Here's a replacement for /pub/archive/macros/latex/misc/wrapfig.sty from its author, Donald Arseneau (asnd@erich.triumf.ca). I know that picinpar is suggested by the doggie book as a solution to the problem of flowing text past figures, but this works well, too. People who have already made some use of it might like the bug fix.

———————————- cut here ———————————

pythonatomistics

Release 0.1.0

Yavar

Jul 30, 2023

CONTENTS

CHAPTER

ONE

INTRODUCTION

Welcome to the "Python for Atomistic Simulation: Bridging Solid State and Quantum Chemistry" course!

1.1 Course Overview

This course is designed to provide learners with a comprehensive understanding of performing atomistic simulations using Python. The course covers two essential domains: solid-state materials using Quantum Espresso and quantum chemistry calculations using NWChem. By the end of this course, you will gain hands-on experience in using the Atomic Simulation Environment (ASE) library to set up and analyze DFT calculations for both solid-state and quantum chemistry applications.

1.2 Course Objectives

- Introduce the fundamental concepts of Density Functional Theory (DFT) and its relevance in atomistic simulations.
- Familiarize learners with the Atomic Simulation Environment (ASE) and its Python API for DFT calculations.
- Provide practical examples using Quantum Espresso for solid-state materials simulations.
- Explore quantum chemistry calculations with NWChem for molecular properties and reactions.
- Introduce some advanced tools to generate descriptors for machine learning based simulations.

1.3 Who Should Take This Course?

This course is suitable for individuals interested in computational chemistry, materials science, and anyone looking to expand their knowledge of DFT simulations using Python. Basic knowledge of Python programming is beneficial, but not mandatory, as we will cover the necessary Python concepts throughout the course.

1.4 Prerequisites

- Basic understanding of Python programming (recommended, but not required).
- Familiarity with fundamental chemistry and physics concepts.

Let's get started on this exciting journey into the world of atomistic simulations with Python!

Note: The examples in this course are provided with code snippets and step-by-step instructions. You can find the complete code and materials on GitHub repository for this course https://yavar-azar.github.io/pythonatomistics .

CHAPTER

TWO

INSTALLING UBUNTU ON VIRTUALBOX

2.1 Introduction

In this section, we will guide you through the process of installing Ubuntu on VirtualBox. VirtualBox is a powerful virtualization software that allows you to create and run virtual machines on your host operating system. Installing Ubuntu in a virtual machine enables you to practice the course material without affecting your main system.

2.2 Step 1: Downloading Ubuntu ISO Image

Before you begin, download the latest Ubuntu Desktop ISO image from the official website. Make sure to choose the appropriate version for your system, such as a 64-bit or 32-bit image.

`Ubuntu_22.04.2`

2.3 Step 2: Creating a New Virtual Machine

1. Open VirtualBox and click on the "New" button to create a new virtual machine.
2. Give your virtual machine a name (e.g., "Ubuntu_ASE") and select "Linux" as the Type, and "Ubuntu (64-bit)" as the Version (or choose the appropriate version based on your ISO image).
3. Allocate memory to the virtual machine. We recommend at least 4GB for smooth performance, but you can adjust this based on your system's resources.
4. Choose "Create a virtual hard disk now" and click "Create."

2.4 Step 3: Installing Ubuntu on the Virtual Machine

1. In the VirtualBox Manager, select the newly created virtual machine and click on the "Start" button.
2. When prompted, browse and select the Ubuntu ISO image you downloaded earlier.
3. Follow the on-screen instructions to install Ubuntu on the virtual machine. You can choose the default options or customize the installation based on your preferences.

2.5 Step 4: Essential Post-Installation Setup

After Ubuntu is installed on the virtual machine, you may need to perform some post-installation setup:

1. Update Ubuntu: Open a terminal and run the following commands to update the system:

```
sudo apt update
sudo apt upgrade
```

2. Install Guest Additions: In the VirtualBox menu, go to "Devices" -> "Insert Guest Additions CD Image." Then, open a terminal and run:

```
sudo apt install build-essential dkms
sudo mount /dev/cdrom /media/cdrom
cd /media/cdrom
sudo ./autorun.sh
```

2.6 Step 5: Install gcc and gfortran libraries

```
sudo apt install gcc gfortran
```

2.7 Conclusion

Congratulations! You have successfully installed Ubuntu on VirtualBox. Your virtual machine is now ready to be used for the course. You can now proceed with the rest of the course content and practice your atomistic simulations with ease.

