Libra/CP2K Interface

Mohammad Shakiba

SUNY Buffalo

Akimov Research Group

University at Buffalo The State University of New York

CP2K

- CP2K is a quantum chemistry and solid state physics software package that can perform atomistic simulations of solid state, liquid, molecular, periodic, material, crystal, and biological systems.
- CP2K provides a general framework for different modeling methods such as DFT using the mixed Gaussian and plane waves approaches GPW and GAPW.
- Supported theory levels include DFTB, LDA, GGA, MP2, RPA, semi-empirical methods (AM1, PM3, PM6, RM1, MNDO, …), and classical force fields (AMBER, CHARMM, …).
- CP2K can do simulations of molecular dynamics, metadynamics, Monte Carlo, Ehrenfest dynamics, vibrational analysis, core level spectroscopy, energy minimization, and transition state optimization using NEB or dimer method.

Theory

- A molecular orbital (MO) is defined as a linear combination of atomic orbitals:

$$|\psi_n>=\sum_i c_i|\phi_i>$$

- The MO overlap:

$$S_{nm}=<\psi_n|\psi_m>=\sum_{i,j} c_i^* c_j<\phi_i|\phi_j>$$

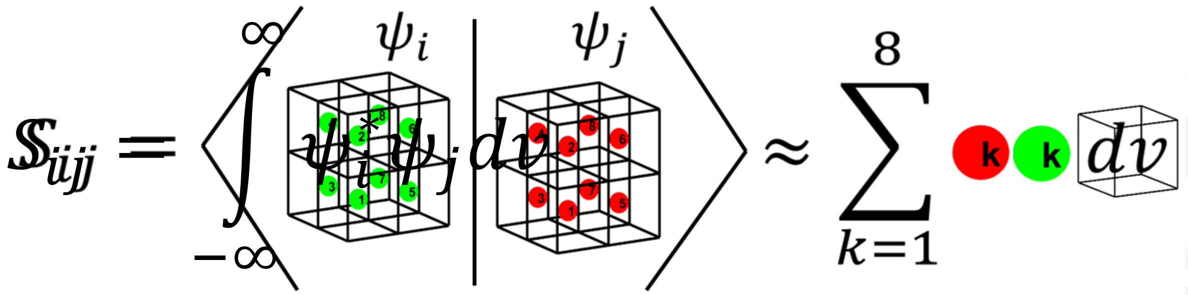
the $<\phi_i|\phi_j>$ is the atomic orbital overlap. So the MO overlap matrix ($\boldsymbol{S^{MO}}$) can be written as follows where $\boldsymbol{c}$ is the matrix of molecular orbital coefficients and $\boldsymbol{S^{AO}}$ is the atomic orbital overlap matrix:

$$\boldsymbol{S^{MO}}=\boldsymbol{c^T S^{AO} c}$$

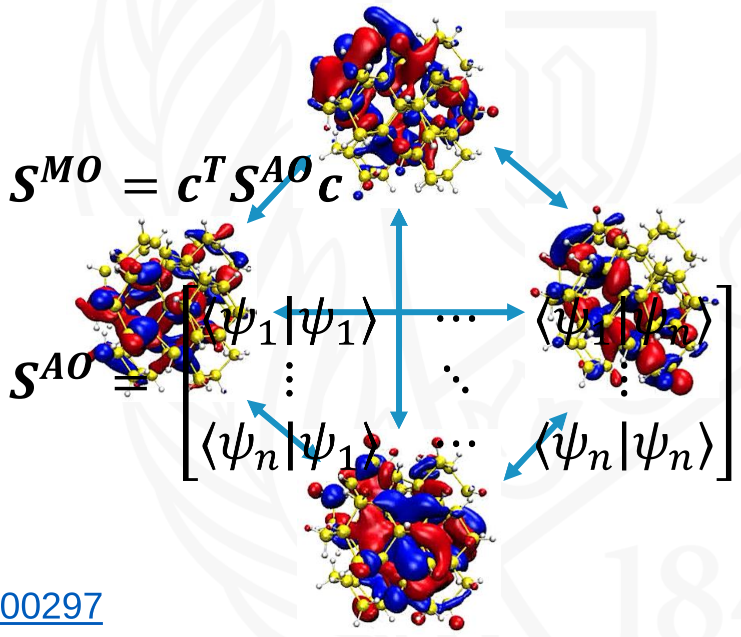
Molecular orbital integrals

- Grid-based approach using *.cube* files
  - Easy to implement
  - Most codes can output these file
  - Not suitable for large structures with large number of states

- Analytical approach
  - Suitable for large systems and large number of states
  - One interface cannot directly be used for many codes



$$S_{ij} = \left\langle \psi_i \middle| \psi_j \right\rangle = \int_{-\infty}^{\infty} \psi_i^* \psi_j \, dv \approx \sum_{k=1}^{8} dv$$



$$S^{MO} = c^T S^{AO} c$$

$$S^{AO} = \begin{bmatrix} \langle \psi_1 | \psi_1 \rangle & \cdots & \langle \psi_1 | \psi_n \rangle \\ \vdots & \ddots & \vdots \\ \langle \psi_n | \psi_1 \rangle & \cdots & \langle \psi_n | \psi_n \rangle \end{bmatrix}$$

Gaussian type orbitals (GTO)

