November 26, 2024

# Contents

## 0.1 Bloch-Redfield and Redfield Failing for degenerate Hamiltonians

**In The SYK model:** Inspired by the steady state of this dissipator being the maximally entangled state let us construct a qubit entangler out of it

The Hamiltonian in this example is then given by

```
Quantum object: dims=[[2, 2], [2, 2]], shape=(4, 4), type='oper', dtype=CSR, isherm=True
Qobj data =
[[ -1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0. -1.]]
```

While the coupling operator to the bath is simply

$Q = \sum_i a_i \psi_i$

Where each $a_i$ is a real number, randomly generated from a Gaussian

```
Quantum object: dims=[[2, 2], [2, 2]], shape=(4, 4), type='oper', dtype=CSR, isherm=True
Qobj data =
[[ 0.+0.j  1. -1.j  1. -1.j  0.+0.j]
 [ 1.+1.j  0.+0.j  0.+0.j -1.+1.j]
 [ 1.+1.j  0.+0.j  0.+0.j  1. -1.j]
 [ 0.+0.j -1. -1.j  1.+1.j  0.+0.j]]
```

We consider the initial state to be

```
Quantum object: dims=[[2, 2], [2, 2]], shape=(4, 4), type='oper', dtype=Dense, isherm=True
Qobj data =
[[0.5 0.5 0.  0. ]
 [0.5 0.5 0.  0. ]
 [0.  0.  0.  0. ]
 [0.  0.  0.  0. ]]
```

```
from qutip import concurrence
```

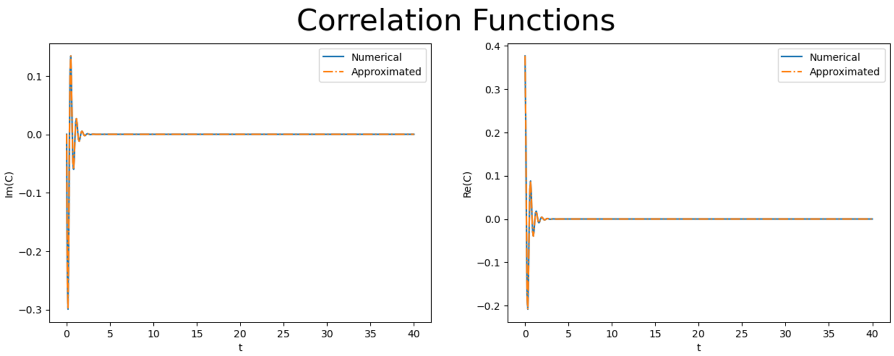
```
concurrence(rho0)
```

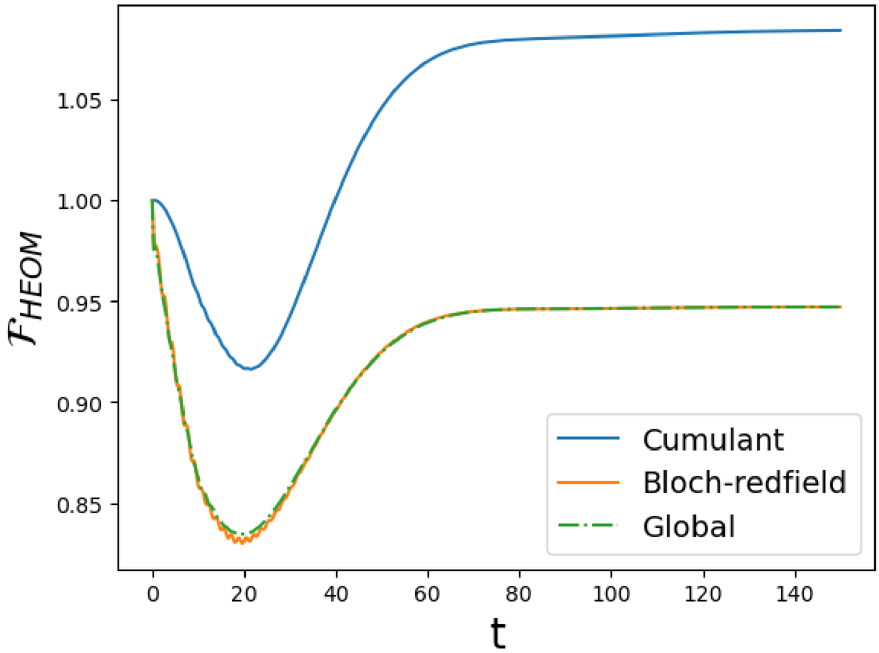
```
0.0
```

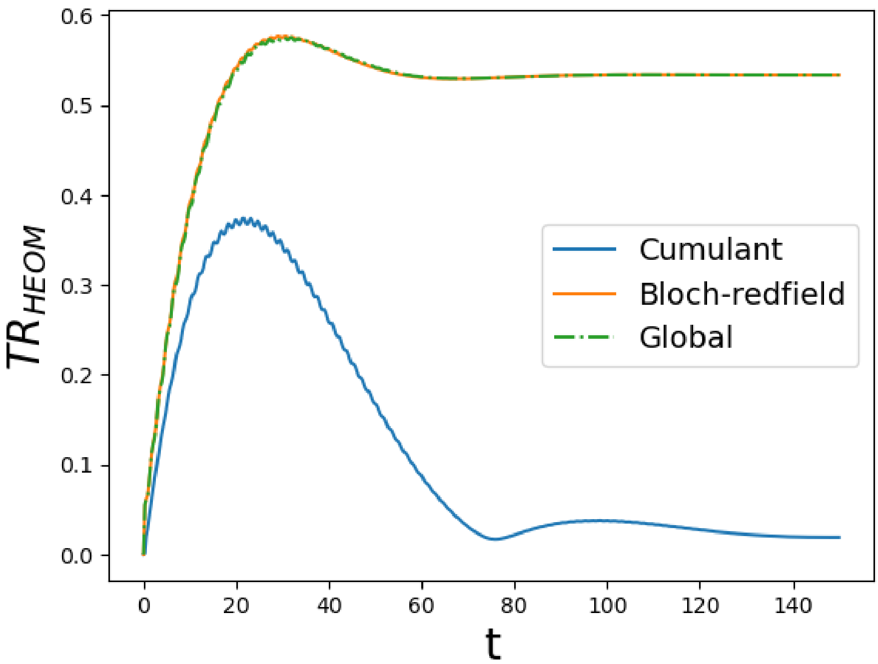
And consider an underdamped spectral density at zero temperature with $\gamma$ =5.0 , $\lambda$ =2.942830956382712 ,$\omega_0$ =10. After fitting the correlation function one obtains

```
Correlation function fit:

Result of fitting the real part of                                      |Result of fitting the im
the correlation function with 3 terms:                                  |of the correlation funct
                                                                        |
 Parameters|    a      |     b      |     c                             | Parameters|    a     |
 1         | -1.04e -01 | -9.53e+00 |3.27e -10                              | 1         | -4.47e
 2         | -7.02e+00 | -2.42e+00 |9.13e+00                              |
 3         | 7.50e+00 | -2.44e+00 |9.16e+00                               |A normalized RMSE of  1
                                                                        |of the correlation funct
A normalized RMSE of  4.40e -06 was obtained for the the real part of |
the correlation function.                                               |
The current fit took  11.647825 seconds.                                |The current fit took  0.
```