Remember to save your progress and take additional snapshots as you progress through the course to have checkpoints to revert to if needed.

Happy learning and experimenting with Python for Atomistic Simulation!

CHAPTER

THREE

PYTHON FOR ATOMISTIC SIMULATION

3.1 Setting up a Virtual Environment

First, let's set up a virtual environment using $virtualenv$. If you haven't installed $virtualenv$ yet, you can do it by running the following command:

```
sudo apt install python3-pip
sudo apt install python3-tk
pip3 install virtualenv
```

Once $virtualenv$ is installed, let's create the virtual environment named "envase":

```
virtualenv envase
```

Next, activate the virtual environment (on macOS/Linux):

```
source envase/bin/activate
which python
which pip
```

3.2 Installing ASE

Now that we have the virtual environment set up, let's proceed to install ASE (Atomic Simulation Environment). We'll use pip to install it within the virtual environment:

```
pip install ase
```

ASE is now successfully installed in your virtual environment and ready to use!

3.3 Part 3: Python Basics Review

Let's start with a brief review of some Python basics. We'll cover the introduction to Python, variables and data types, control flow, loops, functions, and an overview of NumPy and Matplotlib.

3.4 Introduction to Python

Python is a versatile, high-level programming language that's easy to learn and widely used in various fields, including scientific computing and data analysis.

3.4.1 Variables and Data Types

In Python, you can declare variables and assign values to them. Python is dynamically typed, meaning you don't need to specify the data type explicitly.

Python Code for Variables and Data Types:

```
# Variables and Data Types
name = "John"
age = 25
height = 1.75
is_student = True
```

3.4.2 Control Flow - Conditional Statements and Loops

Python provides various control flow constructs, such as if-else statements and loops (for and while), to control the program's flow based on conditions.

Python Code for Control Flow:

```
# Control Flow
if age < 18:
    print("You are a minor.")
elif age >= 18 and age < 60:
    print("You are an adult.")
else:
    print("You are a senior citizen.")

# Loops
for i in range(5):
    print(f"Loop iteration: {i}")

# While Loop
counter = 0
while counter < 5:
    print(f"While loop iteration: {counter}")
    counter += 1
```

3.5 Functions

Functions allow us to group a block of code and execute it whenever needed. They promote code reusability and modularity.

Python Code for Functions:

```
# Functions
def greet_user(username):
    print(f"Hello, {username}! Welcome to our course.")

greet_user("Alice")
```

3.6 NumPy Basics

NumPy is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with an extensive collection of high-level mathematical functions to operate on these arrays.

Python Code for NumPy Basics:

```
import numpy as np

# Creating arrays
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.arange(10, 21, 2)
arr3 = np.zeros((2, 3))
arr4 = np.ones((3, 2))

# Array operations
sum_array = arr1 + arr2
dot_product = np.dot(arr3, arr4)
```

3.7 Introduction to Matplotlib

Matplotlib is a widely-used library for creating static, interactive, and animated plots in Python. It enables data visualization with a wide range of customization options.

Python Code for Matplotlib:

```
import matplotlib.pyplot as plt

# Creating simple plots
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y)
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("Sine Function")
plt.grid(True)
plt.show()
```

3.8 Conclusion

Congratulations! You've completed the Python basics review and set up the ASE environment within your virtual environment. In the next section, we'll delve deeper into atomistic simulations with ASE and Python.

Remember to activate the virtual environment whenever you work on the course materials related to ASE to ensure a clean and isolated environment for your simulations.

Happy learning and happy experimenting with Python for Atomistic Simulation!

CHAPTER

FOUR

BASICS OF CRYSTALLOGRAPHY

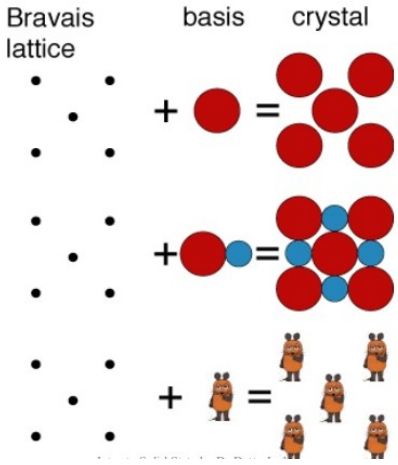
Before delving into ab-initio calculations with solid-state packages, it is essential to review some fundamental concepts in solid-state physics.

