Documentation and Rearchitecture

Vincent Sacksteder IV*

(Dated: December 7, 2023)

I. COMSUITE DFT+DMFT

A. Variables

- i is an index over impurities.
- o is an index over orbitals at a specific impurity.
- q is an index over momenta, the same momenta used by CASTEP.
- k is an index over momenta on a fine mesh, after interpolation using Wannier functions.
- s is an index over spins.
- b is an index over bands.
- $\beta = (8.6173303 \times 10^{-5} \times T)$ is the inverse temperature.
- n_omega = N_ω is the number of frequencies on the frequency grid. If the user supplies a self-energy file then N_ω is read from that file. Otherwise it is set to $N_\omega = 150 \times \beta/\pi$.
- ω is a grid of real numbers: $\pi/\beta, 3\pi/\beta, 5\pi/\beta, ..., (2N_\omega - 1)/\beta$.
  - comdmft.py does not know anything about whether it is doing its calculations in real or imaginary frequencies.
  - comdmft.py writes ω to sig.dat, gimp.dat, sig_bare.dat, sig_smth.dat, delta.dat, and some files used by ComDC including g_loc.dat.
  - During a qsGW+DMFT calculation comdmft.py uses the first element of the grid, π/β, when processing information returned by ComDC.
  - comdmft.py also uses the grid when doing gaussian broadening of the self-energy's frequency dependence.
  - The ComCTQMC solver does not receive N_ω or the ω grid in its input json files; instead it receives only β, and instructions to use Matsubara frequencies. However I think it can count N_ω from the number of entries in hyb.json. On the other hand, it also receives energy cutoffs, so maybe it decides how many Matsubara frequencies to use based on the cutoffs.
  - ComLowH receives the omega grid in sig.dat. When called during self-consistent DFT+DMFT or qsGW+DMFT iterations it multiplies the grid by $\imath$. It does the same when calculating qsGW spectra. When calculating other observables it leaves the grid at its real value, without any imaginary part.
- The 'spin_orbit' parameter in comdmft.ini, if it is false, means that the impurity orbitals are cubic spherical harmonics. If it is true then they are spin-angle harmonics. It seems that it might be fully implemented only for 'f' shell impurities, and not for s, p, or d shells. It controls the following things:
  - If true, and if the impurity is 'f' shell, then it doubles the number of entries in the 'impurity_wan' variable from 7 to 14.
  - If true, and if the impurity is 'f' shell, then it doubles the basis size of Σ_{DC}.
  - It changes how the contents of wannier.inip are read in to the wan_hmat variable. This file, if it exists, sets a preferred axis for the impurity orbitals.
  - If true, it doubles the size of three matrices embedded in params.json: ["basis"]["transformation"] which contains the T variable, ["hybridisation"]["matrix"], and ["hloc"]["one body"].
  - If true, it changes the ["basis"]["type"] variable in params.json from "product" to "coupled".

  - If true, it instructs ComCTQMC to measure "J2" and "Jz" observables instead of "S2" and "Sz" observables.
  - It becomes the is_spinorbit variable in ComLowH. Here again it controls how wannier.inip is read in. It also doubles the electron count when calculating the Fermi level. And, if true, it ensures that the number of bands determined by find_bnd_in_win (inside the energy window) is odd.
  - It is the second entry in comdc.ini, and becomes the is_spinorbit variable. ComDC is used only if doing a qsGW+DMFT calculation.

- The 'impurity_problem' variable in comdmft.ini is used to form 'impurity_wan' variable. It is also used for processing the output of ComCoulomb, which is run only if doing a qsGW+DMFT calculation.

- The 'problem' variable (The s, p, d, or f in 'impurity_problem' in comdmft.ini) determines the basis size and value of Σ_{DC}, and also the number of $F(0), ...F(6)$ variables which are used.

- The 'impurity_problem_equivalence' variable in comdmft.ini sets the total number of impurities, and defines which impurity problems are equivalent. Its values are positive integers starting from 1. If two entries have the same integer, the corresponding impurities are equivalent. If there is a negative entry, then that impurity is equivalent to the entry with its opposite (positive) value, but is related to that impurity by AFM (antiferromagnetic) symmetry.

- 'para': (Not in comdmft.ini.) When an impurity is marked by a negative equivalence number, signaling that it is connected by antiferromagnetic symmetry to another impurity, then the 'para' variable is set to False. This variable is not in comdmft.ini, only in comdmft.py. It is used when choosing an initial value of Σ, when doubling the size of three matrices (going from one spin value to two) in params.json which are fed to ComCTQMC, when writing to hyb.json, when smoothing the self-energy generated by ComCTQMC, and when figuring out how much Σ was changed by ComCTQMC.

- The 'impurity_matrix' variable keeps track of whether certain orbitals within an impurity are equivalent. It is ignored if the 'trans_basis_mode' is equal to 2, but that variable's default value is zero, so usually it is not ignored. It is used to reduce the size of matrices associated with impurities. The 'impurity_matrix' variable is also used for interfacing with ComCTQMC, and supplies the matrix structure of the hybridization term, multiplying $\hat{\Delta}_{imp}(i,\omega)$ which is a scalar. It is also used when matrices are converted from matrix format to array (vector format), or vice versa.

- The 'doping' parameter in comdmft.ini is passed to ComLowH, where it becomes the delta_charge variable. It is added to the computed charge during the search for the Fermi energy μ.

- $\lambda_{DFT}(q,b,s)$ are the CASTEP DFT eigenvalues, and are stored in wannier.dat.

- $V_{w90}(q,b,o)$ is the projectors created by Wannier 90. Stored in wannier.dat.

- $\eta(i)$ is a user-supplied 'nominal_n' parameter in comdmft.ini, i.e. a user-supplied occupancy that controls the double-counting energy.

- $F(0,i)$, $F(2,i)$, $F(4,i)$, $F(6,i)$ are the parameters of the interaction term. These come from the 'f0', 'f2', 'f4', and 'f6' entries in comdmft.ini.

- T is a user-supplied transformation matrix that defaults to the identity if if is not supplied. Stored in trans_basis.dat

- $\Sigma_{DC}(i)$ is stored in dc_mat.dat and dc.dat. It is produced by comdmft.py, and is an input to ComLowH. In a qsGW+DMFT calculation it is produced by ComDC.

- $\Sigma(i,\omega)$ is stored in sig.dat. It is initialized by comdmft.py, and mixed and smoothed and written again by comdmft.py using data from ComCTQMC. It is used by ComLowH.

- μ, the Fermi energy, is stored in ef.dat. It is initialized by comdmft.py and re-calculated by ComLowH. It is used only by ComLowH.

- $\hat{E}_{imp}(i)$ is stored in e_projected_mat.dat. It is a projected hamiltonian. It is produced by ComLowH, processed by comdmft.py, and used by ComCTQMC.

- $\hat{\Delta}_{imp}(i,\omega)$ is stored in delta_mat.dat. It is produced by ComLowH, processed by comdmft.py, and used by ComCTQMC.
- The density matrix $\hat{\rho}$ is stored in wannier_den_matrix.dat. It is produced by ComLowH, and used by FlapwMBPT.
- $\hat{G}_{loc}(i,\omega)$ is stored in g_loc_mat.dat and g_loc.dat. It is a projected Green's function. It is produced by ComLowH. It is used only by ComDC, and only if doing qsGW+DMFT.
- ComLowH produces local_spectral_matrix_ef.dat. This is used only if the 'trans_basis_mode' control parameter is equal to 2, and then is used only to determine trans_basis.dat. The default value of the 'trans_basis_mode' control parameter is 0.
- ComLowH produces n_loc.dat, which is never used elsewhere.

B. Initialization for the Outer Loop

- Run FlapwMBPT to do the DFT calculation, using the density matrix $\hat{\rho}$ created by ComLowH. It uses an ini file created by flapwmbpt_ini.read_comdmft_ini_wan().
- Run ComWann, using check_wannier_function_input and run_comwann. ComWann extracts information from the DFT calculation and puts that information in wannier.dat which will be used by comdmft.py and ComLowH, and wannier.inip which is used only by ComLowH. The ini file for ComWann is created by create_comwann_ini.
- Set $\mu = 0$.
- Run ComWann, using check_wannier_function_input, run_comwann, and comwann_postprocessing. ComWann extracts information from the DFT calculation and puts that information in wannier.dat which will be used by comdmft.py and ComLowH, and wannier.inip which is used only by ComLowH. The ini file for ComWann is created by flapwmbpt_ini.read_comdmft_ini_wan() .
- Initialize T in trans_basis.dat, using generate_initial_transformation. The default value is the identity. If the 'trans_basis_mode' variable in comdmft.ini is different from zero then T is set up differently. The default value is zero.
- If dc_mat.dat has been supplied by the user, leave that as is, as it contains our permanent value of Σ_{DC}.
- If dc_mat.dat has not been supplied, create $\Sigma_{DC}(i)$, using cal_nominal_dc and U and J, and store it in dc_mat.dat. dc_mat.dat holds matrices of size equal to the number of impurity orbitals, but they are proportional to the identity. The following equations give the values of $\Sigma_{DC}(i)$ and of N which is the size of the self-energy matrix. $\Sigma_{DC}(i)$ is permanent; it remains unchanged throughout the whole calculation.
  - The initialization here sets $\Sigma_{DC}(i)$ to a value that does not depend on spin.
  - 's' orbital: $N = 1,\ U = F(0,i),\ J = 0$
  - 'p' orbital: $N = 3,\ U = F(2,i),\ J = F(2,i)/5$
  - 'd' orbital: $N = 5,\ U = F(0,i),\ J = (F(2,i) + F(4,i))/14$
  - 'f' orbital without spin-orbit: $N = 7,\ U = F(0,i),\ J = (F(2,i) + F(4,i) + F(6,i))/(6435.0/(286 + 195 * 0.668 + 250 * 0.494) \times (1.0 + 0.668 + 0.494))$
  - 'f' orbital with spin-orbit: same as without spin-orbit, except $N = 14$.
  - $\Sigma_{DC}(i) = U(\eta(i) - 1/2) - J(\eta(i) - 1)/2$, where $\eta(i)$ is a user-supplied 'nominal_n' parameter, i.e. a user-supplied occupancy that controls the double-counting energy.
- Initialize T in trans_basis.dat, using generate_initial_transformation. The default value is the identity. If the 'trans_basis_mode' variable in comdmft.ini is different from zero then T is set up differently.
- cal_dc_diagonal copies $\Sigma_{DC}(i)$ from dc_mat.dat to dc.dat. This changes the format from a matrix to a one-dimensional array containing only one element for each non-equivalent orbital.
- If 'initial_self_energy' is in comdmft.ini and sig.dat is available, then use this file as the initial value of the self-energy $\Sigma(i)$.

