August 20, 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 $\psi_i$ 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

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

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 $\gamma =4.984282088163084$ , $\lambda =3.1478880316804854$ ,$\omega_0 =10.217778280734322$. 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 the Correlation Function with 2 terms: | | | | | Result of fitting The 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 the Correlation Function | | | | | | Of the Correlation |
| The current fit took  0.391309 seconds | | | | | | The current fit took |

## Correlation Functions
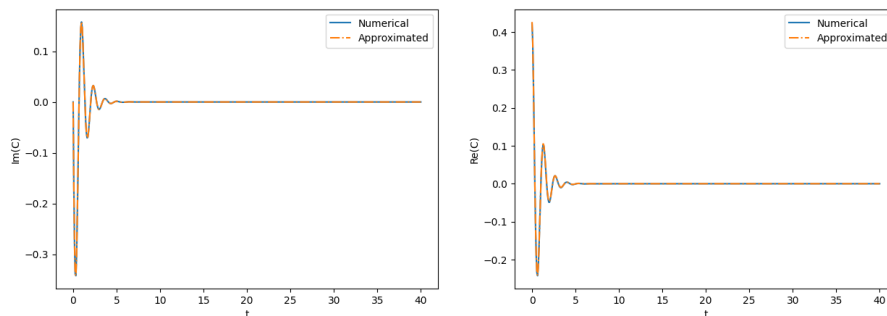
```python
def solve_dynamics(H,Q,bath,bath1,rho0,depth=7,levels=3,modes=2,times=times):
    print("===========================")
    print("Solving HEOM")
    print("===========================")
    solver = heom.HEOMSolver(H,
                             [bath1], max_depth=depth, options={"atol": 1e -14})
    result = solver.run(rho0, times)
    print("===========================")
    print("HEOM Done")
    print("===========================")
    print("===========================")
    print("Solving Cumulant")
    print("===========================")
    bath.bose=None
    cum = csolve(
        Hsys=H, t=times, baths=[bath],
        Qs=[Q],
        eps=1e -6, cython=False)

    result_cum = cum.evolution(rho0)

    result_cum = rotation(result_cum, H, times)
    print("===========================")
    print("Cumulant Done")
    print("===========================")
    print("===========================")
    print("Solving Redfield")
    print("===========================")
    bath.bose=None
    red=redfield.redfield(Hsys=H, t=times, baths=[bath],
    Qs=[Q],
    eps=1e -12,matsubara=False)
    result_red = red.evolution(rho0)
    result_red = [Qobj(i) for i in result_red]
    for i in result_red:
        i.dims=H.dims
    result_red = rotation(result_red, H, times)
    print("===========================")
    print("Redfield Done")
    print("===========================")
    print("===========================")
    print("Solving Bloch -Redfield")
    print("===========================")
    a_ops = [[Q, bath.power_spectrum]]
    resultBR = brmesolve(H, rho0, times, a_ops=a_ops, options={
        "rtol": 1e -14}, sec_cutoff= -1)

    a_ops = [[Q, bath.power_spectrum]]
    resultBR2 = brmesolve(H, rho0, times, a_ops=a_ops, options={
        "rtol": 1e -14})
    print("===========================")
    print("Bloch -Redfield Done")
    print("===========================")
```

```
    print("===========================")
    print("Solving Global")
    print("===========================")
    global_one=cum.jump_operators(Q) # Global Jump Operators for Bath 1 2 ->4
    c_ops2=[Qobj((np.sqrt(bath.power_spectrum(k))*v).data) for k, v in global_one.items()]
    for i in range(len(c_ops2)):
        c_ops2[i].dims=H.dims
    result_lindblad_global2 = mesolve(H, rho0, times, c_ops2)
    print("===========================")
    print("Global Done")
    print("===========================")
    print("===========================")
    print("Solving Pseudomodes")
    print("===========================")
    Ncutoff=levels
    bathu = zero_temp_bath(Q, tfit, lam, gamma, w0, N=modes)
    print(bathu.finfo["summary"])
    example = pseudomode(Hsys=H, Q=Q, bath=bathu)
    ans = example.evolution(rho0, Ncutoff, times, options={
                        "atol": 1e -14, "normalize_output": False, "store_states": True})
    ans = [i.ptrace(range(N))for i in ans.states]
    print("===========================")
    print("Pseudomodes done")
    print("===========================")
    results=[result,result_cum,resultBR,result_lindblad_global2,result_red,ans]
    return results


results_syk=solve_dynamics(H,Q,bath,bath1,rho0)

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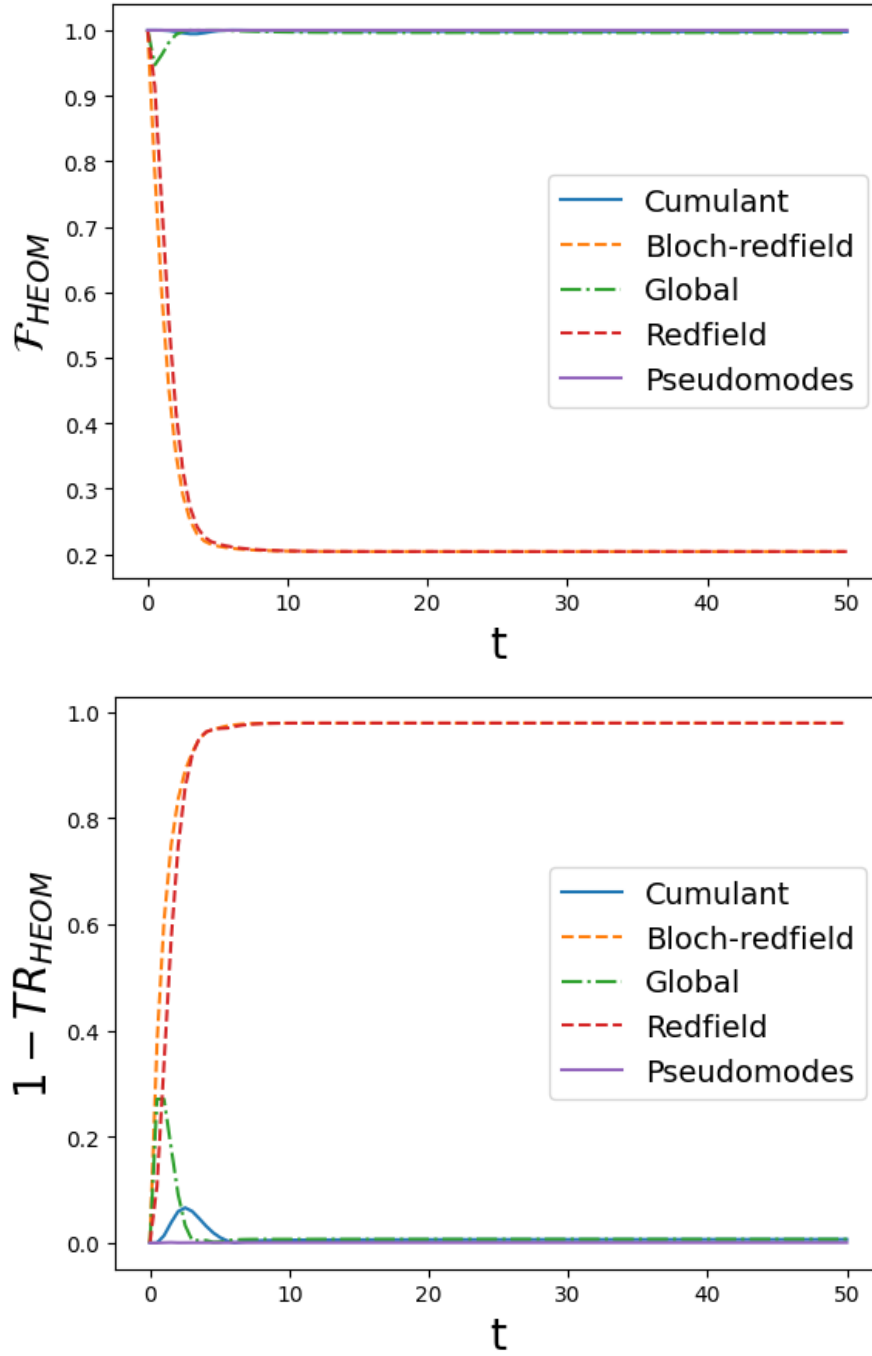
From what we see in both the trace distance and fidelity plots, the Bloch-Redfield approach does terribly when we consider this scenario (multiple implementations where checked). Notice that this issue seems to be about the coupling operator, rather than the Hamiltonian. Consider a different coupling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

```
results_syk2=solve_dynamics(H,Q2,bath,bath1,rho0,modes=2)


