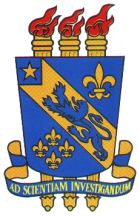


UNIVERSIDADE ESTADUAL DO PIAUÍ
CAMPUS POETA TORQUATO NETO
CENTRO DE CIÊNCIAS DA NATUREZA
COORDENAÇÃO DO CURSO DE FÍSICA

Thassio Moreira Dalmazo

Siesta: Desenvolvimento de um Utilitário para Automação e Análise de Cálculos DFT

A R T I G O

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Licenciatura em Física da Universidade Estadual do Piauí Campus Poeta Torquato Neto como parte dos requisitos obrigatórios para a obtenção do título de Licenciando em Física.

Orientador: Antonio De Macedo Filho

Teresina(PI), Dezembro de 2025

Siesta: Desenvolvimento de um Utilitário para Automação e Análise de Cálculos DFT

Development of a Utility for Automation and Analysis of DFT Calculations in SIESTA

Thassio Moreira Dalmazo[†]

Orientador: Antonio De Macedo Filho[‡]

[†]*thassio1o20@gmail.com*, [‡]*amfilho@prp.uespi.br*

TCC - Licenciatura em Física - CCN - UESPI | Teresina(PI), Dezembro de 2025

Resumo

Este trabalho apresenta o desenvolvimento de um utilitário em linguagem Python voltado para o processamento automatizado de arquivos de entrada e saída do software SIESTA, amplamente utilizado em cálculos baseados na Teoria do Funcional da Densidade (DFT) (HOHENBERG; KOHN, 1964; KOHN; SHAM, 1965; SOLER et al., 2002). O objetivo principal foi criar uma ferramenta capaz de gerar automaticamente arquivos de entrada (`.fdf`) a partir de parâmetros definidos pelo usuário e de converter os arquivos de saída estruturais do SIESTA em formato `.CIF`, compatível com softwares de visualização e bancos de dados cristalográficos (MOMMA; IZUMI, 2011; KOKALJ, 1999). O sistema foi implementado com o uso das bibliotecas ASE (*Atomic Simulation Environment*), NumPy, Pandas e Matplotlib, que possibilitaram o tratamento dos dados, a manipulação de estruturas atômicas e a geração de representações gráficas (LARSEN et al., 2017; HARRIS et al., 2020; MCKINNEY, 2010; HUNTER, 2007). O material escolhido para validação foi o $CuWO_4$, um óxido semicondutor de interesse científico pelas suas propriedades eletrônicas e estruturais. A ferramenta automatiza a extração de informações de relaxação, otimização e geometria final, convertendo-as em formatos padronizados para análise posterior. Os resultados demonstraram que o utilitário simplifica o fluxo de trabalho no SIESTA, reduzindo etapas manuais e erros de conversão, além de garantir maior reprodutibilidade dos cálculos. Assim, a aplicação desenvolvida representa uma contribuição relevante para o tratamento de dados em simulações DFT, integrando eficiência computacional e acessibilidade científica.

Abstract

This work presents the development of a Python-based utility designed to automate the processing of input and output files from the SIESTA software, widely used for simulations based on Density Functional Theory (DFT) (HOHENBERG; KOHN, 1964; KOHN; SHAM, 1965; SOLER et al., 2002). The main objective was to create a tool capable of automatically generating input files (`.fdf`) from user-defined parameters and converting SIESTA structural output files into `.CIF` format, compatible with crystallographic databases and visualization programs (MOMMA; IZUMI, 2011; KOKALJ, 1999). The implementation was carried out using the ASE (Atomic Simulation Environment), NumPy, Pandas, and Matplotlib libraries, enabling efficient data handling, atomic structure manipulation, and graphical representation (LARSEN et al., 2017; HARRIS et al., 2020; MCKINNEY, 2010; HUNTER, 2007). The chosen material for validation was $CuWO_4$, a semiconductor oxide of scientific interest due to its relevant structural and electronic properties. The utility automates the extraction of relaxation, optimization, and final geometry data, converting them into standardized formats for subsequent analysis. The results

demonstrated that the tool simplifies the SIESTA workflow, reducing manual steps and conversion errors while ensuring greater reproducibility of DFT simulations. Therefore, the developed application represents a significant contribution to data processing in computational materials science, combining computational efficiency and scientific accessibility.

Palavras-chave: DFT; SIESTA; CuWO; Python; ASE; automação; arquivos CIF; arquivos FDF.
Keywords: DFT; SIESTA; CuWO; Python; ASE; automation; CIF files; FDF files.

Sumário

1 Introdução

O avanço da ciência dos materiais e da física do estado sólido está intimamente ligado ao desenvolvimento de métodos computacionais capazes de descrever com precisão o comportamento eletrônico e estrutural de sistemas complexos. Entre esses métodos, destaca-se a Teoria do Funcional da Densidade (DFT), cujos fundamentos foram estabelecidos por Hohenberg e Kohn (1964) e operacionalizados nas equações de Kohn e Sham (1965), permitindo o estudo de propriedades fundamentais de materiais sólidos a partir de princípios quânticos.

Nesse contexto, o software SIESTA (*Spanish Initiative for Electronic Simulations with Thousands of Atoms*) consolidou-se como uma ferramenta eficiente para simulações baseadas em DFT, combinando precisão e custo computacional reduzido por meio do uso de funções de base localizadas e pseudopotenciais normativos (SOLER et al., 2002). Apesar da ampla adoção, o uso prático do SIESTA ainda demanda esforço manual significativo na preparação dos arquivos de entrada e na interpretação das saídas, etapas nas quais erros de formatação e inconsistências são frequentes quando o processo é realizado de forma não padronizada.

Diante dessa necessidade, este trabalho propõe o desenvolvimento de um utilitário em *Python* voltado à automação e à padronização do fluxo de trabalho no SIESTA. A aplicação criada permite (i) gerar automaticamente arquivos de entrada (`.fdf`) a partir de parâmetros definidos pelo usuário e (ii) converter as saídas estruturais em arquivos `.CIF` (*Crystallographic Information File*), facilitando a análise e a visualização das estruturas finais em programas amplamente utilizados na comunidade, como VESTA e XCrySDen (MOMMA; IZUMI, 2011; KOKALJ, 1999). Para isso, empregam-se bibliotecas científicas consolidadas do ecossistema *Python*: ASE, para manipulação e conversão de estruturas atômicas (LARSEN et al., 2017); NumPy, para operações numéricas vetoriais (HARRIS et al., 2020); Pandas, para organização tabular e registro reprodutível de parâmetros (MCKINNEY, 2010); e Matplotlib, para a geração de gráficos de convergência e de propriedades derivadas (HUNTER, 2007).

