Trabalho

Título em Português:	Implementação e manutenção de código computacional em framework de cálculo de estrutura eletrônica de semicondutores
Título em Inglês:	Implementation and maintenance of computational code in a semiconductor electronic structure calculation framework
Autor:	João Pedro Fernandes
Instituição:	Universidade de São Paulo
Unidade:	Instituto de Física de São Carlos
Orientador:	Guilherme Matos Sipahi
Área de Pesquisa / SubÁrea:	Física da Matéria Condensada
Agência Financiadora:	USP - Programa Unificado de Bolsas

Implementação e manutenção de código computacional em um framework de cálculo de estrutura eletrônica de semicondutores

Estudante de Graduação Autor: João Pedro Fernandes

Orientador: Prof. Dr. Guilherme Matos Sipahi

Universidade de São Paulo - USP

E-mail: joao.pedro.fernandes.jotape7@usp.br

Objetivos

Este projeto possui como objetivo principal dar continuidade à implementação e manutenção do framework desenvolvido no LFC (Laboratório de Física Computacional) da USP em São Carlos, o qual é um código computacional em Fortran que realiza cálculos da estrutura eletrônica de semicondutores. O alvo final é a completa integração Python-Fortran, oferecendo maior acessibilidade ao usuário. Nesta etapa, o objetivo teve como fases específicas duas etapas: a validação dos módulos do código-fonte através da expansão da suíte de testes unitários, e também a investigação qualitativa do framework quanto aos testes desenvolvidos através da análise de cobertura de código (Code Coverage). O propósito dessas etapas então se estendeu à uma obtenção e estudo de métricas qualitativas e quantitativas, visando a manutenção do código.

Métodos e Procedimentos

Os métodos desta fase do projeto focaram em duas frentes:

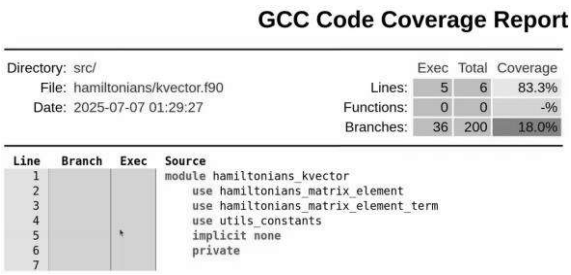
1. Expansão da suíte de testes unitários: as testagens foram referentes ao diretório de "ferramentas/utilidades" necessárias aos cálculos em "src/utils" e também quanto ao diretório "src/hamiltonians" principalmente em relação aos arquivos kvector e matrix, essenciais para a manipulação de matrizes hamiltonianas. Tais testes seguiram um padrão já desenvolvido anteriormente no framework com o Fortran Package Manager (FPM), tendo como procedimento uma sequência de passos já estabelecida.
2. Cobertura de Código: esse procedimento utiliza uma ferramenta chamada Gcovr juntamente com o compilador Gfortran para provar a eficácia dos testes. Esta metodologia que visa avançar quanto à manutenção do framework, também é gerenciada pelo FPM e utiliza flags específicas de comando como "--coverage" para gerar os dados de cobertura e gerar um relatório HTML interativo e filtrado exibindo ao usuário a análise quantitativa e qualitativa do código-fonte do framework. Esse relatório usa um sistema de cores (verde, vermelho e amarelo) e fornece a cobertura de Linhas, Funções e Branches (Ramificações).

Resultados

Os resultados relativos à expansão dos testes foram positivos, mostrando o seguinte quanto aos arquivos:

- A suíte de testes continuada para kvector.f90 obteve 100% de sucesso com todos os testes passando (4 de 4).
- A suíte de testes criada para matrix.f90 foi desenvolvida do zero obtendo mais de 30 testes quanto as funcionalidades de uma matriz. Destes, 28 de 31 testes passaram com sucesso, mostrando-se uma boa depuração do código, embora exigindo uma avaliação e análise daqueles que não obtiveram sucesso.

Entretanto, o principal resultado veio mesmo quando foi aplicada a análise de Cobertura de Código com Gcovr, o qual revelou que para o "kvector", embora houve sucesso em todos os testes, a análise de cobertura apontou 83,3% de linhas cobertas e apenas 18,0% das branches (ramificações de possibilidades do código – 36 linhas de um total de 200 encontradas), alegando com isso que a validação do código-fonte não estava completa. Veja abaixo o explicitado:


GCC Code Coverage Report

Directory: src/
File: hamiltonians/kvector.f90
Date: 2025-07-07 01:29:27

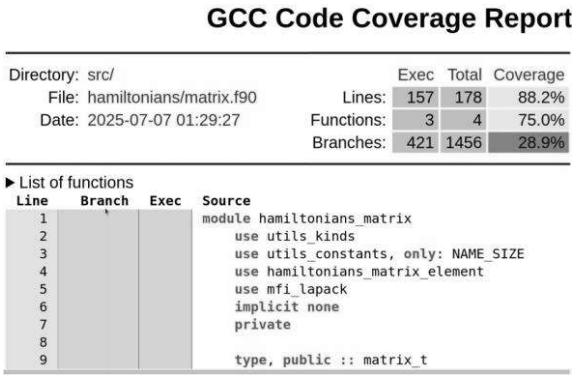
	Exec	Total	Coverage
Lines:	5	6	83.3%
Functions:	0	0	-%
Branches:	36	200	18.0%

```
Line  Branch  Exec  Source
   1                module hamiltonians_kvector
   2                    use hamiltonians_matrix_element
   3                    use hamiltonians_matrix_element_term
   4                    use utils_constants
   5        *           implicit none
   6                    private
   7
```

Figura 1: Code Coverage do código-fonte ao kvector

Esse total de 6 linhas identificadas se referem somente as que podem ser executadas, estando mais abaixo das linhas de código mostradas na imagem.

Já para o "matrix", a análise mostrou uma boa qualidade da suíte de testes obtendo 88,2% de suas linhas cobertas, entretanto uma função não foi executada por nenhum dos testes criados e além disso, as possibilidades de execução de testagem nos blocos de código foi, no entanto, limitada, faltando muitas ramificações (branches) a serem executadas para trazer uma cobertura diversa e completa do código. Veja abaixo a análise mencionada:


GCC Code Coverage Report

Directory: src/
File: hamiltonians/matrix.f90
Date: 2025-07-07 01:29:27

	Exec	Total	Coverage
Lines:	157	178	88.2%
Functions:	3	4	75.0%
Branches:	421	1456	28.9%

▶ List of functions

```
Line  Branch  Exec  Source
   1        *        module hamiltonians_matrix
   2                     use utils_kinds
   3                     use utils_constants, only: NAME_SIZE
   4                     use hamiltonians_matrix_element
   5                     use mfi_lapack
   6                     implicit none
   7                     private
   8
   9                     type, public :: matrix_t
```

Figura 2: Code Coverage do código-fonte ao matrix

Conclusões

Conclui-se que a expansão e criação dos testes, além da implementação da análise de Cobertura de Código se mostraram essenciais e um avanço para a manutenção do framework. O principal resultado e conclusão foi o entendimento que um sucesso grande dos arquivos de teste não garante a validação completa do código-fonte, pois através da análise de cobertura foram identificadas lacunas nos blocos de código, como linhas e funções que não foram executadas, e além do fornecimento quantitativo de ramificações de possibilidades não executadas nos blocos de condicionais e repetições do framework.

A utilização dessa nova metodologia trouxe um diagnóstico qualitativo preciso e que valida a eficácia dos testes criados, e com isso sendo uma ferramenta de implementação de confiabilidade do código-fonte. Logo, visando a manutenção agora do framework, uma metodologia de fácil visualização ao usuário se mostrou extremamente relevante, se preparando ao futuro com a integração final Python-Fortran do framework.

Referências

https://fpm.fortran-lang.org
https://github.com/luizpbraga/fortran4duck
https://gcovr.com/en/8.3/

Implementation and maintenance of computational code in a semiconductor electronic structure calculation framework

Undergraduate Student Author: João Pedro Fernandes

Supervisor: Prof. Dr. Guilherme Matos Sipahi

University of São Paulo - USP

E-mail: joao.pedro.fernandes.jotape7@usp.br

Objectives

This project has as its main objective to continue the implementation and maintenance of the framework developed at the LFC (Computational Physics Laboratory) of USP in São Carlos, which is a Fortran computational code that performs electronic structure calculations of semiconductors. The final goal is the complete Python-Fortran integration, providing greater accessibility to the user. In this stage, the objective had two specific phases: the validation of the source code modules through the expansion of the unit test suite, and also the qualitative investigation of the framework regarding the developed tests through code coverage analysis. The purpose of these phases then extended to obtaining and studying qualitative and quantitative metrics, aiming at the maintenance of the code.

