

QE FFT FOR BCC HYDROGEN

Fourier Basis

For bcc lattice with $r_s = 1.31$, the lattice constant is $a = 2.66$ bohr. I used a wave function plane wave cutoff of 150 Ry. QE defaults to using a (24,24,24) FFT grid. The following code in QE generates the lowest-frequency plane wave in the x-direction.

```
nrxxs= dffts%nnr ! = nr1s*nr2s*nr3s
ALLOCATE( jastrow(nrxxs) )
jastrow(:)=(0.0_DP,0.0_DP)

print *,g(:,5) ! 0,0,0.376
jastrow(nls(5)) = 1.
CALL invfft ('Wave', jastrow, dffts)
```

The corresponding g-vector at index 5 is $(0,0,1/a)$. A slice along x of `jastrow`, which contains the lowest-frequency plane wave in real space is shown in left plot of Fig. 1. Setting the counter basis coefficient to 1 produces a cos as expected (right plot of Fig. 1).

```
print *,g(:,4) ! 0,0,-0.376
jastrow(nls(4)) = 1.
jastrow(nls(5)) = 1.
CALL invfft ('Wave', jastrow, dffts)
```

This confirms that `invfft` in QE follows the same convention as `np.fft.invfft`. The question remains as to which reciprocal lattice vectors actually contribute to the `invfft('Wave',...)` routine. **WARNING!** Apply `np.fft.fftshift` to the entire FFT grid `jas[:, :, :, 0]` and `jas[:, :, :, 1]`, NOT individual slices `jas[:, 12, 12, 0]`.

Fig. 1 shows that the basis functions used in `invfft` are NOT normalized over the

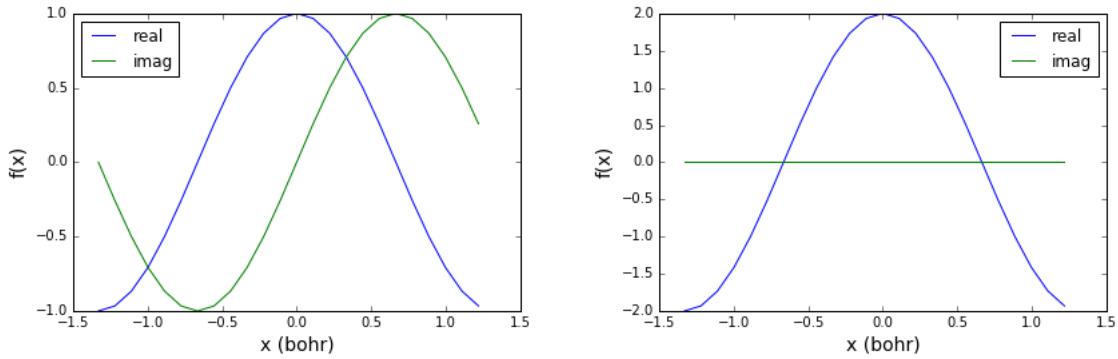


FIG. 1. FFT basis

supercell. To compare to analytical forms of real space functions, one should normalize by the volume of the supercell Ω . See the next section (**3D Gaussian**) for an example.

3D Gaussian

The following QE code should generate a Gaussian with width 1: $G(\mathbf{r}) = \frac{1}{(2\pi)^{3/2}} e^{-r^2/2}$.

```

do ng = 1,ngm
    q2 = sum( ( g(:,ng) )**2 )*tpiba2
    jastrow(nls(ng)) = EXP(-q2/2.)
enddo
CALL invfft ('Wave', jastrow, dffts)

```

This can be checked with `np.fft` as follows (`nx=24, alat=2.66, omega=18.83`)

```

kgrid = np.zeros([nx,nx,nx])
for kx in np.arange(-nx/2,nx/2):
    for ky in np.arange(-nx/2,nx/2):
        for kz in np.arange(-nx/2,nx/2):
            kvec = 2*np.pi/alat * np.array([kx,ky,kz])
            kgrid[kx,ky,kz] = np.exp(-np.dot(kvec,kvec)/2.)

```

```

# end for

# end for

# end for

fr = np.fft.fftshift( np.fft.ifftn(kgrid) ) * nx**3.

