```
(*Pauli operators and spins*)
```

```
σI={{1,0},{0,1}};
σX={{0,1},{1,0}};
σY={{0,-I},{I,0}};
σZ={{1,0},{0,-1}};
spinUp={{1},{0}};
spinDown={{0},{1}};
```

```
(*Model and Hamiltonian*)
```

```
HeisenbergHam:=Module[{Ham,hTempX,hTempY,hTempZ,numQubit},(*J=1*)
numQubit=10;
Ham=ConstantArray[0,{2^numQubit,2^numQubit}];
Do[
   Do[
      If[(i==1 && j==2)||(i==2 && j==3)||(i==3 && j==4)||(i==4 && j==5)||(i==5 && j==6)||
         (i==6 && j==7)||(i==7 && j==8)||(i==8 && j==9)||(i==9 && j==10),(*lattice, i<j*)
         hTempX={{1}};hTempY={{1}};hTempZ={{1}};
         Do[
            If[k==i || k==j,
               hTempX=KroneckerProduct[hTempX,σX];hTempY=KroneckerProduct[hTempY,σY];
              ,hTempX=KroneckerProduct[hTempX,σI];hTempY=KroneckerProduct[hTempY,σI];
               ];
         ,{k,1,numQubit}];
         Ham=Ham+hTempX+hTempY+hTempZ;
      ];
   ,{j,1,numQubit}];
,{i,1,numQubit}];
Ham=N[Ham]]
```

```
HubbardHam[U_]:=Module[{Ham,J,hTempX,hTempY,hTempZ,numSite,numQubit},
J=1;
numSite=5;
numQubit=2*numSite;
Ham=ConstantArray[0,{2^numQubit,2^numQubit}];
(*interaction*)
Do[
hTempZ={{1}};
   Do[
      If[k==i||k==i+numSite,hTempZ=KroneckerProduct[hTempZ,σZ],hTempZ=KroneckerProduct
   ,{k,1,numQubit}];
Ham=Ham+U/4 hTempZ;
,{i,1,numSite}];
(*hopping*)
Do[
   Do[
      If[(i==1 && j==2)||(i==1 && j==3)||(i==2 && j==3)||(i==2 && j==4)||(i==3 && j==4)||
         (*hopping_spinup*)
         hTempX={{1}};hTempY={{1}};
```

```
            If[i≠1,
               Do[
                  hTempX=KroneckerProduct[hTempX,σI];hTempY=KroneckerProduct[hTempY,σI];
               ,{k,1,i-1}];
            ];
            Do[
               If[k==i||k==j,
                   hTempX=KroneckerProduct[hTempX,σX];hTempY=KroneckerProduct[hTempY,σY]
                  ,hTempX=KroneckerProduct[hTempX,σZ];hTempY=KroneckerProduct[hTempY,σZ]
            ,{k,i,j}];
            Do[
               hTempX=KroneckerProduct[hTempX,σI];hTempY=KroneckerProduct[hTempY,σI];
            ,{k,j+1,numQubit}];
            Ham=Ham-J/2 hTempX-J/2 hTempY;
            (*hopping_spindown*)
            hTempX={{1}};hTempY={{1}};
            Do[
               hTempX=KroneckerProduct[hTempX,σI];hTempY=KroneckerProduct[hTempY,σI];
            ,{k,1,i+numSite-1}];
            Do[
               If[k==i+numSite||k==j+numSite,
                   hTempX=KroneckerProduct[hTempX,σX];hTempY=KroneckerProduct[hTempY,σY]
                  ,hTempX=KroneckerProduct[hTempX,σZ];hTempY=KroneckerProduct[hTempY,σZ]
            ,{k,i+numSite,j+numSite}];
            If[j≠numSite,
               Do[
                  hTempX=KroneckerProduct[hTempX,σI];hTempY=KroneckerProduct[hTempY,σI];
               ,{k,j+numSite+1,numQubit}];
            ];
            Ham=Ham-J/2 hTempX-J/2 hTempY;
         ];
   ,{j,1,numSite}];
,{i,1,numSite}];
Ham=N[Ham]]
```

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

```
φHeisenberg=(1./Sqrt[2])^5 (KroneckerProduct[
KroneckerProduct[spinUp,spinDown]-KroneckerProduct[spinDown,spinUp],
KroneckerProduct[spinUp,spinDown]-KroneckerProduct[spinDown,spinUp],
KroneckerProduct[spinUp,spinDown]-KroneckerProduct[spinDown,spinUp],
