September 18, 2024

Contents

0.1 Bloch-Redfield and Redfield Failing for degenerate Hamiltonians

In The SYK model:

In this section we consider the SYK model whose Hamiltonian is given by (Ryu paper, introduction)

$$H = \sum_{i<j<k<l=1}^{N} J_{i,j,k,l}\psi_i\psi_j\psi_k\psi_l \tag{1}$$

Where $J_{i,j,k,l}$ is drawn randomly from a Gaussian ensemble with mean $\mu = 0$ and variance $\sigma = \sqrt{3!}\frac{J}{N^{3/2}}$ where J is a constant with dimension of mass. And the ψ_i denote the operators of the majorana fermions which are representations of the clifford algebra. They satisfy

$$\{\psi_i, \psi_j\} = \delta_{i,j} \tag{2}$$

For convenience people usually just consider the even case and one dimensional majorana fermions (appendix A). We introduce the new basis

$$c_i = \frac{1}{\sqrt{2}}(\psi_{2i} - i\psi_{2i+1}) \tag{3}$$

$$c_i^\dagger = \frac{1}{\sqrt{2}}(\psi_{2i} + i\psi_{2i+1}) \tag{4}$$

These satisfy

$$\{C_i, C_j^\dagger\} = \delta_{i,j} \tag{5}$$

$$\{C_i^\dagger, C_j^\dagger\} = 0 \tag{6}$$

To construct this basis we consider picking a vacuum annhilated by all modes such that

$$(C_1^\dagger)^{n_1}\dots(C_k^\dagger)^{n_k}0\dots0 = 0 \tag{7}$$

There are $2^{N/2} = 2^K$ such states. This is the only irreducible representation of (2) , upn top unitary equailvalence, the representation is given by 2^k matriceswhich can be found by the recursion relaton

$$\psi_i^K = \psi_i^{k-1} \otimes \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{for} \quad i = 1, 2, \dots, N-2 \tag{8}$$

$$\psi_{N-1}^K = \frac{1}{\sqrt{2}}1_{2^{K-1}} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{9}$$

$$\psi_N^K = \frac{1}{\sqrt{2}}1_{2^{K-1}} \otimes \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{10}$$

The superscript K is omitted in the Hamiltonian for convenience. Though not a great example, Let us use $N = 2$. To illustrate how solving by Bloch-Redfield may fail

The Hamiltonian in this example is then given by

```
H=20*H #energy rescaling
H
```

```
Quantum object: dims=[[2, 2], [2, 2]], shape=(4, 4), type='oper', isherm=True
Qobj data =
[[ -2.18176916  0.          0.          0.        ]
 [ 0.          2.18176916  0.          0.        ]
 [ 0.          0.          2.18176916  0.        ]
 [ 0.          0.          0.         -2.18176916]]
```

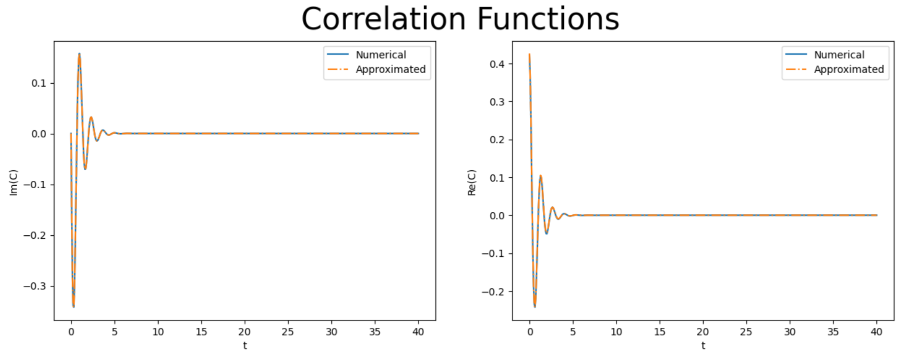
While the coupling operator to the bath is simply
$Q = \sum_i a_i \psi_i$
Where each a_i is a real number

```
Quantum object: dims=[[2, 2], [2, 2]], shape=(4, 4), type='oper', isherm=True
Qobj data =
[[ 0.        +0.j          0.53033009 -0.70710678j  0.44194174 -0.47140452j
   0.        +0.j        ]
 [ 0.53033009+0.70710678j  0.        +0.j          0.        +0.j
  -0.44194174+0.47140452j]
 [ 0.44194174+0.47140452j  0.        +0.j          0.        +0.j
   0.53033009 -0.70710678j]
 [ 0.        +0.j         -0.44194174 -0.47140452j  0.53033009+0.70710678j
   0.        +0.j        ]]
```

We consider the initial state to be

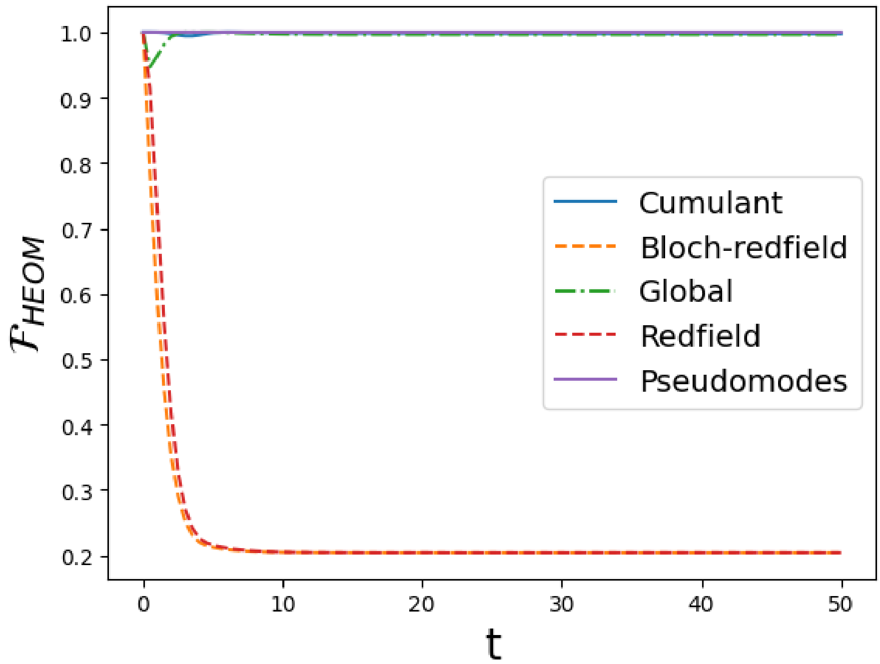
And consider an underdamped spectral density at zero temperature with γ =4.984282088163084 , λ =3.1478880316804854 ,ω_0 =10.217778280734322. After fitting the correlation function one obtains