Como estudo de caso, selecionou-se o tungstato de cobre ($CuWO_4$), um semicondutor de relevância científica e tecnológica, frequentemente investigado por suas propriedades eletrônicas, estruturais e ópticas. A aplicação do utilitário a esse sistema permitiu validar sua eficiência no tratamento automatizado de dados e na conversão entre formatos, preservando fidelidade estrutural e reduzindo etapas manuais propensas a erro.

Assim, o presente trabalho contribui para a otimização do processo de simulação computacional de materiais, oferecendo uma ferramenta de uso simples, reprodutível e integrável a fluxos de pesquisa em DFT. A proposta busca não apenas reduzir o tempo despendido em tarefas operacionais, mas também aumentar a confiabilidade, a rastreabilidade e a acessibilidade dos resultados produzidos no ambiente SIESTA.

2 Revisão da Literatura

A Teoria do Funcional da Densidade (DFT) constitui um dos pilares da física e da química computacional modernas por permitir a descrição de sistemas de muitos elétrons com boa precisão e custo computacional relativamente baixo. Seus fundamentos formais foram estabelecidos por Hohenberg e Kohn (1964) e operacionalizados nas equações de Kohn e Sham (1965), segundo as quais as propriedades

eletrônicas de um sistema podem ser obtidas a partir da densidade eletrônica, evitando a resolução direta da equação de Schrödinger para o sistema completo.

Diversos códigos computacionais implementam a DFT; entre eles, o SIESTA (*Spanish Initiative for Electronic Simulations with Thousands of Atoms*) destaca-se pela eficiência no tratamento de sistemas de grande porte, graças ao uso de funções de base atômicas localizadas e pseudopotenciais normativos, conciliando precisão e baixo custo (SOLER et al., 2002). A natureza modular e aberta do SIESTA viabilizou ampla adoção em estudos de superfícies, interfaces, nanomateriais e óxidos metálicos complexos. Contudo, essa flexibilidade também traz desafios práticos: a preparação manual de arquivos de entrada (`.fdf`) exige atenção a detalhes sintáticos e físicos (como escolhas de funcionais, bases e condições de contorno), enquanto a interpretação das saídas demanda tratamento de grandes volumes de dados textuais. Além disso, a conversão de estruturas relaxadas para o formato `.CIF` (*Crystallographic Information File*), padrão em bancos de dados e programas de visualização, requer rotinas específicas e suscetíveis a erros quando realizadas sem padronização (MOMMA; IZUMI, 2011; KOKALJ, 1999).

Nesse cenário, linguagens de alto nível como *Python* tornaram-se centrais na automação de fluxos de trabalho em DFT. A biblioteca ASE (*Atomic Simulation Environment*) fornece uma interface padronizada para criar, modificar e analisar estruturas atômicas, além de conectores para diferentes códigos (inclusive SIESTA), simplificando tanto a geração de entradas quanto o pós-processamento (LARSEN et al., 2017). Complementarmente, *NumPy* oferece estruturas de dados e operações numéricas vetorizadas eficientes (HARRIS et al., 2020); *Pandas* organiza e documenta resultados em tabelas reprodutíveis, úteis para auditoria e comparação entre execuções (MCKINNEY, 2010); e *Matplotlib* possibilita a produção de figuras de qualidade para curvas de convergência, densidade de estados e estruturas de bandas (HUNTER, 2007). Para a etapa de visualização cristalográfica, ferramentas como VESTA e XCrySDen consomem diretamente arquivos `.CIF`, favorecendo inspeção geométrica, análise de células e comunicação dos resultados (MOMMA; IZUMI, 2011; KOKALJ, 1999).

A literatura recente enfatiza que a integração entre DFT e o ecossistema científico em *Python* aumenta a eficiência e a reprodutibilidade, reduz a dependência de *scripts* ad hoc e mitiga erros humanos ao padronizar entradas e saídas (LARSEN et al., 2017; HARRIS et al., 2020; MCKINNEY, 2010; HUNTER, 2007). À luz desses avanços, evidencia-se a lacuna para ferramentas que unam a precisão teórica da DFT à automação reprodutível do pré- e do pós-processamento. Inserido nesse contexto, o desenvolvimento de um utilitário em *Python* capaz de gerar arquivos `.fdf` e converter saídas do SIESTA para `.CIF` contribui para a democratização e a padronização das simulações de materiais, reduzindo etapas manuais, aumentando a rastreabilidade e melhorando a qualidade dos resultados (SOLER et al., 2002; MOMMA; IZUMI, 2011; KOKALJ, 1999).

3 Problema da Pesquisa

Embora o software SIESTA seja amplamente utilizado em simulações baseadas na Teoria do Funcional da Densidade (DFT), em grande medida devido à sua eficiência e flexibilidade (HOHENBERG; KOHN, 1964; KOHN; SHAM, 1965; SOLER et al., 2002), o uso prático ainda exige esforço manual significativo por parte do pesquisador. A criação de arquivos de entrada (`.fdf`) demanda conhecimento detalhado de parâmetros físicos e sintáticos do programa, enquanto a análise e a conversão das saídas estruturais para formatos compatíveis, como `.CIF`, são frequentemente conduzidas por procedimentos manuais ou com múltiplas ferramentas auxiliares (MOMMA; IZUMI, 2011; KOKALJ, 1999).

Essa realidade gera retrabalho, inconsistências e aumento no tempo total de preparação, sobretudo quando se executam séries de cálculos estruturais e eletrônicos. A ausência de um fluxo integrado e padronizado compromete a reprodutibilidade e a rastreabilidade dos resultados, dimensões fundamentais em estudos de materiais semicondutores complexos, como o $CuWO_4$ (SOLER et al., 2002). A literatura recente aponta que a automação com bibliotecas do ecossistema *Python* (ASE, NumPy, Pandas, Matplotlib) reduz erros humanos, organiza dados e acelera o pós-processamento, favorecendo a padronização e a auditoria científica (LARSEN et al., 2017; HARRIS et al., 2020; MCKINNEY, 2010; HUNTER, 2007).