===========================
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 the Correlation Function with 2 terms: | | | | Result of fitting The 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
============================
```
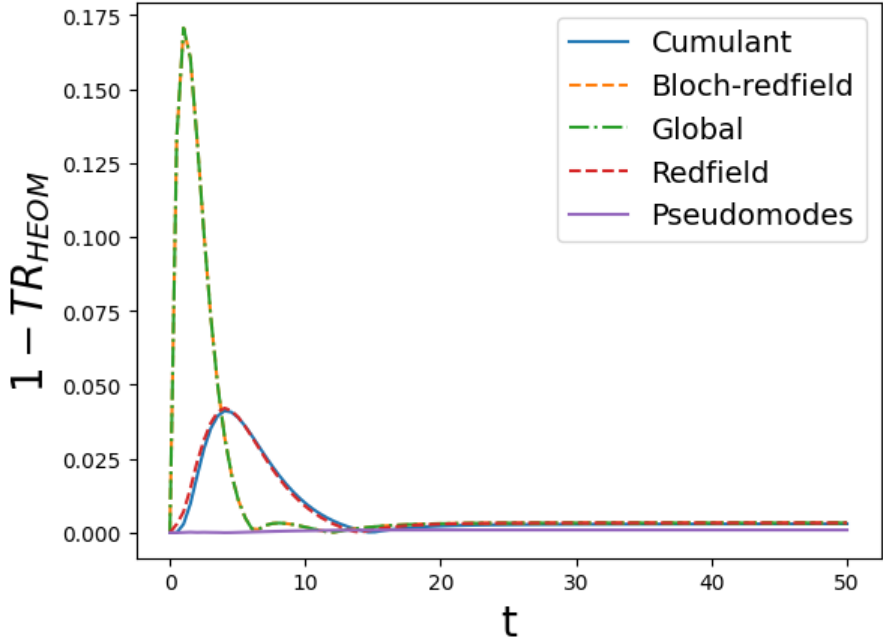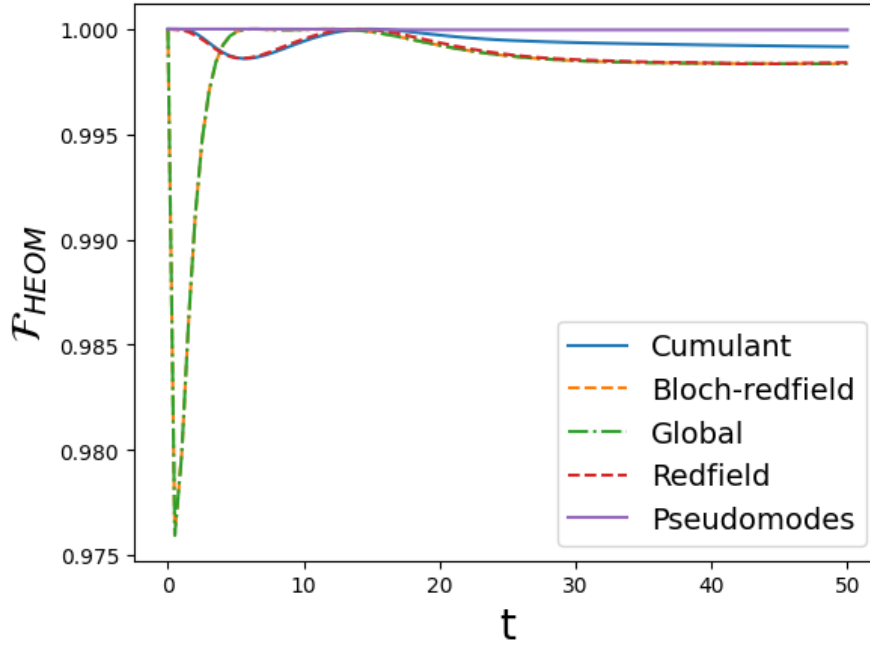
> **Tip**
> Whenever pseudomodes don't work it mainly can be fix with a more delicate fit (usually increasing levels is not worthwile pursuing in this regime)

**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 $\gamma = 4.984282088163084$ , $\lambda = 3.1478880316804854$ ,$\omega_0 = 10.217778280734322$. 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
```

```python
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()
```
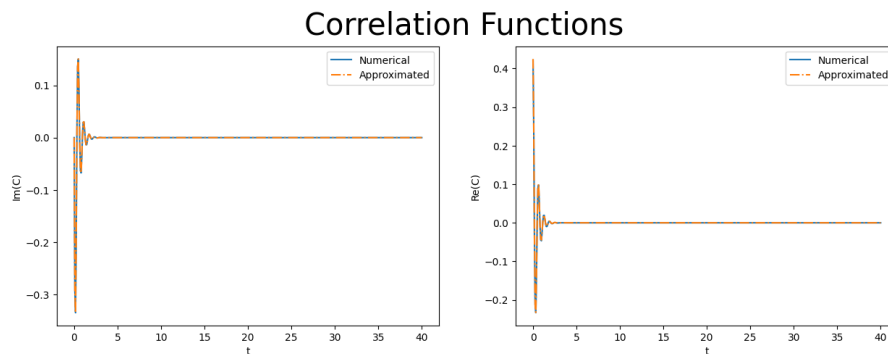


```python
N=3
state_list = [basis(2, 1)] + [ -1j*basis(2, 0)] * (N - 1) # change the initial state to be awa
state_list2 = [basis(2, 1)] + [basis(2, 0)] * (N - 1) # change the initial state to be away fr
state_list.reverse()
psi0 = (tensor(state_list)+tensor(state_list2))/np.sqrt(2)
rho0=psi0*psi0.dag()

example_ising=solve_dynamics(H,Q,bath,bath1,rho0,modes=1,depth=5h,times=np.linspace(0,200,50))
```

```
==========================
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 ->

plot_fidelities(example_ising,H,times)

trd(example_ising,H,times)

def plot_populations(states,l=3,m=3):
    times=states[0].times
    labels=["HEOM","Cumulant","Bloch -redfield","Bloch -redfield PS","Pseudomodes"]
    for k,i in enumerate(states):
        try:
            sdd=np.array([j[l,m] for j in i.states])
        except:
            sdd=np.array([j[l,m] for j in i])

        plt.plot(times,sdd,label=labels[k])
    plt.legend(fontsize=14)
    plt.ylabel(rf"$\rho_{l,m}$",fontsize=20)
    plt.xlabel(r"t",fontsize=20)
    plt.show()

plot_populations(example_ising)
```

### 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_th
```

```python
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

$$H = \begin{bmatrix} -a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & -a \end{bmatrix}$$

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

$$q = \begin{bmatrix} 0 & b_0 - ib_1 & b_2 - ib_3 & 0 \\ b_0 + ib_1 & 0 & 0 & -b_2 + ib_3 \\ b_2 + ib_3 & 0 & 0 & b_0 - ib_1 \\ 0 & -b_2 - ib_3 & b_0 + ib_1 & 0 \end{bmatrix}$$

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 $(\omega, \omega')$ 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

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

For convenience we also define

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

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

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

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

$$\Gamma(w) = \frac{2\gamma\lambda^2 w \left(1 + \frac{1}{e^{\frac{w}{T}}-1}\right)}{\gamma^2 w^2 + \left(-w^2 + w_0^2\right)^2}$$

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

$\frac{d}{dt}\rho_1(t) = 4ac_1 \left(c_2\rho_{10}(t)\overline{c_3} + c_2\rho_6(t)\overline{c_2} + c_3\rho_{11}(t)\overline{c_3} + c_3\rho_7(t)\overline{c_2}\right)$

$\frac{d}{dt}\rho_2(t) = c_0\rho_2(t)$

$\frac{d}{dt}\rho_3(t) = c_0\rho_3(t)$

$\frac{d}{dt}\rho_4(t) = 4ac_1 \left(-\rho_{10}(t)\overline{c_3}^2 + \rho_{11}(t)\overline{c_2c_3} - \rho_6(t)\overline{c_2c_3} + \rho_7(t)\overline{c_2}^2\right)$

$\frac{d}{dt}\rho_5(t) = ac_1 \left(4c_2^2\rho_2(t) + 4c_2c_3\rho_3(t) - 4c_2\rho_{14}(t)\overline{c_3} - 4c_3\rho_{15}(t)\overline{c_3} - 2.0\left(c_2\overline{c_2} + c_3\overline{c_3}\right)\rho_5(t)e^{4iat}\right)e^{-4iat}$

$\frac{d}{dt}\rho_6(t) = 2c_0\rho_6(t)$

$\frac{d}{dt}\rho_7(t) = 2c_0\rho_7(t)$

$\frac{d}{dt}\rho_8(t) = ac_1 \left(-4c_2\rho_2(t)\overline{c_3} + 4c_2\rho_3(t)\overline{c_2} - 2.0\left(c_2\overline{c_2} + c_3\overline{c_3}\right)\rho_8(t)e^{4iat} + 4\rho_{14}(t)\overline{c_3}^2 - 4\rho_{15}(t)\overline{c_2c_3}\right)e^{-4iat}$

$\frac{d}{dt}\rho_9(t) = ac_1 \left(4c_2c_3\rho_2(t) + 4c_2\rho_{14}(t)\overline{c_2} + 4c_3^2\rho_3(t) + 4c_3\rho_{15}(t)\overline{c_2} - 2.0\left(c_2\overline{c_2} + c_3\overline{c_3}\right)\rho_9(t)e^{4iat}\right)e^{-4iat}$

$\frac{d}{dt}\rho_{10}(t) = 2c_0\rho_{10}(t)$

$\frac{d}{dt}\rho_{11}(t) = 2c_0\rho_{11}(t)$

$\frac{d}{dt}\rho_{12}(t) = ac_1 \left(-4c_3\rho_2(t)\overline{c_3} + 4c_3\rho_3(t)\overline{c_2} - 2.0\left(c_2\overline{c_2} + c_3\overline{c_3}\right)\rho_{12}(t)e^{4iat} - 4\rho_{14}(t)\overline{c_2c_3} + 4\rho_{15}(t)\overline{c_2}^2\right)e^{-4iat}$

$\frac{d}{dt}\rho_{13}(t) = 4ac_1 \left(c_2^2\rho_{10}(t) + c_2c_3\rho_{11}(t) - c_2c_3\rho_6(t) - c_3^2\rho_7(t)\right)$

$\frac{d}{dt}\rho_{14}(t) = c_0\rho_{14}(t)$

$\frac{d}{dt}\rho_{15}(t) = c_0\rho_{15}(t)$

$\frac{d}{dt}\rho_{16}(t) = 4ac_1 \left(-c_2\rho_{10}(t)\overline{c_3} + c_2\rho_{11}(t)\overline{c_2} + c_3\rho_6(t)\overline{c_3} - c_3\rho_7(t)\overline{c_2}\right)$

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

$c_0 = \frac{2.0a\gamma\lambda^2\left(-b_0^2 - b_1^2 - b_2^2 - b_3^2\right)}{16.0a^4 + 4.0a^2\gamma^2 - 8.0a^2w_0^2 + 1.0w_0^4}$

$c_1 = \frac{\gamma\lambda^2}{4a^2\gamma^2 + \left(4a^2 - w_0^2\right)^2}$

$c_2 = b_0 + ib_1$

$c_3 = b_2 + ib_3$

By substituting these into the differential equation we obtain

$\frac{d}{dt}\rho_1(t) = 4ac_1 \left(c_2\rho_{10}(t)\overline{c_3} + c_2\rho_6(t)\overline{c_2} + c_3\rho_{11}(t)\overline{c_3} + c_3\rho_7(t)\overline{c_2}\right)$

$\frac{d}{dt}\rho_2(t) = c_0\rho_2(t)$

$\frac{d}{dt}\rho_3(t) = c_0\rho_3(t)$

$\frac{d}{dt}\rho_4(t) = 4ac_1 \left(-\rho_{10}(t)\overline{c_3}^2 + \rho_{11}(t)\overline{c_2c_3} - \rho_6(t)\overline{c_2c_3} + \rho_7(t)\overline{c_2}^2\right)$

$\frac{d}{dt}\rho_5(t) = ac_1 \left(4c_2^2\rho_2(t) + 4c_2c_3\rho_3(t) - 4c_2\rho_{14}(t)\overline{c_3} - 4c_3\rho_{15}(t)\overline{c_3} - 2.0\left(c_2\overline{c_2} + c_3\overline{c_3}\right)\rho_5(t)e^{4iat}\right)e^{-4iat}$

