Título em Português: Implementação e manutenção de cálculo de estrutura eletrônica de semicondutores

Título em Inglês: Implementation and maintenance of semiconductor electronic structure calculation

Autor: Ian Giestas Pauli

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: CNPq - PIBIC

Implementação e manutenção de cálculo de estrutura eletrônica de semicondutores

Ian Giestas Pauli

Guilherme Matos Sipahi

Universidade de São Paulo

Instituto de Física de São Carlos

iangiestas@usp.br

Objetivos

O Laboratório de Física Computacional, o *LFC*, vem desenvolvendo um software de cálculo de estrutura eletrônica, que permite usar um computador de mesa para realizar cálculos de estrutura de bandas, mesmo em sistemas cuja matriz hamiltoniana não caberia na memória ao usar outros métodos de diagonalização, além disso, o código podia realizar procedimentos auto-consistentes e cálculos feitos em pós-processamento, o que permite caracterizar novos materiais. No entanto, com o passar dos anos e as diversas alterações para a inclusão de novos recursos, o código cresceu sem o planejamento necessário para a sua organização e algumas escolhas tornaram-no pouco prático de usar e realizar melhorias ou implementar novos recursos. Com o intuito de resolver este problema, iniciou-se a escrita de uma nova versão, buscando sanar estas lacunas do projeto.

Métodos e Procedimentos

O novo código foi escrito usando fortran moderno e técnicas de orientação a objetos, visando facilitar a introdução de novos hamiltonianos, bem como a configuração por seus usuários, além de tornar possível uma rápida adição de recursos através de uma arquitetura flexível. Além disso, foi integrado um sistema de compilação e gerenciamento de dependências, o *fpm*(1), que permite integrar códigos de propósito geral, o que faz com que se passe mais tempo resolvendo os problemas de Física, do que problemas de ordem técnica.

Resultados

O código agora conta com um programa principal cuja configuração pode ser realizada através de um arquivo de texto plano estruturado em TOML, graças a facilidade da incorporação de sua biblioteca fortran e com isso é possível controlar o cálculo, os parâmetros dos materiais, método de solução e configurar a geração de gráficos.

```
# Solvers settings
[ settings ]
solver       = "lapack"
hamiltonian  = "luttinger-kohn.f90"
window       = "none"
output       = "quantum_well_test.res"

# Specific solver settings
[ settings.lobpcg ]
num_eigen = 8
tolerance = 1e-9
max_iters = 100
renorm    = 0.03
```

Figura 1: Exemplo de algumas configurações possíveis escritas em TOML

Outra funcionalidade adicionada é a introdução da possibilidade de incluir novos hamiltonianos por um arquivo com uma sintaxe idêntica a do fortran, em tempo de execução, ou seja, não é preciso recompilar o código para trocar o modelo a ser usado, por meio de um *parser*

simples que usa o algoritmo de Dijkstra, o *shunting yard*(2), desenvolvido em uma biblioteca a parte. É possível definir expressões que podem ser reutilizadas posteriormente ao declarar os elementos da matriz do modelo e utilizar os nomes de parâmetros definidos no arquivo apropriado.

```
!= Luttinger-Kohn model
!= 6x6

Q = neg((gamma1+gamma2)*(kx**2+ky**2) + (gamma1-2*gamma2)*kz**2)
T = neg((gamma1-gamma2)*(Kx**2+Ky**2) + (gamma1+2*gamma2)*Kz**2)
R = neg(sqrt(3)*(gamma2*(Kx**2-Ky**2) - 2*i*gamma3*(Kx*Ky)))
S = 2*i*sqrt(3)*gamma3*(Kx-i*Ky)*Kz

! Main diagonal
H(1,1) = Q + EVB*ONE
H(2,2) = T + EVB*ONE
H(3,3) = T + EVB*ONE
H(4,4) = Q + EVB*ONE
H(5,5) = 0.5*(Q+T) - delta*ONE + EVB*ONE
H(6,6) = 0.5*(Q+T) - delta*ONE + EVB*ONE
```

Figura 2: Exemplo do começo do arquivo que define o hamiltoniano 6 x 6 de Luttinger-Kohn.

