Nonequilibrium quantum transport in molecular contacts including GW level interaction approximation

Tiangang Zhou[1]

[1]School of Physics, Peking University, China

June 23, 2019

Abstract

Nonequilibrium quantum transport offer a new way to probe the property of the system. Both Physics community and Chemistry community study system like quantum dots between lead or molecular junction between electronodes. By using nonequilibrium Keldysh formalism, we can include the coupling between the lead and molecular, and get the transportation property like current I and differential resistent. Also we choose quasi-particle GW approximation to calculate self-energy(RPA like), which could faithfully produce band structures and spectroscopic properties in some material. This work mainly work on the model hamiltonian, but we can describe how this works on real material by the sanity of the wanneir function basis.

Contents

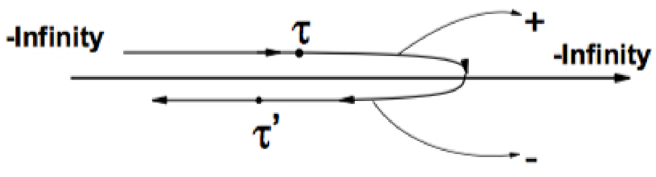

Figure 1: Contour in keldysh formulism

1 Introduction

Keldysh formulism is a great tool to deal with the nonequilibrium quantum transport problem[Onida *et al.*(2002)Onida, Reining, and Rubio]. By divide the contour to the upper part and the lower part(fig. 1), we can describe the dynamical property of the system using the nonequilibrium green function. In this scheme, Landauer-Buttiker equation[Jauho *et al.*(1994)Jauho, can describe the current of the system when the chemical potential μ between the left and right system are not equal.

The basic set up of the system could be understood in the following picture(fig. 2). This picture discribe the HOMO and LUMO energy levels of molecular as it approaches a metal surface. For weak coupling (physisorbed molecule) the gap is reduced due to image charge formation in the metal.[Thygesen and Rubio(2009)] Also we can calculate the property of the transportation like I and $\frac{\mathrm{d}I}{\mathrm{d}V}$

Using the method described in [Thygesen and Rubio(2008)] and the model using the (Sec 2), we can simulate the situation the molecular have contact with metal. I use *Julia* to do the majority of the programing, and **a large amount of derivation** can be seen in the later second.

Also we can using wannier function to construct the real space site if the material is crystal or molecular. The wannier function coperated with pseudopotential can effectively reduce the size of the sites, which make the numerical calculation feasible. For example, the beneze molecular using pseudopotential with PlaneWave basis only have 18 wannier basis, but 6-31g gaussian basis gives size of 40+, make the GW hard to carry out.

2 Model

Left Lead

$$\hat{H}_{\mathrm{L}} = \sum_{i=-\infty}^{0} \sum_{\sigma=\uparrow,\downarrow} \left(t(c_{i\sigma}^{\dagger} c_{i-1\sigma} + c_{i-1\sigma}^{\dagger} c_{i\sigma}) + \mu_L c_{i\sigma}^{\dagger} c_{i\sigma} \right)$$

Left lead and right lead can have different chemical potential (measured from self energy), which is in the unequilibrium case. Then we can use Landauer-Buttiker equation to calcaulate the I and differential resistent $\frac{\mathrm{d}I}{\mathrm{d}V}$.

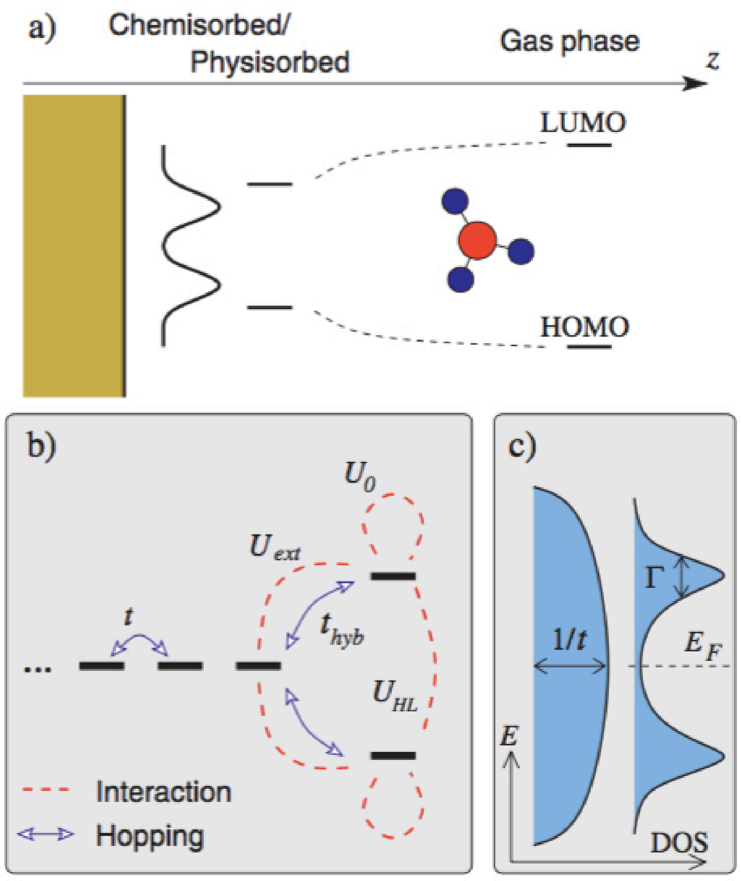

Figure 2: Molecular Junction contacted with lead

$$I_\alpha = \int \frac{d\omega}{2\pi} \operatorname{Tr}\left[\Sigma_\alpha^<(\omega) G_C^>(\omega) - \Sigma_\alpha^>(\omega) G_C^<(\omega)\right]$$