```

As shown in the left plot of Fig. 2, QE and Python results match. They both differ from the true Gaussian because of the limited spatial extend of the FFT grid, which forces periodicity. If the supercell size (i.e. spatial extend of FFT grid) is extended, then the correct Gaussian can be obtained as shown in the right plot of Fig. 2.

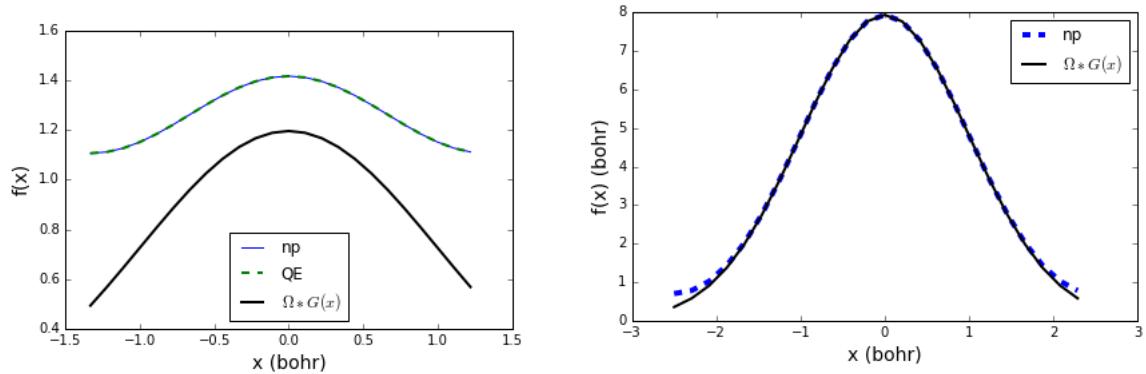


FIG. 2. Gaussian.

RPA Jastrow

Given current understanding of Fourier transform, the RPA Jastrow can be constructed with the python code in the appendix **Construct RPA Jastrow in Python**. The resulting “quasi potential” has a cusp of about 1, which is independent of r_s as shown in Fig. 3. I calculated the cusp by taking the slope of the closest two grid points to the cusp. The value of this slope approaches 1 as the FFT grid becomes denser. Notice that no normalization by Ω was applied. This is probably due to different conventions used to construct the RPA Jastrow plane wave coefficients.

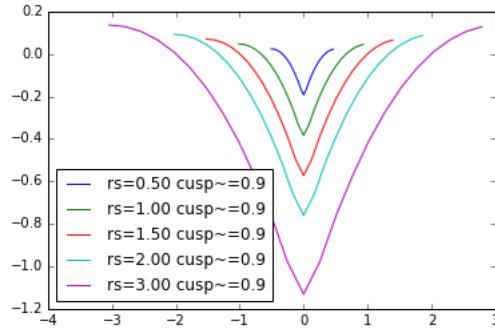


FIG. 3. RPA Jastrow cusp is independent of r_s .

Delta Function

Setting the coefficients of all available plane waves to 1 should produce the best approximation to the Dirac delta function. Initializing the entire FFT grid in QE to 1 gives the delta function shown in left plot of Fig. 4. However, initializing the entire FFT grid to 1 in Python gives the right plot in Fig. 4. The functions are off by a factor of 4.465.

```
jastrow(:)=(1.0_DP,0.0_DP)
```

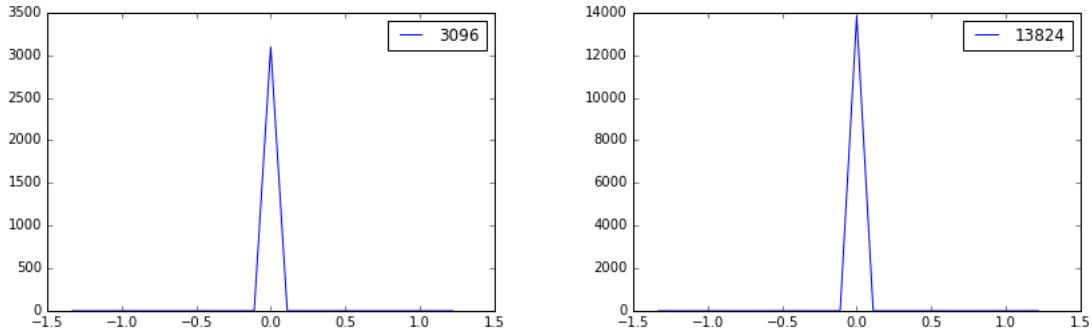


FIG. 4. Initialize entire FFT grid to 1.