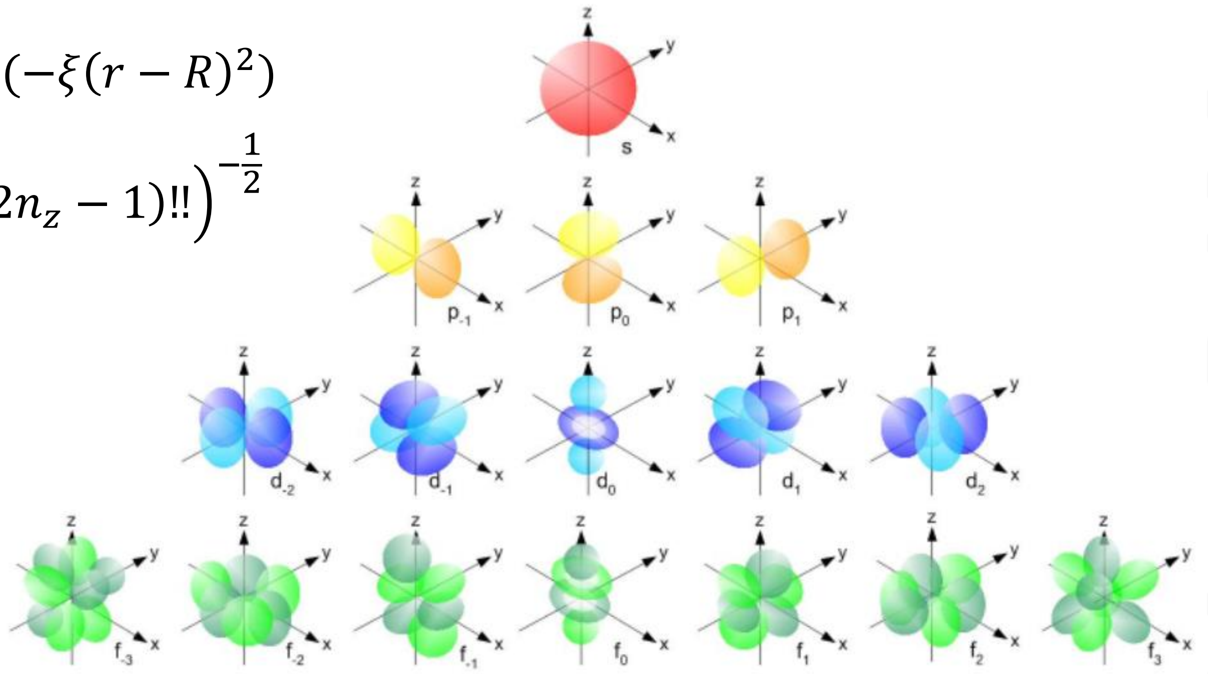
- Atom centered basis sets

$$\varphi(r-R;n,\xi) = N(x-R_x)^{n_x}\left(y-R_y\right)^{n_y}(z-R_z)^{n_z} \times \exp(-\xi(r-R)^2)$$

$$N = \left(\frac{2\xi}{\pi}\right)^{\frac{3}{4}} (4\xi)^{(n_x+n_y+n_z)/2} \times \left((2n_x-1)!!\,(2n_y-1)!!\,(2n_z-1)!!\right)^{-\frac{1}{2}}$$

$$\Phi_i(\vec{r}) = \varphi(r;n_i,\xi).\,Y_{n_i,m_i}(\theta,\phi)$$

n shows the angular momentum value:
s-orbital $n=0$
p-orbital $n=1$
d-orbital $n=2$
f-orbital $n=3$



Recurrence relations between GTOs

$$\langle s|s\rangle = \left(\frac{\pi}{\xi}\right)^{\frac{3}{2}} \exp(-\xi(A-B)^2)$$

$$\langle p_i|s\rangle = (P_i - A_i)\langle s|s\rangle$$

$$\langle p_i|p_j\rangle = (P_j - B_j)\langle p_i|s\rangle + \frac{\delta_{ij}}{2\xi}\langle s|s\rangle$$

$$\langle d_{ij}|s\rangle = (P_j - A_j)\langle p_i|s\rangle + \frac{\delta_{ij}}{2\xi}\langle s|s\rangle$$

$$\langle d_{ij}|p_k\rangle = (P_k - B_k)\langle d_{ij}|s\rangle + \frac{\delta_{ik}}{2\xi}\langle p_j|s\rangle + \frac{\delta_{jk}}{2\xi}\langle p_i|s\rangle$$

$$\langle d_{ij}|d_{kl}\rangle = (P_l - B_l)\langle d_{ij}|p_k\rangle + \frac{\delta_{il}}{2\xi}\langle p_j|p_k\rangle + \frac{\delta_{jl}}{2\xi}\langle p_i|p_k\rangle + \frac{\delta_{kl}}{2\xi}\langle d_{ij}|s\rangle$$

$i, j, k, l = x, y, z$

LIBINT

Open-source and OpenMP parallelized code for computing integrals between GTOs

Obara, and Saika, J. Chem. Phys. 1986, 84, 3963-3974.
Obara, and Saika, J. Chem. Phys. 1988, 89, 1540-1559.
Libint, Version 2.6.0 Edward F. Valeev, http://libint.valeyev.net.

Periodic basis functions

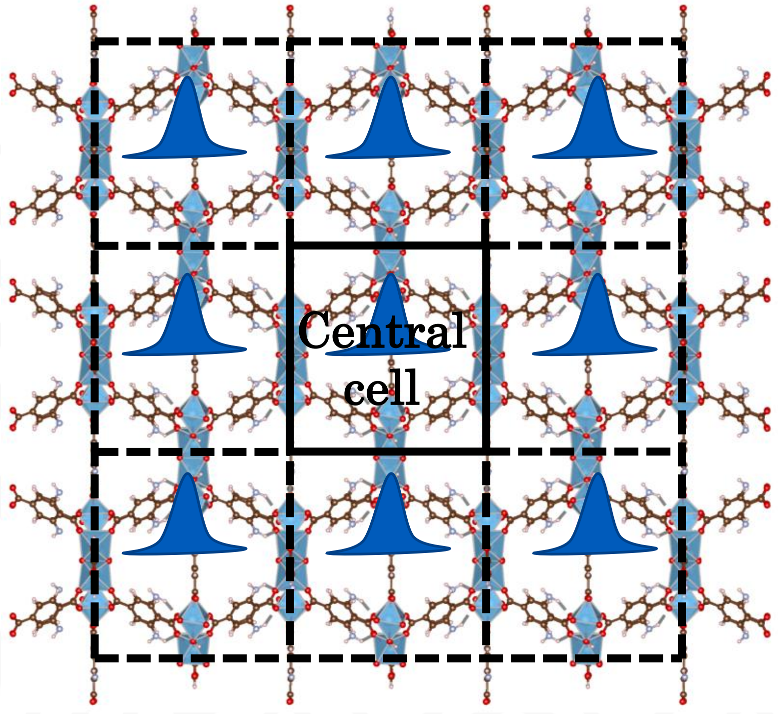
- The Bloch function for K-point in a periodic structure is defined as:

$$\beta_a^k(r) = \frac{1}{\sqrt{N}} \sum_R \varphi_a(r-R)e^{ikR}$$

- Overlaps between Bloch functions of two different K-points:

$$S_{a,b}^k = \langle \beta_a^k | \beta_b^{k'} \rangle = \frac{1}{N} \int dr \sum_{R,R'} e^{-ikR} \varphi_a^*(r-R) e^{ik'R'} \varphi_b(r-R')$$

$$= \frac{1}{N} \int dr \sum_{R,R'} e^{i(kR-k'R')} \varphi_a^*(r-R)\varphi_b(r-R')$$



MIL-125-NH_2

Aradi, Hourahine, Frauenheim, J. Phys. Chem. A 2007, 111, 5678–5684

Shakiba, Stippell, Li, Akimov, J. Chem. Theory Compu. 2022, DOI: 10.1021/acs.jctc.2c00297

How Libra computes the MO overlap?

