Explanation of the Projector Selection Loop in `run_acbn0`

PAOFLOW Documentation Supplement

September 29, 2025

1 Context

The ACBN0 workflow constructs reduced bases for the density matrix and two-electron Coulomb integrals prior to evaluating the Hubbard parameters U and J. Within `run_acbn0`, the highlighted loop processes the Quantum ESPRESSO `projwfc.out` projectors to identify the contiguous subset of states associated with a given Hubbard shell (e.g. `Fe-d`). The loop operates once per entry in `self.uVals`, ensuring that each targeted shell has a consistent set of projectors for subsequent density and two-electron calculations.

2 Relevant Code

```
for k, v in self.uVals.items():
    ostates = []
    ustates = []
    s = k.split('-')[0]
    horb = self.hubbard_orbital(k)
    for n, sl in enumerate(state_lines):
        stateN = re.findall('\(([^")]+)\)', sl)
        oele = stateN[0].strip()
        oL = int(re.split('=| ', stateN[1])[1])
        if s in oele and oL == horb:
            ostates.append(n)
            if s == oele:
                ustates.append(n)
    sstates = [ustates[0]]
    for i, us in enumerate(ustates[1:]):
        if us == 1 + sstates[i]:
            sstates.append(us)
        else:
            break
```

3 Full Narrative Description

Before the highlighted loop is executed, `state_lines` has already been trimmed to the portion of `projwfc.out` that lists projectors in the form

```
 state #   1 (Fe  ) l=2 m= -2    ...
 state #   2 (Fe1 ) l=2 m= -1    ...
```

so each entry supplies both the chemical symbol (possibly with numeric suffixes distinguishing inequivalent atoms) and the angular momentum channel l.

The outer loop iterates over `self.uVals`, which contains keys such as `"Fe-d"` or `"O-p"` paired with the current Hubbard parameters. Splitting the key at the hyphen isolates the species label s, while `hubbard_orbital(k)` converts the orbital letter to the integer angular momentum L that Quantum ESPRESSO reports in `projwfc.out`. Two lists are prepared:

- `ostates` collects every projector index matching the species (allowing suffixes like `Fe1`, `Fe2`) and the requested L. This set will become the reduced basis for the density-matrix calculation.
- `ustates` retains only those indices whose label is exactly the bare symbol (e.g. `Fe` without numeric decorations). These correspond to the canonical representative of the shell and seed the two-electron basis.

Inside the inner `for`, each projector line `sl` is parsed with a regular expression to extract the parenthesized token containing the species identifier, along with the substring reporting `l=`. If both the species and L criteria are met, the current index n is appended to `ostates`. When the species matches exactly (no suffix), the index is also appended to `ustates`. The distinction ensures that all symmetry-equivalent projectors contribute to the density matrix, while the two-electron block focuses on a single contiguous set of $2L + 1$ magnetic sublevels anchored on one atom.

After scanning all projectors, the code assumes that `ustates` begins with the first magnetic sublevel and that subsequent sublevels appear in consecutive order. The list `sstates` is seeded with the first element of `ustates` and is then extended only while the next entries remain consecutive integers. As soon as a gap is encountered, the growth stops. This protects the downstream Coulomb integrals from inadvertently mixing in projectors belonging to another atom with the same orbital character that may be interleaved later in `projwfc.out`.

Finally, `ostates` and `sstates` are serialized when writing the temporary ACBN0 input file: the first defines `reduced_basis_dm` for the density-matrix reconstruction, and the second defines `reduced_basis_2e` used in the Hartree energy evaluation. By the time `acbn0()` reads these inputs, each Hubbard shell has a coherent, minimal projector set that mirrors the original logic proposed in the plain-text explanation.

4 Step-by-Step Explanation

Iteration over shells: Keys k like `Fe-d` label the species and orbital. The angular momentum quantum number L is obtained via `hubbard_orbital`, mapping to the Quantum ESPRESSO conventions ($L = 0$ for s, 1 for p, 2 for d, etc.).

Projector scanning: Each `state_lines` entry corresponds to a `projwfc` projector. The regular expression extracts (i) the species label `oele` appearing inside parentheses (e.g. `Fe`, `Fe1`) and (ii) the $2L+1$ resolved quantum number reported as `l=`. When the line matches both the species s (up to suffixes like `Fe1`) and the desired L, the projector index n is appended to `ostates`.

Exact species matches: Some projectors may include suffixed species labels (e.g. `Fe1`) created by Quantum ESPRESSO to distinguish inequivalent atoms. The subset `ustates` retains only indices where the label equals the bare chemical symbol (e.g. `s == oele`), ensuring that subsequent two-electron integrals focus on the canonical atom within the shell.