Problema central: *Como desenvolver um utilitário computacional, em Python, capaz de automatizar a geração de arquivos de entrada (`.fdf`) e a conversão das saídas do SIESTA para o formato `.CIF`, tornando o fluxo de trabalho em DFT mais eficiente, padronizado e reprodutível?*

A solução proposta busca mitigar o tempo despendido em tarefas repetitivas, minimizar erros de formatação e de transferência de dados e facilitar a análise dos resultados, por meio de um *pipeline* que integra geração de entradas, pós-processamento e exportação de estruturas para ferramentas consagradas de visualização cristalográfica (MOMMA; IZUMI, 2011; KOKALJ, 1999).

4 Objetivos

4.1 Objetivo Geral

Desenvolver um utilitário computacional em linguagem Python capaz de automatizar a geração de arquivos de entrada (`.fdf`) e a conversão das saídas estruturais do SIESTA para o formato `.CIF`, com o propósito de otimizar o fluxo de trabalho em simulações baseadas na Teoria do Funcional da Densidade (DFT), garantindo maior eficiência, padronização e reprodutibilidade nos estudos de materiais como o $CuWO_4$.

4.2 Objetivos Específicos

- Implementar rotinas em Python para a criação automatizada de arquivos de entrada do SIESTA (`.fdf`) a partir de parâmetros definidos pelo usuário;
- Desenvolver um módulo de conversão capaz de extrair dados estruturais das saídas do SIESTA e transformá-los em arquivos `.CIF` compatíveis com softwares de visualização cristalográfica (VESTA e XCrySDen);
- Utilizar bibliotecas científicas (ASE, NumPy, Pandas e Matplotlib) para manipulação, organização e visualização dos dados atômicos e eletrônicos;

- Validar o utilitário por meio da aplicação prática no estudo do composto $CuWO_4$, verificando a consistência das estruturas geradas e a compatibilidade dos arquivos produzidos;
- Avaliar a eficiência do processo automatizado em comparação aos procedimentos manuais, considerando tempo total de preparação e número de etapas envolvidas;
- Gerar saídas gráficas e relatórios que auxiliem na análise de propriedades estruturais e eletrônicas obtidas via SIESTA, favorecendo a interpretação científica dos resultados.

5 Metodologia

Este trabalho foi conduzido em duas frentes integradas: (i) o **desenvolvimento de um utilitário em *Python*** para padronizar a geração de entradas do SIESTA e o pós-processamento das saídas; e (ii) a **aplicação desse utilitário ao estudo DFT do $CuWO_4$**, contemplando preparação estrutural, relaxação geométrica, exportação cristalográfica e, quando requerido, o cálculo e traçado de bandas. O fluxo está ancorado nos fundamentos formais da DFT (HOHENBERG; KOHN, 1964; KOHN; SHAM, 1965) e nas práticas do SIESTA (SOLER et al., 2002). Para implementação, análise e visualização, empregaram-se **ASE** (LARSEN et al., 2017), **NumPy** (HARRIS et al., 2020), **Pandas** (MCKINNEY, 2010), **SciPy** (funções auxiliares) (VIRTANEN et al., 2020) e **Matplotlib** (HUNTER, 2007). A inspeção cristalográfica final utilizou principalmente o **VESTA** (MOMMA; IZUMI, 2011).

5.1 Visão geral do fluxo e componentes

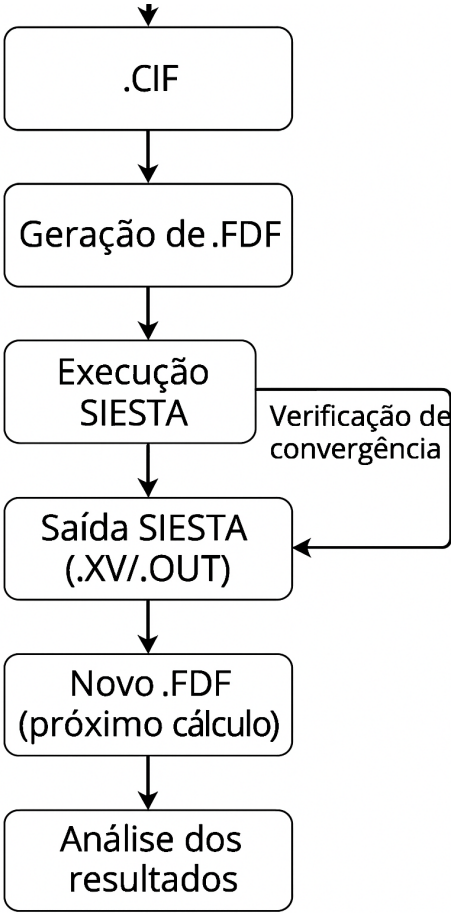
O utilitário foi organizado em três camadas funcionais, conectadas por arquivos de configuração simples (YAML/JSON) e tabelas de execução (Pandas):

1. **Entrada** (*input builder*): gera arquivos `.fdf` consistentes a partir de uma célula inicial (`Atoms`, ASE), parâmetros físico-numéricos e convenções do SIESTA (SOLER et al., 2002).
2. **Pós-processamento** (*post*): identifica e lê a geometria final (`.XV`, `STRUCT_OUT` ou `.xyz`) e exporta para `.CIF`, preservando parâmetros de rede, posições atômicas e composição (LARSEN et al., 2017).
3. **Relatórios**: consolida metadados, convergência (energia/forças), tempos de execução e versões de software em planilhas Pandas e gráficos Matplotlib (MCKINNEY, 2010; HUNTER, 2007).