4.1 Crystal Structure

In crystallography, the *crystal structure* refers to the ordered arrangement of atoms, ions, or molecules in a crystalline material. These ordered structures arise from the intrinsic nature of the constituent particles, forming symmetric patterns that repeat along the principal directions in three-dimensional space.

4.2 Unit Cell and Lattice

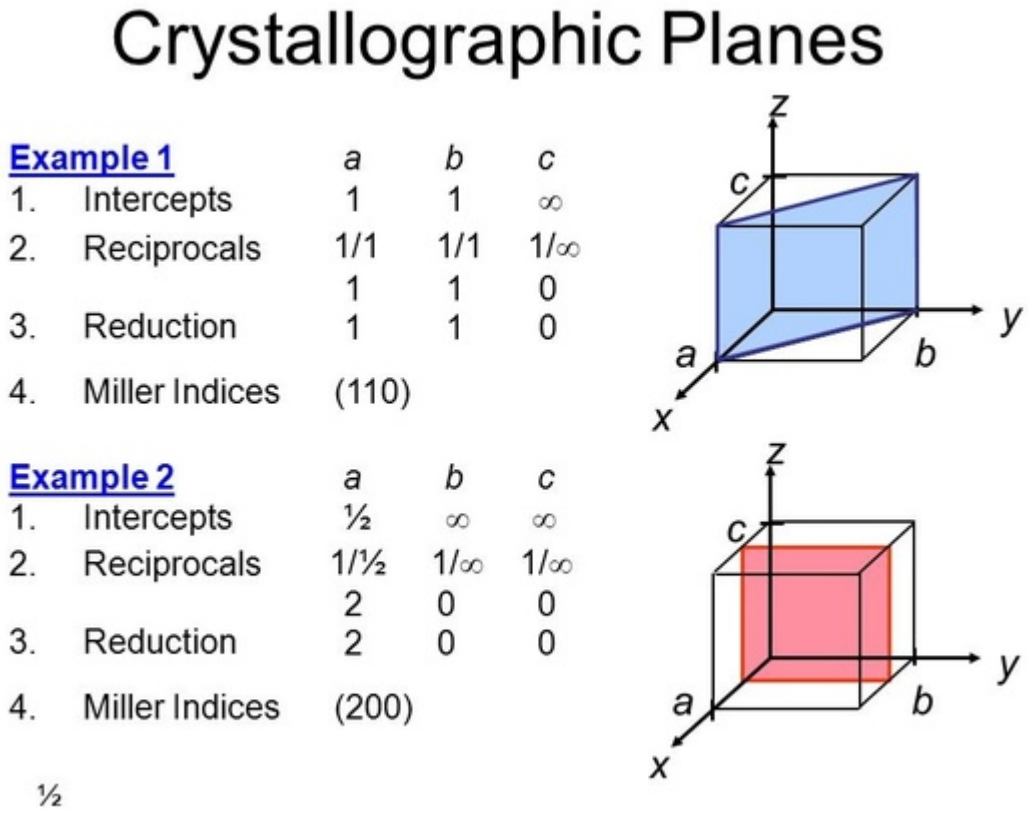
The smallest group of particles in the material that constitutes this repeating pattern is known as the *unit cell*. It fully reflects the symmetry and structure of the entire crystal, resulting from the repetitive translation of the unit cell along its principal axes. The translation vectors define the nodes of the *Bravais lattice*, an imaginary concept.



The crystal structure is formed when the group of atoms or molecules is arranged identically at the lattice points. This group of atoms or molecules is called the *basis*.

