Projection with Gaussian basis set: counterpoise method

Problem

At the moment the method that I use for projecting orbitals and determining transfer integrals and site energies works as follows :

1) Run a calculation on either molecule then on the dimer and obtain the three sets of orbitals (A, B and dim)

2) Orthogonalise the resulting orbitals by finding $\sqrt{S_{A,B,\mathrm{dim}}}$, where S is the overlap matrix and the suffix labels the calculation

3) Project the orthogonalised *dimer* orbitals on the space formed by the orthogonalised MOs of both A and B

4) Express the Fock matrix in this new basis set

This method has some short comings, firstly it is somewhat incorrect because it does not take into account the basis set super position energy, i.e. the fact that each calculation is performed on a different basis set. Secondly, the method might be rather slow because three large matrices need to be diagonalised. Thirdly the present implementation is a bit messy. I think there is another way to do it which uses a counterpoise calculation to make the computation, since I was at it I decided to rewrite the whole method in python and make it more "luser friendly".

Method

Summary of counterpoise method:

1) Perform three compuations *in the same basis set* but including different nuclear charges: only on A , only on B and on both AB (dim)

2) Project the non−orthogonalised orbitals of molecule A (and of molecule B) onto the orbitals of the dimer

3) Compute $\langle \Phi^A \left| F \right| \Phi^B \rangle$ to compute transfer integrals and $\langle \Phi^A \left| F \right| \Phi^A \rangle$to compute site energies. This is easy because we know what F is in the orthogonal basis set of the molecular orbitals (MO) of the dimer: it is simply a diagonal matrix with elements corresponding to the MO energies.

Definitions

The list of symbols I shall use:

$\phi^{(A,B)}_{i,j}$: coefficient of the j^{th}atomic orbital of the i^{th} molecular orbital for an isolated molecule (labelled A,B) (plain english: the orbitals of A,B)

$\Psi_{i,j}$: coefficient of the j^{th}atomic orbital of the i^{th}molecular orbital of the dimer (plain english the orbitals of AB)

$S_{i,j}$: overlap integral of the i^{th} and j^{th} atomic orbitals (the overlap matrix)

$\tilde{\phi}^{(A,B)}_{i,j}$: the coefficient of thej^{th} molecular orbitals of the dimer for the i^{th}molecular orbital of an isolated molecule (the orbitals of A,B in the basis set of the dimer)

ϵ : the diagonal matrix of orbital energies for the dimer

Projection of the orbitals

The overlap matrix forms a metric for the vector space of vectors in the basis set of atomic orbitals, therefore if we wanted to express a set of orbitals $\phi_{i,j}$in a new basis set defined by $\Psi_{i,j}$ the inner (dot) product would take form (repeated indeces denote summation btw):

$$\tilde{\phi}_{i,j} = \phi_{i,l}\, S_{l,k}\, \Psi_{j,k}$$

I think this is the form that is most obvious : the left hand side is all about the monomer orbital "i" in the dimer orbital "j" and the right hand side reflects that.

A bit of manipulation. S is symmetric

$$\tilde{\phi}_{i,j} = \phi_{i,l}\, S_{k,l}\, \Psi_{j,k}$$

Order does not matter

$$\tilde{\phi}_{i,j} = \phi_{i,l}\, \Psi_{j,k}\, S_{k,l}$$

Remembering that if A=BC matrix multiplication means $A_{i,j} = B_{i,k}\, C_{k,l}$:

$$\tilde{\phi}_{i,j} = \phi_{i,l} (\Psi . S)_{j,l}$$

Transpose the last term:

$$\tilde{\phi}_{i,j} = \phi_{i,l} (\Psi . S)^{t}_{l,j}$$

Notice that this too is a product and finally:

$$\tilde{\phi}^{A} = \phi^{A} . (\Psi . S)^{t}$$
$$\tilde{\phi}^{B} = \phi^{B} . (\Psi . S)^{t}$$

If we are interested in the transfer integral between the i^{th} orbital on A and the j^{th} orbital on B:

$$J^{AB}_{i,j} = \phi^{A}_{i,l}\, \epsilon_{l,k}\, \phi^{B}_{j,k}$$

Similar derivation to above yields:

$$J^{AB} = \left(\epsilon . \tilde{\phi}^{B}\right)^{t} . \tilde{\phi}^{A}$$

Implementation

The implementation consists of two scripts.

counterpoise.sh prepares the relevant Gaussian input files
ProJ.py contains the functions to compute the transfer integral

Counterpoise.sh

Takes three arguments : an xyz file with the coordinates of the first molecule , an xyz file with the coordinates of the second molecule , and a label for naming files

modifiy this script to chage memory usage, n processors, etc

The script does not start gaussian: you have to run the three jobs manually.

ProJ.py

To use ProJ:

1) sit in the root directory where the directories created by counterpoise are
2) open a python window
3) import ProJ
4) run ‘ProJ.CalcJ(*name*)‘ where *name* is the label defined in counterpoise

The output of ProJ are three arrays, respectively the transfer matrix, the site energies for the first molecule and the site energies for the second molecule.

I include a test directory with a simple case. Try it!