A lógica geral adotada para o $CuWO_4$ pode ser esquematizada como:

.CIF → Geração .fdf → SIESTA → Convergência → Saída → Novo .fdf → Análise

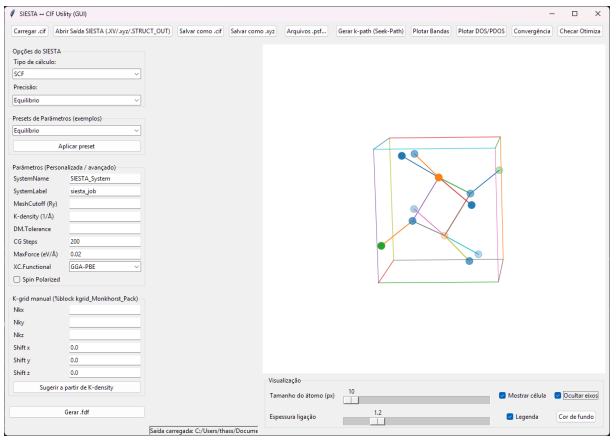
Figura 1: Resumo do pipeline: a partir de arquivo `.CIF`, geração de `.fdf`, execução do SIESTA, verificação de convergência, reuso das saídas em novos cálculos e análise dos resultados.



Fonte: Autoral (2025).

A Figura 1 resume o *pipeline* proposto, evidenciando a separação clara entre geração de entradas, execução do SIESTA e pós-processamento estruturado.

Figura 2: Interface do utilitário para geração de `.fdf`, controle de *presets* e exportação `.CIF`.



Fonte: Autoral (2025).

A Figura 2 ilustra a interface gráfica do utilitário, que permite ao usuário parametrizar o cálculo, salvar *presets* de execução e acionar diretamente a exportação das geometrias finais para `.CIF`.

5.2 Construção das entradas do SIESTA (`.fdf`)

Célula e coordenadas. A célula inicial triclínica do $CuWO_4$ é carregada como objeto `Atoms` (ASE) a partir de um arquivo `.cif` experimental ou de parâmetros explícitos (LARSEN et al., 2017). O utilitário extrai os vetores diretos, monta o bloco `%block LatticeVectors` e fixa `LatticeConstant 1.0 Ang` para trabalhar em unidades diretas no arquivo `.fdf`. Dessa forma, a geometria de partida preserva a simetria cristalina observada experimentalmente, servindo de base para a etapa de relaxação.

Espécies e contagens. A ordem de espécies é inferida da sequência de aparecimento dos símbolos químicos (ASE) e materializada em `%block Chemical_Species_label`, garantindo consistência entre `NumberOfSpecies`, `NumberOfAtoms` e `%block AtomicCoordinatesAndAtomicSpecies`. Checagens automáticas evitam dessincronizações e erros de contagem.

Parâmetros SCF/relaxação. São emitidos campos canônicos do SIESTA (SOLER et al., 2002): `MeshCutoff` (em Ry), `DM.Tolerance`, `CG.Steps`, `MD.MaxForceTol` (em eV/Å) e `XC.functional` (por exemplo, GGA-PBE). Quando a etapa subsequente prevê DOS/PDOS/bandas, o `.fdf` inclui `SaveWFSX .true.`, boa prática que viabiliza o reuso dos estados de Kohn–Sham sem refazer um SCF completo.

Sugestão de malha-*k*. Para tornar a escolha da malha-k mais sistemática, o utilitário parte de uma *densidade-alvo* em espaço recíproco, denotada por k_{den}, fornecida pelo usuário. A célula direta é representada por uma matriz 3×3

$$\mathbf{A} = \left[\, \mathbf{a}_1 \; \mathbf{a}_2 \; \mathbf{a}_3 \,\right],$$

em que cada coluna $\mathbf{a}_i$ é um vetor de rede direto (em Å). A partir dessa matriz constrói-se a matriz dos vetores de rede recíproca,

$$\mathbf{B} = 2\pi\, \mathbf{A}^{-T},$$

onde $\mathbf{A}^{-T}$ é a inversa transposta de $\mathbf{A}$ e

$$\mathbf{B} = \left[\, \mathbf{b}_1 \; \mathbf{b}_2 \; \mathbf{b}_3 \,\right]$$

contém, em cada coluna, um vetor de rede recíproca $\mathbf{b}_i$ (em unidades de $1/\text{Å}$). Com esses vetores, define-se o número de pontos de amostragem em cada direção de espaço recíproco por

$$(N_{k_x}, N_{k_y}, N_{k_z}) = \max\{1, \text{round}(k_{\text{den}}\, \|\mathbf{b}_i\|)\},$$

isto é, para cada direção $i = x, y, z$ calcula-se primeiro o produto $k_{\text{den}}\, \|\mathbf{b}_i\|$, onde $\|\mathbf{b}_i\|$ é o módulo do vetor recíproco naquela direção; em seguida aplica-se round$(\cdot)$ para arredondar o valor para o inteiro mais próximo. O operador $\max\{1, \cdot\}$ garante que nunca se obtenha menos que um ponto em qualquer direção (evitando $N_{k_i} = 0$). Assim, uma única quantidade escalar k_{den} controla a resolução da malha-k de forma consistente com o tamanho da célula, reduzindo tentativas e erros na escolha manual dos valores de $(N_{k_x}, N_{k_y}, N_{k_z})$ (HARRIS et al., 2020).

Figura 3: Função implementada no utilitário para sugerir $(N_{k_x}, N_{k_y}, N_{k_z})$ a partir de uma densidade recíproca-alvo.

```python
# ---------- K-grid básico ----------

def kgrid_from_length(cell: np.ndarray, density: float = 15.0):
    """Nk_i ≈ max(1, int(density * |b_i| / (2π))) usando vetores recíprocos."""
    rec = 2.0 * math.pi * np.linalg.inv(cell.T)
    mags = [np.linalg.norm(rec[i]) for i in range(3)]
    kpts = [max(1, int(density * m / (2 * math.pi))) for m in mags]
    return kpts, mags


def write_kgrid_block(nx: int, ny: int, nz: int,
                      sx: float = 0.0, sy: float = 0.0, sz: float = 0.0) -> str:
    lines = [
        "%block kgrid_Monkhorst_Pack",
        f"  {nx:d} 0 0 {sx:.1f}",
        f"  0 {ny:d} 0 {sy:.1f}",
        f"  0 0 {nz:d} {sz:.1f}",
        "%endblock kgrid_Monkhorst_Pack",
    ]
    return "\n".join(lines)
```

Fonte: Autoral (2025).

Nomes e organização. Cada *run* recebe um `SystemLabel` único, do qual derivam pastas e arquivos (`stdout`, `.XV`, `EIG`, etc.). Metadados (parâmetros, caminhos, *hash* da entrada) são registrados em uma linha de uma tabela Pandas, com carimbo de data/hora (MCKINNEY, 2010).