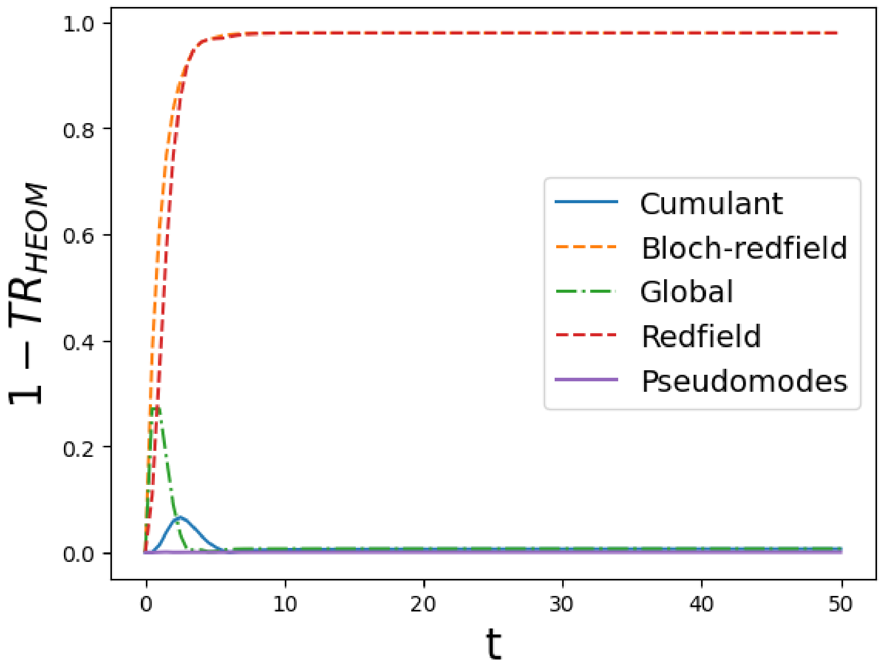
```
/home/mcditoos/github/qutip_gsoc_app/qutip/solver/heom/bofin_baths.py:925: RuntimeWarning: inv
  * (1 / np.tanh(beta * (Om + 1.0j * Gamma) / 2)),
/home/mcditoos/github/qutip_gsoc_app/qutip/solver/heom/bofin_baths.py:927: RuntimeWarning: inv
  * (1 / np.tanh(beta * (Om - 1.0j * Gamma) / 2)),

Fit correlation class instance:

Result of fitting The Real Part Of                                        |Result of fitting The
 the Correlation Function with 2 terms:                                   | Of the Correlation F
                                                                          |
 Parameters|     a      |     b      |     c                              | Parameters|     a
 1         | 5.15e -01 | -1.19e+00 |4.65e+00                              | 1          | -5.00e
 2         | -8.86e -02 | -4.25e+00 |7.19e -23                              |
                                                                          |A  normalized RMSE of
A  normalized RMSE of  7.60e -06 was obtained for the The Real Part Of    | Of the Correlation
 the Correlation Function                                                 |
 The current fit took  0.391309 seconds                                   | The current fit took
```



```
============================
Solving HEOM
============================
10.1%. Run time:  10.51s. Est. time left: 00:00:01:33
20.2%. Run time:  17.08s. Est. time left: 00:00:01:07
30.3%. Run time:  23.21s. Est. time left: 00:00:00:53
40.4%. Run time:  27.80s. Est. time left: 00:00:00:41
50.5%. Run time:  32.72s. Est. time left: 00:00:00:32
60.6%. Run time:  37.25s. Est. time left: 00:00:00:24
70.7%. Run time:  41.96s. Est. time left: 00:00:00:17
80.8%. Run time:  48.52s. Est. time left: 00:00:00:11
90.9%. Run time:  54.29s. Est. time left: 00:00:00:05
100.0%. Run time:  59.50s. Est. time left: 00:00:00:00
Total run time:  59.50s
============================
HEOM Done
============================
============================
Solving Cumulant
============================

Calculating Integrals ...: 100%|| 4/4 [00:01<00:00,  2.31it/s]
Calculating time independent matrices...: 100%|| 4/4 [00:00<00:00, 1734.44it
Calculating time dependent generators: 100%|| 4/4 [00:00<00:00, 1439.73it/s]
Computing Exponential of Generators . . . .: 100%|| 100/100 [00:00<00:00, 11

============================
Cumulant Done
============================
============================
Solving Redfield
============================
Started interpolation
============================
Redfield Done
============================
============================
Solving Bloch -Redfield
============================
============================
Bloch -Redfield Done
============================
============================
Solving Global
============================
============================
Global Done
============================
============================
Solving Pseudomodes
============================
Fit correlation class instance:

Result of fitting The Real Part Of                                      |Result of fitting The
```

```
 the Correlation Function with 2 terms:                                   | Of the Correlation F
                                                                          |
 Parameters|     a      |     b      |     c                              | Parameters|     a
 1          | -5.55e -02 | -6.55e+00 |6.62e -12                              | 1           | 0.00
 2          | -2.03e -02 | -1.36e+00 |7.21e -16                              |
                                                                          |A  normalized RMSE of
A  normalized RMSE of  1.79e -05 was obtained for the The Real Part Of    | Of the Correlation
 the Correlation Function                                                 |
 The current fit took  0.363063 seconds                                   | The current fit took

============================
Pseudomodes done
============================
```