- Libra uses ‘*molden’* file formats to read the molecular orbital coefficients, energies, occupation, spin, atomic coordinates, and basis set although it can use other file formats as well including *MOLog* printed out by CP2K.
- Here, we’ll show an example of the inner functions that compute the MO overlap matrix for one geometry. Note that the workflow is not like this and you only need to specify a couple of variables. The following shows how one can work with the Libra functions, for example, if one intends to write an on-the-fly computation of the NACs.

```
import os
import numpy as np
import matplotlib.pyplot as plt
from liblibra_core import *
from libra_py import CP2K_methods,molden_methods, data_conv, units
```

```
# Molden file name
molden_file_name = 'test.molden'
# Number of processors
nprocs = 16
# Spherical or Cartesian coordiantes? Spherical!
is_spherical = True
```

```
# The integration shells and angular momentum values
shell_1, l_vals =
molden_methods.molden_file_to_libint_shell(molden_file_name,
is_spherical)
```

```
# All of the eigenvectors and energies of the system
eig_vect_1, energies_1 =
molden_methods.eigenvectors_molden(molden_file_name, nbasis(shell_1),
l_vals)
```

```
# Resorting the molden indices
new_indices = CP2K_methods.resort_molog_eigenvectors(l_vals)
eigenvectors_1 = []
for j in range(len(eig_vect_1)):
    # the new and sorted eigenvector
    eigenvector_1 = eig_vect_1[j]
    eigenvector_1 = eigenvector_1[new_indices]
    # append it to the eigenvectors list
    eigenvectors_1.append(eigenvector_1)
eigenvectors_1 = np.array(eigenvectors_1)
```

```
# Alpha and Beta spin eigenvectors
# alpha -> even indices
alpha_eig_vects = eigenvectors_1[0::2]
alpha_eig_vals = energies_1[0::2]
# beta -> odd indices
beta_eig_vects = eigenvectors_1[1::2]
beta_eig_vals = energies_1[1::2]
# Compute AO overlap matrix
AO_S = compute_overlaps(shell_1, shell_1, nprocs)
# Converting to numpy array
AO_S = data_conv.MATRIX2nparray(AO_S)
```

```
print('The shape of the AO matrix...\n', AO_S.shape)
S_alpha = np.linalg.multi_dot([alpha_eig_vects, AO_S,
alpha_eig_vects.T])
S_beta = np.linalg.multi_dot([beta_eig_vects, AO_S,
beta_eig_vects.T])
```

Periodic calculations

```
AO_S = compute_overlaps(shell_1,shell_1,nprocs)
if is_periodic:
    cell = []
    cell.append(params['A_cell_vector'])
    cell.append(params['B_cell_vector'])
    cell.append(params['C_cell_vector'])
    cell = np.array(cell)*units.Angst
    # Generating translational vectors
    translational_vectors = params['translational_vectors']
    for i1 in range(len(translational_vectors)):
        translational_vector = np.array(translational_vectors[i1])
        print(F'Computing the AO overlaps between R({translational_vector[0]},{translational_vector[1]},
               {translational_vector[2]}) and R(0,0,0)')
        shell_1p, l_vals =
        molden_methods.molden_file_to_libint_shell(molden_filename,is_spherical,is_periodic,cell,
        translational_vector)
        AO_S += compute_overlaps(shell_1,shell_1p, nprocs)
```

What do we need to specify in the input?

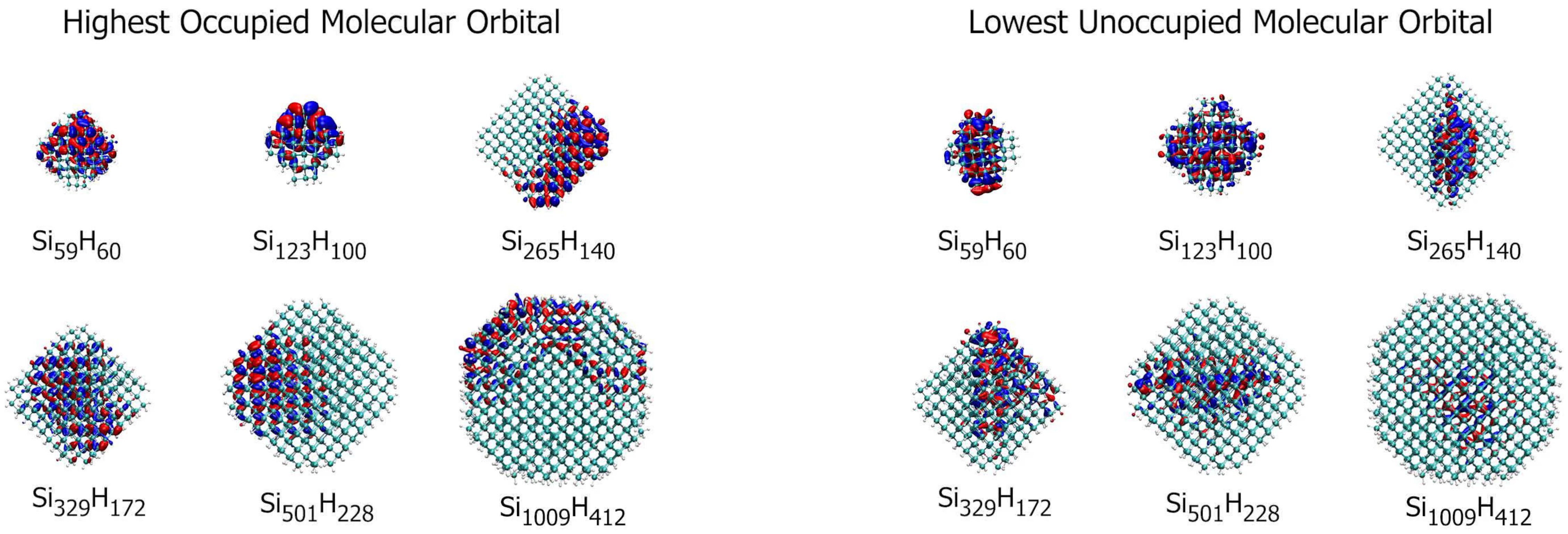
```
params['nprocs']
params['mpi_executable']
params['istep']
params['fstep']
params['lowest_orbital']
params['highest_orbital']
params['isxTB']
params['isUKS']
params['is_periodic']
if params['is_periodic']:
    params['A_cell_vector']
    params['B_cell_vector']
    params['C_cell_vector']
    params['periodicity_type'] # example: 'XYZ'
    origin = [0,0,0]
    params['translational_vectors'] =
    CP2K_methods.generate_translational_vectors(
    origin, [2,2,2], params['periodicity_type'])
params['is_spherical']
params['remove_molden']
params['res_dir']
params['all_pdosfiles']
params['all_logfiles']
```

```
params['cp2k_exe']
params['cp2k_ot_input_template'] # just for xTB
params['cp2k_diag_input_template']
params['trajectory_xyz_filename']

# For cube visualization
params['cube_visualization']
params['vmd_input_template']
params['states_to_plot']
params['plot_phase_corrected']
params['vmd_exe']
params['tachyon_exe']
params['x_pixels']
params['y_pixels']
params['image_format']
params['remove_cube']
params['all_images']


step2.run_cp2k_libint_step2(params)
```

