Initialization

```
$HistoryLength = 0;
```

Load packages

```
<< qmatrix.m
```

```
Needs["Notation`"]
```

Define symbols

```
Symbolize[γ₁];
Symbolize[γϕ];
Symbolize[HJ-C];
Symbolize[H₂];
Symbolize[σˣ];
Symbolize[σʸ];
Symbolize[σᶻ];
Symbolize[σ⁺];
Symbolize[σ⁻];
Symbolize[â];
Symbolize[â†];
```

```
Symbolize[ωa];
InfixNotation[·, NonCommutativeMultiply];
Symbolize[T₁];
Symbolize[T₂];
Symbolize[t₁];
Symbolize[t₂];
Symbolize[H₀];
Symbolize[ωr];
Symbolize[ωd];
Symbolize[Hd];
Symbolize[Δd];
```

```
Symbolize[n̂];
Symbolize[q̂];
Symbolize[α_r];
Symbolize[E_J];
Symbolize[E_C];
Symbolize[n_g];
Symbolize[H_Q];
Symbolize[H_g];
Symbolize[ℒ_];
Symbolize[ρ̂];
Symbolize[ĝ];
```

System modes

```
qubitletter = Characters["GEFH"] ~ Join ~ CharacterRange["J", "Z"];
```

```
levels::usage =
  "levels represents the number of levels kept in the truncation of the
    qubit and cavity Hilbert spaces. Change it only using setlevels[]";
```

```
setlevels::toofew = "Too few levels `1`; at least 2 needed";
setlevels::usage = "setlevels[n] sets things
    up to keep n transmon levels and n cavity levels";
setlevels[n_Integer ? (# > 1 || Message[setlevels::toofew, #] &)] := (
  Unprotect[levels];
  levels = n;
  Protect[levels];
  setSystem[qubit, cavity];
  setModeType[qubit, {bosonic, levels}];
  setModeType[cavity, {bosonic, levels}];
  "System set to dimension: " <> ToString@dimension[system])
```

Notations

Superoperators

```
𝒟 /: 𝒟[A_matrix ? properMatrixQ][ρ_matrix ? properMatrixQ] :=
 A · ρ · hc[A] - hc[A] · A · ρ / 2 - ρ · hc[A] · A / 2
```

Operators

```
σ⁺ := matrix[op[ad, qubit]];
σ⁻ := matrix[op[a, qubit]];
â† := matrix[op[ad, cavity]];
â := matrix[op[a, cavity]];
n̂ := â† · â;
q̂ := σ⁺ · σ⁻;
```

```
AddInputAlias["sp" → σ⁺];
AddInputAlias["sm" → σ⁻];
AddInputAlias["ad" → â†]
AddInputAlias["nh" → n̂];
AddInputAlias["qh" → q̂];
```

Options

```
SetOptions[Manipulator, Appearance → "Labeled"];
```

Transmon Calculations

Do the matrix solve

This function egtrans[] gives the eigenenergies e_j and the coupling terms g_{ij} and then also calculates the derivative of these wrt E_J / E_C.
Because it calculates the derivative by 1st-order perturbation theory, it has problems with degeneracies when E_J / E_C is low enough compared to cutoff that there are levels with (almost) degeneracies at $n_g \in \{0, 1/2\}$. Consider using $n_g = 0.5 + \epsilon$ instead.
We have to manually correct the signs of the g_{ij} because `Eigensystem[]` doesn't guarantee a consistent phase for the eigenvectors.
I haven't checked whether it's better to use a sparse solver or the dense one, but either way we need to get all of the eigenstates for the perturbation theory, so we should not use Krylov methods.
We also normalize things so that $e_0 \equiv 0$, $e_1 \equiv 1$, $g_{12} = g_{21} \equiv 1$.

```
egtrans::usage =
  "egtrans[ng, EjEc, cutoff] gives {e, g, ⅆe/ⅆ(Ej/Ec), ⅆg/ⅆ(Ej/Ec)}";
egtrans::toofew = "Cutoff `1` is too low; must be at least 2";
Block[{fx, gx, hx, part, x},
```

```
Hold[egtrans[ng_?NumericQ, EjEc_?NumericQ,
      cutoff_Integer?(# > 1 || Message[egtrans::toofew, #] &)] := Module[
      {h = SparseArray[{Band[{1, 1}] → 4 (Range[-cutoff, cutoff] - ng)^2}],
       hv = SparseArray[
         {Band[{1, 2}], Band[{2, 1}]} → -1., {2 cutoff + 1, 2 cutoff + 1}],
       n = SparseArray[{Band[{1, 1}] → Table[m - ng, {m, -cutoff,
             cutoff}]}], e, v, e2, v2, o, g, de, dv2, dg, sgn},
      {e, v} = Eigensystem[h + EjEc/2 hv];
      o = Ordering@e;
      e2 = e[[o]];
      v2 = v[[o]];
      g = v2.n.v2^T;
      sgn = Sign@g;
      g = sgn g;
      de = #.hv.# & /@ v2 / 2;
      dv2 = Table[Sum[If[i == j, 0, (v2[[j]] (v2[[j]].hv.v2[[i]]))/(e2[[i]] - e2[[j]])],
         {j, 2 cutoff + 1}], {i, 2 cutoff + 1}];
      dg = sgn (dv2.n.v2^T + v2.n.dv2^T) / 2;
      {(e2 - e2[[1]])/(e2[[2]] - e2[[1]]),
       (D[(fx[x] - gx[x])/(hx[x] - gx[x]), x] /. {fx'[x] → de, fx[x] → e2,
            gx'[x] → part[de, 1], gx[x] → part[e2, 1],
            hx'[x] → part[de, 2], hx[x] → part[e2, 2]}) //
         FullSimplify // Experimental`OptimizeExpression,
       g/g[[1, 2]],
       (D[fx[x]/gx[x], x] /. {fx[x] → g, fx'[x] → dg,
            gx[x] → part[g, 1, 2], gx'[x] → part[dg, 1, 2]}) //
         FullSimplify // Experimental`OptimizeExpression}
     ];
    ] /. x_Experimental`OptimizeExpression :→ RuleCondition[x] /.
   Experimental`OptimizedExpression[x_] :→ x /.
  HoldPattern[part] → Part // ReleaseHold;
]
```

Now we need to interpolate the results of the numerical calculation of e_i and g_{ij}.

The indices i,j are zero-based...

