schneiderfelipe Always use np.einsum with optimize=True Latest commit 422313d on Aug 28, 2019 History

5 contributors

448 lines (448 sloc) 24.5 KB Raw Blame

In [1]:
```
"""Tutorial: Second-Order Moller--Plesset Perturbation Theory (MP2)"""

__author__   = "Dominic A. Sirianni"
__credit__   = ["Dominic A. Sirianni", "Daniel G. A. Smith"]

__copyright__ = "(c) 2014-2018, The Psi4NumPy Developers"
__license__   = "BSD-3-Clause"
__date__      = "2017-05-23"
```

Second-Order Moller-Plesset Perturbation Theory (MP2)

Moller-Plesset perturbation theory [also referred to as many-body perturbation theory (MBPT)] is an adaptation of the more general Rayleigh-Schrodinger perturbation theory (RSPT), applied to problems in molecular electronic structure theory. This tutorial will provide a brief overview of both RSPT and MBPT, before walking through an implementation of second-order Moller-Plesset perturbation theory (specifically referred to as MP2) which uses conventional, 4-index ERIs.

I. Overview of Rayleigh-Schrodinger Perturbation Theory

Given the Hamiltonian operator $\hat{f}_\beta$ for a system, perturbation theory solves the Schrodinger equation for that system by rewriting $\hat{f}_\beta$ as
$$ \hat{H}(\lambda) = \hat{H}^{(0)} + \lambda\hat{V}, \tag{Szabo:1996}, pp. 322, Eqn. 6.3) $$

were $\hat{f}_\beta^{(0)}$ is the Hamiltonian operator corresponding to a solved problem which resembles $\hat{f}_\beta$, and $\hat{V}$ is the *perturbation* operator, defined as $\hat{V} = \hat{H} - \hat{H}^{(0)}$. Then the Schrodinger equation for the system becomes
$$ \hat{H}\Psi_n = E_n\Psi_n \tag{Szabo:1996}, pp. 322, Eqn. 6.2) $$

The energies E_n and wavefunctions $\left|\Psi_n\right\rangle$ will both be functions of λ; they can therefore be written as a Taylor series expansion about $\lambda = 0$ ([Szabo:1996], pp. 322, Eqns. 6.4a & 6.4b):

$$\begin{aligned} E_n &= E_n^{(0)} + \lambda E_n^{(1)} + \lambda^2E_n^{(2)} + \ldots \tag{Szabo:1996}, pp. 322, Eqn. 6.4a)\\ \Psi_n &= \Psi_n^{(0)} + \lambda\Psi_n^{(1)} + \lambda^2\Psi_n^{(2)} + \ldots \tag{Szabo:1996}, pp. 322, Eqn. 6.4b) \end{aligned}$$

In practice, these perturbation expansions may be truncated to a given power of λ. Substituting the perturbation series above back into the Schrodinger equation yields

$$(\hat{H}^{(0)} + \lambda\hat{V})(\left|\Psi_n^{(0)}\right\rangle + \lambda\left|\Psi_n^{(1)}\right\rangle + \lambda^2\left|\Psi_n^{(2)}\right\rangle + \ldots) = (E_n^{(0)} + \lambda E_n^{(1)} + \lambda^2E_n^{(2)} + \ldots)(\left|\Psi_n^{(0)}\right\rangle + \lambda\left|\Psi_n^{(1)}\right\rangle + \lambda^2\left|\Psi_n^{(2)}\right\rangle + \ldots),$$

which by equating powers of λ ([Szabo:1996], pp. 323, Eqns. 6.7a-6.7d) gives expressions for the $E_n^{(n)}$and $\left|\Psi_n^{(n)}\right\rangle$. Note that for λ^0, $E_n^{(0)}$ and $\left|\Psi_n^{(0)}\right\rangle$ are known, as they are the solution to the zeroth-order problem $\hat{f}_\beta^{(0)}$. For λ^1 and λ^2, the expressions for $E_n^{(1)}$ and $E_n^{(2)}$ are given by

$$\begin{aligned} \lambda^1 &: E_n^{(1)} = \left\langle\Psi_n^{(0)}\middle|\hat{V}\middle|\Psi_n^{(0)}\right\rangle \tag{Szabo:1996}, pp. 323, Eqn. 6.8b)\\ \lambda^2 &: E_n^{(2)} = \sum_{m\neq n}\frac{\left|\left\langle\Psi_m^{(0)}\middle|\hat{V}\middle|\Psi_n^{(0)}\right\rangle\right|^2}{E_n^{(0)} - E_m^{(0)}} \tag{Szabo:1996}, pp. 324, Eqn. 6.12) \end{aligned}$$