4.3 Lattice Planes and Directions

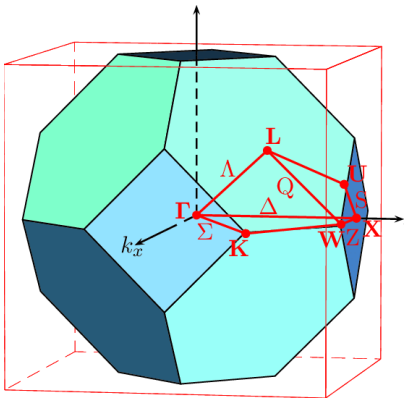
The notation system of *Miller indices* represents planes in crystal (Bravais) lattices. A family of lattice planes is determined by three integers h, k, and l, which are the Miller indices.



4.4 Reciprocal Lattice

In addition to the direct lattice in ordinary space, it is crucial to consider the *reciprocal lattice* in dual (or reciprocal) space when studying crystals. The reciprocal lattice is defined by three primitive vectors $\vec{g_1}$, $\vec{g_2}$, and $\vec{g_3}$.

The *Brillouin zone* is a uniquely defined primitive cell in reciprocal space.



4.5 Crystal Symmetry and Space Groups

The space groups in three dimensions are formed from combinations of the 32 crystallographic point groups with the 14 Bravais lattices, each belonging to one of the 7 lattice systems.

4.6 Additional Resources

To explore and analyze problems of structural and mathematical crystallography, solid-state physics, and structural chemistry, the Bilbao Crystallographic Server offers an open-access website with an online crystallographic database and programs.

4.7 Example

Open a CIF (./CsPbI3.cif) file and find the lattice constants and basis vectors.

CHAPTER

FIVE

ASE BASICS

5.1 Section Three: ASE Basic Concepts

5.2 Part 1: Graphical Interface with ASE

In this part, we'll dive into the Atomic Simulation Environment (ASE) graphical interface to explore its powerful graphical tools. The graphical interface allows us to visualize atomic structures, manipulate them, and perform basic operations visually.

5.3 Step 1: Launching ASE's GUI

To get started, let's launch the ASE graphical interface. Open your Python environment, and we'll use the following Python code:

```
from ase.visualize import view

# Create an example atomic structure (you can use your own coordinates)
# For example, let's create a simple hydrogen molecule
from ase import Atoms
atoms = Atoms("H2", positions=[[0, 0, 0], [0, 0, 0.74]])

# Visualize the structure
view(atoms)
```

ASE's graphical interface should now open, displaying the atomic structure of the hydrogen molecule. You can rotate, zoom, and interact with the 3D visualization to explore the molecule.

5.4 Step 2: Basic Visualization and Manipulation

In the GUI, you can access various tools to manipulate the atomic structure:

- Use the mouse to rotate, pan, and zoom the 3D view.
- Right-click on atoms to see their properties and edit their attributes.
- Select and move atoms by left-clicking and dragging.
- Use the "Add" tool to add new atoms to the structure.
- Delete atoms using the "Remove" tool.

5.5 Step 3: Periodic Models and Reciprocal Lattice

ASE also allows us to work with periodic systems. Let's explore how to create periodic models and visualize their reciprocal lattice.

- Create a periodic model of a crystal, such as FCC or BCC.
- Visualize the reciprocal lattice of the crystal to understand its Brillouin zone.

5.6 Step 4: Building Nanostructures and Nanotubes

ASE makes it easy to construct nanostructures and nanotubes. Let's build a carbon nanotube as an example:

- Create a graphene sheet.
- Roll the graphene sheet to form a carbon nanotube.
- Visualize the nanotube's structure and properties.

5.7 Part 2: Command Line Interface with ASE

In this part, we'll explore the ASE command line interface using the terminal or Jupyter (or IPython) to interact with ASE programmatically.

5.8 Step 1: Terminal or IPython Setup

If you haven't installed Jupyter or IPython, you can install it using the following command:

```
pip install jupyter
```

or

```
pip install ipython
```

To access the command line interface, launch Jupyter or IPython in your terminal:

```
jupyter notebook
```

or

```
ipython
```

5.9 Step 2: Creating Atomic Structures with the Atoms Object

ASE represents atomic structures using the $Atoms$ object. Let's create and manipulate atomic structures programmatically:

```
from ase import Atoms

# Create a water molecule
water = Atoms("H2O", positions=[[0, 0, 0], [0.74, 0.74, 0], [0, 0.74, 0]])

# Print the atomic structure
print(water)
```

