FYSC23

Local Density of States, Surfaces, and Adsorbates

Authors
Fredrik Bergelv
Max Eriksson
`fredrik.bergelv@live.se`
`maxerikss@gmail.com`

LUND UNIVERSITY

February 12, 2025

1 LDOS 1D

1.1 First problem

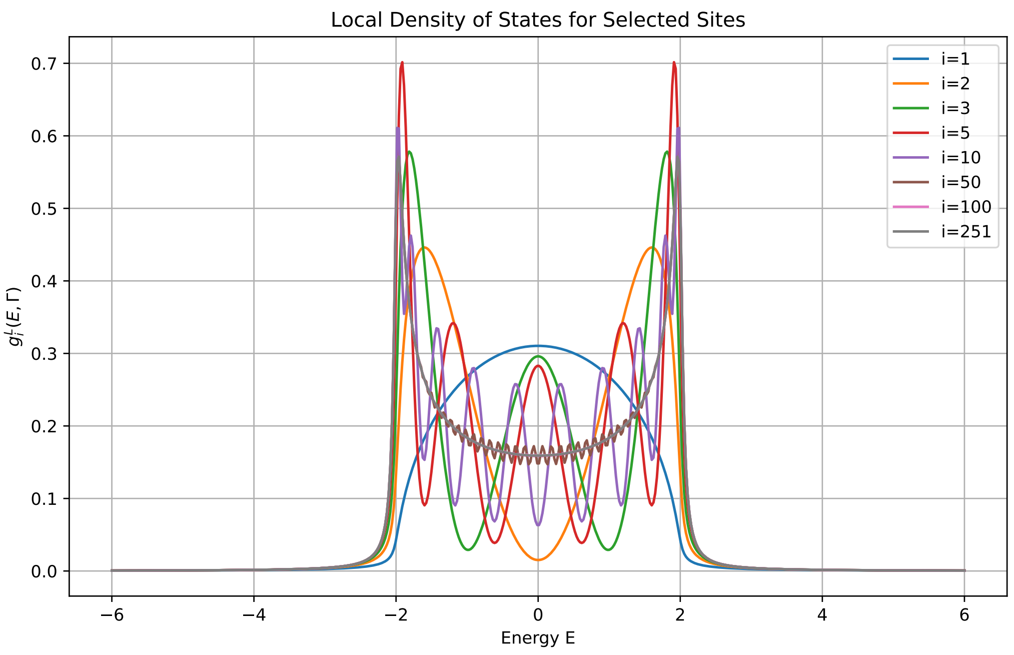

Figure 1: Local density of states plotted for a 1D chain of length $N = 501$. The index i corresponds to the site number in the chain.

A1 Noticeable in Fig. 1 is that the LDOS is symmetric around the central energy. For increasing i the number of nodes in the LDOS oscillation increases. The central amplitude also decreases while the amplitude at $E = \pm 2$ increases.