$\frac{d}{dt}\rho_6(t) = 2c_0\rho_6(t)$

$\frac{d}{dt}\rho_7(t) = 2c_0\rho_7(t)$

$\frac{d}{dt}\rho_8(t) = ac_1 \left(-4c_2\rho_2(t)\overline{c_3} + 4c_2\rho_3(t)\overline{c_2} - 2.0\left(c_2\overline{c_2} + c_3\overline{c_3}\right)\rho_8(t)e^{4iat} + 4\rho_{14}(t)\overline{c_3}^2 - 4\rho_{15}(t)\overline{c_2c_3}\right)e^{-4iat}$

$\frac{d}{dt}\rho_9(t) = ac_1 \left(4c_2c_3\rho_2(t) + 4c_2\rho_{14}(t)\overline{c_2} + 4c_3^2\rho_3(t) + 4c_3\rho_{15}(t)\overline{c_2} - 2.0\left(c_2\overline{c_2} + c_3\overline{c_3}\right)\rho_9(t)e^{4iat}\right)e^{-4iat}$

$\frac{d}{dt}\rho_{10}(t) = 2c_0\rho_{10}(t)$

$\frac{d}{dt}\rho_{11}(t) = 2c_0\rho_{11}(t)$

$\frac{d}{dt}\rho_{12}(t) = ac_1\left(-4c_3\rho_2(t)\overline{c_3} + 4c_3\rho_3(t)\overline{c_2} - 2.0\left(c_2\overline{c_2} + c_3\overline{c_3}\right)\rho_{12}(t)e^{4iat} - 4\rho_{14}(t)\overline{c_2}\overline{c_3} + 4\rho_{15}(t)\overline{c_2}^2\right)e^{-4iat}$

$\frac{d}{dt}\rho_{13}(t) = 4ac_1\left(c_2^2\rho_{10}(t) + c_2c_3\rho_{11}(t) - c_2c_3\rho_6(t) - c_3^2\rho_7(t)\right)$

$\frac{d}{dt}\rho_{14}(t) = c_0\rho_{14}(t)$

$\frac{d}{dt}\rho_{15}(t) = c_0\rho_{15}(t)$

$\frac{d}{dt}\rho_{16}(t) = 4ac_1\left(-c_2\rho_{10}(t)\overline{c_3} + c_2\rho_{11}(t)\overline{c_2} + c_3\rho_6(t)\overline{c_3} - c_3\rho_7(t)\overline{c_2}\right)$

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

$$\rho_5(t) = 1.0C_9e^{-t(2.0ac_1c_2\overline{c_2}+2.0ac_1c_3\overline{c_3})} + \left(4.0C_3ac_1c_2^2 + 4.0C_4ac_1c_2c_3 - 4.0C_7ac_1c_2\overline{c_3} - 4.0C_8ac_1c_3\overline{c_3}\right)\left(\left\{\begin{array}{l}\frac{8.0e^{c_0t}e^{-4\cdots}}{16.0ac_1c_2\overline{c_2}\cdots} \\ t\end{array}\right.\right.$$

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

$$\rho(0) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5i & 0 \\ 0 & -0.5i & 0.5 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The solution to the system of equations is

$\rho_1(t) = 1.0C_1 + 1.0C_2e^{2.0c_0t}$

$\rho_2(t) = C_3e^{c_0t}$

$\rho_3(t) = C_4e^{c_0t}$

$\rho_4(t) = 1.0C_5 + 1.0C_6e^{2.0c_0t}$

$$\rho_5(t) = 1.0C_9e^{-t(2.0ac_1c_2\overline{c_2}+2.0ac_1c_3\overline{c_3})} + \left(4.0C_3ac_1c_2^2 + 4.0C_4ac_1c_2c_3 - 4.0C_7ac_1c_2\overline{c_3} - 4.0C_8ac_1c_3\overline{c_3}\right)\left(\left\{\begin{array}{l}\frac{8.0e^{c_0t}e^{-4\cdots}}{16.0ac_1c_2\overline{c_2}\cdots} \\ t\end{array}\right.\right.$$

$$\rho_6(t) = \left(\frac{C_{10}c_0c_3\overline{c_3}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2}\overline{c_3}+2.0c_3^2\overline{c_3}^2\right)} - \frac{C_{11}c_0\overline{c_2}\overline{c_3}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2}\overline{c_3}+2.0c_3^2\overline{c_3}^2\right)} + \frac{C_2c_0c_2\overline{c_2}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2}\overline{c_3}+2.0c_3^2\overline{c_3}^2\right)} - \frac{\cdots}{ac_1\cdots}\right.$$

$$\rho_7(t) = -\left(\frac{C_{10}c_0c_2\overline{c_3}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2}\overline{c_3}+2.0c_3^2\overline{c_3}^2\right)} + \frac{C_{11}c_0\overline{c_3}^2}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2}\overline{c_3}+2.0c_3^2\overline{c_3}^2\right)} - \frac{C_2c_0c_2\overline{c_3}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2}\overline{c_3}+2.0c_3^2\overline{c_3}^2\right)} - \frac{\cdots}{ac\cdots}\right.$$

$$\rho_8(t) = 1.0C_{12}e^{-t(2.0ac_1c_2\overline{c_2}+2.0ac_1c_3\overline{c_3})} - \left(4.0C_3ac_1c_2\overline{c_3} - 4.0C_4ac_1c_2\overline{c_2} - 4.0C_7ac_1\overline{c_3}^2 + 4.0C_8ac_1\overline{c_2}\overline{c_3}\right)\left(\left\{\begin{array}{l}\frac{8.0e^{c_0t}e^{\cdots}}{16.0ac_1c_2\cdots} \\ t\end{array}\right.\right.$$

$$\rho_9(t) = 1.0C_{13}e^{-t(2.0ac_1c_2\overline{c_2}+2.0ac_1c_3\overline{c_3})} + \left(4.0C_3ac_1c_2c_3 + 4.0C_4ac_1c_3^2 + 4.0C_7ac_1c_2\overline{c_2} + 4.0C_8ac_1c_3\overline{c_2}\right)\left(\left\{\begin{array}{l}\frac{8.0e^{c_0t}e^{-\cdots}}{16.0ac_1c_2\overline{c_2}\cdots} \\ t\end{array}\right.\right.$$

$$\rho_{10}(t) = -\left(\frac{C_{10}c_0c_3\overline{c_2}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2}\overline{c_3}+2.0c_3^2\overline{c_3}^2\right)} - \frac{C_{11}c_0\overline{c_2}^2}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2}\overline{c_3}+2.0c_3^2\overline{c_3}^2\right)} - \frac{C_2c_0c_3\overline{c_2}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2}\overline{c_3}+2.0c_3^2\overline{c_3}^2\right)} + \cdots\right.$$

$$\rho_{11}(t) = \left(\frac{C_{10}c_0c_2\overline{c_2}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2}\overline{c_3}+2.0c_3^2\overline{c_3}^2\right)} + \frac{C_{11}c_0\overline{c_2}\overline{c_3}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2}\overline{c_3}+2.0c_3^2\overline{c_3}^2\right)} + \frac{C_2c_0c_3\overline{c_3}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2}\overline{c_3}+2.0c_3^2\overline{c_3}^2\right)} + \frac{\cdots}{ac_1\cdots}\right.$$

$$\rho_{12}(t) = 1.0C_{14}e^{-t(2.0ac_1c_2\overline{c_2}+2.0ac_1c_3\overline{c_3})} - \left(4.0C_3ac_1c_3\overline{c_3} - 4.0C_4ac_1c_3\overline{c_2} + 4.0C_7ac_1\overline{c_2}\overline{c_3} - 4.0C_8ac_1\overline{c_2}^2\right)\left(\left\{\begin{array}{l}\frac{8.0e^{c_0t}\cdots}{16.0ac_1\cdots} \\ t\end{array}\right.\right.$$

$\rho_{13}(t) = 1.0C_{11}e^{2.0c_0t} + 1.0C_{15}$

$\rho_{14}(t) = C_7e^{c_0t}$

$\rho_{15}(t) = C_8e^{c_0t}$

$\rho_{16}(t) = 1.0C_{10}e^{2.0c_0t} + 1.0C_{16}$

Then by substituting this into the solution we obtain to find the constants we obtain

$\rho_1(t) = \frac{1.0ac_1(c_2\overline{c_2}-ic_2\overline{c_3}+ic_3\overline{c_2}+c_3\overline{c_3})e^{2.0c_0t}}{c_0} - \frac{1.0ac_1(c_2\overline{c_2}-ic_2\overline{c_3}+ic_3\overline{c_2}+c_3\overline{c_3})}{c_0}$

$\rho_2(t) = 0$

$\rho_3(t) = 0$

$$\rho_4(t) = \frac{1.0iac_1\left(\overline{c_2}^2+\overline{c_3}^2\right)e^{2.0c_0t}}{c_0} - \frac{1.0iac_1\left(\overline{c_2}^2+\overline{c_3}^2\right)}{c_0}$$

$$\rho_5(t) = 0$$

$$\rho_6(t) = \frac{c_0\left(-\frac{iac_1c_2c_3\left(\overline{c_2}^2+\overline{c_3}^2\right)}{c_0}+\frac{ac_1c_2(c_2\overline{c_2}-ic_2\overline{c_3}+ic_3\overline{c_2}+c_3\overline{c_3})\overline{c_2}}{c_0}+\frac{1.0ac_1c_3(c_2\overline{c_2}+ic_2\overline{c_3}-ic_3\overline{c_2}+c_3\overline{c_3})\overline{c_3}}{c_0}-\frac{iac_1\left(-c_2^2-c_3^2\right)\overline{c_2c_3}}{c_0}\right)e^{2.0c_0t}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2c_3}+2.0c_3^2\overline{c_3}^2\right)}$$

$$\rho_7(t) = \frac{c_0\left(\frac{iac_1c_2^2\left(\overline{c_2}^2+\overline{c_3}^2\right)}{c_0}+\frac{ac_1c_2(c_2\overline{c_2}-ic_2\overline{c_3}+ic_3\overline{c_2}+c_3\overline{c_3})\overline{c_3}}{c_0}-\frac{1.0ac_1c_2(c_2\overline{c_2}+ic_2\overline{c_3}-ic_3\overline{c_2}+c_3\overline{c_3})\overline{c_3}}{c_0}-\frac{iac_1\left(-c_2^2-c_3^2\right)\overline{c_3}^2}{c_0}\right)e^{2.0c_0t}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2c_3}+2.0c_3^2\overline{c_3}^2\right)}$$