5.10 Step 3: Reading and Writing Atomic Structure Files

ASE supports various file formats for reading and writing atomic structures. Let's explore how to read and write structures:

```
# Save the structure to a file in XYZ format
water.write("water.xyz")

# Load a structure from a file
from ase.io import read
loaded_water = read("water.xyz")
```

5.11 Step 4: Using Calculators for Energy Calculations

ASE provides calculators to perform energy and force calculations for atomic structures. Let's use a calculator to optimize the water molecule's geometry:

```
from ase.calculators.emt import EMT

# Set up the EMT calculator
water.set_calculator(EMT())

# Optimize the structure
from ase.optimize import BFGS
optimizer = BFGS(water)
optimizer.run(fmax=0.01)
```

5.12 Step 5: Creating Supercells and Applying Transformations

ASE allows you to create supercells by replicating an existing structure. We'll apply transformations to structures and create supercells programmatically:

```
# Create a supercell by replicating the water molecule
supercell = water * (2, 2, 2)

# Apply a transformation matrix to rotate the structure
from ase import matrix
transformation_matrix = matrix.transformation_matrix([1, 1, 1], [0, 0, 1])
transformed_water = water.copy()
transformed_water.set_cell(transformed_water.cell @ transformation_matrix)
```

Now you have explored ASE's graphical and command line interfaces, including periodic models, reciprocal lattice, nanostructures, and important concepts like the $Atoms$ object, file IO, calculators, and supercells. These skills provide a strong foundation for your atomistic simulation journey!

That's it for this section. In the next section, we'll delve into more advanced concepts using ASE for solid-state physics and quantum chemistry simulations.

Happy exploring and happy learning with Python for Atomistic Simulation!

CHAPTER

SIX

DENSITY FUNCTIONAL THEORY (DFT)

6.1 Introduction to Density Functional Theory (DFT)

In this section, we'll introduce you to the fundamental principles of Density Functional Theory (DFT), a widely used approach in computational materials science and quantum chemistry.

1. What is Density Functional Theory (DFT)? - DFT basics: Electronic density, total energy, and the Hohenberg-Kohn theorem. - Kohn-Sham approach: Solving the many-body Schrödinger equation using fictitious non-interacting electrons.

$$\hat{H}_{KS}\psi_i = \left(-\frac{\hbar^2}{2m}\nabla^2 + V_{ext} + V_H[\rho] + V_{XC}[\rho]\right)\psi_i = \epsilon_i\psi_i$$

   In the Kohn-Sham approach, we map the interacting system to an auxiliary non-interacting system of electrons with effective potentials.

2. The Kohn-Sham Equations - Kohn-Sham equations derivation. - Self-consistent field (SCF) method for finding the electronic structure.

$$V_{eff}[\rho](r) = V_{ext}(r) + \int \frac{\rho(r')}{|r-r'|}dr' + \frac{\delta E_{XC}}{\delta\rho(r)}$$

   The Kohn-Sham equations represent a set of equations where the electron wavefunctions and energies are obtained self-consistently to minimize the total energy of the system.

6.2 Exchange-Correlation Functionals in DFT

In this section, we'll focus on exchange-correlation functionals, a crucial aspect of DFT that accounts for electron-electron interactions.

1. Introduction to Exchange-Correlation Functionals - The role of exchange and correlation in DFT. - Local Density Approximation (LDA) and Generalized Gradient Approximation (GGA).

$$E_{XC}^{LDA}[\rho] = \int \epsilon_{XC}^{LDA}[\rho](r)\rho(r)drE_{XC}^{GGA}[\rho] = \int \epsilon_{XC}^{GGA}[\rho](r)\rho(r)dr$$

   Exchange-correlation functionals capture the quantum mechanical exchange and correlation effects of electrons, essential for an accurate description of the electronic system.

2. Beyond LDA and GGA - Meta-GGA and hybrid functionals. - Overview of hybrid functionals like B3LYP and PBE0.

$$E_{XC}^{meta-GGA}[\rho] = \int \epsilon_{XC}^{meta-GGA}[\rho](r)\rho(r)drE_{XC}^{hybrid}[\rho] = (1-\alpha)E_{XC}^{GGA}[\rho] + \alpha E_{XC}^{HF}[\rho]$$

   Beyond LDA and GGA, meta-GGA and hybrid functionals offer improved accuracy for specific systems, combining the advantages of both local and non-local functionals.