Fig. 4 implies that QE does not use all of the $24^3 = 13824$ plane waves in `invfft`.

Fig. 5 and 6 show that only plane waves in the x and y directions are cutoff by

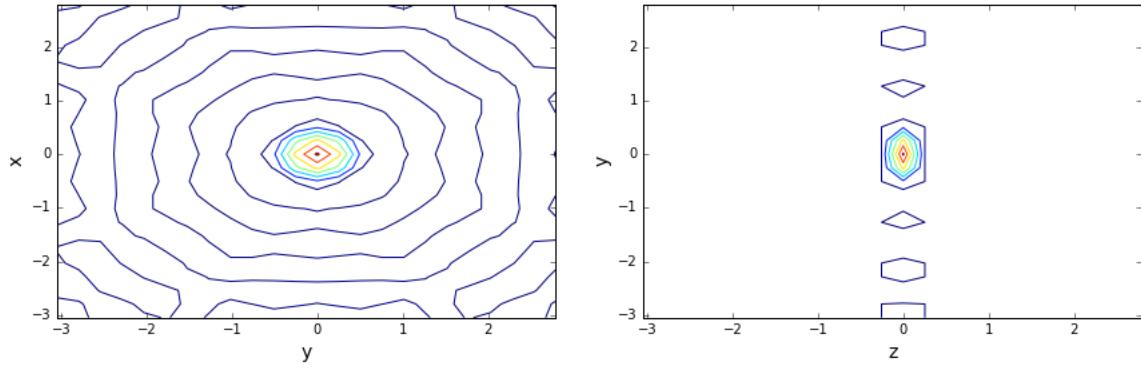


FIG. 5. Slices of QE delta function.

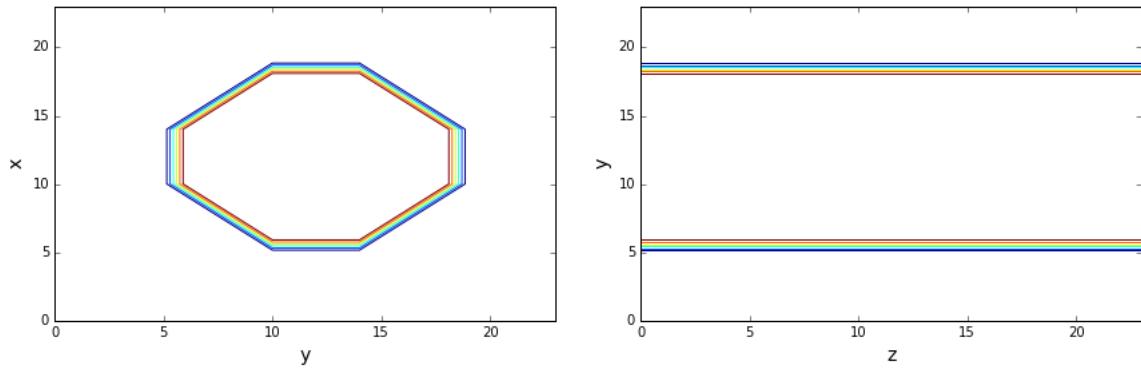


FIG. 6. Slices of QE delta function in reciprocal space.

