TimeEvolvingMPO

Release 0.1.2

TEMPO Collaboration

Mar 08, 2021

INTRODUCTION

A Python 3 package to efficiently compute non-Markovian open quantum systems.

Github	https://github.com/tempoCollaboration/TimeEvolvingMPO
Documentation	https://TimeEvolvingMPO.readthedocs.io
PyPI	https://pypi.org/project/time-evolving-mpo/
Tutorial	launch 

CHAPTER

ONE

INTRODUCTION

This open source project aims to facilitate versatile numerical tools to efficiently compute the dynamics of quantum systems that are possibly strongly coupled to a structured environment. It allows to conveniently apply the so called time evolving matrix product operator method (TEMPO) [1], as well as the process tensor TEMPO method (PT-TEMPO) [2].

- **[1]** A. Strathearn, P. Kirton, D. Kilda, J. Keeling and B. W. Lovett, *Efficient non-Markovian quantum dynamics using time-evolving matrix product operators*, Nat. Commun. 9, 3322 (2018).
- **[2]** G. E. Fux, E. Butler, P. R. Eastham, B. W. Lovett, and J. Keeling, *Efficient exploration of Hamiltonian parameter space for optimal control of non-Markovian open quantum systems*, arXiv2101.03071 (2021).

CHAPTER

TWO

INSTALLATION

2.1 How To Install

- Make sure you have *python3.6* or higher installed

```
$ python3 --version
Python 3.6.9
```

- Make sure you have *pip3* version 20.0 or higher installed

```
$ python3 -m pip --version
pip 20.0.2 from /home/gefux/.local/lib/python3.6/site-packages/pip (python 3.6)
```

- Install TimeEvolvingMPO via pip

```
$ python3 -m pip install time_evolving_mpo
```

2.2 Test Installation

Open a interactive python3 session and type:

```
>>> import time_evolving_mpo as tempo
>>> tempo.say_hi()
```

This should give you the following message:

```
'0.1.0'
```

2.3 Uninstall

Uninstall TimeEvolvingMPO with pip:

```
$ python3 -m pip uninstall time_evolving_mpo
```

CHAPTER

THREE

TUTORIAL 01 - QUICKSTART

A quick introduction on how to use the TimeEvolvingMPO package to compute the dynamics of a quantum system that is possibly strongly coupled to a structured environment. It illustrates this by applying the TEMPO method to the strongly coupled spin boson model.

You can follow this tutorial using any of these options:

- launch binder 

(runs in browser),
- download the `jupyter file`,
- download the `python3 file`,
- read through it and code along below.

First, let's import TimeEvolvingMPO and some other packages we are going to use

```
import sys
sys.path.insert(0,'..')

import time_evolving_mpo as tempo
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

and check what version of tempo we are using.

```
tempo.__version__
```

```
'0.1.0'
```

Contents:

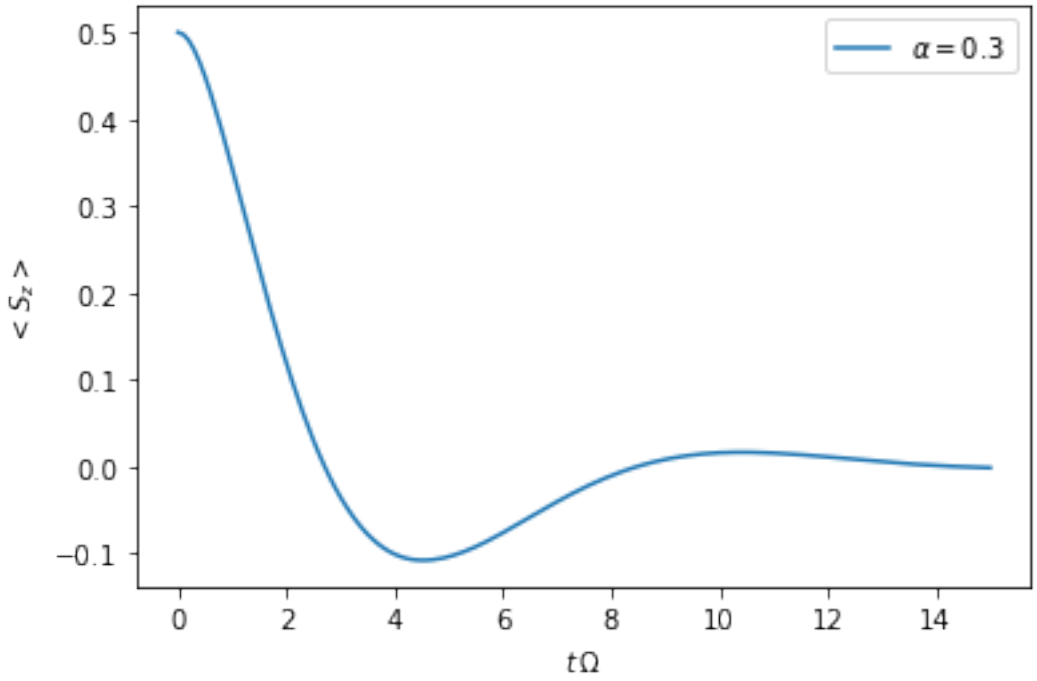
3.1 Example A - The Spin Boson Model

As a first example let's try to reconstruct one of the lines in figure 2a of [Strathearn2018] (Nat. Comm. 9, 3322 (2018) / arXiv:1711.09641v3). In this example we compute the time evolution of a spin which is strongly coupled to an ohmic bath (spin-boson model). Before we go through this step by step below, let's have a brief look at the script that will do the job - just to have an idea where we are going:

```
Omega = 1.0
omega_cutoff = 5.0
alpha = 0.3

system = tempo.System(0.5 * Omega * tempo.operators.sigma("x"))
correlations = tempo.PowerLawSD(alpha=alpha,
                                zeta=1,
                                cutoff=omega_cutoff,
                                cutoff_type='exponential',
                                max_correlation_time=8.0)
bath = tempo.Bath(0.5 * tempo.operators.sigma("z"), correlations)
tempo_parameters = tempo.TempoParameters(dt=0.1, dkmax=30, epsrel=10**(-4))

dynamics = tempo.tempo_compute(system=system,
                               bath=bath,
                               initial_state=tempo.operators.spin_dm("up"),
                               start_time=0.0,
                               end_time=15.0,
                               parameters=tempo_parameters)
t, s_z = dynamics.expectations(0.5*tempo.operators.sigma("z"), real=True)

plt.plot(t, s_z, label=r'$\alpha=0.3$')
plt.xlabel(r'$t\,\Omega$')
plt.ylabel(r'$<S_z>$')
plt.legend()
```

```
100.0%  150 of  150 [########################################] 00:00:10
Elapsed time: 10.7s
```

```
<matplotlib.legend.Legend at 0x7fbcfc6cf4e0>
```



3.1.1 A.1: The Model and its Parameters

We consider a system Hamiltonian

$$H_S = \frac{\Omega}{2}\hat{\sigma}_x,$$

a bath Hamiltonian

$$H_B = \sum_k \omega_k \hat{b}_k^\dagger \hat{b}_k,$$

and an interaction Hamiltonian

$$H_I = \frac{1}{2}\hat{\sigma}_z \sum_k \left(g_k \hat{b}_k^\dagger + g_k^* \hat{b}_k\right),$$

where $\hat{\sigma}_i$ are the Pauli operators, and the g_k and ω_k are such that the spectral density $J(\omega)$ is

$$J(\omega) = \sum_k |g_k|^2 \delta(\omega - \omega_k) = 2\,\alpha\,\omega\,\exp\left(-\frac{\omega}{\omega_{\text{cutoff}}}\right).$$

Also, let's assume the initial density matrix of the spin is the up state

$$\rho(0) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

and the bath is initially at zero temperature.

For the numerical simulation it is advisable to choose a characteristic frequency and express all other physical parameters in terms of this frequency. Here, we choose Ω for this and write:

- $\Omega = 1.0\Omega$

- $\omega_c = 5.0\Omega$
- $\alpha = 0.3$

```
Omega_A = 1.0
omega_cutoff_A = 5.0
alpha_A = 0.3
```

3.1.2 A.2: Create System, Spectral Density and Bath Objects

To input the operators you can simply use numpy matrices. For the most common operators you can, more conveniently, use the `tempo.operators` module:

```
tempo.operators.sigma("x")
```

```
array([[0.+0.j, 1.+0.j],
       [1.+0.j, 0.+0.j]])
```

```
tempo.operators.spin_dm("up")
```

```
array([[1.+0.j, 0.+0.j],
       [0.+0.j, 0.+0.j]])
```

System

$$H_S = \frac{\Omega}{2}\hat{\sigma}_x,$$