**Steady states (Not really steady but at $t = 50$)**

**HEOM**

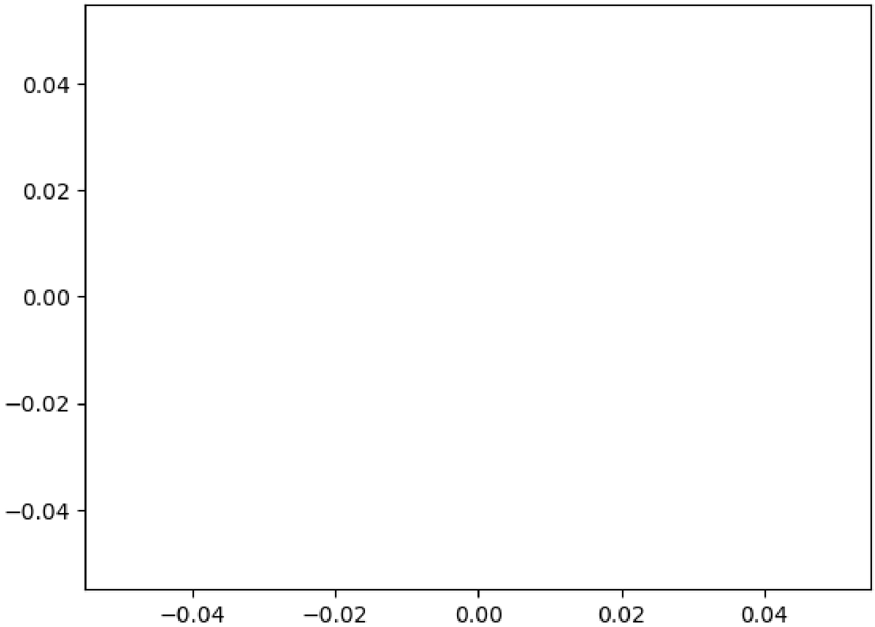
```
from qutip import concurrence
```

```
results_syk[0].states[ -1]
```

```
Quantum object: dims=[[2, 2], [2, 2]], shape=(4, 4), type='oper', dtype=Dense, isherm=False
Qobj data =
[[ 7.74773463e -01+3.22111203e -11j  4.17294148e -04 -3.66713072e -03j
   6.67324336e -07+2.39009067e -05j -1.07291624e -11+2.58247904e -01j]
 [ 4.17294145e -04+3.66713072e -03j -2.47585882e -02 -3.21988331e -11j
  -8.26277948e -03 -1.07419532e -11j  2.39009067e -05+6.67324376e -07j]
 [ 6.67324376e -07 -2.39009067e -05j -8.26277948e -03 -1.07415156e -11j
  -8.26277948e -03 -1.07403046e -11j  2.39009067e -05+6.67324376e -07j]
 [ 1.07293865e -11 -2.58247904e -01j  2.39009067e -05 -6.67324336e -07j
   2.39009067e -05 -6.67324336e -07j  2.58247904e -01+1.07279338e -11j]]
```

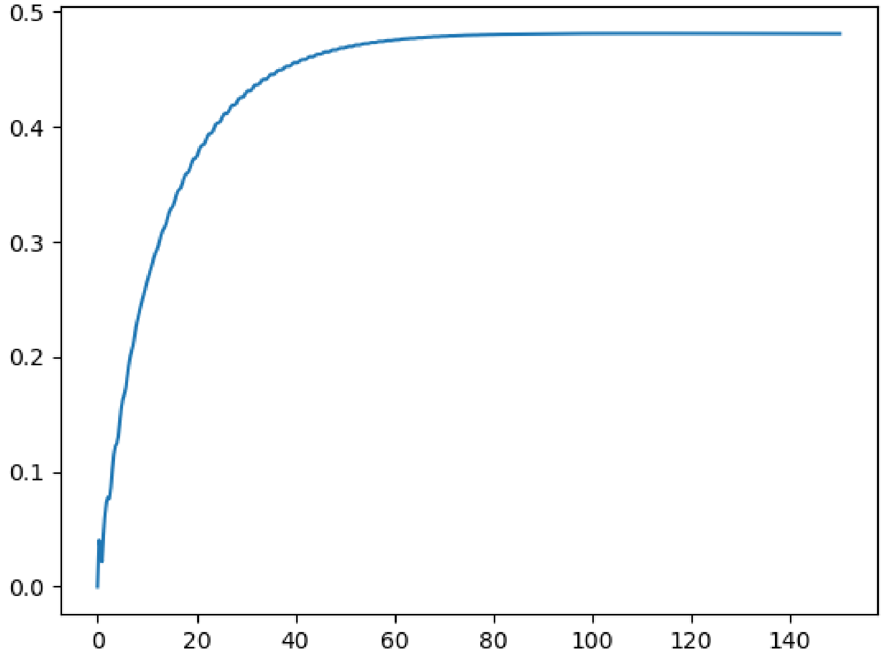
```
/tmp/ipykernel_7099/1673502128.py:1: RuntimeWarning: divide by zero encountered in divide
  plt.plot(lam*times/E01,[concurrence(i) for i in results_syk[0].states])
/tmp/ipykernel_7099/1673502128.py:1: RuntimeWarning: invalid value encountered in divide
  plt.plot(lam*times/E01,[concurrence(i) for i in results_syk[0].states])
```

```
[<matplotlib.lines.Line2D at 0x7eff921c4fe0>]
```