KroneckerProduct[spinUp,spinDown]-KroneckerProduct[spinDown,spinUp],
KroneckerProduct[spinUp,spinDown]-KroneckerProduct[spinDown,spinUp]]);
```

```
φHubbard:=Module[{Ham,vals,vecs,ψ},
Ham=HubbardHam[0];
{vals,vecs}=funSpectrum[Ham];
ψ=Transpose[{vecs[[1]]}];
ψ]
```

```
(*Spectrum*)
```

```
funSpectrum[Ham_]:=Module[{vals,vecs},
{vals,vecs}=Eigensystem[Ham];
vals=Re[vals];
{vals,vecs}=Transpose@SortBy[Transpose[{vals,vecs}],First];
(*Total[Total[Abs[Transpose[vecs].DiagonalMatrix[vals].Conjugate[vecs]-Ham]]]*)
{vals,vecs}]
```

```
(*Subspace diagonalization*)
```

```
funSubDiag[Hmat_,Smat_]:=Module[{Svals,Svecs,V,Heff,vals,vecs,EK,cn},
{Svals,Svecs}=funSpectrum[Smat];
cn=Max[Abs[Svals]]/Min[Abs[Svals]];
V=Transpose[Svecs] . DiagonalMatrix[1./Sqrt[Svals]];
Heff=ConjugateTranspose[V] . Hmat . V;
{vals,vecs}=funSpectrum[Heff];
EK=vals⟦1⟧;
{EK,cn}]
```

```
(*(a^\dag S a)/(a^\dag a)*)
```

```
funaSa[Hmat_,Smat_]:=Module[{Svals,Svecs,V,Heff,vals,vecs,EK,cn,a,aSa},
{Svals,Svecs}=funSpectrum[Smat];
cn=Max[Abs[Svals]]/Min[Abs[Svals]];
V=Transpose[Svecs] . DiagonalMatrix[1./Sqrt[Svals]];
Heff=ConjugateTranspose[V] . Hmat . V;
{vals,vecs}=funSpectrum[Heff];
EK=vals⟦1⟧;
a=V . Transpose[vecs⟦1⟧];
aSa=Re[ConjugateTranspose[a] . Smat . a]/Re[ConjugateTranspose[a] . a];
{EK,cn,aSa}]
```

```
(*Cost for Gaussian power*)
```

```
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]
)]
```

```
(*funCostGP[τ_,n_]:=Module[{χ,A,cost},
χ=0.125;(*χ=1/8 for n=0 and χ=1/16 for n>0*)
A=1.537931098297999;
If[n==0,
   cost=Sqrt[1./(1.-2.*χ)];,
   cost=A*n^(n/2)/Exp[n/2]/τ^n;
   ];
cost]*)
```

```
(*Hmat and Smat for different cases in Table I*)
```

```
(*1. Power*)
```

```
funMatP[Λ_,E0_,d_,probφ_]:=Module[{Hmat,Smat},
Hmat=ConstantArray[0,{d,d}];
Smat=ConstantArray[0,{d,d}];
Do[
   Do[
      If[k+q-2==0,
         Hmat[[k,q]]=Total[probφ*Λ];
         Smat[[k,q]]=Total[probφ];
        ,Hmat[[k,q]]=Total[probφ*(Λ-E0)^(k+q-2)*Λ];
         Smat[[k,q]]=Total[probφ*(Λ-E0)^(k+q-2)];];
   ,{q,1,d}];
,{k,1,d}];
Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
Smat=(Smat+ConjugateTranspose[Smat])/2.;
{Hmat,Smat}]
```

```
(*2. Chebyshev polynomial*)
```

```
funMatCP[Λ_,E0_,d_,probφ_,htot_]:=Module[{Hmat,Smat},
Hmat=ConstantArray[0,{d,d}];
Smat=ConstantArray[0,{d,d}];
Do[
   Do[
      Hmat⟦k,q⟧=Total[probφ*ChebyshevT[k-1,(Λ-E0)/htot]*ChebyshevT[q-1,(Λ-E0)/htot]*
      Smat⟦k,q⟧=Total[probφ*ChebyshevT[k-1,(Λ-E0)/htot]*ChebyshevT[q-1,(Λ-E0)/htot]]
   ,{q,1,d}];
,{k,1,d}];
Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
Smat=(Smat+ConjugateTranspose[Smat])/2.;
{Hmat,Smat}]
```

```
(*3. Gaussian power*)
```

```
funMatGP[Λ_,E0_,τ_,d_,probφ_,htot_]:=Module[{Hmat,Smat,probGφ,costList},
    costList=funCost[htot,τ,d];
    probGφ=Exp[-(Λ-E0)^2*τ^2]*probφ;
    Hmat=ConstantArray[0,{d,d}];
    Smat=ConstantArray[0,{d,d}];
    Do[
        Do[
            If[k+q-2==0,
               Hmat⟦k,q⟧=Total[probGφ*Λ]/(costList⟦k⟧*costList⟦q⟧);
               Smat⟦k,q⟧=Total[probGφ]/(costList⟦k⟧*costList⟦q⟧);
              ,Hmat⟦k,q⟧=Total[probGφ*(Λ-E0)^(k+q-2)*Λ]/(costList⟦k⟧*costList⟦q⟧);
