Project Title: System Verification and Validation Plan for EOMEE

Gabriela Sánchez Díaz

December 12, 2020

1 Revision History

Date	Version	Notes
29-10-2020	1.0	Created VnV first draft
12-12-2020	1.0	Add Unit tests and make correction from feedback

Contents

List of Tables

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
VnV	Verification and Validation
DD	Data Definition
DEA	Double Electron affinity
DIP	Double Ionization Potential
EA	Electron affinity
EOMEE	Equation-of-motion for excited states
Exc	Excitation
IM	Instance Model
IP	Ionization Potential
MO	Molecular orbital(s)
HOMO	Highest Occupied Molecular Orbital
LUMO	Lowest Unoccupied Molecular Orbital
R	Requirement
SO	Spinorbital(s)
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

2.1 Notation

[This section was introduced to avoid issues with long tables splitting twrough pages. —Author]

symbol	**description**
$\mathbf{h}$	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
$\mathbb{R}^{m \times n}$	A 2D matrix with dimensions m, n
$\mathbb{R}^{m \times n \times l \times k}$	A 4D tensor with dimensions m, n, l, k

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: checks for valid input variables.
- **Subsection** 5.1.2 Energies and TDMs Evaluation: verification of the calculation results corresponding to IM1-IM6 (see Section 4.2.5 of the SRS **?**).
- **Subsection** 5.2.1 Usability Verification Plan: tests with the purpose of identifying potential deficiencies in the user-software interaction.
- **Subsection** 5.2.2 Reusability verification: check whether the modular decomposition of the software was done correctly.
- **Subsection** 5.2.3 Performance: check the numerical stability of the eigenvalue solution method for the implemented IM1-IM5 [**?**].
- **Subsection** 6.2.1 Input Module: Unit tests for the type and values of the inputs.
- **Subsection** 6.2.2 Integrals Module: Unit tests of electron integrals format.
- **Subsection** 6.2.3 Density Module: Unit tests of density matrices format.
- **Subsection** 6.2.4 Solver Module: Unit tests for the solver orthogonalization method.
- **Subsection** 6.2.5 Output Module: Unit tests for the output formats.

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 verify EOMEE results, in particular during the system tests, these will be compared against the ones obtained with the package for electronic structure calculations Gaussian [**?**]. Additionally, a 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

Valid and invalid inputs

1. valid_inputs: ST1
   Control: Automatic

   Initial State: N/A

   Input: The following sample input files stored under *test/data/*:

   - file1: test_input1.in
   - file2: test_input2.in
   - file3: test_input3.in
   - file4: test_input4.in
   - file5: test_input5.in

Each of this files uses a different EOM method as shown in column 3 of Table 1 The istegrals and RDMs used in these tests are the same as

Case	File	Method	Expected # outputs
C1	file1	IP	2
C2	file2	EA	3
C3	file3	Exc	2
C4	file4	DIP	2
C5	file5	DEA	2

Table 1: Test cases for valid input parameters. The last column indicates the expected number of files to be produced as output.

detailed in Table 2

Parameters	Value
oneint_file	test/data/be_sto3g_oneint_spino.npy
twoint_file	test/data/be_sto3g_twoint_spino.npy
dm1_file	test/data/be_sto3g_onedm_spino.npy
dm2_file	test/data/be_sto3g_twodm_spino.npy

Table 2: Common parameters between all input files.

Output: The number of output files specified in the last column of Table 1

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 the expected number of files is generated.

2. invalid_inputs: ST2

Control: Automatic

Initial State: N/A

Input: The following input files stored under *test/data/*:

- file1: test_input_mos.in
- file2: test_wrong_dms.in

The options inside each file that will cause the program to fail are specified next (Table 3):

File	Parameters	Value
file1	oneint_file	test/data/be_sto3g_oneint.npy
file1	twoint_file	test/data/be_sto3g_twoint.npy
file2	dm1_file	test/data/b_sto3g_onedm_spino.npy
file1	dm2_file	test/data/b_sto3g_twodm_spino.npy

Table 3: Incorrect parameters on each input files.

Output: A ValueError exception is raised

Test Case Derivation: In file1 the electron integrals are in the incorrect format (molecular orbital (MO) basis instead of the spinorbital one (SO)) (see Table 3), therefore their symmetry will not match the one expected by EOMEE. For the other input file (file2) the density matrices used correspond to a different system than the electron integrals, therefore the number of electrons associated with the trace of this matrices won't match the expected value (specified by nelec in the input file).

How test will be performed: The testing framework will pass the inputs to EOMEE and verify that a ValueError 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 MOs were included in the test cases below. We will use the absolute error ($|E_{IM\#} - E_{expected}|$) to measure the method's accuracy.

