In[●]:=

Dicke Hamiltonian

In[ ]:=
```
(*Constants*)
bsize = 25; ω0 = 1.0; ωc  = 1.0; K  = 4; j  = 0.07;
(*Identity matrices for TLS and QHO*)
idTSS = SparseArray[IdentityMatrix[2]];
idHO = SparseArray[IdentityMatrix[bsize]];

(*TLS initial Hamiltonian*)
H0TSS = SparseArray[Band[{1, 1}] → {ω0/2, -ω0/2}];
(*QHO Hamiltonian*)
H0HO = ωc * SparseArray[Band[{1, 1}] → Table[n + 1/2, {n, 0, bsize - 1}]];
(*TLS raising and lowering operators*)
σm = {{0, 0}, {1, 0}};
σp = {{0, 1}, {0, 0}};
σx  =  σm + σp;

(*Annihilation operator definition*)
a = SparseArray[Band[{1, 2}] → Table[Sqrt[n], {n, 1, bsize - 1}], {bsize, bsize}];

(*Scaled harmonic oscillator Hamiltonian,
using convention with TLS on the left.*)
Htot  =  KroneckerProduct[IdentityMatrix[2^K], H0HO];

Do[
  (*Tensor product adjustment for the i-th TLS*)
  leftIds = If[i > 1, Table[idTSS, {i - 1}], {IdentityMatrix[1]}];
  rightIds = If[i < K, Table[idTSS, {K - i}], {IdentityMatrix[1]}];
  (*TLS Hamiltonian for the i-th TLS*)
  H0TSSi = KroneckerProduct[
    KroneckerProduct[Sequence @@ leftIds, H0TSS], Sequence @@ rightIds];
  (*Print[Normal[H0TSSi]//MatrixForm];*)
  (*Adding ith TLS Hamiltonian to the total Hamiltonian*)
  Htot += KroneckerProduct[H0TSSi, idHO];


  σxi = KroneckerProduct[
    KroneckerProduct[Sequence @@ leftIds, σx], Sequence @@ rightIds];
  Htot += j * (KroneckerProduct[σxi, a] + KroneckerProduct[σxi, a†]);
  , {i, K}];
```

Initial State

In[ ]:=
```
ψ0[w_, x0_] = 1/(Sqrt[Sqrt[π] w]) Exp[-(x - x0)^2/(2 w^2)];(*Define initial Gaussian state*)
EigState[n_, x_] = π^(-1/4)/(Sqrt[2^n n!]) Exp[-x^2/2] HermiteH[n, x];
coeff[n_, w_, x0_] := NIntegrate[EigState[n, x] × ψ0[w, x0],
   {x, -∞, ∞}, PrecisionGoal → 12, AccuracyGoal → 5]
ψ0HO = SparseArray[{1 → 1.0}, bsize];
(*ψ0HO=Table[coeff[n,1,0],{n,0,bsize-1}];*)
(*α=3.5;
ψ0HO=Table[Exp[-Abs[α]^2/2]*(α^n/Sqrt[n!]),{n,0,bsize-1}];
(*in number/fock basis*)
ψ0HO=SparseArray[ψ0HO];*)
Print[Total[ψ0HO^2]];
Print[Normal[ψ0HO]];

(*excited states in TSS and Gaussian in the QHO*)
ψ0Vec = KroneckerProduct[{1, 0}, {1, 0}, {0, 1}, {0, 1}, ψ0HO] // Flatten;
Print["Norm of initial state: ", Norm[ψ0Vec]];
Print[Norm[ψ0Vec]];
```

```
1.
{1., 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
Norm of initial state: 1.
1.
```

Observable Matrices

Oscillator Position

In[ ]:=
```
xM = KroneckerProduct[IdentityMatrix[2^K], 1/(Sqrt[2]) (a† + a)];
(*Position of the oscillator*)
(*Expected x value for initial state*)
ConjugateTranspose[ψ0Vec].xM.ψ0Vec
```

