Annexure A
Nano-Language Scripting

This section will include all the Scripts used during the theoretical study of DDQ based molecular device characterisation. Entire scripting section is divided in three sections: parameter optimisation, analysis, and characterization.

A.1 Parameter optimization python script

This section will cover python scripts related to parameter optimization like mesh cut-off energy, k-point sampling and geometry optimization.

A.1.1 Python script of a molecular device configuration

```
# -------------------------------------------------------------
# TwoProbe configuration
# -------------------------------------------------------------

# -------------------------------------------------------------
# Left electrode
# -------------------------------------------------------------

# Set up lattice
vector_a = [8.65127, 0.0, 0.0]*Angstrom
vector_b = [-4.32564, 7.49222, 0.0]*Angstrom
vector_c = [0.0, 0.0, 7.06373620597]*Angstrom
left_electrode_lattice = UnitCell(vector_a, vector_b, vector_c)

# Define elements
left_electrode_elements = [Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold,
                           Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold,
                           Gold, Gold, Gold, Gold, Gold, Gold, Gold]

# Define coordinates
left_electrode_coordinates = [[ 1.44187912,  0.8324693 ,  1.17722557],
                              [ 4.32563735,  0.8324693 ,  1.17722557],
                              [ 7.20939558,  0.8324693 ,  1.17722557],
                              [-0.        ,  3.32987718,  1.17722557],
                              [ 2.88375823,  3.32987718,  1.17722557],
                              [ 5.76751646,  3.32987718,  1.17722557],
                              [-1.44187912,  5.82728507,  1.17722557],
                              [ 1.44187912,  5.82728507,  1.17722557],
                              [ 4.32563735,  5.82728507,  1.17722557],
                              [-0.        ,  1.66493859,  3.53180431],
                              [ 2.88375823,  1.66493859,  3.53180431],
                              [ 5.76751646,  1.66493859,  3.53180431],
                              [-1.44187912,  4.16234648,  3.53180431],
                  [ 1.44187912,  4.16234648,  3.53180431],
                  [ 4.32563735,  4.16234648,  3.53180431],
                  [-2.88375823,  6.65975436,  3.53180431],
                  [-0.        ,  6.65975436,  3.53180431],
                  [ 2.88375823,  6.65975436,  3.53180431],
                  [-0.        ,  0.        ,  5.88638304],
                  [ 2.88375823,  0.        ,  5.88638304],
                  [ 5.76751646,  0.        ,  5.88638304],
                  [-1.44187912,  2.49740789,  5.88638304],
                  [ 1.44187912,  2.49740789,  5.88638304],
                  [ 4.32563735,  2.49740789,  5.88638304],
                  [-2.88375823,  4.99481577,  5.88638304],
                  [-0.        ,  4.99481577,  5.88638304],
                  [ 2.88375823,  4.99481577,  5.88638304]]*Angstrom

# Set up configuration
left_electrode = BulkConfiguration(
   bravais_lattice=left_electrode_lattice,
   elements=left_electrode_elements,
   cartesian_coordinates=left_electrode_coordinates
   )

# -------------------------------------------------------------
# Right electrode
# -------------------------------------------------------------

# Set up lattice
vector_a = [8.65127, 0.0, 0.0]*Angstrom
vector_b = [-4.32564, 7.49222, 0.0]*Angstrom
vector_c = [0.0, 0.0, 7.06373620597]*Angstrom
right_electrode_lattice = UnitCell(vector_a, vector_b, vector_c)

# Define elements
right_electrode_elements = [Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold,
                  Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold,
                  Gold, Gold, Gold, Gold, Gold, Gold, Gold]

# Define coordinates
right_electrode_coordinates = [[ 0.00000177,  0.00000282,  1.17735316],
                  [ 2.88376   ,  0.00000282,  1.17735316],
                  [ 5.76751823,  0.00000282,  1.17735316],
                  [-1.44187735,  2.4974107 ,  1.17735316],
                  [ 1.44188088,  2.4974107 ,  1.17735316],
                  [ 4.32563912,  2.4974107 ,  1.17735316],
                  [-2.88375646,  4.99481859,  1.17735316],
                  [ 0.00000177,  4.99481859,  1.17735316],
                  [ 2.88376   ,  4.99481859,  1.17735316],
                  [ 0.00000177,  1.66494141,  3.5319319 ],
                  [ 2.88376   ,  1.66494141,  3.5319319 ],
                  [ 5.76751823,  1.66494141,  3.5319319 ],
                  [-1.44187735,  4.1623493 ,  3.5319319 ],
                  [ 1.44188088,  4.1623493 ,  3.5319319 ],
                  [ 4.32563912,  4.1623493 ,  3.5319319 ],
                  [-2.88375646,  6.65975718,  3.5319319 ],
                  [ 0.00000177,  6.65975718,  3.5319319 ],
                  [ 2.88376   ,  6.65975718,  3.5319319 ],
                  [ 1.44188088,  0.83247211,  5.88651063],
                    [ 4.32563912,  0.83247211,  5.88651063],
                    [ 7.20939735,  0.83247211,  5.88651063],
                    [ 0.00000177,  3.32988   ,  5.88651063],
                    [ 2.88376   ,  3.32988   ,  5.88651063],
                    [ 5.76751823,  3.32988   ,  5.88651063],
                    [-1.44187735,  5.82728789,  5.88651063],
                    [ 1.44188088,  5.82728789,  5.88651063],
                    [ 4.32563912,  5.82728789,  5.88651063]]*Angstrom

# Set up configuration
right_electrode = BulkConfiguration(
    bravais_lattice=right_electrode_lattice,