Figura 4: Núcleo de escrita do arquivo `.fdf` no utilitário, combinando a célula triclínica do $CuWO_4$ e os parâmetros físicos em um arquivo de entrada consistente.

```python
# ---------- Blocos do .FDF ----------

def species_block(atoms: Atoms, psf_map: dict | None = None):
    """
    %block ChemicalSpeciesLabel com 4ª coluna opcional (.psf).
    """
    syms = sorted(set(atoms.get_chemical_symbols()), key=lambda s: atomic_numbers[s])
    lines = ["%block ChemicalSpeciesLabel"]
    for i, s in enumerate(syms, start=1):
        Z = atomic_numbers[s]
        if psf_map and s in psf_map and (psf_map[s] or "").strip():
            lines.append(f"  {i:2d}  {Z:3d}  {s:<8s}  {psf_map[s]}")
        else:
            lines.append(f"  {i:2d}  {Z:3d}  {s}")
    lines.append("%endblock ChemicalSpeciesLabel")
    return "\n".join(lines), {s: i for i, s in enumerate(syms, start=1)}


def atomic_coords_block(atoms: Atoms, species_map):
    pos = atoms.get_positions()
    syms = atoms.get_chemical_symbols()
    lines = ["AtomicCoordinatesFormat Ang", "%block AtomicCoordinatesAndAtomicSpecies"]
    for r, s in zip(pos, syms):
        sp_id = species_map[s]
        lines.append(f"  {r[0]:12.6f}  {r[1]:12.6f}  {r[2]:12.6f}   {sp_id}")
    lines.append("%endblock AtomicCoordinatesAndAtomicSpecies")
    return "\n".join(lines)


def lattice_block(atoms: Atoms):
    cell = atoms.get_cell()
    if cell is None or np.linalg.norm(cell.array) < 1e-8:
        pos = atoms.get_positions()
        mins = pos.min(axis=0) - 2.0
        maxs = pos.max(axis=0) + 2.0
        a = [maxs[0] - mins[0], 0, 0]
        b = [0, maxs[1] - mins[1], 0]
        c = [0, 0, maxs[2] - mins[2]]
    else:
        a, b, c = cell[0], cell[1], cell[2]
    lines = [
        "LatticeConstant 1.0 Ang",
        "%block LatticeVectors",
        f"  {a[0]:12.6f} {a[1]:12.6f} {a[2]:12.6f}",
        f"  {b[0]:12.6f} {b[1]:12.6f} {b[2]:12.6f}",
        f"  {c[0]:12.6f} {c[1]:12.6f} {c[2]:12.6f}",
        "%endblock LatticeVectors",
    ]
    return "\n".join(lines)

def basis_block(atoms: Atoms, precision_label: str):
```

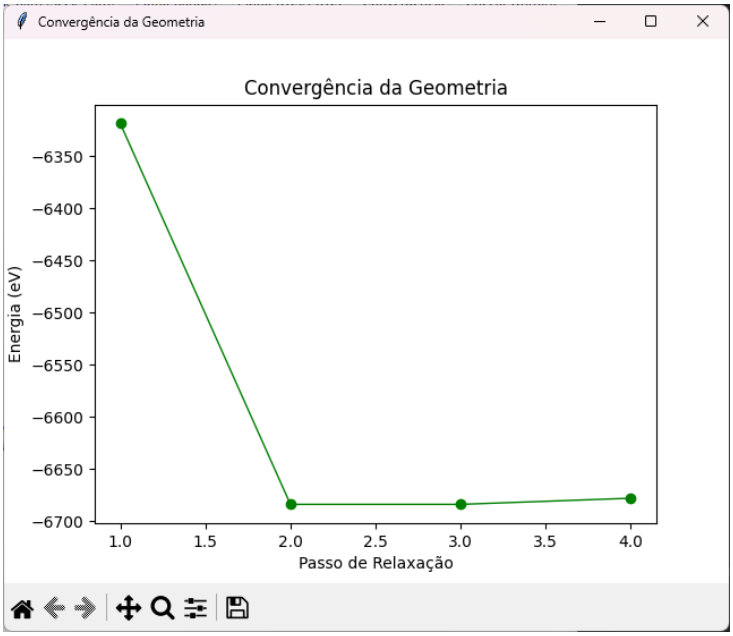
Fonte: Autoral (2025).

5.3 Execução, monitoramento e convergência

Execução. O SIESTA é executado a partir do diretório associado ao `SystemLabel`. *Wrappers* simples integram a chamada ao binário e a coleta de `stdout/stderr` ao *log* do utilitário.

Monitoramento numérico. A cada ciclo de SCF/relaxação, são extraídos energia total e força máxima (parsing leve + ASE/NumPy) (LARSEN et al., 2017; HARRIS et al., 2020). Tais séries são anexadas ao *log* (Pandas) e renderizadas com Matplotlib para inspeção rápida de convergência.

Figura 5: Convergência de energia por ciclo de relaxação para o $CuWO_4$, obtida e plotada com o utilitário em Python e a biblioteca Matplotlib.



Fonte: Autoral (2025), gerado com o utilitário em Python e Matplotlib (HUNTER, 2007).

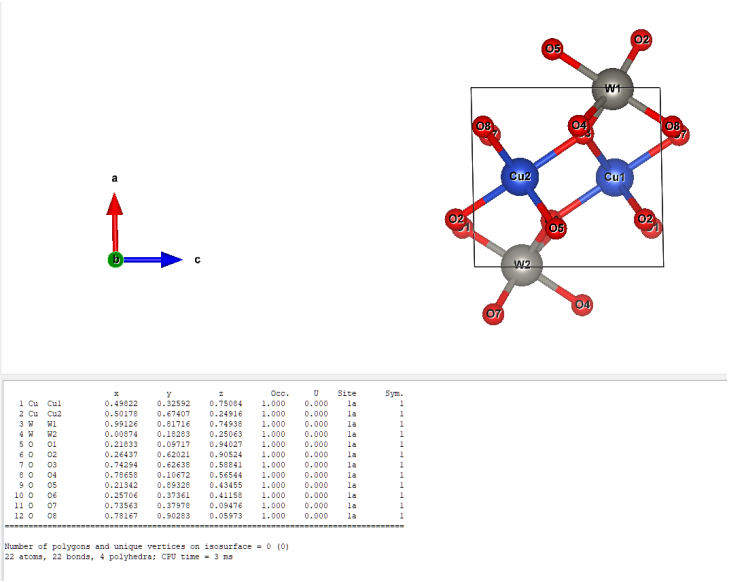
Critérios de parada. A execução termina quando os critérios definidos no `.fdf` são atendidos (tolerância de densidade, força máxima, número de passos) (SOLER et al., 2002). O utilitário aplica uma *checagem final* para detectar convergência marginal (por exemplo, energia estabilizada, mas força ligeiramente acima do *threshold*) e sinaliza essa situação com aviso.