$$\rho_8(t) = 0$$

$$\rho_9(t) = 0$$

$$\rho_{10}(t) = \frac{c_0\left(-\frac{iac_1c_3^2\left(\overline{c_2}^2+\overline{c_3}^2\right)}{c_0}+\frac{ac_1c_3(c_2\overline{c_2}-ic_2\overline{c_3}+ic_3\overline{c_2}+c_3\overline{c_3})\overline{c_2}}{c_0}-\frac{1.0ac_1c_3(c_2\overline{c_2}+ic_2\overline{c_3}-ic_3\overline{c_2}+c_3\overline{c_3})\overline{c_2}}{c_0}+\frac{iac_1\left(-c_2^2-c_3^2\right)\overline{c_2}^2}{c_0}\right)e^{2.0c_0t}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2c_3}+2.0c_3^2\overline{c_3}^2\right)}$$

$$\rho_{11}(t) = \frac{c_0\left(\frac{iac_1c_2c_3\left(\overline{c_2}^2+\overline{c_3}^2\right)}{c_0}+\frac{1.0ac_1c_2(c_2\overline{c_2}+ic_2\overline{c_3}-ic_3\overline{c_2}+c_3\overline{c_3})\overline{c_2}}{c_0}+\frac{ac_1c_3(c_2\overline{c_2}-ic_2\overline{c_3}+ic_3\overline{c_2}+c_3\overline{c_3})\overline{c_3}}{c_0}+\frac{iac_1\left(-c_2^2-c_3^2\right)\overline{c_2c_3}}{c_0}\right)e^{2.0c_0t}}{ac_1\left(2.0c_2^2\overline{c_2}^2+4.0c_2c_3\overline{c_2c_3}+2.0c_3^2\overline{c_3}^2\right)}$$

$$\rho_{12}(t) = 0$$

$$\rho_{13}(t) = \frac{1.0iac_1\left(-c_2^2-c_3^2\right)e^{2.0c_0t}}{c_0} - \frac{1.0iac_1\left(-c_2^2-c_3^2\right)}{c_0}$$

$$\rho_{14}(t) = 0$$

$$\rho_{15}(t) = 0$$

$$\rho_{16}(t) = \frac{1.0ac_1(c_2\overline{c_2}+ic_2\overline{c_3}-ic_3\overline{c_2}+c_3\overline{c_3})e^{2.0c_0t}}{c_0} - \frac{1.0ac_1(c_2\overline{c_2}+ic_2\overline{c_3}-ic_3\overline{c_2}+c_3\overline{c_3})}{c_0}$$

By reverting the change of variables. The analytical answer of the Bloch redfield equation is given by

$$\rho_1(t) = \frac{0.5\left(e^{\frac{4.0a\gamma\lambda^2t\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}{16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1.0w_0^4}}-1\right)\left(16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1.0w_0^4\right)((b_0-ib_1)(b_0+ib_1)+i(b_0-ib_1)(b_2+ib_3)-i(b_0+ib_1)(b_2-\ }{\left(4a^2\gamma^2+\left(4a^2-w_0^2\right)^2\right)\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}$$

$$\rho_2(t) = 0$$

$$\rho_3(t) = 0$$

$$\rho_4(t) = \frac{0.5i\left((b_0-ib_1)^2+(b_2-ib_3)^2\right)\left(e^{\frac{4.0a\gamma\lambda^2t\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}{16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1.0w_0^4}}-1\right)\left(16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1.0w_0^4\right)e^{-\frac{4.0a\gamma\lambda^2t\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}{16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1}}}{\left(4a^2\gamma^2+\left(4a^2-w_0^2\right)^2\right)\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}$$

$$\rho_5(t) = 0$$

$$\rho_6(t) = 0.5e^{-\frac{4.0a\gamma\lambda^2t\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}{16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1.0w_0^4}}$$

$$\rho_7(t) = 0.5ie^{-\frac{4.0a\gamma\lambda^2t\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}{16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1.0w_0^4}}$$

$$\rho_8(t) = 0$$

$$\rho_9(t) = 0$$

$$\rho_{10}(t) = -0.5ie^{-\frac{4.0a\gamma\lambda^2t\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}{16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1.0w_0^4}}$$

$$\rho_{11}(t) = 0.5e^{-\frac{4.0a\gamma\lambda^2t\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}{16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1.0w_0^4}}$$

$$\rho_{12}(t) = 0$$

$$\rho_{13}(t) = \frac{0.5i\left(1-e^{\frac{4.0a\gamma\lambda^2t\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}{16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1.0w_0^4}}\right)\left((b_0+ib_1)^2+(b_2+ib_3)^2\right)\left(16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1.0w_0^4\right)e^{-\frac{4.0a\gamma\lambda^2t\left(b_0^2+b_1^2+b_2^2+b\right)}{16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+}}}{\left(4a^2\gamma^2+\left(4a^2-w_0^2\right)^2\right)\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}$$

$$\rho_{14}(t) = 0$$

$$\rho_{15}(t) = 0$$

$$\rho_{16}(t) = \frac{0.5\left(e^{\frac{4.0a\gamma\lambda^2 t\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}{16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1.0w_0^4}}-1\right)\left(16.0a^4+4.0a^2\gamma^2-8.0a^2w_0^2+1.0w_0^4\right)\left((b_0-ib_1)(b_0+ib_1)-i(b_0-ib_1)(b_2+ib_3)+i(b_0+ib_1)(b_2)\right.}{\left(4a^2\gamma^2+\left(4a^2-w_0^2\right)^2\right)\left(b_0^2+b_1^2+b_2^2+b_3^2\right)}$$

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

$$\rho(t) = \begin{bmatrix} 0.55247613848278\left(e^{0.94679965816826t}-1.0\right)e^{-0.94679965816826t} & 0 \\ 0 & 0.5e^{-0.946799} \\ 0 & -0.5ie^{-0.94679} \\ (-0.486572940196368-0.102435485836685i)\left(1.0-e^{0.94679965816826t}\right)e^{-0.94679965816826t} & 0 \end{bmatrix}$$

Then we may evaluate for long times

$$\rho(50) = \begin{bmatrix} 0.55247613848278 & 0 & 0 & 0.486572940196368-0.102435485836685i \\ 0 & 0.0 & 0 & 0 \\ 0 & 0 & 0.0 & 0 \\ 0.486572940196368+0.102435485836685i & 0 & 0 & 0.44752386151722 \end{bmatrix}$$

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)`

$$U = \begin{bmatrix} e^{-iat} & 0 & 0 & 0 \\ 0 & e^{iat} & 0 & 0 \\ 0 & 0 & e^{iat} & 0 \\ 0 & 0 & 0 & e^{-iat} \end{bmatrix}$$

$$U\rho U^\dagger = \begin{bmatrix} \rho_1 & \rho_2 e^{-2iat} & \rho_3 e^{-2iat} & \rho_4 \\ e^{2iat}\overline{\rho_2} & \rho_6 & \overline{\rho_{10}} & \rho_8 e^{2iat} \\ e^{2iat}\overline{\rho_3} & \rho_{10} & \rho_{11} & \rho_{12}e^{2iat} \\ \overline{\rho_4} & e^{-2iat}\overline{\rho_8} & e^{-2iat}\overline{\rho_{12}} & -\rho_1-\rho_{11}-\rho_6+1 \end{bmatrix}$$

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

$$\rho(50) = \begin{bmatrix} 0.55247613848278 & 0 & 0 & 0.486572940196368-0.102435485836685i \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.486572940196368+0.102435485836685i & 0 & 0 & 0.44752386151722 \end{bmatrix}$$

### 0.2.1   RC picture of the Hamiltonian

For the RC I simply follow one of your papers https://arxiv.org/pdf/1511.05181

So If I didn't misunderstand it then
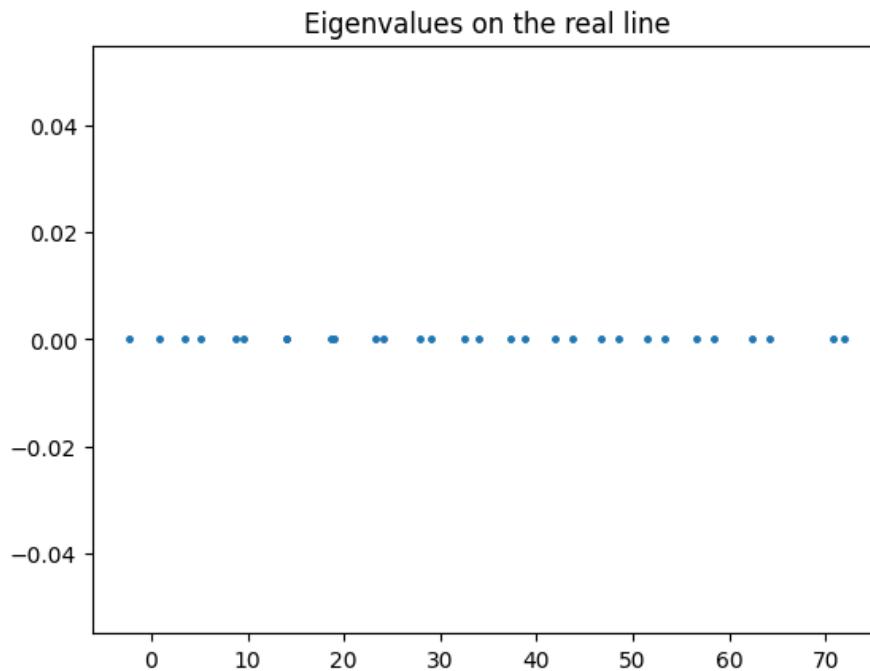
$$\Omega = w_0 \tag{18}$$