In the center region, the hamiltonian gives

$$\hat{H}_{\text{mol}} = \xi_H \hat{n}_H + (\xi_H + \Delta_0)\, \hat{n}_L + \hat{U}_{\text{mol}}$$

Where

$$\hat{n}_H = c_{H\uparrow}^{\uparrow} c_{H\uparrow} + c_{H\downarrow}^{\dagger} c_{H\downarrow}$$

and

$$\hat{U}_{\text{mol}} = U_0 \hat{n}_{H\uparrow} \hat{n}_{H\downarrow} + U_0 \hat{n}_{L\uparrow} \hat{n}_{L\downarrow} + U_{HL} \hat{n}_H \hat{n}_L$$

The coupling term

$$\hat{V} = \sum_{\nu=H,L} \sum_{\sigma=t,\downarrow} t_{\text{hyb}} \left(c_{0\sigma}^{\dagger} c_{\nu\sigma} + c_{\nu\sigma}^{\dagger} c_{0\sigma} \right)$$

For simplicity we only have hoping between lead and molecular as coupling term

3 To get the initial guess using HF

3.1 Equation of motion of green function method

We can apply equation of the green function to get the 1st-order cutting approximation(hartree fock approximation

In the heisenberg picture, we can get the derivation of the green function

$$i\frac{\partial}{\partial t} G_{ij}(t,t') = i\frac{\partial}{\partial t}\left\{ -i\theta(t-t') \left\langle \left\{ c_i(t), c_j^{\dagger}(t') \right\} \right\rangle \right\}$$

If we define

$$\left\langle \left\langle c_i(t) | c_j^{\dagger}(t') \right\rangle \right\rangle^r = G_{i,j}^r(t,t')$$

Then

$$i\frac{\partial}{\partial t} \left\langle \left\langle c_i(t) | c_j^{\dagger}(t') \right\rangle \right\rangle^r = \delta(t-t') \left\langle \left\{ c_i(t), c_j^{\dagger}(t') \right\} \right\rangle + \left\langle \left\langle \left[c_i(t), H(t) \right] | c_j^{\dagger}(t') \right\rangle \right\rangle^r$$

Fourier transformation gives

$$\left(\omega + i0^+\right) \left\langle \left\langle c_i | c_j^{\dagger} \right\rangle \right\rangle_\omega^r = \langle \{c_i, c_j\} \rangle + \left\langle \left\langle [c_i, H] \, | c_j^{\dagger} \right\rangle \right\rangle_\omega^r$$

Here state order in hilbert space

$$(|L\uparrow\rangle, |L\downarrow\rangle, |H\uparrow\rangle, ||H\downarrow\rangle)$$

We have

$$\left(\omega+i0^+\right)\left\langle\left\langle c_{H\uparrow}|c_j^\dagger\right\rangle\right\rangle_\omega^r=\delta_{H\uparrow,j}+\left\langle\left\langle(\xi_H c_{H\uparrow}+U_0 c_{H\uparrow}n_{H\downarrow}+U_{HL}c_{H\uparrow}n_L)|c_j^\dagger\right\rangle\right\rangle_\omega^r$$
$$\left(\omega+i0^+\right)\left\langle\left\langle c_{H\downarrow}|c_j^\dagger\right\rangle\right\rangle_\omega^r=\delta_{H\downarrow,j}+\left\langle\left\langle(\xi_H c_{H\downarrow}+U_0 c_{H\downarrow}n_{H\uparrow}+U_{HL}c_{H\downarrow}n_L)|c_j^\dagger\right\rangle\right\rangle_\omega^r$$
$$\left(\omega+i0^+\right)\left\langle\left\langle c_{L\uparrow}|c_j^\dagger\right\rangle\right\rangle_\omega^r=\delta_{L\uparrow,j}+\left\langle\left\langle((\xi_H+\Delta_0)c_{L\uparrow}+U_0 c_{L\uparrow}n_{L\downarrow}+U_{HL}c_{L\uparrow}n_H)|c_j^\dagger\right\rangle\right\rangle_\omega^r$$
$$\left(\omega+i0^+\right)\left\langle\left\langle c_{L\downarrow}|c_j^\dagger\right\rangle\right\rangle_\omega^r=\delta_{L\downarrow,j}+\left\langle\left\langle((\xi_H+\Delta_0)c_{L\downarrow}+U_0 c_{L\downarrow}n_{L\uparrow}+U_{HL}c_{L\downarrow}n_H)|c_j^\dagger\right\rangle\right\rangle_\omega^r$$

Using 1st order cutting approximation(hartree-fock mean field approximation), which means

$$\left\langle\left\langle(U_0 c_{L\downarrow}n_{L\uparrow}+U_{HL}c_{L\downarrow}n_H)|c_j^\dagger\right\rangle\right\rangle_\omega^r=U_0 n_{L\uparrow}\left\langle\left\langle(c_{L\downarrow}|c_j^\dagger\right\rangle\right\rangle_\omega^r+U_{HL}n_H\left\langle\left\langle c_{L\downarrow}|c_j^\dagger\right\rangle\right\rangle_\omega^r$$

Finally the green function matrix gives

$$G=\mathrm{diag}((\omega^+-(\xi_H+U_0 n_{H\downarrow}+U_{HL}n_L))^{-1},(\omega^+-(\xi_H+U_0 n_{H\uparrow}+U_{HL}n_L))^{-1},(\omega^+-(\xi_H+\Delta_0+U_0 n_{L\downarrow}+U_{HL}n_H))$$