    elements=right_electrode_elements,
    cartesian_coordinates=right_electrode_coordinates
    )

# -------------------------------------------------------------
# Central region
# -------------------------------------------------------------

# Set up lattice
vector_a = [8.65127, 0.0, 0.0]*Angstrom
vector_b = [-4.32564, 7.49222, 0.0]*Angstrom
vector_c = [0.0, 0.0, 25.0862936766]*Angstrom
central_region_lattice = UnitCell(vector_a, vector_b, vector_c)

# Define elements
central_region_elements = [Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold,
                  Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold,
                  Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Oxygen,
                  Nitrogen, Chlorine, Carbon, Carbon, Carbon, Carbon, Carbon,
                  Carbon, Carbon, Carbon, Chlorine, Nitrogen, Oxygen, Gold, Gold,
                  Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold,
                  Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold, Gold,
                  Gold, Gold, Gold, Gold, Gold, Gold]

# Define coordinates
central_region_coordinates = [[  1.44187912,   0.8324693 ,   1.17722557],
                   [  4.32563735,   0.8324693 ,   1.17722557],
                   [  7.20939558,   0.8324693 ,   1.17722557],
                   [ -0.        ,   3.32987718,   1.17722557],
                   [  2.88375823,   3.32987718,   1.17722557],
                   [  5.76751646,   3.32987718,   1.17722557],
                   [ -1.44187912,   5.82728507,   1.17722557],
                   [  1.44187912,   5.82728507,   1.17722557],
                   [  4.32563735,   5.82728507,   1.17722557],
                   [ -0.        ,   1.66493859,   3.53180431],
                   [  2.88375823,   1.66493859,   3.53180431],
                   [  5.76751646,   1.66493859,   3.53180431],
                   [ -1.44187912,   4.16234648,   3.53180431],
                   [  1.44187912,   4.16234648,   3.53180431],
                   [  4.32563735,   4.16234648,   3.53180431],
                   [ -2.88375823,   6.65975436,   3.53180431],
                   [ -0.        ,   6.65975436,   3.53180431],
                   [  2.88375823,   6.65975436,   3.53180431],
                   [ -0.        ,   0.        ,   5.88638304],
                   [  2.88375823,   0.        ,   5.88638304],
                  [ 5.76751646,  0.        ,  5.88638304],
                  [-1.44187912,  2.49740789,  5.88638304],
                  [ 1.44187912,  2.49740789,  5.88638304],
                  [ 4.32563735,  2.49740789,  5.88638304],
                  [-2.88375823,  4.99481577,  5.88638304],
                  [-0.        ,  4.99481577,  5.88638304],
                  [ 2.88375823,  4.99481577,  5.88638304],
                  [ 2.88375823,  3.32987718,  7.59638304],
                  [ 2.88376   ,  3.32988   ,  9.87331063],
                  [ 4.2697899 ,  6.50057015, 10.51760257],
                  [ 1.81014503,  0.87261165, 10.9662409 ],
                  [ 2.88065486,  3.32120741, 11.08421748],
                  [ 3.86806601,  5.580336  , 11.11210805],
                  [ 3.3879012 ,  4.48135282, 11.86026397],
                  [ 2.37863679,  2.17192217, 11.86260049],
                  [ 2.37881148,  2.17223723, 13.22375989],
                  [ 3.38809211,  4.48150716, 13.22545822],
                  [ 3.86838518,  5.58043205, 13.97323259],
                  [ 2.88091287,  3.32154093, 14.00128499],
                  [ 1.81091185,  0.87404624, 14.12226832],
                  [ 4.27011418,  6.50047904, 14.56821635],
                  [ 2.88376   ,  3.32988   , 15.21296224],
                  [ 2.88376   ,  3.32988   , 17.48991063],
                  [ 0.00000177,  0.00000282, 19.19991063],
                  [ 2.88376   ,  0.00000282, 19.19991063],
                  [ 5.76751823,  0.00000282, 19.19991063],
                  [-1.44187735,  2.4974107 , 19.19991063],
                  [ 1.44188088,  2.4974107 , 19.19991063],
                  [ 4.32563912,  2.4974107 , 19.19991063],
                  [-2.88375646,  4.99481859, 19.19991063],
                  [ 0.00000177,  4.99481859, 19.19991063],
                  [ 2.88376   ,  4.99481859, 19.19991063],
                  [ 0.00000177,  1.66494141, 21.55448937],
                  [ 2.88376   ,  1.66494141, 21.55448937],
                  [ 5.76751823,  1.66494141, 21.55448937],
                  [-1.44187735,  4.1623493 , 21.55448937],
                  [ 1.44188088,  4.1623493 , 21.55448937],
                  [ 4.32563912,  4.1623493 , 21.55448937],
                  [-2.88375646,  6.65975718, 21.55448937],
                  [ 0.00000177,  6.65975718, 21.55448937],
                  [ 2.88376   ,  6.65975718, 21.55448937],
                  [ 1.44188088,  0.83247211, 23.9090681 ],
                  [ 4.32563912,  0.83247211, 23.9090681 ],
                  [ 7.20939735,  0.83247211, 23.9090681 ],
                  [ 0.00000177,  3.32988   , 23.9090681 ],
                  [ 2.88376   ,  3.32988   , 23.9090681 ],
                  [ 5.76751823,  3.32988   , 23.9090681 ],
                  [-1.44187735,  5.82728789, 23.9090681 ],
                  [ 1.44188088,  5.82728789, 23.9090681 ],
                  [ 4.32563912,  5.82728789, 23.9090681 ]]*Angstrom

# Set up configuration
central_region = BulkConfiguration(
    bravais_lattice=central_region_lattice,
    elements=central_region_elements,
    cartesian_coordinates=central_region_coordinates
    )
```

```
device_configuration = DeviceConfiguration(
   central_region,
   [left_electrode, right_electrode]
   )
```