Interpolation of the solutions

```
energyinterp::usage =
  "energyinterp[{f2, f3, ...}, ng, cutoff, {min, max, step}] represents
    a function that interpolates the transmon energies.";
couplinginterp::usage = "energyinterp[{f2, f3, ...}, ng,
     cutoff, {min, max, step}] represents a
    function that interpolates the transmon couplings.";

interpf::level =
  "Tried to calculate for transmon level:`1`, but interpolating
    function was only defined for levels 0..`2`";
interpf::dom = "Tried to calculate for EJ/EC of `1`, but
    interpolating function was only defined for `2`≤ EJ/EC ≤`3`";
interpf::invalidform = "Invalid form for a transmon interpolation";
```

```
Unprotect[energyinterp, couplinginterp];
```

```
idx::usage = "idx[] has the attribute NHoldAll";
SetAttributes[idx, NHoldAll];
```

```
transmoninfo[ng_, c_, {min_, max_, step_}] :=
  Column[{"Ei[EJ/EC]", "i:0.." <> ToString[c], HoldForm[min ≤ "EJ/EC" ≤ max],
    "interp step: " <> ToString@step, HoldForm["ng" == ng]}];
```

- **energyinterp[]**

```
energyinterp[a__][i : Except[_idx]] := energyinterp[a][idx@i];

energyinterp[___][idx@0] = 0. &;
energyinterp[___][idx@1] = 1. &;

energyinterp[_, _, c_, _][idx@i_] /;
   (If[NumericQ[i] && ! TrueQ[0 ≤ i ≤ c && i ∈ Integers],
     Message[interpf::level, i, c]; Abort[]];
    False) := None;

energyinterp[l_, _, c_, {min_, max_, _}][idx@i_][x_] /;
   (If[NumericQ[x] && ! TrueQ[min < x < max],
     Message[interpf::dom, x, min, max]; Abort[]];
    NumericQ[x] && NumericQ[i] && min ≤ x ≤ max && 2 ≤ i ≤ c) := l⟦i - 1⟧[x];

Derivative[d_Integer /; d ≥ 1][
     energyinterp[l_, _, c_, {min_, max_, _}][idx@i_]][x_] /;
   (If[NumericQ[x] && ! TrueQ[min < x < max],
     Message[interpf::dom, x, min, max]; Abort[]];
    NumericQ[x] && NumericQ[i] && min ≤ x ≤ max && 2 ≤ i ≤ c) :=
  Derivative[d][l⟦i - 1⟧][x];

Format[energyinterp[l : {__InterpolatingFunction},
      ng_?NumericQ, c_Integer ? (2 ≤ # &), mm : {min_, max_, step_} /;
       0 < min < max && 0 < 10 step < max - min][idx@i_] /; Length[l] + 1 == c] :=
  Tooltip[HoldForm["E"_i], transmoninfo[ng, c, mm]];

Format[energyinterp[l : {__InterpolatingFunction},
    ng_?NumericQ, c_Integer ? (2 ≤ # &), mm : {min_, max_, step_} /;
     0 < min < max && 0 < 10 step < max - min]] := energyinterp["<>", ng, c, mm];
```

- ## couplinginterp[]

```
couplinginterp[___][idx@1, idx@0] =
  couplinginterp[___][idx@0, idx@1] = 1. &;

couplinginterp[a_, b_, c_, d_][i : Except[_idx], j : Except[_idx]] :=
 couplinginterp[a, b, c, d][idx@i, idx@j]

couplinginterp[_, _, c_, _][idx@i_, idx@j_] /;
   (If[NumericQ[i] && ! TrueQ[0 ≤ i ≤ c && i ∈ Integers],
     Message[interpf::level, {i, j}, c]; Abort[]];
    If[NumericQ[j] && ! TrueQ[0 ≤ j ≤ c && j ∈ Integers],
     Message[interpf::level, {i, j}, c]; Abort[]];
    False) := None;

couplinginterp[l_, _, c_, {min_, max_, _}][idx@i_, idx@j_][x_] /;
   (If[NumericQ[x] && ! TrueQ[min < x < max],
     Message[interpf::dom, x, min, max]; Abort[]];
    NumericQ[x] && NumericQ[i] && NumericQ[j] && min ≤ x ≤ max &&
     0 ≤ i ≤ c && 0 ≤ j ≤ c) := l⟦i + 1, j + 1⟧[x];
Derivative[d_][couplinginterp[l_, _, c_, {min_, max_, _}][idx@i_, idx@j_]][
    x_] /;
   (If[NumericQ[x] && ! TrueQ[min < x < max],
     Message[interpf::dom, x, min, max]; Abort[]];
    NumericQ[x] && NumericQ[i] && NumericQ[j] && min ≤ x ≤ max &&
     0 ≤ i ≤ c && 0 ≤ j ≤ c) := Derivative[d][l⟦i + 1, j + 1⟧][x];

Format[couplinginterp[l_, ng_?NumericQ, c_Integer?(2 ≤ # &),
      mm : {min_, max_, step_} /; 0 < min < max && 0 < 10 step < max - min]
     [idx@i_, idx@j_] /; Dimensions[l] == {c, c} + 1] :=
  Tooltip[HoldForm[g_ij], transmoninfo[ng, c, mm]];

Format[couplinginterp[l_?MatrixQ, ng_?NumericQ, c_Integer?(2 ≤ # &),
    mm : {min_, max_, step_} /; 0 < min < max && 0 < 10 step < max - min]] :=
  couplinginterp["<>", ng, c, mm];
```

- ## Finish up defining tags

```
(e : energyinterp[___][_][_])* ^:= e;
```

```
(c : couplinginterp[___][__][_])* ^:= c;
SetAttributes[{energyinterp, couplinginterp}, {NHoldAll}];
Protect[energyinterp, couplinginterp];
```

```
SetAttributes[evalinterp, HoldAll];
evalinterp[x_] := x /. {idx[i_] :→ i,
   energyinterp[{l__}, __] :→ ({0, 1, l}[[# + 1]] &),
   couplinginterp[l_, __] :→ (l[[#1 + 1, #2 + 1]] &)}
```

Construct interpolations

```
makeinterp::usage =
  "makeinterp[ng, cutoff, levels, {min, max, step}] gives e[i, Ej/Ec],
    g[i, j, Ej/Ec] for min ≤ Ej/Ec ≤ max, (i,j = 0,...,levels-1),
    using 2cutoff+1 charge-basis transmon levels in the calculation";
makeinterp::levcut = "Require 2≤levels≤cutoff , but levels=`1`, cutoff=`2`";
makeinterp::step = "Require 0 < 10*step < max-min";
makeinterp::minmax = "Require 0<min<max but min=`1`, max=`2`";

Options[makeinterp] = {InterpolationOrder → 7};
```