               Smat⟦k,q⟧=Total[probGφ*(Λ-E0)^(k+q-2)]/(costList⟦k⟧*costList⟦q⟧);];
        ,{q,1,d}]
    ,{k,1,d}];
    Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
    Smat=(Smat+ConjugateTranspose[Smat])/2.;
{Hmat,Smat}]
```

```
(*funMatGP[Λ_,E0_,τ_,d_,probφ_]:=Module[{Hmat,Smat,probGφ},
probGφ=Exp[-(Λ-E0)^2*τ^2]*probφ;
Hmat=ConstantArray[0,{d,d}];
Smat=ConstantArray[0,{d,d}];
Do[
   Do[
      If[k+q-2==0,
         Hmat[[k,q]]=Total[Re[probGφ*Λ]]/(funCostGP[τ,k-1]*funCostGP[τ,q-1]);
         Smat[[k,q]]=Total[Re[probGφ]]/(funCostGP[τ,k-1]*funCostGP[τ,q-1]);
        ,Hmat[[k,q]]=Total[Re[probGφ*(Λ-E0)^(k+q-2)*Λ]]/(funCostGP[τ,k-1]*funCostGP[τ
         Smat[[k,q]]=Total[Re[probGφ*(Λ-E0)^(k+q-2)]]/(funCostGP[τ,k-1]*funCostGP[τ,q
      ];
   ,{q,1,d}];
,{k,1,d}];
Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
Smat=(Smat+ConjugateTranspose[Smat])/2.;
{Hmat,Smat}
]*)
```

```
(*4. Inverse power*)
```

```
funMatIP[Λ_,E0_,d_,probφ_]:=Module[{Hmat,Smat},
Hmat=ConstantArray[0,{d,d}];
Smat=ConstantArray[0,{d,d}];
Do[
   Do[
      If[k+q-2==0,
         Hmat[[k,q]]=Total[probφ*Λ];
         Smat[[k,q]]=Total[probφ];
        ,Hmat[[k,q]]=Total[probφ*(Λ-E0)^(-k-q+2)*Λ];
         Smat[[k,q]]=Total[probφ*(Λ-E0)^(-k-q+2)];];
   ,{q,1,d}];
,{k,1,d}];
Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
Smat=(Smat+ConjugateTranspose[Smat])/2.;
{Hmat,Smat}
]
```

```
(*5. Imaginary time evolution*)
```

```
funMatITE[Λ_,E0_,τ_,d_,probφ_]:=Module[{Hmat,Smat,ITE},
ITE=Exp[-(Λ-E0)*τ];
Hmat=ConstantArray[0,{d,d}];
Smat=ConstantArray[0,{d,d}];
Do[
   Do[
      Hmat⟦k,q⟧=Total[probφ*ITE^(k+q-2)*Λ];
      Smat⟦k,q⟧=Total[probφ*ITE^(k+q-2)];
   ,{q,1,d}];
,{k,1,d}];
Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
Smat=(Smat+ConjugateTranspose[Smat])/2.;
{Hmat,Smat}]
```

```
(*6. Real time evolution*)
```

```
funMatRTE[Λ_,E0_,Δt_,d_,probφ_]:=Module[{Hmat,Smat,RTE},
RTE=Exp[I*(Λ-E0)*Δt];
Hmat=ConstantArray[0,{d,d}];
Smat=ConstantArray[0,{d,d}];
Do[
   Do[
      Hmat⟦k,q⟧=Total[probφ*RTE^(k-q)*Λ];(*Hmat and Smat are complex Hermitian-Toepl
      Smat⟦k,q⟧=Total[probφ*RTE^(k-q)];
   ,{q,1,d}];
,{k,1,d}];
Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
Smat=(Smat+ConjugateTranspose[Smat])/2.;
{Hmat,Smat}]
```

```
(*7. Filter*)
```

```
funMatF[Λ_,E0_,ΔE_,τ_,d_,probφ_]:=Module[{Hmat,Smat},
Hmat=ConstantArray[0,{d,d}];
Smat=ConstantArray[0,{d,d}];
Do[
   Do[
      Hmat⟦k,q⟧=Total[probφ*Sinc[(Λ-(E0+(k-1)*ΔE))*τ]*Sinc[(Λ-(E0+(q-1)*ΔE))*τ]*Λ]
      Smat⟦k,q⟧=Total[probφ*Sinc[(Λ-(E0+(k-1)*ΔE))*τ]*Sinc[(Λ-(E0+(q-1)*ΔE))*τ]];
   ,{q,1,d}];
,{k,1,d}];
Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
Smat=(Smat+ConjugateTranspose[Smat])/2.;
{Hmat,Smat}]
```

```
(*funTransformW[L_,d_,△t_,E0_,△E_]:=Module[{W},
W=ConstantArray[0,{L,d}];
Do[
   Do[
      W〚i,j〛=Exp[-I*(i-(L+1)/2)*△t*(E0+△E*(j-1))]/L;
   ,{j,1,d}];
,{i,1,L}];
W(*Dimension is L*d*)
]*)
```

```
(*funMatTAF[Λ_,△t_,L_,probφ_,d_,E0_,△E_]:=Module[{Hmat,Smat,W},
{Hmat,Smat}=funMatRTE[Λ,△t,L,probφ];
W=funTransformW[L,d,△t,E0,△E];
Hmat=ConjugateTranspose[W].Hmat.W;
Smat=ConjugateTranspose[W].Smat.W;