A.1.2 Python script for geometry optimization of molecular device

```
# -------------------------------------------------------------
# Calculator
# -------------------------------------------------------------
#----------------------------------------
# Basis Set
#----------------------------------------
basis_set = [
   LDABasis.Carbon_DoubleZetaPolarized,
   LDABasis.Nitrogen_DoubleZetaPolarized,
   LDABasis.Oxygen_DoubleZetaPolarized,
   LDABasis.Chlorine_DoubleZetaPolarized,
   LDABasis.Gold_SingleZetaPolarized,
   ]

#----------------------------------------
# Poisson Solver Settings
#----------------------------------------
left_electrode_poisson_solver = FastFourier2DSolver(
   boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                 [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                 [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
   )

right_electrode_poisson_solver = FastFourier2DSolver(
   boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                 [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                 [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
   )

#----------------------------------------
# Electrode Calculators
#----------------------------------------
left_electrode_calculator = LCAOCalculator(
   basis_set=basis_set,
   poisson_solver=left_electrode_poisson_solver,
   )

right_electrode_calculator = LCAOCalculator(
   basis_set=basis_set,
   poisson_solver=right_electrode_poisson_solver,
   )

#----------------------------------------
# Device Calculator
#----------------------------------------
calculator = DeviceLCAOCalculator(
   basis_set=basis_set,
   electrode_calculators=
      [left_electrode_calculator, right_electrode_calculator],
  )

device_configuration.setCalculator(calculator)
nlprint(device_configuration)
device_configuration.update()
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_test1_traj.nc',
device_configuration)

device_configuration = OptimizeGeometry(
    device_configuration,
    max_forces=0.005*eV/Ang,
    max_steps=200,
    max_step_length=0.2*Ang,

trajectory_filename='/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_test1_traj.nc',
    disable_stress=True,
    optimizer_method=QuasiNewton(),
    )
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_test1_traj.nc',
device_configuration)
nlprint(device_configuration)
```

A.1.3 Python script for the calculation of mesh cut-off energy for the atomic configuration

```
# -------------------------------------------------------------
# Calculator
# -------------------------------------------------------------
#----------------------------------------
# Basis Set
#----------------------------------------
basis_set = DFTBDirectory("cp2k/scc/")

#----------------------------------------
# Pair Potentials
#----------------------------------------
pair_potentials = DFTBDirectory("cp2k/scc/")

# List of values for the grid mesh cutoff
cutoffs = [2.5, 5, 10, 20, 30, 40, 60, 80]

# List to hold the chemical potential for each calculation.
chemical_potentials = []

# Loop over the grid mesh cutoffs
for cutoff in cutoffs:

  numerical_accuracy_parameters = NumericalAccuracyParameters(
    k_point_sampling=(7, 7, 51),
    density_mesh_cutoff=cutoff*Hartree
    )

  iteration_control_parameters = IterationControlParameters()

  calculator = SlaterKosterCalculator(
    basis_set=basis_set,
    pair_potentials=pair_potentials,
    numerical_accuracy_parameters=numerical_accuracy_parameters,
        iteration_control_parameters=iteration_control_parameters,
        )

    bulk_configuration.setCalculator(calculator)
    bulk_configuration.update()

    # ----------------------------------------------------------
    # Chemical Potential
    # ----------------------------------------------------------
    chemical_potential = ChemicalPotential(bulk_configuration)
    value = chemical_potential.evaluate().inUnitsOf(eV)
    chemical_potentials.append(value)

# Plot data.
import pylab

pylab.plot(cutoffs, chemical_potentials, 'ro-')
pylab.xlabel('Grid mesh cut-off (Ha)')
pylab.ylabel('Fermi level (eV)')

# Show the plot.
pylab.show()
```

A.1.4 Python script to determine k-point sampling values at different energy values

```
# Restart from the previous calculation
device_configuration =
nlread('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_ZBTS.nc',
object_id='gID000')[0]

# Array with k-values
numbers_of_kpoints = range(2,40)

# Energies at which to calculate the transmission function
Energies = [-3,0,3]

# Array with transmission values
T = numpy.zeros([len(Energies),len(numbers_of_kpoints)])

# Loop over numbers of k-points.
for i, num_k in enumerate(numbers_of_kpoints):
    # ----------------------------------------------------------
    # Transmission spectrum
    # ----------------------------------------------------------
    transmission_spectrum = TransmissionSpectrum(
        configuration=device_configuration,
        energies=numpy.array(Energies)*eV,
        kpoints=MonkhorstPackGrid(num_k,num_k),
        energy_zero_parameter=AverageFermiLevel,
        infinitesimal=1e-06*eV,
        self_energy_calculator=KrylovSelfEnergy(),
        )

    nlprint(transmission_spectrum)
    # Put the k-point averaged transmission into the array
    T[:,i] = transmission_spectrum.evaluate()
```

```
# prepare for plotting
import pylab as pl

# convert to numpy array
numbers_of_kpoints  = numpy.array(numbers_of_kpoints)

# plot the results
pl.subplot(311)
pl.plot(numbers_of_kpoints , T[0,:] ,'ro-',label='E=-3 eV')
pl.xlabel('# k-points',fontsize=16)
pl.ylabel('Transmission',fontsize=16)
pl.legend()

pl.subplot(312)
pl.plot(numbers_of_kpoints , T[1,:] ,'ro-',label='E=0 eV')
pl.xlabel('# k-points',fontsize=16)
pl.ylabel('Transmission',fontsize=16)
pl.legend()

pl.subplot(313)
pl.plot(numbers_of_kpoints , T[2,:] ,'ro-',label='E=3 eV')
pl.xlabel('# k-points',fontsize=16)
pl.ylabel('Transmission',fontsize=16)
pl.legend()
pl.show()
```