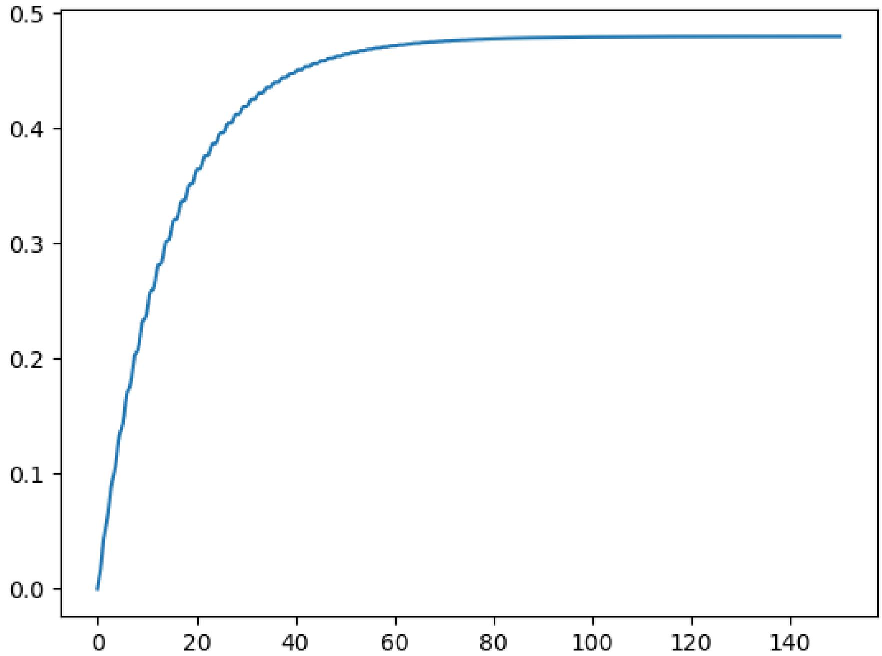

**Cumulant**

```
[<matplotlib.lines.Line2D at 0x7eff85b5f620>]
```




**Bloch-Redfield**

```
[<matplotlib.lines.Line2D at 0x7eff85bcd130>]
```




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.

**Every Equation is ok**

Consider a different coupling operator just the majorana fermion denoted by the index 0 coupled to the environment or in the notes notation ($b_0 = -i$,$b_1 = 0$)

```
Quantum object: dims=[[2, 2], [2, 2]], shape=(4, 4), type='oper', dtype=CSR, isherm=True
Qobj data =
```

```
[[0.+0.j 0. -1.j 0.+0.j 0.+0.j]
 [0.+1.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0. -1.j]
 [0.+0.j 0.+0.j 0.+1.j 0.+0.j]]

Correlation function fit:

Result of fitting the real part of                                      |Result of fitting the im
the correlation function with 2 terms:                                  |of the correlation funct
                                                                        |
 Parameters|    a     |    b     |    c                                 | Parameters|    a     |
 1         | 4.63e -01 | -2.56e+00 |9.66e+00                             | 1         | -4.47e -0
 2         | -8.49e -02 | -9.15e+00 |3.96e -08                             |
                                                                        |A normalized RMSE of  1.
A normalized RMSE of  5.49e -06 was obtained for the the real part of |of the correlation func
the correlation function.                                               |
The current fit took  0.426879 seconds.                                 |The current fit took  0.

============================
Solving HEOM
============================
10.1%. Run time:   3.18s. Est. time left: 00:00:00:28
20.2%. Run time:   6.56s. Est. time left: 00:00:00:25
30.3%. Run time:   9.96s. Est. time left: 00:00:00:22
40.4%. Run time:  13.29s. Est. time left: 00:00:00:19
50.5%. Run time:  16.75s. Est. time left: 00:00:00:16
60.6%. Run time:  19.54s. Est. time left: 00:00:00:12
70.7%. Run time:  22.15s. Est. time left: 00:00:00:09
80.8%. Run time:  24.71s. Est. time left: 00:00:00:05
90.9%. Run time:  27.26s. Est. time left: 00:00:00:02
100.0%. Run time:  29.73s. Est. time left: 00:00:00:00
Total run time:  29.73s
============================
HEOM Done
============================
============================
Solving Cumulant
============================

Calculating Integrals ...: 100%|| 4/4 [00:00<00:00, 3486.54it/s]
Calculating time independent matrices...: 100%|| 4/4 [00:00<00:00, 2278.89it/s]
Calculating time dependent generators: 100%|| 4/4 [00:00<00:00, 1265.54it/s]
Computing Exponential of Generators . . . .: 100%|| 100/100 [00:00<00:00, 391.30it/s]

============================
Cumulant Done
============================
============================
Solving Redfield
============================
Started integration and Generator Calculations
Finished integration and Generator Calculations
Computation Time:0.011269092559814453
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Started interpolation
Finished interpolation
Computation Time:0.004649162292480469
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Started Solving the differential equation
Finished Solving the differential equation
Computation Time:0.7239315509796143
============================
Redfield Done
============================
============================
Solving Bloch -Redfield
============================
============================
Bloch -Redfield Done
============================
============================

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
AttributeError                            Traceback (most recent call last)
Cell In[48], line 6, in trd(states, H, times)
      5 try:
 - - - -> 6     sdd=np.array([tracedist(i.states[j],states[0].states[j]) for j in range(len(ti
      7 except:

AttributeError: 'list' object has no attribute 'states'

During handling of the above exception, another exception occurred:

IndexError                                Traceback (most recent call last)
Cell In[58], line 1
 - - - -> 1 trd(results_syk2,H,times)

Cell In[48], line 8, in trd(states, H, times)
      6         sdd=np.array([tracedist(i.states[j],states[0].states[j]) for j in range(len(ti
      7     except:
 - - - -> 8         sdd=np.array([tracedist(i[j],states[0].states[j]) for j in range(len(times
      9     plt.plot(times,sdd,label=labels[k],linestyle=style[k])
     10 plt.legend(fontsize=14)

IndexError: list index out of range
```

```
from qutip import bell_state,concurrence
bell=bell_state("01")
bell_dens=bell*bell.dag()
bell_dens
```

```
results_syk[0].states[ -1]
```

```
fidelity(results_syk2[0].states[ -1],bell_dens)
```

