```
(*Model*)
```

```
funGraph[Nv_,GraphType_]:=Module[{EL,vv},(
    If[StringMatchQ[GraphType,"Chain"],(
        EL={};
        Do[(
            AppendTo[EL,UndirectedEdge[v,Mod[v+1,Nv]]]
        ),{v,0,Nv-2}];
    )];

    If[StringMatchQ[GraphType,"Ladder"],(
        EL={};
        Do[(
            AppendTo[EL,UndirectedEdge[v,Mod[v+1,Nv]]];
        ),{v,0,Nv-1,2}];
        Do[(
            AppendTo[EL,UndirectedEdge[v,Mod[v+2,Nv]]];
            AppendTo[EL,UndirectedEdge[Mod[v+1,Nv],Mod[v+3,Nv]]]
        ),{v,0,Nv-3,2}];
    )];

    If[!StringMatchQ[GraphType,"Chain"]&&!StringMatchQ[GraphType,"Ladder"],(
        EL={};
        Do[(
            Do[(
                vv=RandomChoice[Drop[Table[vv,{vv,0,Nv-1}],{v+1}]];
                AppendTo[EL,UndirectedEdge[v,vv]]
            ),{i,1,ToExpression[GraphType]}];
        ),{v,0,Nv-1}]
        (*RG=RandomGraph[{Nv,ToExpression[GraphType]*Nv}];
        EL=EdgeList[RG]*)
    )];

    Return[EL]
)]
```

```
funHeisenberg[Nq_,EL_]:=Module[{Istr,Model,q,qq,ps},(
    Istr="";
    Do[(
        Istr=StringInsert[Istr,"I",q+1];
    ),{q,0,Nq-1}];
    Model={};

    Do[(
        q=Mod[EL⟦l,1⟧,Nq];
        qq=Mod[EL⟦l,2⟧,Nq];

        ps=StringReplacePart[Istr,"X",{q+1,q+1}];
        ps=StringReplacePart[ps,"X",{qq+1,qq+1}];
        AppendTo[Model,{1.,ps}];

        ps=StringReplacePart[Istr,"Y",{q+1,q+1}];
        ps=StringReplacePart[ps,"Y",{qq+1,qq+1}];
        AppendTo[Model,{1.,ps}];

        ps=StringReplacePart[Istr,"Z",{q+1,q+1}];
        ps=StringReplacePart[ps,"Z",{qq+1,qq+1}];
        AppendTo[Model,{1.,ps}];
    ),{l,1,Length[EL]}];

    Return[Model]
)]
```

```
funFermiHubbard[Nq_,EL_,u_]:=Module[{Istr,Model,v,vv,q,qq,ps},(
    Istr="";
    Do[(
        Istr=StringInsert[Istr,"I",q+1];
    ),{q,0,Nq-1}];
    Model={};

    Do[(
        v=Mod[EL⟦l,1⟧,Nq/2];
        vv=Mod[EL⟦l,2⟧,Nq/2];

        q=2*Min[{v,vv}];
        qq=2*Max[{v,vv}];
        ps=StringReplacePart[Istr,"X",{q+1,q+1}];
        Do[(
            ps=StringReplacePart[ps,"Z",{qqq+1,qqq+1}];
        ),{qqq,q+1,qq-1}];
        ps=StringReplacePart[ps,"X",{qq+1,qq+1}];
        AppendTo[Model,{-1./2.,ps}];
        ps=StringReplacePart[Istr,"Y",{q+1,q+1}];
        Do[(
            ps=StringReplacePart[ps,"Z",{qqq+1,qqq+1}];
        ),{qqq,q+1,qq-1}];
```

```
        ps=StringReplacePart[ps,"Y",{qq+1,qq+1}];
        AppendTo[Model,{-1./2.,ps}];

        q=2*Min[{v,vv}]+1;
        qq=2*Max[{v,vv}]+1;
        ps=StringReplacePart[Istr,"X",{q+1,q+1}];
        Do[(