A.2 Molecule/Device Analysis Python Script

This section cover python scripts of some molecule/ device-based analysis.

A.2.1 Python script for electrostatic difference potential, total energy and molecular energy spectrum analysis.

```
# -------------------------------------------------------------
# Calculator
# -------------------------------------------------------------
calculator = LCAOCalculator()

molecule_configuration.setCalculator(calculator)
nlprint(molecule_configuration)
molecule_configuration.update()
nlsave('/home/giri/.vnl/example_project_13.8.1/ddq_OG_test5_MES_test1.nc',
molecule_configuration)

# -------------------------------------------------------------
# Electrostatic difference potential
# -------------------------------------------------------------
electrostatic_difference_potential = ElectrostaticDifferencePotential(molecule_configuration)
nlsave('/home/giri/.vnl/example_project_13.8.1/ddq_OG_test5_MES_test1.nc',
electrostatic_difference_potential)

# -------------------------------------------------------------
# Total energy
# -------------------------------------------------------------
total_energy = TotalEnergy(molecule_configuration)
nlsave('/home/giri/.vnl/example_project_13.8.1/ddq_OG_test5_MES_test1.nc', total_energy)
nlprint(total_energy)
```

```
# -------------------------------------------------------------
# Molecular energy spectrum
# -------------------------------------------------------------
molecular_energy_spectrum = MolecularEnergySpectrum(
   configuration=molecule_configuration,
   energy_zero_parameter=AbsoluteEnergy,
   projection_list=ProjectionList(All)
   )
nlsave('/home/giri/.vnl/example_project_13.8.1/ddq_OG_test5_MES_test1.nc',
molecular_energy_spectrum)
nlprint(molecular_energy_spectrum)
```

A.2.2 Python script for eigen state analysis near fermi energy level as calculated from the log file of molecular energy spectrum analysis.

```
# -------------------------------------------------------------
# Calculator
# -------------------------------------------------------------
calculator = LCAOCalculator()

molecule_configuration.setCalculator(calculator)
nlprint(molecule_configuration)
molecule_configuration.update()
nlsave('/home/giri/.vnl/example_project_13.8.1/ddq_OG_test5_ES_test32to36.nc',
molecule_configuration)

# -------------------------------------------------------------
# Eigenstate
# -------------------------------------------------------------
eigenstate = Eigenstate(
   configuration=molecule_configuration,
   projection_list=ProjectionList(elements=[Carbon, Chlorine, Nitrogen, Oxygen]),
   quantum_number=32,
   )
nlsave('/home/giri/.vnl/example_project_13.8.1/ddq_OG_test5_ES_test32to36.nc', eigenstate)

# -------------------------------------------------------------
# Eigenstate
# -------------------------------------------------------------
eigenstate = Eigenstate(
   configuration=molecule_configuration,
   projection_list=ProjectionList(elements=[Carbon, Chlorine, Nitrogen, Oxygen]),
   quantum_number=33,
   )
nlsave('/home/giri/.vnl/example_project_13.8.1/ddq_OG_test5_ES_test32to36.nc', eigenstate)

# -------------------------------------------------------------
# Eigenstate
# -------------------------------------------------------------
eigenstate = Eigenstate(
   configuration=molecule_configuration,
   projection_list=ProjectionList(elements=[Carbon, Chlorine, Nitrogen, Oxygen]),
   quantum_number=34,
   )
nlsave('/home/giri/.vnl/example_project_13.8.1/ddq_OG_test5_ES_test32to36.nc', eigenstate)
```

```
# -------------------------------------------------------------
# Eigenstate
# -------------------------------------------------------------
eigenstate = Eigenstate(
   configuration=molecule_configuration,
   projection_list=ProjectionList(elements=[Carbon, Chlorine, Nitrogen, Oxygen]),
   quantum_number=35,
   )
nlsave('/home/giri/.vnl/example_project_13.8.1/ddq_OG_test5_ES_test32to36.nc', eigenstate)

# -------------------------------------------------------------
# Eigenstate
# -------------------------------------------------------------
eigenstate = Eigenstate(
   configuration=molecule_configuration,
   projection_list=ProjectionList(elements=[Carbon, Chlorine, Nitrogen, Oxygen]),
   quantum_number=36,
   )
nlsave('/home/giri/.vnl/example_project_13.8.1/ddq_OG_test5_ES_test32to36.nc', eigenstate)
```