- If sig.dat is not available then generate_initial_self_energy initializes Σ and stores it in sig.dat. sig.dat contains only one entry for each non-equivalent impurity orbital. $\Sigma(i,\omega) = \Sigma_{DC} + \omega$, if the impurity's 'para' variable is true.
- If the 'para' variable is false, then $\Sigma(i,\omega,s) = \Sigma_{DC}(i,s) + \omega$, plus a small real number, -0.001 multiplied by a sign associated with the impurity.

C. Initialization for the inner self-consistent loop

- lda_dmft runs FlapwMBPT to do DFT, using the density matrix $\hat{\rho}$ created by ComLowH. This density is used to create the charge and spin densities inside of FlapwMBPT. For details of how ComLowH calculates $\hat{\rho}$ and of how FlapwMBPT uses it, see the subsection on "How the Density Matrix Is Produced".
- todocomlowh describe how wannier_den_matrix.dat is used inside of FlapwMBPT.
- Run ComWann, using check_wannier_function_input, run_comwann, and comwann_postprocessing. This extracts information from the DFT calculation and puts that information wannier.dat which will be used by comdmft.py and ComLowH, and wannier.inip which is used only by ComLowH. todo and document ComWann.

D. Inner Loop

- Run ComLowh, which uses the DFT eigenvalues λ_{DFT}, the Wannier90 projectors V_{w90}, Σ and Σ_{DC}, and various other variables. It recalculates the Fermi level E_F, and produces the the frequency-independent single-particle Hamiltonian $\hat{E}_{imp}$, the "hybridization" frequency-dependent single-particle Hamiltonian $\hat{\Delta}_{imp}$, and the local Green's function $\hat{G}_{imp}$. It also writes otu several other observables: local_spectral_matrix_ef.dat, n_loc.dat, and spectral_matrix_ef.dat.
- If the 'trans_basis_mode' variable in comdmft.ini is equal to two then do some work on trans_basis.dat, i.e. on T. Otherwise do nothing. The default is to do nothing.
- Copy $\hat{E}_{imp}(i)$ from e_projected_mat.dat (if it exists) to projected_eig.dat. This changes the format from a matrix to a one-dimensional array. e_projected_mat.dat contains matrices. projected_eig.dat contains only the same number of elements as are marked as distinct impurity orbitals.
- Copy $\hat{\Delta}_{imp}(i,\omega)$ from dc_mat.dat (if it exists) to dc.dat. dc_mat.dat stores matrices of size equal to the number of impurity orbitals. dc.dat contains a vector with elements only for non-equivalent orbitals within each matrix.
- Don't copy data from zinv_m1_mat.dat to zinv_m1.dat, because zinv_m1_mat.dat exists only in qsgw+dmft calculations, not in dft+dmft calculations.
- Read in $\hat{E}_{imp}(i)$ from projected_eig.dat and $\Sigma_{DC}(i)$ from dc.dat, and write $\hat{E}_{imp}(i) - \Sigma_{DC}(i)$ to e_imp.dat. (If the 'embed_mode' option in comdmft.ini is set to 'hfc', then the contents of hartree.dat is also subtracted.) These are pure reads and writes, with no processing other than the subtraction. e_imp.dat contains a vector with elements only for non-equivalent orbitals within each matrix.
- Copy $\hat{\Delta}_{imp}(i,\omega)$ from delta_mat.dat (if it exists) to delta.dat. delta_mat.dat stores matrices of size equal to the number of impurity orbitals. delta.dat contains a vector with elements only for non-equivalent orbitals within each matrix. If all orbitals are marked as equivalent by the 'impurity_matrix' entry in comdmft.ini, then delta.dat contains a single complex number for each frequency.
- Read in $\hat{\Delta}_{imp}(i,\omega)$ from delta.dat. Determine whether causality is broken, i.e. if if the imaginary part of any element of $\hat{\Delta}_{imp}$ is positive. If causality is broken, print an error message.
- Read in $\hat{\Delta}_{imp}(i,\omega)$ from delta_mat.dat and then write it out to hyb.json file, along with the inverse temperature β. If the impurity's 'para' member is false, the format of what is written out to hyb.json is a bit different.
- Read in $\hat{E}_{imp}(i) - \Sigma_{DC}(i)$ from e_imp.dat to e_imp. e_imp.dat contains a vector with elements only for non-equivalent orbitals within each matrix. In contrast, e_imp is a matrix of size equal to the number of impurity orbitals, so some reconstruction is involved, and the reconstruction is not lossless unless the original matrix has a very restricted form.

- Read in T from trans_basis.dat to trans_basis. This involves using the 'impurity_wan' control parameter to figure out the basis size of T.
- Transfer $\hat{E}_{imp}(i) - \Sigma_{DC}(i)$ from e_imp to e_imp_key, T from trans_basis to trans_key, and imp[key]['impurity_matrix'] to equivalence_key. This involves:
  - Keeping track of the impurity equivalence, based on 'impurity_matrix' which specifies which impurities are equivalent to each other.
  - If this calculation is not spin-orbit, then double the basis size.
    - If 'para' is true, e_imp goes both in the spin up-up block and also in the spin down-down block of e_imp_key. trans_key is the same. equivalence_key is also the same except for a following line of code involving lists, maps, and lambda functions, that I don't understand.
    - If 'para' is false, for e_imp_key and trans_key: The spin up-up block is the same, but the spin down-down block comes from the impurity which is related by AFM symmetry.
    - If 'para' is false, for equivalence_key: The spin up-up block is the same, but the spin down-down block is a modified version of the 'impurity_matrix' variable. I don't understand the modification, but it is simple. Just as with the 'para' = true case, there is a following line of code involving lists, maps, and lambda functions, that I don't understand.
    - There is also some logic that I don't understand for figuring out what goes in the up-up and down-down blocks of equivalence_key.
    - If this is a spin-orbit calculation, there is no basis doubling and everything is much simpler. Just as when not doing spin-orbit, for equivalence_key is modified by a last line of code involving lists, maps, and lambda functions, that I don't understand.
- Write params.json:
  - ["mu"] = -e_imp_key$[0, 0]$
  - ["hloc"] ["one body"] = e_imp_key - e_imp_key[0,0]; the [0,0] element is null. e_imp_key = a massaged $\hat{E}_{imp}(i) - \Sigma_{DC}(i)$.
  - ["hybridisation"] ["matrix"] = equivalence_key = a massaged 'impurity_matrix'. This is the matrix structure for the hybridization term; it multiplies hyb.json.
  - ["hybridisation"] ["functions"] = "hyb.json"
  - ["beta"] = 'beta'
  - ["hloc"] ["two body"] ["parametrisation"] = "slater-condon"
  - ["hloc"] ["two body"] ["F0"] = 'f0', and similar for F2, F4, F6.
  - ["basis"] ["transformation"] = trans_key = a massaged T
  - ["basis"] ["orbitals"] = 'problem' , i.e. a list of shells
- params.json also contains some comdmft.ini parameters that are aimed at controlling the algorithm implemented by the CTQMC solver:
  - If 'spin_orbit' is true, then ["basis"] ["type"] = "coupled" and the observables include "J2" and "Jz". Otherwise it is "product" and the observables include "S2" and "Sz".
  - If 'coulomb' is 'full', then ["hloc"] ["two body"] ["approximation"] = "none". If 'coulomb' is 'ising', then it is 'ising'.
  - ["measurement time"] = 'measurement_time'
  - ["thermalisation time"] = 'thermalization_time'
  - ['partition']["green matsubara cutoff"] = 'green_cutoff'
  - ['partition']["susceptibility cutoff"] = 'susceptibility_cutoff'
  - ['partition']["susceptibility tail"] = 'susceptibility_tail'
  - Also there are a few hard-coded params.json entries that do not depend on comdmft.ini at all.
- Don't write dyn.json because this is not a qsGW+CTQMC calculation.

- Run ComCTQMC, both CTQMC and EVALSIM.

  Afterwards the results of ComCTQMC are processed and written to files. The results of ComCTQMC have a number of entries equal to the number of nonequivalent impurity orbitals. The output files, g_loc.dat, sig.dat, sig_smth.dat, and sig_bare.dat, all contain a number of entries equal to the number of nonequivalent impurity orbitals.

  – Copy g_{loc} from the "green" section of the params.obs.json to g_loc.dat. This is used only by ComDC, which is used only if doing a qsGW+DMFT calculation. It is not the same as $\hat{G}_{loc}(i,\omega)$, which is computed by ComLowH.

  – Read $\Sigma_{CTQMC}(i,\omega)$ in from the "self-energy" section of params.obs.json. This is immediately saved in sig_bare.dat, but that file is never used.

  – Read $\Sigma_{old}(i,\omega)$ in from sig.dat.

  – Obtain $\Sigma_{smeared}(i,\omega)$ by performing Gaussian broadening on $\Sigma_{CTQMC}(i,\omega)$, matrix element by matrix element. Most values of ω are smeared, but if ω is too close to its minimum or maximum values within the ω array then no smearing is done and $\Sigma_{smeared}(i,\omega) = \Sigma_{CTQMC}(i,\omega)$.

  $$\begin{aligned} \lambda(\omega_0) &= (0.85\pi T \times 8.6173303 \times 10^{-5}) + 0.05\omega_0 \\ \Sigma_{smeared}(i,\omega_0) &= \frac{\sum_\omega \Sigma_{CTQMC}(i,\omega) \times \exp(-(\omega-\omega_0)^2/2\lambda^2(\omega_0))}{\sum_\omega \exp(-(\omega-\omega_0)^2/2\lambda^2(\omega_0))} \end{aligned} \tag{1}$$

  The sums in this smearing calculation are over all elements in the ω array.
  Smearing is not done if (a) ω_0 less than $3\lambda(\omega_0)$, (b) ω_0 is within $3\lambda(\omega_0)$ of the maximum value of the ω array, or (c) $\omega_0 > \Gamma + 3\lambda(\Gamma)$, where Γ is the 'green_cutoff' parameter in comdmft.ini. If smearing is not done for a particular value of ω_0, then $\Sigma_{smeared}(i,\omega_0) = \Sigma_{CTQMC}(i,\omega_0)$.