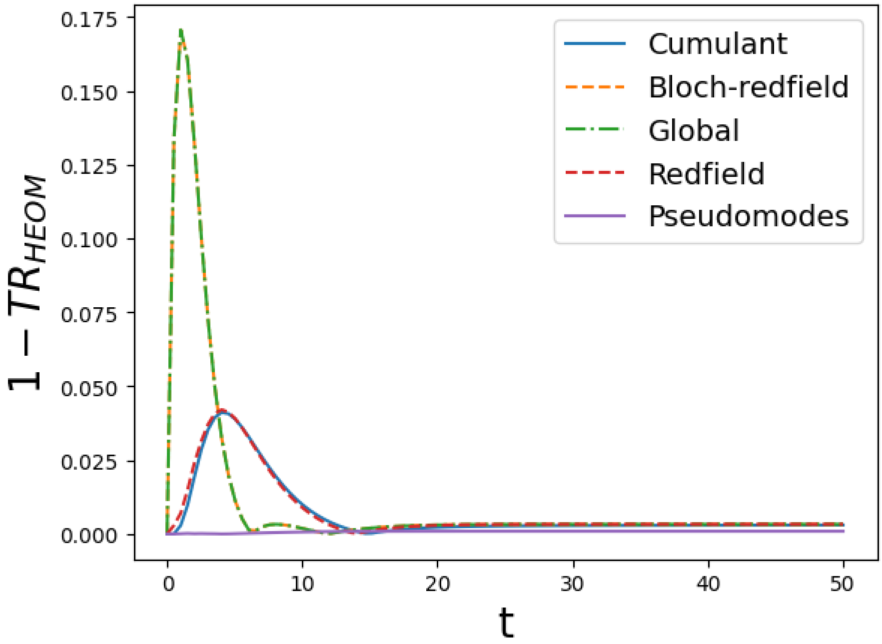


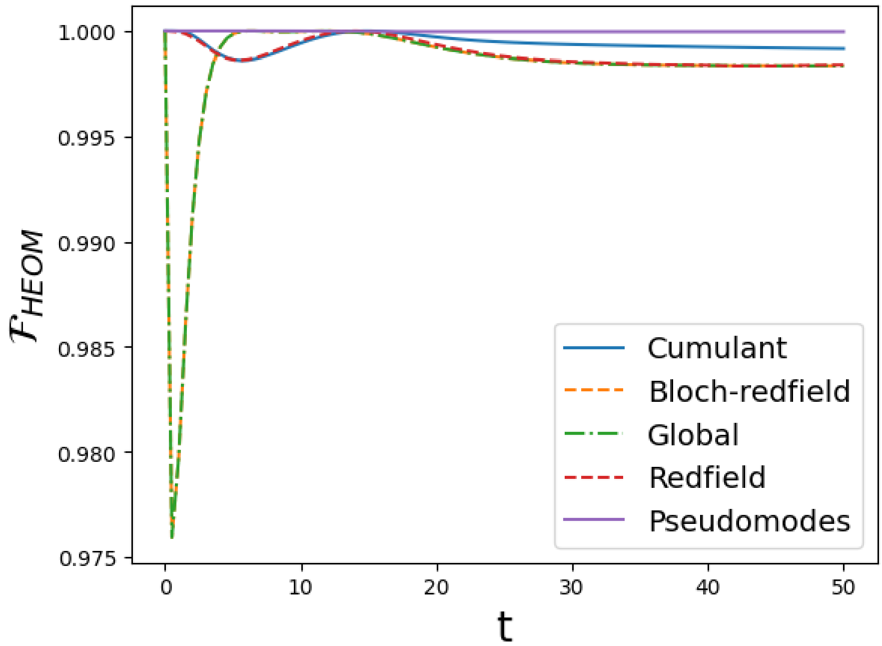


From what we see in both the trace distance and fidelity plots, the Bloch-Redfield approach does terribly when we consider this scenario (multiple implementations where checked). Notice that this issue seems to be about the coupling operator, rather than the Hamiltonian. Consider a different coupling operator just the majorana fermion denoted by the index 0 coupled to the environment

```
Quantum object: dims=[[2, 2], [2, 2]], shape=(4, 4), type='oper', isherm=True
Qobj data =
[[0.+0.j         0. -0.70710678j 0.+0.j         0.+0.j        ]
 [0.+0.70710678j 0.+0.j         0.+0.j         0.+0.j        ]
 [0.+0.j         0.+0.j         0.+0.j         0. -0.70710678j]
 [0.+0.j         0.+0.j         0.+0.70710678j 0.+0.j        ]]

Fit correlation class instance:

Result of fitting The Real Part Of                                       |Result of fitting The
 the Correlation Function with 2 terms:                                  | Of the Correlation F
                                                                         |
 Parameters|    a     |    b     |    c                                  | Parameters|    a
 1         | 5.15e -01 | -1.19e+00 |4.65e+00                             | 1         | -5.00e
 2         | -8.86e -02 | -4.25e+00 |7.19e -23                             |
                                                                         |A  normalized RMSE of
A  normalized RMSE of  7.60e -06 was obtained for the The Real Part Of   | Of the Correlation
 the Correlation Function                                                |
 The current fit took  0.351794 seconds                                  | The current fit took

============================
Solving HEOM
============================
10.1%. Run time:   3.59s. Est. time left: 00:00:00:31
20.2%. Run time:   7.10s. Est. time left: 00:00:00:28
30.3%. Run time:  10.64s. Est. time left: 00:00:00:24
40.4%. Run time:  14.14s. Est. time left: 00:00:00:20
50.5%. Run time:  17.54s. Est. time left: 00:00:00:17
60.6%. Run time:  20.99s. Est. time left: 00:00:00:13
70.7%. Run time:  24.80s. Est. time left: 00:00:00:10
80.8%. Run time:  29.79s. Est. time left: 00:00:00:07
90.9%. Run time:  34.83s. Est. time left: 00:00:00:03
100.0%. Run time:  38.71s. Est. time left: 00:00:00:00
Total run time:  38.71s
============================
HEOM Done
============================
============================
Solving Cumulant
============================

Calculating Integrals ...: 100%|| 4/4 [00:01<00:00,  2.32it/s]
Calculating time independent matrices...: 100%|| 4/4 [00:00<00:00, 1350.39it
Calculating time dependent generators: 100%|| 4/4 [00:00<00:00, 950.60it/s]
Computing Exponential of Generators . . . .: 100%|| 100/100 [00:00<00:00, 12

============================
Cumulant Done
============================
============================
Solving Redfield
============================
```

```
Started interpolation
============================
Redfield Done
============================
============================
Solving Bloch -Redfield
============================
============================
Bloch -Redfield Done
============================
============================
Solving Global
============================
============================
Global Done
============================
============================
Solving Pseudomodes
============================
Fit correlation class instance:

Result of fitting The Real Part Of                                          |Result of fitting The
 the Correlation Function with 2 terms:                                     | Of the Correlation F
                                                                            |
 Parameters|    a     |    b     |    c                                     | Parameters|    a
 1         | -5.55e -02 | -6.55e+00 |6.62e -12                                 | 1         | 0.00
 2         | -2.03e -02 | -1.36e+00 |7.21e -16                                 |
                                                                            |A  normalized RMSE of
A  normalized RMSE of  1.79e -05 was obtained for the The Real Part Of       | Of the Correlation
 the Correlation Function                                                   |
 The current fit took  0.457797 seconds                                     | The current fit took

============================
Pseudomodes done
============================
```