```
system_A = tempo.System(0.5 * Omega_A * tempo.operators.sigma("x"))
```

Correlations

$$J(\omega) = 2\,\alpha\,\omega\,\exp\left(-\frac{\omega}{\omega_{\text{cutoff}}}\right)$$

Because the spectral density is of the standard power-law form,

$$J(\omega) = 2\alpha\frac{\omega^{\zeta}}{\omega_c^{\zeta-1}}X(\omega,\omega_c)$$

with $\zeta = 1$ and X of the type `'exponential'` we define the spectral density with:

```
correlations_A = tempo.PowerLawSD(alpha=alpha_A,
                                  zeta=1,
                                  cutoff=omega_cutoff_A,
                                  cutoff_type='exponential',
                                  max_correlation_time=8.0)
```

Bath

The bath couples with the operator $\frac{1}{2}\hat{\sigma}_z$ to the system.

```
bath_A = tempo.Bath(0.5 * tempo.operators.sigma("z"), correlations_A)
```

3.1.3 A.3: The TEMPO Computation

Now, that we have the system and the bath objects ready we can compute the dynamics of the spin starting in the up state, from time $t = 0$ to $t = 5\,\Omega^{-1}$

```
dynamics_A_1 = tempo.tempo_compute(system=system_A,
                                   bath=bath_A,
                                   initial_state=tempo.operators.spin_dm("up"),
                                   start_time=0.0,
                                   end_time=5.0,
                                   tollerance=0.01)
```

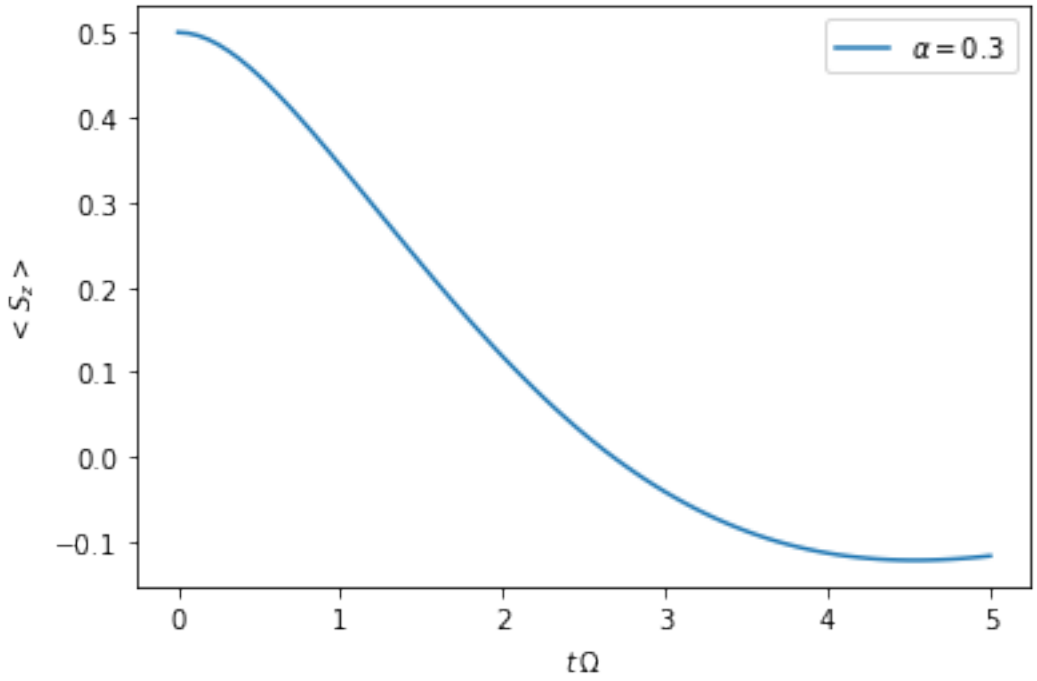
```
../time_evolving_mpo/tempo.py:495: UserWarning: Estimating parameters for TEMPO␣
↪computation. No guarantie that resulting TEMPO computation converges towards the␣
↪correct dynamics! Please refere to the TEMPO documentation and check convergence by␣
↪varying the parameters for TEMPO manually.
  warnings.warn(GUESS_WARNING_MSG, UserWarning)
WARNING: Estimating parameters for TEMPO computation. No guarantie that resulting␣
↪TEMPO computation converges towards the correct dynamics! Please refere to the␣
↪TEMPO documentation and check convergence by varying the parameters for TEMPO␣
↪manually.
```

