Siesta: A Complete Guide
Hand-on 2: First Steps with Siesta

Dr. Carlos Maciel de Oliveira Bastos
Postdoctoral Research Fellow

University of Brasilia

14/11/2024

Outline

Getting Started: Installing SIESTA.

SIESTA's Install

The installation of SIESTA can be perceived as complex, but it's straightforward with the right tools. Minimal understanding of compilers and libraries is needed, and hardware considerations are crucial for efficiency.

Installing SIESTA with Conda

A non-traditional installation method using a Conda environment with binary files eliminates the need for source code compilation. Here are the steps:

- Install Miniforge (minimal Conda package)
- Create a new environment for SIESTA
- Install the parallel version of SIESTA using Conda

Create a Conda Environment

Create a new environment called `siesta`:

conda create -n siesta conda activate siesta

Install the parallel version of SIESTA using Conda:

conda install -c conda-forge "siesta=*=*openmpi*"

To run SIESTA with MPI, use the following command:

mpirun -np <number_of_processes> siesta < file.fdf

To run SIESTA in serial mode:

siesta < file.fdf

Execution Flags

SIESTA has several flags for different configurations:

- `--help`: Provides information on how to use the software.
- `--version`: Displays version information.
- `--out`: Specifies the output file.
- `--fdf`: Specifies flags directly in the FDF file.

Main Input Parameters in SIESTA

Main Input Parameters in SIESTA

To run a SIESTA simulation, at least two files are required:

- The input file in `.fdf` format.
- The pseudopotential file.

Pseudopotential Format in SIESTA

- SIESTA uses norm-conserving pseudopotentials for its calculations.
- Pseudopotentials are used to create an atomic basis set with pseudowavefunctions replacing core radial wavefunctions.
- Users must supply pseudopotentials, essential for defining the basis set.

Pseudopotential File Formats

- SIESTA supports the following pseudopotential file formats:
  - `.vps` (unformatted)
  - `.psf` (ASCII)
  - `.psml` (PSML format)
- Files are searched in the current directory by default or as specified by the `%ChemicalSpeciesLabel` flag.
- Precedence order for file types: `.vps` > `.psf` > `.psml`.
- Absolute paths (e.g., `/home/pseudopotentials/C.psf`) limit the search to that location.

Usage and Additional Notes

- The `ATOM` program generates only `.vps` files.
- For `.psf` and `.psml` formats, third-party tools are required.
- Pseudopotentials can be sourced from databases like Pseudo-Dojo but should be validated for quality.
- `Gen-basis` script in SIESTA's `Util` folder helps generate basis sets for single atomic species.

Key Characteristics of FDF

Comments and Case Sensitivity

- The # character is used for comments.
- FDF labels are case-insensitive: `Mesh.Cutoff` = `mesh.cutOff`.

Ignoring Characters

- Characters such as `-`, `_`, and `.` are ignored: `Mesh.Cutoff` = `Mesh_Cut-off`.

Logical Values

- Can be specified as `T`, `true`, `.true.`, etc.
- If a variable is declared without a value, it defaults to `true`.

Real Values

- Must include a decimal to avoid being read as integers.

Units and Complex Structures

Physical Units

- Values for physical quantities must include units:
  LatticeConstant 10 Ang

Blocks for Complex Structures

- Complex data structures are represented using blocks: Example

Including External Files

- Include parameters from other FDF files: %include cutoff.fdf

Error Handling

- Check the `fdf.log` file for misspelled labels or parameters.

Example: H_2 Molecule FDF

Example Code

Main Parameter Flags

Essential Parameters

SIESTA requires only two mandatory parameter blocks:

- `%block ChemicalSpeciesLabel`: Specifies the atomic species.
- `%block AtomicCoordinatesAndAtomicSpecies`: Defines the atomic positions.

Default Values

Although many other parameters can be left as their default values, it is generally not recommended to rely on them without verification. Controlling various parameters is essential for accurate and optimized simulations.

Parameter Format

We will use the following format for each parameter:

- **Parameter** <*format*> Default value: description

Determining the System and Basis Set

System Labeling

- **SystemLabel** < *string*> siesta: A single word to represent the system, used in output files. The maximum length is 20 characters.
- **SystemName** < *string*> (no default value): A brief description of the system, with a maximum of 150 characters.