Tip
Whenever pseudomodes don't work it mainly can be fix with a more delicate fit (usually increasing levels is not worthwile pursuing in this regime)

We can observe the same behaviour in the ising model when the N is large I should run this example for longer times, so that I can make sure is analogous to the previous case and not just being better in the transient regime (which would still be good) but along the lines of what was claimed in This paper. Than when the cutoff frequency is large bloch redfield does not capture time dependent redfield (I should Also add redfield here and see if the cumulant can do better)

```
from hamiltonians import ising
H,sx,sy,sz=ising(N=3,g=1,Jx=5)
Q=sx[ -1]+ 1.1*sy[ -1]+0.9*sz[ -1]

E01=H.eigenenergies()[2] -H.eigenenergies()[0]#it is mostly degenerate, this does not help muc
```

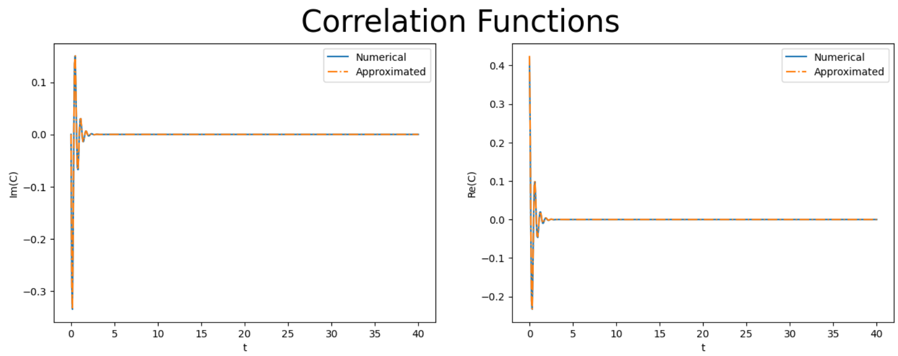
```
w0=1.1 *E01#since I have no g parameter then It doesn't scale uniformingly as ising
gamma=w0/2.05
Gamma=gamma/2
Omega=np.sqrt(w0**2 -Gamma**2)
lam=np.sqrt(Omega)
```

And consider an underdamped spectral density at zero temperature with γ =4.984282088163084 , λ =3.1478880316804854 ,ω_0 =10.217778280734322. After fitting the correlation function one obtains

```
bath = heom.UnderDampedBath(
        Q=Q,
        lam=lam, gamma=gamma, w0=w0, T=0, Nk=5) # fix runtime warning
cfiitter2 = heom.CorrelationFitter(
    Q, 0, tfit, bath.correlation_function)
bath1, fit2info = cfiitter2.get_fit(Ni=1, Nr=2)
# notice one mode is also a pretty good approximation
print(fit2info['summary'])
```

```
Fit correlation class instance:

Result of fitting The Real Part Of                                       |Result of fitting The
 the Correlation Function with 2 terms:                                  | Of the Correlation F
                                                                         |
 Parameters|    a     |    b     |    c                                  | Parameters|    a
 1         | -9.13e -02 | -9.27e+00 |4.44e -09                               | 1         | -5.0
 2         | 5.16e -01 | -2.55e+00 |9.89e+00                               |
                                                                         |A  normalized RMSE of
A  normalized RMSE of  5.27e -06 was obtained for the The Real Part Of   | Of the Correlation
 the Correlation Function                                                |
 The current fit took  0.498846 seconds                                  | The current fit took
```

```
times2 = np.linspace(0,40,500)
cvis = bath.correlation_function(times2)

# using the variable axs for multiple Axes
fig, axs = plt.subplots(1, 2,figsize=(15,5))
axs[0].plot(times2, np.imag(cvis),label="Numerical")
axs[0].plot(times2, np.imag(bath1.correlation_function_approx(times2)), " -.",label="Approxima
axs[0].set_xlabel("t")
axs[0].set_ylabel("Im(C)")
axs[0].legend()
axs[1].plot(times2, np.real(cvis),label="Numerical")
axs[1].plot(times2, np.real(bath1.correlation_function_approx(times2)), " -.",label="Approxima
axs[1].set_xlabel("t")
axs[1].set_ylabel("Re(C)")
axs[1].legend()
fig.suptitle('Correlation Functions', fontsize=30)
plt.show()
```




```
N=3
state_list = [basis(2, 1)] + [ -1j*basis(2, 0)] * (N - 1) # change the initial state to be awa
state_list2 = [basis(2, 1)] + [basis(2, 0)] * (N - 1) # change the initial state to be away fr
state_list.reverse()
psi0 = (tensor(state_list)+tensor(state_list2))/np.sqrt(2)
rho0=psi0*psi0.dag()
```

```
============================
Solving HEOM
============================
10.2%. Run time:   2.38s. Est. time left: 00:00:00:20
20.4%. Run time:   4.32s. Est. time left: 00:00:00:16
30.6%. Run time:   6.29s. Est. time left: 00:00:00:14
40.8%. Run time:   8.20s. Est. time left: 00:00:00:11
51.0%. Run time:  10.13s. Est. time left: 00:00:00:09
61.2%. Run time:  12.08s. Est. time left: 00:00:00:07
71.4%. Run time:  14.06s. Est. time left: 00:00:00:05
81.6%. Run time:  15.93s. Est. time left: 00:00:00:03
91.8%. Run time:  17.79s. Est. time left: 00:00:00:01
100.0%. Run time:  19.30s. Est. time left: 00:00:00:00
Total run time:  19.31s
============================
HEOM Done
============================
============================
Solving Cumulant
============================

Calculating Integrals ...: 100%|| 361/361 [02:42<00:00,  2.22it/s]
Calculating time independent matrices...: 100%|| 361/361 [00:01<00:00, 350.2
Calculating time dependent generators: 100%|| 361/361 [00:00<00:00, 409.76it
Computing Exponential of Generators . . . .: 100%|| 50/50 [00:00<00:00, 131.

============================
Cumulant Done
============================
============================
Solving Redfield
============================
Started interpolation

F0816 22:10:58.745429   48992 pjrt_stream_executor_client.cc:452] Check failed: copy_stream ->
```