1.2 Second problem

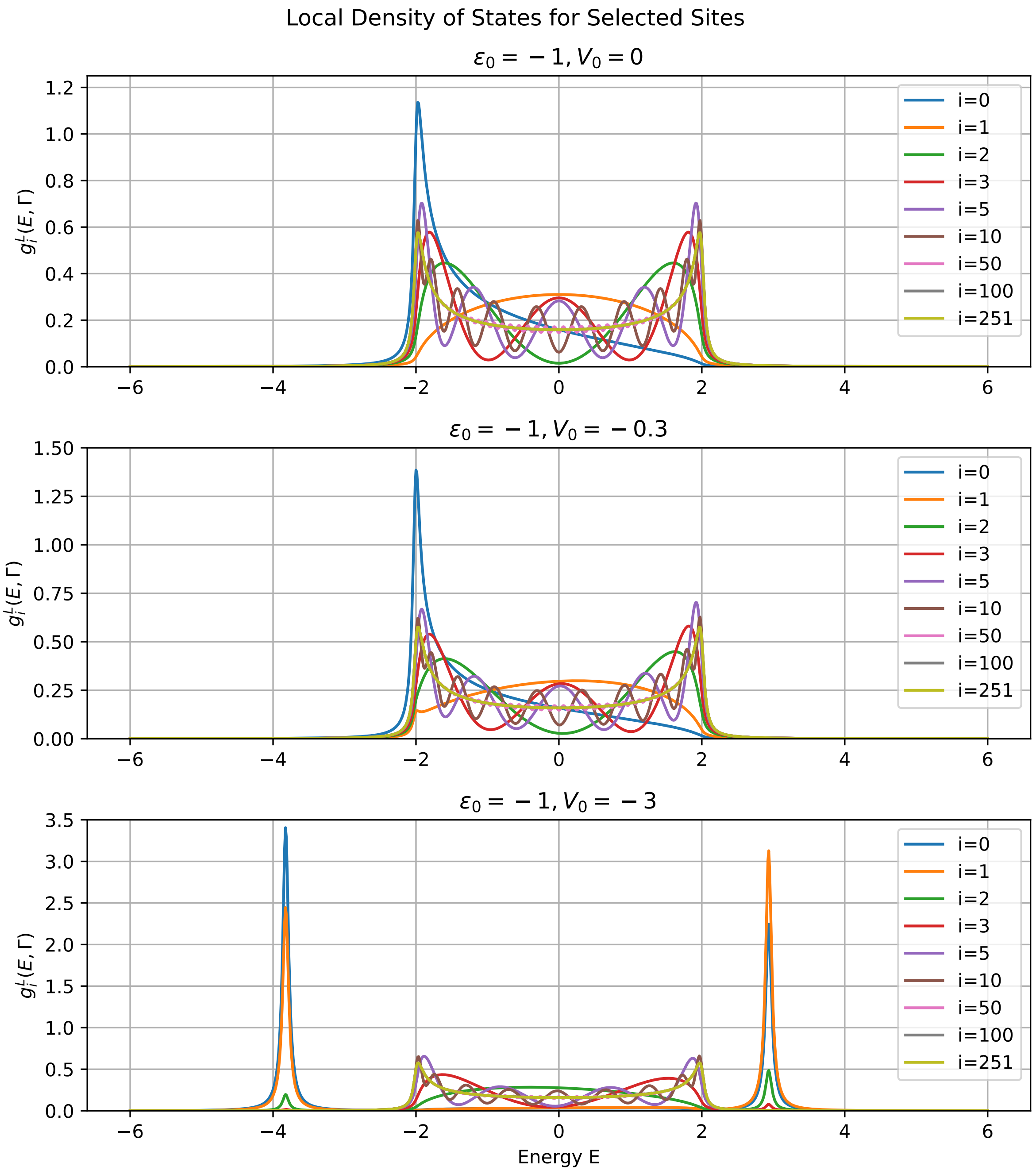

Figure 2: LDOS plotted for a 1D chain of length $N = 501$ with surface at site $i = 1$ and adsorbate at site $i = 0$. The index i corresponds to the site number in the chain.

A2 For the first case, the top plot in Fig. 2, The LDOS is the same for all sites as in task 1, except at the site $i = 0$. That is, the adsorbate atom. This is reasonable since there is no coupling between the adsorbate and the chain since $V_0 = 0$. For the second case, the middle plot in Fig. 2, there is some coupling between the adsorbate and the chain which is mainly noticeable at sites close to the adsorbate. At energy approximately -2 the LDOS is lower for site $i = 1$, so the adsorbate occupies some possible modes for the first atom in the chain. However, most of the other sites are barely unaffected. The effect is hard to see beyond site $i = 3$. For the last case, the bottom plot in Fig. 2, the adsorbate has a much higher coupling of $V_0 = -3$ and is clearly affecting the chain. The sites closest to the adsorbate have modes considerably lower and higher than they had without the adsorbate. However, at site $i = 10$ it is hard to see the effect of the adsorbate. The adsorbate is clearly affecting the atoms closest to it, but it not only the nearest neighbour atom, even though it only has a coupling to that one.

It is also clear that for the adsorbate the LDOS is centred around a lower energy as the coupling potential decreases. The spike is of the LDOS is also much sharper for a stronger coupling. It looks like when the coupling is strong the chain closest to the adsorbate is almost decoupled from the rest of the chain and behaves more like the adsorbate itself than the other atoms in the chain.