6.3 Solving Kohn-Sham Equations with Different Basis Sets

In this section, we'll explore different basis sets used to solve the Kohn-Sham equations in Density Functional Theory.

1. Plane Wave Basis Set - Introduction to the plane wave basis set. - Solving Kohn-Sham equations using plane waves. - Codes that use plane wave basis sets: Quantum ESPRESSO, VASP.

   Plane wave basis sets provide an efficient representation of electronic wavefunctions in periodic systems, making them well-suited for solid-state simulations.

2. Finite Basis Sets - Introduction to finite basis sets (e.g., Gaussian basis sets). - Solving Kohn-Sham equations using finite basis sets. - Codes that use finite basis sets: NWChem, Gaussian.

   Finite basis sets are widely used in quantum chemistry calculations for molecular systems, offering flexibility and accuracy for localized electronic states.

3. Pseudopotentials and Basis Set Quality - Pseudopotentials and their role in reducing the computational cost. - Evaluating the quality of basis sets for accuracy and efficiency.

   Pseudopotentials approximate the effect of core electrons, reducing the computational burden while maintaining accuracy for valence electrons.

6.4 Conclusion

Congratulations on completing the course on Density Functional Theory (DFT) and different basis sets! You've gained essential knowledge about DFT's principles, exchange-correlation functionals, and the Kohn-Sham equations. Additionally, you've explored different basis sets like plane waves and finite basis sets used to solve the Kohn-Sham equations in various DFT codes.

DFT, along with the appropriate basis set, is a powerful tool in materials science and quantum chemistry, enabling accurate and efficient simulations of complex systems. Continue exploring and applying DFT with different basis sets to solve real-world problems, and you'll become a proficient user in the exciting world of computational science!

CHAPTER

SEVEN

QUANTUM ESPRESSO CALCULATIONS

Welcome to the course on "Introduction to Quantum ESPRESSO Calculations and Materials Simulation" using the Atomic Simulation Environment (ASE). In this course, you will learn the basics of quantum mechanical calculations using the Quantum ESPRESSO software package and how to perform materials simulations with ASE.

7.1 Prerequisites

Before starting this course, make sure you have the following installed on your system:

1. Python with pip (to install ASE).
2. Quantum ESPRESSO (installed separately from the ASE).

7.2 Downloading Crystal Structures from Materials Project

In this section, we will download crystal structures of GaAs, Si, and Al from the Materials Project database and read them using ASE.

1. Install ASE - You can install ASE using pip with the following command:

```
pip install ase
```

2. Import ASE modules - Import the necessary ASE modules to work with crystal structures:

```
from ase.io import read
```

3. Download CIF files - Download the CIF files for GaAs, Si, and Al from the Materials Project website (https://materialsproject.org/). - Save the CIF files in your working directory.
4. Read CIF files - Use the $read$ function from ASE to read the CIF files and create ASE $Atoms$ objects:

```
gaas = read('gaas.cif')
si = read('si.cif')
al = read('al.cif')
```

7.3 Creating Quantum ESPRESSO Calculator in ASE

Now that we have our crystal structures loaded as $Atoms$ objects, we will create Quantum ESPRESSO calculators to perform electronic structure calculations.

1. Import Quantum ESPRESSO calculator - Import the $Espresso$ calculator from ASE to interface with Quantum ESPRESSO:

```
from ase.calculators.espresso import Espresso
```

2. Set up Quantum ESPRESSO calculator - Create a Quantum ESPRESSO calculator for GaAs:

```
# Set up the Quantum ESPRESSO calculator for GaAs
gaas_calc = Espresso(pw=400,       # Plane wave cutoff energy in eV
                     calculation='scf', # Self-consistent field calculation
                     kpts=(4, 4, 4), # Monkhorst-Pack k-point mesh
                     xc='PBE',     # Exchange-correlation functional (PBE␣
↪for Perdew-Burke-Ernzerhof)
                     outdir='gaas_calculation', # Output directory for␣
↪Quantum ESPRESSO
                     pseudo_dir='pseudo',  # Directory containing␣
↪pseudopotential files
                     pseudopotentials={'Ga': 'Ga.pbe-n-kjpaw.UPF', 'As':
↪'As.pbe-n-kjpaw.UPF'} # Pseudopotentials for elements
                     )

# Attach the calculator to the GaAs structure
gaas.set_calculator(gaas_calc)
```

- Repeat the above steps to create calculators for Si and Al with appropriate pseudopotential files.