I observe the same behaviour in many spin chain configurations for large $N$

### 0.1.1 Cumulant and global failing (What I didn't see before)

Another example ($b_0 = 1$,$b_1 = -i$)

#### Heom Steady

```
results_syk3[0].states[ -1]
```

#### Global Steady

```
results_syk3[1][ -1]
```

### 0.1.2 Different initial state

To make things more confusing let us consider a different initial state and see how things work, and then don't

But it does definitely seems to have something to do with ergodicity

```
N=2
state_list = [basis(2, 0)] + [ -1j*basis(2, 0)] * (N - 1) # change the initial state to be awa
state_list2 = [basis(2, 0)] + [basis(2, 0)] * (N - 1) # change the initial state to be away fr
state_list.reverse()
psi0 = tensor(state_list)
rho02=psi0*psi0.dag()
H.dims=rho0.dims
Q.dims=rho0.dims
times=np.linspace(0,50,100)
tfit=np.linspace(0, 80, 5000)
rho02
```

We use the first coupling operator

```
Q
```

```
bath=BosonicBath.from_environment(env,Q)
bath.T=0
# notice one mode is also a pretty good approximation
print(fitinfo['summary'])
results_syk4=solve_dynamics(H,Q,bath,bath,rho02)
```

```
trd(results_syk4,H,times)
```

```
plot_fidelities(results_syk4,H,times)
```

### 0.1.3 Finite Temperature

For Finite Temperature I find the exact same behaviour $T = 1$

We use the first initial state

#### Steady state from HEOM

#### Steady state from BR

```
bath=BosonicBath.from_environment(env,Q)
bath.T=1
# notice one mode is also a pretty good approximation
results_syk10=solve_dynamics(H,Q,bath,bath,rho02)
```

```
bath=BosonicBath.from_environment(env,Q2)
bath.T=1
# notice one mode is also a pretty good approximation
results_syk11=solve_dynamics(H,Q2,bath,bath,rho02)
```

```
bath=BosonicBath.from_environment(env,0.01*Q3)
bath.T=1
# notice one mode is also a pretty good approximation
results_syk12=solve_dynamics(H,0.01*Q3,bath,bath,rho02,depth=12)

results_syk12[0].states[ -1]

plot_fidelities(results_syk12,H,times)
```

### 0.1.4 Summary

The effect happens for both finite and zero temperature, it depends on the coupling operator and the initial state mostly

The Coupling operators are:

$Q_1$

$Q_2$

$Q_3$

For the initial state

**At** $T = 0$

**Finite Temperature**

**For the initial state**

**Finite Temperature**

## 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 Analytical check to make sure it's not, by solving the equation for the SYK(2) model symbolically. The Hamiltonian is given by

```
H = diag( -a,a,a, -a)#the times 20 is to rescale time
Eq(S('H'), H, evaluate=False)
```

```
Eq(H, Matrix([
[ -a, 0, 0,  0],
[ 0, a, 0,  0],
[ 0, 0, a,  0],
[ 0, 0, 0, -a]]))
```

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,             b1,   0],
[conjugate(b0),              0,              0, -b1],
[conjugate(b1),              0,              0,  b0],
[            0, -conjugate(b1), conjugate(b0),   0]]))
```

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

```
for key,value in jumps.items():
    display(Eq(S(f"A({key})"), value, evaluate=False))
```

```
Eq(A(2*a), Matrix([
[0,             b0,             b1, 0],
[0,              0,              0, 0],
[0,              0,              0, 0],
[0, -conjugate(b1), conjugate(b0), 0]]))
```

```
Eq(A( -2*a), Matrix([
[            0, 0, 0,   0],
[conjugate(b0), 0, 0, -b1],
[conjugate(b1), 0, 0,  b0],
[            0, 0, 0,   0]]))
```

### 0.2.1 Jump operator checks

The jump operators must satisfy

$$[H, A(\omega)] = -\omega A(\omega) \tag{1}$$

$$[H, A^{\dagger}(\omega)A(\omega)] = 0 \tag{2}$$

$$\sum_{w} A(\omega) = A \tag{3}$$

### 0.2.2 Constucting the Differential equations

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

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

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

$$\dot{\rho_S^I(t)t} = \sum_{\omega,\omega',\alpha,\beta} \gamma_{\beta,\alpha}(\omega,\omega') \left( S_\alpha(\omega')\rho_S^I(t)S_\beta^\dagger(\omega) - \frac{\{S_\beta^\dagger(\omega)S_\alpha(\omega'), \rho_S^I(t)\}}{2} \right) \tag{4}$$

As the sum goes on $(\omega,\omega')$ pairs I construct all combinations