Molecular orbitals visualization

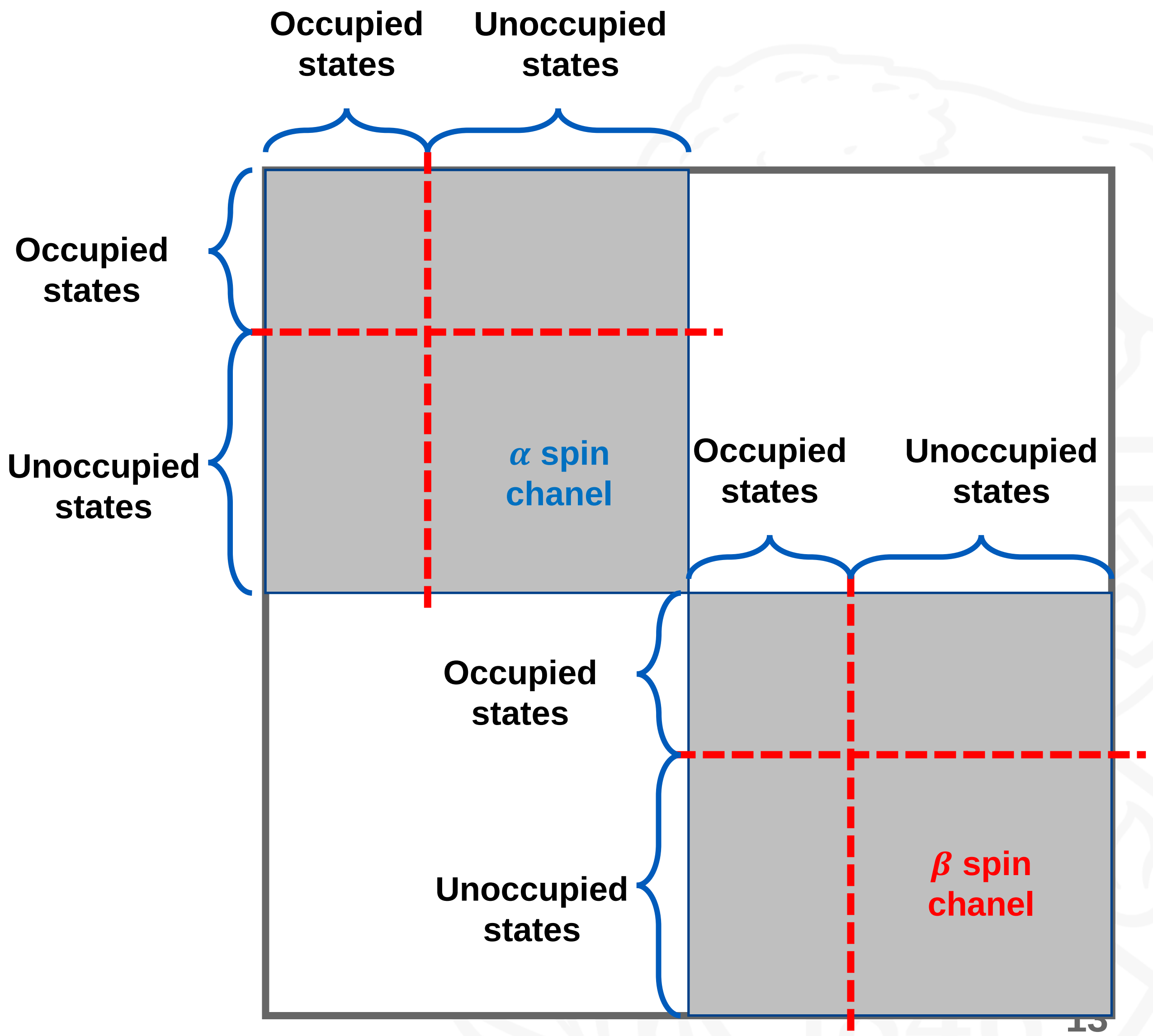


0 0

University at Buffalo The State University of New York

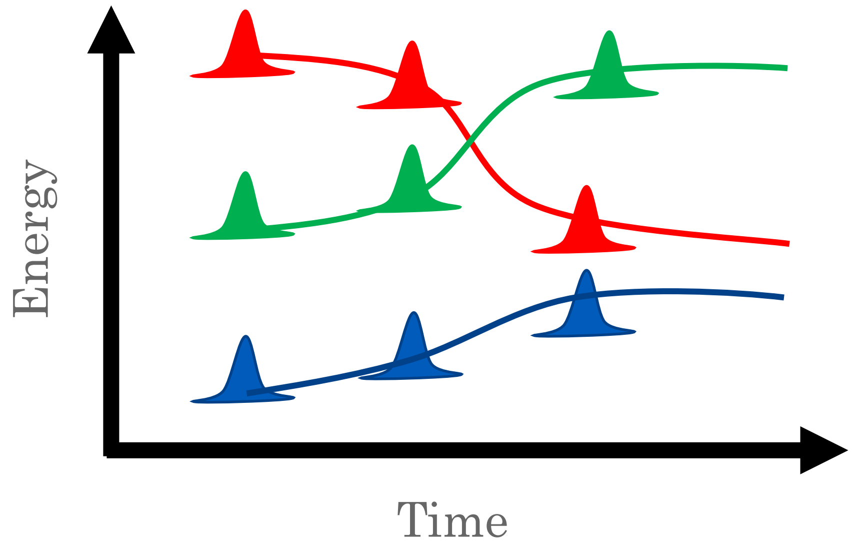
Storing data

- Libra saves MO overlaps in 2-spinor format
- With no spin-orbit coupling, the second and third block of the matrix is zero.
- To efficiently storing the data, we use `scipy.sparse` library of Python



Corrections to time-overlap matrix

- State tracking
  - Min-cost algorithm
  - Stochastic state-tracking

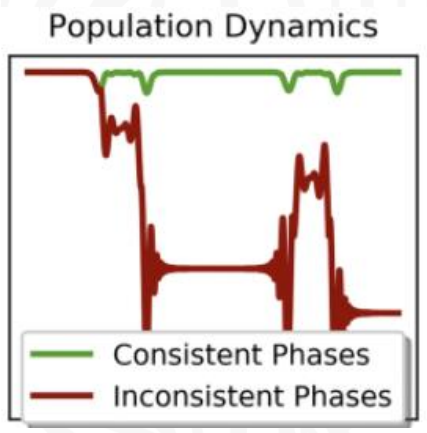


- Phase correction

$$F_i(t+\Delta t) = \frac{\langle \Psi_i(t) | \Psi_i(t+\Delta t) \rangle}{\|\Psi_i(t)\|^* \|\Psi_i(t+\Delta t)\|}$$