7.4 Running Quantum ESPRESSO Calculations

Now that we have set up the Quantum ESPRESSO calculators, let's run the electronic structure calculations for GaAs, Si, and Al.

1. Calculate the potential energy of GaAs - Use the $get_potential_energy()$ method of the $Atoms$ object to perform the calculation:

```
gaas_energy = gaas.get_potential_energy()
print('GaAs total energy:', gaas_energy, 'eV')
```

- Repeat the above step for Si and Al.

2. Accessing Additional Results - ASE allows us to extract additional results from the Quantum ESPRESSO calculations, such as forces, stress tensor, and electronic structure information.

Note: Students are encouraged to explore the ASE documentation to learn how to extract and analyze additional information from Quantum ESPRESSO calculations.

7.5 Density of States

The density of states (DOS) is a fundamental concept in condensed matter physics and quantum mechanics. It provides valuable insights into the distribution of energy levels available to electrons in a material. The DOS is defined mathematically as:

$$g(\varepsilon) = \sum_n \langle\psi_n|\psi_n\rangle\delta(\varepsilon - \varepsilon_n),$$

where ε_n represents the eigenvalue of the eigenstate $|\psi_n\rangle$.

In a given orthonormal basis, the DOS can be expressed as a sum over basis functions:

$$\begin{aligned} g(\varepsilon) &= \sum_i g_i(\varepsilon), \\ g_i(\varepsilon) &= \sum_n \langle\psi_n|i\rangle\langle i|\psi_n\rangle\delta(\varepsilon - \varepsilon_n), \end{aligned}$$

where $g_i(\varepsilon)$ is called the *projected density of states* (PDOS). Additionally, the DOS can also be represented as an integral over the spatial coordinates:

$$\begin{aligned} g(\varepsilon) &= \int \mathrm{d}r g(r, \varepsilon), \\ g(r, \varepsilon) &= \sum_n \langle\psi_n|r\rangle\langle r|\psi_n\rangle\delta(\varepsilon - \varepsilon_n), \end{aligned}$$

where $g(r, \varepsilon)$ is known as the *local density of states* (LDOS).

It is important to note that the completeness of the selected basis sets is crucial in defining the DOS. We have $1 = \sum_i |i\rangle\langle i|$ for discrete basis sets and $1 = \int \mathrm{d}r|r\rangle\langle r|$ for continuous basis sets.

The DOS has several important properties that make it a valuable tool for understanding electronic structure:

1. **Total Number of Electrons (N):** The integral of DOS below the Fermi energy gives the total number of electrons (N) in the system:

   $$\int \mathrm{d}\varepsilon\, n_F(\varepsilon) g(\varepsilon) = N,$$

   where $n_F(\varepsilon)$ is the Fermi distribution function.

2. **Electron Density (n(r)):** The integral of LDOS multiplied by the Fermi distribution provides the electron density at a given point in space (r):

   $$\int \mathrm{d}\varepsilon\, n_F(\varepsilon) g(r, \varepsilon) = n(r).$$

The DOS is a powerful tool in solid-state physics and materials science for understanding the electronic properties of materials and predicting their behavior under different conditions.

Note: In the above definition, the k-dependence of eigenfunctions has been dropped for simplicity. For a more accurate definition, one can refer to "*Quantum Chemistry of Solids: The LCAO First Principles Treatment of Crystals*" by Robert A. Evarestov.

To summarize, the DOS provides crucial information about the distribution of energy levels and electronic states in a material, making it a fundamental concept in the study of condensed matter and quantum systems.

CHAPTER

EIGHT

NWCHEM CALCULATIONS WITH ASE

Welcome to the course on "Running NWChem Calculations and Analyzing Molecular Orbitals and Charge Density" using the Atomic Simulation Environment (ASE). In this course, you will learn how to perform quantum chemical calculations with NWChem and extract molecular orbitals and charge density using ASE.