```
100.0%   80 of   80 [########################################] 00:00:06
Elapsed time: 6.7s
```

and plot the result:

```
t_A_1, z_A_1 = dynamics_A_1.expectations(0.5*tempo.operators.sigma("z"), real=True)
plt.plot(t_A_1, z_A_1, label=r'$\alpha=0.3$')
plt.xlabel(r'$t\,\Omega$')
plt.ylabel(r'$<S_z>$')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fbcfc5fa160>
```



Yay! This looks like the plot in figure 2a [Strathearn2018].

Let's have a look at the above warning. It said:

```
WARNING: Estimating parameters for TEMPO calculation. No guarantie that resulting␣
↪TEMPO calculation converges towards the correct dynamics! Please refere to the␣
↪TEMPO documentation and check convergence by varying the parameters for TEMPO␣
↪manually.
```

We got this message because we didn't tell the package what parameters to use for the TEMPO computation, but instead only specified a `tollerance`. The package tries it's best by implicitly calling the function `tempo.guess_tempo_parameters()` to find parameters that are appropriate for the spectral density and system objects given.

TEMPO Parameters

There are **three key parameters** to a TEMPO computation:

- `dt` - Length of a time step δt - It should be small enough such that a trotterisation between the system Hamiltonian and the environment it valid, and the environment auto-correlation function is reasonably well sampled.
- `dkmax` - Number of time steps $K \in \mathbb{N}$ - It must be large enough such that $\delta t \times K$ is larger than the neccessary memory time τ_{cut}.
- `epsrel` - The maximal relative error ϵ_{rel} in the singular value truncation - It must be small enough such that the numerical compression (using tensor network algorithms) does not truncate relevant correlations.

To choose the right set of initial parameters, we recommend to first use the `tempo.guess_tempo_parameters()` function and then check with the helper function `tempo.helpers.plot_correlations_with_parameters()` whether it satisfies the above requirements:

```
parameters = tempo.guess_tempo_parameters(system=system_A,
                                          bath=bath_A,
```

(continues on next page)

(continued from previous page)

```
                                            start_time=0.0,
                                            end_time=5.0,
                                            tollerance=0.01)
print(parameters)
```

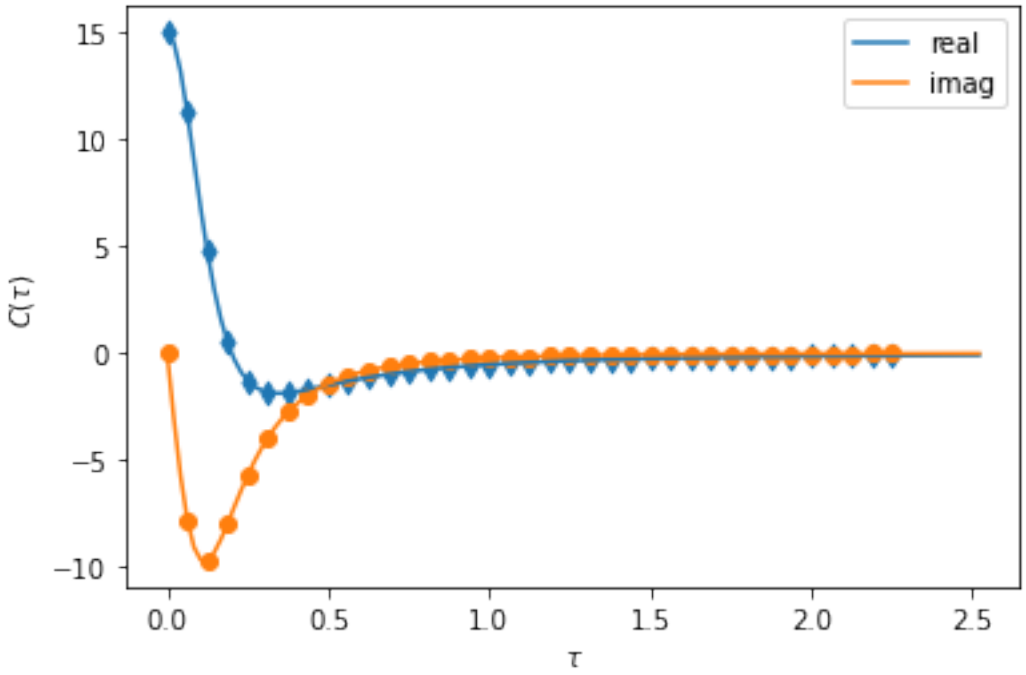
```
../time_evolving_mpo/tempo.py:495: UserWarning: Estimating parameters for TEMPO␣
↪computation. No guarantie that resulting TEMPO computation converges towards the␣
↪correct dynamics! Please refere to the TEMPO documentation and check convergence by␣
↪varying the parameters for TEMPO manually.
  warnings.warn(GUESS_WARNING_MSG, UserWarning)
WARNING: Estimating parameters for TEMPO computation. No guarantie that resulting␣
↪TEMPO computation converges towards the correct dynamics! Please refere to the␣
↪TEMPO documentation and check convergence by varying the parameters for TEMPO␣
↪manually.
```