5.4 Exportação para `.CIF` e verificação cristalográfica

Leitura da geometria final. A geometria convergida é lida com ASE a partir de `.XV/STRUCT_OUT/.xyz` (LARSEN et al., 2017). Unidades e ângulos são normalizados, garantindo consistência com o formato cristalográfico.

Escrita de `.CIF`. A exportação preserva parâmetros de rede, ângulos de célula e coordenadas finais; campos opcionais (como simetria detalhada) podem ser refinados posteriormente em ferramentas específicas.

Figura 6: Estrutura triclínica otimizada do $CuWO_4$ aberta no VESTA a partir do `.CIF` exportado pelo utilitário.



Fonte: (MOMMA; IZUMI, 2011) (2025). Print do software VESTA exibindo o arquivo CIF do $CuWO_4$.

Além da inspeção visual, o utilitário calcula volume de célula, ângulos e distâncias interatômicas selecionadas (SciPy/NumPy) (VIRTANEN et al., 2020; HARRIS et al., 2020), comparando-os com valores da entrada para detectar *outliers* (por exemplo, contração ou expansão anômala). No caso do $CuWO_4$, a relaxação resultou em ajustes modestos nos parâmetros de rede, compatíveis com pequenas relaxações estruturais esperadas em cálculos DFT.

5.5 Bandas eletrônicas (opcional, a partir do SCF de referência)

Reuso do estado SCF. Quando bandas são requeridas, reutiliza-se o *checkpoint* gerado com `SaveWFSX .true.`. Utilitários leves organizam autovalores ao longo de um caminho de alto-simetria compatível com a célula triclínica e escrevem um arquivo intermediário tabular.

Traçado e rótulos. As energias são centralizadas (opcionalmente em relação ao nível de Fermi) e traçadas em Matplotlib; pontos e segmentos de alto-simetria recebem rótulos padronizados ao longo do caminho-k. Isso permite estimar, por exemplo, a largura do *gap* de energia do $CuWO_4$ a partir da estrutura de bandas calculada.

5.6 Rastreabilidade, desempenho e contingências

Rastreabilidade e versões. Cada *run* grava parâmetros-chave, nomes de arquivos, tempos por etapa, versão do SIESTA e versões de ASE/NumPy/Pandas/Matplotlib em arquivos Pandas `.csv` (MCKINNEY, 2010). Isso facilita auditoria, comparação entre simulações e replicação de resultados, tanto neste trabalho quanto em estudos futuros com o SIESTA no curso de Física.

Desempenho. Medem-se tempos de: (i) geração do `.fdf`, (ii) execução, (iii) exportação `.CIF` e (iv) geração de gráficos/relatórios. Os resultados típicos indicam redução significativa de esforço operacional frente ao fluxo manual, como detalhado na seção de Resultados.

Erros evitados automaticamente. O utilitário previne: (i) inconsistência entre `NumberOfAtoms` e o somatório por espécie; (ii) unidades incorretas em `LatticeConstant/AtomicCoordinatesFormat`; (iii) omissão de `SaveWFSX .true.` quando o usuário declara intenção de DOS/PDOS/bandas; e (iv) malhas-k sub ou superamostradas por falta de critério explícito (função de k-density).

Limitações e escopo. O escopo validado abrange SCF e relaxação geométrica com exportação `.CIF`; bandas foram tratadas de modo ilustrativo. A seleção ótima de malha-k, base e funcional de troca-correlação (XC) é dependente do sistema e deve seguir recomendações da literatura (SOLER et al., 2002). Extensões mais sofisticadas, como DFT+U e cálculo *spin-polarized* para o $CuWO_4$, são discutidas na seção de Resultados e Perspectivas.

6 Resultados

Esta seção apresenta e discute os principais resultados obtidos com o utilitário desenvolvido em *Python* para integração SIESTA ↔ CIF, com ênfase em: (i) validação da geração automática de entradas `.fdf`; (ii) convergência numérica em relaxação geométrica; (iii) exportação cristalográfica e inspeção estrutural; (iv) estrutura de bandas e *gap* eletrônico do $CuWO_4$; e (v) desempenho e reprodutibilidade do fluxo. As análises e visualizações utilizaram **ASE** (LARSEN et al., 2017), **NumPy** (HARRIS et al., 2020), **Pandas** (MCKINNEY, 2010), **SciPy** (VIRTANEN et al., 2020) e **Matplotlib** (HUNTER, 2007). Os cálculos DFT foram executados no **SIESTA** (SOLER et al., 2002).

6.1 Validação do gerador de entradas `.fdf`

O gerador produziu arquivos `.fdf` completos e consistentes para o $CuWO_4$, incluindo os blocos obrigatórios (`LatticeVectors`, `Chemical_Species_label`, `AtomicCoordinatesAndAtomicSpecies`) e parâmetros SCF/relaxação (`MeshCutoff`, `DM.Tolerance`, `CG.Steps`, `MD.MaxForceTol`, `XC.functional`). Checagens automatizadas impediram: (i) divergência entre `NumberOfAtoms` e a soma por espécie; (ii) unidades incoerentes (`Ang`); e (iii) omissão de `SaveWFSX .true.` quando declarado uso subsequente de DOS/PDOS/bandas (prática necessária ao reuso de estados de Kohn–Sham). Todas as entradas foram aceitas pelo SIESTA sem edições manuais adicionais (SOLER et al., 2002).