  – $\Sigma_{smeared}(i,\omega)$ is saved to sig_smth.dat, which is never used.

  – After a particular matrix element of $\Sigma_{smeared}(i,\omega)$ is calculated, it is tested for causality. The causality requirement is that, for all values of ω, $Im(\Sigma_{smeared}(i,\omega)) \leq 0$.

    ∗ If this causality requirement is met, then this particular matrix element is mixed with the old self energy to obtain the new self-energy: $\Sigma(i,\omega) = \alpha\Sigma_{smeared}(i,\omega) + (1-\alpha)\Sigma_{old}(i,\omega)$, where α is the 'sigma_mix_ratio' entry in comdmft.ini.

    ∗ If causality is not met, then this particular matrix element is set to the old value: $\Sigma(i,\omega) = \Sigma_{old}(i,\omega)$.

    ∗ If the 'para' variable is false, i.e. if the present impurity is related to another impurity by antimagnetic symmetry, then $\Sigma_{old}(j,\omega)$ is used (with j being the other impurity) instead of $\Sigma_{old}(i,\omega)$.

    ∗ It can happen that some matrix elements of $\Sigma(i,\omega)$ come from Σ_{old}, while others come from $\Sigma_{smeared}$ with subsequent mixing.

  – $\Sigma(i,\omega)$ is saved to sig.dat.

II. COMLOWH

Inputs:

- i is an index over impurities.
- o is an index over orbitals at a specific impurity.
- q is an index over momenta, the same momenta used by CASTEP.
- k is an index over momenta on a fine mesh, after interpolation using Wannier functions.
- s is an index over spins.
- b is an index over bands.
- $\beta = (8.6173303 \times 10^{-5} \times T)$ is the inverse temperature.
- The number of bands N_{bands}, from wannier.dat.

- The number of wanniers N_{wann}, from wannier.dat. This implicitly determines the projection operators $P_w+P_r = 1$.
- The number of atoms/impurities N_a, from comlowh.ini.
- The number of correlated orbitals on each atom/impurities, $N_{orb}(i)$, from comlowh.ini.
- The number of atoms (impurities) that are equivalent, $N_{equiv}(i)$, from comlowh.ini.
- $[E_{min}, E_{max}]$, from comlowh.ini. This interval, and also the interpolated eigenvalues on the fine k-point mesh, are used to define the P_E projection operator.
- wan_correlated, from comlowh.ini. This is a list of which wannier orbitals are correlated, and that is used to defined the P_c projection operator.
- include_bands, from wannier.dat. include_bands_new in wannier_den_matrix.dat is decided by multiplying include_bands by P_E, and permuting bands according to U_{tob}.
- The coarse q-point momentum grid, from wannier.dat.
- The spatial R-point spatial grid, from wannier.dat.
- The fine k-point momentum grid, from wannier.dat.
- A path κ through momentum space for plots of spectral density and spectra; this is read in from a user-supplied file named kpoints. If not doing a plot through momentum space, ComLowH writes out kpoints instead of reading it in; in this case the kpoints written out are a grid, not a path.
- n_omega = N_ω is the number of frequencies on the frequency grid. If the user supplies a self-energy file then N_ω is read from that file. Otherwise it is set to $N_\omega = 150 \times \beta/\pi$.
- ω is a grid of real numbers: $\pi/\beta, 3\pi/\beta, 5\pi/\beta, ..., (2N_\omega - 1)/\beta$.
  - comdmft.py writes ω to sig.dat, gimp.dat, sig_bare.dat, sig_smth.dat, delta.dat.
  - ComLowH receives the omega grid in sig.dat. When called during self-consistent DFT+DMFT or qsGW+DMFT iterations it multiplies the grid by $\imath$. It does the same when calculating qsGW spectra. When calculating other observables it leaves the grid at its real value, apparently without any imaginary part.
- The is_spinorbit variable in ComLowH is read in from comlowh.ini, and originates in the comdmft.ini file. It controls how wannier.inip is read in. The is_spinorbit variable also doubles the electron count when calculating the Fermi level. And, if true, it ensures that the number of bands determined by find_bnd_in_win (inside the energy window) is odd.
- The 'doping' parameter in comdmft.ini is passed to ComLowH, where it becomes the delta_charge variable. It is added to the computed charge during the search for the Fermi energy μ.
- $\lambda_{DFT}(q, b, s)$ are the CASTEP DFT eigenvalues, and are stored in wannier.dat.
- $V_{w90}(q, b, o)$ is the projectors created by Wannier 90. Stored in wannier.dat.
- T is a user-supplied transformation matrix that defaults to the identity if if is not supplied. Stored in trans_basis.dat
- $\Sigma_{DC}(i)$ is stored in dc_mat.dat and dc.dat. Keep in mind thatdc.dat contains only a few matrix elements from the full matrix stored in dc_mat.dat, so ComLowH has to reconstruct the matrix with necessarily less information. To see how this goes, see ComLowH's use of the ioac matrix, which contains entries from the 'impurity_problem_equivalence' variable in comdmft.ini. It is produced by comdmft.py, and is an input to ComLowH. In a qsGW+DMFT calculation it is produced by ComDC.
- $\Sigma(i, \omega)$ is stored in sig.dat. Keep in mind that sig.dat contains only a few matrix elements from the full Σ matrix, so ComLowH has to reconstruct the matrix with necessarily less information. To see how this goes, see ComLowH's use of the ioac matrix, which contains entries from the 'impurity_problem_equivalence' variable in comdmft.ini. It is initialized by comdmft.py, and mixed and smoothed and written again comdmft.py using data from ComCTQMC. It is used by ComLowH.

- μ, the Fermi energy, is stored in ef.dat. It is initialized by comdmft.py and re-calculated by ComLowH. It is used only by ComLowH. This is ComLowH is called in a self-consistent calculation; otherwise ComLowH reads in μ from ef.dat.
- $\hat{E}_{imp}(i)$ is stored in e_projected_mat.dat. It is a projected hamiltonian. It is produced by ComLowH, processed by comdmft.py, and used by ComCTQMC.
- $\hat{\Delta}_{imp}(i, \omega)$ is stored in delta_mat.dat. It is produced by ComLowH, processed by comdmft.py, and used by ComCTQMC.
- The density matrix $\hat{\rho}$ is stored in wannier_den_matrix.dat. It is produced by ComLowH, and used by FlapwMBPT.
- $\hat{G}_{loc}(i, \omega)$ is stored in g_loc_mat.dat and g_loc.dat. It is a projected Green's function. It is produced by ComLowH. It is used only by ComDC, and only if doing qsGW+DMFT.
- ComLowH produces local_spectral_matrix_ef.dat. This is used only if the 'trans_basis_mode' control parameter is equal to 2, and then is used only to determine trans_basis.dat. The default value of the 'trans_basis_mode' control parameter is 0.
- ComLowH produces n_loc.dat, which is never used elsewhere.
- $z_{zinvm1}(i)$ comes from zinv_m1.dat, and is used only when doing qsGW+DMFT calculations. Keep in mind that zinv_m1.dat contains only a few matrix elements from the full matrix stored in zinv_m1_mat.dat, so ComLowH has to reconstruct the matrix with necessarily less information. To see how this goes, see ComLowH's use of the ioac matrix, which contains entries from the 'impurity_problem_equivalence' variable in comdmft.ini. This is the origin of zinv_m1.dat: run_dc runs ComDC, and then takes data from ComDC's sig_mat.dat and puts it zinv_m1_mat. After ComLowH is run, delta_postprocessing/cal_zinv_m1_diagonal takes takes info from zinv_m1_mat.dat and puts it in zinv_m1.dat. If zinv_m1.dat is missing, ComLowH effectively sets it equal to 1.

- The $P_c(i, o, o)$ projection operator. Off-diagonals are zero. P_c selects from the wannier orbitals only the ones which are marked as correlated. comdmft.ini tracks the list of correlated orbitals in control['impurity_wan'][iatom] and writes it to comlowh.ini. ComLowH reads this list into wan_correlated, and its dmft_projector routine uses that list to apply P_c.
- The P_E projection operator projects out bands that are inside of a certain energy interval. The correspondence between the energy interval and the bands selected for projection is a little complex, and is decided by the find_bnd_in_win routine in ComLowH. That routine produces nbnd_k and ind_bnd_k which encode the projection operator. These variables have indexes for momentum but do not actually depend on momentum. The energy interval is controlled by the proj_win_min and proj_win_max user control parameters in the comdmft.ini file, which are transferred to the ewin_min and ewin_max variables in ComLowH.

Variables read in from wannier.dat, generated by Wannier90 or ComWann:

- The coarse q-point grid.
- The spatial R-point grid.
- The number of bands N_{bands}.
- The number of wanniers N_{wann}. This implicitly determines the projection operators $P_w + P_r = 1$.
- include_bands. include_bands_new in wannier_den_matrix.dat is decided by multiplying include_bands by P_E, and permuting bands according to U_{tob}.
- $\lambda_{DFT}(q, i)$, the CASTEP band energies.
- $V_{w90}(q, i, \beta)$, the projection operators generated by Wannier90.

Variables read in from wannier.inip, generated by ComWann:

- wannier.inip contains identifying info about each wannier orbital, including angular momentum numbers and impurity numbr and orientation axis. These are all stored in variables beginning with alimj. These variables are not used for anything except the optics code, which is currently disabled.

