Mathematica HEOM, general method with Matsubara frequencies and multiple baths per site

Written on 26.12.2014 by Tobias Kramer (email: tobias.kramer@mytum.de), tidied up 2014-2019
This notebook implements the HEOM algorithm as described in
[1] Tanimura & Kubo “Time Evolution of a Quantum System in Contact with a Nearly Gaussian-Markoffian Noise Bath”, J. Phys. Soc. Jpn. 58, pp. 101-114 (1989)
[2] Tanimura “Stochastic Liouville, Langevin, Fokker–Planck, and Master Equation Approaches to Quantum Dissipative Systems”, J. Phys. Soc. Jpn. 75, 082001 (2006)
[3] Kreisbeck & Kramer “Long-Lived Electronic Coherence in Dissipative Exciton Dynamics of Light-Harvesting Complexes”, J. Phys. Chem. Lett.319 2828-2833 (2012)
[4] Kramer et al: “Efficient calculation of open quantum system dynamics and time-resolved spectroscopy with distributed memory HEOM (DM-HEOM)”
and further references therein.
If you use this notebook for your research, please acknowledge this implementation and share your modifications publicly.

Physical constants

In[70]:=
```
ħSI = 1.05457148`200*^-34;
kBSI = 1.380650239684840458953781387048428305608616777679979316`200.*^-23;
invcmtoJoule =
  1.986456217327470903861498336971100097137908380913187891`200.*^-23;
```

In[73]:=
```
PRECISION = $MachinePrecision;
```

Basic parameters of the system and baths

Hamiltonian (system part)

In[74]:=
```
SITES = 2; (* Number of sites,
including ground state and double exciton states *)
```

In[75]:=
```
Hex = {{100, 50}, {50, 200}};
```

This is the complete Hamiltonian, including ground state and 2 exciton states

In[76]:=
```
Hex // MatrixForm
```

Out[76]//MatrixForm=

$$\begin{pmatrix} 100 & 50 \\ 50 & 200 \end{pmatrix}$$

In[77]:=
```
HMSI = SetPrecision[Hex * invcmtoJoule, PRECISION];
```

Integration time step

In[78]:=
```
hSI = SetPrecision[5 * 10^-15, PRECISION]; (* time-step for integration [s] *)
```

Bath parameters

In[79]:=
```
NMAX = 4; (* HEOM truncation depth *)
MATSUBARAS = 1; (* Number of Matsubara frequencies *)
```

In[81]:=
```
TSI = 200; (* temperature, common to all baths [K] *)
νinv = 50 * 10^-15; (* inverse bath correlation time [s] *)
λinvcm = 15; (* reorganization energy [1/cm] *)
Ωinvcm = 0; (* shift of spectral density peak,
use pairs of peaks +/- shifted for building structured J(ω) *)
```

In[85]:=
```
βSI = SetPrecision[1 / (kBSI TSI), PRECISION];
ν1SI = SetPrecision[2 π / (βSI ħSI), PRECISION];
νSI = SetPrecision[1 / νinv, PRECISION];
λSI = SetPrecision[λinvcm * invcmtoJoule, PRECISION];
ΩSI = SetPrecision[Ωinvcm * invcmtoJoule / ħSI, PRECISION];
```

In[90]:=
```
NUMBATHS = 2; (* total number of independent baths *)
MAXBATHSPERSITE = 1 ;(* maximum number of baths coupling to a site *)
BASTable = {
    {0}, (* C++ indexing starts at site 0 = ground state,
    -1 indicates NO coupling *)
    {1}};
(* for compatibility construct also the reverse map: *)
(* coupling: bath2sites[[b]][[s]] → bath b is coupled to site s *)
bath2sites = DeleteCases[
  Table[Flatten[Drop[Position[BASTable, s], {}, -1]], {s, 0, SITES - 1}], {}]
```

Out[93]=
```
{{1}, {2}}
```

In[94]:=
```
bath2sites[[1]] (* first bath connects to sites 1 in MM counting *)
bath2sites[[2]] (* second bath connects to sites 2 in MM counting) *)
```

Out[94]=
```
{1}
```

Out[95]=
```
{2}
```

Initialize each bath

No shift Ω for the moment...