Out[ ]=
```
0.
```

Projection Operator Construction

In[•]:=
```
excitedStateProjection[i_Integer] := Module[
  {
   idTSS = IdentityMatrix[2],
   partialExcitedProj = {{1, 0}, {0, 0}},
   leftIds, rightIds, excitedProj
  },
  leftIds = If[i > 1, Table[idTSS, {i - 1}], {IdentityMatrix[1]}];
  rightIds = If[i < K, Table[idTSS, {K - i}], {IdentityMatrix[1]}];
  excitedProj = KroneckerProduct[KroneckerProduct[
     Sequence @@ leftIds, partialExcitedProj, Sequence @@ rightIds], idHO];
  excitedProj (*Return the constructed operator*);

  excitedProj]
(*excitedStateProjection[1]//MatrixForm*)
```

Propagation

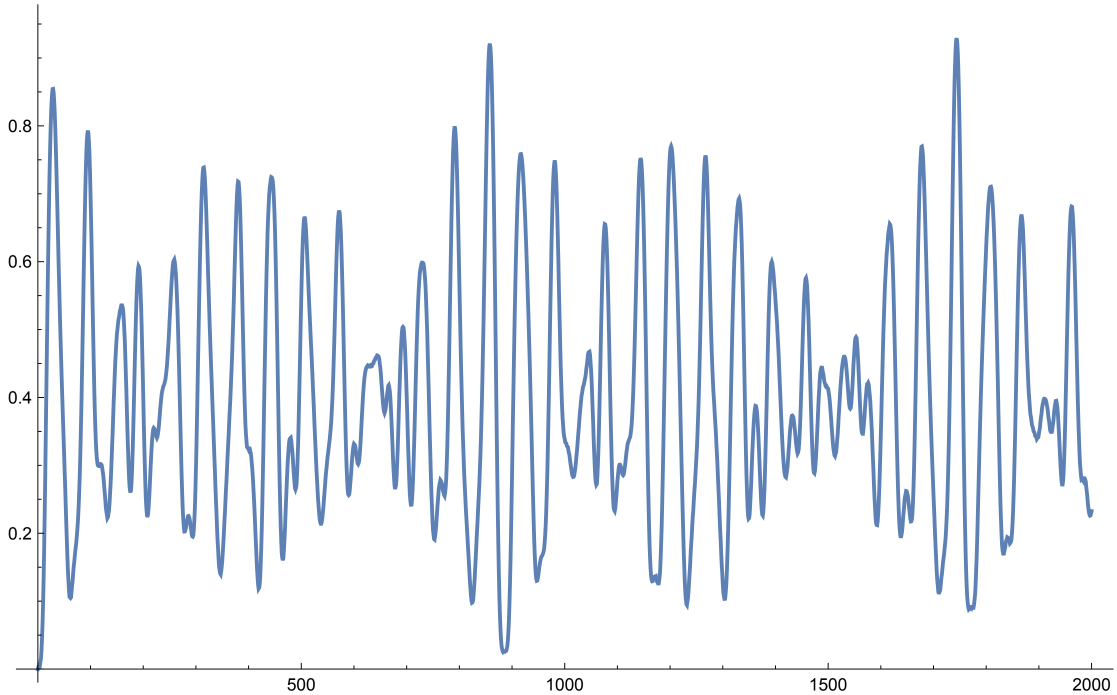
Oscillator Expected Position

In[•]:=
```
stateVector[t_] := MatrixExp[-I * Htot * t, ψ0Vec];
tMax = 2000;
tRange = Range[0, tMax, 1];
ψs = ParallelTable[stateVector[t], {t, tRange}];
(*xAve=Table[Conjugate[ψs〚n〛].xM.ψs〚n〛,{n,Length@tRange}];
ListLinePlot[{tRange,xAve//Re}//Transpose, ImageSize→Full]*)
```