Hmat=(Hmat+ConjugateTranspose[Hmat])/2.;
Smat=(Smat+ConjugateTranspose[Smat])/2.;
{Hmat,Smat} (*Dimension is d*d, instead of L*L*)
]*)
```

```
(*Plot*)
```

```
funEpsilonGamma[Hmat_,Smat_,costH_,costS_,Id_,ηList_,Eg_,pg_]:=Module[{ϵList,γList,η
ϵList=ConstantArray[0,Length[ηList]];
γList=ConstantArray[0,Length[ηList]];
Do[
η=ηList〚j〛;
{EK,cn}=funSubDiag[Hmat+2.*costH*η*Id,Smat+2.*costS*η*Id];
ϵ=EK-Eg;
ϵList〚j〛=ϵ;
γ=(pg^2*ϵ^2)/(16*η^2);
γList〚j〛=γ;
(*Print[{j,η,γ,ϵ}];*)
,{j,1,Length[ηList]}];
{ϵList,γList}]
```

```
funEpsilonM[Hmat_,Smat_,costH_,costS_,Id_,ηList_,Eg_,d_,κ_]:=Module[{ϵList,MList,η,E
ϵList=ConstantArray[0,Length[ηList]];
MList=ConstantArray[0,Length[ηList]];
Do[
η=ηList〚j〛;
MList〚j〛=0.5*d*Log[4d/κ]/η^2;
{EK,cn}=funSubDiag[Hmat+2.*costH*η*Id,Smat+2.*costS*η*Id];
ϵ=EK-Eg;
ϵList〚j〛=ϵ;
,{j,1,Length[ηList]}];
{MList,ϵList}]
```

```
funEpsilonMRTE[Hmat_,Smat_,costH_,costS_,Id_,ηList_,Eg_,d_,κ_]:=Module[{∈List,MList,
∈List=ConstantArray[0,Length[ηList]];
MList=ConstantArray[0,Length[ηList]];
Do[
η=ηList⟦j⟧;
MList⟦j⟧=0.5*(2d-1)*Log[4d/κ]/η^2;
{EK,cn}=funSubDiag[Hmat+2.*costH*η*Id,Smat+2.*costS*η*Id];
∈=EK-Eg;
∈List⟦j⟧=∈;
,{j,1,Length[ηList]}];
{MList,∈List}]
```

```
(*Regularisation in practice*)
```

```
funRegPrac[MList_,rep_,Hmat_,Smat_,d_,costH_,costS_,Id_,Eg_,pg_,complex_]:=Module[{∈
∈List=ConstantArray[0,{rep,Length[MList]}];
γList=ConstantArray[0,{rep,Length[MList]}];
Do[
   Do[
      σ=1/Sqrt[MList⟦j⟧];
      hatH=Hmat+RandomVariate[NormalDistribution[0,costH*σ],{d,d}]+I*complex*RandomVa
      hatH=(hatH+ConjugateTranspose[hatH])/2;(*Hermitian Gaussian noise matrix with n
      hatS=Smat+RandomVariate[NormalDistribution[0,costS*σ],{d,d}]+I*complex*RandomVa
      hatS=(hatS+ConjugateTranspose[hatS])/2;
      η=Max[Norm[hatH-Hmat,2]/costH,Norm[hatS-Smat,2]/costS];(*optimal η*)
      {hatEK,cn}=funSubDiag[hatH+costH*η*Id,hatS+costS*η*Id];
      ∈=hatEK-Eg;
      γList⟦i,j⟧=(pg^2*∈^2)/(16*η^2);
      ∈List⟦i,j⟧=∈;
   ,{j,1,Length[MList]}];
,{i,1,rep}];
{∈List,γList}]
```

```
(*Eta vs RMSE*)
```

```
funEtaGPRMSE[MList_,rep_,ηList_,Hmat_,Smat_,d_,costH_,costS_,Id_,Eg_]:=Module[{∈List
RMSEList=ConstantArray[0,{Length[MList],Length[ηList]}];
Do[
   Do[
      η=ηList⟦j⟧;
      ∈List=ConstantArray[0,rep];
      Do[
      σ=1/Sqrt[MList⟦i⟧];
      (*construct Hankel Gaussian Noise matrix of H*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costH*σ],2*d-1];
      HankelNoiMat=ConstantArray[0,{d,d}];
      Do[
         Do[
            HankelNoiMat⟦i,j⟧=GaussNoiList⟦i+j-1⟧;
            ,{j,1,d}];
      ,{i,1,d}];
      hatH=Hmat+HankelNoiMat;
      (*construct Hankel Gaussian Noise matrix of S*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costS*σ],2*d-1];
      HankelNoiMat=ConstantArray[0,{d,d}];
      Do[
         Do[
            HankelNoiMat⟦i,j⟧=GaussNoiList⟦i+j-1⟧;
            ,{j,1,d}];
      ,{i,1,d}];
      hatS=Smat+HankelNoiMat;
      {hatEK,cn}=funSubDiag[hatH+costH*η*Id,hatS+costS*η*Id];
      ∈=hatEK-Eg;
      ∈List⟦k⟧=∈;
      ,{k,1,rep}];
      RMSEList⟦i,j⟧=RootMeanSquare[∈List];
   ,{j,1,Length[ηList]}];
,{i,1,Length[MList]}];