The phase-corrected wave function at time $t+\Delta t$ is:

$$\overline{\Psi_i}(t+\Delta t) = F_i^*(t+\Delta t)\Psi_i(t+\Delta t)$$

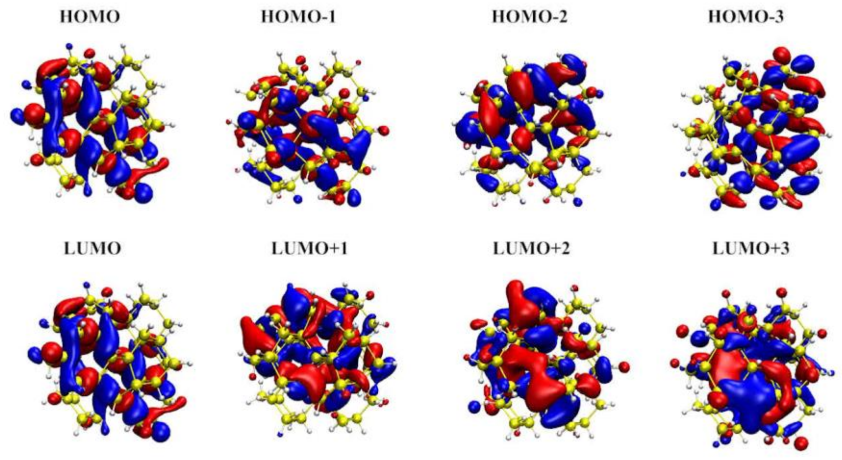


Fernandez-Alberti, Roitberg, Nelson, Tretiak, J. Chem. Phys. 2012, 137, 014512
Temen, Akimov, J. Phys. Chem. Lett. 2021, 12, 850–860.
Akimov, J. Phys. Chem. Lett. 2018, 9, 6096–6102

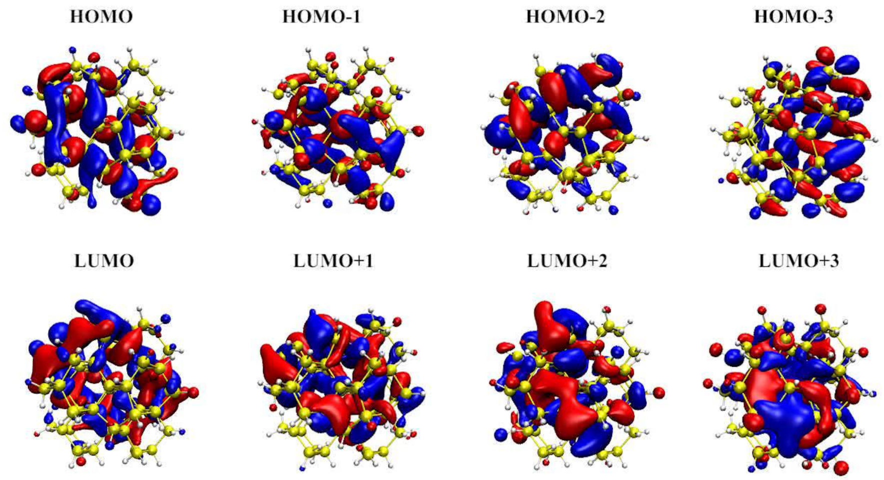
Corrections to time-overlap matrix

- Phase correction

Phase-uncorrected orbitals



Phase-corrected orbitals

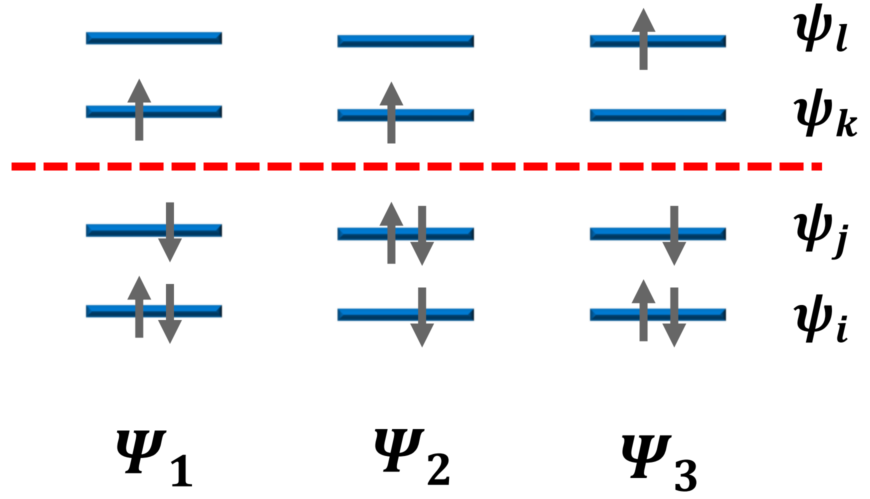


Excited states basis

- In quantum mechanics, electronic wave function has the antisymmetric property which can be expressed in form of Slater determinant.
- For an N-electron system:

$$\Psi(x_1, .., x_N) = \frac{1}{\sqrt{N!}} |\psi_1(x_1)\psi_2(x_2) \ldots \psi_K(x_N)\rangle = \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_1(x_1) \cdots \psi_K(x_1) \\ \vdots \quad \ddots \quad \vdots \\ \psi_1(x_N) \cdots \psi_K(x_N) \end{vmatrix}$$

- Single-particle excitations



$$\Psi_1 = |\psi_i \bar{\psi}_i \psi_k \bar{\psi}_j\rangle$$

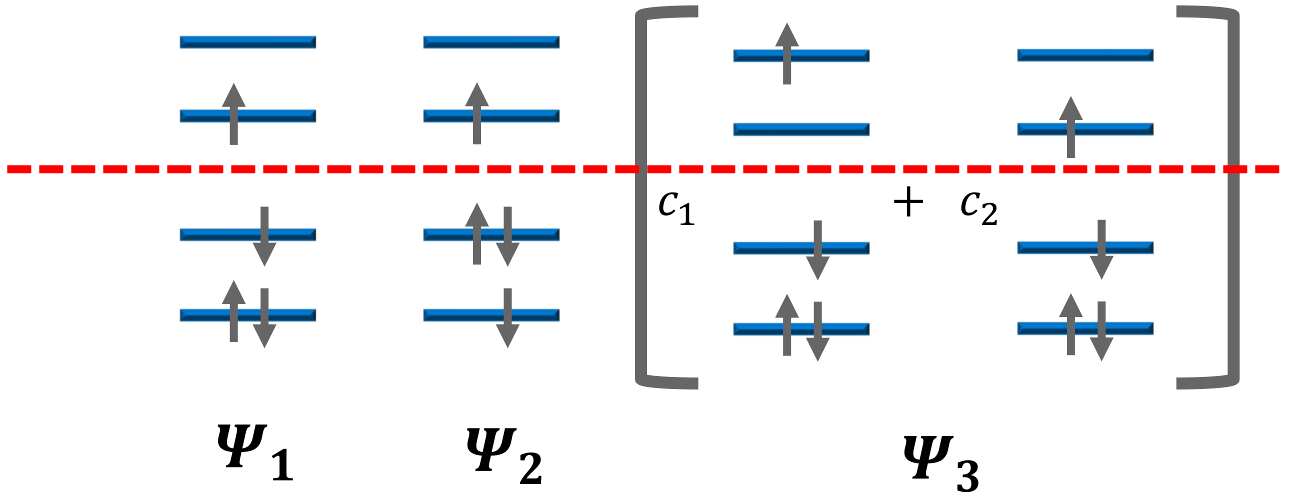
$$\Psi_2 = |\psi_k \bar{\psi}_i \psi_j \bar{\psi}_j\rangle$$

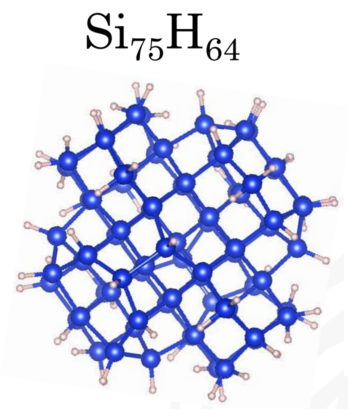
$$\Psi_3 = |\psi_i \bar{\psi}_i \psi_l \bar{\psi}_j\rangle$$

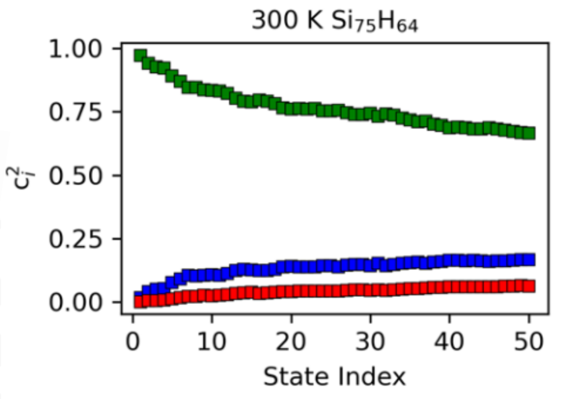
$$\langle \Psi_1 | \Psi_2 \rangle = \det \begin{vmatrix} \langle\psi_i|\psi_k\rangle & \langle\bar{\psi}_i|\psi_k\rangle & \langle\psi_k|\psi_k\rangle & \langle\bar{\psi}_j|\psi_k\rangle \\ \langle\psi_i|\bar{\psi}_i\rangle & \langle\bar{\psi}_i|\bar{\psi}_i\rangle & \langle\psi_k|\bar{\psi}_i\rangle & \langle\bar{\psi}_j|\bar{\psi}_i\rangle \\ \langle\psi_i|\psi_j\rangle & \langle\bar{\psi}_i|\psi_j\rangle & \langle\psi_k|\psi_j\rangle & \langle\bar{\psi}_j|\psi_j\rangle \\ \langle\psi_i|\bar{\psi}_j\rangle & \langle\bar{\psi}_i|\bar{\psi}_j\rangle & \langle\psi_k|\bar{\psi}_j\rangle & \langle\bar{\psi}_j|\bar{\psi}_j\rangle \end{vmatrix}$$