1. correct-lowest-transition: ST3

   Control: Automatic

   Initial State: N/A

   Input: The one- and two-electron integrals corresponding to the systems presented in Table 4, the tolerance value 1.0×10^{-6}.

   Table 4 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 5.

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 4: Input variables. The reference energy values (last column) are 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 4 will be modeled by

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 5: Frontier MO energies in Hartree.

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 4). 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: ST4

   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: ST5

   Type: Manual

   Initial State: There 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: ST6

   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 Reusability verification

1. reusability-test: ST7

   Type: Manual

   Initial State: An existing (working) implementation of EOMEE.

   Input/Condition: The input file test_input_mos.in (located under *text/data/*) This file has the usual inputs, but with the electron integral parameters pointing to the following files in MO basis set format instead of the SO one: *text/data/*

   - be_sto3g_oneint.npy
   - be_sto3g_twoint.npy

Output/Result: The program is ran the same way as before (passing an input file to a control module), but using electron integrals in the MO basis set.

How test will be performed: The Integrals module (Section 8 of MIS [**?**]) will be modified. Three new functions will be added, one to transform the integrals format from MO to SO basis, the other two to give the two-electron integrals the symmetry properties they should have to work with the EOM modules. If after the changes performed the code produces de same result as with the old file format (integral files be_sto3g_oneint_spino.npy and be_sto3g_twoint_spino.npy) the test is considered passed.

5.2.3 Performance

Numerical instability

1. illconditioning-test: ST8

   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 6 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

	ST1	ST2	ST3	ST4	ST5	ST6	ST7	ST8
R1	X							
R2		X						
R3		X						
R4			X	X				
R5			X	X				
Reusability							X	
Usability					X	X		
Portability					X			
Accuracy			X	X				X

Table 6: Traceability for system tests and requirements

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]

In the subsequent sections the unit tests for the modules specified in the MG and MIS are described (see **?** and **?**, respectively). The verification of the Control (M2) and EOM modules (M6-M11) is mainly performed through black box testing (refer to Section 5.1.2), therefore these modules are not considered in this part of the document. Similarly, the Hardware-Hiding module (M1) related to the operating system on which the program runs will not be verified.

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 module is able to parse an input file correctly (test_parse_inputfile) and that incorrect input variables are detected (test_wrong_inputs). Table 7 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.

Parameters	Value
file1	test/data/test_input1.in
file2	test/data/test_input2.in
oneint1	be_sto3g_oneint_spino.npy
twoint1	be_sto3g_twoint_spino.npy
onedm1	be_sto3g_onedm_spino.npy
twodm1	be_sto3g_twodm_spino.npy
orthog	symmetric
eom	ip

Table 7: Common parameters used by Input module tests.

1. test_parse_inputfile: UT1

   Type: Automatic

   Initial State: N/A

   Input: The files file1 and file2 from Table 7. 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 8. The common variables between the two tests are listed in Table 7 (oneint1, twoint1, onedm1, twodm1, orthog, eom).

   Test Case Derivation: While the values for column C2 in Table 7 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)

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

Table 8: Input module stored parameters. Column 1 specifies the variables, columns C1 and C2 their expected values for each test case.

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: UT2

   Type: Automatic

   Initial State:

   Input: A list of input parameters as specified in Table 9:

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

Output: An exception will be raised according to Table 9.

Test Case Derivation: Starting from a list containing the valid parsed instructions from file2 (see Table 7 for its location), they are assigned incorrectly to ParsedParams variables as specified in Table 9.

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

Section 8 of MIS [**?**] describes the interface of the Integrals module, which handles the format of the electron integrals. The unit tests bellow check that the input files are loaded correctly (test_load_integrals) and that integrlas with wrong properties be detected (test_check_invalid_integrals, test_verify _symmetry) . The parameters used for this tests are listed in Table 10. The data files with extensions .in and .npy should be located under the package's *test/data* folder.

Parameters	Value
file1	test/data/test_input1.in
oneint1	be_sto3g_oneint_spino.npy
twoint1	be_sto3g_twoint_spino.npy
oneint2	h2_hf_sto6g_oneint.npy
twoint2	h2_hf_sto6g_oneint.npy
A	$\mathbb{R}^{m \times m}$ (e.g. [[1,2], [3,4]])
	with the condition $A \neq A^T$
B	$\mathbb{R}^{m \times m \times m \times m}$ (e.g. np.arange(16).reshape(2, 2, 2, 2))
	with the condition $B \neq B^T$
C	$\mathbb{R}^{m \times m \times n \times n}$ (e.g. np.zeros((2, 2, 3, 3)))
nspins	10

Table 10: Parameters used by the Integrals module tests.