2 LDOS 2D

2.1 Third problem

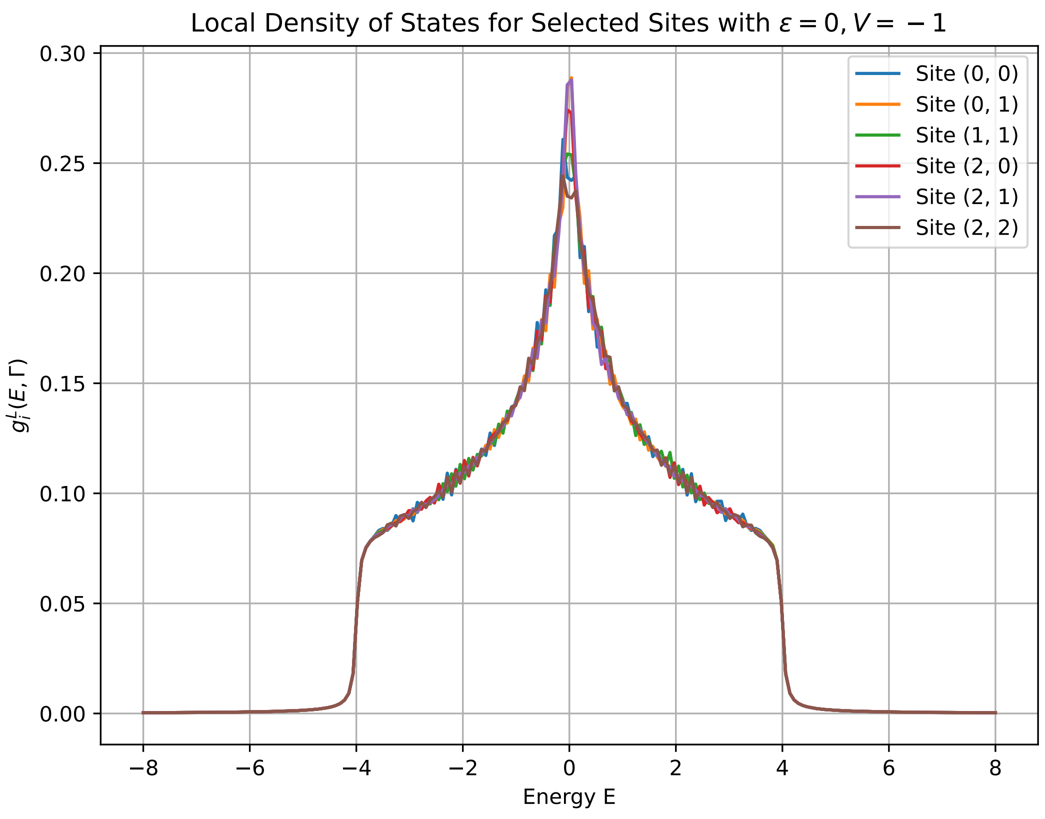

Figure 3: This figure...