RMSEList]
```

```
funEtaRTERMSE[MList_,rep_,ηList_,Hmat_,Smat_,d_,costH_,costS_,Id_,Eg_]:=Module[{∈Lis
RMSEList=ConstantArray[0,{Length[MList],Length[ηList]}];
Do[
   Do[
      η=ηList[[j]];
      ∈List=ConstantArray[0,rep];
      Do[
      σ=1/Sqrt[MList[[i]]];
      (*construct complex Hermite-Toeplitz Gaussian Noise matrix of H*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costH*σ],2*d-1];
      ToeplitzNoiMat=ConstantArray[0,{d,d}];
       Do[
         Do[
            ToeplitzNoiMat[[i,j]]=GaussNoiList[[Abs[i-j]+1]]+Sign[i-j]*I*GaussNoiList[[Ab
            ,{j,1,d}];
      ,{i,1,d}];
      hatH=Hmat+ToeplitzNoiMat;
      (*construct complex Hermite-Toeplitz Gaussian Noise matrix of S*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costS*σ],2*d-1];
      ToeplitzNoiMat=ConstantArray[0,{d,d}];
       Do[
         Do[
            ToeplitzNoiMat[[i,j]]=GaussNoiList[[Abs[i-j]+1]]+Sign[i-j]*I*GaussNoiList[[Ab
            ,{j,1,d}];
      ,{i,1,d}];
      hatS=Smat+ToeplitzNoiMat;
      {hatEK,cn}=funSubDiag[hatH+costH*η*Id,hatS+costS*η*Id];
      ∈=hatEK-Eg;
      ∈List[[k]]=∈;
      ,{k,1,rep}];
      RMSEList[[i,j]]=RootMeanSquare[∈List];
   ,{j,1,Length[ηList]}];
,{i,1,Length[MList]}];
RMSEList]
```

```
funEtaFRMSE[MList_,rep_,ηList_,Hmat_,Smat_,d_,costH_,costS_,Id_,Eg_]:=Module[{∊List,
RMSEList=ConstantArray[0,{Length[MList],Length[ηList]}];
Do[
   Do[
      η=ηList⟦j⟧;
      ∊List=ConstantArray[0,rep];
      Do[
      σ=1/Sqrt[MList⟦i⟧];
      hatH=Hmat+RandomVariate[NormalDistribution[0,costH*σ],{d,d}];
      hatH=(hatH+ConjugateTranspose[hatH])/2;(*Real Hermitian Gaussian noise matrix*)
      hatS=Smat+RandomVariate[NormalDistribution[0,costS*σ],{d,d}];
      hatS=(hatS+ConjugateTranspose[hatS])/2;
      {hatEK,cn}=funSubDiag[hatH+costH*η*Id,hatS+costS*η*Id];
      ∊=hatEK-Eg;
      ∊List⟦k⟧=∊;
      ,{k,1,rep}];
      RMSEList⟦i,j⟧=RootMeanSquare[∊List];
   ,{j,1,Length[ηList]}];
,{i,1,Length[MList]}];
RMSEList]
```

```
(*eta vs Abs[epsilon] in 1-kappa probability*)
```

```
(**suitable for P, GP, IP and ITE*)
funEtaEpsilonGP[MList_,rep_,κ_,ηList_,Hmat_,Smat_,d_,costH_,costS_,Id_,Eg_]:=Module[
EpsilonList=ConstantArray[0,{Length[MList],Length[ηList]}];
ηpracList=ConstantArray[0,Length[MList]];
Do[
M=MList[[i]];
σ=1/Sqrt[M];
hatH=ConstantArray[0,{rep,d,d}];
hatS=ConstantArray[0,{rep,d,d}];
   Do[
      (*construct Hankel Gaussian Noise matrix of H*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costH*σ],2*d-1];
      HankelNoiMat=ConstantArray[0,{d,d}];
      Do[
         Do[
            HankelNoiMat[[i,j]]=GaussNoiList[[i+j-1]];
            ,{j,1,d}];
      ,{i,1,d}];
      hatH[[k]]=Hmat+HankelNoiMat;
      (*construct Hankel Gaussian Noise matrix of S*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costS*σ],2*d-1];
      HankelNoiMat=ConstantArray[0,{d,d}];
      Do[
         Do[
            HankelNoiMat[[i,j]]=GaussNoiList[[i+j-1]];
            ,{j,1,d}];
      ,{i,1,d}];
      hatS[[k]]=Smat+HankelNoiMat;
   ,{k,1,rep}];
   Do[
      η=ηList[[j]];
      ϵList=ConstantArray[0,rep];
      Do[