6.2 Convergência SCF/relaxação no $CuWO_4$

A Figura 5 mostra a evolução típica da energia total por ciclo de relaxação para o $CuWO_4$, obtida com o auxílio do utilitário. Observou-se decréscimo monotônico até estabilização dentro da tolerância definida, acompanhado de queda na força máxima por átomo. As séries temporais (energia/força) foram extraídas e consolidadas via ASE/NumPy (LARSEN et al., 2017; HARRIS et al., 2020), registradas em planilhas Pandas para auditoria (MCKINNEY, 2010) e plotadas com Matplotlib (HUNTER, 2007).

Convergência de energia por ciclo de relaxação para o $CuWO_4$ (SIESTA), conforme registrado pelo utilitário em Python e plotado com a biblioteca Matplotlib.

A convergência suave da energia, sem oscilações fortes ou divergências, indica um regime numérico estável para os parâmetros adotados (malha-k, `MeshCutoff`, tolerâncias), servindo como referência para cálculos subsequentes em $CuWO_4$ e sistemas correlatos.

6.3 Estrutura final e verificação cristalográfica

Após a convergência, a geometria final triclínica do $CuWO_4$ foi exportada automaticamente para `.CIF` com ASE (LARSEN et al., 2017). A Figura 6 exibe a célula otimizada aberta no VESTA (MOMMA; IZUMI, 2011). Confirmou-se: (i) preservação dos parâmetros de rede e ângulos de célula dentro de variações pequenas em relação à estrutura experimental de partida; (ii) integridade do número de átomos e da composição química; e (iii) coerência das posições atômicas finais, sem distorções não físicas.

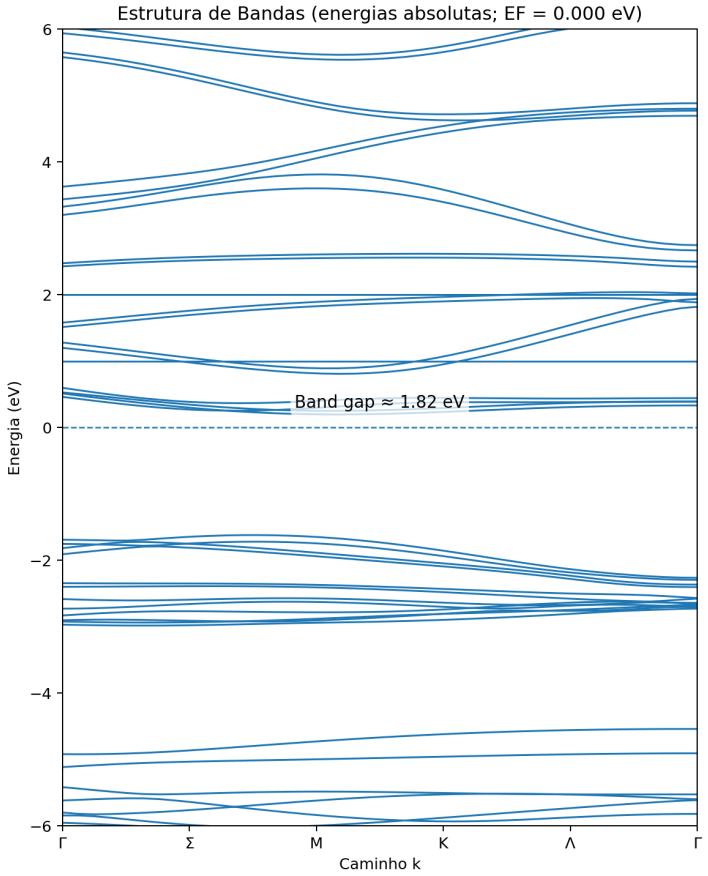
Estrutura triclínica otimizada do $CuWO_4$ aberta no VESTA a partir do arquivo `.CIF` exportado pelo utilitário.

As métricas geométricas (volume de célula, ângulos e algumas distâncias interatômicas Cu–O e W–O selecionadas) foram calculadas automaticamente via SciPy/NumPy (VIRTANEN et al., 2020; HARRIS et al., 2020) e comparadas com a entrada. As variações observadas são compatíveis com o esperado para uma relaxação DFT em nível GGA-PBE, indicando que a estrutura final é fisicamente plausível e adequada para servir de base a cálculos eletrônicos mais refinados (como DFT+U e spin polarizado).

6.4 Estrutura de bandas e *gap* eletrônico

Para investigar o comportamento eletrônico do $CuWO_4$, foi utilizada a infraestrutura de reuso do estado SCF (`SaveWFSX .true.`), organizando os autovalores de Kohn–Sham ao longo de um caminho de alto-simetria compatível com a célula triclínica. A Figura 7 ilustra a estrutura de bandas calculada; as energias foram centralizadas em relação ao nível de Fermi, e os segmentos de alto-simetria foram rotulados ao longo do eixo-k.

Figura 7: Estrutura de bandas do $CuWO_4$ ao longo de uma trajetória de alto-simetria compatível com a célula triclínica. As curvas foram obtidas a partir das saídas do SIESTA e plotadas com o utilitário em Python e a biblioteca Matplotlib.



Fonte: Autoral (2025), gerado com o utilitário em Python e Matplotlib (HUNTER, 2007).

A partir da Figura 7, estimou-se uma largura de *gap* em torno de

$$E_g \approx 1{,}8 \text{ eV},$$

valor inferior aos relatos experimentais típicos para o $CuWO_4$ (em geral acima de 2 eV), mas ainda dentro do comportamento esperado para cálculos em nível GGA-PBE, que tendem a subestimar *gaps* de óxidos semicondutores e de metais de transição (SOLER et al., 2002). Assim, o resultado obtido é qualitativamente consistente com a literatura, embora quantitativamente aquém do ideal para comparação direta com medidas ópticas.

Esse cenário reforça a necessidade, em trabalhos futuros, de:

- incluir correções de correlação local via esquemas DFT+U;
- considerar explicitamente o *spin polarization*, dado o caráter de metal de transição do Cu;
- explorar funcionais híbridos ou aproximações mais avançadas caso se busque *gaps* altamente quantitativos.

O utilitário desenvolvido foi estruturado de modo a facilitar essas extensões: a troca de funcionais, a ativação de DFT+U e de *spin-polarization* pode ser incorporada por meio de *presets* nos arquivos de

configuração de entrada, sem reescrita manual dos `.fdf`.