II. Overview of Moller-Plesset Perturbation Theory

The exact electronic Hamiltonian for an N-electron molecule with a given atomic configuration is (in atomic units):
$$ \hat{H}_{elec} = \sum_i\hat{h}(i) + \sum_{i>j}\frac{1}{\hat{r}_{ij}} \tag{Szabo:1996}, pp. 43, Eqn. 2.10) $$

Moller-Plesset perturbation theory seeks to solve the *molecular* electronic Schrodinger equation with the above Hamiltonian using the techniques of Rayleigh-Schroding perturbation theory, by selecting the zeroth-order Hamiltonian and wavefunctions to be those from Hartree-Fock theory:

$$\begin{aligned} \hat{H}^{(0)}(\lambda) &= \sum_i \hat{h}(i) \\ \hat{H}^{(1)}(\lambda) &= \sum_{i>j}\frac{1}{\hat{r}_{ij}} - \sum_i\hat{v}^{HF}(i) \end{aligned}$$ (Szabo:1996), pp. 350, Eqn. 6.59)

$$\hat{V} = \hat{H} - \hat{H}^{(0)} = \sum_{i>j}\frac{1}{\hat{r}_{ij}} - \sum_i\hat{v}^{HF}(i)$$ (Szabo:1996), pp. 350, Eqn. 6.60)

With these choices of $\hat{f}_\beta^{(0)}$, $\hat{V}$, and $\left|\Psi_n^{(0)}\right\rangle$, the ground-state zeroth-order energy is given by $E_0^{(0)} = \sum_i\epsilon_i$ (Szabo:1996), pp. 351, Eqn. 6.67). Then, the first-order ground state energy may be computed by $E_0^{(1)} = \left\langle\Psi_0^{(0)}\middle|\hat{V}\middle|\Psi_0^{(0)}\right\rangle$ to find that the total ground-state energy through first order, $E_0 = E_0^{(0)} + E_0^{(1)}$, is exactly the Hartree-Fock energy. ([Szabo:1996], pp. 351, Eqn. 6.69) Therefore, the first correction to the Hartree-Fock energy will occur at second-order in the perturbation series, i.e., with $E_0^{(2)}$; truncating the perturbation series at second order is commonly referred to as MP2. The second order correction to the ground state energy will be given by

$$ E_0^{(2)} = \sum_{m\neq 0}\frac{\left|\left\langle\Psi_0^{(0)}\middle|\hat{V}\middle|\Psi_m^{(0)}\right\rangle\right|^2}{E_0^{(0)} - E_m^{(0)}} $$ ([Szabo:1996], pp. 351, Eqn. 6.70) **$$**

For brevity, we will now drop the "HF" from all zeroth-order wavefunctions. This summation is over the eigenstate index $\neq$, each associated with a different eigenstate of the zeroth-order Hamiltonian. For a single-determinant wavefunction constructed from spin orbitals, the summation over the eigenstate index $\neq 0$ therefore refers to determinants which are constructed from different spin orbitals than the ground state determinant. To distinguish such determinants, we will denote MOs occupied in the ground state with indices i, j, ..., and MOs which are unoccupied in the ground state with indices a, b, ... Then a determinant where orbital i is substituted for orbital j is denoted $\left|\Psi_{\ldots}^{\ldots}\right\rangle$, and so on. Before substituting this new notation into the above energy expression, however, we may immediately recognize that many terms $\left\langle\Psi_m\middle|\hat{V}\middle|\Psi_n\right\rangle$ will not contribute to the second order energy:

Term	Determinant	Contribution to $E_0^{(2)}$
Singles	$\left\|\Psi_i^a\right\rangle$	0; Brillouin's Theorem
Doubles	$\left\|\Psi_{ij}^{ab}\right\rangle$	Survive
Higher-order	$\left\|\Psi_{ijk\ldots}^{abc\ldots}\right\rangle$	0; $\hat{V}$ is a two-particle operator