            ps=StringReplacePart[ps,"Z",{qqq+1,qqq+1}];
        ),{qqq,q+1,qq-1}];
        ps=StringReplacePart[ps,"X",{qq+1,qq+1}];
        AppendTo[Model,{-1./2.,ps}];
        ps=StringReplacePart[Istr,"Y",{q+1,q+1}];
        Do[(
            ps=StringReplacePart[ps,"Z",{qqq+1,qqq+1}];
        ),{qqq,q+1,qq-1}];
        ps=StringReplacePart[ps,"Y",{qq+1,qq+1}];
        AppendTo[Model,{-1./2.,ps}];

        (*ps=StringReplacePart[Istr,"Z",{q+1,q+1}];
        ps=StringReplacePart[ps,"Z",{qq+1,qq+1}];
        AppendTo[Model,{1.,ps}];*)
    ),{l,1,Length[EL]}];

    Do[(
        q=2*v;
        qq=2*v+1;
        ps=StringReplacePart[Istr,"Z",{q+1,q+1}];
        ps=StringReplacePart[ps,"Z",{qq+1,qq+1}];
        AppendTo[Model,{u/4.,ps}];
    ),{v,0,Nq/2-1}];

    Return[Model]
)]
```

```
(*funFermiHubbard[Nq_,EL_,u_]:=Module[{Istr,Model,v,vv,q,qq,ps},(
    Istr="";
    Do[(
        Istr=StringInsert[Istr,"I",q+1];
    ),{q,0,Nq-1}];
    Model={};

    Do[(
        v=Mod[EL⟦l,1⟧,Nq/2];
        vv=Mod[EL⟦l,2⟧,Nq/2];

        q=2*Min[{v,vv}];
        qq=2*Max[{v,vv}];
        ps=StringReplacePart[Istr,"X",{q+1,q+1}];
        Do[(
            ps=StringReplacePart[Istr,"Z",{q+1,q+1}];
        ),{qqq,q+1,qq-1}];
```

```
        ps=StringReplacePart[ps,"X",{qq+1,qq+1}];
        AppendTo[Model,{-1./2.,ps}];
        ps=StringReplacePart[Istr,"Y",{q+1,q+1}];
        Do[(
            ps=StringReplacePart[Istr,"Z",{q+1,q+1}];
        ),{qqq,q+1,qq-1}];
        ps=StringReplacePart[ps,"Y",{qq+1,qq+1}];
        AppendTo[Model,{-1./2.,ps}];

        q=2*Min[{v,vv}]+1;
        qq=2*Max[{v,vv}]+1;
        ps=StringReplacePart[Istr,"X",{q+1,q+1}];
        Do[(
            ps=StringReplacePart[Istr,"Z",{q+1,q+1}];
        ),{qqq,q+1,qq-1}];
        ps=StringReplacePart[ps,"X",{qq+1,qq+1}];
        AppendTo[Model,{-1./2.,ps}];
        ps=StringReplacePart[Istr,"Y",{q+1,q+1}];
        Do[(
            ps=StringReplacePart[Istr,"Z",{q+1,q+1}];
        ),{qqq,q+1,qq-1}];
        ps=StringReplacePart[ps,"Y",{qq+1,qq+1}];
        AppendTo[Model,{-1./2.,ps}];

        ps=StringReplacePart[Istr,"Z",{q+1,q+1}];
        ps=StringReplacePart[ps,"Z",{qq+1,qq+1}];
        AppendTo[Model,{1.,ps}];
    ),{l,1,Length[EL]}];

    Do[(
        q=2*v;
        qq=2*v+1;
        ps=StringReplacePart[Istr,"Z",{q+1,q+1}];
        ps=StringReplacePart[ps,"Z",{qq+1,qq+1}];
        AppendTo[Model,{u/4.,ps}];
    ),{v,0,Nq/2-1}];

    Return[Model]
)]*)
```

```
(*Reference state*)
```

```
funPairwiseSinglet[Nq_]:=Module[{U,qq,ψ},(
    U=IdentityMatrix[2^Nq];
    Do[(