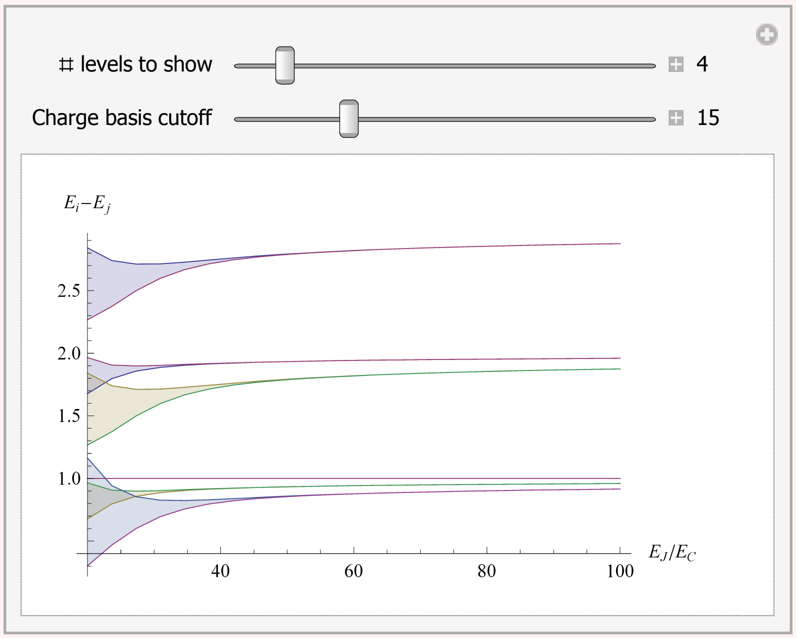
```
makeinterp[ng_?NumericQ, cutoff_Integer, levels_Integer, mms :
    {min_?NumericQ, max_?NumericQ, step_?NumericQ}, OptionsPattern[]] /;
  (2 ≤ levels ≤ cutoff || Message[makeinterp::levcut, levels, cutoff]) &&
   (0 < min < max || Message[makeinterp::minmax, min, max]) &&
   (0 < 10 step < max - min || Message[makeinterp::step]) :=
 Module[{egtab, x},
  egtab = Table[{N@x, egtrans[ng, N@x, cutoff]}, {x, min, max, step}];

  {energyinterp[
    Table[
     Interpolation[Cases[egtab, {x_, {e_, de_, _, _}} :→ {{x}, e[[i]], de[[i]]}],
      InterpolationOrder → OptionValue[InterpolationOrder]],
     {i, 3, levels}], ng, levels - 1, mms],
   couplinginterp[Table[Interpolation[
      Cases[egtab, {x_, {_, _, g_, dg_}} :→ {{x}, g[[i, j]], dg[[i, j]]}],
      InterpolationOrder → OptionValue[InterpolationOrder]],
     {i, levels}, {j, levels}], ng, levels - 1, mms]}
 ]
```

Check the transmon calculations

What does it look like?

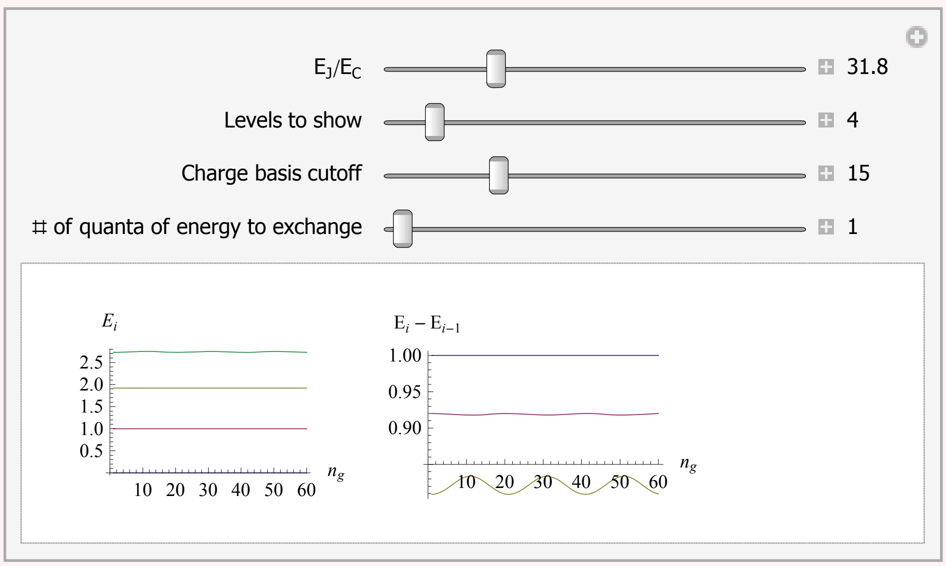
Spectrum vs E_J / E_C

```
Manipulate[Module[
  {x = Transpose[Table[egtrans[ng, ejec, cut][[1, ;; ls]], {ejec, 10, 100, 4},
      {ng, {0.0001, 0.5001}}], {2, 3, 1}]},
  Show[
   Table[
    ListLinePlot[Flatten[x[[δ + 1 ;;]] - x[[ ;; -(δ + 1)]], {1, 3}], PlotRange → All,
     AxesLabel → {"EJ/EC", "Ei-Ej"}, Filling → Table[2 n - 1 → {2 n}, {n, ls - δ}],
     DataRange → {20, 100}], {δ, 1, ls - 1}]]],
 {{ls, 4, "# levels to show"}, 3, cut, 1},
 {{cut, 15, "Charge basis cutoff"}, 10, 30, 1}]
```