Contiguity enforcement: The list `ustates` is expected to contain consecutively numbered projectors corresponding to the $2L+1$ magnetic sublevels. The slice building `sstates` starts from the first exact match and grows the list only while consecutive indices are found. The moment a gap is detected, the loop stops, preserving a contiguous block of projectors. This guards against accidental inclusion of projectors belonging to symmetry-related but distinct atoms or shells.

Outputs: The indices `ostates` feed the reduced density-matrix basis, while `sstates` defines the subset used for two-electron integrals. Both lists are serialized into the ACBN0 input file as comma-separated strings.

5 Edge Cases and Assumptions

- The code assumes at least one exact species match appears in `ustates`; otherwise `ustates[0]` raises an exception. In practice, QE emits the unsuffixed projector first, so the assumption holds.
- Contiguity is measured in the raw projector indices from `projwfc`. If Quantum ESPRESSO ever inserted unrelated projectors between magnetic sublevels, the loop would truncate `sstates` prematurely, reducing the two-electron basis.
- Species labels with suffixes (e.g. `Fe1`, `Fe2`) are still recognized for `ostates` membership because the condition checks `s in oele`.

6 Denominator Formulas

The Hartree numerators described above are paired with occupation-based denominators that normalize the on-site interaction strengths. In the notation of Agapito *et al.* (Phys. Rev. X **5**, 011006, 2015), the shell-averaged expressions are

$$\mathcal{D}_U = \sum_{\sigma\sigma'} \sum_{mm'} n^{\sigma}_{mm}\, n^{\sigma'}_{m'm'}, \tag{1}$$

$$\mathcal{D}_J = \sum_{\sigma} \sum_{m\neq m'} n^{\sigma}_{mm}\, n^{\sigma}_{m'm'}. \tag{2}$$

The implementation mirrors these formulae by first accumulating the k-averaged occupations returned by `Nmm`. For the spin-unpolarized case ($n_{\text{spin}} = 1$), the code forms the aggregates

$$N_{ab} = \sum_{m,m'} n_{mm}\, n_{m'm'}, \qquad N_{aa} = \sum_{m\neq m'} n_{mm}\, n_{m'm'}, \tag{3}$$

and combines them as

$$\mathcal{D}_U = 2\,\Re\big(N_{aa} + N_{ab}\big), \qquad \mathcal{D}_J = 2\,\Re\big(N_{aa}\big). \tag{4}$$

When spin polarization is present, separate spin-up and spin-down occupations are accumulated:

$$N_{aa} = \sum_{m\neq m'} n^{\uparrow}_{mm}\, n^{\uparrow}_{m'm'}, \qquad N_{bb} = \sum_{m\neq m'} n^{\downarrow}_{mm}\, n^{\downarrow}_{m'm'}, \qquad N_{ab} = \sum_{m,m'} n^{\uparrow}_{mm}\, n^{\downarrow}_{m'm'}, \tag{5}$$

which translate directly to the source-code assignments

$$\mathcal{D}_U = \Re\big(N_{aa} + N_{bb} + 2N_{ab}\big), \qquad \mathcal{D}_J = \Re\big(N_{aa} + N_{bb}\big). \tag{6}$$

These denominators appear in the final ratios $U = (27.2114\mathcal{N}_U)/\mathcal{D}_U$ and $J = (27.2114\mathcal{N}_J)/\mathcal{D}_J$, with $U_{\text{eff}} = U - J$.

7 Formula Reference by Function

The following catalogue restates every analytical expression documented in the `ACBNO` docstrings, pairing each formula with the variables that realize it in code and a concise excerpt from `ACBNO.py`. Line numbers refer to the master branch snapshot captured during this analysis.

7.1 `optimize_hubbard_U` (lines 229–271)

Formula.

$$\max_k \left|U_k^{(n+1)} - U_k^{(n)}\right| < \delta_{\text{conv}}$$

Description. Ensures the self-consistent loop halts once all Hubbard parameters change by less than the user-specified threshold.

Variables in code.