Para efeito de comparação, a Figura 3 mostra o arquivo de entrada do código antigo, onde além de não existir indicações do teor dos parâmetros, ainda é utilizado o formato fixo de leitura, o que faz com que espaços em branco sejam relevantes.

```
   2    0

   0
 300.0000  300.0000    5.6611    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000
   3.0990
   0.2800
   1.4904   -0.3148    0.2852
   0.0000
   0.4222
   0.0000    0.0000
   8.8087
   5.6611
   0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000
   0.0000
  12.5600
   0.0000    0.0000    0.0000    0.0000
   0.0000

   4
 150.0000  150.0000    5.6530    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000
   1.5190
   0.3410
   1.3842   -0.7379    0.1321
   0.0000
   1.2157
   0.0000    0.0000
   9.6837
   5.6530
   0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000
   0.0000
  12.5600
   0.3400    0.0940    0.1720    0.0667
   0.4000

   0.00000E+00    0.00000E+00  300.0000  300.0000    5.6611    5.6533
   7    7    0    0    0    0    1   20   20   20    F
   0.0100
   8    1   86   10
   1    0    0
   0
```

Figura 3: Exemplo do arquivo de entrada antigo.

Conclusões

Foi possível adicionar algumas funcionalidades importantes aos usuários não-programadores, em tempo de execução. É possível definir o método de solução, os parâmetros e modificá-los sem a necessidade de recompilação, e sem precisar programar usando a *API* desenvolvida. No entanto, o *parser* ainda é muito rudimentar dado a complexidade existente em ter que definir estruturas de dados genéricas em Fortran e ainda não possui suporte a operações unárias (apenas funções, que é como "-" unário é definido). Ainda falta realizar o porte dos cálculos autoconsistentes, cujas alterações estão sendo realizadas por outro colaborador do projeto, o que será um bom termômetro das alterações. Por fim, testes de desempenho e comparação com o código original precisam ser configurados de modo que se possa medir as diferenças entre os tempos de construção e diagonalização dos hamiltonianos.

Referências Bibliográficas

1. L. J. Kedward *et al.*, "The State of Fortran," in Computing in Science & Engineering, vol. 24, no. 2, pp. 63-72, 1 March-April 2022, doi: 10.1109/MCSE.2022.3159862.
2. Dijkstra E.W. Algol 60 translation: An Algol 60 translator for the X1 and making a translator for Algol 60. Stichting Mathematisch Centrum. Rekenafdeling. Stichting Mathematisch Centrum; 1961.

Implementation and maintenance of semiconductor electronic structure calculation

Ian Giestas Pauli

Guilherme Matos Sipahi

Universidade de São Paulo

Institute of Physics of São Carlos

iangiestas@usp.br

Objectives

The Computational Physics Laboratory, *LFC*, has been developing electronic structure calculation software, which allows using a desktop computer to perform band structure calculations, even on systems whose Hamiltonian matrix would not fit in memory when using other diagonalization methods. In addition, the code could perform self-consistent procedures and post-processing calculations, which allows for the characterization of new materials. However, over the years and the many changes to include new features, the code grew without the necessary planning for your organization and some choices made it impractical to use and make improvements or implement new features. To solve this problem, a new version was written, seeking to remedy these gaps in the project.

Methods and Procedures

The new code was written using modern Fortran and object-oriented techniques, aiming to facilitate the introduction of new Hamiltonian as well as configuration by its users, in addition to making it possible to quickly add features through a flexible architecture. In addition, a compilation and dependency management system, *fpm*(1), was integrated, which allows the integration of general-purpose codes, which means that more time is spent solving physics problems than technical problems.