      {hatEK,cn}=funSubDiag[hatH[[k]]+costH*η*Id,hatS[[k]]+costS*η*Id];
      ϵ=Abs[hatEK-Eg];
      ϵList[[k]]=ϵ;
      ,{k,1,rep}];
      EpsilonList[[i,j]]=Sort[ϵList,Less][[Round[(1-κ)*rep]]];
   ,{j,1,Length[ηList]}];
ηpracList[[i]]=Sqrt[0.5*d/M Log[4*d/κ]];(*Matrix Gaussian series for real Hermite/Hank
,{i,1,Length[MList]}];
{EpsilonList,ηpracList}]
```

```
funEtaEpsilonRTE[MList_,rep_,κ_,ηList_,Hmat_,Smat_,d_,costH_,costS_,Id_,Eg_]:=Module
EpsilonList=ConstantArray[0,{Length[MList],Length[ηList]}];
ηpracList=ConstantArray[0,Length[MList]];
Do[
M=MList⟦i⟧;
σ=1/Sqrt[M];
hatH=ConstantArray[0,{rep,d,d}];
hatS=ConstantArray[0,{rep,d,d}];
   Do[
      (*construct complex Hermite-Toeplitz Gaussian Noise matrix of H*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costH*σ],2*d-1];
      ToeplitzNoiMat=ConstantArray[0,{d,d}];
       Do[
         Do[
            ToeplitzNoiMat⟦i,j⟧=GaussNoiList⟦Abs[i-j]+1⟧+Sign[i-j]*I*GaussNoiList⟦Ab
            ,{j,1,d}];
      ,{i,1,d}];
      hatH⟦k⟧=Hmat+ToeplitzNoiMat;
      (*construct complex Hermite-Toeplitz Gaussian Noise matrix of S*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costS*σ],2*d-1];
      ToeplitzNoiMat=ConstantArray[0,{d,d}];
       Do[
         Do[
            ToeplitzNoiMat⟦i,j⟧=GaussNoiList⟦Abs[i-j]+1⟧+Sign[i-j]*I*GaussNoiList⟦Ab
            ,{j,1,d}];
      ,{i,1,d}];
      hatS⟦k⟧=Smat+ToeplitzNoiMat;
   ,{k,1,rep}];
   Do[
      η=ηList⟦j⟧;
      ϵList=ConstantArray[0,rep];
      Do[
      {hatEK,cn}=funSubDiag[hatH⟦k⟧+costH*η*Id,hatS⟦k⟧+costS*η*Id];
      ϵ=Abs[hatEK-Eg];
      ϵList⟦k⟧=ϵ;
      ,{k,1,rep}];
      EpsilonList⟦i,j⟧=Sort[ϵList,Less]⟦Round[(1-κ)*rep]⟧;
   ,{j,1,Length[ηList]}];
ηpracList⟦i⟧=Sqrt[0.5*(2d-1)/M Log[4*d/κ]];(*Matrix Gaussian series for complex Herm
,{i,1,Length[MList]}];
{EpsilonList,ηpracList}]
```

```
(*suitable for CP and F*)
funEtaEpsilonF[MList_,rep_,κ_,ηList_,Hmat_,Smat_,d_,costH_,costS_,Id_,Eg_]:=Module[{
EpsilonList=ConstantArray[0,{Length[MList],Length[ηList]}];
ηpracList=ConstantArray[0,Length[MList]];
Do[
M=MList⟦i⟧;
σ=1/Sqrt[M];
hatH=ConstantArray[0,{rep,d,d}];
hatS=ConstantArray[0,{rep,d,d}];
   Do[
      hatH⟦k⟧=Hmat+RandomVariate[NormalDistribution[0,costH*σ],{d,d}];
      hatH⟦k⟧=(hatH⟦k⟧+ConjugateTranspose[hatH⟦k⟧])/2;(*Real Hermitian Gaussian nois
      hatS⟦k⟧=Smat+RandomVariate[NormalDistribution[0,costS*σ],{d,d}];
      hatS⟦k⟧=(hatS⟦k⟧+ConjugateTranspose[hatS⟦k⟧])/2;
   ,{k,1,rep}];
   Do[
      η=ηList⟦j⟧;
      ϵList=ConstantArray[0,rep];
      Do[
      {hatEK,cn}=funSubDiag[hatH⟦k⟧+costH*η*Id,hatS⟦k⟧+costS*η*Id];
      ϵ=hatEK-Eg;
      ϵList⟦k⟧=ϵ;
      ,{k,1,rep}];
      EpsilonList⟦i,j⟧=Sort[ϵList,Less]⟦Round[(1-κ)*rep]⟧;
   ,{j,1,Length[ηList]}];
ηpracList⟦i⟧=Sqrt[0.5*d/M Log[4*d/κ]];(*Matrix Gaussian series for real Hermite/Hank
,{i,1,Length[MList]}];
{EpsilonList,ηpracList}]
```

```
(*Practical eta*)(*M vs epsilon in 1-kappa probability using practical eta*)
```

```
funExtract[ϵList_,MList_,rep_,κ_]:=Module[{ϵListκ},
ϵListκ=ConstantArray[0,Length[MList]];
Do[
ϵListκ⟦i⟧=Sort[ϵList⟦;;,i⟧,Less]⟦Round[(1-κ)*rep]⟧;
,{i,1,Length[MList]}];
ϵListκ]
```

```
(*GP,P,IP,ITE*)