        qq=Mod[q+1,Nq];
        U=U . ((funPX[q]-funPX[qq])/N[Sqrt[2]]);
    ),{q,0,Nq-1,2}];

    ψ=Table[{0.},2^Nq];
    ψ[[1,1]]=1.;
    ψ=U . ψ;

    Return[ψ]
)]
```

```
funHartreeFock[Nq_,EL_]:=Module[{Model,Ham,EE,ES,ψ},(
    Model=funFermiHubbard[Nq,EL,0.];
    Ham=funHamiltonianQubit[Model];
    {EE,ES}=funSpectrum[Ham];
    ψ=Transpose[{ES[[1]]}];
    Return[ψ]
)]
```

```
(*Diagonalisation*)
```

```
funDiagonalisation[Hmat_,Smat_]:=Module[{Svals,Svecs,cn,V,Heff,Hvals,Hvecs,EK,SK},(
    {Svals,Svecs}=funSpectrum[Smat];
    cn=Max[Abs[Svals]]/Min[Abs[Svals]];
    V=Transpose[Svecs] . DiagonalMatrix[1./Sqrt[Svals]];
    Heff=ConjugateTranspose[V] . Hmat . V;
    {Hvals,Hvecs}=funSpectrum[Heff];
    EK=Hvals[[1]];
    Return[{EK,cn}]
)]
```

```
(*Functions*)
```

```
funLORfactor[htot_,τ_,NT_,u_]:=Module[{t,factor},(
    t=τ*u/NT;
    factor=(Sqrt[1.+htot^2*t^2]+Exp[htot*t]-(1+htot*t))^NT/Exp[Exp[1.]*htot^2*t^2/2.
    Return[factor]
)]
```

```
funIntegral[htot_,τ_,d_]:=Module[{NT,χ,costList,factor,cost},(
    NT=Ceiling[4.*Exp[1.]*htot^2*τ^2];
    χ=Exp[1.]*htot^2*τ^2/(2.*NT);
    costList={};
    Do[(
        If[n==0,factor=1.,factor=(n/Exp[1.])^(-(n/2))];
        cost=NIntegrate[2(2^(-(n/2)) Abs[HermiteH[n,u/Sqrt[2]]] 1/Sqrt[2π] Exp[-(1/
        AppendTo[costList,cost]
    ),{n,0,d-1}];
    Return[costList]
)]
```

```
funCost[htot_,τ_,d_]:=Module[{costList},(
    costList=funIntegral[htot,τ,d];
    Do[(
        costList[[k]]=costList[[k]]*(k-1)^((k-1)/2)/Exp[(k-1)/2]/τ^(k-1);
    ),{k,2,d}];
    Return[costList]
)]
```

```
(*funCost[τ_,k_]:=Module[{χ,A,Cost},(
    χ=0.125;
    A=1.8946081370976193;
    If[k==1,(
        Cost=Sqrt[1./(1.-2.*χ)];
    ),(
        Cost=A*(k-1)^((k-1)/2)/Exp[(k-1)/2]/τ^(k-1);
    )];
    Return[Cost]
)]*)
```

```
(*funCost[τ_,k_]:=Module[{χ,Cost},(
    χ=0.125;
    If[k==1,(
        Cost=Sqrt[1./(1.-2.*χ)];
    ),(
        Cost=Sqrt[2.*(k-1)!/(1.-4.*χ)]/τ^(k-1);
    )];
    Return[Cost]
)]*)
```

```
(*Matrices*)
```

```
funMatPower[EE_,Proψ_,d_,E0_]:=Module[{Hmat,Smat},(
    Hmat=Table[Table[0.,d],d];
    Smat=Table[Table[0.,d],d];
    Do[(
        Do[(
            If[k+q-2==0,(
                Hmat[[k,q]]=Total[Proψ*EE];
                Smat[[k,q]]=Total[Proψ]
            ),(
                Hmat[[k,q]]=Total[Proψ*(EE-E0)^(k+q-2)*EE];
                Smat[[k,q]]=Total[Proψ*(EE-E0)^(k+q-2)]
            )]

        ),{q,1,d}]
    ),{k,1,d}];
    Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
    Smat=(Smat+ConjugateTranspose[Smat])/2.;