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

Not actually sure if $\pi$ 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', 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', 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()
```
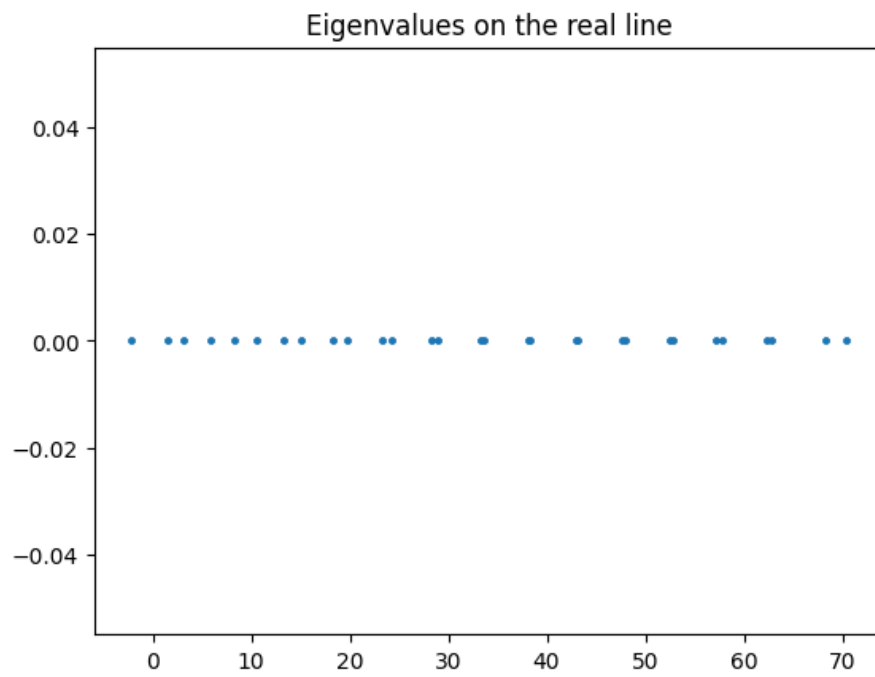
There doesn't seem to be much change in the Hamiltonian if any

## 0.3   Hamiltonian Simulation

### 0.3.1   When the Hamiltonian is Physical

When the Hamiltonian is physical we can take the same steps as people who simulate the Lindblad master equation. I followed the Lin Lin paper. Here's a bit of the paper but instead of using SDE schemes, I do it on the ensemble. To obtain the Krauss operators (I also only do it to first order, because it is simplest and should work when $dt \to 0$ but probably one should consider higher orders).

### 0.3.2   Krauss Operator from a Lindbladian

> **Warning**
> This might be a mistake from the very beginning as Pseudomodes is not a CPTP map.However numerically, I've never seen any issue with positivity when enough levels are considered in the modes. So from here I'm assuming it will be CPTP

The logic here is to start from the master equation

$$\dot{\rho(t)} = \mathcal{L}(\rho(t)) \tag{20}$$

From the definition of derivative this means

$$\lim_{dt \to 0} \frac{\rho(t + dt) - \rho(t)}{dt} = \mathcal{L}(\rho(t)) \tag{21}$$

For now let us forget about the limit, but work our quantities approximately and to order $\mathcal{O}(dt^2)$ so that

$$\rho(t + dt) \approx \rho(t) + \mathcal{L}(\rho(t))dt + \mathcal{O}(dt^2) \tag{22}$$

Since the map is CPTP then it must have a sum operator representation (Krauss representation) so

$$\rho(t + dt) = \sum_k M_k \rho(t) M_k^\dagger \tag{23}$$

What's left now is to find what the krauss operators should be. This is known and can be seen for example in Lidar's lecture notes. Here I do some extra algebra to illustrate how to generalize to higher order schemes

Since this is lowest order then, we propose the Krauss operators

$$M_0 = \infty + A dt \tag{24}$$

$$M_k = \sqrt{dt} B_k \tag{25}$$

Then we find that

> **Warning**
> I did these calculations by hand, but I am really lazy when it comes to latexing so I decided to use sympy for intermediate steps, if there's any inconsistency I can just latex those steps

$\rho(dt + t) = \sum_{k=1}^{N} dt B_k \rho(t) B_k^\dagger + \rho(t) + dt \rho(t) A^\dagger + dt A \rho(t) + O\left(dt^2\right)$

Notice the series of Krauss operators we used have identical contributions (in their form, and could be represented as a sum). Notice on the other hand we have

$$\rho(t + dt) \approx \rho(t) + \mathcal{L}(\rho(t))dt + \mathcal{O}(dt^2) \tag{26}$$

By replacing the lindbladian one obtains

$$\rho(t+dt) \approx \rho(t) + \left(-i[H, \rho(t)] + \sum_k L_k \rho(t) L_k^\dagger - \frac{\{L_k^\dagger L_k, \rho(t)\}}{2}\right) dt \tag{27}$$

where the $L_k$ are the jump operators. Then by comparison we can find the required Krauss operators, First let us note That to generate a commutator A must be an Anti-Hermitian matrix, and to generate an anticommutator A must be Hermitian. If we choose A to be a sum of a Hermitian and Anti-hermitian matrix then

$A = -iH + K$

$\rho(dt+t) = \sum_{k=1}^N dt B_k \rho(t) B_k^\dagger + \rho(t) + dt\rho(t)K + idt\rho(t)H + dtK\rho(t) - idtH\rho(t) + O\left(dt^2\right)$

Which we can simplify to

$\rho(dt+t) = \sum_{k=1}^N dtL\rho(t)L^\dagger + \rho(t) - \frac{dt\left(\sum_{k=1}^N L_k^\dagger L_k\right)\rho(t)}{2} - \frac{dt\rho(t)\sum_{k=1}^N L_k^\dagger L_k}{2} + idt\rho(t)H - idtH\rho(t) + O\left(dt^2\right)$

At this point notice that if we select the $B_k$ to be the jump operators and K to be the corresponding anticommutator term

$\rho(dt+t) = \sum_{k=1}^N dtL\rho(t)L^\dagger + \rho(t) - \frac{dt\left(\sum_{k=1}^N L_k^\dagger L_k\right)\rho(t)}{2} - \frac{dt\rho(t)\sum_{k=1}^N L_k^\dagger L_k}{2} + idt\rho(t)H - idtH\rho(t) + O\left(dt^2\right)$

We have obtained the First order scheme to obtain the Linblad master equation Krauss operators. To obtain Higher order schemes One can notice that the solution to the master equation is

$$\rho(t) = e^{\mathcal{L}t}\rho(0) \tag{28}$$

and

$$\rho(t+dt) = e^{\mathcal{L}(t+dt)}\rho(0) = e^{\mathcal{L}dt}\rho(t) \tag{29}$$

And then expand the series of the exponential. Similarly one should increase the order of the krauss operator guess by one and find the appropiate operators

### 0.3.3 Notice We could have not guess K and use the Completeness relation of krauss operators

In this case it was not needed but it might be useful to find relations in higher order schemes and to check the krauss operators are ok. So next we find K this way

The completeness relation indicates

$$\sum_k M_k^\dagger M_k = 1 \tag{30}$$

Since in our schemes we are numerically approximating to $\mathcal{O}(dt^2)$ then

$$\sum_k M_k^\dagger M_k = 1 + \mathcal{O}(dt^2) \tag{31}$$

$1 = 1 + dt\sum_{k=1}^N L_k^\dagger L_k + 2dtK + O\left(dt^2\right)$

Which we can solve to find

$K = -\frac{\sum_{k=1}^N L_k^\dagger L_k}{2} + O\left(dt\right)$

Now that we have found the Krauss operators one may simply ask if one can follow the same scheme to obtain the Krauss operators of a pseudomode equation. Since I have not seen positivity issues I do think it's possible. but the naive approach to it yields and inconsistency

### 0.3.4 Same derivation with a non-Hermitian Hamiltonian

Any non-Hermitian matrix can be split into the sum of a Hermitian and Anti-Hermitian Matrix such that I can write the unphysical Hamiltonian H as

$$H = H_0 + iH_u \tag{32}$$

Then the Lindblad equation turns into

$$\rho(t + dt) \approx \rho(t) + \left( -i[H_0, \rho(t)] + [H_u, \rho(t)] + \sum_k L_k\rho(t)L_k^\dagger - \frac{\{L_k^\dagger L_k, \rho(t)\}}{2} \right) dt \tag{33}$$

Following the same strategy as before only A changes (the part that generated the commutator), so the change is only on $M_0$ We neglect the part that contains K as that one does not change

$\rho(t) + idt\rho(t)H^\dagger - idtH\rho(t) + O\left(dt^2\right)$

Substitute

$$H = H_0 + iH_u \tag{34}$$

$\rho(t) + dt\left(H_u\rho(t) + \rho(t)H_u - i\left[H_0, \rho(t)\right]\right) + O\left(dt^2\right)$

By comparison we would need

$$\{H_u, \rho(t)\} = [H_u, \rho(t)] \tag{35}$$

Which cannot be satisfied

Even though this calculation was a failure. Perhaps one would need to use higher orders, or reorder the terms in another Fashion. I do believe it would be easier if I don't use the general formulation but the jump operators as $a$ and $a^\dagger$ and the unphysical part of the Hamiltonian to have the form $\sum_k \omega_k a_k^\dagger a_k$ where $\omega_k$ is complex.

While I look into it. Asume the Hamiltonian is Physical and then extrapolation is done. One has several schemes to simulate Krauss operators in a quantum circuit simulator. Let us go with the scheme in the Lin Lin paper

Even though in their case is not so bad, Here I illustrate why I don't like this Hamiltonian approach in the first order. Perhaps it is better to use the other 2nd Lin Lin paper though I think the number of ancillas needed will be bigger. I also need to try the Hush et al. [2015]

### 0.3.5 Hamiltonian Simulation From Krauss Operators

**In this section all traces are partial traces with respect to the ancilla**

The paper suggests using the Stinespring representation of the Krauss Operators Namely. Finding an ancilla and a matrix $\mu$ such that

$$Tr(\mu) = \sum_k M_k\rho(t)M_k^\dagger \tag{36}$$

To achieve this we can use the matrix $\mu$

$$\mu = \begin{pmatrix} M_0\rho(t)M_0^\dagger & M_0\rho(t)M_1^\dagger & \dots & M_0\rho(t)M_k^\dagger \\ M_1\rho(t)M_0^\dagger & M_1\rho(t)M_1^\dagger & \dots & M_1\rho(t)M_k^\dagger \\ \vdots & \vdots & \ddots & \vdots \\ M_k\rho(t)M_0^\dagger & \dots & \dots & M_k\rho(t)M_k^\dagger \end{pmatrix} \tag{37}$$

Which can be easily constructed by requiring one ancilla qubit for each jump operator. We consider all the ancillas to be on the ground state such that the state of the ancillas is

$$00 = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix}$$

Then one can obtain $\mu$ by

$$\mu = A00 \otimes \rho A^\dagger \tag{38}$$

Where

$$A = \begin{pmatrix} M_0 & M_1^\dagger & \dots & M_k^\dagger \\ M_1 & 0 & \dots & 0 \\ \vdots & 0 & \dots & 0 \\ M_k & 0 & \dots & 0 \end{pmatrix} \tag{39}$$

Then one can express the operator sum representation as

$$Tr(A00 \otimes \rho A^\dagger) = \sum_k M_k \rho(t) M_k^\dagger \tag{40}$$