Materials and Methods

The methods of this phase of the project focused on two fronts:

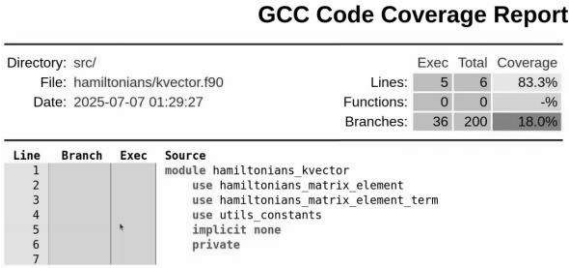
1. Expansion of the unit test suite: The tests were related to the “tools/utilities” directory required for calculations in src/utils, as well as the src/hamiltonians directory, mainly regarding the kvector and matrix files, which are essential for the manipulation of Hamiltonian matrices. These tests followed a pattern previously developed in the framework with the Fortran Package Manager (FPM), following an already established sequence of steps.
2. Code Coverage: This procedure uses a tool called Gcovr together with the Gfortran compiler to prove the effectiveness of the tests. This methodology, which aims to advance the maintenance of the framework, is also managed by FPM and uses specific command flags such as --coverage to generate coverage data and produce an interactive and filtered HTML report, displaying to the user the quantitative and qualitative analysis of the framework's source code. This report uses a color system (green, red, and yellow) and provides coverage metrics for Lines, Functions, and Branches.

Results

The results related to the expansion of the tests were positive, showing the following regarding the files:

- The continued test suite for kvector.f90 achieved 100% success with all tests passing (4 out of 4).
- The test suite created for matrix.f90 was developed from scratch, obtaining more than 30 tests regarding the functionalities of a matrix. Of these, 28 out of 31 tests passed successfully, showing good debugging of the code, although requiring further evaluation and analysis of those that did not succeed.

However, the main result came when the Code Coverage analysis with Gcovr was applied, which revealed that for kvector, although there was success in all tests, the coverage analysis showed 83.3% of lines covered and only 18.0% of branches (branches of code possibilities – 36 lines out of a total of 200 found), indicating that the validation of the source code was not complete. See the explanation below:



GCC Code Coverage Report

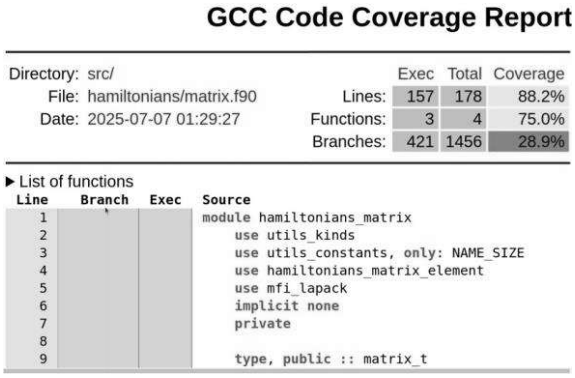
		Exec	Total	Coverage
Directory: src/				
File: hamiltonians/kvector.f90	Lines:	5	6	83.3%
Date: 2025-07-07 01:29:27	Functions:	0	0	-%
	Branches:	36	200	18.0%

Line	Branch	Exec	Source
1			module hamiltonians_kvector
2			use hamiltonians_matrix_element
3			use hamiltonians_matrix_element_term
4			use utils_constants
5			implicit none
6			private
7			

Picture 1: Code Coverage of the source code for kvector

This total of 6 identified lines refers only to those that can be executed, located below the lines of code shown in the image.

As for matrix, the analysis showed good quality of the test suite, achieving 88.2% of its lines covered. However, one function was not executed by any of the created tests, and in addition, the possibilities for executing tests in the code blocks were limited, with many branches still needing to be executed to bring diverse and complete coverage of the code. See the analysis mentioned below:



GCC Code Coverage Report

		Exec	Total	Coverage
Directory: src/				
File: hamiltonians/matrix.f90	Lines:	157	178	88.2%
Date: 2025-07-07 01:29:27	Functions:	3	4	75.0%
	Branches:	421	1456	28.9%

▶ List of functions

Line	Branch	Exec	Source
1			module hamiltonians_matrix
2			use utils_kinds
3			use utils_constants, only: NAME_SIZE
4			use hamiltonians_matrix_element
5			use mfi_lapack
6			implicit none
7			private
8			
9			type, public :: matrix_t

Picture 2: Code Coverage of the source code for matrix

Conclusions

It is concluded that the expansion and creation of tests, in addition to the implementation of the Code Coverage analysis, proved to be essential and an advancement for the maintenance of the framework. The main result and conclusion was the understanding that a high success rate of the test files does not guarantee the complete validation of the source code, since the coverage analysis identified gaps in the code blocks, such as lines and functions that were not executed, as well as providing quantitative information about unexecuted branches in the conditional and loop blocks of the framework.

The use of this new methodology brought a precise qualitative diagnosis that validates the effectiveness of the created tests, thus serving as a tool for implementing source code reliability. Therefore, aiming now at the maintenance of the framework, a methodology that provides easy visualization to the user has proven to be extremely relevant, preparing for the future with the final Python-Fortran integration of the framework.

References

https://fpm.fortran-lang.org
https://github.com/luizpbraga/fortran4duck
https://gcovr.com/en/8.3/