    Return[{Hmat,Smat}]
)]
```

```
funMatChebyshev[EE_,Proψ_,d_,htot_,E0_]:=Module[{Hmat,Smat},(
    Hmat=Table[Table[0.,d],d];
    Smat=Table[Table[0.,d],d];
    Do[(
        Do[(
            Hmat[[k,q]]=Total[Proψ*ChebyshevT[k-1,(EE-E0)/htot]*ChebyshevT[q-1,(EE-E0)
            Smat[[k,q]]=Total[Proψ*ChebyshevT[k-1,(EE-E0)/htot]*ChebyshevT[q-1,(EE-E0)
        ),{q,1,d}]
    ),{k,1,d}];
    Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
    Smat=(Smat+ConjugateTranspose[Smat])/2.;
    Return[{Hmat,Smat}]
)]
```

```
funMatGaussianPower[EE_,Proψ_,d_,htot_,τ_,E0_]:=Module[{costList,ProGψ,Hmat,Smat},(
    costList=funCost[htot,τ,d];
    ProGψ=Exp[-(EE-E0)^2*τ^2]*Proψ;
    Hmat=Table[Table[0.,d],d];
    Smat=Table[Table[0.,d],d];
    Do[(
        Do[(
            If[k+q-2==0,(
                Hmat〚k,q〛=Total[ProGψ*EE]/(costList〚k〛*costList〚q〛);
                Smat〚k,q〛=Total[ProGψ]/(costList〚k〛*costList〚q〛)
            ),(
                Hmat〚k,q〛=Total[ProGψ*(EE-E0)^(k+q-2)*EE]/(costList〚k〛*costList〚q〛)
                Smat〚k,q〛=Total[ProGψ*(EE-E0)^(k+q-2)]/(costList〚k〛*costList〚q〛)
            )]
        ),{q,1,d}]
    ),{k,1,d}];
    Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
    Smat=(Smat+ConjugateTranspose[Smat])/2.;
    Return[{Hmat,Smat}]
)]
```

```
(*funMatGaussianPower[EE_,Proψ_,d_,τ_,E0_]:=Module[{ProGψ,Hmat,Smat},(
    ProGψ=Exp[-(EE-E0)^2*τ^2]*Proψ;
    Hmat=Table[Table[0.,d],d];
    Smat=Table[Table[0.,d],d];
    Do[(
        Do[(
            If[k+q-2==0,(
                Hmat〚k,q〛=Total[ProGψ*EE]/(funCost[τ,k]*funCost[τ,q]);
                Smat〚k,q〛=Total[ProGψ]/(funCost[τ,k]*funCost[τ,q])
            ),(
                Hmat〚k,q〛=Total[ProGψ*(EE-E0)^(k+q-2)*EE]/(funCost[τ,k]*funCost[τ,q
                Smat〚k,q〛=Total[ProGψ*(EE-E0)^(k+q-2)]/(funCost[τ,k]*funCost[τ,q])
            )]
        ),{q,1,d}]
    ),{k,1,d}];
    Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
    Smat=(Smat+ConjugateTranspose[Smat])/2.;
    Return[{Hmat,Smat}]
)]*)
```

```
funMatInversePower[EE_,Proψ_,d_,E0_]:=Module[{Hmat,Smat},(
    Hmat=Table[Table[0.,d],d];
    Smat=Table[Table[0.,d],d];
    Do[(
        Do[(
            If[k+q-2==0,(
                Hmat[[k,q]]=Total[Proψ*EE];
                Smat[[k,q]]=Total[Proψ]
            ),(
                Hmat[[k,q]]=Total[Proψ*(EE-E0)^(-k-q+2)*EE];
                Smat[[k,q]]=Total[Proψ*(EE-E0)^(-k-q+2)]
            )]

        ),{q,1,d}]
    ),{k,1,d}];
    Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
    Smat=(Smat+ConjugateTranspose[Smat])/2.;