To have a Hamiltonian simulation of the Krauss representation we want to find a unitary such that

$$Tr(U00 \otimes \rho U^\dagger) = Tr(A00 \otimes \rho A^\dagger) = \sum_k M_k \rho(t) M_k^\dagger \tag{41}$$

Or at least to order $\mathcal{O}(dt^2)$. One of the insights of the paper is to write U as

$$U = e^{-i\sqrt{dt}\bar{H}} \tag{42}$$

Where

$$\bar{H} = \begin{pmatrix} H_0 & H_1^\dagger & \dots & H_k^\dagger \\ H_1 & 0 & \dots & 0 \\ \vdots & 0 & \dots & 0 \\ H_k & 0 & \dots & 0 \end{pmatrix} \tag{43}$$

with Hermitian $H_0$. Then One can obtain $\bar{H}$ from taylor expanding the exponential and matching the same order terms. For simplicity here I do it for 4 jump operators

While at first order in the exponential we have

$$exp(x) \approx 1 + x \tag{44}$$

$$U = \begin{bmatrix} -i\sqrt{dt}H_0 + 1 & -i\sqrt{dt}H_1^\dagger & -i\sqrt{dt}H_2^\dagger & -i\sqrt{dt}H_3^\dagger & -i\sqrt{dt}H_4^\dagger \\ -i\sqrt{dt}H_1 & 1 & 0 & 0 & 0 \\ -i\sqrt{dt}H_2 & 0 & 1 & 0 & 0 \\ -i\sqrt{dt}H_3 & 0 & 0 & 1 & 0 \\ -i\sqrt{dt}H_4 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Again the goal is to have this emulate the krauss operators, which emulate the master equation

$\rho(dt+t) = \sum_{k=1}^N dt L\rho(t)L^\dagger + \rho(t) - \frac{dt\left(\sum_{k=1}^N L_k^\dagger L_k\right)\rho(t)}{2} - \frac{dt\rho(t)\sum_{k=1}^N L_k^\dagger L_k}{2} + idt\rho(t)H - idtH\rho(t) + O\left(dt^2\right)$

By using

$U\rho U^\dagger = \sqrt{dt}\left(i\rho(t)H_0 - iH_0\rho(t)\right) + dt\left(H_0\rho(t)H_0 + H_1\rho(t)H_1^\dagger + H_2\rho(t)H_2^\dagger + H_3\rho(t)H_3^\dagger + H_4\rho(t)H_4^\dagger\right) + \rho(t)$

We further simiplify it to be

$U\rho U^\dagger = -i\sqrt{dt}\left[H_0, \rho(t)\right] + dt\left(H_0\rho(t)H_0 + \sum_{k=1}^4 H_k\rho(t)H_k\right) + \rho(t)$

To approximate the master equation notice that we can have

$H_0 = \sqrt{(dt)}H$

$H_k = L_k$

Which is the first order in the Lin Lin paper and results in

$U\rho U^\dagger = dt\left(dtH\rho(t)H + \sum_{k=1}^4 L_k\rho(t)L_k\right) - idt\left[H, \rho(t)\right] + \rho(t)$

Then neglecting higher order terms one has

$U\rho U^\dagger = \rho(t) + dt \sum_{k=1}^4 L_k \rho(t) L_k - idt\left[H, \rho(t)\right] + O\left(dt^2\right)$

But this neglects the anticommutator bit This is what I don't like but it can be fixed by higher orders does not seem to affect accuracy too much

---

**TO DO!**

- Use higher order Schemes, and try to have a non-Hermitian Hamiltonian
- Test pseudomodes being CPTP by calculating $\tau := (\mathcal{L} \otimes \mathrm{id}_d)(|\Omega\rangle\langle\Omega|)$ where $\Omega$ is the maximally entangled state (if CPTP $\tau \geq 0$)
- Check other simulation schemes like the other lin lin paper or Clover's paper
- Actually do the Hamiltonian simulation, The ancillas need to be reset every timestep but this suceeds with probabulity one in theory

---

title: 1/f Noise date: 2024-08-08 authors:

- name: Gerardo Suarez

## 0.4 Dertivation of $\frac{1}{f}$ Noise based on 1 and 2

Nearly all optical and electronic systems, are subject to the so called $\frac{1}{f}$ noise. While a physical mechanism for $\frac{1}{f}$ noide is yet to be identified. There are specific physical models that give rise to $\frac{1}{f}$ noise. However, those are not necessarily unique.

Perhaps the Most relevant one, is $\frac{1}{f}$ arising from a superposition of train pulses.

### 0.4.1 Superposition of train pulses

**A Noisy waveform**

Consider a Noisy waveform $x(t)$ with a band pass filtered such that the power spectral density

$$S(\omega) = \begin{cases} \frac{\alpha}{\omega}, & \text{for } \omega_L \leq \omega \leq \omega_H \\ 0, & \text{otherwise} \end{cases} = \frac{\alpha}{\omega}\theta(\omega_H - \omega)\theta(\omega - \omega_L) \tag{45}$$

Then the auto-correlation function of $x(t)$ is then given by

$$C(\tau) = \langle x(t)x(t+\tau) \rangle \tag{46}$$

Let us remember that the power spectrum and the correlation function are related by

$$S(\omega) = \int_{-\infty}^{\infty} dt e^{i\omega t} C(t) \qquad = \mathcal{F}(C(t)) \tag{47}$$

Then

$$\mathcal{F}^{-1}(S(\omega)) = \mathcal{F}^{-1}(\mathcal{F}(C(t))) = C(t) \tag{48}$$

So finally we can write

$$C(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} d\omega e^{-i\omega t} S(\omega) \tag{49}$$

By substituting the power spectrum Eq. (**??**) then

$$C(t) = \frac{\alpha}{2\pi} \int_{\omega_L}^{\omega_H} \frac{d\omega}{\omega} e^{-i\omega t} \tag{50}$$

Which results in

$$C(t) = \frac{\alpha}{2\pi} \left( E_i(\omega_H t) - E_i(\omega_L t) \right) \tag{51}$$

where $E_i$ is the exponential integral function.

> **Warning**
> At this stage we are going to assume that this is a scalar stationary process, this means that the correlation function is even, which is not necessarily the case. In fact this model does not mathematically follow this. One could consider the imaginary (Quantum part of the noise as well)
> Then
>
> $$C(t) = \frac{C(t) - C(-t)}{2} = \frac{\alpha}{2\pi} \int_{\omega_L}^{\omega_H} \frac{d\omega}{\omega} \cos(\omega t) \tag{52}$$
>
> and
>
> $$C(t) = \frac{\alpha}{2\pi} \left( C_i(\omega_H t) - C_i(\omega_L t) \right) \tag{53}$$
>
> where $C_i$ is the cosine integral function.

### 0.4.2 The noisy waveform in terms of Train pulses (From Lorentzians)

Let us consider the waveform behaves like an exponential relaxation process. This would mean that our autocorrelatio function is given by

$$C(\tau) = \begin{cases} A_0 e^{-\lambda t} & \text{for } t > 0 \\ 0 & \text{for } t < 0 \end{cases} \tag{54}$$

It's Fourier transform is given by

$$\mathcal{F}(C(\tau)) = \int_{-\infty}^{\infty} e^{i\omega t} C(\tau) dt = A_0 \int_0^{\infty} e^{-(\lambda - i\omega)t} dt = \frac{A_0}{\lambda - i\omega} \tag{55}$$

If instead of a pulse we have a train of pulses

$$C(\tau) = \sum_k C(t, t_k) = \sum_k \begin{cases} A_0 e^{-\lambda(t - t_k)} & \text{for } t > t_k \\ 0 & \text{for } t < t_k \end{cases} \tag{56}$$

Then it's Fourier transform is given by

$$\mathcal{F}(C(\tau)) = \int_{-\infty}^{\infty} e^{i\omega t} C(\tau) dt = A_0 \int_0^{\infty} e^{-(\lambda - i\omega)t} dt = \frac{A_0}{\lambda - i\omega} \sum_k e^{i\omega t_k} \tag{57}$$

From here we can find the power spectrum via averaging

$$S(\omega) = \lim_{T \to \infty} \frac{1}{T} \langle |\mathcal{F}(C(\tau))|^2 \rangle = \frac{A_0^2}{\lambda^2 + \omega^2} \lim_{T \to \infty} \frac{\langle |\sum_k e^{i\omega t_k}|^2 \rangle}{T} \tag{58}$$

If

$$n = \lim_{T \to \infty} \frac{\langle |\sum_k e^{i\omega t_k}|^2 \rangle}{T} \tag{59}$$

Then we can write

$$S(\omega) = \frac{A_0^2 n}{\lambda^2 + \omega^2} \tag{60}$$

By identifying $\tau_z = \lambda^{-1}$ as the time scale of the pulses, and $g(\tau_z) = A_0^2 n \tau_z^2$ as the pulse generation details we may write the more popular expression

$$S(\omega) = \frac{g(\tau_z)}{1 + (\tau_z \omega)^2} \tag{61}$$

This is indeed a power spectrum, but it does not behave as $\frac{1}{f}$. However, suppose now that the noisy waveform we are considering is made out of a linear superposition of such processes and that it's decay time scales are between $\tau_1$ and $\tau_2$ with probability density $P(\tau_z)$ then the power spectrum of such superposition is given by

$$S(\omega) = \int_{\tau_1}^{\tau_2} d\tau_z P(\tau_z) \frac{g(\tau_z)}{1 + (\tau_z \omega)^2} \tag{62}$$

If we now assume that $P(\tau_z)g(\tau_z) \propto 1$. Meaning that it is independent of the decay time scale. Then

$$S(\omega) = \int_{\tau_1}^{\tau_2} d\tau_z \frac{\alpha}{1 + (\tau_z \omega)^2} = \frac{\alpha}{\omega}\left(\tan^{-1}(\omega\tau_2) - \tan^{-1}(\omega\tau_1)\right) \tag{63}$$

Finally when $\omega\tau_2 \gg 1$ and $0 < \omega\tau_1 \ll 1$ we have

$$\tan^{-1}(\omega\tau_2) \approx \frac{\pi}{2} \quad \tan^{-1}(\omega\tau_2) \approx 0 \tag{64}$$

So that in this regime, the superposition of different relaxation processes gives rise to the $\frac{1}{f}$ spectrum

$$S(\omega) = \frac{\alpha\pi}{2\omega} \tag{65}$$

On the other hand if $P(\tau_z)g(\tau_z) \propto \tau_z^{\beta-1}$ then

$$S(\omega) = \int_{\tau_1}^{\tau_2} d\tau_z \frac{\alpha\tau_z^{\beta-1}}{1 + (\tau_z \omega)^2} = \frac{\alpha}{\beta}\left(\tau_2^{\beta} F_1\left(1, \frac{\beta}{2}, 1 + \frac{\beta}{2}, -\omega^2\tau_2^2\right) - \tau_1^{\beta} F_1\left(1, \frac{\beta}{2}, 1 + \frac{\beta}{2}, -\omega^2\tau_1^2\right)\right) \tag{66}$$

using the same limit

$$S(\omega) \propto \frac{\alpha\pi}{2\omega^{\beta}} \tag{67}$$