```
(jumps[(2*a)]*jumps[ -2*a]*rho+ rho*jumps[(2*a)]*jumps[ -2*a]).expand()
```

```
Matrix([
[                         2*b0*rho_1*conjugate(b0) + 2*b1*rho_1*conjugate(b1),
[    b0*conjugate(b0)*conjugate(rho_2) + b1*conjugate(b1)*conjugate(rho_2),
[    b0*conjugate(b0)*conjugate(rho_3) + b1*conjugate(b1)*conjugate(rho_3),
[2*b0*conjugate(b0)*conjugate(rho_4) + 2*b1*conjugate(b1)*conjugate(rho_4), b0*conjugate(b0)*c
```

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

$$\Gamma_{\alpha,\beta}(\omega,t) = \int_0^t ds e^{i\omega s}\langle B_\alpha^\dagger(t)B_\beta^\dagger(t-s)\rangle_B \tag{5}$$

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

$$\Gamma_{\alpha,\beta}(\omega,\omega',t) = \int_0^\infty ds e^{i\omega s}\langle B(s)B(0)\rangle_B = \Gamma_{\alpha,\beta}(\omega) \tag{6}$$

Where $\Gamma_{\alpha,\beta}(\omega)$ is the power spectrum
The decay rate in the redfield equation is given by
$\gamma(\omega,\omega',t) = e^{i(\omega-\omega')t}(\Gamma(\omega') + \overline{\Gamma(\omega)})$

```
gmm,gpp=symbols("\\gamma_{ - -} \\gamma_{++}",real=True)
gpm=symbols("\\gamma_{+ -}")
```

Assuming an underdamped spectral density

$$J(\omega) = \frac{\lambda^2\gamma\omega}{((\omega_0^2-\omega^2)^2+\gamma^2\omega^2)} \tag{7}$$

Then the generator of the dynamics is given by

```
Eq(S('G'), gene, evaluate=False)
```

```
Eq(G, Matrix([
[
[                        -0.5*\gamma_{++}*(b0*conjugate(b0) + b1*conjugate(b1))*conjugate(rho_2)
[                        -0.5*\gamma_{++}*(b0*conjugate(b0) + b1*conjugate(b1))*conjugate(rho_3)
[\gamma_{++}*((rho_10*conjugate(b0) - rho_6*conjugate(b1))*conjugate(b0) + (rho_11*conjugate(b
```

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), \gamma_{++}*(b0*rho_7(t)*conjugate(b1) + b1*rho_10(t)*conjugate(b0

Eq(Derivative(rho_2(t), t), \gamma_{++}*( -rho_2(t)*Abs(b0)**2/2 - rho_2(t)*Abs(b1)**2/2) + \g

Eq(Derivative(rho_3(t), t), \gamma_{++}*( -rho_3(t)*Abs(b0)**2/2 - rho_3(t)*Abs(b1)**2/2) + \g

Eq(Derivative(rho_4(t), t), \gamma_{++}*(b0**2*rho_7(t) + b0*b1*rho_11(t) - b0*b1*rho_6(t) - b

Eq(Derivative(rho_5(t), t), \gamma_{++}*( -rho_5(t)*Abs(b0)**2/2 - rho_5(t)*Abs(b1)**2/2) + \g

Eq(Derivative(rho_6(t), t), \gamma_{++}*( -rho_6(t)*Abs(b0)**2 - rho_6(t)*Abs(b1)**2) + \gamma

Eq(Derivative(rho_7(t), t), \gamma_{++}*( -rho_7(t)*Abs(b0)**2 - rho_7(t)*Abs(b1)**2) + \gamma

Eq(Derivative(rho_8(t), t), \gamma_{++}*( -rho_8(t)*Abs(b0)**2/2 - rho_8(t)*Abs(b1)**2/2) + \g

Eq(Derivative(rho_9(t), t), \gamma_{++}*( -rho_9(t)*Abs(b0)**2/2 - rho_9(t)*Abs(b1)**2/2) + \g

Eq(Derivative(rho_10(t), t), \gamma_{++}*( -rho_10(t)*Abs(b0)**2 - rho_10(t)*Abs(b1)**2) + \ga

Eq(Derivative(rho_11(t), t), \gamma_{++}*( -rho_11(t)*Abs(b0)**2 - rho_11(t)*Abs(b1)**2) + \ga

Eq(Derivative(rho_12(t), t), \gamma_{++}*( -rho_12(t)*Abs(b0)**2/2 - rho_12(t)*Abs(b1)**2/2) +

Eq(Derivative(rho_13(t), t), \gamma_{++}*(rho_10(t)*conjugate(b0)**2 + rho_11(t)*conjugate(b0)

Eq(Derivative(rho_14(t), t), \gamma_{++}*( -rho_14(t)*Abs(b0)**2/2 - rho_14(t)*Abs(b1)**2/2) +

Eq(Derivative(rho_15(t), t), \gamma_{++}*( -rho_15(t)*Abs(b0)**2/2 - rho_15(t)*Abs(b1)**2/2) +

Eq(Derivative(rho_16(t), t), \gamma_{++}*( -b0*rho_7(t)*conjugate(b1) - b1*rho_10(t)*conjugate
```