1. test_load_integrals: UT3

   Type: Automatic

   Initial State: N/A

Input: The files oneint1 and twoint1 (see Table 10).

Output: the loaded integral matrices and the number of spinorbitals.

Test Case Derivation: N/A

How test will be performed: Automated on unit testing framework. The content of oneint1 and twoint1 will be read and stored externally into variables and compared against the ones loaded by the module. The number of sinorbitals will be compared with nspins (reported in Table 10).

2. test_check_invalid_integrals: UT4

   Type: Automatic

   Initial State: N/A

   Input: The electron integrals as specified for each test case (C1-C7) in Table 11. The values of the variables in columns 2 and 3 are reported in Table 10.

Case	**h**	**v**	Exception
C1	file1	twoint1	ValueError
C2	4	twoint1	TypeError
C3	oneint1	4	TypeError
C4	twoint1	twoint1	ValueError
C5	oneint1	oneint1	ValueError
C6	oneint1	C	ValueError
C7	oneint1	B	ValueError

Table 11: Invalid input values. Seven test cases are defined (C1-C7) with corresponding input files in columns 2 and 3. Column 4 specifies the expected raised exception.

   Output: An exception will be raised according to Table 11.

   Test Case Derivation: For C1 the **h** file is not a NumPy .npy. In C2 and C3, **h** integers are passed as inputs instead of matrices. The remaining test cases use inputs with incorrect number of dimensions (**h** has to be a square matrix, **v** must be 4D with equivalent dimensions)

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

3. test_verify_symmetry: UT5

   Type: Automatic

   Initial State: N/A

   Input: The electron integrals for each test case (C1-C3) are listed in columns 2 and 3 of Table 12. The variables are reported in Table 10.

Case	**h**	**v**	Exception
C1	A	twoint2	ValueError
C2	oneint2	B	ValueError
C3	oneint2	twoint2	ValueError

Table 12: Invalid input values. Three test cases are defined (C1-C3) with corresponding input files in columns 2 and 3. Column 4 specifies the expected raised exception.

   Output: An exception raised as indicated in Table 12.

   Test Case Derivation: The matrices A and B must not have any element of symmetry. While the file oneint2 has the correct symmetry for a one-electron integral, twoint2 does not meet the asymmetric permutations of a two-electron interal (see Eq. (8) for Data Definition 3 in SRS [**?**]).

   How test will be performed: Automated on unit testing framework.

6.2.3 Density Module

The Density module handles the format of the RDMs; its interface is specified in Section 9 of the MIS [**?**]. The tests to verify that the RDMs have the right characteristics to be used in an EOM calculation are described below. The first unit test only checks that module is able to load a valid input (test_assign_rdms). The remaining tests are to detect incorrect inputs given by the type, or symmetry properties of the inputs (test_check_invalid_rdms, test_verify_symmetry, test_verify_normalization). The parameters used through

this tests are listed in Table 13. The data files with extensions .in and .npy should be located under the package's *test/data* folder.

Parameters	Value
file1	test/data/test_input1.in
onedm1	be_sto3g_onedm_spino.npy
twodm1	be_sto3g_twodm_spino.npy
twoint2	h2_hf_sto6g_oneint.npy
A	$\mathbb{R}^{m\times m}$ (e.g. [[1,2], [3,4]]) with the condition $A \neq A^T$
B	$\mathbb{R}^{m\times m\times m\times m}$ (e.g. np.arange(16).reshape(2, 2, 2, 2)) with the condition $B \neq B^T$
C	$\mathbb{R}^{m\times m\times n\times n}$ (e.g. np.zeros((2, 2, 3, 3)))
nspins	10
nelec	4

Table 13: Parameters used by the Density module tests.

1. test_assign_rdms: UT6

   Type: Automatic

   Initial State: N/A

   Input: Files onedm1 and twodm1, and number of electrons nelec (see Table 13).

   Output: the loaded RDM matrices and the number of electrons and spinorbitals.

   Test Case Derivation: N/A

   How test will be performed: Automated on unit testing framework. The content of onedm1 and twodm1 will be read and stored externally into variables and compared against the ones loaded by the module. The number of sinorbitals will be compared with nspins (reported in Table 13).

2. test_check_invalid_rdms: UT7

   Type: Automatic

   Initial State: N/A

   Input: nelec and the RDMs as specified by test case (C1-C) in Table 14. The values of nelec and the variables in columns 2 and 3 are reported in Table 13.

Case	$\boldsymbol{\gamma}$	$\boldsymbol{\Gamma}$	Exception
C1	file1	twodm1	ValueError
C2	twodm1	twodm1	ValueError
C3	onedm1	C	ValueError
C4	onedm1	B	ValueError

Table 14: Invalid RDMs input values. The test cases defined (C1-C4) have corresponding input files in columns 2 and 3. Column 4 specifies the expected raised exception.