0.1.1 The schwinger model

Same thing happening here, where I expected Bloch redfield to be better

But Again I should run this example to longer times to make sure is not a terribly innacurate transient effect but rather the equation breaking down. Also should add redfield to the mix

```
#example_schwinger= qload("results_cluster/N=4_schwinger_1.9679896712654306_nocheating_m_0.0_t

#import numpy as np
#import matplotlib.pyplot as plt
#from qutip import fidelity,tracedist

#plot_fidelities2(example_schwinger)

#trd2(example_schwinger)
```

0.2 Redfield Issue check "Analytically"

Since there seems to be an issue with the Bloch-Redfield Solver in qutip and my not so good implementation of the time dependent redfield, here's a quick sympy check to make sure it's not the solvers. By solving the equation for the SYK(2) model symbolically. The Hamiltonian is given by

Here I define the coupling operator q that make things break for Bloch-Redfield on the SYK model

```
Eq(S("q"), Q, evaluate=False)
```

```
Eq(q, Matrix([
[         0,  b0 - I*b1, b2 - I*b3,          0],
[b0 + I*b1,          0,         0, -b2 + I*b3],
[b2 + I*b3,          0,         0,  b0 - I*b1],
[         0, -b2 - I*b3, b0 + I*b1,          0]]))
```

I then get the eigenvalues and eigenvectors to obtain the jump operators (this steps are hidden in the pdf)

Jump operator checks

The jump operators must satisfy

$$[H, A(\omega)] = -\omega A(\omega) \tag{11}$$

$$[H, A^{\dagger}(\omega)A(\omega)] = 0 \tag{12}$$

$$\sum_{w} A(\omega) = A \tag{13}$$

There's a check below hidden in the pdf

Constucting the Differential equations

We now construct the differential equations from the GKLS form of the bloch Redfield generator

$$\dot{\rho_S^I}(t)t = \sum_{\omega,\omega',\alpha,\beta} \gamma_{\beta,\alpha}(\omega,\omega') \left(S_\alpha(\omega')\rho_S^I(t)S_\beta^\dagger(\omega) - \frac{\{S_\beta^\dagger(\omega)S_\alpha(\omega'), \rho_S^I(t)\}}{2} \right) + i \sum_{\omega,\omega',\alpha,\beta} S_{\beta,\alpha}(\omega,\omega') \left[\rho_S^I(t), S_\beta^\dagger(\omega)S_\alpha(\omega')\right]$$

I solve in the interaction picture generally, I did the same in my numerics so it should not be an issue (I rotate in the end). By neglecting Lambshift as in the numerics

$$\dot{\rho_S^I}(t)t = \sum_{\omega,\omega',\alpha,\beta} \gamma_{\beta,\alpha}(\omega,\omega') \left(S_\alpha(\omega')\rho_S^I(t)S_\beta^\dagger(\omega) - \frac{\{S_\beta^\dagger(\omega)S_\alpha(\omega'), \rho_S^I(t)\}}{2} \right) \tag{14}$$

As the sum goes on (ω, ω') pairs I construct all combinations

```
ws = list(jumps.keys())
combinations = list(itertools.product(ws, ws))
combinations
```

```
[(2*a, 2*a), (2*a, -2*a), ( -2*a, 2*a), ( -2*a, -2*a)]
```

I get the GKLS form of each of those combinations, as a dictionary

Then I construct the generator by multiplying the appropiate coefficient to each of the GKLS from matrices. Now for the coefficients we have

$$\Gamma_{\alpha,\beta}(\omega,t)=\int_0^t dse^{i\omega s}\langle B_\alpha^\dagger(t)B_\beta^\dagger(t-s)\rangle_B \tag{15}$$

For convenience we also define

$$\Gamma_{\alpha,\beta}(\omega,\omega',t)=e^{i(\omega'-\omega)t}\int_0^t dse^{i\omega s}\langle B_\alpha^\dagger(t)B_\beta(t-s)\rangle_B=e^{i(\omega'-\omega)t}\Gamma_{\alpha,\beta}(\omega,t) \tag{16}$$

Since I mainly care about bloch-redfield I make $t\to\infty$ (in the integral) so

$$\Gamma_{\alpha,\beta}(\omega,\omega',t)=e^{i(\omega'-\omega)t}\int_0^\infty dse^{i\omega s}\langle B(s)B(0)\rangle_B=e^{i(\omega'-\omega)t}\Gamma_{\alpha,\beta}(\omega) \tag{17}$$

Where $\Gamma_{\alpha,\beta}(\omega)$ is the power spectrum

```
Eq(S('Gamma(w)'), power_spectrum(S('w')), evaluate=False)
```

```
Eq(Gamma(w), 2*gamma*lambda**2*w*(1 + 1/(exp(w/T) - 1))/(gamma**2*w**2 + ( -w**2 + w0**2)**2))
```

