Daily Notes

王石嵘
wsr@smail.nju.edu.cn

June 25, 2019

Contents

1 06/23

1.1 PySCF dft routine

in `dft.rks`

```python
def get_veff(ks, mol=None, dm=None, dm_last=0, vhf_last=0, hermi=1):
    ...
    n, exc, vxc = ks._numint.nr_rks(mol, ks.grids, ks.xc, dm)
    # add vj,ecoul
    # add vk, HF-X if hyb
```

in `dft.numint`

```python
def nr_rks(ni, mol, grids, xc_code, dms, relativity=0, hermi=0, max_memory=2000, verbose=None)
    :
    ...
    xctype = ni._xc_type(xc_code)
    make_rho, nset, nao = ni._gen_rho_evaluator(mol, dms, hermi)

    shls_slice = (0, mol.nbas)
    ao_loc = mol.ao_loc_nr()

    nelec = numpy.zeros(nset)
    excsum = numpy.zeros(nset)
    vmat = numpy.zeros((nset,nao,nao))
    aow = None
    if xctype == 'LDA':
        ao_deriv = 0
        for ao, mask, weight, coords in ni.block_loop(mol, grids, nao, ao_deriv, max_memory):
            aow = numpy.ndarray(ao.shape, order='F', buffer=aow)
            for idm in range(nset):
                rho = make_rho(idm, ao, mask, 'LDA')
                exc, vxc = ni.eval_xc(xc_code, rho, 0, relativity, 1, verbose)[:2]
                vrho = vxc[0]
                den = rho * weight
                nelec[idm] += den.sum()
                excsum[idm] += (den * exc).sum()  # E_xc
                # *.5 because vmat + vmat.T
                aow = numpy.einsum('pi,p->pi', ao, .5*weight*vrho, out=aow)
                vmat[idm] += _dot_ao_ao(mol, ao, aow, mask, shls_slice, ao_loc)
                rho = exc = vxc = vrho = None
```

thus

$$E_{xc} = \sum_i \varepsilon_{xc}(\mathbf{r}_i)\rho(\mathbf{r}_i)w(\mathbf{r}_i) \tag{1}$$

But how we allocate `exc, rho, weight` to each atom?

Let's try `dft.gen_grid`

```python
class Grids(lib.StreamObject):
    def build(self, mol=None, with_non0tab=False):
        atom_grids_tab = self.gen_atomic_grids(mol, self.atom_grid, self.radi_method, self.
    level, self.prune)
        self.coords, self.weights = self.gen_partition(mol, atom_grids_tab, self.radii_adjust,
     self.atomic_radii, self.becke_scheme)
        if with_non0tab:
            self.non0tab = self.make_mask(mol, self.coords)
        else:
            self.non0tab = None
        return self.coords, self.weights
```

```python
def gen_partition(mol, atom_grids_tab, radii_adjust=None, atomic_radii=radi.BRAGG_RADII,
                  becke_scheme=original_becke):
    atm_coords = numpy.asarray(mol.atom_coords() , order='C')
    atm_dist = radi._inter_distance(mol)
    #  if default settings
    def gen_grid_partition(coords):
        coords = numpy.asarray(coords, order='F')
        ngrids = coords.shape[0]
        pbecke = numpy.empty((mol.natm,ngrids))
        libdft.VXCgen_grid(pbecke.ctypes.data_as(ctypes.c_void_p),
                           coords.ctypes.data_as(ctypes.c_void_p),
                           atm_coords.ctypes.data_as(ctypes.c_void_p),
                           p_radii_table,
                           ctypes.c_int(mol.natm), ctypes.c_int(ngrids))
        return pbecke
    coords_all = []
    weights_all = []
    for ia in range(mol.natm):
        coords, vol = atom_grids_tab[mol.atom_symbol(ia)] # grid coords wrt atom coord
        coords = coords + atm_coords[ia] # get real coords
        pbecke = gen_grid_partition(coords) # do becke partition.
        weights = vol * pbecke[ia] * (1./pbecke.sum(axis=0))
        coords_all.append(coords)
        weights_all.append(weights)
    return numpy.vstack(coords_all), numpy.hstack(weights_all)
```

Actually,

$$E_{xc} = \sum_i \varepsilon_{xc}(\mathbf{r}_i)\rho(\mathbf{r}_i)V_iW_i \tag{2}$$

$$W_i = \frac{P_{i,A}}{\sum_A P_{i,A}} \tag{3}$$

where W_i measures the extent to which a grid point belongs to some atom A, in Becke partition. $W_i = 1$ when close to only one atom, and $W_i = 0 \sim 1$ in other cases (i.e. shared). V_i is `vol` above.