8.1 Prerequisites

Before starting this course, make sure you have the following installed on your system:

1. Python with pip (to install ASE).
2. NWChem (installed separately from the ASE).

8.2 Downloading Amino Acid Structures from PDB

In this section, we will download the structures of amino acids from the Protein Data Bank (PDB) and read them using ASE.

1. Install ASE - You can install ASE using pip with the following command:

```
pip install ase
```

2. Import ASE modules - Import the necessary ASE modules to work with molecular structures:

```
from ase.io import read
```

3. Download PDB files - Download the PDB files for amino acids from the Protein Data Bank (https://www.rcsb.org/). - Save the PDB files in your working directory.
4. Read PDB files - Use the $read$ function from ASE to read the PDB files and create ASE $Atoms$ objects:

```
amino_acid1 = read('amino_acid1.pdb')
amino_acid2 = read('amino_acid2.pdb')
# Add more amino acids as needed
```

8.3 Creating NWChem Calculator in ASE

Now that we have our amino acid structures loaded as $Atoms$ objects, we will create NWChem calculators to perform quantum chemical calculations.

1. Import NWChem calculator - Import the $Nwchem$ calculator from ASE to interface with NWChem:

```
from ase.calculators.nwchem import Nwchem
```

2. Set up NWChem calculator - Create an NWChem calculator for the first amino acid:

```
# Set up the NWChem calculator for amino acid1
amino_acid1_calc = Nwchem(
    label='amino_acid1', # Calculation label
    task='scf',          # Self-consistent field calculation
    basis='6-31G',       # Basis set (6-31G for a balanced compromise␣
↪between accuracy and efficiency)
    charge=0,            # Charge of the system (0 for neutral amino acid)
    multiplicity=1,      # Multiplicity of the system (1 for singlet)
    xc='B3LYP',          # Exchange-correlation functional (B3LYP for good␣
↪accuracy in molecular properties)
    coordinates='amino_acid1.xyz' # XYZ file with atomic coordinates
)

# Attach the calculator to the amino acid1 structure
amino_acid1.set_calculator(amino_acid1_calc)
```

- Repeat the above steps to create calculators for other amino acids with appropriate basis sets and parameters.

8.4 Running NWChem Calculations

Now that we have set up the NWChem calculators, let's run the quantum chemical calculations for the amino acids.

1. Calculate the potential energy of amino acid1 - Use the $get_potential_energy()$ method of the $Atoms$ object to perform the calculation:

```
aa1_energy = amino_acid1.get_potential_energy()
print('Amino Acid 1 total energy:', aa1_energy, 'eV')
```

- Repeat the above step for other amino acids.

8.5 Analyzing Molecular Orbitals and Charge Density

ASE allows us to extract molecular orbitals and charge density from NWChem calculations.

1. Access molecular orbitals - Use the $get_molecular_orbitals()$ method of the $Atoms$ object to obtain molecular orbital information:

```
# Get molecular orbitals for amino acid1
mo_coefficients, mo_energies, mo_occ = amino_acid1.get_molecular_orbitals()
```

(continues on next page)

(continued from previous page)

```
# Print molecular orbital energies for amino acid1
print('Molecular Orbital Energies for Amino Acid 1:')
print(mo_energies)
```

- Repeat the above step for other amino acids.

2. Access charge density - Use the $get_atomic_numbers()$ and $get_atomic_spins()$ methods to extract atomic numbers and spins for the system:

```
# Get atomic numbers and spins for amino acid1
atomic_numbers = amino_acid1.get_atomic_numbers()
atomic_spins = amino_acid1.get_atomic_spins()
```

- Use the above information to extract and analyze the charge density from the NWChem calculation results.

8.6 Conclusion

Congratulations! You have completed the course on "Running NWChem Calculations and Analyzing Molecular Orbitals and Charge Density" using ASE. You have learned how to set up NWChem calculators in ASE, perform quantum chemical calculations for amino acids, and extract molecular orbital and charge density information from the calculations.

Continue your exploration of computational chemistry and materials science with NWChem and ASE. Apply the knowledge gained in this course to study more complex molecular systems, perform electronic structure calculations, and analyze molecular properties. Happy computational chemistry with NWChem and ASE!

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search