Where $\omega^+=\omega+i0^+$

Now implement using this method

```julia
using LinearAlgebra
using Printf
using Plots
using LaTeXStrings
using FFTW

mutable struct Hamiltonian
    N::Int64
    H_0::Array{Float64,2}
    V::Array{Float64,2}
    Nb::Int64

    function Hamiltonian(N,H_0,V)
        @assert size(H_0,1) == size(H_0,2) == N
        @assert size(V,1) == size(V,2) == N

        new(N, H_0, V)
    end
end # struct Ham

mutable struct SCFOptions
    tol::Float64
    max_iter::Int
    alpha::Float64
    verbose::Int

    function SCFOptions()
        tol = 1e-5
        max_iter = 100
        alpha = 1.0
        verbose = 1
        new(tol,max_iter,verbose)
    end
end # struct SCFOptions
```

```julia
mutable struct GreenFunc
    N::Int64
    N_g::Int64
    delta_N_g::Float64
    G::Array{ComplexF64,3}
    η::Float64

    function GreenFunc(N, N_g, delta_N_g, G, η)
        @assert size(G,3) == N_g # fortran like array, the last to be larger
        @assert (N_g - 1) % 2 == 0 # default 101
        @assert size(G,1) == size(G,2) == N

        new(N, N_g, delta_N_g, G, η)
    end
end # struct GreenFunc


mutable struct Parameter
    N::Int64
    Δ_0::Float64
    μ_L::Float64
    μ_R::Float64
    U_0::Float64
    U_HL::Float64
    t_hyb::Float64
    t::Float64
    ξ_H::Float64

    function Parameter()
        N = 4 # 2 orbital(LUMO HOMO), 2 spin index
        # parameter
        Δ_0 = 2.0
        μ_L = 0.0
        μ_R = 0.0
        U_0 = 4.0
        U_HL = 3.0
        t_hyb = 0.4
        t = 10.0
        # ξ_H need to be adjust?  (must < 0)
        ξ_H = -6.0
        new(N, Δ_0, μ_L, μ_R, U_0, U_HL, t_hyb, t, ξ_H)
    end
end # struct GreenFunc


mutable struct Occupation_num
    n_H_up::Float64
    n_H_down::Float64
    n_L_up::Float64
    n_L_down::Float64

    function Occupation_num()
        n_H_up = 1
        n_H_down = 1
        n_L_up = 0
        n_L_down = 0
        new(n_H_up, n_H_down, n_L_up, n_L_down)
    end
    function Occupation_num(n_H_up, n_H_down, n_L_up, n_L_down)
        new(n_H_up, n_H_down, n_L_up, n_L_down)
    end
end # struct GreenFunc

function get_zero_freq(GF::GreenFunc)
    N_g_zero::Int64 = 1 + (GF.N_g - 1) // 2
# @printf("zero_freq_number = %4d\n", N_g_zero)
    return N_g_zero
end

function get_freq(GF::GreenFunc)
    freq = Array{Float64,1}(undef, GF.N_g)
    zero_f = get_zero_freq(GF)
    for i in 1:GF.N_g
        freq[i] = (i - zero_f) * GF.delta_N_g
    end
    return freq
end

# multidispatch of the function
function get_zero_freq(N_g::Int64)
    N_g_zero::Int64 = 1 + (N_g - 1) // 2
# @printf("zero_freq_number = %4d\n", N_g_zero)
    return N_g_zero
end

function get_freq(N_g::Int64, delta_N_g::Float64)
    freq = Array{Float64,1}(undef, N_g)
    zero_f = get_zero_freq(N_g)
    for i in 1:N_g
        freq[i] = (i - zero_f) * delta_N_g
    end
    return freq
end
```

```
get_freq (generic function with 2 methods)
```

```
occ = Occupation_num()
opts = SCFOptions()
param = Parameter()
N = param.N # 2 orbital(LUMO HOMO), 2 spin index
H_0 = zeros(N, N)
V = zeros(N, N)

# initial H_0
H_0[1, 1] = param.ξ_H
H_0[2, 2] = param.ξ_H
H_0[3, 3] = param.ξ_H + param.Δ_0
H_0[4, 4] = param.ξ_H + param.Δ_0

# construct to symmetry vertex matrix
V[1, 2] = param.U_0
V[3, 4] = param.U_0
V[1, 3] = param.U_HL
V[1, 4] = param.U_HL
V[2, 3] = param.U_HL
V[2, 4] = param.U_HL
V = (V + V') * 0.5
```

```
ham = Hamiltonian(N, H_0, V)

Main.WeaveSandBox3.Hamiltonian(4, [-6.0 0.0 0.0 0.0; 0.0 -6.0 0.0 0.0; 0.0
0.0 -4.0 0.0; 0.0 0.0 0.0 -4.0], [0.0 2.0 1.5 1.5; 2.0 0.0 1.5 1.5; 1.5 1.5
 0.0 2.0; 1.5 1.5 2.0 0.0], 4559405360)
```