    Return[{Hmat,Smat}]
)]
```

```
funMatITE[EE_,Proψ_,d_,τ_,E0_]:=Module[{ITE,Hmat,Smat},(
    ITE=Exp[-(EE-E0)*τ];
    Hmat=Table[Table[0.,d],d];
    Smat=Table[Table[0.,d],d];
    Do[(
        Do[(
            If[k+q-2==0,(
                Hmat[[k,q]]=Total[Proψ*EE];
                Smat[[k,q]]=Total[Proψ]
            ),(
                Hmat[[k,q]]=Total[Proψ*ITE^(k+q-2)*EE];
                Smat[[k,q]]=Total[Proψ*ITE^(k+q-2)]
            )]
        ),{q,1,d}]
    ),{k,1,d}];
    Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
    Smat=(Smat+ConjugateTranspose[Smat])/2.;
    Return[{Hmat,Smat}]
)]
```

```
funMatRTE[EE_,Proψ_,d_,△t_,E0_]:=Module[{RTE,Hmat,Smat},(
    RTE=Exp[I*(EE-E0)*△t];
    Hmat=Table[Table[0.,d],d];
    Smat=Table[Table[0.,d],d];
    Do[(
        Do[(
            If[k+q-2==0,(
                Hmat[[k,q]]=Total[Proψ*EE];
                Smat[[k,q]]=Total[Proψ]
            ),(
                Hmat[[k,q]]=Total[Proψ*RTE^(k-q)*EE];
                Smat[[k,q]]=Total[Proψ*RTE^(k-q)]
            )]
        ),{q,1,d}]
    ),{k,1,d}];
    Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
    Smat=(Smat+ConjugateTranspose[Smat])/2.;
    Return[{Hmat,Smat}]
)]
```

```
funMatFilter[EE_,Proψ_,d_,T_,E0_,△E_]:=Module[{ProGψ,Hmat,Smat},(
    Hmat=Table[Table[0.,d],d];
    Smat=Table[Table[0.,d],d];
    Do[(
        Do[(
            ProFψ=Sinc[(EE-(E0+(k-1)*△E))*T]*Sinc[(EE-(E0+(q-1)*△E))*T]*Proψ;
            Hmat[[k,q]]=Total[ProFψ*EE];
            Smat[[k,q]]=Total[ProFψ]
        ),{q,1,d}]
    ),{k,1,d}];
    Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
    Smat=(Smat+ConjugateTranspose[Smat])/2.;
    Return[{Hmat,Smat}]
)]
```

```
(*Plot*)
```

```
funGammaEpsilon[Eg_,pg_,Hmat_,Smat_,Ide_,CH_,CS_,logηList_]:=Module[{γList,∈List,log
    γList=0.*logηList;
    ∈List=0.*logηList;
    Do[(
        logη=logηList[[j]];
        η=10.^logη;

        {EK,cn}=funDiagonalisation[Hmat+2.*CH*η*Ide,Smat+2.*CS*η*Ide];
        ∈=EK-Eg;

        γ=(pg^2∈^2)/(16η^2);
        γList[[j]]=γ;
        ∈List[[j]]=∈;
        (*Print[{"j",j,η,γ,∈,ToString[Now]}];*)
    ),{j,1,Length[logηList]}];
    Return[{γList,∈List}]
)]
```

```
funInterpolation[xList_,yList_,x_]:=Module[{i,y},(
    i=Position[(x-xList)^2,Min[(x-xList)^2]][[1,1]];
    If[(xList[[i]]<x&&i<Length[xList])||(xList[[i]]>x&&i==1),y=yList[[i]]+(x-xList[[i]]) (yL
    If[xList[[i]]==x,y=yList[[i]]];
    If[(xList[[i]]>x&&i>1)||(xList[[i]]<x&&i==Length[xList]),y=yList[[i]]+(x-xList[[i]]) (yL
    Return[y]
)]
```