Then We make $\gamma_{--} = 0$ as we consider zero temperature $T = 0$

```
(eqs[0].rhs + eqs[5].rhs + eqs[10].rhs + eqs[15].rhs ).subs(gmm,0).expand()
```

```
0
```

```
for i in eqs:
    display(i.subs(gmm,0))
```

```
Eq(Derivative(rho_1(t), t), \gamma_{++}*(b0*rho_7(t)*conjugate(b1) + b1*rho_10(t)*conjugate(b0

Eq(Derivative(rho_2(t), t), \gamma_{++}*( -rho_2(t)*Abs(b0)**2/2 - rho_2(t)*Abs(b1)**2/2) + 8*

Eq(Derivative(rho_3(t), t), \gamma_{++}*( -rho_3(t)*Abs(b0)**2/2 - rho_3(t)*Abs(b1)**2/2) + 8*

Eq(Derivative(rho_4(t), t), \gamma_{++}*(b0**2*rho_7(t) + b0*b1*rho_11(t) - b0*b1*rho_6(t) - b

Eq(Derivative(rho_5(t), t), \gamma_{++}*( -rho_5(t)*Abs(b0)**2/2 - rho_5(t)*Abs(b1)**2/2) + \g

Eq(Derivative(rho_6(t), t), \gamma_{++}*( -rho_6(t)*Abs(b0)**2 - rho_6(t)*Abs(b1)**2))

Eq(Derivative(rho_7(t), t), \gamma_{++}*( -rho_7(t)*Abs(b0)**2 - rho_7(t)*Abs(b1)**2))

Eq(Derivative(rho_8(t), t), \gamma_{++}*( -rho_8(t)*Abs(b0)**2/2 - rho_8(t)*Abs(b1)**2/2) + \g
```

```
Eq(Derivative(rho_9(t), t), \gamma_{++}*( -rho_9(t)*Abs(b0)**2/2 - rho_9(t)*Abs(b1)**2/2) + \g

Eq(Derivative(rho_10(t), t), \gamma_{++}*( -rho_10(t)*Abs(b0)**2 - rho_10(t)*Abs(b1)**2))

Eq(Derivative(rho_11(t), t), \gamma_{++}*( -rho_11(t)*Abs(b0)**2 - rho_11(t)*Abs(b1)**2))

Eq(Derivative(rho_12(t), t), \gamma_{++}*( -rho_12(t)*Abs(b0)**2/2 - rho_12(t)*Abs(b1)**2/2) +

Eq(Derivative(rho_13(t), t), \gamma_{++}*(rho_10(t)*conjugate(b0)**2 + rho_11(t)*conjugate(b0)

Eq(Derivative(rho_14(t), t), \gamma_{++}*( -rho_14(t)*Abs(b0)**2/2 - rho_14(t)*Abs(b1)**2/2) +

Eq(Derivative(rho_15(t), t), \gamma_{++}*( -rho_15(t)*Abs(b0)**2/2 - rho_15(t)*Abs(b1)**2/2) +

Eq(Derivative(rho_16(t), t), \gamma_{++}*( -b0*rho_7(t)*conjugate(b1) - b1*rho_10(t)*conjugate
```

Then at $T = 0$, and in the Schrodinger picture (So that there is no phase and $\gamma$s are real). We have $\gamma_{+-} = \gamma_{-+} = \frac{\gamma_{++}}{2}$

```
for i in eqs:
    display(i.subs(gmm,0).subs(gpm,gpp/2).subs(conjugate(gpm),gpp/2))
```

```
Eq(Derivative(rho_1(t), t), \gamma_{++}*(b0*rho_7(t)*conjugate(b1) + b1*rho_10(t)*conjugate(b0

Eq(Derivative(rho_2(t), t), \gamma_{++}*( -rho_2(t)*Abs(b0)**2/2 - rho_2(t)*Abs(b1)**2/2) + \g

Eq(Derivative(rho_3(t), t), \gamma_{++}*( -rho_3(t)*Abs(b0)**2/2 - rho_3(t)*Abs(b1)**2/2) + \g

Eq(Derivative(rho_4(t), t), \gamma_{++}*(b0**2*rho_7(t) + b0*b1*rho_11(t) - b0*b1*rho_6(t) - b

Eq(Derivative(rho_5(t), t), \gamma_{++}*( -rho_5(t)*Abs(b0)**2/2 - rho_5(t)*Abs(b1)**2/2) + \g

Eq(Derivative(rho_6(t), t), \gamma_{++}*( -rho_6(t)*Abs(b0)**2 - rho_6(t)*Abs(b1)**2))

Eq(Derivative(rho_7(t), t), \gamma_{++}*( -rho_7(t)*Abs(b0)**2 - rho_7(t)*Abs(b1)**2))

Eq(Derivative(rho_8(t), t), \gamma_{++}*( -rho_8(t)*Abs(b0)**2/2 - rho_8(t)*Abs(b1)**2/2) + \g

Eq(Derivative(rho_9(t), t), \gamma_{++}*( -rho_9(t)*Abs(b0)**2/2 - rho_9(t)*Abs(b1)**2/2) + \g

Eq(Derivative(rho_10(t), t), \gamma_{++}*( -rho_10(t)*Abs(b0)**2 - rho_10(t)*Abs(b1)**2))

Eq(Derivative(rho_11(t), t), \gamma_{++}*( -rho_11(t)*Abs(b0)**2 - rho_11(t)*Abs(b1)**2))

Eq(Derivative(rho_12(t), t), \gamma_{++}*( -rho_12(t)*Abs(b0)**2/2 - rho_12(t)*Abs(b1)**2/2) +

Eq(Derivative(rho_13(t), t), \gamma_{++}*(rho_10(t)*conjugate(b0)**2 + rho_11(t)*conjugate(b0)

Eq(Derivative(rho_14(t), t), \gamma_{++}*( -rho_14(t)*Abs(b0)**2/2 - rho_14(t)*Abs(b1)**2/2) +

Eq(Derivative(rho_15(t), t), \gamma_{++}*( -rho_15(t)*Abs(b0)**2/2 - rho_15(t)*Abs(b1)**2/2) +

Eq(Derivative(rho_16(t), t), \gamma_{++}*( -b0*rho_7(t)*conjugate(b1) - b1*rho_10(t)*conjugate
```

```
eqs2=[i.subs(gmm,0).subs(gpm,gpp/2).subs(conjugate(gpm),gpp/2) for i in eqs]
```

At this point, we are still preserving the trace

```
Eq(eqs2[0].lhs + eqs2[5].lhs + eqs2[10].lhs + eqs2[15].lhs,(eqs2[0].rhs + eqs2[5].rhs + eqs2[1
```

```
Eq(Derivative(rho_1(t), t) + Derivative(rho_11(t), t) + Derivative(rho_16(t), t) + Derivative(
```