- H_0: the single particle hamiltonian
- V: interaction vertex. V can be represent as $V_{ii,jj}$, which can simplify the diagramatic expansion technique.

```
function hf_analytical(param::Parameter, occ::Occupation_num, N_g::Int64,
    delta_N_g::Float64, η::Float64)
    # initial green function
    G0 = zeros(ComplexF64, param.N, param.N, N_g)
    freq = get_freq(N_g, delta_N_g)
    for grid_index in 1:N_g
        G0[1, 1, grid_index] = 1.0 / (freq[grid_index] + η * 1im -
            (param.ξ_H + param.U_0 * occ.n_H_down + param.U_HL * (occ.n_L_up +
    occ.n_L_down)  ) )
        G0[2, 2, grid_index] = 1.0 / (freq[grid_index] + η * 1im -
            (param.ξ_H + param.U_0 * occ.n_H_up + param.U_HL * (occ.n_L_up +
    occ.n_L_down)) )
        G0[3, 3, grid_index] = 1.0 / (freq[grid_index] + η * 1im -
            (param.ξ_H + param.Δ_0 + param.U_0 * occ.n_L_down + param.U_HL *
    (occ.n_H_up + occ.n_H_down)) )
        G0[1, 1, grid_index] = 1.0 / (freq[grid_index] + η * 1im -
            (param.ξ_H + param.Δ_0 + param.U_0 * occ.n_L_up + param.U_HL * (occ.n_H_up
    + occ.n_H_down) ) )
    end
    GF = GreenFunc(param.N, N_g, delta_N_g, G0, η)
    return GF
end

hf_analytical (generic function with 1 method)

# Plot spectrum
# maximum(imag(GF.G))
# minimum(imag(GF.G))
# GF.G[:,:,2002]
# Plot G_{n_{H\uparrow, H\uparrow}

GF = hf_analytical(param, occ, 10001, 0.001, 0.001);

freq = get_freq(10001, 0.001)
G_H_up = GF.G[1,1,:]
G_H_down = GF.G[2,2,:]
G_L_up = GF.G[3,3,:]
G_L_down = GF.G[4,4,:]
plot(freq, - imag(G_H_up) ,
    title="spectral of hartree-fock", grid=false,
    xlabel=L"\omega", ylabel=L"A(\omega)",
    label=L"G_{H\uparrow, H\uparrow}",
    legend=:topleft)

plot!(freq, - imag(G_H_down) ,
    label=L"G_{H\downarrow, H\downarrow}",
    legend=:topleft)
```

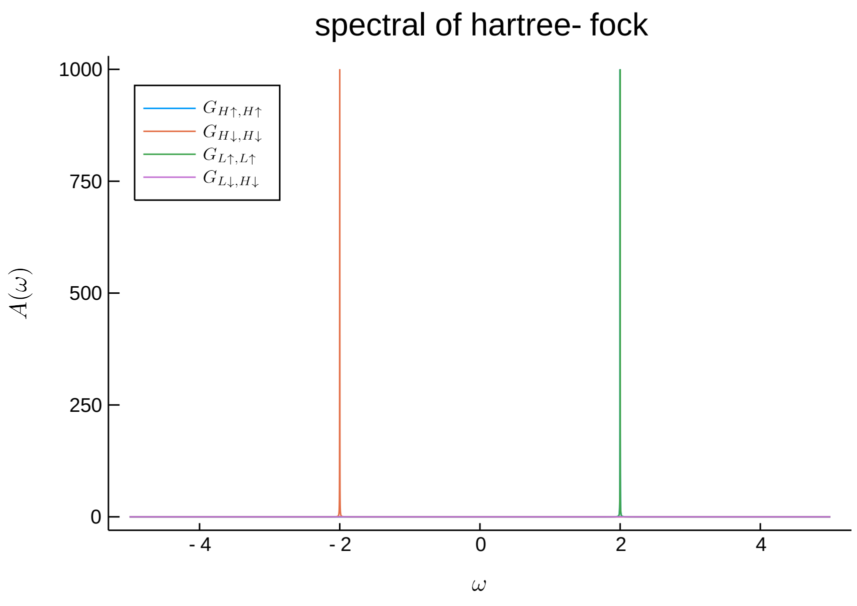

Figure 3: Hartree-Fock excitation spectrum

```
plot!(freq, - imag(G_L_up) ,
    label=L"G_{L\uparrow, L\uparrow}",
    legend=:topleft)

plot!(freq, - imag(G_L_down) ,
    label=L"G_{L\downarrow, H\downarrow}",
    legend=:topleft)

savefig("Model6-1.pdf")
```

From the figure we can find hartree-fock have inifity lifetime

3.2 Diagramatic expansion method

The first order self energy expansion:

$$\begin{aligned}\Sigma^{(1)}_{ij} &= \sum_k G_{kk} V_{ij,kk} \delta_{ij} - G_{ij} V_{ii,jj} && (1)\\ &= \sum_k G_{kk} V_{ii,kk} \delta_{ij} - G_{ij} V_{ii,jj} && (2)\\ & && (3)\end{aligned}$$

Which is called as hartree fock diagram

In this way, we notice that the interaction vertex V_{ijkl} can be compresed to $V_{ii,jj}$. Namely, the storage goes from $O(N^4)$ to $O(N^2)$. Also, $V_{ii,jj}$ will be sparse due to the form of the hamiltonian H_{mol}

To include the time/freq argument, the self energy of hartree fock can be written as

$$\Sigma^{(1)}_{ij} = \sum_k G_{kk}(t-t) V_{ij,kk} \delta_{ij} - G_{ij}(t-t) V_{ii,jj}$$

That's why we call the hartreee fock term are static. So we only calcualte $\omega = 0$ (Since $\Sigma^{(1)}_{ij}(\omega) = -i \sum_k G_{kk}(t=0) V_{ij,kk} \delta_{ij} \delta(\omega) + i G_{ij}(t=0) V_{ii,jj} \delta(\omega)$).