In[96]:=
```
νNUMBATHS[ν_, Ω_] = Table[ν, {i, 1, NUMBATHS}]
λNUMBATHS[λ_] = Table[λ, {i, 1, NUMBATHS}]
```

Out[96]= {ν, ν}

Out[97]= {λ, λ}

HEOM: construction of auxiliary density operators (ADO) and indexing setup

In[98]:=
```
ADOWIDTH = NUMBATHS * MATSUBARAS;
ADOTuple =.;
ADOTuple = {};
For[nd = 0, nd ≤ NMAX,
  iplist = IntegerPartitions[nd, ADOWIDTH];
  For[i = 1, i ≤ Length[iplist],
   AppendTo[ADOTuple, Permutations[PadRight[iplist[[i]], ADOWIDTH]]];
   i++];
  nd++];
ADOTuple = Partition[Flatten[ADOTuple], ADOWIDTH];
```

In[103]:=
```
ADOTupleDict =
  Table[FromCharacterCode[ADOTuple[[i]]], {i, 1, Length[ADOTuple]}];
assoc = Association[Table[ADOTupleDict[[i]] → i, {i, 1, Length[ADOTuple]}]];
```

To implement the PLUS/MINUS actions on index tuples, we define the DD possibilities to add/subtract 1 from the tuple elements.

In[105]:=
```
null = Table[0, {i, 1, ADOWIDTH}];
poperator = Permutations[Table[If[i == 1, 1, 0], {i, 1, ADOWIDTH}]];
moperator = Permutations[Table[If[i == 1, -1, 0], {i, 1, ADOWIDTH}]];
```

For each tuple we now perform the PLUS operation and look up in the association map to which uid the PLUS operator leads.
Note that only index tuples with a depth < NMAX do have a PLUS link. Non-existing links are pointing to uid=-1.

In[108]:=
```
PlusIndex =.;
PlusIndex = Table[-1, {i, 1, ADOWIDTH * Length[ADOTuple]}];
For[uid = 0, uid < Length[ADOTuple],
  For[j = 0, j < ADOWIDTH,
   tc = FromCharacterCode[ADOTuple[[uid + 1]] + poperator[[j + 1]]];
   PlusIndex[[ADOWIDTH * uid + j + 1]] = Lookup[assoc, tc, -1];
   j++];
  uid++];
PlusIndex = Partition[PlusIndex, ADOWIDTH];
```

For each tuple we now perform the MINUS operation and look up in the association map to which uid the MINUS operator leads.
Note that only tuple elements >=1 do have a MINUS link. Non-existing links are pointing to uid=-1.

In[112]:=
```
MinusIndex =.;
MinusIndex = Table[-1, {i, 1, ADOWIDTH * Length[ADOTuple]}];
For[uid = 0, uid < Length[ADOTuple],
  For[j = 0, j < ADOWIDTH,
   testADOTuple = ADOTuple[[uid + 1]] + moperator[[j + 1]];
   If[MemberQ[testADOTuple, -1],
    MinusIndex[[ADOWIDTH * uid + j + 1]] = -1;
    ,
    tc = FromCharacterCode[testADOTuple];
    MinusIndex[[ADOWIDTH * uid + j + 1]] = Lookup[assoc, tc, -1];
   ];
   j++];
  uid++];
MinusIndex = Partition[MinusIndex, ADOWIDTH];
```

In[116]:=
```
Print["Total number of ADOs: ", Length[ADOTupleDict]]
```
```
Total number of ADOs: 15
```

Setup of the operators

In[117]:=
```
νbk[b_, k_, ħ_, ν_, λ_, β_, ν1_, Ω_] := If[k == 0, νNUMBATHS[ν, Ω][[b]], 2 π k / (β ħ)]
```