Excited states basis

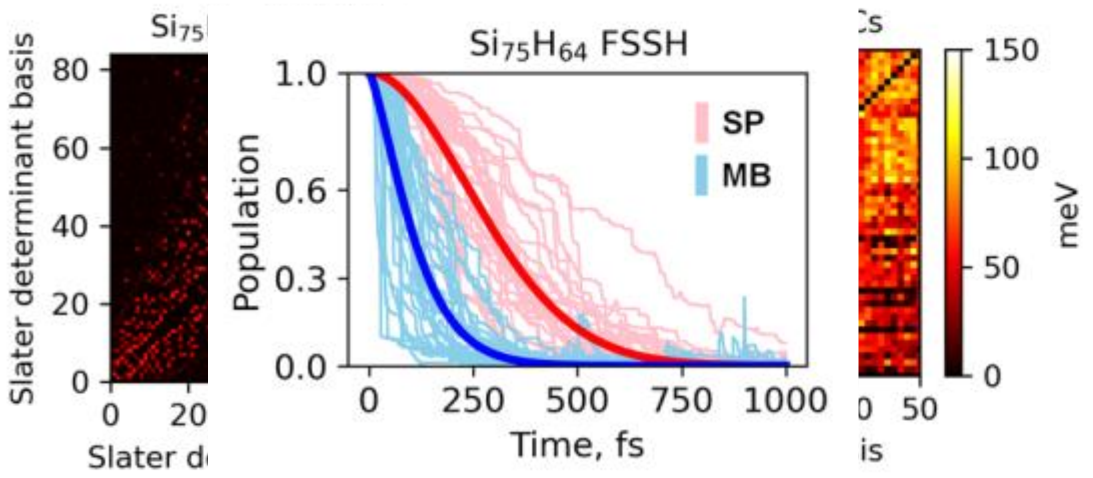
- Many-body (TD-DFT) excitation







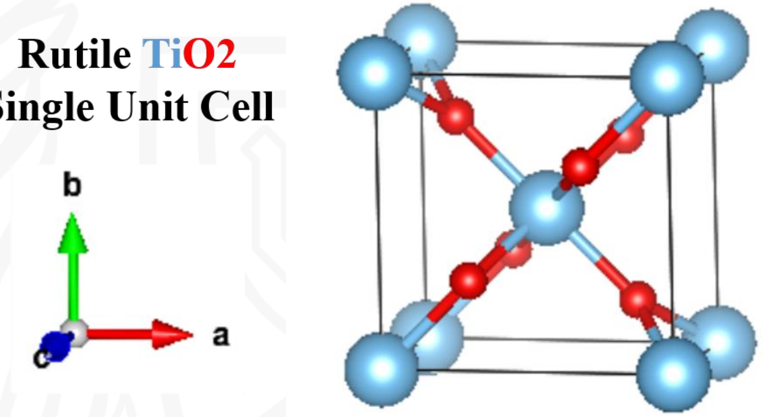
Excitation energy



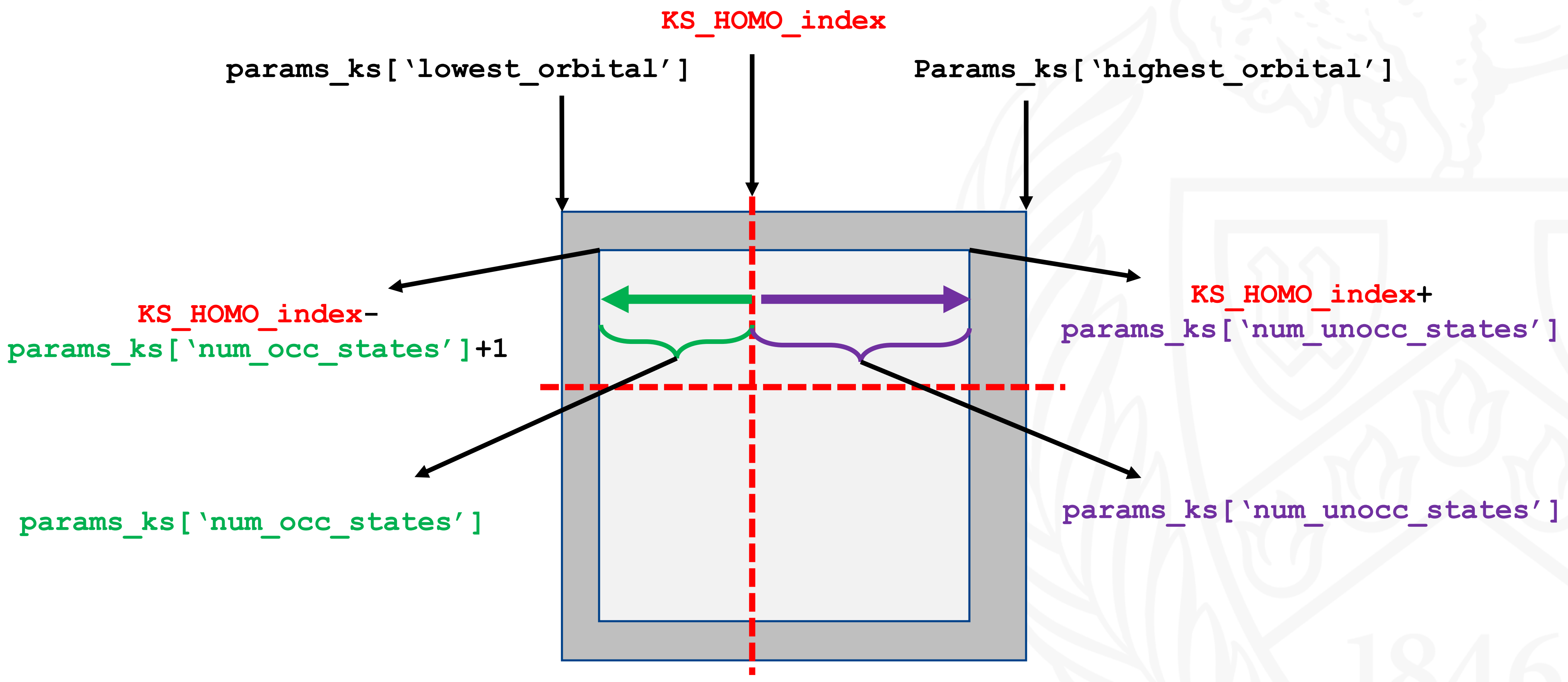
NAC in KS basis

```
params_ks = { 'lowest_orbital': 24-10, 'highest_orbital': 24+11,
'num_occ_states': 10, 'num_unocc_states': 10, 'use_multiprocessing': True,
'nprocs': 8, 'time_step': 1.0, 'es_software': 'cp2k', 'path_to_npz_files':
os.getcwd()+'/res', 'logfile_directory': os.getcwd()+'/all_logfiles',
'path_to_save_ks_Hvibs': os.getcwd()+'/res-ks-DFT', 'start_time': 1200,
'finish_time': 1401, 'apply_phase_correction': True,
'apply_orthonormalization': True, 'do_state_reordering': 2,
'state_reordering_alpha':0, 'nac_algo': 0 }
# For KS states - Applying correction to KS overlaps and computing the
NACs in KS space
step3.run_step3_ks_nacs_libint(params_ks)
```