Variables read in from comlowh.ini:

- cal_mode. cal_mode=1 means do ComLowH's work for a self-consistent iteration. cal_mode=2 means calculate the density of states and partial density of states. cal_mode=3 means calculate the spectral density on a path through momentum space, and the path through momentum space is read in from the file named kpoints. cal_mode=5 means produce qsGW spectra
- Γ, an imaginary part that is added to the real self-energy when doing cal_mode=2,3 calculations. comdmft.py sets this to zero; it is always zero.
- β
- The number of atoms/impurities N_a.
- The number of correlated orbitals on each atom/impurities, $N_{orb}(i)$.
- The number of atoms (impurities) that are equivalent, $N_{equiv}(i)$.
- The fine k-point grid.
- $[E_{min},\, E_{max}]$. This interval, and also the interpolated eigenvalues on the fine k-point mesh, are used to define the P_E projection operator.
- wan_correlated, a list of which wannier orbitals are correlated, and that is used to defined the P_c projection operator.

III. COMLOWH CALLED DURING SELF-CONSISTENT DFT+DMFT OR QSGW+DMFT ITERATIONS; CAL_MODE = 1

Bear in mind that throughout this documentation of ComLowH I set the basis transformation matrix T to the identity.

Also keep in mind that for cal_mode = 1 the frequencies used in ComLowH are imaginary. They are obtained by multiplying the frequency grid supplied in sig.dat by $\imath$.

$$
\begin{aligned}
H_{w90}(q) &= V^{\dagger}_{w90}(q)\lambda_{DFT}(q)V_{w90}(q) && (2)\\
H_{w90}(k) &= N_q^{-1}\sum_{R,q}\exp(\imath R\cdot(k-q))H_{w90}(q) && (3)\\
H_{w90}(k) &= U^{\dagger}_{w90}(k)\lambda_{w90}(k)U_{w90}(k) && (4)\\
P_{pr}(k,i) &= \frac{1}{\sqrt{\chi\chi^{\dagger}}}\chi,\quad \chi(i,k) = \; P_c(i)U_{w90}(k)\; P_E(k) && (5)\\
G_l^{-1}(\omega,k) &= E_F+\omega-\lambda_{w90}(k)-\sum_i P^{\dagger}_{pr}(k,i)(\Sigma(\omega,i)-\Sigma_{DC}(i)-(\imath\Gamma=0))P_{pr}(i,k) && (6)
\end{aligned}
$$

At this point, if cal_mode = 1, the Fermi energy E_F is recomputed and then written to ef.dat. For other values of cal_mode it is not recomputed and instead read in from ef.dat.

E_F is adjusted to find a solution of the following equation (keep firmly in mind that ω is an imaginary Matsubara frequency):

$$\begin{aligned}
N_{elec} &= N_k^{-1}\sum_k Tr(n_{imp}(k,E_F)) \\
n_{imp}(k,E_F) &= -2\beta^{-1}\sum_\omega \exp(-\beta\omega)\ (G_l(k,\omega,E_F)-\omega^{-1}-g_2(k,\omega_{max})/\omega^2-g_3(k,\omega_{max})/\omega^3) \\
&+ (1/2-g_2(k,\omega_{max})\beta/4) \\
g_2(k,\omega_{max}) &= \omega_{max}^2(G_l(k,\omega_{max})+G_l^\dagger(k,\omega_{max}))/2 \\
g_3(k,\omega_{max}) &= \omega_{max}^3(G_l(k,\omega_{max})-G_l^\dagger(k,\omega_{max})-2/\omega_{max})/2 \\
N_{elec} &= N_d+\delta/2, if\, no\, spin-orbit \\
N_{elec} &= N_d+\delta, if\, spin-orbit \\
N_f &= N_q^{-1}\sum_{q,b,o,\lambda_{DFT}(q,b)<0}|V_{w90}(q,b,o)|^2
\end{aligned}$$

If no spin-orbit then N_f is rounded to integer or half-integer values and this is the value of N_d. If spin-orbit then N_f is rounded to integer values but not to half-integer values, and this is the value of N_d. δ is the 'doping' user control parameter supplied in comdmft.ini. ω_{max} is the maximum Matsubara frequency in our frequency grid.

After choosing E_F, ComLowH goes on with computing its outputs:

$$\begin{aligned}
g(\omega,i) &= N_k^{-1}\sum_k P_{pr}(i,k)\ \ G_l(\omega,k)\ \ P_{pr}^\dagger(i,k) && (7)\\
\hat{E}_{imp}(i) &= N_k^{-1}(N_{equiv}(i))^{-1}\sum_k P_{pr}(i,k)\lambda_{w90}(k)P_{pr}^\dagger(i,k)-E_F && (8)\\
\hat{\Delta}_{imp}(\omega,i) &= (N_{equiv}(i))^{-1}(\omega-\hat{E}_{imp}(i)-\Sigma(\omega,i)+\Sigma_{DC}(i))-(N_{equiv}(i))^{-1}g(\omega,i)^{-1} && (9)\\
\hat{\Delta}_{imp}(\omega,i) &= (N_{equiv}(i))^{-1}(E_F+\omega-N_k^{-1}(N_{equiv}(i))^{-1}\sum_k P_{pr}(i,k)\lambda_{w90}(k)P_{pr}^\dagger(i,k)-\Sigma(\omega,i)+\Sigma_{DC}(i)) \\
&- (N_{equiv}(i))^{-1}g(\omega,i)^{-1} && (10)\\
\hat{G}_{loc}(\omega,i) &= (N_{equiv}(i))^{-1}\ g(\omega,i) && (11)\\
g_{diag}(\omega,k,o) &= diag(U_{w90}(k)\ \ G_l(\omega,k)\ \ U_{w90}^\dagger(k)) && (12)
\end{aligned}$$

$\hat{E}_{imp}$ is written to e_projected_mat.dat, $\hat{G}_{loc}$ is written to g_loc_mat.dat, $\hat{\Delta}_{imp}$ is written to delta_mat.dat, and $g_{diag}(\omega,k)$ is written to spectral_matrix_ef.dat.

$$\begin{aligned}
spectral(i) &= \beta^{-1}\sum_\omega \exp(-\beta\omega/2)\ (G_{loc}(i,\omega)-\omega^{-1}-g_2(i,\omega_{max})/\omega^2-g_3(i,\omega_{max})/\omega^3) \\
&+ \beta^{-1}\sum_\omega \exp(-\beta\omega/2)\ (G_{loc}^\dagger(i,\omega)+\omega^{-1}-g_2(i,\omega_{max})/\omega^2-g_3(i,\omega_{max})/\omega^3) \\
&+ (1/2+g_3(i,\omega_{max})\beta^2/16) \\
n_{loc}(i) &= -\beta^{-1}\sum_\omega \exp(-\beta\omega)\ (G_{loc}(i,\omega)-\omega^{-1}-g_2(i,\omega_{max})/\omega^2-g_3(i,\omega_{max})/\omega^3) \\
&- \beta^{-1}\sum_\omega \exp(-\beta\omega)\ (G_{loc}^\dagger(i,\omega)+\omega^{-1}-g_2(i,\omega_{max})/\omega^2+g_3(i,\omega_{max})/\omega^3) \\
&+ (1/2-g_2(i,\omega_{max})\beta/4) \\
g_2(i,\omega_{max}) &= \omega_{max}^2(G_{loc}(i,\omega_{max})+G_{loc}^\dagger(i,\omega_{max}))/2 \\
g_3(i,\omega_{max}) &= \omega_{max}^3(G_{loc}(i,\omega_{max})-G_{loc}^\dagger(i,\omega_{max})-2/\omega_{max})/2
\end{aligned}$$

$-\beta\pi^{-1}\times Re(spectral(i))$ is written to local_spectral_matrix_ef.dat, and $n_{loc}(i)$ is written to n_loc.dat. Under certain circumstances local_spectral_matrix_ef.dat used by comdmft.py to create a transformation matrix T. $n_{loc}(i)$ is never used by comsuite.

Lastly spectral_matrix_ef.dat is calculated:

$$\begin{aligned}
spectral_{lat}(o) &= \beta^{-1}\sum_{\omega}\exp(-\beta\omega/2)\ (g_{lat}(o,\omega) - \omega^{-1} - g_2(o,\omega_{max})/\omega^2 - g_3(o,\omega_{max})/\omega^3) \\
&+ \beta^{-1}\sum_{\omega}\exp(\beta\omega/2)\ (g^*_{lat}(o,\omega) + \omega^{-1} - g_2(o,\omega_{max})/\omega^2 + g_3(o,\omega_{max})/\omega^3) \\
&+ (-1/2 + g_3(o,\omega_{max})\beta^2/16) \\
g_2(o,\omega_{max}) &= \omega^2_{max} Re(g_{lat}(o,\omega_{max})) \\
g_3(o,\omega_{max}) &= \omega^3_{max}(Im(g_{lat}(o,\omega_{max})) - 1/\omega_{max}) \\
g_{lat}(o,\omega_{max}) &= diag(N_k^{-1}\sum_k P_{pr}(i,k)\ \ G_l(\omega,k)\ \ P^\dagger_{pr}(i,k))
\end{aligned} \tag{13}$$

$-\beta\pi^{-1}\times Re(spectral_{lat}(i))$ is written to spectral_matrix_ef.dat, which is never used by comsuite.

In qsGW+DMFT calculations where Z is available, $H_{ni}(k)$ replaces $\lambda_{w90}(k)$ in the equations for $\hat{E}_{imp}(i)$, $\hat{\Delta}_{imp}(\omega,i)$, and $G_l^{-1}(\omega,k)$:

$$\begin{aligned}
Z^{-1}(k) &= 1 + \sum_i P^\dagger_{pr}(k,i) z_{zinvm1}(i) P_{pr}(k,i) \\
H_{ni}(k) &= Z^{1/2}(k)\lambda_{w90}(k)Z^{1/2}(k)
\end{aligned} \tag{14}$$

A. How the Density Matrix is Produced

Bear in mind that throughout this documentation of ComLowH I set the basis transformation matrix T to the identity. Also bear in mind that while calculating the density matrix the ω's are all imaginary Matsubara frequencies.

$$\begin{aligned}
n_{imp}(q) &= -2\beta^{-1}\sum_{\omega} Re[\exp(-\beta\omega)]\ (G_l(q,\omega,E_F) - \omega^{-1} - g_2(q,\omega_{max})/\omega^2 - g_3(q,\omega_{max})/\omega^3) \\
&+ (1/2 - Re[g_2(q,\omega_{max})]\beta/4) \\
g_2(q,\omega_{max}) &= \omega^2_{max}(G_l(q,\omega_{max}) + G_l^\dagger(q,\omega_{max}))/2 \\
g_3(q,\omega_{max}) &= \omega^3_{max}(G_l(q,\omega_{max}) - G_l^\dagger(q,\omega_{max}) - 2/\omega_{max})/2
\end{aligned}$$

ω_{max} is the maximum Matsubara frequency in our frequency grid. $n_{imp}(q,E_F)$ is evaluated on the coarse k-point grid that is used by flapwmbpt, which is different from the $n_{imp}(k,E_F)$ used to find the Fermi level, which is evaluated on a fine k-point grid. Also, the z_{zinvm1} factor is ignored here while calculating $G_l(k,\omega,E_F)$. I'm not sure why this is so.