2.2 Fourth problem

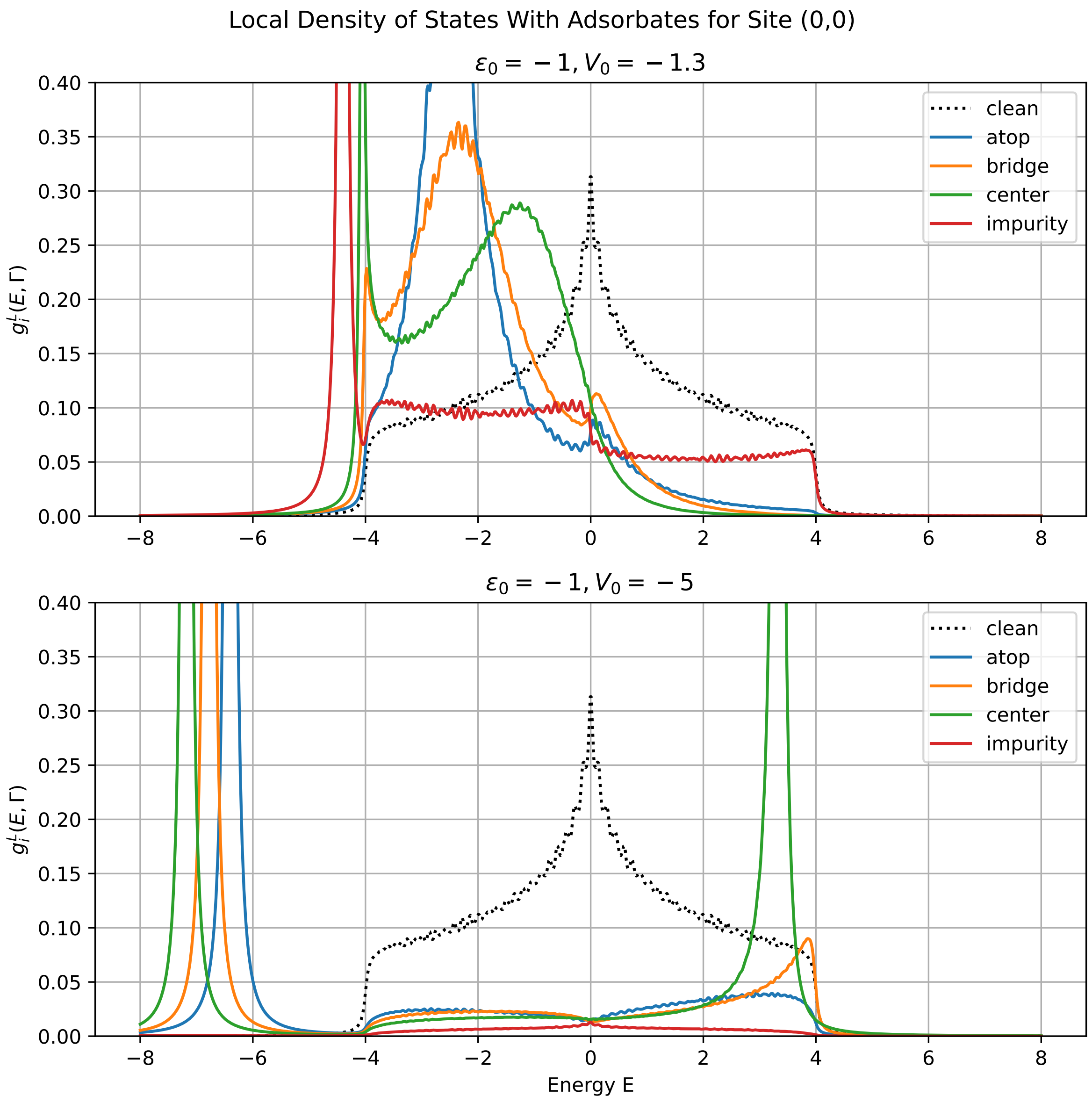

Figure 4: This figure...

