be performed: The test framework will pass the input files to the module and verify that the expected outputs were obtained.

2. test_wrong_inputs

   Type: Automatic

   Initial State:

   Input: A list of input parameters as specified in Table 8:

Parameter	Wrong_Value	Exception
nelec	0.1	TypeError
tol	4	TypeError
roots	0.1	TypeError
get_tdm	yes	TypeError
h_file	temp.npy	FileNotFoundError
v_file	temp.npy	FileNotFoundError
dm1_file	temp.npy	FileNotFoundError
dm2_file	temp.npy	FileNotFoundError
orthog	ip	ValueError
eom	symmetric	ValueError

Table 8: Invalid input values. Column 3 specifies the expected raised exception.

   Output: An exception will be raised according to Table 8.

   Test Case Derivation: Starting from a list containing the valid parsed instructions from file2 (see Table 6 for its location), they are assigned incorrectly to ParsedParams variables as specified in Table 8.

   How test will be performed: The test framework will pass the input to the module and verify that the expected exceptions are raised.

6.2.2 Integrals Module

[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_load_integrals

   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_verify_integrals

   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. ...

6.2.3 Density Module

[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_assign_rdms

   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_verify_rdms

   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. ...

6.2.4 EOM IP Module

[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_one_body_term

   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:

6.2.5 EOM EA Module

[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_one_body_term

   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:

6.2.6 EOM Excitation Module

[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_one_body_term

   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:

6.2.7 EOM DIP Module

[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_one_body_term

   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_two_body_term

   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:

6.2.8 EOM DEA Module

[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_one_body_term

   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_right_hand_side

   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:

6.2.9 Solver Module

[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_dense_orthogonalization

   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_dense_tolerance

   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:

6.2.10 Output Module

The test cases under this section verify the Output module described in Subsection 16 of the MIS [?]. The module takes a filename, the energies and coefficients solution to the EOM eigenvalue problem, the corresponding TDM of the EOM method (this is optional) and a data structure storing input parameters (that will be called MockParsedParams). Three test cases have been considered whose target is to check that the expected output files are generated (in all cases a .out and .npz files must be produced). The number of lines in the .out file is also verified in each test case. This is particularly relevant when information about the transition energies is requested as part of the .ou file (trough the parameter "MockParsedParams.roots"). Table 9 lists common input variables between these tests.

Parameters	Description
filename	example_output.in
delta_E	[1.0, 2.0]
coeffs	[[1.0, 2.0], [1.0, 2.0]]
tdm	numpy.zeros((2, 2, 2))

Table 9: Parameters used by all Output module tests.

1. test_output_dump

   Type: Automatic

   Initial State: N/A

   Input: The parameters specified in Table 9 (which are common for all cases) and the inputs that change in each test, listed in Table 10 (C1-C3). Also, a MockParsedParams (structure that stores the usual inputs of EOMEE (see the Input module from MIS [?])) .

   Output: An .out output file, a NumPy .npz file containing the energies and coefficients, a NumPy .npy file with the TDMs for the case in Table 10 where a TDM was specified.

   Test Case Derivation: N/A

	Test Cases		
Parameters	C1	C2	C3
TDM	None	tdm	None
MockParsedParams.roots	None	None	1
#files	2	3	2
#lines	17	17	21

Table 10: Input parameters for the Output module tests. Column 1 specifies the variables, columns C1 to C3 their values in each test case. #files are the expected number of generated output files. #lines are the expected number of lines printed in the output file.

How test will be performed: The test framework will pass the inputs to the Output module.

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]

7 Appendix

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

1. Were the installation instructions for your platform clear? Could you successfully install EOMEE by following them?

2. Was the code's documentation helpful when learning how to run the calculations?

3. Did you look at the tests files when learning how to run the calculations? Were they helpful?

4. Could you successfully run an excited state calculation?

5. Could you successfully determine the transition density matrices for your chosen EOM method?

6. Was the naming of the functions descriptive (it was clear the task they were supposed to perform)?

7. Do you have any improvement suggestions regarding the documentation or code?

8. How would you rate the learning curve for this software?(Very easy, easy, hard, very hard)

7.3 SRS Questionnaire

Because the members of the VnV team might have limited time to review the document some of the question bellow have been specifically assigned:

1. Are the energy units in the Table of units (section 1.2) appropriate for the physical problems EOMEE addresses?

2. Was the Table of Symbols (section 1.2) complete?

3. Does the used notation in the Table of Symbols (section 1.2) matches the one generally used in the literature? [Dr. Ayers]

4. Were the abbreviations and acronyms (section 1.3) properly used through the document?

5. Is Figure 1 for the system context (section 3.1) along with its description correct? Do the listed user and software responsibilities make sense? [Dr. Smith and Mohamed AbuElAla]

6. Was the description of the problem EOMEE intends to solve clear? Were all the terminology definitions needed for the understanding of subsequent sections included? (sections 4.1 and 4.1.1) [Dr. Smith and Mohamed AbuElAla]

7. Were the terminology definitions listed in section 4.1.1 correct? [Dr. Ayers]

8. Is section 4.1.2 (Physical System Description) clear? Do you have any suggestions that might help improve the understanding of EOMEE's physical system? [Mohamed AbuElAla]

9. Were all assumptions pertinent to the EOM models included? Are the ones listed correct?(section 4.2.1) [Dr. Ayers]

10. Were the connections between the assumptions and the appropriate IM, GS, DD, T or LC included? (section 4.2.1)

11. Were the defined theoretical models the correct general equations that EOMEE is based on? Were the descriptions and derivations appropriate? (section 4.2.2) [Dr. Ayers]

12. In section 4.2.3, are the definitions of the transition operator and EOM approximations correct? Are the descriptions and derivations appropriate? [Dr. Ayers]

13. Are DD2, DD3, DD4 and DD5 properly described? Are the presented symmetry properties of DD3 and DD5 correct? (section 4.2.4) [Dr. Ayers]

14. Are the IMs inputs and outputs appropriate? (section 4.2.5) [Dr. Smith]

15. After reading the Input Data Constraints section (4.2.6), are the physical constraints presented in the Table 1 correct? Are the typical values assigned to the one- and two-electron integrals reasonable? What changes do you suggested making to the values presented in this table? [Dr. Ayers]
16. Do the output constraints presented in Table 3 (section 4.2.7) make sense? If not, could you suggest changes to be made? [Dr. Ayers]
17. Do the listed Functional and Nonfunctional Requirements make sense (sections 5.1 and 5.2 respectively) [Dr. Smith]
18. Were the cross-references made correctly through the document? [Mohamed AbuElAla]

These questions are inspired on the SRS review plans by Malavika Srinivasan and Spencer Smith[**?**].