Rutile TiO_2
Single Unit Cell



NAC in KS basis


KS_HOMO_index
params_ks['lowest_orbital']
Params_ks['highest_orbital']
KS_HOMO_index-
params_ks['num_occ_states']+1
KS_HOMO_index+
params_ks['num_unocc_states']
params_ks['num_occ_states']
params_ks['num_unocc_states']

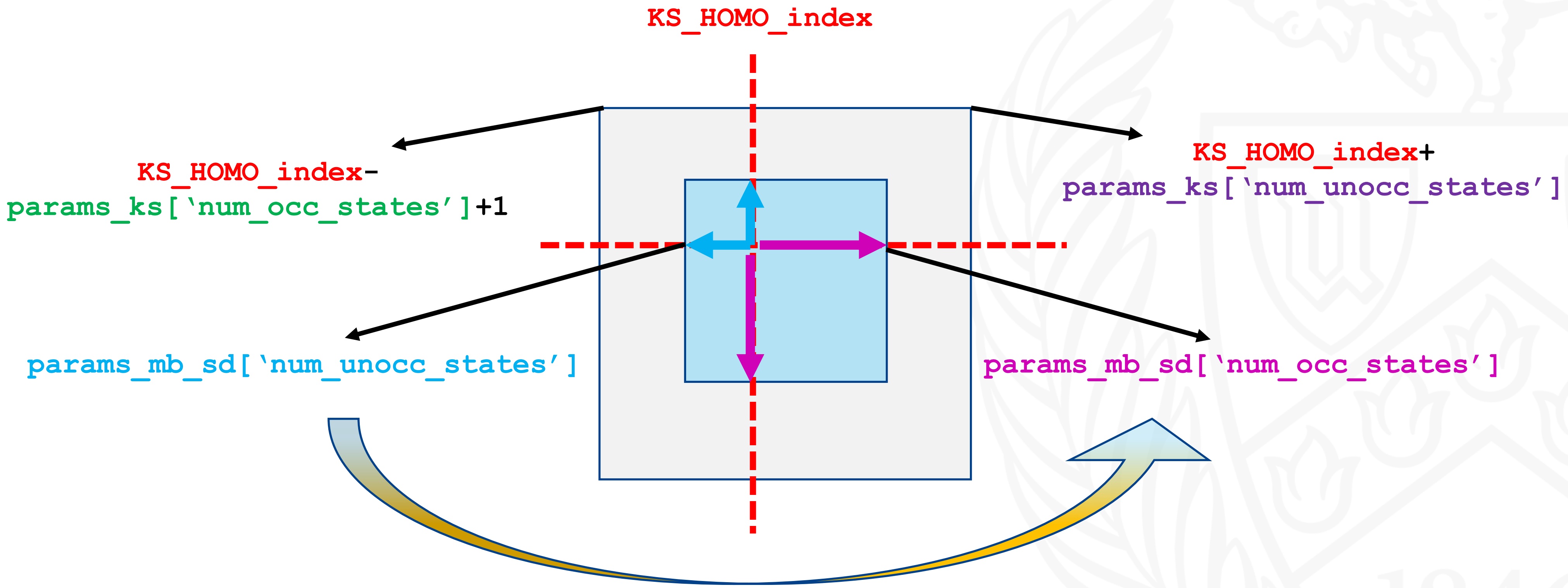
University at Buffalo The State University of New York

NAC in excited states basis

```
params_mb_sd = {
          'lowest_orbital': 24-params_ks['num_occ_states']+1, 'highest_orbital': 24+params_ks['num_unocc_states'],
          'num_occ_states': 10, 'num_unocc_states': 10,
          'isUKS': 0, 'number_of_states': 10, 'tolerance': 0.01, 'verbosity': 0,
          'use_multiprocessing': True, 'nprocs': 12,
          'is_many_body': True, 'time_step': 1.0, 'es_software': 'cp2k',
          #'path_to_npz_files': os.getcwd()+'/../../7_step2_cp2k/1_DFT/2_hpc/1_example_TiO2/res',
          'path_to_npz_files': os.getcwd()+'/res-ks-DFT',
          'logfile_directory': os.getcwd()+'/../../7_step2_cp2k/1_DFT/2_hpc/1_example_TiO2/all_logfiles',
          'path_to_save_sd_Hvibs': os.getcwd()+'/res-mb-sd-DFT',
          'outdir': os.getcwd()+'/res-mb-sd-DFT',
          'start_time': 1200, 'finish_time': 1401, 'sorting_type': 'identity',
          'apply_phase_correction': True, 'apply_orthonormalization': True,
          'do_state_reordering': 2, 'state_reordering_alpha':0, 'nac_algo': 0
         }


step3.run_step3_sd_nacs_libint(params_mb_sd)
```

NAC in excited states basis


KS_HOMO_index
KS_HOMO_index-
params_ks['num_occ_states']+1
KS_HOMO_index+
params_ks['num_unocc_states']
params_mb_sd['num_unocc_states']
params_mb_sd['num_occ_states']

Summary

- All these methodologies are implemented and available in Libra software package
  - Open-source code for quantum dynamics methodologies such as trajectory surface hopping
  - The underlying code is written in C++ for faster computation and the functions can be called from Python
  - Libint is used for computation of overlaps between GTOs
  - Sparse representation of the overlap matrices using `scipy.sparse` library in Python
  - High-throughput computation for generating the overlap matrices
  - Applicable to large systems in different electronic structure calculations frameworks such as density functional tight-binding
  - It is interfaced with many quantum chemistry codes such as CP2K, Quantum ESPRESSO, and Gaussian but the Libint interface is only available for CP2K code.

Thank You!

Questions?