I first solve the really easy ones: $(\rho_{10}, \rho_{11}, \rho_7, \rho_6)$

```
sols_easy=dsolve([eqs2[i] for i in (5,6,9,10)])
sols_subs={i.lhs:i.rhs for i in sols_easy}

eqs3_full=[i.subs(sols_subs) for i in eqs2]
eqs3=[i.simplify() for i in eqs3_full if i.simplify()!=True ]
for i in eqs3:
    display(i)
```

```
Eq(Derivative(rho_1(t), t), \gamma_{++}*(C1*Abs(b0)**2 + C2*b0*conjugate(b1) + C3*b1*conjugate

Eq(Derivative(rho_2(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_2(t)/2 + \gamma_{++}*(b

Eq(Derivative(rho_3(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_3(t)/2 + \gamma_{++}*(b

Eq(Derivative(rho_4(t), t), -\gamma_{++}*(C1*b0*b1 - C2*b0**2 + C3*b1**2 - C4*b0*b1)*exp( -\ga

Eq(Derivative(rho_5(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_5(t)/2 - \gamma_{++}*(b

Eq(Derivative(rho_8(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_8(t)/2 - \gamma_{++}*(b

Eq(Derivative(rho_9(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_9(t)/2 + \gamma_{++}*(b

Eq(Derivative(rho_12(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_12(t)/2 + \gamma_{++}*

Eq(Derivative(rho_13(t), t), -\gamma_{++}*(C1*conjugate(b0)*conjugate(b1) + C2*conjugate(b1)**

Eq(Derivative(rho_14(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_14(t)/2 - \gamma_{++}*

Eq(Derivative(rho_15(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_15(t)/2 + \gamma_{++}*

Eq(Derivative(rho_16(t), t), \gamma_{++}*(C1*Abs(b1)**2 - C2*b0*conjugate(b1) - C3*b1*conjugat
```

The trace is still preserved as expected

```
Eq(eqs2[0].lhs + eqs2[5].lhs + eqs2[10].lhs + eqs2[15].lhs,(eqs3_full[0].rhs + eqs3_full[5].rh
```

```
Eq(Derivative(rho_1(t), t) + Derivative(rho_11(t), t) + Derivative(rho_16(t), t) + Derivative(
```

```
sols_easy2=dsolve([eqs3[i] for i in (0,3,8,11)])
sols_subs2={i.lhs:i.rhs for i in sols_easy2}

eqs4_full=[i.subs(sols_subs).subs(sols_subs2) for i in eqs2]
eqs4=[i.simplify() for i in eqs4_full if i.simplify()!=True ]
for i in eqs4:
    display(i)
```

```
Eq(Derivative(rho_2(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_2(t)/2 + \gamma_{++}*(b

Eq(Derivative(rho_3(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_3(t)/2 + \gamma_{++}*(b

Eq(Derivative(rho_5(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_5(t)/2 - \gamma_{++}*(b

Eq(Derivative(rho_8(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_8(t)/2 - \gamma_{++}*(b
```

```
Eq(Derivative(rho_9(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_9(t)/2 + \gamma_{++}*(b
```

```
Eq(Derivative(rho_12(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_12(t)/2 + \gamma_{++}*
```

```
Eq(Derivative(rho_14(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_14(t)/2 - \gamma_{++}*
```

```
Eq(Derivative(rho_15(t), t), -\gamma_{++}*(Abs(b0)**2 + Abs(b1)**2)*rho_15(t)/2 + \gamma_{++}*
```

The set of equation does preserve the trace, we made a mistake in person

```
Eq(eqs2[0].lhs + eqs2[5].lhs + eqs2[10].lhs + eqs2[15].lhs,(eqs4_full[0].rhs + eqs4_full[5].rh
```

```
Eq(Derivative(rho_1(t), t) + Derivative(rho_11(t), t) + Derivative(rho_16(t), t) + Derivative(
```

I cannot solve analytically for the general case due to the degree of the polynomial, I can then use specific values of $b_0$ and $b_1$ so that I can express it as radicals ( I did not try mathematica though but even If I do it would be a root object). But I can solve the diagonal (But that does not help because the diagonal works)

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

```
Eq(rho_6(t), C1*exp( -t*(\gamma_{++}*Abs(b0)**2 + \gamma_{++}*Abs(b1)**2)))
```

```
Eq(rho_7(t), C2*exp( -t*(\gamma_{++}*Abs(b0)**2 + \gamma_{++}*Abs(b1)**2)))
```

```
Eq(rho_10(t), C3*exp( -t*(\gamma_{++}*Abs(b0)**2 + \gamma_{++}*Abs(b1)**2)))
```

```
Eq(rho_11(t), C4*exp( -t*(\gamma_{++}*Abs(b0)**2 + \gamma_{++}*Abs(b1)**2)))
```

```
for i in sols_easy2:
    display(i)
```

```
Eq(rho_1(t), C5 - (C1*Abs(b0)**2 + C2*b0*conjugate(b1) + C3*b1*conjugate(b0) + C4*Abs(b1)**2)*
```

```
Eq(rho_4(t), C6 + (C1*b0*b1 - C2*b0**2 + C3*b1**2 - C4*b0*b1)*exp( -t*(\gamma_{++}*Abs(b0)**2
```

```
Eq(rho_13(t), C7 + (C1*conjugate(b0)*conjugate(b1) + C2*conjugate(b1)**2 - C3*conjugate(b0)**2
```

```
Eq(rho_16(t), C8 - (C1*Abs(b1)**2 - C2*b0*conjugate(b1) - C3*b1*conjugate(b0) + C4*Abs(b0)**2)
```

And No contradiction because the trace is preserved, but the time dependent trace does have an issue because

```
trace_pres=Eq(sols_easy2[0].lhs + sols_easy[0].lhs +sols_easy[ -1].lhs+sols_easy2[ -1].lhs ,so
```

```
trace_pres
```

```
Eq(rho_1(t) + rho_11(t) + rho_16(t) + rho_6(t), C1*exp( -t*(\gamma_{++}*Abs(b0)**2 + \gamma_{+
```