A.2.3 Python script for zero biased transmission spectrum analysis of molecular device configuration.

```
# -------------------------------------------------------------
# Calculator
# -------------------------------------------------------------
#----------------------------------------
# Basis Set
#----------------------------------------
basis_set = [
   LDABasis.Carbon_DoubleZetaPolarized,
   LDABasis.Nitrogen_DoubleZetaPolarized,
   LDABasis.Oxygen_DoubleZetaPolarized,
   LDABasis.Chlorine_DoubleZetaPolarized,
   LDABasis.Gold_SingleZetaPolarized,
   ]

#----------------------------------------
# Numerical Accuracy Settings
#----------------------------------------
left_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(10, 10, 100),
   )

right_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(10, 10, 100),
   )

device_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(10, 10, 100),
   )

#----------------------------------------
# Poisson Solver Settings
#----------------------------------------
```

```
left_electrode_poisson_solver = FastFourier2DSolver(
  boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
               [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
               [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
  )

right_electrode_poisson_solver = FastFourier2DSolver(
  boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
               [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
               [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
  )

#----------------------------------------
# Electrode Calculators
#----------------------------------------
left_electrode_calculator = LCAOCalculator(
  basis_set=basis_set,
  numerical_accuracy_parameters=left_electrode_numerical_accuracy_parameters,
  poisson_solver=left_electrode_poisson_solver,
  )

right_electrode_calculator = LCAOCalculator(
  basis_set=basis_set,
  numerical_accuracy_parameters=right_electrode_numerical_accuracy_parameters,
  poisson_solver=right_electrode_poisson_solver,
  )

#----------------------------------------
# Device Calculator
#----------------------------------------
calculator = DeviceLCAOCalculator(
  basis_set=basis_set,
  numerical_accuracy_parameters=device_numerical_accuracy_parameters,
  electrode_calculators=
    [left_electrode_calculator, right_electrode_calculator],
  )

device_configuration.setCalculator(calculator)
nlprint(device_configuration)
device_configuration.update()
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_ZBTS.nc',
device_configuration)

# -------------------------------------------------------------
# Transmission spectrum
# -------------------------------------------------------------
transmission_spectrum = TransmissionSpectrum(
  configuration=device_configuration,
  energies=numpy.linspace(-10,10,250)*eV,
  kpoints=MonkhorstPackGrid(7,7),
  energy_zero_parameter=AverageFermiLevel,
  infinitesimal=1e-06*eV,
  self_energy_calculator=RecursionSelfEnergy(),
  )
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_ZBTS.nc',
transmission_spectrum)
nlprint(transmission_spectrum)
```

A.2.4 Python script for calculation of device density of states.

```
# -------------------------------------------------------------
# Calculator
# -------------------------------------------------------------
#----------------------------------------
# Basis Set
#----------------------------------------
basis_set = [
   LDABasis.Carbon_DoubleZetaPolarized,
   LDABasis.Nitrogen_DoubleZetaPolarized,
   LDABasis.Oxygen_DoubleZetaPolarized,
   LDABasis.Chlorine_DoubleZetaPolarized,
   LDABasis.Gold_SingleZetaPolarized,
   ]

#----------------------------------------
# Numerical Accuracy Settings
#----------------------------------------
left_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(3, 3, 100),
   )

right_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(3, 3, 100),
   )

device_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(3, 3, 100),
   )

#----------------------------------------
# Poisson Solver Settings
#----------------------------------------
left_electrode_poisson_solver = FastFourier2DSolver(
   boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                  [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                  [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
   )

right_electrode_poisson_solver = FastFourier2DSolver(
   boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                  [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                  [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
   )

#----------------------------------------
# Electrode Calculators
#----------------------------------------
left_electrode_calculator = LCAOCalculator(
   basis_set=basis_set,
   numerical_accuracy_parameters=left_electrode_numerical_accuracy_parameters,
   poisson_solver=left_electrode_poisson_solver,
   )

right_electrode_calculator = LCAOCalculator(
   basis_set=basis_set,
   numerical_accuracy_parameters=right_electrode_numerical_accuracy_parameters,
   poisson_solver=right_electrode_poisson_solver,
   )

#----------------------------------------
# Device Calculator
#----------------------------------------
calculator = DeviceLCAOCalculator(
   basis_set=basis_set,
   numerical_accuracy_parameters=device_numerical_accuracy_parameters,
   electrode_calculators=
      [left_electrode_calculator, right_electrode_calculator],
   )

device_configuration.setCalculator(calculator)
nlprint(device_configuration)
device_configuration.update()
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_DDOS.nc',
device_configuration)

# -------------------------------------------------------------
# Device density of states
# -------------------------------------------------------------
device_density_of_states = DeviceDensityOfStates(
   configuration=device_configuration,
   energies=numpy.linspace(-10,10,500)*eV,
   kpoints=MonkhorstPackGrid(3,3),
   contributions=All,
   energy_zero_parameter=AverageFermiLevel,
   infinitesimal=1e-06*eV,
   self_energy_calculator=RecursionSelfEnergy(),
   )
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_DDOS.nc',
device_density_of_states)
nlprint(device_density_of_states)
```