Energy levels and spectra vs n_g

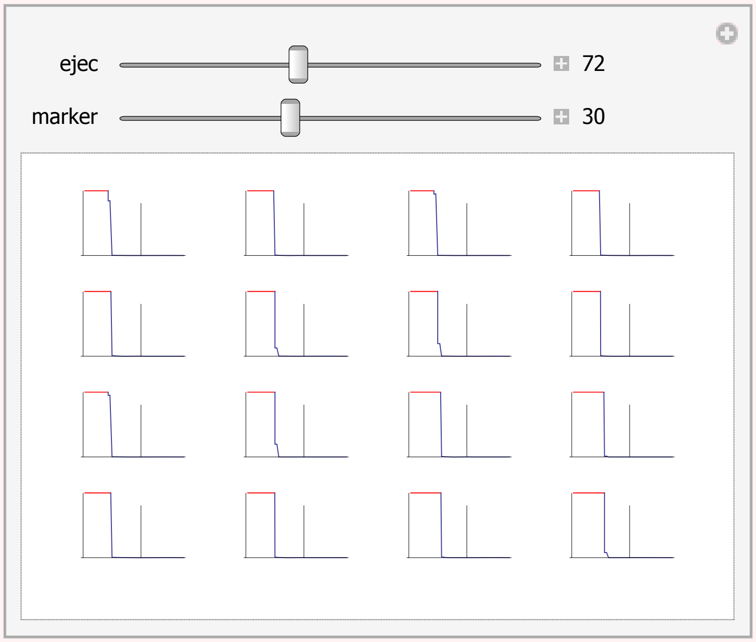
```
Manipulate[Module[
  {x = Table[egtrans[ng, egec, cut][[1, ;; ls]], {ng, -.4999, .5, .05}], xxx},
  xxx = Flatten[Table[x, {3}], 1]ᵀ;
  GraphicsRow[{
    ListLinePlot[xxx, AxesLabel → {"n_g", "E_i"}],
    ListLinePlot[xxx[[δ + 1 ;;]] - xxx[[ ;; -(δ + 1)]], PlotRange → All,
     AxesLabel → {"n_g", With[{δ = δ}, HoldForm["E"_i - "E"_(i-δ)]]}]}]],
 {{egec, 50., "E_J/E_C"}, 10., 100.},
 {{ls, 4, "Levels to show"}, 3, cut, 1},
 {{cut, 15, "Charge basis cutoff"}, 10, 30, 1},
 {{δ, 1, "# of quanta of energy to exchange"}, 1, ls - 1, 1}]
```



- ## Choose a cutoff

```
et[ng_?NumericQ, EjEc_?NumericQ, cutoff_?IntegerQ] := Module[{e, v, v2},
   {e, v} = Eigensystem[SparseArray[
      {{i_, i_} :> 4 (i - Floor[cutoff / 2] - ng - 1)^2,
       {i_, j_} /; Abs[i - j] == 1 → -EjEc / 2}, {cutoff, cutoff}]];
   v2 = v[[Ordering[e]]];
   v2.DiagonalMatrix[Table[m - Floor[cutoff / 2], {m, 0, cutoff - 1}]].v2ᵀ];
```

```
Manipulate[Module[{ccm = 60, etm, ll = 4},
  etm = Abs[et[.5, ejec, ccm][[ ;; ll,  ;; ll]]];
  GraphicsGrid@Map[ListLinePlot[#, PlotRange → {0, 10^-12},
      Ticks → Dynamic@{{{marker, "", {.5, 0}}}, None}, ClippingStyle → Red] &,
    Transpose[Table[Abs[Abs[et[.5, ejec, cc][[ ;; ll,  ;; ll]] - etm],
      {cc, 2 ll + 1, ccm}], {3, 1, 2}], {2}]],
 {{ejec, 72}, 30, 130}, {{marker, 30}, 10, 60, 1}]
```



Mathieu function calculation, for comparison

```
k[m_, ng : _] := Sum[(Round[2 ng + l / 2] ~Mod~ 2)
    (Round[ng] - l (-1)^m ((m + 1) ~Quotient~ 2)), {l, {-1, 1}}];
aν_[x_] := MathieuCharacteristicA[ν, x];
Em_[ng : _, EJ : _, EC : _] := EC a-2 (ng-k[m,ng]) [-EJ/(2 EC)];
```

Quantities derived from the transmon solutions

$$\epsilon_{m\_}[E_J : \_, E_C : \_] := (-1)^m E_C \frac{2^{4m+5}}{m!} \sqrt{\frac{2}{\pi}} \left(\frac{E_J}{2 E_C}\right)^{\frac{m}{2}+\frac{3}{4}} e^{-\sqrt{8 E_J/E_C}}$$

```
ε̃m_[EJ : _, EC : _] := Abs[Em[0.0001, EJ, EC] - Em[0.4999, EJ, EC]]
```

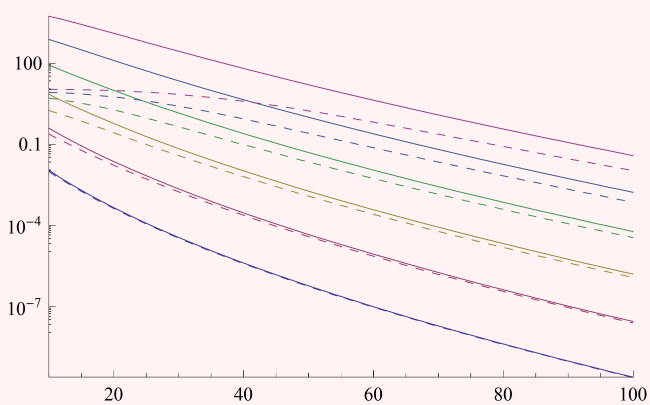
```
ϵ[_] := Σ_m^levels ϵ_m[72, 1] matrix@op[basis, qubit, m]
```

```
E_{m_,n_} = E_m[.0001, EjEc, 1] - E_n[.0001, EjEc, 1];
En[EjEc_?NumericQ]_{m_,n_} := Module[{q = etrans[.5, EjEc]}, q[[m + 1]] - q[[n + 1]]]
```

```
H_Q[EjEc_] := Σ_{m=0}^{levels-1} (En[EjEc]_{m0} / En[EjEc]_{10}) matrix@op[basis, qubit, m + 1];
```

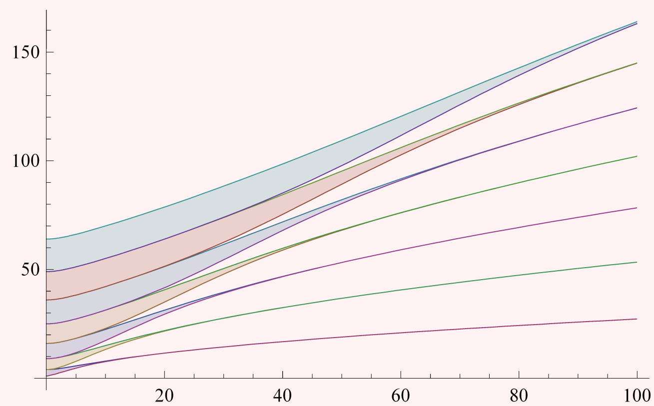
■ Asymptotic expression compared with exact

```
Show[LogPlot[Evaluate@Table[Tooltip[Abs[ϵ_m[E_J, 1]], m], {m, 0, 5}],
   {E_J, 10, 100}, PlotRange → All],
  LogPlot[Evaluate@Table[Tooltip[Abs[ϵ̃_m[E_J, 1]], m], {m, 0, 5}],
   {E_J, 10, 100}, PlotRange → All, PlotStyle → Dashed],
  Plot[x, {x, 0, 100}]]
```