```
/tmp/ipykernel_1381387/1893926815.py:2: RuntimeWarning: invalid value encountered in divide
  return (expi(1j*x)+expi( -1j*x))/2
/tmp/ipykernel_1381387/3153800274.py:2: RuntimeWarning: invalid value encountered in subtract
  cc=alpha*(ci(wh*t) -ci(wl*t))/(2)


Fit correlation class instance:

Result of fitting The Real Part Of                                    |Result of fitting The
 the Correlation Function with 8 terms:                               | Of the Correlation F
                                                                      |
Parameters|    a     |    b     |    c     |    d                     | Parameters|    a
1         | 2.73e -04 | -7.82e -02 | -9.97e -01 | -2.91e -03           | 1         | 0
2         | 1.70e -01 | -1.15e -02 | -9.94e -01 | -4.47e -02           |
3         | -1.70e -01 | -1.15e -02 | -9.94e -01 |4.40e -02            |A  normalized
4         | -8.58e -06 | -1.22e -03 | -1.00e+00 | -8.72e -05           | Of the Correl
5         | -4.60e -02 | -1.87e+01 | -5.99e -01 |1.62e+00             |
6         | 9.13e -01 | -1.87e+00 | 1.00e+00 | -9.18e -01             |
7         | -8.70e -01 | -1.92e+00 | 9.88e -01 |1.03e+00             |
8         | 2.91e -03 | -3.22e -01 | 9.95e -01 |1.00e -02              |
                                                                      |
A  normalized RMSE of  7.66e -07 was obtained for the The Real Part Of  |
 the Correlation Function                                             |
The current fit took  131.204410 seconds                             | The current fit took
```

**Issue:** The correlation function does not decay like one would expect. Most likely this is going to make things difficult, unless one devices how to do it with lorentzians in a limit, or some alternate formulation that yields an easier fit

```
/tmp/ipykernel_1381387/1893926815.py:2: RuntimeWarning: invalid value encountered in divide
  return (expi(1j*x)+expi( -1j*x))/2
/tmp/ipykernel_1381387/3153800274.py:2: RuntimeWarning: invalid value encountered in subtract
  cc=alpha*(ci(wh*t) -ci(wl*t))/(2)
/home/gerardo/miniconda3/envs/spinchain/lib/python3.12/site -packages/matplotlib/cbook.py:1762
  return math.isfinite(val)
/home/gerardo/miniconda3/envs/spinchain/lib/python3.12/site -packages/matplotlib/cbook.py:1398
  return np.asarray(x, float)
```
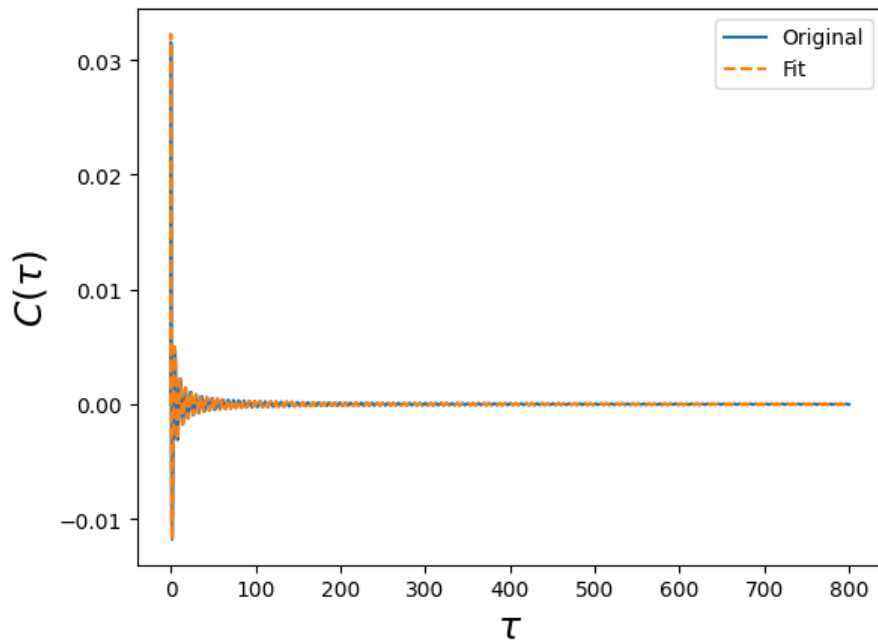


The fit of the correlation function is good, does this approximation actually show $\frac{1}{f}$ spectrum

```
/home/gerardo/qutip_gsoc_app/qutip/solver/heom/bofin_baths.py:510: RuntimeWarning: invalid val
  S += 2 * np.real((coeff) / (exp.vk - 1j*w))
/tmp/ipykernel_1381387/1336645498.py:3: RuntimeWarning: divide by zero encountered in divide
  s=np.heaviside(np.abs(w) -1,0)*np.heaviside(10000 -np.abs(w),0)*(alpha/abs(w))/(2*np.pi)
/tmp/ipykernel_1381387/1336645498.py:3: RuntimeWarning: invalid value encountered in multiply
  s=np.heaviside(np.abs(w) -1,0)*np.heaviside(10000 -np.abs(w),0)*(alpha/abs(w))/(2*np.pi)
```
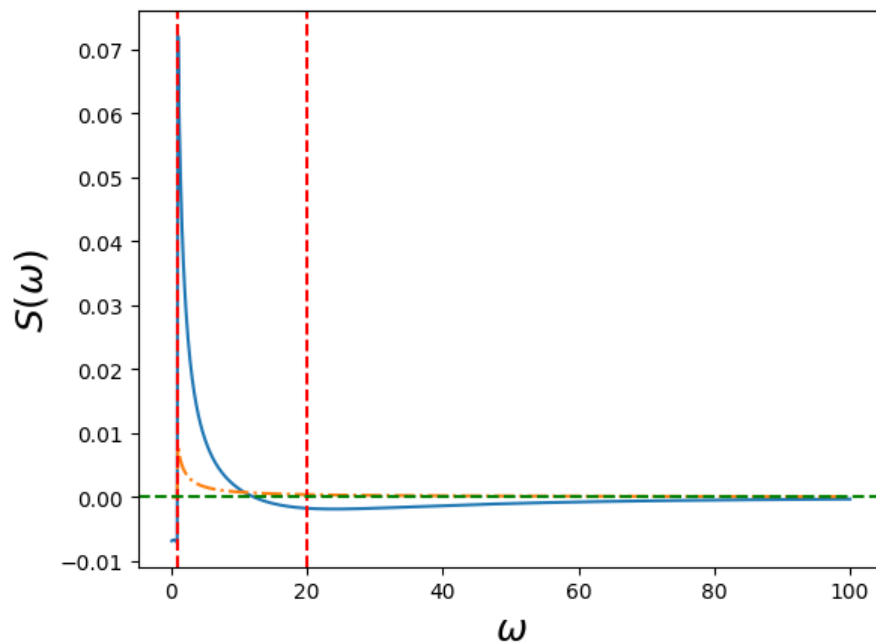
it does behave sort of like 1/f as wanted, Maybe using AAA would be better since overthere one actually fits the spectrum. Better support points here and lower/upper/guesses may also yield faster simulations. But for now maybe it's good

## 0.5  Classical Noise approach

To cross-check results from heom we take the approach outline in Costa-Filho et al. [2017]

Thanks again for the code neill !

## 0.6  Using AAA

AAA works great but to get great fits I need a lot of RAM, For some reason I can't connect to Pitchfork or Torch. So perhaps the results in the section below can be way better

The idea is that since one fits the spectrum directly. It may provide faster and better fits

```
/tmp/ipykernel_1381387/958776843.py:3: DeprecationWarning: Bitwise inversion '~' on bool is de
  if ~ (type(F)==np.array):
/tmp/ipykernel_1381387/958776843.py:38: ComplexWarning: Casting complex values to real discard
  R[J] = N / D
```
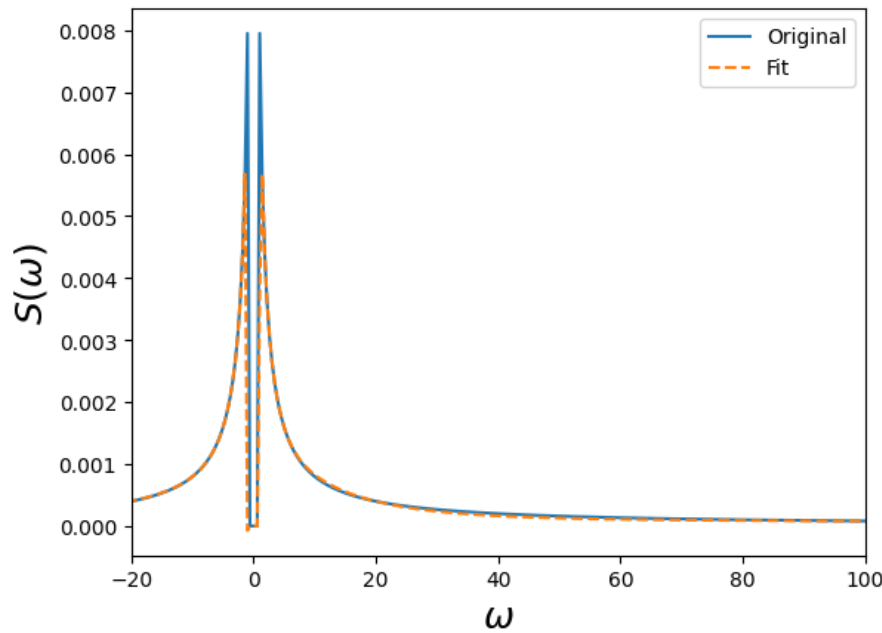
Way better $\frac{1}{f}$ behaviour except at the origin but still great. However, when I limit the number of exponents this is just as bad as the other fitting approach (but a lot faster). Could be better when tried with more RAM, specially since we only care about the spectrum here. Not the correlation function