The density matrix that is returned to flapwmbpt is modified as follows:

$$\begin{aligned}
n_{mat}(q,E_F) &= P_E(P_w n_{imp}(q)P_w + P_r f_{FD}(\beta(\lambda_r(q) - E_F))P_r)P_E \\
n_{mat}(q,E_F) &= P_E n_{imp}(q) P_E,\ if\ \ P_w = 1,\ P_r = 0
\end{aligned} \tag{15}$$

where $f_{FD}(E) = (1+\exp(E))^{-1}$ is the Fermi-Dirac distribution. In a basis of size equal to the number of DFT bands, where the wannier functions are diagonal, P_w is the operator that projects out the wannier functions, and $P_r = 1 - P_w$ is the operator that projects out everything else. $\lambda_r(q)$ is the eigenvalues of the non-wannier bands after they have been modified by disentangling them from the wannier functions.

The point of this modification is to include contributions to the density matrix which come from outside the space spanned by the Wannier orbitals $U_{w90}(k)$, i.e. from the P_r sector.

It remains to calculate $\lambda_r(q)$, which are the eigenvalues of the non-wannier bands after they have been modified by disentangling them from the wannier functions. We simultaneously calculate $v_{wmc}(q)$, which in its P_r sector contains the eigenvectors which are the counterparts of $\lambda_r(q)$. In the P_w sector it contains the wannier functions' representations in the flapwmbpt band basis.

$$\begin{aligned}
v_{wmc}(q) &= V_{w90}(q)U_{w90}(q)P_w + P_V(q)P_r U_r(q)P_r,\ \ 1 = v^\dagger_{wmc}v_{wmc} \\
v_{wmc}(q) &= V_{w90}(q)U_{w90}(q),\ \ if\ P_w = 1,\ P_r = 0
\end{aligned}$$

$V_{w90}(q)$ is a matrix from Wannier90, found in wannier.dat, which contains the mapping from wanniers to the flapwmbpt band basis. $P_V(q)$ is the complement to $V_{w90}(q)$; $P_V(q)$ and $V_{w90}(q)$ are orthogonal to each other, and $P_V(q)$ spans the space that remains after subtracting the wanniers from the bands basis. $P_V(q)$ is constructed by diagonalization:

$$P_V^\dagger(q)\lambda_V P_V(q) \;=\; 1 - V_{w90}(q)V_{w90}^\dagger(q),\;\; P_V^\dagger V = 0 \tag{16}$$

Next, we construct and diagonalize a Hamiltonian within the P_r sector:

$$H_r(q) \;=\; P_r P_V^\dagger(q)\lambda_{DFT}(q)P_V(q)P_r \tag{17}$$
$$U_r^\dagger(q)\lambda_r(q)U_r(q) \;=\; H_r(q) \tag{18}$$

This gives us the eigenvalues $\lambda_r(q)$ and the eigenvectors $U_r(q)$ of the non-wannier bands after they have been modified by disentangling them from the wannier functions.

Lastly, we try to compensate for that fact that our the disentangled eigenvectors in $v_{wmc}(q)$ may no longer match the character of the wannier orbitals that we started with. Therefore we construct a permutation matrix $U_{tob}(q)$ which permutes $v_{wmc}(q)$ (equal to $V_{w90}(q)U_{w90}(q)$ if $P_w = 1,\ P_r = 0$) to give a closer match with the original wanniers. This best match is estimated using the following heuristic: the mapping from orbital i to j more or less maximizes the value of $|v_{wmc}(q,i,j)|^2$.

The following outputs are written to wannier_den_matrix.dat:

- temperature T
- nqdiv, the number of points in the coarse momentum grid
- n_dmft_bnd, the number of bands inside the energy window P_E, and also the size of the first two indices of $v_{wmn}(q)$.
- n_dmft_wan, the number of bands inside the energy window P_E, and also the size of the first two indices of $v_{wmn}(q)$.
- include_band_new, a copy of the include_bands array in wannier.dat.
- $v_{wmn}(q) = p(p\ \ p^\dagger)^{-1/2},\ p = U_{tob}(q)P_E v_{wmc}(q,i,j)P_E$. This is the disentangled eigenvectors after orthonormalization. If $P_w = 1,\ P_r = 0$ then $p = U_{tob}(q)P_E V_{w90}(q)U_{w90}(q)P_E$.
- size, a three-element vector containing the size of v_{wmn}.
- n_{mat}

FlapwMBPT reads in wannier_den_matrix.dat and uses it to alter the occupation matrix that is used to evaluate the charge and spin density, which are used for the next DFT iteration. The interstitial sector of the charge density is as follows. The muffin-tin sector is evaluated using considerably more algebra, but still starting with the same $\mathcal{G}_0(q)$. The updated $\mathcal{G}_0(q)$ is also used to evaluate the Fock term, Green's functions, orbital moments, etc.

$$\begin{aligned}
\rho_{int} &= N_q^{-1}\sum_q\sum_n \langle\psi_{int}(n,q)|\mathcal{G}_0(q)|\psi_{int}(n,q)\rangle \\
\mathcal{G}_0(q) &= -g_0(q) + \mathcal{P}(q)n_{mat}(q)\mathcal{P}^\dagger(q) - \mathcal{P}(q)\left(\mathcal{P}^\dagger(q)\left(-g_0(q)\right)\mathcal{P}(q)\right)\mathcal{P}^\dagger(q) \\
\mathcal{P}(q) &= \mathcal{U}(q)\times v_{wmn}(q)
\end{aligned}$$

- $|\psi(n,q)\rangle$ is the n-th eigenstate of the DFT Hamiltonian, and $\psi_{int}(n,q)$ is the interstitial sector of those eigenstates.
- $$\begin{aligned}
g_0(q) &= -\frac{\exp((\beta-\tau)(\lambda_{DFT}(q)-E_F))}{1+\exp(\beta(\lambda_{DFT}(q)-E_F))},\;\; \lambda_{DFT}(q)-E_F > 0 \\
g_0(q) &= -\exp(\tau(\lambda_{DFT}(q)-E_F))(1-\frac{\exp(-\beta(\lambda_{DFT}(q)-E_F))}{1+\exp(-\beta(\lambda_{DFT}(q)-E_F))}),\;\; \lambda_{DFT}(q)-E_F < 0
\end{aligned}$$
- After ρ_{int} is computed as shown above, it symmetrized by ro_int_g's call to symscal or symvec, which do sums over all the crystal group symmetrization operations.
- $\mathcal{U}(q)$ is a unitary matrix with basis size equal to the number of bands. $\mathcal{U}(q)$ is prepared by dft_basis_rotmat_k, and then force_mat_unitary uses a singular matrix decomposition to force it to be unitary. I am not sure what dft_basis_rotmat_k does, but it seems to take the overlap between bands.

IV. POSTPROCESSING

Postprocessing is done entirely by ComLowH. The only role of comdmft.ini is to construct the comlowh.ini file and execute ComLowH. The choice of which postprocessing to do is encode in the method user control parameter in comdmft.ini, which can have values of 'dos', 'spectral', 'dos_qp', or 'band'.

In all postprocessing the Fermi level is never recalculated.

Bear in mind that throughout this documentation of ComLowH I set the basis transformation matrix T to the identity.

A. Real-Frequency Observables; cal_mode=2 or 3; method = 'dos' or 'spectral'.

The frequencies used in ComLowH are the same real frequencies supplied in sig.dat. It is assumed that the Σ in sig.dat has been analytically continued to real energies. Except for the change from imaginary frequencies to real frequencies, and also the difference that the re-calculation of E_F is omitted, the calculation of G_l and U_{w90} is exactly the same as when cal_mode=1.

If cal_mode = 2, i.e. the 'dos' method is specified in comdmft.ini, then tdos is written to pdos.dat, and dos is written to pdos.dat:

$$tdos(\omega) = -N_k^{-1}\pi^{-1}\sum_k Tr\ Im[G_l(\omega,k)] = \sum_o pdos(\omega,o) \tag{19}$$

$$pdos(\omega,o) = -N_k^{-1}\pi^{-1}\sum_k Im[diag(U_{w90}(k)\ \ G_l(\omega,k)\ \ U^\dagger_{w90}(k))\ \ ] \tag{20}$$

$$\tag{21}$$

Moreover, $\hat{E}_{imp}$ is written to e_projected_mat.dat, $\hat{G}_{loc}$ is written to g_loc_mat.dat, and $\hat{\Delta}_{imp}$ is written to delta_mat.dat. These will be different from the values produced by the self-consistent calculation, because they are now produced using real frequencies instead of imaginary Matsubara frequencies.