Hence we see that only doubly-substituted determinants $\left|\Psi_{ij}^{ab}\right\rangle$ will contribute to $E_0^{(2)}$. From Hartree-Fock theory, we know that
$$ \langle\Psi_0|\hat{H}_0|\Psi_0\rangle = \sum_i\epsilon_i \tag{Szabo:1996}, pp. 72, Thl. 2.2) $$

where $|ia\ ||\ jb| = |ia\ |\ jb| - |ib\ |\ ja|$ is an antisymmetrized two-electron integral, and the square brackets $\langle\ldots|\ldots\rangle$ indicate that we are employing chemists' notation. What about the energies of these doubly substituted determinants? Recognizing that the difference between the energies of the newly- and formerly-occupied orbitals in each substitution must modify the total energy of the ground state determinant,

$$ E_{ij}^{ab} = E_0 - (\epsilon_i - \epsilon_a + \epsilon_j - \epsilon_b). $$

Substituting these expressions into the one for the second-order energy, we have that
$$ E_0^{(2)} = \sum_{i<j}\sum_{a<b}\frac{\left|\left\langle\Psi_{ij}^{ab}\middle|\hat{V}\middle|\Psi_0\right\rangle\right|^2}{\epsilon_i+\epsilon_a+\epsilon_j+\epsilon_b} $$ (Szabo:1996), pp. 352, Eqn. 6.74) Note that an exchange integral arises between particles of the same spin; this is because the motions of particles with identical spins are correlated due to the requirement that $|\Psi|^2$ remain invariant to the exchange of the spatial and spin coordinates of any pair of electrons. Finally, the total MP2 correction energy $E_0^{(2)} = E_{0,\ ss}^{(2)} + E_{0,\ os}^{(2)}$.

Implementation of Conventional MP2

Let's begin by importing Psi4 and NumPy, and setting memory and output file options:

In [2]:
```
# ==> Import statements & Global Options <==
import psi4
import numpy as np

psi4.set_memory(int(2e9))
numpy_memory = 2
psi4.core.set_output_file('output.dat', False)
```

Next, we can define our molecule and Psi4 options. Notice that we are using `scf_type pk` to indicate that we wish to use conventional, full 4-index ERIs, and that we have specified `mp2_type conv` so that the MP2 algorithm we check against also uses the conventional ERIs.

In [3]:
```
# ==> Molecule & Psi4 Options Definitions <==
mol = psi4.geometry("""
O
H 1 1.1
H 1 1.1 2 104
symmetry c1
""")

psi4.set_options({'basis':        '6-31g',
                  'scf_type':     'pk',
                  'mp2_type':     'conv',
                  'e_convergence': 1e-8,
                  'd_convergence': 1e-8})
```

Since MP2 is a perturbation on the zeroth-order Hartree-Fock description of a molecular system, all of the relevant information (Fock matrix, orbitals, orbital energies) about the system can be computed using any Hartree-Fock program. We could use the RHF program that we wrote in tutorial 3a, but we could just as easily use Psi4 to do our dirty work. In the cell below, use Psi4 to compute the RHF energy and wavefunction, and store them using the `return_wfn=True` keyword argument to `psi4.energy()`:

In [4]:
```
# Get the SCF wavefunction & energies
scf_e, scf_wfn = psi4.energy('scf', return_wfn=True)
```

In the expression for $E_0^{(2)}$, the two summations are over occupied and virtual indices, respectively. Therefore, we'll need to get the number of occupied orbitals and the total number of orbitals. Additionally, we must obtain the MO energy eigenvalues; again since the sums are over occupied and virtual orbitals, it is good to separate the occupied orbital energies from the virtual orbital energies. From the SCF wavefunction you generated above, get the number of doubly occupied orbitals, number of molecular orbitals, and MO energies:

In [5]:
```
# ==> Get orbital information & energy eigenvalues <==
# Number of Occupied orbitals & MOs
ndocc = scf_wfn.nalpha()
nmo = scf_wfn.nmo()

# Get orbital energies, cast into NumPy array, and separate occupied & virtual
eps = np.asarray(scf_wfn.epsilon_a())
e_i = eps[:ndocc]
e_ab = eps[ndocc:]
```

Unlike the orbital information, Psi4 does not return the ERIs when it does a computation. Fortunately, however, we can just build them again using the `psi4.core.MintsHelper()` class. Recall that these integrals will be generated in the AO basis; before using them in the $E_0^{(2)}$ expression, we must transform them into the MO basis. To do this, we first need to obtain the orbital coefficient matrix, **C**. In the cell below, generate the ERIs for our molecule, get **C** from the SCF wavefunction, and obtain occupied- and virtual-orbital slices of **C** for future use.

