Annotated PW codes (more or less ...)

Fadjar Fathurrahman

1 PWGrid version 1

1.1 The implementation: `PWGrid_01.jl`

The type definition

```
type PWGrid
  Ns::Array{Int64}
  LatVecs::Array{Float64,2}
  RecVecs::Array{Float64,2}
  Npoints::Int
  Ω::Float64
  r::Array{Float64,2}
  G::Array{Float64,2}
  G2::Array{Float64}
end
```

The constructor

```
function PWGrid( Ns::Array{Int,1},LatVecs::Array{Float64,2} )
  Npoints = prod(Ns)
  RecVecs = 2*pi*inv(LatVecs')
  Ω = det(LatVecs)
  R,G,G2 = init_grids( Ns, LatVecs, RecVecs )
  return PWGrid( Ns, LatVecs, RecVecs, Npoints, Ω, R, G, G2 )
end
```

Mapping between half of the sampling points to the negative ones

```
function mm_to_nn(mm::Int,S::Int)
  if mm > S/2
    return mm - S
  else
    return mm
  end
end
```

Initialization of real space and reciprocal space grid points

```
function init_grids( Ns, LatVecs, RecVecs )
  Npoints = prod(Ns)
  r = Array(Float64,3,Npoints)
  ip = 0
  for k in 0:Ns[3]-1
  for j in 0:Ns[2]-1
  for i in 0:Ns[1]-1
    ip = ip + 1
    r[1,ip] = LatVecs[1,1]*i/Ns[1] + LatVecs[2,1]*j/Ns[2] + LatVecs[3,1]*k/Ns[3]
    r[2,ip] = LatVecs[1,2]*i/Ns[1] + LatVecs[2,2]*j/Ns[2] + LatVecs[3,2]*k/Ns[3]
    r[3,ip] = LatVecs[1,3]*i/Ns[1] + LatVecs[2,3]*j/Ns[2] + LatVecs[3,3]*k/Ns[3]
  end
  end
```

```
  end
  #
  G  = Array(Float64,3,Npoints)
  G2 = Array(Float64,Npoints)
  ip    = 0
  for k in 0:Ns[3]-1
  for j in 0:Ns[2]-1
  for i in 0:Ns[1]-1
    gi = mm_to_nn( i, Ns[1] )
    gj = mm_to_nn( j, Ns[2] )
    gk = mm_to_nn( k, Ns[3] )
    ip = ip + 1
    G[1,ip] = RecVecs[1,1]*gi + RecVecs[2,1]*gj + RecVecs[3,1]*gk
    G[2,ip] = RecVecs[1,2]*gi + RecVecs[2,2]*gj + RecVecs[3,2]*gk
    G[3,ip] = RecVecs[1,3]*gi + RecVecs[2,3]*gj + RecVecs[3,3]*gk
    G2[ip] = G[1,ip]^2 + G[2,ip]^2 + G[3,ip]^2
  end
  end
  end
  return r,G,G2
end
```

1.2 Testing the `PWGrid_01`

Include files:

```
include("../common/PWGrid_v01.jl")
include("../common/gen_lattice.jl")
```

Main driver

```
function test_main_hexagonal()
  # call these functions for other types of lattice
  #LL = 16.0*diagm([1.0, 1.0, 1.0]) # cubic
  #LL = gen_lattice_fcc(16.0)       # face-centered cubic
  #LL = gen_lattice_bcc(16.0)       # body-centered cubic

  Ns = [10, 10, 20]
  LL = gen_lattice_hexagonal(10.0, coa=2.0)
  pw = PWGrid( Ns, LL )
  atpos = pw.r

  println(pw.LatVecs)
  println(pw.RecVecs)

  write_XSF("R_grid_hexagonal.xsf", LL, atpos)

  Rec = pw.RecVecs*Ns[1]/2.0
  atpos = pw.G
  write_XSF("G_grid_hexagonal.xsf", LL, atpos, molecule=true)

  for ii = 1:3
    @printf("LatVecLen %d %18.10f\n", ii, norm(pw.LatVecs[ii,:]))
  end
  @printf("Ratio coa: %18.10f\n", norm(pw.LatVecs[1,:])/norm(pw.LatVecs[3,:]))

  @printf("\n")
  for ii = 1:3
    @printf("RecVecLen %d %18.10f\n", ii, norm(pw.RecVecs[ii,:]))
  end
```

```
  @printf("Ratio coa: %18.10f\n", norm(pw.RecVecs[1,:])/norm(pw.RecVecs[3,:]))

end
```

2 Solving Poisson equation

```
include("../common/PWGrid_v01.jl")
#include("../common/wrappers_fft_v01.jl")
include("../common/wrappers_fft.jl")

include("gen_dr.jl")
include("gen_rho.jl")
include("solve_poisson.jl")

function test_main()
  #
  const Ns = [64, 64, 64]
  const LatVecs = 16.0*diagm( ones(3) )
  #
  pw = PWGrid( Ns, LatVecs )
  #
  const Npoints = pw.Npoints
  const Ω = pw.Ω
  const r = pw.r
  const Ns = pw.Ns
  #
  # Generate array of distances
  #
  center = sum(LatVecs,2)/2
  dr = gen_dr( r, center )
  #
  # Generate charge density
  #
  const σ1 = 0.75
  const σ2 = 0.50
  rho = gen_rho( dr, σ1, σ2 )
  #
  # Solve Poisson equation and calculate Hartree energy
  #
  phi = solve_poisson( pw, rho )
  Ehartree = 0.5*dot( phi, rho ) * Ω/Npoints
  #
  Uanal = ( (1/σ1 + 1/σ2)/2 - sqrt(2) / sqrt( σ1^2 + σ2^2 ) ) / sqrt(pi)
  @printf("Num, ana, diff = %18.10f %18.10f %18.10e\n", Ehartree, Uanal,
 ↪  abs(Ehartree-Uanal))
end

#@code_native test_main()
test_main()
```

3 Solving Schrodinger equation