If cal_mode = 3, i.e. the 'spectral' method is specified in comdmft.ini, then G_{path} is written to spectral.matrix, G_{pt} is written to spectral.dat, G_{po} is written to spectral_orb.dat, and $\lambda_{w90}(k)$ is written to wannier_band_non_interpolated.dat:

$$G_{path}(\omega,k) = (2\pi)^{-1}\imath U_{w90}(k)\ \ (G_l(\omega,k) - G_l^\dagger(\omega,k))\ \ U^\dagger_{w90}(k) \tag{22}$$

$$G_{po}(\omega,k) = diag(Re[G_{path}(\omega,k)]) \tag{23}$$

$$G_{pt}(\omega,k) = (2\pi)^{-1}Re[\imath Tr(G_l(\omega,k) - G_l^\dagger(\omega,k))] \tag{24}$$

B. Imaginary-Frequency Observables; cal_mode=4 or 5; method = 'dos_qp' or 'band'.

The frequencies used in ComLowH are the same real frequencies supplied in sig.dat, multiplied by $\imath$ to become Matsubara frequencies. The re-calculation of E_F is omitted. As far as I can tell, the files written by cal_mode=4 are exactly the same as the files written by cal_mode=5. The text output is a bit different.

$\Sigma(\omega = 0, i)$ and $-d\Sigma(\omega,i)/d\omega)$ are calculated by fitting the smallest six Matsubara frequencies of $\Sigma(\omega,i)$. The fitting is done by pfit_mod.F, which says that is a "Module for polynomial least-squares fitting".

$$\begin{aligned}
H_{w90}(q) &= V_{w90}^\dagger(q)\lambda_{DFT}(q)V_{w90}(q)\\
H_{w90}(k) &= N_q^{-1}\sum_{R,q}\exp(\imath R\cdot(k-q))H_{w90}(q)\\
H_{w90}(k) &= U_{w90}^\dagger(k)\lambda_{w90}(k)U_{w90}(k)\\
P_{pr}(k,i) &= \frac{1}{\sqrt{\chi\chi^\dagger}}\chi,\ \ \chi(i,k) =\ P_c(i)U_{w90}(k)\ P_E(k)\\
H_{ni}(k) &= -E_F+\lambda_{w90}(k)+\sum_i P_{pr}^\dagger(i,k)(\Sigma(\omega=0,i)-\Sigma_{DC}(i))P_{pr}(i,k)\\
Z^{-1}(k) &= 1+\sum_i P_{pr}^\dagger(i,k)(-d\Sigma(\omega,i)/d\omega)P_{pr}(i,k)\\
H_{qp}(k) &= U_{w90}(k)\,Z^{1/2}(k)\,H_{ni}(k)\,Z^{1/2}(k)\,U_{w90}^\dagger(k)\\
U_{qp}^\dagger(k)\lambda_{qp}(k)U_{qp}^\dagger(k) &= H_{qp}(k)
\end{aligned}$$

$\lambda_{qp}(k)$ is written out as wannier_band_qp_interpolated.dat. From the eigenvalues in $\lambda_{qp}(k)$, a subset is written out to wannier_band_qp_interpolated_orb.dat. The size of the subset is equal to the number of wannier functions. The selection follows a very simple heuristic that finds maximum absolute values of the matrix elements of $U_{qp}^\dagger(k)$. The heuristic does not guarantee against using the same eigenvalue more than once.

$\lambda_{w90}(k)$ is written out as wannier_band_non_interpolated.dat. From the eigenvalues in $\lambda_{w90}(k)$, a subset is written out to wannier_band_non_interpolated_orb.dat. The size of the subset is equal to the number of wannier functions. The selection follows a very simple heuristic that finds maximum absolute values of the matrix elements of $U_{qp}^\dagger(k)$. The heuristic does not guarantee against using the same eigenvalue more than once.

V. NOTES ON: OBTAIN WANNIER FUNCTIONS $\langle b\vec{k}|W(\vec{k})\rangle$

This is copied from WannierMatrixElementsDesign.tex

This discussion is about the work that would be necessary to get Wannier functions (local orbitals) for the 2021 code. For the comsuite version of flapwmbpt Wannier functions are already available.

There are at least three options available:

A. Sang's ansatz for Wannier functions

- Port (from comsuite to the 2021 version of rspflapw) Sang's code for creating Wannier functions. The code is in the routine named cal_overlapmax_bandprojection_to_mtorb_low, and is about 400 lines long.
  - There are three different versions of this routine in Sang's code. Which of the three is determined by an rmode parameter. In the official 2021 release of comsuite rmode is hard-coded to 0. In the version of comsuite released at the 2021 school rmode is equal to 0 for dft+dmft calculations and 2 for qsgw+dmft calculations. In the following I discuss only rmode=0. The rmode=2 version involves Sang running radial solver code to get some home-grown (Sang-grown) muffin-tin orbitals.
  - Wannier functions $|W^{als}\rangle$ are calculated separately for each value of atomic index a, spin s, and l quantum number.
  - Sang's code always produces wave-functions for ALL values of m within a given shell specified by a, l, s. These have the exact same values of W^{als}.
  - todo discuss dim_radial_orb, the size of the overlap matrix and of the wannier functions:
  - $\langle o_3|U^{als}|o_4\rangle = \frac{1}{N_k}\sum_{\vec{k},m,o_1,o_2,b,\epsilon_l\leq E_{b\vec{k}}\leq\epsilon_h}\langle o_3|\sqrt{O}|o_1\rangle\langle mo_1|b\vec{k}\rangle\ n_{FD}((E_{b\vec{k}}-\mu)/T)\ \langle b\vec{k}|mo_2\rangle\langle o_2|\sqrt{O}|o_4\rangle$, where:
    * O is the overlap matrix for muffin-tin orbitals.
    * The o's are indices over all orbitals with given a, l, s (todo and m?)
    * b is a band index and $|b\vec{k}\rangle$ is the bands. If doing a qsgw calculation these are quasiparticle bands. Only bands inside the energy interval $[\epsilon_l, \epsilon_h]$ are included.

-
    -
        * The bands are symmetrized using the sym_z_0 routine. There is something very similar in Andrey's 2021 code named symz.F, so porting the symmetrization should not be too hard. What these routines do is map the wave-function on one atom to the wave-function on symmetry-equivalent atoms. This includes a rotation that mixes the m quantum number and depends on the k-point, and multiplication by a phase.
        * the values of μ and T are hard-coded and never change.
    - Next we diagonalize U^{als} and find eigenvalues e_U and eigenvector v_U; $U^{als} = v_U^\dagger e_U v_U$. From this, the only thing that we use is the last eigenvector, $|v_{nU}\rangle$.
    - Next we create the proposed Wannier function $|W^{als}\rangle = \sqrt{O}|sv_{nU}\rangle$. Since $\sqrt{O}$ is a matrix and $|v_{nU}\rangle$ is a vector, W is also a vector.
    - An occupation for this Wannier orbital is calculated: $occ^{als} = \frac{1}{N_k}\sum_{\vec{k},m,o_1,o_2,b}\langle W|b\vec{k}\rangle \; n_{FD}((E_{b\vec{k}} - \mu)/T) \; \langle b\vec{k}|W\rangle$. Note that all bands are included in this sum. The occupation is printed out but never used.
    - A weight for this Wannier orbital is calculated: $weight^{als} = \frac{1}{N_k}\sum_{\vec{k},m,o_1,o_2,b,\epsilon_l \le E_{b\vec{k}} \le \epsilon_h}\langle W|b\vec{k}\rangle\langle b\vec{k}|W\rangle$. This weight is used to determine whether this particular Wannier orbital is in the right energy range and should be computed by wannier90, or whether it should be left out because its energy is wrong.
    - Also an array named bound_radfun is calculated and printed out in a file named radial_function_Fe_1.dat, for example. This is equal to $\langle r|O^{-1}|W^{als}\rangle$, where r is the radial coordinate inside the muffin tin.

- Port Sang's code for projecting the Wannier functions onto the bands. This code is in the routine named poorman_wan_proj, and is about 500 lines long.
    - This routine's job is to project the proposed Wanniers onto the bands, and then normalize the resulting projector.
    - The wanniers are indexed by a the atomic index, angular momentum l, m, and (implicitly) spin s.
    - Each atom has a local x axis and a local z axis. The user can change these from their default values using a file named local_axis.dat. When computing the wanniers associated with atom a, the bands are rotated by a rotation matrix R which is determined by the x and z axis. R mixes m-values of the muffin-tin states. The code for the rotation should be completely portable to Andrey's new code.
    - $P_k = \langle b\vec{k}|R|W\rangle$. As before, the bands are symmetrized by sym_z_0, which is like symz.F in Andrey's 2021 code.
    - Next the projector is renormalized in exactly the same way as Ran does his normalization. The final Wannier functions are $P_k N$, where $N = \frac{1}{\sqrt{P_k^\dagger P_k}}$.

- Notice that when doing a qsgw calculation Sang uses quasiparticle bands, not the bands of the static hamiltonian, to define his ansatz. Probably we want to do the same.

- Finally Sang's symmetric_orthonormalization massages $\langle i\vec{k}|t(l,m)\rangle$ so that it contains orthonormal states that are the wannier functions. This code can be simply copied without changes.

B. Wannier90 functions

This option would require all the work from Sang's ansatz, plus one additional work item: Port Sang's code for calculating the overlaps between article bands at different k-points. This is a subroutine named wannier_mmn, and is about 350 lines long.

- This time not only the muffin-tin sector of the bands must be symmetrized using sym_z_0; also the interstitial part must be symmetrized using sym_a_0. sym_a_0's equivalent in Andrey's 2021 code is sym_a.F.

- wannier_mmn iterates over every pair of neighboring k-points.

- Inside the loop over pairs, wannier_mmn calculates the overlap matrix between muffin-tin orbitals, using the fifi_j_prepare_onepoint and integral_band_pw_band_wan and integral_band_pw_band_mt_wan routines.

  - fifi_j_prepare_onepoint is responsible for doing the radial integral $\int dr\, r^2\, f(r)\, g(r)\, J_l(r)$, where f and g are the muffin-tin functions at different k-points, and J_l is a bessel function. Andrey's 2021 code has a similar routine named fifi_j_prepare.
  - integral_band_pw_band_mt_wan is responsible for angular variables, using clebsch-gordan coefficients. Andrey's 2021 code has a similar routine named wan_wan_pw_0.

- Inside the loop, wannier_mmn calculates overlaps between bands using the integral_band_pw_band_wan routine. This is easy to understand and should be easy to port.