A.2.5 Python script for calculation molecular projected self-consistent Hamiltonian (MPSH).

```
# -------------------------------------------------------------
# Calculator
# -------------------------------------------------------------
#----------------------------------------
# Basis Set
#----------------------------------------
basis_set = [
   LDABasis.Carbon_DoubleZetaPolarized,
   LDABasis.Nitrogen_DoubleZetaPolarized,
   LDABasis.Oxygen_DoubleZetaPolarized,
   LDABasis.Chlorine_DoubleZetaPolarized,
   LDABasis.Gold_SingleZetaPolarized,
   ]

#----------------------------------------
# Numerical Accuracy Settings
#----------------------------------------
left_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(3, 3, 100),
   )

right_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(3, 3, 100),
   )

device_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(3, 3, 100),
   )

#----------------------------------------
# Poisson Solver Settings
#----------------------------------------
left_electrode_poisson_solver = FastFourier2DSolver(
   boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
   )

right_electrode_poisson_solver = FastFourier2DSolver(
   boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
   )

#----------------------------------------
# Electrode Calculators
#----------------------------------------
left_electrode_calculator = LCAOCalculator(
   basis_set=basis_set,
   numerical_accuracy_parameters=left_electrode_numerical_accuracy_parameters,
   poisson_solver=left_electrode_poisson_solver,
   )

right_electrode_calculator = LCAOCalculator(
   basis_set=basis_set,
   numerical_accuracy_parameters=right_electrode_numerical_accuracy_parameters,
   poisson_solver=right_electrode_poisson_solver,
   )

#----------------------------------------
# Device Calculator
#----------------------------------------
calculator = DeviceLCAOCalculator(
   basis_set=basis_set,
   numerical_accuracy_parameters=device_numerical_accuracy_parameters,
   electrode_calculators=
      [left_electrode_calculator, right_electrode_calculator],
   )

device_configuration.setCalculator(calculator)
nlprint(device_configuration)
device_configuration.update()
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_MPSH1.nc',
device_configuration)
```

```
# -------------------------------------------------------------
# Molecular energy spectrum
# -------------------------------------------------------------
molecular_energy_spectrum = MolecularEnergySpectrum(
   configuration=device_configuration,
   energy_zero_parameter=FermiLevel,
   projection_list=ProjectionList(atoms=[28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41],
elements=[Carbon, Chlorine, Nitrogen, Oxygen])
   )
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_MPSH1.nc',
molecular_energy_spectrum)
nlprint(molecular_energy_spectrum)
```

A.2.6 Python script for the calculation of transmission eigen values near fermi energy level as obtained from the analysis results of MPSH and their corresponding transmission eigen states.

```
# -------------------------------------------------------------
# Calculator
# -------------------------------------------------------------
#----------------------------------------
# Basis Set
#----------------------------------------
basis_set = [
   LDABasis.Carbon_DoubleZetaPolarized,
   LDABasis.Nitrogen_DoubleZetaPolarized,
   LDABasis.Oxygen_DoubleZetaPolarized,
   LDABasis.Chlorine_DoubleZetaPolarized,
   LDABasis.Gold_SingleZetaPolarized,
   ]

#----------------------------------------
# Numerical Accuracy Settings
#----------------------------------------
left_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(3, 3, 100),
   )

right_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(3, 3, 100),
   )

device_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(3, 3, 100),
   )

#----------------------------------------
# Poisson Solver Settings
#----------------------------------------
left_electrode_poisson_solver = FastFourier2DSolver(
   boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
   )

right_electrode_poisson_solver = FastFourier2DSolver(
    boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                  [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                  [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
    )

#----------------------------------------
# Electrode Calculators
#----------------------------------------
left_electrode_calculator = LCAOCalculator(
    basis_set=basis_set,
    numerical_accuracy_parameters=left_electrode_numerical_accuracy_parameters,
    poisson_solver=left_electrode_poisson_solver,
    )

right_electrode_calculator = LCAOCalculator(
    basis_set=basis_set,
    numerical_accuracy_parameters=right_electrode_numerical_accuracy_parameters,
    poisson_solver=right_electrode_poisson_solver,
    )

#----------------------------------------
# Device Calculator
#----------------------------------------
calculator = DeviceLCAOCalculator(
    basis_set=basis_set,
    numerical_accuracy_parameters=device_numerical_accuracy_parameters,
    electrode_calculators=
        [left_electrode_calculator, right_electrode_calculator],
    )

device_configuration.setCalculator(calculator)
nlprint(device_configuration)
device_configuration.update()
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_TEV.nc',
device_configuration)

# -------------------------------------------------------------
# Transmission eigenvalues
# -------------------------------------------------------------
transmission_eigenvalues = TransmissionEigenvalues(
    configuration=device_configuration,
    energy=9.83968*eV,
    k_point=[0, 0],
    energy_zero_parameter=AverageFermiLevel,
    )
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_TEV.nc',
transmission_eigenvalues)
nlprint(transmission_eigenvalues)

# -------------------------------------------------------------
# Transmission eigenvalues
# -------------------------------------------------------------
transmission_eigenvalues = TransmissionEigenvalues(
    configuration=device_configuration,
    energy=-1.38277*eV,
    k_point=[0, 0],
    energy_zero_parameter=AverageFermiLevel,
   )
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_TEV.nc',
transmission_eigenvalues)
nlprint(transmission_eigenvalues)

# -------------------------------------------------------------
# Transmission eigenvalues
# -------------------------------------------------------------
transmission_eigenvalues = TransmissionEigenvalues(
   configuration=device_configuration,
   energy=-2.50501*eV,
   k_point=[0, 0],
   energy_zero_parameter=AverageFermiLevel,
   )
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_TEV.nc',
transmission_eigenvalues)
nlprint(transmission_eigenvalues)

# -------------------------------------------------------------
# Transmission eigenvalues
# -------------------------------------------------------------
transmission_eigenvalues = TransmissionEigenvalues(
   configuration=device_configuration,
   energy=-2.86573*eV,
   k_point=[0, 0],
   energy_zero_parameter=AverageFermiLevel,
   )
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_TEV.nc',
transmission_eigenvalues)
nlprint(transmission_eigenvalues)
```

```
# -------------------------------------------------------------
# Calculator
# -------------------------------------------------------------
#----------------------------------------
# Basis Set
#----------------------------------------
basis_set = [
   LDABasis.Carbon_DoubleZetaPolarized,
   LDABasis.Nitrogen_DoubleZetaPolarized,
   LDABasis.Oxygen_DoubleZetaPolarized,
   LDABasis.Chlorine_DoubleZetaPolarized,
   LDABasis.Gold_SingleZetaPolarized,
   ]

#----------------------------------------
# Numerical Accuracy Settings
#----------------------------------------
left_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(3, 3, 100),
   )

right_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(3, 3, 100),
   )

device_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(3, 3, 100),
   )

#----------------------------------------
# Poisson Solver Settings
#----------------------------------------
left_electrode_poisson_solver = FastFourier2DSolver(
   boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                 [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                 [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
   )

right_electrode_poisson_solver = FastFourier2DSolver(
   boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                 [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                 [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
   )

#----------------------------------------
# Electrode Calculators
#----------------------------------------
left_electrode_calculator = LCAOCalculator(
   basis_set=basis_set,
   numerical_accuracy_parameters=left_electrode_numerical_accuracy_parameters,
   poisson_solver=left_electrode_poisson_solver,
   )

right_electrode_calculator = LCAOCalculator(
   basis_set=basis_set,
   numerical_accuracy_parameters=right_electrode_numerical_accuracy_parameters,
   poisson_solver=right_electrode_poisson_solver,
   )

#----------------------------------------
# Device Calculator
#----------------------------------------
calculator = DeviceLCAOCalculator(
   basis_set=basis_set,
   numerical_accuracy_parameters=device_numerical_accuracy_parameters,
   electrode_calculators=
      [left_electrode_calculator, right_electrode_calculator],
   )

device_configuration.setCalculator(calculator)
nlprint(device_configuration)
device_configuration.update()
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_TES.nc',
device_configuration)

# -------------------------------------------------------------
# Transmission eigenstate
# -------------------------------------------------------------
transmission_eigenstate = TransmissionEigenstate(
   configuration=device_configuration,
   energy=-1.38277*eV,
   k_point=[0, 0],
   quantum_number=0,
    contributions=Left,
    energy_zero_parameter=AverageFermiLevel,
    )
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_TES.nc',
transmission_eigenstate)

# -------------------------------------------------------------
# Transmission eigenstate
# -------------------------------------------------------------
transmission_eigenstate = TransmissionEigenstate(
    configuration=device_configuration,
    energy=-2.50501*eV,
    k_point=[0, 0],
    quantum_number=0,
    contributions=Left,
    energy_zero_parameter=AverageFermiLevel,
    )
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_TES.nc',
transmission_eigenstate)

# -------------------------------------------------------------
# Transmission eigenstate
# -------------------------------------------------------------
transmission_eigenstate = TransmissionEigenstate(
    configuration=device_configuration,
    energy=-2.86573*eV,
    k_point=[0, 0],
    quantum_number=0,
    contributions=Left,
    energy_zero_parameter=AverageFermiLevel,
    )
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_TES.nc',
transmission_eigenstate)
```