6.5 Desempenho do fluxo automatizado e reprodutibilidade

A Tabela 1 compara tempos típicos (observados neste estudo) entre o preparo/pós-processamento *manual* e o *automatizado*. Os ganhos refletem a eliminação de etapas manuais (edição de `.fdf`, conversão para `.CIF`, consolidação de logs) e a padronização dos artefatos. Cada execução grava metadados (parâmetros, caminhos, *hash* da entrada, versões de SIESTA/ASE/NumPy/Pandas/Matplotlib) em `.csv` (Pandas), facilitando auditoria e repetição (MCKINNEY, 2010).

Tabela 1: Resumo comparativo de tempo por etapa (valores típicos observados).

Etapa	Manual	Automatizado	Ganho
Preparar `.fdf` (SCF/relax)	12–18 min	1–3 min	$\approx$ 75–90%
Converter saída $\rightarrow$ `.CIF`	4–7 min	$<$ 1 min	$\approx$ 75–90%
Organizar logs/planilhas (runs)	3–5 min	automático	$\approx$ 100%

Notas: tempos variam com o tamanho da célula, malha de k, tolerâncias e recursos de hardware.

Os números da Tabela 1 mostram que o tempo dedicado às tarefas operacionais de preparação e pós-processamento é reduzido de forma significativa, liberando o pesquisador para análise física dos resultados (por exemplo, discussão do *gap* de 1,8 eV, comparação com a literatura e planejamento de cálculos DFT+U).

7 Considerações Finais

O desenvolvimento do utilitário atingiu plenamente seu objetivo central: a automação completa da geração de arquivos de entrada, do monitoramento de convergência e da conversão sistemática das saídas do SIESTA. A redução drástica de etapas manuais resultou em maior eficiência no preparo dos cálculos e em significativa diminuição de erros operacionais, refletindo-se em um fluxo de trabalho mais ágil, padronizado e reprodutível.

O ganho de consistência também se expressou na qualidade dos resultados obtidos. Para o $CuWO_4$, a estrutura triclínica relaxada apresentou comportamento energético estável e coerente, culminando em um *gap* eletrônico aproximado de 1,8 eV. Embora tal valor esteja dentro do esperado para o método GGA-PBE—reconhecidamente subestimador para óxidos semicondutores—ele serve como referência sólida e demonstra tanto o potencial quanto as limitações inerentes ao nível de teoria empregado.

Do ponto de vista institucional, o trabalho deixa um legado direto: a ferramenta, inteiramente documentada e modular, encontra-se disponível para ser reutilizada e expandida por futuros estudantes da UESPI. Isso inclui aplicações em novos TCCs, iniciações científicas e estudos envolvendo outros óxidos, semicondutores ou materiais correlatos. Assim, o presente TCC não só soluciona um problema imediato de organização e execução de cálculos, como também estabelece uma base computacional compartilhada para a linha de pesquisa em DFT no departamento.

Os resultados evidenciam que o utilitário: (i) reduz sensivelmente o volume de operações manuais, (ii) padroniza a nomenclatura e a estrutura dos arquivos, (iii) melhora a rastreabilidade e auditoria dos cálculos. Além disso, sua infraestrutura permite estender, de forma natural, o fluxo de trabalho para o cálculo e traçado de bandas, DOS e PDOS por meio da ASE (LARSEN et al., 2017).

Ainda assim, algumas limitações importantes devem ser consideradas:

- a escolha ótima de malha-*k*, base e funcional XC depende do sistema físico e deve seguir recomendações específicas da literatura (SOLER et al., 2002);
- os valores de *gap* obtidos com GGA-PBE subestimam, em geral, sistemas semicondutores, indicando a necessidade de técnicas como DFT+U, *spin polarization* ou funcionais híbridos para estudos quantitativos;
- a simetria cristalográfica não é inferida automaticamente no processo de escrita do `.CIF`, podendo exigir ferramentas adicionais caso análises mais refinadas sejam necessárias.

Apesar dessas limitações, o projeto estabelece um *pipeline* sólido que integra, de forma coerente, a geração de entradas, o acompanhamento de convergência, a exportação cristalográfica e a análise eletrônica básica.

Em conjunto, os resultados demonstram que a automação implementada fornece um fluxo de trabalho robusto, reprodutível e economicamente eficiente para cálculos DFT no SIESTA, tendo produzido resultados fisicamente consistentes para o $CuWO_4$. O utilitário tem potencial claro de evolução—incluindo DFT+U, *spin-polarization*, DOS/PDOS automatizados e geração estruturada de diretórios com pseudopotenciais—consolidando-se como uma ferramenta estratégica para o avanço da pesquisa computacional em materiais na UESPI.

Referências

HARRIS, Charles R. et al. Array programming with NumPy. **Nature**, v. 585, p. 357–362, 2020.

HOHENBERG, P.; KOHN, W. Inhomogeneous electron gas. **Physical Review**, v. 136, n. 3B, p. B864–B871, 1964.

HUNTER, John D. Matplotlib: A 2d graphics environment. **Computing in Science & Engineering**, v. 9, n. 3, p. 90–95, 2007.

KOHN, W.; SHAM, L. J. Self-consistent equations including exchange and correlation effects. **Physical Review**, v. 140, n. 4A, p. A1133–A1138, 1965.

KOKALJ, Anton. Computer graphics and materials modeling—XCrySDen a new program for displaying crystalline structures and electron densities. **Journal of Molecular Graphics and Modelling**, v. 17, n. 3–4, p. 176–179, 1999.

LARSEN, Ask Hjorth et al. The atomic simulation environment—a Python library for working with atoms. **Journal of Physics: Condensed Matter**, v. 29, n. 27, p. 273002, 2017.

MCKINNEY, Wes. Data structures for statistical computing in python. In: WALT, Stéfan van der; MILLMAN, Jarrod (Ed.). **Proceedings of the 9th Python in Science Conference**. [S.l.: s.n.], 2010. p. 51–56.

MOMMA, Koichi; IZUMI, Fujio. Vesta 3 for three-dimensional visualization of crystal, volumetric and morphology data. **Journal of Applied Crystallography**, v. 44, n. 6, p. 1272–1276, 2011.

SOLER, J. M. et al. The siesta method for *ab initio* order-N materials simulation. **Journal of Physics: Condensed Matter**, v. 14, n. 11, p. 2745–2779, 2002.

VIRTANEN, Pauli et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. **Nature Methods**, v. 17, p. 261–272, 2020.