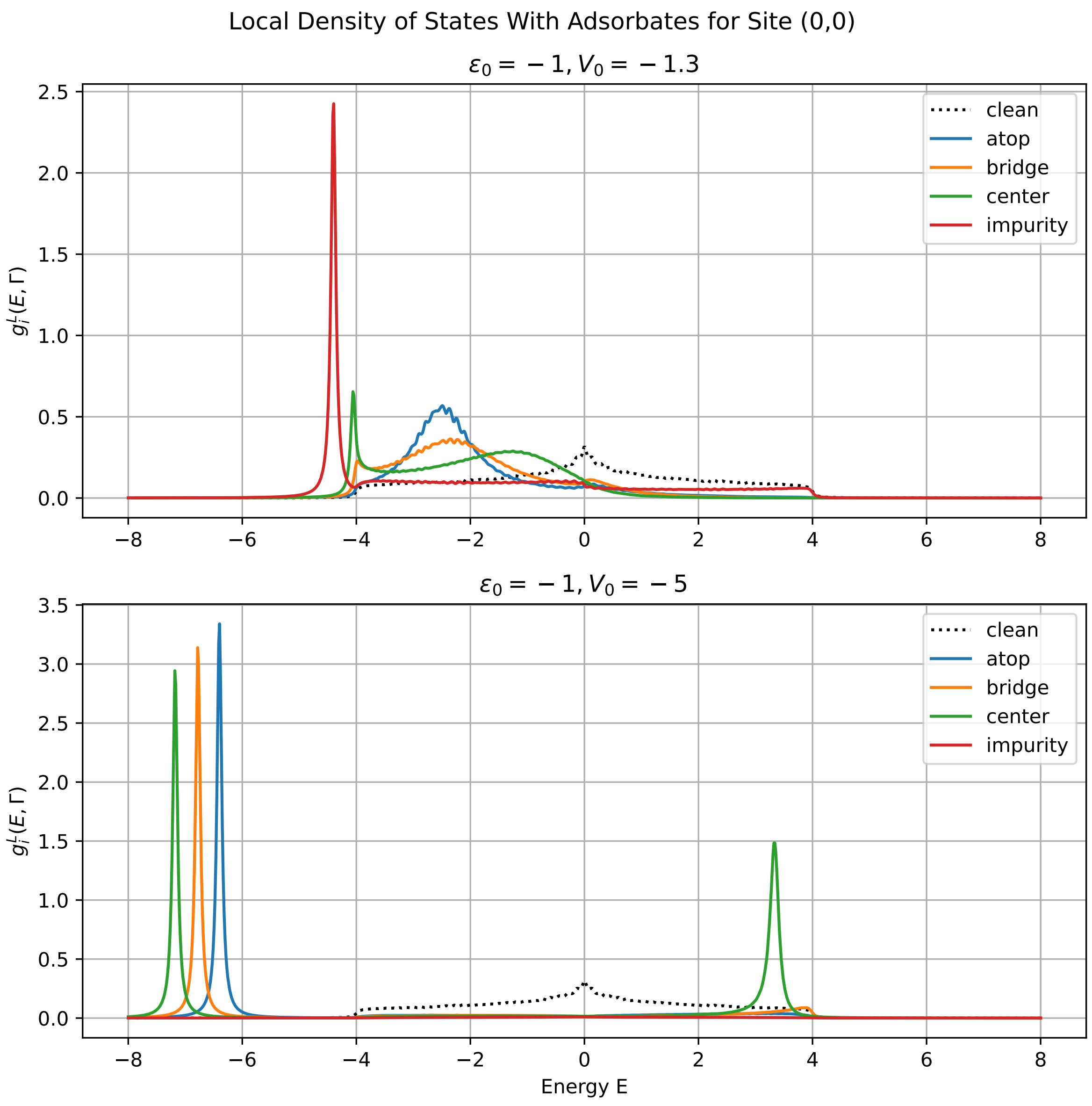

Figure 5: This figure...

3 The code

3.1 First problem

```
import numpy as np
import matplotlib.pyplot as plt

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 10 10:28:49 2025

@author: fredrik
"""

N = 501         # Chain length
V = -1          # Hopping term
epsilon = 0     # Energy at site
gamma = 0.05    # Broadening factor
LDOS_sites = [1, 2, 3, 5, 10, 50, 100, 251] # Sites we want to plot
energy_range = np.linspace(-6, 6, 500)  # Energy range for LDOS calculation


def hamiltonian(n, epsilon, V):
    "Create a hamilitonian"
    upper =  np.diag(V * np.ones(n-1), 1)
    middle = np.diag(epsilon * np.ones(n), 0)
    lower = np.diag(V * np.ones(n-1), -1)
    return upper + middle + lower

def sums(gamma, eigenvec, lamb, energy, eigenergy, site):
    "Each term in the sum"
    return (gamma / np.pi) * (eigenvec[site-1, lamb] ** 2) / ((energy - eigenergy[lamb])**2 + gamma**

# Find the hamltonian
H = hamiltonian(N, epsilon, V)

# Find eigenergies and eigenvectors
eigenenergies, eigenvectors = np.linalg.eigh(H)

# initlaize the LDOS as a dictonary
LDOS = {site: np.zeros(len(energy_range)) for site in LDOS_sites}

# Do the calculation for each site which we are intrested in
for site in LDOS_sites:
    # Make a loop where we look through each position with the energy at that poition
    for i, E in enumerate(energy_range):
        # Calculate the sum
        for lamb in range(N):
            LDOS[site][i] += sums(gamma=gamma, eigenvec=eigenvectors,
                                  lamb=lamb, energy=E,
                                  eigenergy=eigenenergies, site=site)

# Plot LDOS for the selected sites
plt.figure(figsize=(10, 6))
```

```
for site in LDOS_sites:
    plt.plot(energy_range, LDOS[site], label=f"i={site}")
plt.xlabel("Energy E")
plt.ylabel(r"$g^L_{i}(E, \Gamma)$")
plt.title("Local Density of States for Selected Sites")
plt.legend()
plt.grid()

plt.savefig("Comp_Proj1/Figures/task1.pdf")
plt.show()
```