In the finite temperature, Matsubara frequency summation gives

$$G_{kk}(t=0) = \sum_m G_{kk}(i\omega_m) e^{i\omega_m 0^+} \propto n_F(\epsilon_k)$$

Then Self energy gives

$$\Sigma \propto n_F(\epsilon_k) V$$

It seems like hartree term

Notice that the G and H are in molecular single particle basis, not in fock space, since we want to formulate a effective system(Just like HF or Kohn-Sham system). In fact, the lattice version of HF and DFT can achieve this too.[Schönhammer *et al.*(1995)Schönhammer, Gunnarsson, and Noa

```julia
function hartree_fock(H::Hamiltonian, GF::GreenFunc, opt::SCFOptions)
    N = H.N
    @assert GF.N == N
    G0 = copy(GF.G)
    G0new = copy(GF.G)

    Sigma1 = zeros(N,N)

    zero_freq = get_zero_freq(GF)
    freq = get_freq(GF)
    η = GF.η

    for iter = 1 : opt.max_iter
        rho = diag(G0[:,:,zero_freq])
        Sigma1 = diagm(0 => H.V * rho) - (H.V.*(G0[:,:,zero_freq]))
        Sigma = Sigma1
        for grid_index in 1:GF.N_g
# println("here")
            G0new[:,:,grid_index] = inv((freq[grid_index] + η * 1im) * I - H.H_0-Sigma)
        end
        nrmerr = norm(G0[:,:,zero_freq]-G0new[:,:,zero_freq])/norm(G0[:,:,zero_freq])
        if( opt.verbose > 1 )
            @printf("iter = %4d, nrmerr = %15.5e\n", iter, nrmerr)
        end
        if( nrmerr < opt.tol )
            @printf("Convergence reached for HF. nrmerr = %g\n", nrmerr)
          break
        end
        # mixing scheme
        G0 = (1-opt.alpha) * G0 + opt.alpha * G0new
    end

    # Symmetrization
    for grid_index in 1:GF.N_g
        G0[:,:,grid_index] = (G0[:,:,grid_index]+G0[:,:,grid_index]') * 0.5
    end

    GF.G = G0
    return GF
end # function hartree_fock
```

```
hartree_fock (generic function with 1 method)
```

3.3 A simple test of HF

State order in hamiltonian

$$(|L\uparrow\rangle, |L\downarrow\rangle, |H\uparrow\rangle, ||H\downarrow\rangle)$$

Vertex in the form $V_{ij} \equiv V_{ii,jj}$

```
# initial guess of green function
G0 = ones(ComplexF64, 4, 4, 10001)
GF = GreenFunc(4, 10001, 0.0001, G0, 0.001)
# G0 = zeros(4, 4)

G = hartree_fock(ham, GF, opts);
```

```
Convergence reached for HF. nrmerr = 6.60646e-06
```

4 Get coupling self energy

The center region will couple to the transition region, and the green function of the transition region can be obtained with the coupling of lead. Represent Hamiltonian in the real site basis(like wannier basis), the hamiltonian of the lead and center part can be wrtten as:

$$h_L = \begin{pmatrix} \ddots & \vdots & \vdots & \vdots \\ \dots & \vdots & \vdots & \vdots \\ \dots & h_0 & v_0 & 0 \\ \dots & v_0^\dagger & h_0 & v_T \\ \dots & 0 & v_T^\dagger & h_T \end{pmatrix}$$

In the bulk of the lead, the hamiltonian can be writen as

$$h_L^{bulk} = \begin{pmatrix} \ddots & \vdots & \vdots & \vdots \\ \dots & \vdots & \vdots & \vdots \\ \dots & h_0 & v_0 & 0 \\ \dots & v_0^\dagger & h_0 & v_0 \\ \dots & 0 & v_0^\dagger & h_0 \end{pmatrix}$$

There, we can get the green function of the transition region(the right corner) by using the relation

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1}_{2,2} = (D - CA^{-1}B)^{-1}$$

We can easily apply it to our model. The transition region is just $\hat{c}_0$ site, and it is the end of the lead.

$$\left[g^r_{0,L}\right]_T = \begin{pmatrix} \ddots & & 0 \\ & \omega - h_0 & v_T \\ 0 & v_T^\dagger & (\omega - h_T) \end{pmatrix}^{-1}_{n,n} = ((\omega + i\eta)I - h_T - v_T^\dagger [g^{per}_{0,L}] v_T)^{-1}$$

$[g^{per}_{0,L}]$ can get from standard decimation technique.[Guinea *et al.*(1983)Guinea, Tejedor, Flores, and Louis]

4.1 standard decimation technique

using $(\omega^+\mathbf{I} - \mathbf{H})\mathbf{g}^r = \mathbf{I}$, we can get

$$\begin{pmatrix} & & \cdots & \\ t & \omega + i^{0+} - \mu_L & t & 0 \\ 0 & t & \omega + i^{0+} - \mu_L & t \\ 0 & 0 & t & \omega + i^{0+} - \mu_L \end{pmatrix} \begin{pmatrix} & & g_{31} & g_{30} \\ \vdots & \vdots & g_{21} & g_{20} \\ & & g_{11} & g_{10} \\ & & g_{01} & g_{00} \end{pmatrix} = \mathbf{I}$$

Consider the rightmost line

$$\begin{aligned} tg_{10} + (\omega + i0^+ - \mu_L)g_{00} &= 1 \\ tg_{20} + (\omega + i0^+ - \mu_L)g_{10} + tg_{00} &= 1 \\ tg_{30} + (\omega + i0^+ - \mu_L)g_{00} + tg_{10} &= 1 \end{aligned}$$

Generize to the iterative equaition

$$\begin{aligned} tg_{2n,0} + (\omega + i0^+ - \mu_L)g_{2n-1,0} + tg_{2n-2,0} &= 1 \\ tg_{2n+1,0} + (\omega + i0^+ - \mu_L)g_{2n,0} + tg_{2n-1,0} &= 1 \end{aligned}$$