Next we simply vectorize the density matrix and construct the system of ODES

```
for i in eqs:
    display(i)
```

```
Eq(Derivative(rho_1(t), t), 4*a*gamma*lambda**2*((b0 + I*b1)*((b0 - I*b1)*rho_6(t) + (b2 - I*b

Eq(Derivative(rho_2(t), t), 2.0*a*gamma*lambda**2*( -b0**2 - b1**2 - b2**2 - b3**2)*rho_2(t)/(

Eq(Derivative(rho_3(t), t), 2.0*a*gamma*lambda**2*( -b0**2 - b1**2 - b2**2 - b3**2)*rho_3(t)/(

Eq(Derivative(rho_4(t), t), 4*a*gamma*lambda**2*((b0 - I*b1)*((b0 - I*b1)*rho_7(t) + (b2 - I*b

Eq(Derivative(rho_5(t), t), a*gamma*lambda**2*(4*(b0 + I*b1)*((b0 + I*b1)*rho_2(t) - (b2 - I*b

Eq(Derivative(rho_6(t), t), 4.0*a*gamma*lambda**2*( -b0**2 - b1**2 - b2**2 - b3**2)*rho_6(t)/(

Eq(Derivative(rho_7(t), t), 4.0*a*gamma*lambda**2*( -b0**2 - b1**2 - b2**2 - b3**2)*rho_7(t)/(

Eq(Derivative(rho_8(t), t), a*gamma*lambda**2*(4*(b0 - I*b1)*((b0 + I*b1)*rho_3(t) - (b2 - I*b

Eq(Derivative(rho_9(t), t), a*gamma*lambda**2*(4*(b0 + I*b1)*((b0 - I*b1)*rho_14(t) + (b2 + I*

Eq(Derivative(rho_10(t), t), 4.0*a*gamma*lambda**2*( -b0**2 - b1**2 - b2**2 - b3**2)*rho_10(t)

Eq(Derivative(rho_11(t), t), 4.0*a*gamma*lambda**2*( -b0**2 - b1**2 - b2**2 - b3**2)*rho_11(t)

Eq(Derivative(rho_12(t), t), a*gamma*lambda**2*(4*(b0 - I*b1)*((b0 - I*b1)*rho_15(t) + (b2 + I

Eq(Derivative(rho_13(t), t), 4*a*gamma*lambda**2*((b0 + I*b1)*((b0 + I*b1)*rho_10(t) - (b2 + I

Eq(Derivative(rho_14(t), t), 2.0*a*gamma*lambda**2*( -b0**2 - b1**2 - b2**2 - b3**2)*rho_14(t)

Eq(Derivative(rho_15(t), t), 2.0*a*gamma*lambda**2*( -b0**2 - b1**2 - b2**2 - b3**2)*rho_15(t)

Eq(Derivative(rho_16(t), t), 4*a*gamma*lambda**2*((b0 - I*b1)*((b0 + I*b1)*rho_11(t) - (b2 + I
```

Let me make a few change of variables, and call the new variables c_k

```
j=0
for i in [c0,c1,c2,c3]:
    display(Eq(c[j],i))
    j+=1
```

```
Eq(c0, 2.0*a*gamma*lambda**2*( -b0**2 - b1**2 - b2**2 - b3**2)/(16.0*a**4 + 4.0*a**2*gamma**2

Eq(c1, gamma*lambda**2/(4*a**2*gamma**2 + (4*a**2 - w0**2)**2))

Eq(c2, b0 + I*b1)

Eq(c3, b2 + I*b3)
```

By substituting these into the differential equation we obtain

```
for i in eqs:
    display(i)
```

```
Eq(Derivative(rho_1(t), t), 4*a*c1*(c2*rho_10(t)*conjugate(c3) + c2*rho_6(t)*conjugate(c2) + c

Eq(Derivative(rho_2(t), t), c0*rho_2(t))

Eq(Derivative(rho_3(t), t), c0*rho_3(t))

Eq(Derivative(rho_4(t), t), 4*a*c1*( -rho_10(t)*conjugate(c3)**2 + rho_11(t)*conjugate(c2)*con

Eq(Derivative(rho_5(t), t), a*c1*(4*c2**2*rho_2(t) + 4*c2*c3*rho_3(t) - 4*c2*rho_14(t)*conjuga

Eq(Derivative(rho_6(t), t), 2*c0*rho_6(t))

Eq(Derivative(rho_7(t), t), 2*c0*rho_7(t))

Eq(Derivative(rho_8(t), t), a*c1*( -4*c2*rho_2(t)*conjugate(c3) + 4*c2*rho_3(t)*conjugate(c2)

Eq(Derivative(rho_9(t), t), a*c1*(4*c2*c3*rho_2(t) + 4*c2*rho_14(t)*conjugate(c2) + 4*c3**2*rh

Eq(Derivative(rho_10(t), t), 2*c0*rho_10(t))

Eq(Derivative(rho_11(t), t), 2*c0*rho_11(t))

Eq(Derivative(rho_12(t), t), a*c1*( -4*c3*rho_2(t)*conjugate(c3) + 4*c3*rho_3(t)*conjugate(c2)

Eq(Derivative(rho_13(t), t), 4*a*c1*(c2**2*rho_10(t) + c2*c3*rho_11(t) - c2*c3*rho_6(t) - c3**

Eq(Derivative(rho_14(t), t), c0*rho_14(t))

Eq(Derivative(rho_15(t), t), c0*rho_15(t))

Eq(Derivative(rho_16(t), t), 4*a*c1*( -c2*rho_10(t)*conjugate(c3) + c2*rho_11(t)*conjugate(c2)
```

With The number of symbols reduced the symbolic computation is feasible. However the default solver with initial conditions yields

```
sols[4]
```

```
Eq(rho_5(t), -(4.0*C3*a*c1*c2**2*Piecewise((8.0/(16.0*a*c1*c2*conjugate(c2) + 16.0*a*c1*c3*con
```

Unfortunately there's a bug in the sympy analytical solver when substituting the initial value conditions to obtain the constants It's not so bad because it is evidend that the weird term is zero. But I check it with manual substitutions anyway below

> **Warning**
> It's only evident if the solution of the equation is a valid density matrix