`ecut=150Ry`, whereas all plane waves in the `z` direction are used. Indeed, using the following cutoff scheme in Python reproduces the QE delta function.

```

cut = 15.

for kx in np.arange(-nx/2,nx/2):
    for ky in np.arange(-nx/2,nx/2):
        for kz in np.arange(-nx/2,nx/2):
            kvec = (2*np.pi/alat) * np.array([kx,ky,kz])
            if np.dot(kvec[1:],kvec[1:]) < cut**2.:
                kgrid[kx,ky,kz] = 1.

# end if

# end for kz

```

```

# end for ky
# end for kx

```

Appendix: Construct RPA Jastrow in QE

```

if (cusp_corr) then

RS1 = (3.0_DP*omega/(4.0_DP*pi*pi*nelec))**((1.0_DP/3.0_DP))

ALLOCATE( jastrow(nrxxs) )
jastrow(:)=(0.0_DP,0.0_DP)

! Construct RPA Jastrow:
! This is specific to hydrogen right now
do ng = 1, ngm
    q2 = sum( ( g(:,ng) )**2 )*tpiba2
    IF(ABS(q2) < 0.000001d0) CYCLE
    sf0 = (0.0_DP,0.0_DP)
    do na = 1, nat
        arg = (g (1, ng) * tau (1, na) + g (2, ng) * tau (2, na) &
               + g (3, ng) * tau (3, na) ) * tpi
        sf0= sf0 + CMPLX(cos (arg), -sin (arg),kind=DP)
    enddo
    temp = 12.0_DP/(RS1*RS1*RS1*q2*q2)
    uep = -0.5_DP*temp/SQRT(1.0_DP + temp)
    jastrow(nls(ng)) = sf0 * uep / nelec
    enddo
    CALL invfft ('Wave', jastrow, dffts)
    do ik=1,nrxxs

```

```

jastrow(ik)=CDEXP(-jastrow(ik))

enddo

endif ! cusp_corr

```

Appendix: Construct RPA Jastrow in Python

```

import numpy as np
import matplotlib.pyplot as plt

def rs2a(rs,atoms_per_unit_cell):
    space_per_atom = 4.*np.pi/3*rs**3.
    cell_volume      = space_per_atom*atoms_per_unit_cell
    alat = cell_volume**(1./3)
    return alat

# end def rs2a

# RPA Jastrow
# =====
def lattice_factor(gvec,basis):
    factor = 0.
    for Rb in basis:
        factor += np.exp(1j*np.dot(gvec,Rb))
    # end for
    return factor

# end def

def kUep(gvec,rs):
    # ref: Ceperley and Alder, Physica 108B (1981)
    ak = 12./(np.linalg.norm(gvec)**4. *rs**3.)

```

```

    return -0.5*ak/np.sqrt( 1+ak )

# end def

def kUep_lattice(gvec,basis,ncell,rs=1.31):
    # ref: Natoli Thesis *2 (Natoli kUep shoud /2)
    natom = len(basis)
    factor = lattice_factor(gvec,basis)
    kUep_prime = factor * kUep(gvec,rs)
    return kUep_prime*ncell/natom

# end def

nx      = 24
atom_per_cell = 2

# check that RPA Jastrow cusp is independent of rs
for rs in [0.5,1.0,1.5,2.0,3.0]:

    alat = rs2a(rs,atom_per_cell)
    basis = [[0,0,0],[alat/2,alat/2,alat/2]]
    x = np.arange(-alat/2,alat/2,alat/nx)

    # fill FFT grid
    kgrid = np.zeros([nx,nx,nx],dtype=complex)
    for kx in np.arange(-nx/2,nx/2):
        for ky in np.arange(-nx/2,nx/2):
            for kz in np.arange(-nx/2,nx/2):
                kvec = 2*np.pi/alat * np.array([kx,ky,kz])
                if np.allclose(kvec,[0,0,0]):
                    continue
                # end if
                kgrid[kx,ky,kz] = kUep_lattice(kvec,basis,1,rs)

```

```
# end for

# end for

# end for

fr = np.fft.fftshift( np.fft.ifftn(kgrid) ) * nx**3.

y = fr[nx/2:nx/2,:].real

cusp = (y[nx/2+2] - y[nx/2+1]) / (x[1]-x[0])

plt.plot(x,y,label="rs=%1.2f cusp~=%1.1f"%(rs,cusp))

# end for rs

plt.legend(loc="best")

plt.tight_layout()

plt.savefig("rpa_jastrow_cusp_vs_rs.png")
```