Number of Species and Atoms

- **NumberOfSpecies** < *integer*>: The number of lines in the `ChemicalSpeciesLabel` block, indicating the number of atom types.
- **NumberOfAtoms** < *integer*>: The number of lines in the `AtomicCoordinatesAndAtomicSpecies` block, representing the total number of atoms.

Main Flags

Chemical Species Label

%block ChemicalSpeciesLabel < *block*> (No default value): Specifies the chemical species or pseudopotentials used in the simulation. For example:
SHOW THE EXAMPLE

- First column: Unique number identifying the chemical species.
- Second column: Atomic number (e.g., carbon has atomic number 6).
- Third column: Label for the atom (e.g., `C_bond`).
- Fourth column: Pseudopotential file (e.g., `C_bond.vps`).

Main Flags

Basis Set Parameters

- **PAO.BasisSize** < *string*> DZP: Defines the basis set size.
- Possible values:
  - SZ: Single Zeta (minimum basis)
  - DZ: Double Zeta
  - SZP: Single Zeta Polarized
  - DZP: Double Zeta Polarized
- **PAO.EnergyShift** < *float*> 0.01 Ry: Controls orbital cutoff radii.

Atomic Structure

General Atomic Structure Flags

- **LatticeConstant** $<$ *float*$>$ 1 Å: Specifies the lattice constant. Other units such as bohr can be used by appending the unit to the number.
- **%block LatticeVectors** $<$ *block*$>$ None: Specifies the lattice vectors in units of the lattice constant. Each vector should be on a separate line, as shown below:

SHOW THE EXAMPLE

Atomic Coordinates

Atomic Coordinates

- **%block AtomicCoordinatesAndAtomicSpecies** < *block*> None: Specifies the position of each atom in the structure. The first three columns represent the x, y, z coordinates of the atom according to the format chosen in `AtomicCoordinatesFormat`. The fourth column is the atom identifier corresponding to the label in `ChemicalSpeciesLabel`.

SHOW THE EXAMPLE

Atomic Coordinates

Supercell

- **%block SuperCell** $<$ *block*$>$ None: Specifies a 3×3 matrix defining the supercell in terms of the unit cell. Any value greater than 1 in the matrix expands the unit cell in the corresponding direction. For example:

SHOW THE EXAMPLE

Important: This will create 4 unit cells. Use with caution, as the system size can grow quickly.

- Symmetry from the original unit cell may be lost, impacting calculation efficiency.
- The `NumberOfAtoms` flag refers only to the unit cell.
- All other input parameters apply to the supercell, including `Kgrid.MonkhorstPack`.

Tip: Use visualization software to verify the correct creation of the supercell.

Atomic Coordinates

Atomic Coordinates Format

- **AtomicCoordinatesFormat** < *string*> Bohr :
  - **Bohr**: Cartesian coordinates in Bohr units.
  - **Ang**: Cartesian coordinates in Angstroms.
  - **LatticeConstant**: Cartesian coordinates in units of the lattice constant.
  - **Fractional**: Atomic positions given as projections on the lattice vectors.

Convergence and Self-Consistent Cycle

K-point Grid (Monkhorst-Pack)

- **%block Kgrid.MonkhorstPack** $<$ *block*$>$ Γ: Specifies the k-point grid for the self-consistent cycle using the Monkhorst-Pack scheme. It is defined using a 3×4 matrix, where the first three columns represent the discretization grid and the last column indicates the displacement of the origin, depending on symmetry.

Example for a $4 \times 4 \times 1$ grid with an origin displacement of 0.5:
SHOW EXAMPLE SIESTA will optimize the grid based on symmetry and warn if the displacement is non-optimal.

Convergence and Self-Consistent Cycle

Simplified K-point Grid

- **Kgrid.MonkhorstPack [M N P]** $<$ *list*$>$ None: Specifies a simplified k-point grid using only diagonal elements, i.e., discretization in the M $\hat{x}$, N $\hat{y}$, and P $\hat{z}$ directions without displacement.

Convergence and Self-Consistent Cycle

Exchange-Correlation Functional

- **XC.Functional** < *string*> LDA: Defines the exchange-correlation functional class. Options include:
  - LDA (Local Density Approximation)
  - GGA (Generalized Gradient Approximation)
  - VDW (van der Waals)
- **XC.Authors** < *string*> PZ: Specifies the formulation of the exchange-correlation functional (e.g., PBE). The formulation must match the functional class defined by `XC.Functional`.