3.2 Second problem

```
    #!/usr/bin/env python3
    # -*- coding: utf-8 -*-
    """
    Created on Mon Feb 10 10:28:49 2025

    @author: fredrik
    """

    import numpy as np
    import matplotlib.pyplot as plt

    N = 501         # Chain length
    V = -1          # Hopping term
    epsilon = 0     # Energy at site
    gamma = 0.05    # Broadening factor
    LDOS_sites = [0, 1, 2, 3, 5, 10, 50, 100, 251] # Sites we want to plot
    energy_range = np.linspace(-6, 6, 1000)  # Energy range for LDOS calculation

    def hamiltonian(n, epsilon, V, e0, v0):
        "Create a Hamiltonian"
        upper = np.diag(V * np.ones(n-1), 1)
        middle = np.diag(epsilon * np.ones(n), 0)
        lower = np.diag(V * np.ones(n-1), -1)
        H = upper + middle + lower
        H[0, -1] = v0
        H[-1, 0] = v0
        H[-1, -1] = e0
        return H

    def sums(gamma, eigenvec, lamb, energy, eigenergy, site):
        "Each term in the sum"
        return (gamma / np.pi) * (eigenvec[site-1, lamb] ** 2) / ((energy - eigenergy[lamb])**2 + gam

    def compute_LDOS(LDOS_sites, energy_range, H):
        "Function for fining the LDOS"
        # Find eigenenergies and eigenvectors
        eigenenergies, eigenvectors = np.linalg.eigh(H)

        # Initialize LDOS as a dictionary
        LDOS = {site: np.zeros(len(energy_range)) for site in LDOS_sites}

        # Compute LDOS for each site of interest
        for site in LDOS_sites:
            for i, E in enumerate(energy_range):
                for lamb in range(N):
                    LDOS[site][i] += sums(gamma=gamma, eigenvec=eigenvectors,
                                          lamb=lamb, energy=E,
                                          eigenergy=eigenenergies, site=site)
        return LDOS # Return the LDOS

    # Different parameter sets (e0, v0)
    param_list = [(-1, 0), (-1, -0.3), (-1, -3)]

    # Create figure with 3 subplots
    fig, axes = plt.subplots(3, 1, figsize=(8, 9))
    fig.suptitle("Local Density of States for Selected Sites")

    # Loop over different parameter sets and plot in subplots
    for idx, (e0, v0) in enumerate(param_list):
        H = hamiltonian(N, epsilon, V, e0, v0)  # Compute Hamiltonian
        LDOS = compute_LDOS(LDOS_sites, energy_range, H)  # Compute LDOS

        for site in LDOS_sites:
            axes[idx].plot(energy_range, LDOS[site], label=f"i={site}")

        axes[idx].set_title(f"$\epsilon_0={e0}, V_0={v0}$")
        axes[idx].set_ylabel(r"$g^L_{i}(E, \Gamma)$")
        axes[idx].legend()
        axes[idx].grid()

    # Set common x-axis
    axes[0].set_ylim(0, 1.25)
    axes[1].set_ylim(0, 1.5)
    axes[2].set_ylim(0, 3.5)
    axes[2].set_xlabel("Energy E")
    plt.tight_layout()

    plt.savefig("Comp_Proj1/Figures/task2.pdf")
    plt.show()
```

3.3 The third problem

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 10 10:51:38 2025

@author: fredrik
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import csr_matrix
import time

start_time = time.time()
```

```
Nl = 81         # Lattice size along one dimension
Ns = Nl**2      # Total number of atoms
V = -1          # Hopping parameter
epsilon = 0     # On-site energy
gamma = 0.05    # Broadening factor

Nh = (Nl + 1) // 2
Nhp = (Nl - 1) // 2

# Sites of interest for LDOS calculation
LDOS_sites = [(0, 0), (0, 1), (1, 1), (2, 0), (2, 1), (2, 2)]
energy_range = np.linspace(-8, 8, 200)


def coord_to_index(i, j):
    "Convert (i', j') coordinates to matrix index m."
    "This was done by the method provided"
    ibar = i + Nhp
    jbar = j + Nhp
    return Nl*ibar + jbar


def get_neighbors(i, j):
    "This function check if there are any neighbors nearby that exsists"
    # List all possible nearest neighbors (up, down, left, right)
    neighbors = [(i-1,j), (i+1,j), (i,j-1), (i,j+1)]

    valid_neighbors = []

    # Check each potential neighbor to ensure it is within the valid range
    for ni, nj in neighbors:
        if -Nhp <= ni <= Nhp and -Nhp <= nj <= Nhp:
            valid_neighbors.append((ni, nj))

    return valid_neighbors # Return the list of valid nearest neighbors


def hamiltonian(Ns, epsilon, V):
    "Construct the Hamiltonian matrix for a 2D square lattice."
    H = csr_matrix((Ns, Ns), dtype=complex).tolil()
    H.setdiag([epsilon] * (Ns+1))
    for i in range(-Nhp, Nhp + 1):
        for j in range(-Nhp, Nhp + 1):
            m = coord_to_index(i, j)
            for ni, nj in get_neighbors(i, j):
                n = coord_to_index(ni, nj)
                H[m, n] = V  # Nearest-neighbor hopping
    return H.tocsr()


def sums(gamma, eigenvecs, lamb, energy, eigvals, site):
    "Compute each term in the LDOS sum."
    return (gamma / np.pi) * (np.abs(eigenvecs[site, lamb]) ** 2) / ((energy - eigvals[lamb])**2


def compute_LDOS(LDOS_sites, energy_range, H):
    "Compute the Local Density of States (LDOS)."
```

```
        # Find eigenenergies and eigenvectors
        eigenenergy, eigenvectors = np.linalg.eigh(H.toarray()) # Convert sparse matrix to dense calc