```
/tmp/ipykernel_1381387/1893926815.py:2: RuntimeWarning: invalid value encountered in divide
  return (expi(1j*x)+expi( -1j*x))/2
/tmp/ipykernel_1381387/3153800274.py:2: RuntimeWarning: invalid value encountered in subtract
  cc=alpha*(ci(wh*t) -ci(wl*t))/(2)
```



Correlation function from Spectrum Fit
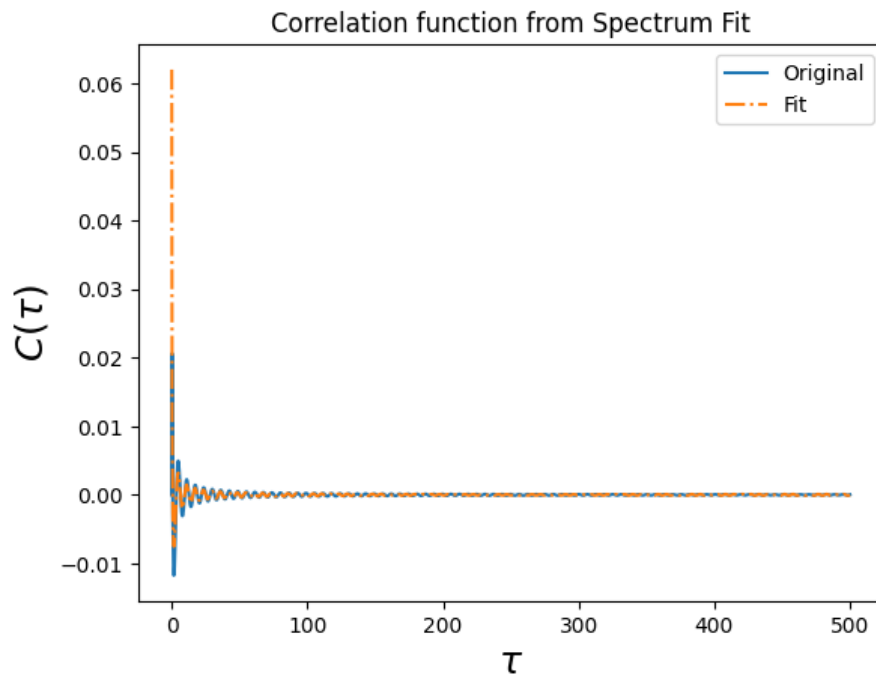
Problems here (Initially the list was longer)

1. The exponents generated from the fit are complex, They give rise to 1/F. But from this fitting I don't see how to make them real for now

Note this is fast even when the number of exponents is higher, probably due to the combine feature
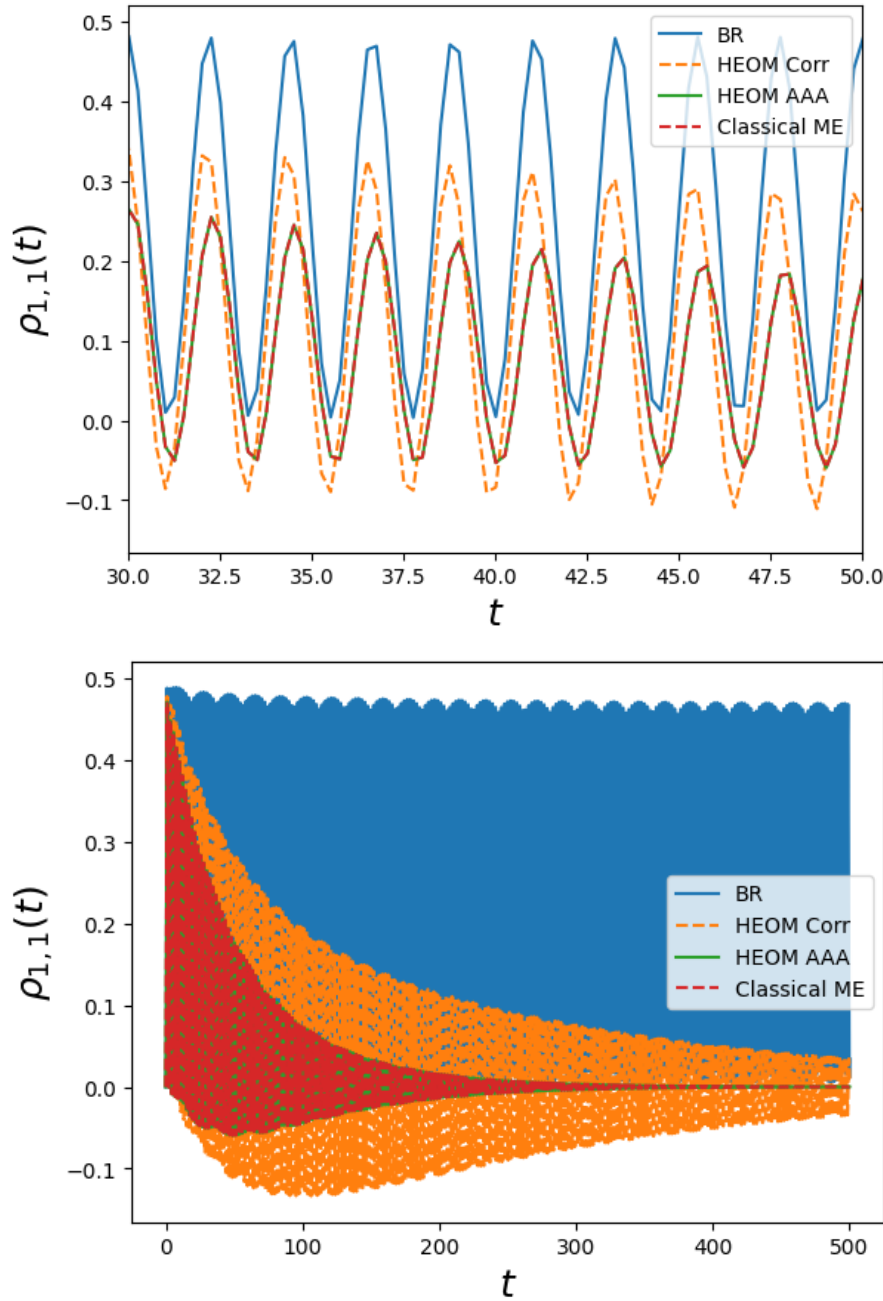
Well the classical bit is pretty good. AAA works nicely in this regime because we only care about the spectrum. And there's difficulties using the correlation function. Perhaps making it mire efficient would be the way to go. AAA is fast ~3s per simulation. I bet rescaling would make it faster :). Since there's at least one situation where AAA is better (without too much ram). We should perhaps implement spira which is a better version of AAA for this. While here I tried prony but did not like it :(

### 0.6.1   Using the spectral density

Probably not the best idea, because the number of exponents would grow drastically as this behaves like $T = 0$. I just want to check whether one can fit the corresponding power spectrum using

$$S(\omega) = 2J(\omega)\left(n(\omega) + 1\right) \tag{68}$$

At $T = 0$

$$S(\omega) = 2J(\omega) \tag{69}$$

so now one may see if it's easy to fit or whether it requires many exponents

```
Result of fitting The Spectral Density with 10 terms:

 Parameters|    lam    |   gamma   |   w0
 1          | 1.33e -02 | 6.70e -01 |1.37e+00
 2          | 7.23e -01 | 1.28e+01 |1.12e -01
 3          | -2.62e -01 | 1.10e+01 |1.21e+01
 4          | 9.10e -02 | 1.42e+00 |1.02e+00
 5          | -1.68e -01 | 6.88e+00 |6.97e+00
 6          | 7.23e -01 | 5.56e+01 |5.44e+01
 7          | -8.24e -02 | 1.17e+00 |1.06e+00
 8          | 4.00e -02 | 1.42e -01 |1.16e+00
 9          | 7.65e -02 | 1.10e -01 |1.14e+00
 10         | -1.15e -01 | 1.21e -01 |1.15e+00
```

```
A  normalized RMSE of  2.31e -05 was obtained for the The Spectral Density
 The current fit took  1.074890 seconds
```

Evemthough the fit is good the effective spectral density is not because it only has 5 exponents per term in the fit

```
/home/gerardo/qutip_gsoc_app/qutip/solver/heom/bofin_baths.py:425: RuntimeWarning: overflow en
  return (1 / (np.exp(w / self.T) - 1))
```
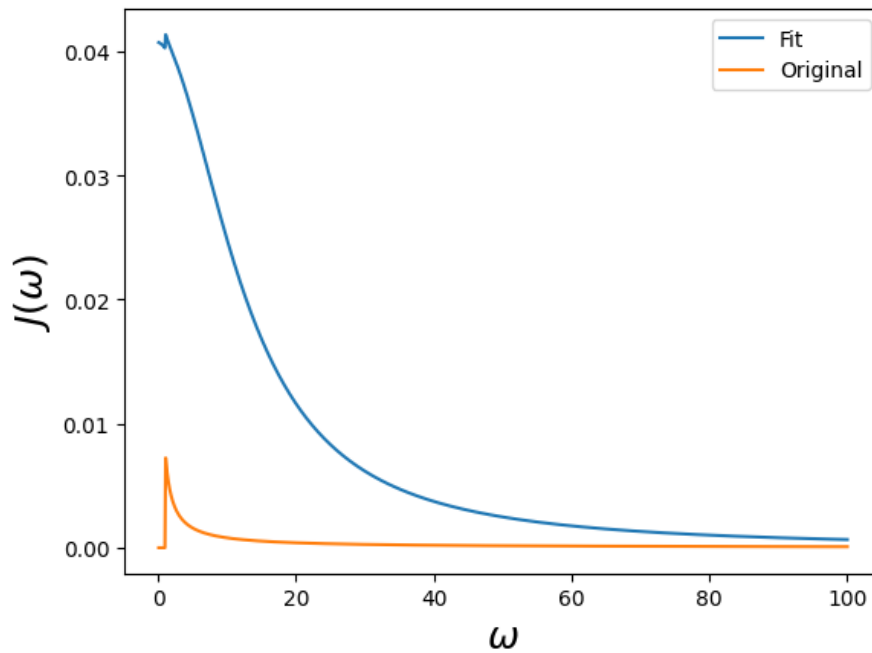


While the fit is ok, really hard to converge on the number of exponents as expected.

# Bibliography

J. I. Costa-Filho, R. B. B. Lima, R. R. Paiva, P. M. Soares, W. A. M. Morgado, R. L. Franco, and D. O. Soares-Pinto. Enabling quantum non-Markovian dynamics by injection of classical colored noise. *Physical Review A*, 95(5), 5 2017. ISSN 2469-9934. doi: 10.1103/physreva.95.052126. URL `http://dx.doi.org/10.1103/PhysRevA.95.052126`.

M. R. Hush, I. Lesanovsky, and J. P. Garrahan. Generic map from non-Lindblad to Lindblad master equations. *Physical Review A*, 91(3), 3 2015. ISSN 1094-1622. doi: 10.1103/physreva.91.032113. URL `http://dx.doi.org/10.1103/PhysRevA.91.032113`.