In[118]:=
```
cbk[b_, k_, ħ_, ν_, λ_, β_, ν1_, Ω_] :=
 If[k == 0, νNUMBATHS[ν, Ω][[b]] λNUMBATHS[λ][[b]] Cot[β ħ νNUMBATHS[ν, Ω][[b]] / 2],
  (4 λNUMBATHS[λ][[b]] νNUMBATHS[ν, Ω][[b]]) / (β ħ) (2 π k / (β ħ)) / ((2 π k / (β ħ))^2 - (νNUMBATHS[ν, Ω][[b]])^2)]
```

In[119]:=
```
bktt[b_, k_] := (b - 1) * MATSUBARAS + k + 1
```

In[120]:=
```
kkOp[k_] := DiagonalMatrix[Table[If[i == k, 1, 0], {i, 1, SITES}]]
```

In[121]:=
```
ϕk[k_, A_] := i (kkOp[k].A - A.kkOp[k])
Vcross[k_, A_] := kkOp[k].A - A.kkOp[k]
Vcirc[k_, A_] := kkOp[k].A + A.kkOp[k]
```

In[124]:=
```
ϴkMA[b_, s_, k_, A_, ħ_, ν_, λ_, β_, ν1_, Ω_] :=
 If[k == 0, (i λ ν Cot[(β ν ħ)/2]) / ħ Vcross[bath2sites[[b]][[s]], A] +
  Vcirc[bath2sites[[b]][[s]], A] ν λ / ħ ,
  4 i ν λ νbk[b, k, ħ, ν, λ, β, ν1, Ω] / (-β ν^2 ħ^2 + β ħ^2 νbk[b, k, ħ, ν, λ, β, ν1, Ω]^2) Vcross[bath2sites[[b]][[s]], A]]
```

The final HEOM equation (in similar notation to [R3])

In[125]:=
```
dσdtMASC[HM_, σM_, uid_, ħ_, ν_, λ_, β_, ν1_, Ω_] :=
 (-(i/ħ) (HM.σM[[uid]] - σM[[uid]].HM) - σM[[uid]]
   Sum[Sum[ADOTuple[[uid]][[bktt[b, k]]] νbk[b, k, ħ, ν, λ, β, ν1, Ω] -
     Sum[(2 λNUMBATHS[λ][[b]]/(β ħ^2 νNUMBATHS[ν, Ω][[b]]) -
       Sum[cbk[b, k, ħ, ν, λ, β, ν1, Ω]/(ħ νbk[b, k, ħ, ν, λ, β, ν1, Ω]),
        {k, 0, MATSUBARAS-1}]) Vcross[bath2sites[[b]][[s]], Vcross[bath2sites[[b]][[s]], σM[[uid]]],
     {s, 1, Length[bath2sites[[b]]]}], {b, 1, NUMBATHS}] +
   Sum[Sum[Sum[Sqrt[(ADOTuple[[uid]][[bktt[b, k]]] + 1)
       Abs[cbk[b, k, ħ, ν, λ, β, ν1, Ω]/ħ]]
      ϕk[bath2sites[[b]][[s]], σM[[PlusIndex[[uid]][[bktt[b, k]]]]]],
     {k, 0, MATSUBARAS-1}], {s, 1, Length[bath2sites[[b]]]}], {b, 1, NUMBATHS}] +
   Sum[Sum[Sum[Sqrt[ADOTuple[[uid]][[bktt[b, k]]]/
       Abs[cbk[b, k, ħ, ν, λ, β, ν1, Ω]/ħ]]
      θkMA[b, s, k, σM[[MinusIndex[[uid]][[bktt[b, k]]]]], ħ,
       νNUMBATHS[ν, Ω][[b]], λNUMBATHS[λ][[b]], β, ν1, Ω]),
     {k, 0, MATSUBARAS-1}], {s, 1, Length[bath2sites[[b]]]}], {b, 1, NUMBATHS}])
```

Running the HEOM algorithm within *Mathematica*

Now use the matrices and include as last element a zero matrix, this ensures that whenever σM[-1] is called, it does not contribute, since the array index -1 maps to the last element (=the zeroed matrix).