        # Initialize LDOS as a dictionary
        LDOS = {site: np.zeros(len(energy_range)) for site in LDOS_sites}

        # Compute LDOS for each site of interest
        for site in LDOS_sites:
            site_index = coord_to_index(site[0], site[1])
            for i, E in enumerate(energy_range):
                for lamb in range(Ns):
                    LDOS[site][i] += sums(gamma=gamma, eigenvecs=eigenvectors,
                                          lamb=lamb, energy=E,
                                          eigvals=eigenenergy, site=site_index)
        return LDOS

    # Compute Hamiltonian and LDOS
    H = hamiltonian(Ns, epsilon, V)
    LDOS = compute_LDOS(LDOS_sites, energy_range, H)

    # Plot LDOS
    plt.figure(figsize=(8, 6))
    for site in LDOS_sites:
        plt.plot(energy_range, LDOS[site], label=f"Site {site}")
    plt.xlabel("Energy E")
    plt.ylabel(r"$g^L_{i}(E, \Gamma)$")
    plt.title(f"Local Density of States for Selected Sites with $\epsilon={epsilon}, V={V}$")
    plt.legend()
    plt.grid()

    plt.savefig("Comp_Proj1/Figures/task3.pdf")
    plt.show()


    end_time = time.time()
    print(f"Total time taken: {np.round((end_time - start_time)/60,1)} minutes")
```

3.4 Forth problem

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 10 11:29:10 2025

@author: fredrik
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import csr_matrix
import time


Nl = 81                                 # Lattice size along one dimension
```

```
Ns = Nl**2
Nh = (Nl+1) // 2
Nhp = (Nl-1) // 2
energy_range = np.linspace(-8, 8, 1000)  # Energy range
V = -1                                    # Hopping parameter
epsilon = 0                               # On-site energy
gamma = 0.05                              # Broadening factor


def coord_to_index(i, j, Nhp=Nhp, Nl=Nl):
    """ Convert (i', j') coordinates to matrix index m. """
    ibar = i + Nhp
    jbar = j + Nhp
    return Nl*ibar + jbar

def get_neighbors(i, j):
    "This function check if there are any neighbors nearby that exsists"
    # List all possible nearest neighbors (up, down, left, right)
    neighbors = [(i-1,j), (i+1,j), (i,j-1), (i,j+1)]

    valid_neighbors = []

    # Check each potential neighbor to ensure it is within the valid range
    for ni, nj in neighbors:
        if -Nhp <= ni <= Nhp and -Nhp <= nj <= Nhp:
            valid_neighbors.append((ni, nj))

    return valid_neighbors # Return the list of valid nearest neighbors


def hamiltonian(Ns, epsilon, V):
    "Construct the Hamiltonian matrix for a 2D square lattice."
    H = csr_matrix((Ns, Ns), dtype=complex).tolil()
    for i in range(-Nhp, Nhp + 1):
        for j in range(-Nhp, Nhp + 1):
            m = coord_to_index(i, j)
            H[m, m] = epsilon  # On-site energy
            for ni, nj in get_neighbors(i, j):
                n = coord_to_index(ni, nj)
                H[m, n] = V  # Nearest-neighbor hopping
    return H.tocsr()


def hamiltonian_adsorbate(Ns, epsilon, V, adsorbate_type, epsilon_0, V_0, Nhp=Nhp, Nl=Nl):
    """ Construct the Hamiltonian matrix including adsorbate effects """
    H = csr_matrix((Ns+1, Ns+1)).tolil()

    # Fill the original surface Hamiltonian
    for i in range(-Nhp, Nhp + 1):
        for j in range(-Nhp, Nhp + 1):
            m = coord_to_index(i, j, Nhp, Nl)
            H[m, m] = epsilon  # On-site energy
            for ni, nj in get_neighbors(i, j):
                n = coord_to_index(ni, nj, Nhp, Nl)
                H[m, n] = V  # Nearest-neighbor hopping

    # Define adsorbate interaction
    adsorbate = Ns  # The additional adsorbate index
    H[adsorbate, adsorbate] = epsilon_0  # On-site energy of adsorbate

    if adsorbate_type == "atop":
        m = coord_to_index(0, 0)
        H[m, adsorbate] = V_0
        H[adsorbate, m] = V_0

    elif adsorbate_type == "bridge":
        m1 = coord_to_index(0, 0)
        m2 = coord_to_index(1, 0)
        H[m1, adsorbate] = V_0 / np.sqrt(2)
        H[m2, adsorbate] = V_0 / np.sqrt(2)