- `self.uVals[k]`: previous iteration value $U_k^{(n)}$.
- `v` / `new_U[k]`: newly computed $U_k^{(n+1)}$.
- `convergence_threshold`: tolerance δ_{conv}.
- `converged` flag updated when any component violates the bound.

```
for k, v in new_U.items():  # ACBN0.py lines 263-269
    print(f'  {k} : {v}')
    if converged and np.abs(self.uVals[k] - v) > convergence_threshold:
        converged = False
```

7.2 `acbn0` (lines 274–373)

Formulas.

$$U = \frac{\sum_{\sigma\sigma'} \sum_{m_1m_2m_3m_4} \langle m_1m_3|\hat{v}|m_2m_4\rangle \, D^{\sigma}_{m_1m_2} D^{\sigma'}_{m_3m_4}}{\sum_{\sigma\sigma'} \sum_{mm'} n^{\sigma}_{mm} n^{\sigma'}_{m'm'}}, \tag{7}$$

$$J = \frac{\sum_{\sigma} \sum_{m_1m_2m_3m_4} \langle m_1m_4|\hat{v}|m_3m_2\rangle \, D^{\sigma}_{m_1m_2} D^{\sigma}_{m_3m_4}}{\sum_{\sigma} \sum_{m\neq m'} n^{\sigma}_{mm} n^{\sigma}_{m'm'}}, \tag{8}$$

$$U_{\text{eff}} = U - J. \tag{9}$$

Numerators $\mathcal{N}_U$, $\mathcal{N}_J$ are built from the spin-resolved density matrices $D^{\sigma}_{mm'}$ returned by `DR`, while denominators $\mathcal{D}_U$, $\mathcal{D}_J$ use the on-site occupations $n^{\sigma}_{mm'}$ from `Nmm` as detailed in Section 6.

Variables in code.

- `num_U`, `num_J`: Hartree numerators $\mathcal{N}_U$, $\mathcal{N}_J$ assembled from $D^{\sigma}_{mm'}$.
- `den_U`, `den_J`: denominators $\mathcal{D}_U$, $\mathcal{D}_J$.
- `U`, `J`, `U_eff`: final interaction strengths.
- `nlm`, `nlmd`: on-site occupations $n^{\sigma}_{mm'}$.

```
if nspin == 1:  # ACBNO.py lines 331-340
    ...  # accumulate Naa, Nab
    den_U = 2 * (Naa.real + Nab.real)
    den_J = 2 * Naa.real
else:  # lines 342-354
    ...  # accumulate Naa, Nbb, Nab with spin channels
    den_U = Naa.real + Nbb.real + 2 * Nab.real
    den_J = Naa.real + Nbb.real

num_U, num_J = self.hartree_energy(DR_0, DR_0_dn, gauss_basis, basis_2e)
hartree_to_eV = 27.211396132
U = U_eff = hartree_to_eV * num_U / den_U
if den_J == 0:
    J = 'Inf'
else:
    J = hartree_to_eV * num_J / den_J
    U_eff -= J
```

7.3 `getbasis` (lines 659–690)

Formula.

$$\chi_\mu(\mathbf{r}) = \sum_p c_{\mu p}(x-x_0)^{\ell_x}(y-y_0)^{\ell_y}(z-z_0)^{\ell_z}e^{-\zeta_{\mu p}\|\mathbf{r}-\mathbf{r}_0\|^2}$$

Description. Constructs contracted Gaussian basis functions by combining primitive Gaussians centered on each atomic site.

Variables in code.

- `bf = CGBF(pos*1.88973, a)`: creates basis function centred at $\mathbf{r}_0$.
- `bf.pcoefs`, `bf.pexps`, `bf.powers`: store $c_{\mu p}$, $\zeta_{\mu p}$, and (ℓ_x, ℓ_y, ℓ_z).
- `basis_functions`: collection of all χ_μ.

```
bf = CGBF(pos * 1.88973, a)          # ACBNO.py lines 681-688
for lx, ly, lz, coeff, zeta in subshell:
    bf.pnorms.append(1.0)
    bf.pexps.append(zeta)
    bf.pcoefs.append(coeff)
    bf.powers.append((lx, ly, lz))
basis_functions.append(bf)
```

7.4 `Dk` (lines 693–746)

Formulas.

$$H^\sigma(\mathbf{k})\,\mathbf{c}_{\nu\mathbf{k}\sigma} = \varepsilon_{\nu\mathbf{k}\sigma}\,S(\mathbf{k})\,\mathbf{c}_{\nu\mathbf{k}\sigma}, \tag{10}$$

$$n^\sigma_{mm'}(\mathbf{k}) = \sum_\nu f_{\nu\mathbf{k}\sigma}\,\langle\phi_m|\psi_{\nu\mathbf{k}\sigma}\rangle\langle\psi_{\nu\mathbf{k}\sigma}|\phi_{m'}\rangle, \tag{11}$$

$$D^\sigma(\mathbf{k}) = \sum_\nu f_{\nu\mathbf{k}\sigma}\,\mathbf{c}_{\nu\mathbf{k}\sigma}\mathbf{c}^\dagger_{\nu\mathbf{k}\sigma}. \tag{12}$$

Variables in code.