Expected Excited State Populations

In[•]:=
```
pExcited1 = excitedStateProjection[3];
exAve1 = Table[Conjugate[ψs⟦n⟧].pExcited1.ψs⟦n⟧, {n, Length@tRange}];
ListLinePlot[{tRange, exAve1 // Re} // Transpose, ImageSize → Large]
(*pExcited2 = excitedStateProjection[2];
exAve2=Table[Conjugate[ψs⟦n⟧].pExcited2.ψs⟦n⟧,{n,Length@tRange}];
ListLinePlot[{tRange,exAve2//Re}//Transpose]*)
```

Out[•]=

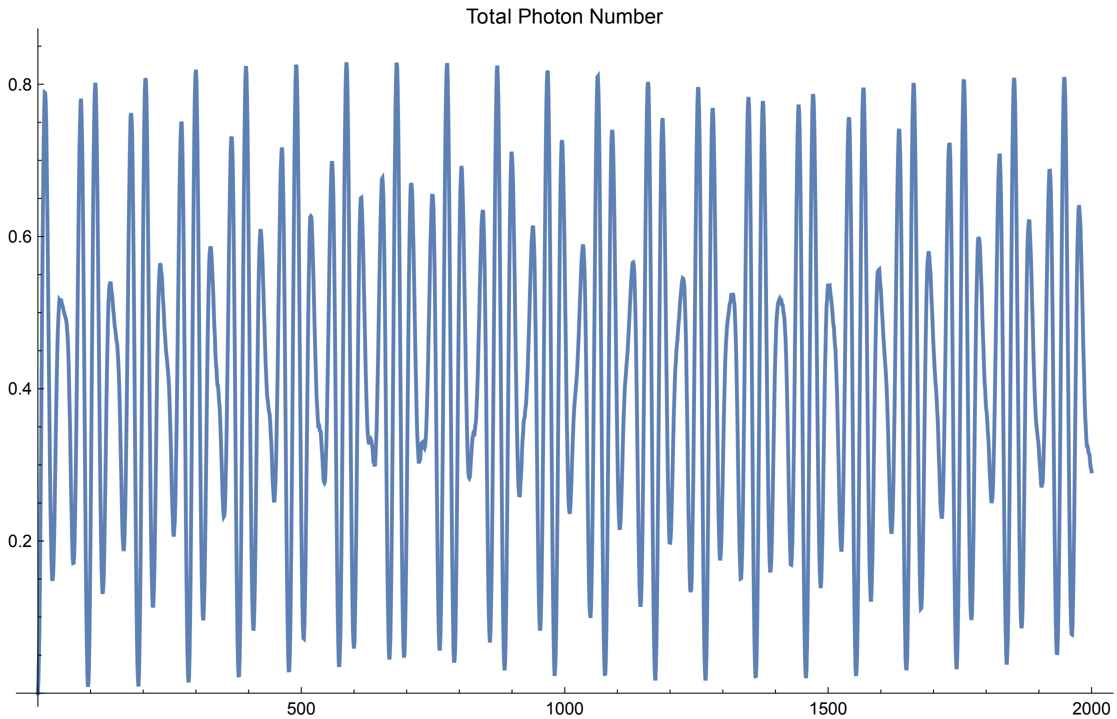


Superradiance

We keep the same initial state, but now we plot the photon emission statistics. We also change our time-scale as needed.

In[ ]:=
```
newtMax = 2000;
newtRange = Range[0, newtMax, 1];
aDaggerA = KroneckerProduct[IdentityMatrix[2^K], a†.a];
aDaggerAsr = aDaggerA.aDaggerA;
photons = Table[Conjugate[ψs⟦n⟧].aDaggerA.ψs⟦n⟧, {n, Length@newtRange}];
ListLinePlot[{newtRange, photons // Re} // Transpose,
 PlotRange → All, PlotLabel → "Total Photon Number", ImageSize → Large]
newPhotons =
  Table[Conjugate[ψs⟦n⟧].aDaggerAsr.ψs⟦n⟧, {n, Length@tRange}] - photons^2;
ListLinePlot[{tRange, newPhotons // Re} // Transpose,
 PlotRange → All, PlotLabel → "Photon Variance", ImageSize → Large]
```

Out[ ]=



Out[ ]=