Convergence and Self-Consistent Cycle

Spin Treatment

- **Spin** $<$ *string*$>$ non-polarized: Defines spin treatment in the calculation. Options include:
  - **non-polarized**: Treats spin with degeneracy.
  - **polarized**: Considers collinear spin with 2 components.
  - **non-collinear**: Considers non-collinear spin with 4 components.
  - **spin-orbit**: Includes spin-orbit coupling (requires a relativistic pseudopotential).

Convergence and Self-Consistent Cycle

Mesh Cutoff and SCF Iterations

- **Mesh.Cutoff** < *float*> 300 Ry: Specifies the cutoff for the kinetic energy of plane waves used in constructing the 3D integration mesh. Setting this parameter determines the precision of the mesh and affects the accuracy of the integrals. Proper convergence testing is essential for reliable results.
- **MaxSCFIterations** < *integer*> 100: The maximum number of SCF iterations allowed for convergence. For molecular dynamics, this applies to each time step.

Output and Input Structural Information

Density Matrix Use and Storage

- **DM.UseSaveDM** < *logical*> true: Use a density matrix stored in `System.DM` calculated in a previous run and use this matrix to start the calculation.
- **Write.DM** < *logical*> true: Save the density matrix in a file. This file can be used to continue the calculation or for post-processing purposes.

Note:

These flags are considered some of the most relevant for basic SIESTA calculations. Naturally, there are many additional flags that control various approximations and numerical details, which can be found in the SIESTA manual. Main flags specific to different types of calculations, such as relaxation, electronic structure, and optical properties, will be analyzed in their respective chapters.

Output Files

Output Files in SIESTA

SIESTA produces various output files that are essential for analyzing the results and understanding the properties of the simulated system.

- **Standard Output (stdout)**:
  - **Format**: Plain text printed in the terminal or redirected to a file.
  - **Contents**: Provides real-time information on simulation progress, including initial settings, SCF cycle convergence, warnings, and final results.
- **SystemLabel.ion.xml**:
  - **Format**: XML
  - **Contents**: Contains ion configuration and structural data. Used for restarting or continuing simulations and for post-processing.

Output Files in SIESTA

The following files are related to the density matrix and its post-processing:

- **SystemLabel.DM**:
  - **Format**: Binary
  - **Contents**: Stores the density matrix from the calculation. If the `Write.DM` flag is set to true, this file can be reused for subsequent calculations with the `DM.UseSaveDM` flag.
- **SystemLabel.WFSX and SystemLabel.PDOS**:
  - **SystemLabel.WFSX**: Stores wavefunctions for post-processing and visualizing molecular orbitals.
  - **SystemLabel.PDOS**: Provides the projected density of states (PDOS), showing contributions from different atoms or orbitals.

Output Files in SIESTA

These files contain information on atomic coordinates, forces, and relaxation steps:

- **SystemLabel.XV**:
  - **Format**: Plain text
  - **Contents**: Includes final atomic coordinates and cell vectors after relaxation or molecular dynamics. Useful for verifying the relaxed structure.
- **SystemLabel.FA**:
  - **Contents**: Contains forces acting on atoms at the last step of the calculation, helping confirm convergence in geometry optimization.

Output Files in SIESTA

These files are related to charge density and potential energy analysis:

- **SystemLabel.CHR**:
  - **Format**: Binary
  - **Contents**: Represents the total charge density of the system for further analysis (e.g., electron density maps).
- **SystemLabel.VH**:
  - **Format**: Binary
  - **Contents**: Stores the Hartree potential on the grid for potential energy analysis and visualization.

Output Files in SIESTA

These files are related to the electronic structure:

- **SystemLabel.bands**:
  - **Contents**: Provides band structure data if a band structure calculation was performed. Used for visualizing the electronic band structure.
- **SystemLabel.EIG**:
  - **Format**: Plain text
  - **Contents**: Lists eigenvalues for each k-point, essential for electronic structure analysis.

Output Files in SIESTA

These files contain information on the density of states and logging details:

- **SystemLabel.DOS**:
  - **Contents**: Contains the density of states (DOS), providing insight into the distribution of electronic states and identifying band gaps.
- **Log File (siesta.log)**:
  - **Contents**: Lists all parameters read from the input file and explicitly shows any default values, useful for verifying settings.

Output Files in SIESTA

The following file records the dynamics of the system:

- **SystemLabel.MD (For Molecular Dynamics)**:
  - **Contents**: Records atomic positions, velocities, and other properties during molecular dynamics simulations. Essential for analyzing system behavior over time.

First Steps: Obtaining the Molecular Properties