Then we can get the matrix form

$$\begin{pmatrix} g_{2n+1,0} \\ g_{2n,0} \end{pmatrix} = \begin{pmatrix} -t^{-1}(\omega + i0^+) & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} -t^{-1}(\omega + i0^+) & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} g_{2n-1,0} \\ g_{2n-2,0} \end{pmatrix}$$

We have to get the eigenstates corresponding to the eigenvalue $|\lambda| < 1$, other wise the green function will diverge.

$$\begin{pmatrix} g_{2n+1,0} \\ g_{2n,0} \end{pmatrix} = T^{2n} \begin{pmatrix} g_{2n-1,0} \\ g_{2n-2,0} \end{pmatrix}$$

Get the eigenstate of T, then

$$\begin{pmatrix} g_{1,0} \\ g_{0,0} \end{pmatrix} = \begin{pmatrix} v_1 g \\ v_2 g \end{pmatrix}$$

finally

$$[g^{per}_{0,L}] = g_{0,0} = (\omega + i0^+ - \mu_L - tv_2v_1^{-1})^{-1}$$

4.2 get coupling Σ_L^r

Now consider the coupling self energy between transition region and center region

$$\Sigma_L^r = h_{CT} \left[g_{0,L}^r \right]_T h_{TC}$$

Apply to our model, the center region and the transition couple with $\hat{V}$, and only the **same spin direction** have the coupling

$$\hat{V} = \sum_{\nu=H,L} \sum_{\sigma=t,\downarrow} t_{\text{hyb}} \left(c_{0\sigma}^\dagger c_{\nu\sigma} + c_{\nu\sigma}^\dagger c_{0\sigma} \right)$$

Therefore, the coupling self-energy matrix will be

$$\Sigma_L^r = \begin{pmatrix} h_{hyb} & 0 & h_{hyb} & 0 \\ 0 & h_{hyb} & 0 & h_{hyb} \end{pmatrix}^T \begin{pmatrix} \left[g_{0,L}^r \right]_{T,\uparrow\uparrow} & 0 \\ 0 & \left[g_{0,L}^r \right]_{T,\downarrow\downarrow} \end{pmatrix} \begin{pmatrix} h_{hyb} & 0 & h_{hyb} & 0 \\ 0 & h_{hyb} & 0 & h_{hyb} \end{pmatrix}$$

less self energy

$$\Sigma_L^< = \begin{pmatrix} h_{hyb} & 0 & h_{hyb} & 0 \\ 0 & h_{hyb} & 0 & h_{hyb} \end{pmatrix}^T \begin{pmatrix} \left[g_{0,L}^< \right]_{T,\uparrow\uparrow} & 0 \\ 0 & \left[g_{0,L}^< \right]_{T,\downarrow\downarrow} \end{pmatrix} \begin{pmatrix} h_{hyb} & 0 & h_{hyb} & 0 \\ 0 & h_{hyb} & 0 & h_{hyb} \end{pmatrix}$$