- eig, vec: eigenvalues $\varepsilon_{\nu\mathbf{k}\sigma}$ and eigenvectors $\mathbf{c}_{\nu\mathbf{k}\sigma}$.
- lm_dm, lm_2e: projected overlaps $\langle\phi_m|\psi_{\nu\mathbf{k}\sigma}\rangle$.
- D_k: assembled $D^\sigma(\mathbf{k})$ tensors.
- nlm_k: projected occupation matrices $n^\sigma_{mm'}(\mathbf{k})$.

```
eig, vec = eigh(Hks[:, :, ik], Sks[:, :, ik])  # ACBNO.py lines 724-743
occ_ind = np.where(eig <= 0.0)[0]
lm_dm = np.zeros((size_dm, len(occ_ind)), dtype=complex)
lm_2e = np.zeros((size_2e, len(occ_ind)), dtype=complex)
...
nlm_k[:, :len(occ_ind), ik] = lm_2e
lm_dm = np.sum(lm_dm, axis=0)
D_k[:, :, ik] = np.tensordot(np.conj(vec[:, occ_ind] * lm_dm),
                              vec[:, occ_ind], axes=([1], [1]))
```

7.5 Nmm (lines 749–772)

Formula.

$$N^\sigma_{mm'} = \frac{1}{\sum_{\mathbf{k}} w_{\mathbf{k}}} \sum_{\mathbf{k}} w_{\mathbf{k}}\, n^\sigma_{mm'}(\mathbf{k})$$

Variables in code.

- kwght: k-point weights $w_{\mathbf{k}}$.
- nlm: projected coefficients $n^\sigma_{mm'}(\mathbf{k})$.
- nlm_aux: accumulator for the weighted sum.

```
nlm_aux = np.zeros((lm_size, nbasis), dtype=complex)  # ACBNO.py lines 767-772
for ik, wght in enumerate(kwght):
    nlm_aux += wght * nlm[:, :, ik]
return np.sum(nlm_aux / np.sum(kwght), axis=1)
```

7.6 DR (lines 775–798)

Formula.

$$D^\sigma = \frac{1}{\sum_{\mathbf{k}} w_{\mathbf{k}}} \sum_{\mathbf{k}} w_{\mathbf{k}}\, D^\sigma(\mathbf{k})$$

Variables in code.

- Dk: matrices $D^\sigma(\mathbf{k})$.
- kwght: k-point weights.
- D: accumulator for the weighted density matrix.

```
D = np.zeros((nawf, nawf), dtype=complex)  # ACBNO.py lines 794-798
for ik, wght in enumerate(kwght):
    D += wght * Dk[:, :, ik]
return D.real / np.sum(kwght)
```

7.7 `hartree_energy` (lines 801–836)

Formulas.

$$\mathcal{N}_U = \sum_{\sigma\sigma'} \sum_{m_1m_2m_3m_4} \langle m_1m_3|\hat{v}|m_2m_4\rangle \, D^{\sigma}_{m_1m_2} D^{\sigma'}_{m_3m_4}, \tag{13}$$

$$\mathcal{N}_J = \sum_{\sigma} \sum_{m_1m_2m_3m_4} \langle m_1m_4|\hat{v}|m_3m_2\rangle \, D^{\sigma}_{m_1m_2} D^{\sigma}_{m_3m_4}. \tag{14}$$

Variables in code.

- `basis`, `basis_2e`: specify orbital quartet indices entering $\langle m_im_j|\hat{v}|m_km_l\rangle$.
- `int_U`, `int_J`: Coulomb integrals computed by `self.coulomb`.
- `DR_up`, `DR_dn`: spin-resolved density matrices $D^{\sigma}_{mm'}$.
- `tmp_U`, `tmp_J`: accumulated numerators $\mathcal{N}_U$, $\mathcal{N}_J$.

```
for k, l, m, n in itertools.product(basis_2e, repeat=4):  # ACBNO.py lines 826-835
    int_U = self.coulomb(basis[m], basis[n], basis[k], basis[l])
    int_J = self.coulomb(basis[m], basis[k], basis[n], basis[l])
    a_b = DR_up[m, n] * DR_up[k, l] + DR_dn[m, n] * DR_dn[k, l]
    ab_ba = DR_dn[m, n] * DR_up[k, l] + DR_up[m, n] * DR_dn[k, l]
    tmp_U += int_U * (a_b + ab_ba)
    tmp_J += int_J * a_b
```

8 Summary

The highlighted loop filters the `projwfc` projector list to deliver (i) every projector for the target species and orbital needed in the density-matrix basis and (ii) a contiguous subset anchored on the canonical species for the two-electron basis. This guarantees that each Hubbard shell in `self.uVals` is parameterized by a coherent and minimal projector set before the downstream evaluation of the ACBN0 Hubbard parameters.