- **Transmon dispersion**

```
Plot[Evaluate@Table[{
    Tooltip[E_m[0.00001, E_J, 1] - E_0[0.00001, E_J, 1], m],
    Tooltip[E_m[0.4999, E_J, 1] - E_0[0.00001, E_J, 1], m]}, {m, 1, 7}],
 {E_J, 0, 100}, PlotRange → All, Filling → Table[2 n - 1 → {2 n}, {n, 7}]]
```



Solve the system

- **Parameters**

NB: These quantites are protected because everything here depends on them being symbols.
They should only have values assigned to them in a Block[] or similar structure.

```
params =
  {ωr, ωd, δ, g, ξ, ejec, γϕ, (*γϕ2,*)γ, pm, κ(*,pf1,pf2,pf3,pf4,pf5*)};
ωr::usage = "ωr is cavity frequency";
ωd::usage = "ωd is the frequency of the drive";
δ::usage = "δ is given by ωr-ωqubit == δ";
g::usage =
  "g is the coupling strength g01 (between the 0↔1 transition of the
    transmon and the cavity annihilation operator)";
ξ::usage = "ξ is the drive strength";
ejec::usage = "ejec is the EJ/EC ratio for the transmon";
γϕ::usage = "γϕ is the transmon dephasing strength";
γ::usage = "γ is the transmon relaxation rate";
κ::usage = "κ is the cavity relaxation rate";
pf1* ^= pf1;
pf2* ^= pf2;
pf3* ^= pf3;
pf4* ^= pf4;
pf5* ^= pf5;
Protect[Evaluate@params];
$Assumptions = params ∈ Reals && ℏ > 0;
```

Hamiltonian

Do the normal transmon interpolations

This is the standard interpolation:

```
$maxlevels::usage =
  "$maxlevels is the number of transmon levels calculated so
    far. We need to recalculate the interpolations
    and some other stuff if we want to go higher...";
Unprotect[$maxlevels];
$maxlevels = 8;
Protect[$maxlevels];

{ef1, gf1} = makeinterp[0.4999, 15, $maxlevels, {10, 200, 1}];
{ef2, gf2} = makeinterp[0.0001, 15, $maxlevels, {10, 100, 1}];
{ef1, gf1}
ef1[3][72]
```

```
{energyinterp[<>, 0.4999, 7, {10, 200, 1}],
 couplinginterp[<>, 0.4999, 7, {10, 200, 1}]}
```

```
2.84936
```

Subspace

Set up the basis states (sstates), the projectors onto the degenerate subspaces (psstates) and the size of the Hilbert space for subsequent calculations (nn):

```
ClearAll["bket*"];
sstates :=
 Table[Symbol["bket" <> qubitletter[[j]] <> ToString[i - j]], {i, levels}, {j, i}]
states := Flatten@sstates;
Array[
  (Evaluate@Symbol["bket" <> qubitletter[[#1]] <> ToString[#2 - 1]] := basisKet[
       qubit, #1 ] · basisKet[cavity, #2]) &, {$maxlevels, $maxlevels}];
psstates := projector /@ sstates;
nn := Length@states;
```

Set up the Hamiltonian

H_Q is in units of ω_{01}

```
setlevels[3]
{ef, gf} = {ef1, gf1};
```

```
System set to dimension: 9
```

```
HQ := ℏ Sum[ef[m][ejec] matrix@op[basis, qubit, m + 1], {m, 0, levels-1}];
(*like (â·σ⁺+â†·σ⁻) *)
ĝ := Sum[gf[i - 1, i][ejec] matrix@op[basis, qubit, i, i + 1], {i, levels-1}];
Hg := ℏ g (# + hc[#] &@(ĝ · â†));
```

We are in the rotating frame and make the RWA:

```
Hd := ℏ ξ (â + â†);
(* H0=(ω01 HQ-ωd q̂)+ℏ (ωc- ωd) n̂+g Hg *)
H0 := ((ωr - δ) HQ - ℏ ωd q̂) + ℏ (ωr - ωd) n̂ + Hg;
```

Here's the matrix version of the Hamiltonian (a list of the matrices in each n-excitation subspace, n=1...levels) :

```
H0s := Table[Simplify@Table[trace[hc[sstates[[n, i]]] · H0 · sstates[[n, j]]],
      {i, n}, {j, n}], {n, levels}];
```

Diagonalizing the Hamiltonian

```
diagfns::usage =
  "diagfns[] returns {energies[...],vectors[...]} functions.";
diagfns[] := Block[{ωd = 0, ℏ = 1, ef = ef1, gf = gf1,
   Eigenvalues, Eigenvectors, PadRight, map},
  With[{H0s = H0s, levels = levels, nn = nn},
   With[{sp = {ωr, δ, g, ejec}},
    {Function[Evaluate@sp, Evaluate[Eigenvalues /@ evalinterp[H0s]]],
     Function[Evaluate@sp,
       Evaluate[Table[With[{ic = i (i - 1) / 2}, PadRight[#, nn, 0., ic] &~
           map~Eigenvectors[H0s[[i]]]], {i, levels}]]] /. map → Map}]]]
```

```
diagfns2::usage =
  "diagfns2[] returns {energies[...],vectors[...]} functions.";
diagfns2[] := Block[{ℏ = 1, ef = ef1, gf = gf1,
   Eigenvalues, Eigenvectors, PadRight, map},
  With[{H0s = H0s, levels = levels, nn = nn},
   With[{sp = {ωr, ωd, δ, g, ejec}},
    {Function[Evaluate@sp, Evaluate[Eigenvalues /@ evalinterp[H0s]]],
     Function[Evaluate@sp,
       Evaluate[Table[With[{ic = i (i - 1) / 2}, PadRight[#, nn, 0., ic] &~
           map~Eigenvectors[H0s[[i]]]], {i, levels}]]] /. map → Map}]]]
```

Show the energy levels and transitions:

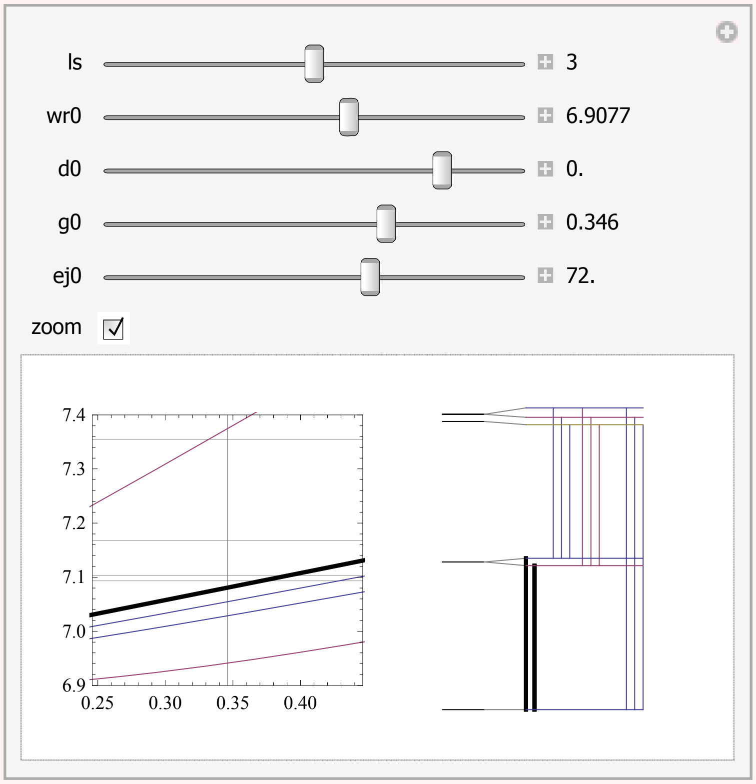
```
transAnn::usage =
  "transAnn[i₁,j₁,i₂,j₂] is a tag representing the transition
    between the j₁th level of the i₁-excitation subspace
    and the j₂th level of the i₂-excitation subspace";
levelAnn::usage = "levelAnn[i,j] is a tag representing
    the jth level of the i-excitation subspace";
Protect[transAnn, levelAnn];

$hilited::usage =
  "$hilited contains the tag of the currently selected item";

flash::usage = "flash[list,t] flashes
    between styles in the list l, over a total time t";
flash[l_List, t_] := l⟦Clock[{1, Length@l, 1}, t]⟧;
flashing[s_] :=
  flash[{Directive[s, Dashed], Directive[s, Dashing[{}]]}, 1];
maybeflashing[a_, s_] := Dynamic@If[a === $hilited, flashing@s, s];
handlemouse[g_] :=
  EventHandler[g, "MouseClicked" :→ ($hilited = MouseAnnotation[]),
   PassEventsDown → Automatic];

With[{x1 = 1, x2 = 2, x4 = 0.2`, x5 = 0.15`, x6 = 0.1`, x7 = 0.2`},
  leveldiagram[e0_List, e1_List, ls_Integer] :=
   DynamicModule[{q1, q2},
    {q1, q2} = (5 (ls - 1) # / #⟦-1, -1, -1⟧) &@{e0, e1};
    handlemouse@
     Graphics[Dynamic@Flatten[{Antialiasing → False,
         Table[{Line[{{0, q1[[i, j]]}, {x1, q1[[i, j]]}}]}, {i, ls}, {j, i}],
         Table[{Gray,
           Line[{{x1, q1[[i, j]]}, {x2, q2[[i, j]]}}]}, {i, ls}, {j, i}],
         Module[{xx = x2 - x4 - x5 - x6 - x7},
          Flatten[{Table[
             xx += KroneckerDelta[i, j1, j2, 1] x4 +
               KroneckerDelta[j1, j2, 1] x5 + KroneckerDelta[j2, 1] x6 + x7;
             With[{s = transstyle[i, j1, i + k, j2], a =
                 transAnn[i, j1, i + k, j2]},
              {maybeflashing[a, s],
               Annotation[Line[
                  {{xx, q2[[i, j1]]}, {xx, q2[[i + k, j2]]}}], a, "Mouse"]}],
             {k, ls - 1}, {i, ls - k}, {j1, i}, {j2, i + k}],
            Table[
             With[{s = levelstyle[i, j], a = levelAnn[i, j]}, {{{maybeflashing[
                   a, s], Annotation[Line[{{x2, q2[[i, j]]}, {xx, q2[[i,
                        j]]}}], a, "Mouse"]}}}], {i, ls}, {j, i}]}, 4]]}, 1]]]];
```

```
transsstyle[i1_, j1_, i2_, j2_] := Directive[
   Flatten@{ColorData[1][j1], If[i1 == 1 && i2 == 2, {Thick, Black}, {}],
     If[MatchQ[$hilited, levelAnn[i1, j1] | levelAnn[i2, j2]], Red, {}]}];
levelstyle[i_, j_] := ColorData[1][j];
```

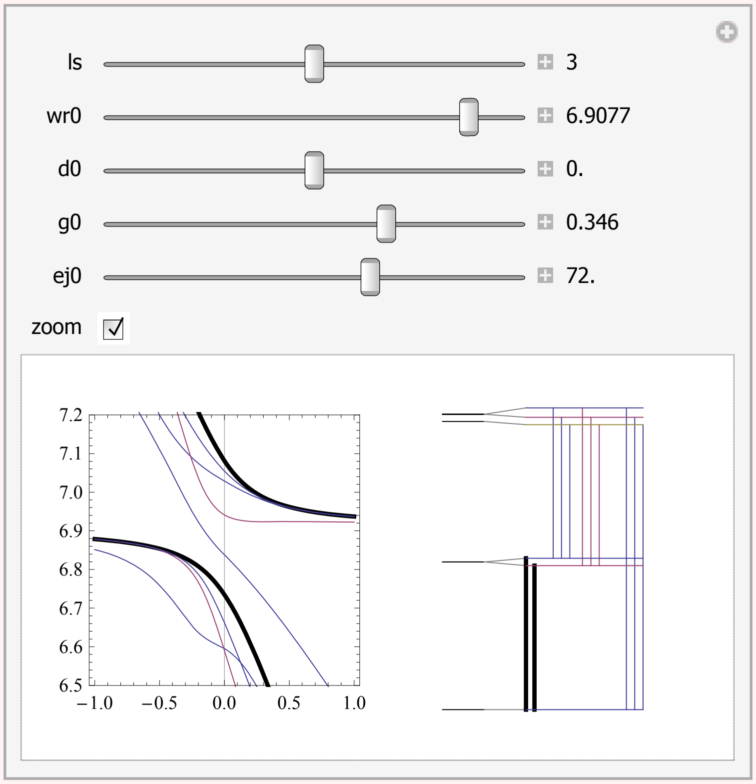
```
setlevels[4]
{energiestt, vectorstt} = diagfns[];
```

```
System set to dimension: 16
```

```
Manipulate[
 DynamicModule[{evtab},
  evtab = Table[{g, energiestt[wr0, d0, g / 2, ej0]}, {g, 0, g0 + .2, g0 / 10}];
  Deploy@GraphicsRow[
    {handlemouse@Graphics[
       Dynamic@Flatten[Table[
          With[{s = transstyle[i, j1, i + k, j2], a = transAnn[i, j1, i + k, j2]},
           {maybeflashing[a, s],
            Annotation[Line[{evtab[[All, 1]], (evtab[[All, 2, i + k, j2]] -
                   evtab[[All, 2, i, j1]]) / k}ᵀ], a, "Mouse"]}],
          {k, ls - 1}, {i, ls - k}, {j1, i}, {j2, i + k}], 3],
       Frame → True, AspectRatio → 1, PlotRangeClipping → True, PlotRange →
        Dynamic[If[zoom, {{g0 - .1, g0 + .1}, {6.9, 7.4}}, All]], GridLines →
        {{g0}, (*{7.365,7.11,7.175,7.31}*){7.355, 7.103, 7.168, 7.093}}],
     leveldiagram[energiestt[wr0, d0, 0, ej0],
      energiestt[wr0, d0, g0 / 2, ej0], ls]}]],
 {{ls, 3}, 2, levels, 1},
 {{wr0, 6.9077}, 6.89, 6.92},
 {{d0, 0.}, -.5, .1},
 {{g0, .346}, 0, .5},
 {{ej0, 72.}, 20, 100},
 {zoom, {True, False}},
 TrackedSymbols → Full,
 Bookmarks → {
   "get Ec" :→
    {ls = 3, wr0 = 6.917458, d0 = -.44265, g0 = 93.88 / 1000, ej0 = 52.12},
   "expt" :→ {ls = 3, wr0 = 6.915, d0 = -.006, g0 = 93.88 / 1000, ej0 = 50}}]
```