C. Ran's ansatz

Portobello creates projectors in two steps. The first step chooses specific muffin-tin orbitals that will be the candidate Wannier functions, and matrices enforcing symmetries between those orbitals:

- The OrbitalsDefinition object makes a list of all of flapwmbpt's muffin-tin orbitals, and determines which of these match the user's requirements; for instance iron 3d-shell muffin-tin orbitals. For each of these it generates candidate Wannier functions. It keeps track of n, l, m quantum numbers, so that a full d-shell would have five candidate Wannier functions.
  - There may be several distinct muffin-tin orbitals at specific n, l, m, corresponding to different LAPW states. Portobello ignores orbitals which are derivatives; these are not used as candidate Wannier functions.
  - Orbitals with $l > 2$ are ignored unless they are marked by the user as correlated.
  - Corey implemented subshells.
- Portobello analyzes the crystal symmetry. It identifies which atoms are equivalent under crystal symmetry, while keeping track of magnetism and antiferromagnetism.
- Based on this analysis the SetRotMatrixAndRep routine calculates linear combinations of the muffin-tin orbitals, which are expressed using variables named orbs.rotation and orbs.subM. Equivalent atoms are not mixed together.
- The final candidate Wannier functions are expressed by a matrix $V = \text{orbs.rotation} \times \text{orbs.subM} \times \frac{1}{\sqrt{\langle i|O|i\rangle}}$. The matrix element $\langle i|O|i\rangle$ is a diagonal element of the overlap matrix between muffin-tin-orbitals; it is the normalization factor of a muffin tin orbital.
  - The overlap matrix O is stored by the GetLAPWBasisImp routine.
- All this information - the selected Wannier function candidates, the symmetrization matrix V, and more - are stored in a file named projector.h5.

The second step occurs when it is time to project the chosen muffin-tin orbitals onto bands:

- Portobello reads in projector.h5, with all its information about muffin-tin orbitals and their symmetries. This info is encapsulated in the V matrix, which has a size equal to the number of wannier functions.
- Portobello selects a subset of the bands which fit into an energy window. These are packaged in a matrix named Z_k, which depends on the k-point k.
  - The candidate wannier functions have only muffin-tin components, not interstitial components. Therefore Z_k needs only muffin-tin components of the bands.
- The rest of the projection work is performed in the CalculateP routine.
- Next it calculates $P_k = Z_k O V$. Z_k is the bands in the energy window, O is the overlap matrix between muffin-tin orbitals, and V is the candidate Wannier functions. P_k's dimensions are num_bands × num_wann; it maps from wannier functions to bands.
- The Wannier functions are not yet orthogonal, so portobello orthogonalizes them with a matrix $N = \frac{1}{\sqrt{P_k^\dagger P_k}}$.
- The final Wannier functions are: $P_k N$.

Conclusions: If I omit the symmetrization logic in portobello, then the remainder of its calculation of Wanniers is simpler and more portable [to Andrey's 2021 code] than Sang's calculation of Wanniers. Even if we did include the work of porting the symmetrization logic, it would still be relatively easy to port.

VI. ABOUT: COMWANN, WANNIER90, AND WANNIER-RELATED PORTIONS OF COMCOULOMB

Copied from Documentation of UOptimization.tex.

A. Flowchart

- Based on an energy window [dis_win_min and dis_win_max] supplied by the user in comwann.ini, comwann completely excludes bands outside that window from contributing to wannier functions.
- comwann chooses wave-functions inside the muffin tins with specific values of angular momentum quantum number l, m. These are named $|t_i\rangle$. The coeff_radial orbitals $|t_i\rangle$ are functions of r only. They are a kind of average of the radial structure of bands, for all bands whose energy $E(n, k)$ are inside [dis_froz_min, dis_froz_max].
- Based on $|t_i\rangle$'s overlap with the bands from rspflapw, and on user configuration parameters, comwann selects which atomic shells should have corresponding wannier functions, and thus arrives at a total number of wannier functions. It writes to disk a file named seedname.inip which contains information about each of the wannier functions: which atom, the l, m angular momentum quantum numbers, and the local x and z axis.
- Starting with the overlap matrix $\langle\psi_{nk}|t_i\rangle$ between bands and the chosen muffin-tin orbitals, comwann orthogonalizes this to get trial wannier functions, i.e. proposals to wannier90 for the wannier functions.
- comwann calls wannier90, passing in the trial wannier functions and several user control parameters from comwann.ini. wannier90 disentangles the desired bands from bands outside an energy window, and minimizes the spatial spread, and returns the resulting wannier matrices to comwann.
- comwann prints out diagnostic information and saves wannier.dat, which contains information about the wanniers that will be used by other parts of comsuite.
- If the write_wan user configuration parameter in comwann.ini is set to 1, then comwann interpolates the wannier functions on to a cartesian mesh and saves this to .xsf files so that the xcrysden viewer can be used to view the wanniers. It also computes its own estimate of the spatial spreads and centers of the wannier functions.
- In comcoulomb.ini the user tells comsuite which of the wannier orbitals are correlated and which are not. comcoulomb uses this information to calculate the cRPA screened interaction. It calculates the full polarizability and subtracts the correlated polarizability to obtain the cRPA polarizability. There are four schemes for using the user's designation of correlated orbitals, along with the information in wannier.dat and seedname.inip and certain user configuration parameters in comcoulomb.ini, to calculate different versions of the correlated polarizability.
- comcoulomb uses the wannier information in wannier.dat to calculate products of wannier functions, and then takes matrix elements of the screened interaction with respect to these products.
- Using the matrix elements and information from wannier.inip about which orbitals are in which atomic shell, comcoulomb calculates the Slater integrals for each atomic shell and saves them to disk. These are used for the double counting calculation and the dmft calculation.

B. Inputs

- local_axis.dat
- comwann.ini
- comcoulomb.ini
- files from rspflapw, including info.rst, info2.rst, seedname.rst.

1. *local_axis.dat*

comwann uses the axis information from local_axis.dat (atom_xaxis and atom_zaxis for each atom type) to:

- Rotate muffin-tin functions (inside of bands from rspflapw). This is done inside of wannier_orbital, as part of calculating proj, which helps choose wannier orbitals. proj is the overlap of the rotated muffin-tin functions with the coeff_radial array. coeff_radial contains trial orbitals $|t_i\rangle$ with distinct angular momentum character sitting on each atom. The coeff_radial orbitals $|t_i\rangle$ are functions of r only. They are a kind of average of the radial structure of bands, for all bands whose energy $E(n,k)$ are inside [dis_froz_min, dis_froz_max].
- Create seedname.inip which contains info about each wannier orbital.
- Rotate muffin-tin functions (inside of bands from rspflapw). This is done inside of poorman_wan_proj, as part of calculating a_matrix which tells wannier90 what the initial trial wannier orbitals to start with. These trial orbitals are called $|g_i\rangle$. a_matrix is computed by taking the overlap of the rotated muffin-tin functions inside the rspflapw bands with the coeff_radial array, and then orthonormalizing the overlap matrix. coeff_radial contains trial orbitals $|t_i\rangle$ with distinct angular momentum character sitting on each atom. So if $\langle\psi_{nk}|$ is the bands, then a_matrix $|g_i\rangle$. is calculated by forming the matrix $\langle\psi_{nk}||t_i\rangle$ and then orthonormalizing it. The coeff_radial orbitals $|t_i\rangle$ are functions of r only. They are a kind of average of the radial structure of bands, for all bands whose energy $E(n,k)$ are inside [dis_froz_min, dis_froz_max].

If local_axis.dat doesn't exist, then atom_xaxis = $\hat{x}$ and atom_zaxis = $\hat{z}$.
local_axis is not used for anything else in comsuite. comwann is the only program that uses it.
The docs for ComWann say that axis rotation is not implemented for systems with spin-orbit coupling.

2. *comwann.ini*

comwann.ini is written to by comrisb.py and comdmft.py.

- dis_win_min and dis_win_max :
  - Passed on to wannier90, where it will be used as the outer energy window for disentangling bands. In wannier90 states outside of this window are left entirely out of the wannier functions. In eV.
  - Used by comwann as an energy window - bands outside of this window are removed entirely from consideration, are not passed on to wannier90, and make no contribution to the wannier functions. todo: I don't like this since wannier90 does its own disentanglement, so have an option to turn it off or to make comwann's window larger than wannier90's window. However the meaning of this cutoff in wannier90 is the same as in comwann, so maybe it's OK to use it twice. Still it seems like overkill because comwann does something that wannier90 would do anyway.
- dis_froz_max and dis_froz_min :
  - ComWann adjusts dis_froz_max; see the call to reset_dis_froz_max.
  - dis_froz_min is used to decide the value of weight_low, which in turn is used to select which shells will be included as wannier functions.
  - dis_froz_max and dis_froz_min are passed on to wannier90, where they will be used as the inner (frozen) energy window for disentangling bands. Inside this window wannier90 makes sure that it uses the exact bands and energies from rspflapw. In eV.
  - dis_froz_max and dis_froz_min are used as input into the routine that is responsible for calculating weight_in_froz_win and coeff_radial. coeff_radial contains trial orbitals $|t_i\rangle$ with distinct angular momentum character sitting on each atom. The coeff_radial orbitals $|t_i\rangle$ are functions of r only. They are a kind of average of the radial structure of bands, for all bands whose energy $E(n,k)$ are inside [dis_froz_min, dis_froz_max].
- num_iter : Passed on to wannier90, where it limits the number of iterations in the search for wannier functions which optimize the spatial spreads.
- dis_num_iter : Passed on to wannier90, where it limits the number of iterations in the disentanglement procedure.