Dimension equal to the centrel system size

A remark about this model is that, the transition region doesn't have different t_T with periodic part hoping t. Therefore, we can directly calculate the g_{00}

```
function sigma_r_lead(param::Parameter, N_g::Int64, delta_N_g::Float64, η::Float64)
    # diagnol transfer matrix
    # dimension of g_0_0 = 2
    # the third index in T is Left lead(T_L) or right lead(T_R)
    h_0 = [param.μ_L, param.μ_R]

    T = Array{ComplexF64,4}(undef,2, 2, 2, N_g)
    freq = get_freq(N_g, delta_N_g)
    for grid_index in 1:N_g
        for lead_index in 1:2
            T[1,1,lead_index,grid_index] = - 1.0 / param.t * (freq[grid_index] + η * 1im
    - h_0[lead_index])
            T[1,2,lead_index,grid_index] = - 1.0
            T[2,1,lead_index,grid_index] = 1.0
            T[2,2,lead_index,grid_index] = 0.0
        end
    end

    # g_00 index upspin/downspin/lead/grid_index
    g_00 = Array{ComplexF64,4}(undef,2, 2, 2, N_g)

    for grid_index in 1:N_g
        for lead_index in 1:2
            eigen_info = eigen(T[:,:,lead_index, grid_index])
            v = Array{ComplexF64,1}(undef,2)
            for eigenval_index in 1:2
```

```
                if norm(eigen_info.values[eigenval_index]) < 1.0
                    v = eigen_info.vectors[:,eigenval_index]
                end
            end
            g_00[:,:,lead_index, grid_index] = 1.0 / (freq[grid_index] + η * 1im -
    h_0[lead_index] - param.t * v[2] / v[1]) *
                        Matrix{ComplexF64}(I, 2, 2)
        end
    end

    Σ_r = Array{ComplexF64,4}(undef,param.N, param.N, 2, N_g)
    h_CL = [param.t_hyb 0 param.t_hyb 0; 0 param.t_hyb 0 param.t_hyb]

    for grid_index in 1:N_g
        for lead_index in 1:2
            Σ_r[:,:,lead_index, grid_index]  = h_CL' * g_00[:,:,lead_index, grid_index]
    * h_CL
        end
    end
    return Σ_r
end


function occ_func(freq::Float64)
    occ = 0.0
    if freq < 0
        occ = 1.0
    elseif freq == 0.0
        occ = 0.5
    end
    return occ
end

function less_great_factor(freq::Float64, less_great::Bool)
    if less_great == true
        occ_factor =  occ_func(freq)
    else
        occ_factor = (occ_func(freq) - 1)
    end
    return occ_factor
end

function sigma_less_great_lead(param::Parameter, N_g::Int64, delta_N_g::Float64,
    η::Float64, less_great::Bool)
    # True is less GF ; False is great GF

    # diagnol transfer matrix
    # dimension of g_0_0 = 2
    # the third index in T is Left lead(T_L) or right lead(T_R)
    h_0 = [param.μ_L, param.μ_R]

    T = Array{ComplexF64,4}(undef,2, 2, 2, N_g)
    freq = get_freq(N_g, delta_N_g)
    for grid_index in 1:N_g
        for lead_index in 1:2
            T[1,1,lead_index,grid_index] = - 1.0 / param.t * (freq[grid_index] + η * 1im
    - h_0[lead_index])
            T[1,2,lead_index,grid_index] = - 1.0
            T[2,1,lead_index,grid_index] = 1.0
            T[2,2,lead_index,grid_index] = 0.0
        end
    end

    # g_00 index upspin/downspin/lead/grid_index
    g_00 = Array{ComplexF64,4}(undef,2, 2, 2, N_g)

    for grid_index in 1:N_g
        for lead_index in 1:2
            eigen_info = eigen(T[:,:,lead_index, grid_index])
            v = Array{ComplexF64,1}(undef,2)
            for eigenval_index in 1:2
                if norm(eigen_info.values[eigenval_index]) < 1.0
                    v = eigen_info.vectors[:,eigenval_index]
                end
            end
# occ_factor = 0.0
# if less_great == true
# occ_factor = - occ_func(freq[grid_index])
# else
# occ_factor = - (occ_func(freq[grid_index]) - 1)
# end
            g_00[:,:,lead_index, grid_index] =  - less_great_factor(freq[grid_index],
    less_great) * (1.0 / (freq[grid_index] + η * 1im - h_0[lead_index] - param.t * v[2]
    / v[1]) -
                    1.0 / (freq[grid_index] - η * 1im - h_0[lead_index] - param.t * v[2]
    / v[1]) )*
                        Matrix{ComplexF64}(I, 2, 2)
        end
    end

    Σ_less_great = Array{ComplexF64,4}(undef,param.N, param.N, 2, N_g)
    h_CL = [param.t_hyb 0 param.t_hyb 0; 0 param.t_hyb 0 param.t_hyb]

    for grid_index in 1:N_g
        for lead_index in 1:2
            Σ_less_great[:,:,lead_index, grid_index]  = h_CL' * g_00[:,:,lead_index,
    grid_index] * h_CL
        end
    end
    return Σ_less_great
end

sigma_less_great_lead (generic function with 1 method)

Σ_r = sigma_r_lead(param, 10001, 0.001, 0.001);
Σ_less = sigma_less_great_lead(param, 10001, 0.001, 0.001, true);
```

5 Calculate retard green function G^r and less green function $G^<$

5.1 Keldysh Formulism

In the **keldysh formulism**, the time contour goes from $-\infty \to \infty$ in up contour + and from $\infty \to -\infty$ in down contour -.

$$
\begin{aligned}
G^r_{i,j}(t,t') &= -i\theta(t-t')\left\langle\left\{a_i(t), a_j^\dagger(t')\right\}\right\rangle \\
G^a_{i,j}(t,t') &= i\theta(t'-t)\left\langle\left\{a_i(t), a_j^\dagger(t')\right\}\right\rangle \\
G^<_{i,j}(t,t') &= i\left\langle a_j^\dagger(t')\, a_i(t)\right\rangle \\
G^>_{i,j}(t,t') &= -i\left\langle a_i(t) a_j^\dagger(t')\right\rangle
\end{aligned}
$$

Only the retarded green function G^r and less green $G^<$ function are independent variable.

These green function can be obtained from the following equation:

Dyson equatiton

$$G^r_C(\omega) = g^r_{0,C}(\omega) + g^r_{0,C}(\omega)\Sigma^r_{tot}(\omega)G^r_C(\omega)$$

Insert the formula of total self energy, it can be represented by

$$G^r_C(\omega) = [(\omega+i\eta)I_C - h_C - \Sigma^r_L(\omega) - \Sigma^r_R(\omega) - \Sigma^r(\omega)]^{-1}$$

Here, using the green function of the hartree fock as initial input, the Σ^r can be direct calculated by GW formulism. We will refer to the detailed calculation in the next section.

Then the keldysh equation can be written as:

$$G^{</>}_C = G^r_C \Sigma^{</>}_{tot} G^a_C(\omega) + \Delta^{</>}$$

Where

$$\Delta^{</>} = [I_C + G^r_C\Sigma^r_{tot}]\, g^{</>}_{0,C}\, [I_C + \Sigma^a_{tot}G^a_C]$$

$$< / >$$

means less green function or greater function

Since after fourier transformation

$$
\begin{aligned}
G^r_{jj}(\epsilon) &= \frac{1}{\epsilon - \epsilon_j + i0^+} \\
G^<_j(\epsilon) &= i2\pi f(\epsilon_j)\delta(\epsilon-\epsilon_j)
\end{aligned}
$$

We have the relation

$$
\begin{aligned}
\Sigma^{r/a}_{tot} &= \left(g^{r/a}_{0,C}\right)^{-1} - \left(G^{r/a}_C\right)^{-1} \\
g^<_{0,C} &= -f(\omega)\left[g^r_{0,C} - g^a_{0,C}\right]
\end{aligned}
$$

Using these relation we can get the calculation feasible result

$$
\begin{aligned}
\Delta^<(\omega) &= 2i\eta f(\omega)\, G^r_C(\omega)G^a_C(\omega) \\
\Delta^>(\omega) &= 2i\eta\,[f(\omega)-1]\, G^r_C(\omega)G^a_C(\omega)
\end{aligned}
$$

If we assume that the system in zero temeperature $T = 0K$, $f(\omega) = \theta(-\omega)$ becomes a step occupation function.