   Output: An ValueError exception will be raised.

   Test Case Derivation: In test C1 the $\boldsymbol{\gamma}$ file is not a NumPy .npy. In C2 to C3, the inputs have incorrect dimensions ($\boldsymbol{\gamma}$ has to be a square matrix, $\boldsymbol{\Gamma}$ must be 4D with equivalent dimensions)

   How test will be performed: Automated on unit testing framework.

3. test_verify_symmetry: UT8

   Type: Automatic

   Initial State: N/A

   Input: The RDMs for each test case (C1-C3) are listed in columns 2 and 3 of Table 15. The number of electrons nelec is also an input. (see Table 13 for A, B, twodm1, onedm1 and nelec values).

   Output: An ValueError exception is raised.

   Test Case Derivation: The matrices A and B must not have any element of symmetry. While the file onedm1 has the correct symmetry for a

Case	$\boldsymbol{\gamma}$	$\boldsymbol{\Gamma}$	Exception
C1	A	twodm1	ValueError
C2	onedm1	B	ValueError
C3	onedm1	twodm1	ValueError

Table 15: Inputs with invalid symmetries. Three test cases are defined (C1-C3) with corresponding input files in columns 2 and 3. Column 4 specifies the expected raised exception.

one-RDM, twodm1 does not meet the asymmetric permutations of a two-RDM (see Data Definition 5 in SRS [**?**]).

How test will be performed: Automated on unit testing framework.

4. test_verify_normalization: UT9

   Type: Automatic

   Initial State: N/A

   Input:

Case	electrons	$\boldsymbol{\gamma}$	$\boldsymbol{\Gamma}$	Exception
C1	2	onedm1	twodm1	ValueError
C2	4	onedm1	2*twodm1	ValueError

Table 16: Tests to verify the normalization condition of RDMS. Column 5 specifies the expected raised exception.

Output: An ValueError exception is raised.

Test Case Derivation: The normalization condition of the RDMs is given by their matrix trace (specified in Table 1 in the SRS [**?**]); for $\boldsymbol{\gamma}$ it must match the number of electrons, however in test C1 the trace of onedm1 is 4. In test C2, the matrix for $\boldsymbol{\Gamma}$ was multiplied by a factor to make the test fail.

How test will be performed: Automated on unit testing framework.

6.2.4 Solver Module

This section verify the Solver module described in Subsection 17 of the MIS [**?**]. This is mainly a Software Decision Module (external libraries), therefore only the implementation component related to the transformation of a generalized eigenvalue problem to an ordinary one (the orthogonalization method, abbreviated as orthog) will be verified. The module takes two matrices (right and left-hand sides of the eigenvalue problem), a parameter to handle division by zero errors during matrix inversion, and an orthog method (one of symmetric or asymmetric). The test described bellow (UT10) checks that for a specified N-electron system (e.g. an atom) the same result is obtained with either of the orthog methods.

1. test_dense_orthogonalization: UT10

   Type: Automatic

   Initial State: N/A

   Input: The electron integrals and RDMs corresponding to the systems presented in Table 17 and an orthogonalization method (the symmetric and asymmetric methods will be tried for each system).

File	IM	**Expected**	**tol**
Be (STO-3G)	1	0.91414765	1e-6
He (STO-3G)	1	0.25403769	1e-8

Table 17: Parameters for the tests: the systems whose integrals and RDMs will be used (column 1), an IM (column 2), the expected energy value for comparison (in Hartree) and the tolerance in the difference between expected and calculated values (column 3).

   Output: The eigenvalues of IM1.

   Test Case Derivation: N/A

   How test will be performed: The testing framework will pass the input parameters to the program. From the solutions, the lowest eigenvalue will be selected and compared against the "expected" value. If the absolute error is less or equal than the tolerance criteria, the test will pass.

6.2.5 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 18 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 18: Parameters used by all Output module tests.

1. test_output_dump: UT11

   Type: Automatic

   Initial State: N/A

   Input: The parameters specified in Table 18 (which are common for all cases) and the inputs that change in each test, listed in Table 19 (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 19 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 19: 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

This section is not applicable for the present project.

6.4 Traceability Between Test Cases and Modules

All modules considered as part of the Unit Testing Scope (Section 6.1) have been verified. Table 20 shows the correspondence between modules and tests:

	M1	M2	M3	M4	M5	M6-M11	M12	M13
UT1-UT2			X					
UT3-UT5				X				
UT6-UT9					X			
UT10							X	
UT11								X

Table 20: Traceability for modules and unit test sections

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 [**?**].