- cut_low : Used only to select atomic shells (atom number + angular momentum number l) to be included in the wannier functions. Shells are not included in the wannier functions unless (weight_low .le. cut_low). weight_low is the overlap between coeff_radial and the muffin-tin sector of those bands from rspflapw which have eigenvalues less than dis_froz_min. coeff_radial contains trial orbitals $|t_i\rangle$ with distinct angular momentum character sitting on each atom. So if $\langle\psi_{nk}|$ is the bands, then proj = $\langle\psi_{nk}||t_i\rangle$. The coeff_radial orbitals $|t_i\rangle$ are functions of r only. They are a kind of average of the radial structure of bands, for all bands whose energy $E(n,k)$ are inside [dis_froz_min, dis_froz_max].
- cut_froz : Used only to select atomic shells (atom number + angular momentum number l) to be included in the wannier functions. Shells are not included in the wannier functions unless (weight_in_froz_win .ge. cut_froz).
- rmode and mt_fac are used as input into the routine that is responsible for calculating weight_in_froz_win and coeff_radial. These user configuration parameters have been removed in comsuite version 2.1. coeff_radial contains trial orbitals $|t_i\rangle$ with distinct angular momentum character sitting on each atom.
- cut_total is never used.
- writewan : If this is true, then comwann constructs and saves representations of the wannier functions on grid and calculates and prints their spreads and center. It also saves out .xsf files containing these wanniers. These .xsf files are in a format for the xcrysden viewer, and could possibly could be used by pymatgen.
- rstpath : a path telling where to find files from rspflapw.
- ubi_txt : depending on whether it is 'dft' or 'qp' decides whether commwann/wannier90 will use dft or qp eigenvalues and bands.

3. comcoulomb.ini

Except for wanpath, all of the following should be moved into comwann.ini:

- wanpath : the path to wannier.dat and seedname.inip.
- u_mode : selects the crpa scheme for calculating the correlated polarizability. The possible values are wnds, gtop, bnse, and enpj. And maybe edmf but that seems to be commented out.
- projemin, projemax assist are an energy window that is used to control the crpa calculation of the correlated polarizability. Not used if u_mode = enpj. In eV.
- enpj_emin, enpj_emax are an energy window that is used to control the crpa calculation of the correlated polarizability. Used only if u_mode = enpj. In eV.
- num_orb_cor, wan_correlated : specify which of the wannier orbitals should be considered to be correlated. Not used if u_mode = enpj, in which case all wannier orbitals are considered to be correlated.
- p_low_mode : should always be set to 1.

C. Control of wannier90

Passed into wannier_setup, wannier_run, and also via wannier.win:

- ndiv, from rspflapw, called mp_grid by wannier90, in .chk file
- nqdiv, from rspflapw, called num_kpts by wannier90
- transpose(rbas)*par*bohr, from rspflapw, called real_lattice by wannier90, in .chk file
- transpose(gbas)*2.0d0*pi/par/bohr, from rspflapw, called recip_lattice by wannier90, in .chk file
- kpt_latt, from rspflapw, in .chk file
- natom, from rspflapw

- atom_symbols, from rspflapw
- eigenvalues, from rspflapw and reduced by comwann
- num_iter, from comwann.ini
- dis_num_iter, from comwann.ini
- dis_win_max, from comwann.ini
- dis_win_min, from comwann.ini
- dis_froz_max, from comwann.ini
- dis_froz_min, from comwann.ini
- bands_plot, from comwann
- bands_num_points, from comwann
- bands_plot_format, from comwann
- kpoint_path, from comwann
- tau*par*bohr, from rspflapw, called atoms_cart by wannier90
- nbndf, from rspflapaw, called num_bands_tot by wannier90
- gamma_only, always false
- spinors, always false
-
- nntot, received from wannier_setup, fed into wannier_run
- num_bands, from comwann, in .chk file
- num_wann, from comwann, in .chk file
- exclude_bands, from comwann, goes to wannier.win and wannier_run, not the same as exclude_bands(i) in .chk file.
- M_matrix, from comwann, in .chk file
- A_Matrix, from comwann
- restart=plot, to tell comwann to use the .chk file.
- write_hr for writing out ham_r,nrpts,irvec,ndegen.
- write_u_matrices for writing out files with u_matrix, u_matrix_opt.

Additional variables needed for the .chk file. All of these variables are written by wannier90. However comwann will want to alter u_matrix incorporating the unitaries that are generated by optimization and also the permutation matrix, header (Date and time), chkpt. The w90chk2chk utility will be necessary.

- num_exclude_bands
- exclude_bands(i)
- have_disentangled, should always be true
- u_matrix
- u_matrix_opt
- lwindow

- header
- nntot
- chkpt
- omega_invariant
- ndimwin
- wannier_centres
- wannier_spreads

Other variables received from wannier_setup, necessary for comwann's preparation for wannier_run:

- nntot
- nnlist
- nncell

D. Output to comsuite

At present comwann does not properly handle spin polarized calculations. comwann gets the number of spins, nspin, from info.rst. If this number is 2, i.e. if doing a spin polarized calculation, then comwann does first spin=1 and saves out wannier.dat and seedname.inip. Next it does spin=2 and saves out wannier.dat and seedname.inip, overwriting the values from spin=1.

1. wannier.dat

- transpose(rbas)*par*bohr, from rspflapw, called real_lattice by wannier90, in .chk file
- transpose(gbas)*2.0d0*pi/par/bohr, from rspflapw, called recip_lattice by wannier90, in .chk file
- ndiv, from rspflapw, called mp_grid by wannier90, in .chk file
- kpt_latt, from rspflapw, in .chk file
- nrpts, can be obtained from wannier90: plot_main/hamiltonian_write_hr writes out ham_r,nrpts,irvec,ndegen to a file, if write_hr
- irvec, can be obtained from wannier90: plot_main/hamiltonian_write_hr writes out ham_r,nrpts,irvec,ndegen to a file, if write_hr
- ndegen, can be obtained from wannier90: plot_main/hamiltonian_write_hr writes out ham_r,nrpts,irvec,ndegen to a file, if write_hr
- ham_r, can be obtained from wannier90: plot_main/hamiltonian_write_hr writes out ham_r,nrpts,irvec,ndegen to a file, if write_hr
- have_disentangled, should always be true
- u_matrix , from wannier90, in .chk file, can also be obtained from seedname_u.mat, Written if write_u_matrices = .TRUE.
- u_matrix_opt , from wannier90, in .chk file, can also be obtained from seedname_u_dis.mat, Written if write_u_matrices = .TRUE.
- lwindow , from wannier90, in .chk file, todo can this be dumped out in any other way?
-
- num_bands, from comwann, in .chk file

- num_wann, from comwann, in .chk file
- include_bands, from comwann
- v_matrix, from wannier90 via comwann, v_matrix = u_matrix * u_matrix_opt
- eigenvalues, from rspflapw, reduced by comwann
- todo: z_wan_bnd and ev_wan_bnd should be saved here - otherwise the clients of wannier.dat have to reconstruct this info, and can go wrong.
- ham_k , generated by comwann from u_matrix, u_matrix_opt, eigenvalues, etc.

2. seedname.inip File

Comwann chooses wannier orbitals that will be computed, and writes this info to seedname.inip.

For each wannier orbital, the .inip file tells its atom number, its angular momentum l, m numbers, and also atom_xaxis and atom_zaxis.

The angular momentum l, m numbers specify "real spherical harmonics" which are related to the usual ones by taking the real and imaginary parts. See https://en.wikipedia.org/wiki/Table_of_spherical_harmonics#Real_spherical_harmonics . See also https://en.wikipedia.org/wiki/Spherical_harmonics .

In the p shell, the $m = -1$ harmonic is oriented along the y axis, $m = 0$ along the z axis, and $m = 1$ along the x axis. In the d shell, the $m = -2$ harmonic is the d_{xy} orbital, $m = -1$ is the d_{yz} orbital, $m = 0$ is the d_{z^2} orbital, $m = 1$ is the d_{zx} orbital, and $m = 2$ is the $d_{x^2-y^2}$ orbital.

The axis information is taken from local_axis.dat which is supplied by the user and contains atom_xaxis and atom_zaxis for each atom type. If local_axis.dat doesn't exist, then atom_xaxis = $\hat{x}$ and atom_zaxis = $\hat{z}$.

3. Additional Output that should be handled by ComWann, not inside ComCoulomb as done now

Probably this info should go in comwann.dat. The code that generates it is located inside of ComCoulomb, and should be moved into ComWann.

- The states and energies that will be used to build the Green's function and the polarizability. Represented in the bands basis. Also a count of these states.
- The pseudo-unitary matrix named proj_renorm which is used to construct Wannier functions.
- Information about the Wannier functions: itinerant vs. correlated, which atom they are associated with, their angular momentum numbers l, m, the x and z axis associated with their atom.
- The supporting information which is required for calculating Slater integrals.

4. Slater integral files from comcoulomb

These contain the Slater integrals, as a function of frequency, for U, V, and W. They are printed out for each shell that has been designated as correlated. They are used by ComDC, CTQMC, and probably also ComLowH.

5. Output that is never used by other parts of comsuite

- ComWann computes atom_wan, atom_wan_distance, distance,tempvec,tempvec2, which atom each wannier is centered at, distances between wanniers - this is printed out but not saved anywhere or used by anything.
- ComWann calculate and print out overlaps between initial trial orbitals and wanniers. overlap = v_matrix† * a_matrix

- if the writewan flag in comwann.ini is on, ComWann
  wannier_realgrid constructs and saves representations of the wanniers on grid and calculates and prints their spreads and center. It also saves out .xsf files containing these wanniers. These .xsf files are in a format for the xcrysden viewer, and could possibly could be used by pymatgen.
- wannier90 produces output in wannier.wout and a .chk file that can be used by others. wannier.wout contains wannier_centres and wannier_spreads.
- ComWann produces weight_trial.dat, occ_trial.dat, orb_for_Froz_win.dat, weight_final.dat, occ_final.dat, radial_function_zzz.dat

E. Other

Other functionalities that we might want to add:

- The latest wannier90 has a feature for preserving crystal symmetry during wannier optimization, but it requires additional info from comwann.
- Wannier90 has a mechanism for suggesting to comwann what the trial orbitals will be.
- Take advantage of seedname_band.kpt , allowing band structure plots to have flexible user specified paths through k-space, instead of the hardcoded 11 paths that Andrei has implemented. seedname_band.kpt is written if bands_plot=.TRUE.; The k-points used for the interpolated band structure, in units of the reciprocal lattice vectors. This file can be used to generate a comparison band structure from a first-principles code.
- Get comwann to work properly with spin polarized calculations.

Additional notes:

- comwann quits if magn=2, i.e. if doing a noncollinear calculation.
- comwann tells wannier90 that the wavefunctions are not spinors.

---

[*] Electronic address: `vincent@sacksteder.com`