Results

The code now has a main program whose configuration can be carried out through a plain text file structured in TOML, thanks to the ease of incorporation of its Fortran library. With this, it is possible to control the calculation, the parameters of the materials, the method of solution, and configure the graphics generation.

```
# Solvers settings
[ settings ]
solver       = "lapack"
hamiltonian  = "luttinger-kohn.f90"
window       = "none"
output       = "quantum_well_test.res"

# Specific solver settings
[ settings.lobpcg ]
num_eigen = 8
tolerance = 1e-9
max_iters = 100
renorm    = 0.03
```

Figure 1: Example of some possible configurations written in TOML

Another functionality added is the introduction of the possibility of including a new Hamiltonian by a file with a syntax identical to that of Fortran, at runtime. It is not necessary to recompile the code to change the model to be used, through a *parser* that uses Dijkstra's algorithm, the *shunting yard*(2), developed in a separate library. You can define expressions that can be reused later when declaring the elements from the model matrix and use the

parameter names defined in the appropriate file.

```
!= Luttinger-Kohn model
!= 6x6

Q = neg((gamma1+gamma2)*(kx**2+ky**2) + (gamma1-2*gamma2)*kz**2)
T = neg((gamma1-gamma2)*(Kx**2+Ky**2) + (gamma1+2*gamma2)*Kz**2)
R = neg(sqrt(3)*(gamma2*(Kx**2-Ky**2) - 2*i*gamma3*(Kx*Ky)))
S = 2*i*sqrt(3)*gamma3*(Kx-i*Ky)*Kz

! Main diagonal
H(1,1) = Q + EVB*ONE
H(2,2) = T + EVB*ONE
H(3,3) = T + EVB*ONE
H(4,4) = Q + EVB*ONE
H(5,5) = 0.5*(Q+T) - delta*ONE + EVB*ONE
H(6,6) = 0.5*(Q+T) - delta*ONE + EVB*ONE
```

Figure 2: Example of the beginning of the file that defines the 6×6 Hamiltonian of Luttinger-Kohn.

For comparison purposes, Figure 3 shows the input file of the old code, where in addition to the absence of indications of the content of the parameters, the fixed reading format is still used, which makes blank spaces relevant.

```
   2    0

   0
 300.0000  300.0000    5.6611    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000
   3.0990
   0.2800
   1.4904   -0.3148    0.2852
   0.0000
   0.4222
   0.0000    0.0000
   8.8087
   5.6611
   0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000
   0.0000
  12.5600
   0.0000    0.0000    0.0000    0.0000
   0.0000

   4
 150.0000  150.0000    5.6530    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000
   1.5190
   0.3410
   1.3842   -0.7379    0.1321
   0.0000
   1.2157
   0.0000    0.0000
   9.6837
   5.6530
   0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000
   0.0000
  12.5600
   0.3400    0.0940    0.1720    0.0667
   0.4000

   0.00000E+00    0.00000E+00  300.0000  300.0000    5.6611    5.6533
   7    7    0    0    0    0    1   20   20   20    F
   0.0100
   8    1   86   10
   1    0    0
   0
```

Figure 3: Example of the old input file.

Conclusions

It was possible to add some vital functionality to non-programmer users, at runtime. It is possible to define the solution method, and parameters and modify them without the need for recompilation, and without having to program using the *API* developed. However, the *parser* is still very rudimentary given the complexity of having to define generic data structures in Fortran, and it still doesn't support unary operations (only functions, which is how unary "-" is specified). There is still a need to carry out the porting of the self-consistent calculations, whose changes are being carried out by another project collaborator, which will be a good thermometer of the changes. Finally, performance tests and comparisons with the original code need to be configured so that the differences between the construction times and diagonalization of the Hamiltonian can be measured.

Bibliographic References

1. L. J. Kedward *et al.*, "The State of Fortran," in Computing in Science & Engineering, vol. 24, no. 2, pp. 63-72, 1 March-April 2022, doi: 10.1109/MCSE.2022.3159862.
2. Dijkstra E.W. Algol 60 translation: An Algol 60 translator for the X1 and making a translator for Algol 60. Stichting Mathematisch Centrum. Rekenafdeling. Stichting Mathematisch Centrum; 1961.