ls 3
wr0 6.9077
d0 0.
g0 0.346
ej0 72.
zoom
7.4
7.3
7.2
7.1
7.0
6.9
0.25 0.30 0.35 0.40

```
Manipulate[
 DynamicModule[{evtab},
  evtab = Table[{d0, energiestt[wr0, d0, g0 / 2, ej0]}, {d0, -1, 1, .01}];
  Deploy@GraphicsRow[{
     handlemouse@Graphics[
       Dynamic@Flatten[Table[
          With[{s = transstyle[i, j1, i + k, j2], a = transAnn[i, j1, i + k, j2]},
           {maybeflashing[a, s],
            Annotation[Line[{evtab⟦All, 1⟧, (evtab⟦All, 2, i + k, j2⟧ -
                   evtab⟦All, 2, i, j1⟧) / k}ᵀ], a, "Mouse"]}],
          {k, ls - 1}, {i, ls - k}, {j1, i}, {j2, i + k}], 3],
       Frame → True, AspectRatio → 1, PlotRangeClipping → True, PlotRange →
        If[zoom, {All, {6.5, 7.2}}, All], GridLines → {{d0}, None}],
     leveldiagram[energiestt[wr0, d0, 0, ej0],
      energiestt[wr0, d0, g0 / 2, ej0], ls]}]],
 {{ls, 3}, 2, levels, 1},
 {{wr0, 6.9077}, 6, 7},
 {{d0, 0.}, -1, 1},
 {{g0, .346}, 0, .5},
 {{ej0, 72.}, 20, 100},
 {zoom, {True, False}},
 TrackedSymbols → Full]
```



Density matrices

Lindblad operators

Here's the Lindblad form of the RHS of the master equation for $\dot{\rho}$:

```
pr := projector[states]
```

$$\mathcal{L}_1[\rho\_?\text{operatorMatrixQ}] := -\frac{i}{\hbar}\,\text{commutator}[H_0 + H_d, \rho] + \kappa\,\mathcal{D}[\hat{a}][\rho] + \gamma\,\mathcal{D}[\sigma^-][\rho] + \gamma\,\text{pm}\,\mathcal{D}[\text{pr}\cdot\sigma^+\cdot\text{pr}][\rho] + \gamma_\phi\,\mathcal{D}[\hat{q}][\rho]\big/2$$

$$\mathcal{L}_2[\rho\_?\text{operatorMatrixQ}] := -\frac{i}{\hbar}\,\text{commutator}[H_0 + H_d, \rho] + \kappa\,\mathcal{D}[\hat{a}][\rho] + \gamma\,\mathcal{D}[\hat{g}][\rho] + \gamma\,\text{pm}\,\mathcal{D}\big[\text{pr}\cdot\text{hc}[\hat{g}]\cdot\text{pr}\big][\rho] + 10^7\,\gamma_\phi\,\mathcal{D}\Big[\sum_{m=0}^{\text{levels}-1}(\text{ef1}[m][\text{ejec}] - \text{ef2}[m][\text{ejec}])\,\text{matrix@op}[\text{basis}, \text{qubit}, m+1]\Big][\rho]$$

```
ℒ3[ρ_?operatorMatrixQ] :=
  -(i/ħ) commutator[H0 + Hd, ρ] + κ 𝒟[â][ρ] + γ 𝒟[ĝ][ρ] + κ pm 𝒟[pr · â† · pr][ρ] + 10^7 γϕ
    𝒟[Sum[(ef1[m][ejec] - ef2[m][ejec]) matrix@op[basis, qubit, m + 1], {m, 0, levels-1}]][ρ]
```

```
ℒ4[ρ_?operatorMatrixQ] := -(i/ħ) commutator[H0 + Hd, ρ] + κ 𝒟[â][ρ] +
   γ 𝒟[ĝ][ρ] + γ pm 𝒟[pr · hc[ĝ] · pr][ρ] + κ pm 𝒟[pr · â† · pr][ρ] + 10^7 γϕ
    𝒟[Sum[(ef1[m][ejec] - ef2[m][ejec]) matrix@op[basis, qubit, m + 1], {m, 0, levels-1}]][ρ]
```

```
ℒ5[ρ_?operatorMatrixQ] :=
  -(i/ħ) commutator[H0 + Hd, ρ] + κ 𝒟[â][ρ] + γ 𝒟[ĝ][ρ] + γ pm 𝒟[pr · hc[ĝ] · pr][ρ] +
   κ pm 𝒟[pr · â† · pr][ρ] + 10^7 𝒟[Sum[pϕ[[m]] matrix@op[basis, qubit, m + 1], {m, 1, levels-1}]][ρ]
pϕ = {pf1, pf2, pf3, pf4, pf5};
```

```
ℒ6[ρ_?operatorMatrixQ] := -(i/ħ) commutator[H0 + Hd, ρ] + κ 𝒟[â][ρ] +
   γ 𝒟[ĝ][ρ] + γ pm 𝒟[pr · hc[ĝ] · pr][ρ] + κ pm 𝒟[pr · â† · pr][ρ] +
   10^7 γϕ 𝒟[Sum[(ef1[m][ejec] - ef2[m][ejec]) matrix@op[basis, qubit, m + 1], {m, 0, levels-1}]][
     ρ] + γϕ2 𝒟[q̂][ρ] / 2
```