funEtaPracGP[MList_,rep_,n_,Hmat_,Smat_,d_,κ_,costH_,costS_,Id_,Eg_,pg_]:=Module[{∈L
∈List=ConstantArray[0,{rep,Length[MList]}];
γList=ConstantArray[0,{rep,Length[MList]}];
Do[
   Do[
      M=MList⟦j⟧;
      σ=1/Sqrt[M];
      (*construct Hankel Gaussian Noise matrix of H*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costH*σ],2*d-1];
      HankelNoiMat=ConstantArray[0,{d,d}];
      Do[
         Do[
            HankelNoiMat⟦i,j⟧=GaussNoiList⟦i+j-1⟧;
            ,{j,1,d}];
      ,{i,1,d}];
      hatH=Hmat+HankelNoiMat;
      (*construct Hankel Gaussian Noise matrix of S*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costS*σ],2*d-1];
      HankelNoiMat=ConstantArray[0,{d,d}];
      Do[
         Do[
            HankelNoiMat⟦i,j⟧=GaussNoiList⟦i+j-1⟧;
            ,{j,1,d}];
      ,{i,1,d}];
      hatS=Smat+HankelNoiMat;
      ηprac=Sqrt[0.5*d/M Log[4*d/κ]];(*Matrix Gaussian series for real Hermite/Hanke
      η=n*ηprac;
      {hatEK,cn}=funSubDiag[hatH+costH*η*Id,hatS+costS*η*Id];
      ∈=hatEK-Eg;
      γList⟦i,j⟧=(pg^2*∈^2)/(16*η^2);
      ∈List⟦i,j⟧=∈;
   ,{j,1,Length[MList]}];
,{i,1,rep}];
{∈List,γList}]
```

```
funEtaPracRTE[MList_,rep_,n_,Hmat_,Smat_,d_,κ_,costH_,costS_,Id_,Eg_,pg_]:=Module[{ϵ
ϵList=ConstantArray[0,{rep,Length[MList]}];
γList=ConstantArray[0,{rep,Length[MList]}];
Do[
   Do[
      M=MList⟦j⟧;
      σ=1/Sqrt[M];
      (*construct complex Hermite-Toeplitz Gaussian Noise matrix of H*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costH*σ],2*d-1];
      ToeplitzNoiMat=ConstantArray[0,{d,d}];
       Do[
         Do[
            ToeplitzNoiMat⟦i,j⟧=GaussNoiList⟦Abs[i-j]+1⟧+Sign[i-j]*I*GaussNoiList⟦Ab
            ,{j,1,d}];
      ,{i,1,d}];
      hatH=Hmat+ToeplitzNoiMat;
      (*construct complex Hermite-Toeplitz Gaussian Noise matrix of S*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costS*σ],2*d-1];
      ToeplitzNoiMat=ConstantArray[0,{d,d}];
       Do[
         Do[
            ToeplitzNoiMat⟦i,j⟧=GaussNoiList⟦Abs[i-j]+1⟧+Sign[i-j]*I*GaussNoiList⟦Ab
            ,{j,1,d}];
      ,{i,1,d}];
      hatS=Smat+ToeplitzNoiMat;
      ηprac=Sqrt[0.5*(2d-1)/M Log[4*d/κ]];(*Matrix Gaussian series for complex Hermi
      η=n*ηprac;
      {hatEK,cn}=funSubDiag[hatH+costH*η*Id,hatS+costS*η*Id];
      ϵ=hatEK-Eg;
      γList⟦i,j⟧=(pg^2*ϵ^2)/(16*η^2);
      ϵList⟦i,j⟧=ϵ;
   ,{j,1,Length[MList]}];
,{i,1,rep}];
{ϵList,γList}]
```

```
(*F,CP,rescaled GP*)
funEtaPracF[MList_,rep_,n_,Hmat_,Smat_,d_,κ_,costH_,costS_,Id_,Eg_,pg_]:=Module[{∈Li
∈List=ConstantArray[0,{rep,Length[MList]}];
γList=ConstantArray[0,{rep,Length[MList]}];
Do[
   Do[
      M=MList[[j]];
      σ=1/Sqrt[M];
      hatH=Hmat+RandomVariate[NormalDistribution[0,costH*σ],{d,d}];
      hatH=(hatH+ConjugateTranspose[hatH])/2;(*Real Hermitian Gaussian noise matrix*)
      hatS=Smat+RandomVariate[NormalDistribution[0,costS*σ],{d,d}];
      hatS=(hatS+ConjugateTranspose[hatS])/2;
      ηprac=Sqrt[0.5*d/M Log[4*d/κ]];(*Matrix Gaussian series for real Hermite/Hankel
      η=n*ηprac;
      {hatEK,cn}=funSubDiag[hatH+costH*η*Id,hatS+costS*η*Id];
      ∈=hatEK-Eg;
      γList[[i,j]]=(pg^2*∈^2)/(16*η^2);
