spacial_indices = [orb // 2 for orb in list(spin_orbs)]
one_elec_xgrid = np.ix_(spacial_indices, spacial_indices)
two_elec_xgrid = np.ix_(spacial_indices, spacial_indices, spacial_indices, spacial_indices)

no differences between the two determinants

<u>one electron are prater - this case is trivial</u>

$\sum_{ij} < \psi\_1|a_i^\dagger a_j|\psi_1 >$

$\delta_{ij} \sum_{ij} < \psi_1|a_i^\dagger a_j|\psi_1 >$

$\sum_{i} < \Psi_1|$a^†_i $a_i|\Psi_1 >|$

one_elec_mel += np.einsum('ii->i, one_elec_ints[one_elec_xgrid])

<u>to electron are provider - as you can probably tell not sure abt this one, what could i do to get a better understanding? i feel like my np.eisnum here is in the right direction, but not sure</u>

$\sum_{rstu} < \Psi_1|a_r^\dagger a_s^\dagger a_t a_u|\Psi_1 >$

r=t and s=u or r=u and s=t

two_elec_mel += (0.5)*(np.einsum(rssr->, two_elec_ints[two_elec_xgrid]) - np.einsum(rsrs->, two_elec_ints[two_elec_xgrid]))

I assume that I want the np.einsum to output a scaler quantity, as I will not have any nested for loops to get this from smth like $\boxed{\text{r,s,t,u}}$. should I think about this diff Then I am currentl?

I imagined that this factor of one half comes so I don't double count ints. Maybe I need to have a better understanding of where this comes from though?

One difference between the two dets

<u>one elec a provider</u>

$\sum_{ij} < vac|a_f \dots a_m \dots a_a a_i^\dagger a_j a_a^\dagger \dots a_p^\dagger \dots a_f^\dagger|vac >$

for count(annhltn) == count(crtn) need m=i and p=j

$< vac|a_f \dots a_m \dots a_a a_m^\dagger a_p a_a^\dagger \dots a_p^\dagger \dots a_f^\dagger|vac >$

now I need to figure out the face factor needed to bring crtn and annhltn together in m and p? in General, I'm not sure how the anti commutator that I built earlier would be used with np.einsum or in this case without np.einsum needed.

one_elec_mel += [m//2, p//2]

<u>To electron or provider</u>

$\sum_{rstu} < vac|a_f \dots a_m \dots a_a a_r^\dagger a_s^\dagger a_t a_u a_a^\dagger \dots a_p^\dagger \dots a_f^\dagger|vac >$

two_elec_mel += np.einsum(rstu->rstt, two_elec_ints[two_elec_xgrid])$\boxed{\text{m,p,t,t}}$ - np.einsum(rstu->rsst, two_elec_ints[two_elec_xgrid])$\boxed{\text{m,s,s,p}}$

I am running into the prob where $\boxed{\text{m,p,t,t}}$ won't be defined because I am not in a nested for loop. idk how to get the np.einsum to output the correct scalar quantity here?

two differences between the two dirt torments

<u>to electron are prater</u>

$\sum_{rstu} < vac|a_f \dots a_m \dots a_n \dots a_a a_r^\dagger a_s^\dagger a_t a_u a_a^\dagger \dots a_p^\dagger \dots a_q^\dagger \dots a_f^\dagger |vac >$

for count(annhltn) == count(crtn) need (m=r and n=s or m=s and n=r) and (p=t and q=u or p=u and q=t)

two_elec_mel += [m//2,p//2,n//2,q//2] - [m//2, q//2, n//2, p//2]

again I need to figure out the face factor needed to bring crtns and annhltns together, but not sure how.

ik my questions are pretty complicated, So if you feel that it would be better for me to first learn from another resource, like some YouTube channel or text, just let me no. I was looking at some YouTube videos earlier dear, but I couldn't find anything that was able to explain the np.einsum very well for my case