Now put it in matrix form and project onto our reduced Hilbert space:

```
lindblad::trnz = "The trace of ρ̇ was not zero!";
lindblad::usage =
  "lindblad[ℒ] returns {ρ̂, ρij, ρ̇ij} for a given Lindblad operator ℒ[ρ̂]";

lindblad[ℒ_] := With[{nn = nn, states = states},
   Module[{ρs, ρ, Π, ℒρ, Πℒρ, δρ},
    ρs =
     Table[Symbol["ρ" <> ToString[i] <> "x" <> ToString[j]], {i, nn}, {j, nn}];
    ρ = Simplify[Sum[ρs⟦i, j⟧ states⟦i⟧ · hc[states⟦j⟧], {i, nn}, {j, nn}]];
    Π = Simplify@projector[states];
    ℒρ = ℒ[ρ];
    Πℒρ = Π · ℒρ · Π;
    δρ = Table[trace[hc[states⟦i⟧] · Πℒρ · states⟦j⟧], {i, nn}, {j, nn}];
    If[! TrueQ[Chop@FullSimplify@Tr@δρ == 0], Message[lindblad::trnz]];
    {ρ, ρs, δρ}]];
```

- **Steady state solver**

```
steadystatevalue[op_?operatorMatrixQ,
  pt : {(_?(MemberQ[params, #] &) → _?NumericQ) ...}] :=

 Block[Evaluate[Join[{sol, vparms}, params]],
  Evaluate[params] = params /. pt;
  vparms = Select[params, ! NumericQ[#] &];

  lusolve := 0^0;
  oldvec := 0^0;

  With[{sparms = Sequence @@ vparms, nn = nn},
   Module[{crys = CoefficientArrays[
       {Tr@ρs - 1} ~ Join ~ Rest[Flatten[ddd = δρ]], Flatten@ρs],

     M1, M2, c1, c2, cf1, cf2, cff1, cff2, M1c, M2c, cfm1,
     cfm2, cffm1, cffm2, ope, opc1, opc2, opm, ss, rparms, nparms,
     repparms, ρte, dρte, nrm, bb, cb, cfb, cffb, bbc, occ},

    nparms := Sequence @@ (Pattern[#, _?NumericQ] & /@ vparms);
    rparms = {#, _Real} & /@ vparms;

    M1 = FullSimplify[crys⟦1⟧];
    M2 = FullSimplify[-crys⟦2⟧^TT];
    bb = FullSimplify[Flatten[D[M2, {vparms}], {{3, 1}, {2}}]];

    c1 = M1 /. HoldPattern@SparseArray[__, {__, a_}] :→ a;
    c2 = M2 /. HoldPattern@SparseArray[__, {__, a_}] :→ a;
    cb = bb /. HoldPattern@SparseArray[__, {__, a_}] :→ a;

    repparms = Thread[vparms → Unique[vparms]];
    cf1 = Compile[Evaluate@rparms, Evaluate@Developer`ToPackedArray@
         evalinterp@c1, CompileOptimizations → All] /. repparms;
    cf2 = Compile[Evaluate@rparms, Evaluate@Developer`ToPackedArray@
         evalinterp@c2, CompileOptimizations → All] /. repparms;
    cfb = Compile[Evaluate@rparms, Evaluate@Developer`ToPackedArray@
         evalinterp@cb, CompileOptimizations → All] /. repparms;

    M1c = (M1 /. HoldPattern@SparseArray[a__, {b__, c_}] :→
        SparseArray[a, {b, cff1[sparms]}]);
    M2c = (M2 /. HoldPattern@SparseArray[a__, {b__, c_}] :→
        SparseArray[a, {b, cff2[sparms]}]);
    bbc = (bb /. HoldPattern@SparseArray[a__, {b__, c_}] :→
        SparseArray[a, {b, cffb[sparms]}]);

    cfm1 = Compile[Evaluate@rparms,
      Evaluate@Developer`ToPackedArray@evalinterp@Normal@M1];
    cfm2 = Compile[Evaluate@rparms, Evaluate@
```

```
      Developer`ToPackedArray@evalinterp@Normal@M2];

   ope = trace[op · ρ];
   occ = {sparms,
     ρs /. Thread[Flatten@ρs → Table[ss[sol, i], {i, Length@Flatten@ρs}]]};
   opm = ope /. Thread[Flatten@ρs → Table[ss[sol, i],
        {i, Length@Flatten@ρs}]];
   nrm = Total@Diagonal@ρs /. Thread[Flatten@ρs →
       Table[ss[sol, i], {i, Length@Flatten@ρs}]];
   {opc1, opc2} = CoefficientArrays[ope, Flatten@ρs];

   ReleaseHold[
    Hold[
      ρte[nparms] := Module[{sol, m1, o1},
        mmm = mat; (*m1=mat;
        o1=off;
        Quiet@Check[
          oldvec=sol=LinearSolve[m1,Normal@o1,Method→
              {"Krylov","Preconditioner"→(lusolve[#]&),MaxIterations→10,
               "StartingVector"→oldvec,Tolerance→10^-4}],

          numlu++;
          lusolve=LinearSolve[m1,Method→"Multifrontal"];
          oldvec=sol=lusolve[o1]];*)
        sol = LinearSolve[mat, off];
        Sow[occ1];
        result];

      dρte[nparms] := Module[{y, c, sol, ls},
        ls = LinearSolve[mat, Method → "Multifrontal"];
        c = off;
        y = ls[c];
        sol = -ls[Partition[B.y, nn^2]^T];
        {c2.y + c1, c2.sol}
       ];

     ] /. {HoldPattern[ρ] → ρ,
      HoldPattern@off → M1c,
      HoldPattern@mat → M2c,
      HoldPattern@B → bbc,
      HoldPattern@offm → cffm1[sparms],
      HoldPattern@matm → cffm2[sparms],
      HoldPattern@result → opm,
      HoldPattern@occ1 → occ,
      HoldPattern@normalize → nrm,
      HoldPattern@c1 → opc1,
      HoldPattern@c2 → opc2,
      HoldPattern@nn → nn
     } /.
```

```
      {ss → Part,
       cff1 → cf1,
       cff2 → cf2,
       cffb → cfb,

       cffm1 :→ cfm1,
       cffm2 :→ cfm2}];
    {vparms, ρte, dρte}]]]
```