In[126]:=
```
nM = DiagonalMatrix[Table[0, {i, 1, SITES}]]
σM0 = Table[nM, {i, 1, Length[ADOTuple]}];
```

Out[126]=
```
{{0, 0}, {0, 0}}
```

Set the initial density matrix

In[128]:=
```
rho0 = DiagonalMatrix[Table[If[i == 1, 1, 0], {i, 1, SITES}]]
```

Out[128]=
```
{{1, 0}, {0, 0}}
```

In[129]:=
```
rho0 // MatrixForm
```

Out[129]//MatrixForm=

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Initialize one member of the hierarchy with the initial density matrix, append the zero matrix at the end for the -1 links

In[130]:=
```
σM0[[1]] = rho0;
AppendTo[σM0, nM];
```

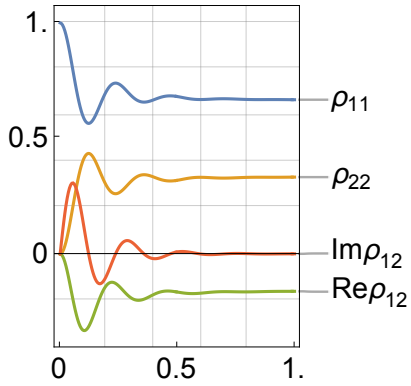
Taylor integration

In[132]:=
```
m = 200; (* number of steps *)
YTA4 = T = Table[0, {m + 1}];
T[[1]] = 0;
YTA4[[1]] = σM0;
Timing[
 For[j = 1, j ≤ m, j++,
  k1 = SetPrecision[
    hSI Table[dσdtMASC[HMSI, YTA4[[j]],       i, ħSI, νSI, λSI, βSI, ν1SI, ΩSI],
      {i, 1, Length[ADOTuple]}], PRECISION];
  AppendTo[k1, nM];
  YTA4[[j + 1]] = SetPrecision[YTA4[[j]] + k1 / 1!, PRECISION];
  k2 = hSI Table[dσdtMASC[HMSI, k1, i, ħSI, νSI, λSI, βSI, ν1SI, ΩSI],
     {i, 1, Length[ADOTuple]}];
  AppendTo[k2, nM];
  YTA4[[j + 1]] = SetPrecision[YTA4[[j + 1]] + k2 / 2!, PRECISION];
  k1 = hSI Table[dσdtMASC[HMSI, k2, i, ħSI, νSI, λSI, βSI, ν1SI, ΩSI],
     {i, 1, Length[ADOTuple]}];
  AppendTo[k1, nM];
  YTA4[[j + 1]] = SetPrecision[YTA4[[j + 1]] + k1 / 3!, PRECISION];
  k2 = hSI Table[dσdtMASC[HMSI, k1, i, ħSI, νSI, λSI, βSI, ν1SI, ΩSI],
     {i, 1, Length[ADOTuple]}];
  AppendTo[k2, nM];
  YTA4[[j + 1]] = SetPrecision[YTA4[[j + 1]] + k2 / 4!, PRECISION];
  T[[j + 1]] = hSI * j;
 ]]
```

Out[136]=
```
{7.79, Null}
```

In[137]:=
```
p = ListPlot[{
    Transpose[{T * 10^12, Re[YTA4[[All, 1, 1, 1]]]}],
    Transpose[{T * 10^12, Re[YTA4[[All, 1, 2, 2]]]}],
    Transpose[{T * 10^12, Re[YTA4[[All, 1, 1, 2]]]}],
    Transpose[{T * 10^12, Im[YTA4[[All, 1, 1, 2]]]}]
  }, PlotLabels → {Text["ρ11"], Text["ρ22"], Text["Reρ12"], Text["Imρ12"]},
  Joined → True, PlotRange → All, ImageSize → 220,
  Frame → True, AspectRatio → 1.4, LabelStyle → Directive[Black, 15],
  FrameTicks → {{{0, 0.5, 1.0}, None}, {{0, 0.5, 1.0}, None}},
  GridLines → Automatic]
```

Out[137]=



In[138]:=
```
(*Export["fig_oqsd.png",p,ImageResolution→200]*)
```