A.2.7 Python script for the calculation of the local device density of states analysis.

```
# -------------------------------------------------------------
# Calculator
# -------------------------------------------------------------
#----------------------------------------
# Basis Set
#----------------------------------------
basis_set = [
    LDABasis.Carbon_DoubleZetaPolarized,
    LDABasis.Nitrogen_DoubleZetaPolarized,
    LDABasis.Oxygen_DoubleZetaPolarized,
    LDABasis.Chlorine_DoubleZetaPolarized,
    LDABasis.Gold_SingleZetaPolarized,
    ]

#----------------------------------------
# Numerical Accuracy Settings
#----------------------------------------
left_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
    k_point_sampling=(3, 3, 100),
    )
```

```
right_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
    k_point_sampling=(3, 3, 100),
    )

device_numerical_accuracy_parameters = NumericalAccuracyParameters(
    k_point_sampling=(3, 3, 100),
    )

#----------------------------------------
# Poisson Solver Settings
#----------------------------------------
left_electrode_poisson_solver = FastFourier2DSolver(
    boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                         [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                         [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
    )

right_electrode_poisson_solver = FastFourier2DSolver(
    boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                         [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
                         [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
    )

#----------------------------------------
# Electrode Calculators
#----------------------------------------
left_electrode_calculator = LCAOCalculator(
    basis_set=basis_set,
    numerical_accuracy_parameters=left_electrode_numerical_accuracy_parameters,
    poisson_solver=left_electrode_poisson_solver,
    )

right_electrode_calculator = LCAOCalculator(
    basis_set=basis_set,
    numerical_accuracy_parameters=right_electrode_numerical_accuracy_parameters,
    poisson_solver=right_electrode_poisson_solver,
    )

#----------------------------------------
# Device Calculator
#----------------------------------------
calculator = DeviceLCAOCalculator(
    basis_set=basis_set,
    numerical_accuracy_parameters=device_numerical_accuracy_parameters,
    electrode_calculators=
        [left_electrode_calculator, right_electrode_calculator],
    )

device_configuration.setCalculator(calculator)
nlprint(device_configuration)
device_configuration.update()
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_LDDOS.nc',
device_configuration)

# -------------------------------------------------------------
# Local device density of states
# ---------------------------------------------------------------
energies = numpy.linspace(-10,10,101)
for e in energies:

   local_device_density_of_states = LocalDeviceDensityOfStates(
      configuration=device_configuration,
      energy=e*eV,
      kpoints=MonkhorstPackGrid(3,3),
      contributions=All,
      energy_zero_parameter=AverageFermiLevel,
      infinitesimal=1e-06*eV,
      self_energy_calculator=RecursionSelfEnergy(),
      )
   nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_LDDOS.nc',
local_device_density_of_states)
```