      ∈List[[i,j]]=∈;
   ,{j,1,Length[MList]}];
,{i,1,rep}];
{∈List,γList}]
```

```
(*Thresholding*)
```

```
thresholding[hatH_,hatS_,θ_]:=Module[{vals,vecs,index,V,Hth,Sth,hatEK,cn},
{vals,vecs}=funSpectrum[hatS];
index={};
Do[
If[vals[[k]]>θ,AppendTo[index,k]];
,{k,1,d}];
V=Transpose[Extract[vecs,Transpose[{index}]]];
Hth=ConjugateTranspose[V] . hatH . V;
Hth=(Hth+ConjugateTranspose[Hth])/2;
Sth=ConjugateTranspose[V] . hatS . V;
Sth=(Sth+ConjugateTranspose[Sth])/2;
{hatEK,cn}=funSubDiag[Hth,Sth];
hatEK]
```

```
funThrPracGP[MList_,rep_,Hmat_,Smat_,d_,costH_,costS_,th_,Eg_]:=Module[{∊List,σ,hatH
∊List=ConstantArray[0,{rep,Length[MList]}];
Do[
   Do[
      σ=1/Sqrt[MList[[j]]];
      (*construct Hankel Gaussian Noise matrix of H*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costH*σ],2*d-1];
      HankelNoiMat=ConstantArray[0,{d,d}];
      Do[
         Do[
            HankelNoiMat[[i,j]]=GaussNoiList[[i+j-1]];
            ,{j,1,d}];
      ,{i,1,d}];
      hatH=Hmat+HankelNoiMat;
      (*construct Hankel Gaussian Noise matrix of S*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costS*σ],2*d-1];
      HankelNoiMat=ConstantArray[0,{d,d}];
      Do[
         Do[
            HankelNoiMat[[i,j]]=GaussNoiList[[i+j-1]];
            ,{j,1,d}];
      ,{i,1,d}];
      hatS=Smat+HankelNoiMat;
      θ=th*σ;
      hatEK=thresholding[hatH,hatS,θ];
      ∊=Abs[hatEK-Eg];
      ∊List[[i,j]]=∊;
   ,{j,1,Length[MList]}];
,{i,1,rep}];
∊List]
```

```
funThrPracRTE[MList_,rep_,Hmat_,Smat_,d_,costH_,costS_,th_,Eg_]:=Module[{ϵList,σ,hat
ϵList=ConstantArray[0,{rep,Length[MList]}];
Do[
   Do[
      σ=1/Sqrt[MList⟦j⟧];
      (*construct complex Hermite-Toeplitz Gaussian Noise matrix of H*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costH*σ],2*d-1];
      ToeplitzNoiMat=ConstantArray[0,{d,d}];
       Do[
         Do[
            ToeplitzNoiMat⟦i,j⟧=GaussNoiList⟦Abs[i-j]+1⟧+Sign[i-j]*I*GaussNoiList⟦Ab
            ,{j,1,d}];
      ,{i,1,d}];
      hatH=Hmat+ToeplitzNoiMat;
      (*construct complex Hermite-Toeplitz Gaussian Noise matrix of S*)
      GaussNoiList=RandomVariate[NormalDistribution[0,costS*σ],2*d-1];
      ToeplitzNoiMat=ConstantArray[0,{d,d}];
       Do[
         Do[
            ToeplitzNoiMat⟦i,j⟧=GaussNoiList⟦Abs[i-j]+1⟧+Sign[i-j]*I*GaussNoiList⟦Ab
            ,{j,1,d}];
      ,{i,1,d}];
      hatS=Smat+ToeplitzNoiMat;
      Θ=th*σ;
      hatEK=thresholding[hatH,hatS,Θ];
      ϵ=Abs[hatEK-Eg];
      ϵList⟦i,j⟧=ϵ;
   ,{j,1,Length[MList]}];
,{i,1,rep}];
ϵList]
```

```
funThrPracF[MList_,rep_,Hmat_,Smat_,d_,costH_,costS_,th_,Eg_]:=Module[{ϵList,σ,hatH,
ϵList=ConstantArray[0,{rep,Length[MList]}];
Do[
   Do[
      σ=1/Sqrt[MList⟦j⟧];
      hatH=Hmat+RandomVariate[NormalDistribution[0,costH*σ],{d,d}];
      hatH=(hatH+ConjugateTranspose[hatH])/2;(*Real Hermitian Gaussian noise matrix*)
      hatS=Smat+RandomVariate[NormalDistribution[0,costS*σ],{d,d}];
      hatS=(hatS+ConjugateTranspose[hatS])/2;
      Θ=th*σ;
      hatEK=thresholding[hatH,hatS,Θ];
      ϵ=Abs[hatEK-Eg];
      ϵList⟦i,j⟧=ϵ;
   ,{j,1,Length[MList]}];
,{i,1,rep}];
ϵList]
```