The initial state considered is
The solution to the system of equations is

```
for i in sols:
    display(i)
```

```
Eq(rho_1(t), 1.0*C1 + 1.0*C2*exp(2.0*c0*t))

Eq(rho_2(t), C3*exp(c0*t))

Eq(rho_3(t), C4*exp(c0*t))

Eq(rho_4(t), 1.0*C5 + 1.0*C6*exp(2.0*c0*t))

Eq(rho_5(t), 1.0*C9*exp( -t*(2.0*a*c1*c2*conjugate(c2) + 2.0*a*c1*c3*conjugate(c3))) + (4.0*C3

Eq(rho_6(t), (C10*c0*c3*conjugate(c3)/(a*c1*(2.0*c2**2*conjugate(c2)**2 + 4.0*c2*c3*conjugate(

Eq(rho_7(t), -(C10*c0*c2*conjugate(c3)/(a*c1*(2.0*c2**2*conjugate(c2)**2 + 4.0*c2*c3*conjugate

Eq(rho_8(t), 1.0*C12*exp( -t*(2.0*a*c1*c2*conjugate(c2) + 2.0*a*c1*c3*conjugate(c3))) - (4.0*C

Eq(rho_9(t), 1.0*C13*exp( -t*(2.0*a*c1*c2*conjugate(c2) + 2.0*a*c1*c3*conjugate(c3))) + (4.0*C

Eq(rho_10(t), -(C10*c0*c3*conjugate(c2)/(a*c1*(2.0*c2**2*conjugate(c2)**2 + 4.0*c2*c3*conjugat

Eq(rho_11(t), (C10*c0*c2*conjugate(c2)/(a*c1*(2.0*c2**2*conjugate(c2)**2 + 4.0*c2*c3*conjugate

Eq(rho_12(t), 1.0*C14*exp( -t*(2.0*a*c1*c2*conjugate(c2) + 2.0*a*c1*c3*conjugate(c3))) - (4.0*

Eq(rho_13(t), 1.0*C11*exp(2.0*c0*t) + 1.0*C15)

Eq(rho_14(t), C7*exp(c0*t))

Eq(rho_15(t), C8*exp(c0*t))

Eq(rho_16(t), 1.0*C10*exp(2.0*c0*t) + 1.0*C16)
```

Then by substituting this into the solution we obtain to find the constants we obtain
By reverting the change of variables. The analytical answer of the Bloch redfield equation is given by

```
for i in newsols:
    display(i)
```

```
Eq(rho_1(t), 0.5*(exp(4.0*a*gamma*lambda**2*t*(b0**2 + b1**2 + b2**2 + b3**2)/(16.0*a**4 + 4.0

Eq(rho_2(t), 0)

Eq(rho_3(t), 0)

Eq(rho_4(t), 0.5*I*((b0 - I*b1)**2 + (b2 - I*b3)**2)*(exp(4.0*a*gamma*lambda**2*t*(b0**2 + b1*

Eq(rho_5(t), 0)
```

```
Eq(rho_6(t), 0.5*exp( -4.0*a*gamma*lambda**2*t*(b0**2 + b1**2 + b2**2 + b3**2)/(16.0*a**4 + 4.

Eq(rho_7(t), 0.5*I*exp( -4.0*a*gamma*lambda**2*t*(b0**2 + b1**2 + b2**2 + b3**2)/(16.0*a**4 +

Eq(rho_8(t), 0)

Eq(rho_9(t), 0)

Eq(rho_10(t), -0.5*I*exp( -4.0*a*gamma*lambda**2*t*(b0**2 + b1**2 + b2**2 + b3**2)/(16.0*a**4

Eq(rho_11(t), 0.5*exp( -4.0*a*gamma*lambda**2*t*(b0**2 + b1**2 + b2**2 + b3**2)/(16.0*a**4 + 4

Eq(rho_12(t), 0)

Eq(rho_13(t), 0.5*I*(1 - exp(4.0*a*gamma*lambda**2*t*(b0**2 + b1**2 + b2**2 + b3**2)/(16.0*a**

Eq(rho_14(t), 0)

Eq(rho_15(t), 0)

Eq(rho_16(t), 0.5*(exp(4.0*a*gamma*lambda**2*t*(b0**2 + b1**2 + b2**2 + b3**2)/(16.0*a**4 + 4.
```

We can then substitute the numerical values for example for the case we explored above

```
Eq(S("rho(t)"),anss,evaluate=False)

Eq(rho(t), Matrix([
[                           0.55247613848278*(exp(0.94679965816826*t) - 1.0)*exp( -0.9467996581
[
[
[( -0.486572940196368 - 0.102435485836685*I)*(1.0 - exp(0.94679965816826*t))*exp( -0.946799658
```

Then we may evaluate for long times

```
Eq(S("rho(150)"),roundMatrix(ans.subs(num_values).subs(t,150).evalf(),18),evaluate=False)

Eq(rho(150), Matrix([
[                        0.55247613848278,   0,   0, 0.486572940196368 - 0.102435485836685*I],
[                                       0, 0.0,   0,                                        0],
[                                       0,   0, 0.0,                                        0],
[0.486572940196368 + 0.102435485836685*I,   0,   0,                         0.44752386151722]])
```

I believe I was careful enough to use the same convention used in the other equations (Pseudomodes, Cumulant and redfield) but currently reviewing the derivations to make sure there's no inconsistencies. The derivations in question are in https://master–gsuarezthesis.netlify.app/redfield . I do think it is now safe to assume that BR/Redfield breaksdown and that it is not a bug in the code, so maybe we can write a paper on redfield breaking down, cumulant/global being good once we figure out why it happens. Though it seems to be about the coupling and not the degeneracies

About Pictures
Technically the above matrix is not correct as it is in the interaction picture and not the Schrodinger picture. In this case it does not make a difference, however, let us do the rotation

```
U=exp(I*H*t)

Eq(S("U"),U,evaluate=False)
```

```
Eq(U, Matrix([
[exp( -I*a*t),          0,          0,           0],
[          0, exp(I*a*t),          0,           0],
[          0,          0, exp(I*a*t),           0],
[          0,          0,          0, exp( -I*a*t)]]))
```

```
Eq(S("U")*S("rho")*Dagger(S("U")),U*rho*Dagger(U),evaluate=False)
```

```
Eq(U*rho*Dagger(U), Matrix([
[                       rho_1,            rho_2*exp( -2*I*a*t),             rho_3*exp( -2*I*a
[exp(2*I*a*t)*conjugate(rho_2),                           rho_6,              conjugate(rho_10
[exp(2*I*a*t)*conjugate(rho_3),                          rho_10,                          rho_1
[             conjugate(rho_4), exp( -2*I*a*t)*conjugate(rho_8), exp( -2*I*a*t)*conjugate(rho_
```

```
rhoss=U*roundMatrix(ans.subs(num_values).subs(t,150).evalf(),18)*Dagger(U)
```

```
Eq(S("rho(150)"),rhoss,evaluate=False)
```

```
Eq(rho(150), Matrix([
[                      0.55247613848278, 0, 0, 0.486572940196368 - 0.102435485836685*I],
[                                     0, 0, 0,                                       0],
[                                     0, 0, 0,                                       0],
[0.486572940196368 + 0.102435485836685*I, 0, 0,                        0.44752386151722]]))
```