A.3 Parameter optimization python script

This section includes python script for current-voltage characterization of the molecular device configuration.

A.3.1 Python script for I-V characterization of the molecular device configuration.

```
# ---------------------------------------------------------------
# Calculator
# ---------------------------------------------------------------
#----------------------------------------
# Basis Set
#----------------------------------------
basis_set = [
   LDABasis.Carbon_DoubleZetaPolarized,
   LDABasis.Nitrogen_DoubleZetaPolarized,
   LDABasis.Oxygen_DoubleZetaPolarized,
   LDABasis.Chlorine_DoubleZetaPolarized,
   LDABasis.Gold_SingleZetaPolarized,
   ]

#----------------------------------------
# Numerical Accuracy Settings
#----------------------------------------
left_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(10, 10, 100),
   )

right_electrode_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(10, 10, 100),
   )

device_numerical_accuracy_parameters = NumericalAccuracyParameters(
   k_point_sampling=(10, 10, 100),
   )

#----------------------------------------
# Poisson Solver Settings
#----------------------------------------
left_electrode_poisson_solver = FastFourier2DSolver(
  boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
             [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
             [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
  )

right_electrode_poisson_solver = FastFourier2DSolver(
  boundary_conditions=[[PeriodicBoundaryCondition,PeriodicBoundaryCondition],
             [PeriodicBoundaryCondition,PeriodicBoundaryCondition],
             [PeriodicBoundaryCondition,PeriodicBoundaryCondition]]
  )

#----------------------------------------
# Electrode Calculators
#----------------------------------------
left_electrode_calculator = LCAOCalculator(
  basis_set=basis_set,
  numerical_accuracy_parameters=left_electrode_numerical_accuracy_parameters,
  poisson_solver=left_electrode_poisson_solver,
  )

right_electrode_calculator = LCAOCalculator(
  basis_set=basis_set,
  numerical_accuracy_parameters=right_electrode_numerical_accuracy_parameters,
  poisson_solver=right_electrode_poisson_solver,
  )

#----------------------------------------
# Device Calculator
#----------------------------------------
calculator = DeviceLCAOCalculator(
  basis_set=basis_set,
  numerical_accuracy_parameters=device_numerical_accuracy_parameters,
  electrode_calculators=
    [left_electrode_calculator, right_electrode_calculator],
  )

device_configuration.setCalculator(calculator)
nlprint(device_configuration)
device_configuration.update()
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_IV0to+3.nc',
device_configuration)

# -------------------------------------------------------------
# IV curve
# -------------------------------------------------------------
biases = [0.000000, 0.030303, 0.060606, 0.090909, 0.121212, 0.151515,
     0.181818, 0.212121, 0.242424, 0.272727, 0.303030, 0.333333,
     0.363636, 0.393939, 0.424242, 0.454545, 0.484848, 0.515152,
     0.545455, 0.575758, 0.606061, 0.636364, 0.666667, 0.696970,
     0.727273, 0.757576, 0.787879, 0.818182, 0.848485, 0.878788,
     0.909091, 0.939394, 0.969697, 1.000000, 1.030303, 1.060606,
     1.090909, 1.121212, 1.151515, 1.181818, 1.212121, 1.242424,
     1.272727, 1.303030, 1.333333, 1.363636, 1.393939, 1.424242,
     1.454545, 1.484848, 1.515152, 1.545455, 1.575758, 1.606061,
     1.636364, 1.666667, 1.696970, 1.727273, 1.757576, 1.787879,
     1.818182, 1.848485, 1.878788, 1.909091, 1.939394, 1.969697,
     2.000000, 2.030303, 2.060606, 2.090909, 2.121212, 2.151515,
        2.181818, 2.212121, 2.242424, 2.272727, 2.303030, 2.333333,
        2.363636, 2.393939, 2.424242, 2.454545, 2.484848, 2.515152,
        2.545455, 2.575758, 2.606061, 2.636364, 2.666667, 2.696970,
        2.727273, 2.757576, 2.787879, 2.818182, 2.848485, 2.878788,
        2.909091, 2.939394, 2.969697, 3.000000]*Volt

iv_curve = IVCurve(
   configuration=device_configuration,
   biases=biases,
   energies=numpy.linspace(-10,10,100)*eV,
   kpoints=MonkhorstPackGrid(10,10),
   self_energy_calculator=KrylovSelfEnergy(),
   energy_zero_parameter=AverageFermiLevel,
   infinitesimal=1e-06*eV,

selfconsistent_configurations_filename="/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_A
u_IV0to+3_SCF.nc",
   )
nlsave('/home/giri/.vnl/example_project_13.8.1/Au_ddq_OG_test5_Au_IV0to+3.nc', iv_curve)
nlprint(iv_curve)
```