        H[adsorbate, m1] = V_0 / np.sqrt(2)
        H[adsorbate, m2] = V_0 / np.sqrt(2)

    elif adsorbate_type == "center":
        sites = [(0, 0), (1, 0), (0, 1), (1, 1)]
        for i, j in sites:
            m = coord_to_index(i, j, Nhp, Nl)
            H[m, adsorbate] = V_0 / 2
            H[adsorbate, m] = V_0 / 2

    elif adsorbate_type == "impurity":
        m = coord_to_index(0, 0, Nhp, Nl)
        H[m, m] = epsilon_0  # Replace surface atom energy
        for ni, nj in get_neighbors(0, 0):
            n = coord_to_index(ni, nj, Nhp, Nl)
            H[m, n] = V_0
            H[n, m] = V_0

    return H.tocsr()

def sums(gamma, eigenvecs, lamb, energy, eigvals, site):
    "Compute each term in the LDOS sum."
    return (gamma / np.pi) * (np.abs(eigenvecs[site, lamb]) ** 2) / ((energy - eigvals[lamb])**2


def compute_LDOS(H, energy_range, Ns=Ns, center=False):
    "Compute the Local Density of States (LDOS)."

    # convert to dense array
    H = H.toarray()

    # Find eigenenergies and eigenvectors
    eigenenergy, eigenvectors = np.linalg.eigh(H)

    # Initialize LDOS as a dictionary
    LDOS = np.zeros(len(energy_range))

    #Check position of paticle
    if center == True:
        site_index = coord_to_index(0, 0)
    else:
        site_index = Ns

    for i, E in enumerate(energy_range):
            for lamb in range(Ns):
                LDOS[i] += sums(gamma=gamma, eigenvecs=eigenvectors,
                                    lamb=lamb, energy=E,
                                    eigvals=eigenenergy, site=site_index)
    return LDOS

#%%

start_time = time.time()


# Clean surface LDOS
clean = compute_LDOS(hamiltonian(Ns, epsilon, V),
                     energy_range, center=True)

# Adsorbate cases 1
atop1 = compute_LDOS(hamiltonian_adsorbate(Ns, epsilon, V, "atop", -2, -1.3),
                     energy_range)

bridge1 = compute_LDOS(hamiltonian_adsorbate(Ns, epsilon, V, "bridge", -2, -1.3),
                     energy_range)

center1 = compute_LDOS(hamiltonian_adsorbate(Ns, epsilon, V, "center", -2, -1.3),
                     energy_range)

impurity1 =  compute_LDOS(hamiltonian_adsorbate(Ns, epsilon, V, "impurity", -2, -1.3),
                     energy_range, center=True)

# Adsorbate cases 2
atop2 = compute_LDOS(hamiltonian_adsorbate(Ns, epsilon, V, "atop", -2, -5),
                     energy_range)

bridge2 = compute_LDOS(hamiltonian_adsorbate(Ns, epsilon, V, "bridge", -2, -5),
                     energy_range)

center2 = compute_LDOS(hamiltonian_adsorbate(Ns, epsilon, V, "center", -2, -5),
                     energy_range)

impurity2 =  compute_LDOS(hamiltonian_adsorbate(Ns, epsilon, V, "impurity", -2, -5),
                     energy_range, center=True)


end_time = time.time()
print(f"Total time taken: {np.round((end_time - start_time)/60,1)} minutes")



#%%

fig, axes = plt.subplots(2, 1, figsize=(8, 8))
fig.suptitle("Local Density of States With Adsorbates for Site (0,0)")

# For the first case
axes[0].set_title(r"$\epsilon_0=-1, V_0=-1.3$")

axes[0].plot(energy_range, clean, label="clean", c='black', linestyle=':')
axes[0].plot(energy_range, atop1, label="atop")
axes[0].plot(energy_range, bridge1, label="bridge")
axes[0].plot(energy_range, center1, label="center")
axes[0].plot(energy_range, impurity1, label="impurity")
```

```
axes[0].set_ylabel(r"$g^L_{i}(E, \Gamma)$")
axes[0].legend()
axes[0].grid()
#axes[0].set_ylim(0,0.4)

# For the second case
axes[1].set_title(r"$\epsilon_0=-1, V_0=-5$")

axes[1].plot(energy_range, clean, label="clean", c='black', linestyle=':')
axes[1].plot(energy_range, atop2, label="atop")
axes[1].plot(energy_range, bridge2, label="bridge")
axes[1].plot(energy_range, center2, label="center")
axes[1].plot(energy_range, impurity2, label="impurity")

axes[1].set_ylabel(r"$g^L_{i}(E, \Gamma)$")
axes[1].legend()
axes[1].grid()
#axes[1].set_ylim(0,0.4)


# Set common x-axis
axes[1].set_xlabel("Energy E")
plt.tight_layout()

plt.savefig("Comp_Proj1/Figures/task4.pdf")
plt.show()
```