0.2.1 RC picture of the Hamiltonian

For the RC I simply follow https://arxiv.org/pdf/1511.05181

So If I didn't misunderstand it then

$$\Omega = w_0 \tag{18}$$

$$\lambda_{rc} = \sqrt{\frac{\pi}{2w_0}}\lambda \tag{19}$$

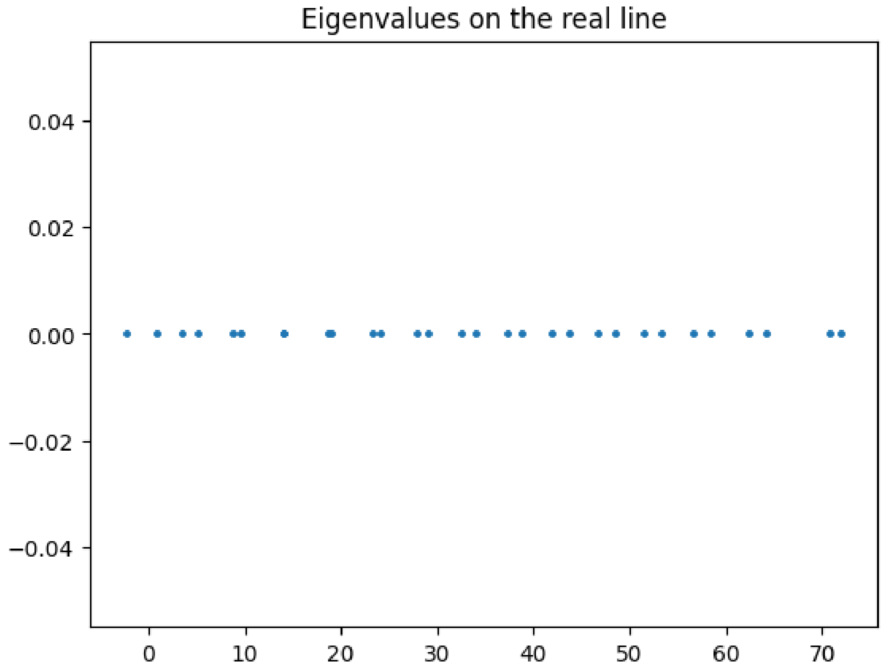
Not actually sure if π should be there, but does not seem to be relevant for the question we are asking

Not sure how many levels to take here, but let us guess 15 is enough

Then I construct the RC Hamiltonian

```
NHRC=HRC.subs(num_values).evalf()
```

```
plt.scatter(np.real(eigenvalues),np.round(np.imag(eigenvalues),10),s=5)
plt.title("Eigenvalues on the real line")
plt.show()
```



Propably I shopuld have done the partial trace so

```
qHRC.ptrace(0)
```

```
Quantum object: dims=[[4], [4]], shape=(4, 4), type='oper', dtype=Dense, isherm=True
Qobj data =
[[471.25867623   0.           0.           0.        ]
 [  0.         536.71867623   0.           0.        ]
 [  0.           0.         536.71867623   0.        ]
 [  0.           0.           0.         471.25867623]]
```

Still degenerate

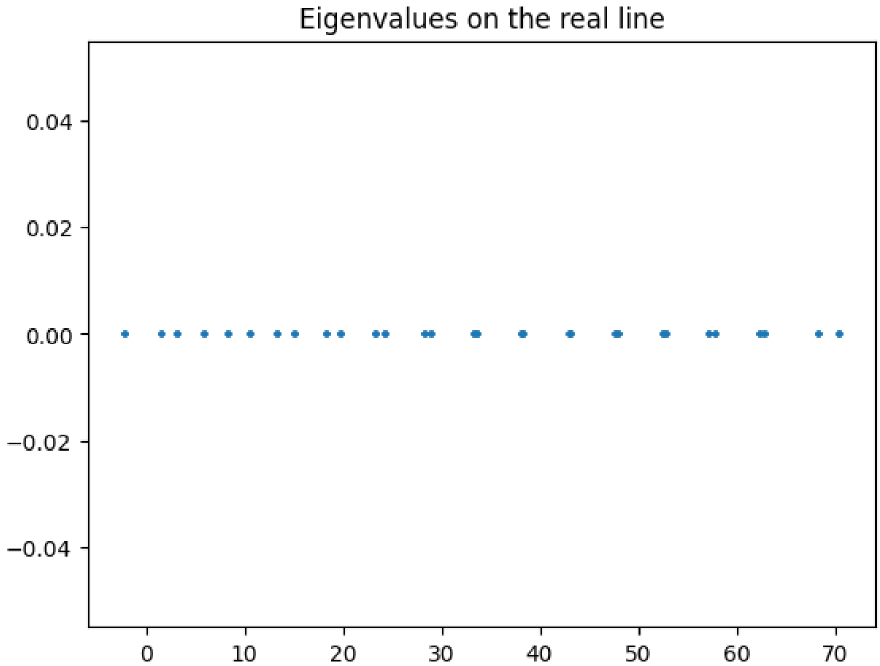
Let us try the other coupling which does not break bloch redfield

```
qHRC.ptrace(0)
```

```
Quantum object: dims=[[4], [4]], shape=(4, 4), type='oper', dtype=Dense, isherm=True
Qobj data =
[[471.25867623   0.           0.           0.        ]
 [  0.         536.71867623   0.           0.        ]
 [  0.           0.         536.71867623   0.        ]
 [  0.           0.           0.         471.25867623]]
```

```
eigenvalues, eigenvectors = np.linalg.eig(ans)
```

```
plt.scatter(np.real(eigenvalues), np.round(np.imag(eigenvalues), 10), s=5)
plt.title("Eigenvalues on the real line")
plt.show()
```



There doesn't seem to be much change in the Hamiltonian if any