Which simplifies to

```
Eq(trace_pres.lhs,trace_pres.rhs.simplify())
```

```
Eq(rho_1(t) + rho_11(t) + rho_16(t) + rho_6(t), C5 + C8)
```

Which is odd, and perhaps inconsistent

```
final=dict((v, k) for k, v in to_funcs.items())
```

So I then try for the steady state

From those equations obviously $\rho_{10}, \rho_6, \rho_{11} = 0$, Substitution then gives us no information on the diagonal, but no inconsistency as we puzzled about before

```
c= symbols('c0:8')

ss2=[Eq(i.rhs,0).subs(final).subs(rho[2,1],0).subs(rho[1,1],0).subs(rho[2,2],0)  for i  in eqs
ss2=[i.simplify() for i in ss2 if i.simplify()!=True ]
new_vars={gpp/2 * (abs(b[0])**2 + abs(b[1])**2 ):c[0]}
for i in ss2:
    display(i.subs(new_vars))

Eq(\gamma_{++}*(b0**2*conjugate(rho_2) + b0*b1*conjugate(rho_3) - b0*rho_8*conjugate(b1) - rho

Eq(\gamma_{++}*(b0*b1*conjugate(rho_2) + b1**2*conjugate(rho_3) + b1*rho_12*conjugate(b0) + rh

Eq(\gamma_{++}*(b1*conjugate(b0)*conjugate(rho_8) - rho_2*conjugate(b0)**2 - rho_3*conjugate(b

Eq(\gamma_{++}*(b0*b1*conjugate(rho_12) - b1**2*conjugate(rho_8) + b1*rho_2*conjugate(b0) - rh

Eq( -\gamma_{++}*(b0*conjugate(b1)*conjugate(rho_12) + rho_2*conjugate(b0)*conjugate(b1) + rho

Eq( -\gamma_{++}*(b0**2*conjugate(rho_12) - b0*b1*conjugate(rho_8) + b0*rho_3*conjugate(b1) -

Eq(\gamma_{++}*(b0*conjugate(b1)*conjugate(rho_2) + rho_12*conjugate(b0)*conjugate(b1) - rho_8

Eq(\gamma_{++}*(b1*conjugate(b0)*conjugate(rho_3) + rho_12*conjugate(b0)**2 - rho_8*conjugate(
```

The problem here is that I have more unknowns that solutions, for exmaple equations 1 and 3 are just one equation, same for each pair so I have 4 equations but I have 8 variables the real and the imaginary part of $\rho_2, \rho_3, \rho_8, \rho_{12}$, the diagonal is also undetermined from this set of equations

```
from sympy.physics.quantum import Commutator

rho = symbols('rho_1:17')
rho = Matrix(
    [rho[0: 4],
     rho[4: 8],
     rho[8: 12],
     rho[12:]])
rho=rho.subs({rho[4]: conjugate(rho[1]), rho[8]: conjugate(rho[2]),
          rho[12]:conjugate(rho[3]),rho[6]:conjugate(rho[9]),
          rho[13]: conjugate(rho[7]), rho[14]:conjugate(rho[11])})
rho

Matrix([
[           rho_1,            rho_2,             rho_3,  rho_4],
[conjugate(rho_2),            rho_6, conjugate(rho_10),  rho_8],
[conjugate(rho_3),           rho_10,            rho_11, rho_12],
[conjugate(rho_4), conjugate(rho_8), conjugate(rho_12), rho_16]])

rhoo,Hh=symbols("rho H",commutative=False)
rhoo

rho
```

```
Eq(Commutator(Hh,rhoo),H*rho -rho*H,evaluate=False)
```

```
Eq([H,rho], Matrix([
[                    0,          -2*a*rho_2,           -2*a*rho_3,          0],
[2*a*conjugate(rho_2),                    0,                    0,  2*a*rho_8],
[2*a*conjugate(rho_3),                    0,                    0, 2*a*rho_12],
[                    0, -2*a*conjugate(rho_8), -2*a*conjugate(rho_12),         0]]))
```

For an operator to commute with $H$ $\rho_2 = \rho_3 = \rho_8 = \rho_{12} = 0$

**Tip**
Because the density matris is hermitian this means we also have $\rho_5 = \rho_9 = \rho_{15} = \rho_{14} = 0$

```
rho=rho.subs({rho[0,1]:0,rho[0,2]:0,rho[1,3]:0,rho[2,3]:0})
rho
```

```
Matrix([
[           rho_1,      0,                 0,  rho_4],
[               0,  rho_6, conjugate(rho_10),      0],
[               0, rho_10,            rho_11,      0],
[conjugate(rho_4),      0,                 0, rho_16]])
```

```
A=jumps[2*a]
A
```

```
Matrix([
[0,             b0,            b1, 0],
[0,              0,             0, 0],
[0,              0,             0, 0],
[0, -conjugate(b1), conjugate(b0), 0]])
```

```
A*rho -rho*A
```

```
Matrix([
[0,                                   -b0*rho_1 + b0*rho_6 + b1*rho_10 + rho_4*conjugate(b1),
[0,                                                                                        0,
[0,                                                                                        0,
[0, -b0*conjugate(rho_4) + rho_10*conjugate(b0) + rho_16*conjugate(b1) - rho_6*conjugate(b1),
```

So an operator that commutes with both needs to satisy

```
eqs_commutant=[Eq(i,0) for i in flatten(A*rho -rho*A) if i!=0]
for i in eqs_commutant:
    display(i)
```

```
Eq( -b0*rho_1 + b0*rho_6 + b1*rho_10 + rho_4*conjugate(b1), 0)

Eq(b0*conjugate(rho_10) - b1*rho_1 + b1*rho_11 - rho_4*conjugate(b0), 0)

Eq( -b0*conjugate(rho_4) + rho_10*conjugate(b0) + rho_16*conjugate(b1) - rho_6*conjugate(b1),

Eq( -b1*conjugate(rho_4) + rho_11*conjugate(b0) - rho_16*conjugate(b0) - conjugate(b1)*conjuga
```

I think this has infinitely many solutions as well!