In [6]:
```
# ==> ERIs <==
# Create instance of MintsHelper class
mints = psi4.core.MintsHelper(scf_wfn.basisset())

# Memory check for ERI tensor
I_size = (nmo**4) * 8.0e-9
print('\nSize of the ERI tensor will be %4.2f GB.' % I_size)
memory_footprint = I_size * 1.5
if I_size > numpy_memory:
    psi4.core.clean()
    raise Exception("Estimated memory utilization (%4.2f GB) exceeds allotted memory \
                     limit of %4.2f GB." % (memory_footprint, numpy_memory))

# Build ERI Tensor
I = np.asarray(mints.ao_eri())

# Get MO coefficients from SCF wavefunction
C = np.asarray(scf_wfn.Ca())
Cocc = C[:, :ndocc]
Cvirt = C[:, ndocc:]
```

```
Size of the ERI tensor will be 0.00 GB.
```

In order to transform the four-index integrals from the AO to the MO basis, we must perform the following contraction:

$$ (ia\ ||\ jb) = C_{\mu}C_{\nu}(\mu\ \nu\ ||\ \lambda\ \sigma)C_{\lambda}C_{\sigma} $$

Again, here we are using i, j as occupied orbital indices and a, b as virtual orbital indices. We could carry out the above contraction all in one step using either `np.einsum()` or explicit loops:

```
    # Naive Algorithm for ERI Transformation
    Imo = np.einsum('pi,qa,prs,rj,sb->iajb', Cocc, Cvirt, I, Cocc, Cvirt, optimize=True)
```

Notice that the transformation from AO index to occupied (virtual) MO index requires only the occupied (virtual) block of the **C** matrix; this allows for computational savings in large basis sets, where the virtual space can be very large. This algorithm, while simple, has the distinct disadvantage of horrendous scaling. Examining the contraction *more closely*, we see that there are 8 unique indices, and thus the step above scales as $\mathcal{O}(N^8)$. With this algorithm, a twofold increase of the number of MO's would result in $2^8 = 256\times$ expense to perform. We can, however, refactor the above contraction such that

$$ (ia\ ||\ jb) = \Big[C_{\mu}\Big[C_{\nu}\big[C_{\sigma}\big[C_{\lambda}(\mu\ \nu\ ||\ \lambda\ \sigma)\big]\big]\Big]\Big], $$

where we have now written the transformation as four $\mathcal{O}(N^5)$ steps instead of one $\mathcal{O}(N^8)$ step. This is a savings of $\frac{1}{N^3}$, and is responsible for the feasibility of the MP2 method for application to any but very small systems and/or basis sets. We may carry out the above $\mathcal{O}(N^5)$ algorithm by carrying out one index transformation at a time, and storing the result in a temporary array. In the cell below, transform the ERIs from the AO to MO basis, using our smarter algorithm:

In [7]:
```
# ==> Transform I -> I_mo @ O(N^5) <==
tmp = np.einsum('pi,pqrs->iqrs', Cocc, I, optimize=True)
tmp = np.einsum('qa,iqrs->iars', Cvirt, tmp, optimize=True)
tmp = np.einsum('iars,rj->iajs', tmp, Cocc, optimize=True)
I_mo = np.einsum('iajs,sb->iajb', tmp, Cvirt, optimize=True)
```

We note here that we can use infrastructure in Psi4 to carry out the above integral transformation; this entails obtaining the occupied and virtual blocks of **C** Psi4-side, and then using the built-in `MintsHelper` function `MintsHelper.mo_eri()` to transform the integrals. Just to check your work above, execute the next cell to see this tech in action:

In [8]:
```
# ==> Compare our Imo to MintsHelper <==
Co = scf_wfn.Ca_subset('AO','OCC')
Cv = scf_wfn.Ca_subset('AO','VIR')
MO = np.asarray(mints.mo_eri(Co, Cv, Co, Cv))
print('Do our transformed ERIs match Psi4\'s? %s' % np.allclose(I_mo, np.asarray(MO)))
```

```
Do our transformed ERIs match Psi4's? True
```

Now we have all the pieces needed to compute $E_0^{(2)}$. This could be done by writing explicit loops over occupied and virtual indices Python side, e.g.,

```
# Compute SS MP2 Correlation
mp2_ss_corr = 0.0
for i in xrange(ndocc):
    for a in xrange(nmo - ndocc):
        for j in xrange(ndocc):
            for b in xrange(nmo - ndocc):
                numerator = I_mo[i,a,j,b] - I_mo[i, b, j, a]
                mp2_ss_corr += numerator / (e_i[i] + e_i[j] - e_ab[a] - e_ab[b])
```

This method, while very clear what is going on and easy to program, has the distinct disadvantage that Python loops are much slower than the same block written in a compiled language like C, C++, or Fortran. For this reason, it is better to use infrastructure available in NumPy like `np.einsum()`, `np.dot`, etc. to explicitly compute the above quantity C-side. It should be clear how to contract the four-index integrals $(ia\ ||\ jb)$ and $(ia\ |\ jb)$ with one another, but what about the energy denominators? We can use a NumPy trick called *broadcasting* to construct a four-index array of all possible energy denominators, which can then be contracted with the full I_{mo} arrays. To do this, we'll use the function `np.reshape()`:

```
# Prepare 4d energy denominator array
e_denom = e_i3.reshape(-1, 1, 1, 1) # Diagonal of 4d array are occupied orbital energies
e_denom -= e_ab.reshape(-1, 1, 1)   # all combinations of (e_i) - e_ab)
e_denom -= e_i1.reshape(-1, 1)      # all combinations of (e_i) - e_ab) + e_ij]
e_denom -= e_ab                     # All combinations of full denominator
e_denom = 1 / e_denom               # take reciprocal for contracting with numerator
```

In the cell below, compute the energy denominator using `np.reshape()` and contract this array with the four-index ERIs to compute the same-spin and opposite-spin MP2 correction using `np.einsum()`. Then, add these quantities to the SCF energy computed above to obtain the total MP2 energy.

Hint: For the opposite-spin correlation, use `np.swapaxes()` to obtain the correct ordering of the indices in the exchange integral.

In [9]:
```
# ==> Compute MP2 Correlation & MP2 Energy <==
# Compute energy denominator array
e_denom = 1 / (e_i.reshape(-1, 1, 1, 1) - e_ab.reshape(-1, 1) + e_ij.reshape(-1, 1) - e_ab)

# Compute SS & OS MP2 Correlation with Einsum
mp2_os_corr = np.einsum('iajb,iajb,iajb->', I_mo, I_mo, e_denom, optimize=True)
mp2_ss_corr = np.einsum('iajb,iajb,iajb->', I_mo, I_mo - I_mo.swapaxes(1,3), e_denom, optimize=True)

# Total MP2 Energy
MP2_E = scf_e + mp2_os_corr + mp2_ss_corr
```

In [10]:
```
# ==> Compare to Psi4 <==
psi4.compare_values(psi4.energy('mp2'), MP2_E, 6, 'MP2 Energy')
```

```
    MP2 Energy..............................................................PASSED
```

Out[10]:
```
True
```

References

1. Original paper: "Note on an Approximation Treatment for Many-Electron Systems"

   [Moller:1934.618] C. Møller and M. S. Plesset, *Phys. Rev.* **46**, 618 (1934)

2. The Laplace-transformation in MP theory: "Minimax approximation for the decomposition of energy denominators in Laplace-transformed Møller–Plesset perturbation theories"

   [Takatsuka:2008.044112] A. Takatsuka, T. Siichiro, and W. Hackbusch, *J. Phys. Chem.*, **129**, 044112 (2008)

3. Equations taken from:

   [Szabo:1996] A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*, Courier Corporation, 1996.

4. Algorithms taken from:

   [CrawfordLprog] T. D. Crawford, "The Second-Order Møller–Plesset Perturbation Theory (MP2) Energy." Accessed via the web at http://github.com/CrawfordGroup/ProgrammingProjects.