5.2 GW self energy calculation

$$P_{ij}\left(\tau,\tau'\right) = -iG_{ij}\left(\tau,\tau'\right)G_{ji}\left(\tau',\tau\right)$$

If the system is time homogeneous (almost all system has this property),

$$P_{ij}\left(t\right) = -iG_{ij}\left(t\right)G_{ji}\left(-t\right)$$

Using the Langreth conversion rules

Contour	Real axis
$C = \int_C AB$	$C^< = \int_t \left[A^r B^< + A^< B^a\right]$
	$C^r = \int_t A^r B^r$
$C\left(\tau,\tau'\right) = A\left(\tau,\tau'\right)B\left(\tau,\tau'\right)$	$C^<\left(t,t'\right) = A^<\left(t,t'\right)B^<\left(t,t'\right)$
	$C^r\left(t,t'\right) = A^<\left(t,t'\right)B^r\left(t,t'\right) + A^r\left(t,t'\right)B^<\left(t,t'\right) + A^r\left(t,t'\right)B^r\left(t,t'\right)$
$D\left(\tau,\tau'\right) = A\left(\tau,\tau'\right)B\left(\tau',\tau\right)$	$D^<\left(t,t'\right) = A^<\left(t,t'\right)B^>\left(t',t\right)$
	$D^r\left(t,t'\right) = A^<\left(t,t'\right)B^a\left(t',t\right) + A^r\left(t,t'\right)B^<\left(t',t\right)$

We can get

$$P^r_{ij}(t) = -iG^r_{ij}(t)G^<_{ji}(-t) - iG^<_{ij}(t)G^a_{ji}(-t)$$
$$P^{</>}_{ij}(t) = -iG^{</>}_{ij}(t)G^{>/<}_{ji}(-t)$$

Using the same techqinue, the GW self energy can be writen as

$$\Sigma^r_{GW,ij}(t) = iG^r_{ij}(t)W^>_{ij}(t) + iG^<_{ij}(t)W^r_{ij}(t)$$
$$\Sigma^{</>}_{GW,ij}(t) = iG^{</>}_{ij}(t)W^{</>}_{ij}(t)$$

Screen columb matrix represent by bare-bubble

$$W^r(\omega) = \tilde{V}\left[I - P^r(\omega)\tilde{V}\right]^{-1}$$
$$W^{</>}(\omega) = W^r(\omega)P^{</>}(\omega)W^a(\omega)$$

For computational reason, we calculate less and greater green function first. Then we use the relation

$$X^r(t) = \theta(-t)[X^>(t) - X^<(t)]$$

To get X^r in the time domain. Then we can use FFT(I use the FFTW.jl package) to switch from time domain to frequency domain.

```julia
function GreenFunc_less_great(param::Parameter, GF_r::GreenFunc, less_great::Bool,
    N_g::Int64, delta_N_g::Float64, η::Float64)
    # True is less GF ; False is great GF
    freq = get_freq(N_g, delta_N_g)
    GF_less_great_data = copy(GF_r.G)

    sigma_less_great_lead = sigma_less_great_lead(param, N_g, delta_N_g, η, less_great)
    # 4 4 2 N_g
    sigma_less_great_lead_sum = Array{ComplexF64,3}(undef, param.N, param.N, N_g)
    sigma_less_great_lead_sum = sigma_less_great_lead[:,:,1,:]
    sigma_less_great_lead_sum += sigma_less_great_lead[:,:,2,:]

    for grid_index in 1:N_g
        GF_less_great_data[:,:,grid_index] = GF_r.G[:,:,grid_index] *
    sigma_less_great_lead_sum *
            GF_r.G[:,:,grid_index]' + 2.0im * η * less_great_factor(freq[grid_index]) *
    GF_r.G[:,:,grid_index] *
            GF_r.G[:,:,grid_index]'
    end

    return GF_less_great_data
end

# freq = get_freq(N_g, delta_N_g)


function GW_scf(param::Parameter, occ::Occupation_num, N_g::Int64, delta_N_g::Float64,
    η::Float64)
    # hartree fock retard
    GF_r = hf_analytical(param, occ, 10001, 0.001, 0.001)
    G_r_freq_init = GF_r.G

    # switch from freq domain to time domain
end

GW_scf (generic function with 1 method)
```

References

[Onida *et al.*(2002)Onida, Reining, and Rubio] G. Onida, L. Reining, and A. Rubio, Reviews of Modern Physics **74**, 601 (2002).

[Jauho *et al.*(1994)Jauho, Wingreen, and Meir] A.-P. Jauho, N. S. Wingreen, and Y. Meir, Physical Review B **50**, 5528 (1994).

[Thygesen and Rubio(2009)] K. S. Thygesen and A. Rubio, PHYSICAL REVIEW LETTERS **102**, 046802 (2009).

[Thygesen and Rubio(2008)] K. S. Thygesen and A. Rubio, Physical Review B **77**, 115333 (2008).

[Schönhammer *et al.*(1995)Schönhammer, Gunnarsson, and Noack] K. Schönhammer, O. Gunnarsson, and R. M. Noack, Physical Review B **52**, 2504 (1995).

[Guinea *et al.*(1983)Guinea, Tejedor, Flores, and Louis] F. Guinea, C. Tejedor, F. Flores, and E. Louis, Physical Review B **28**, 4397 (1983).