```
----------------------------------------------
TempoParameters object: Roughly estimated parameters
 Estimated with 'guess_tempo_parameters()'
  dt             = 0.0625
  dkmax          = 37
  epsrel         = 2.4846963223857106e-05
```

```
tempo.helpers.plot_correlations_with_parameters(bath_A.correlations, parameters)
```

```
../time_evolving_mpo/helpers.py:59: UserWarning: Matplotlib is currently using␣
↪module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot show␣
↪the figure.
  fig.show()
```

```
<AxesSubplot:xlabel='$\tau$', ylabel='$C(\tau)$'>
```



In this plot you see the real and imaginary part of the environments auto-correlation as a function of the delay time

τ and the sampling of it corresponding the the chosen parameters. The spacing and the number of sampling points is given by `dt` and `dkmax` respectively. We can see that the auto-correlation function is close to zero for delay times larger than approx $2\Omega^{-1}$ and that the sampling points follow the curve reasonably well. Thus this is a reasonable set of parameters.

We can choose a set of parameters by hand and bundle them into a `TempoParameters` object,

```
tempo_parameters_A = tempo.TempoParameters(dt=0.1, dkmax=30, epsrel=10**(-4), name=
↪"my rough parameters")
print(tempo_parameters_A)
```

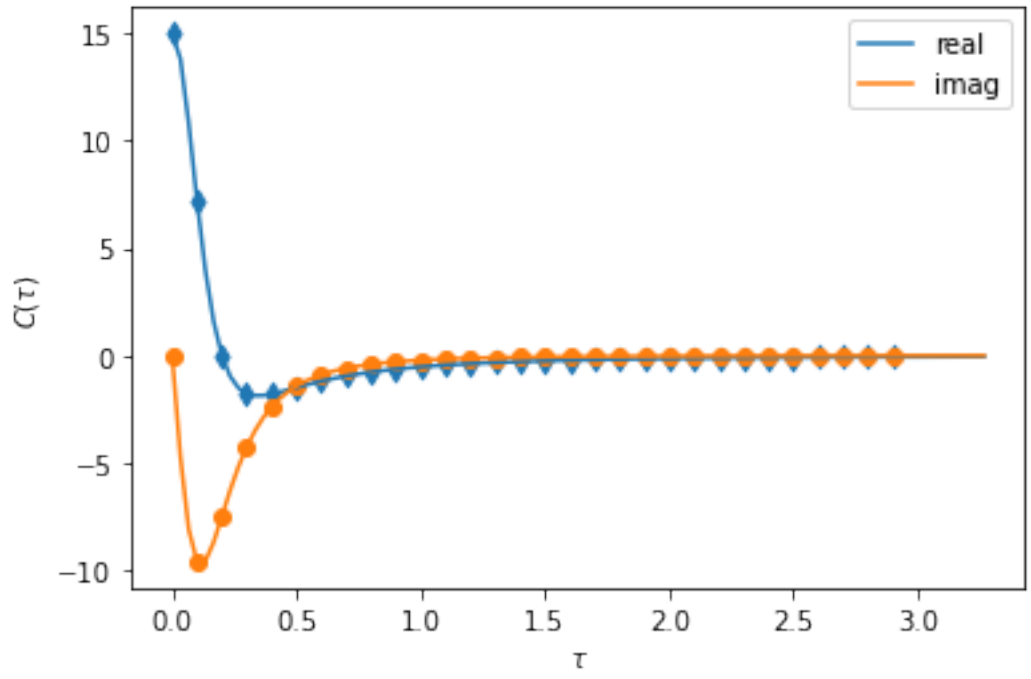
```
------------------------------------------
TempoParameters object: my rough parameters
 __no_description__
  dt             = 0.1
  dkmax          = 30
  epsrel         = 0.0001
```

and check again with the helper function:

```
tempo.helpers.plot_correlations_with_parameters(bath_A.correlations, tempo_parameters_
↪A)
```

```
../time_evolving_mpo/helpers.py:59: UserWarning: Matplotlib is currently using␣
↪module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot show␣
↪the figure.
  fig.show()
```

```
<AxesSubplot:xlabel='$\tau$', ylabel='$C(\tau)$'>
```



We could feed this object into the `tempo.tempo_compute()` function to get the dynamics of the system. However, instead of that, we can split up the work that `tempo.tempo_compute()` does into several steps, which allows us to resume a computation to get later system dynamics without having to start over. For this we start with creating a `Tempo` object:

```
tempo_A = tempo.Tempo(system=system_A,
                      bath=bath_A,
                      parameters=tempo_parameters_A,
                      initial_state=tempo.operators.spin_dm("up"),
                      start_time=0.0)
```

We can start by computing the dynamics up to time $5.0\,\Omega^{-1}$,

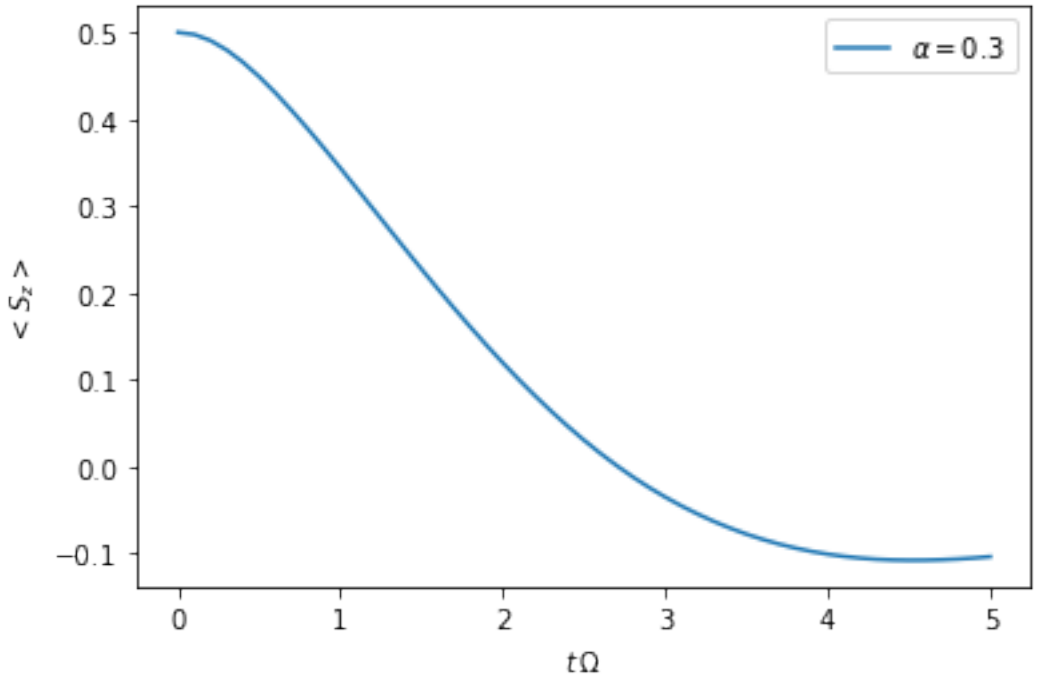
```
tempo_A.compute(end_time=5.0)
```

```
100.0%   50 of   50 [########################################] 00:00:03
Elapsed time: 3.0s
```

then get and plot the dynamics of expecatation values,

```
dynamics_A_2 = tempo_A.get_dynamics()
plt.plot(*dynamics_A_2.expectations(0.5*tempo.operators.sigma("z"),real=True), label=r
↪'$\alpha=0.3$')
plt.xlabel(r'$t\,\Omega$')
plt.ylabel(r'$<S_z>$')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fbcfc54f4a8>
```



then continue the computation to $15.0\,\Omega^{-1}$,

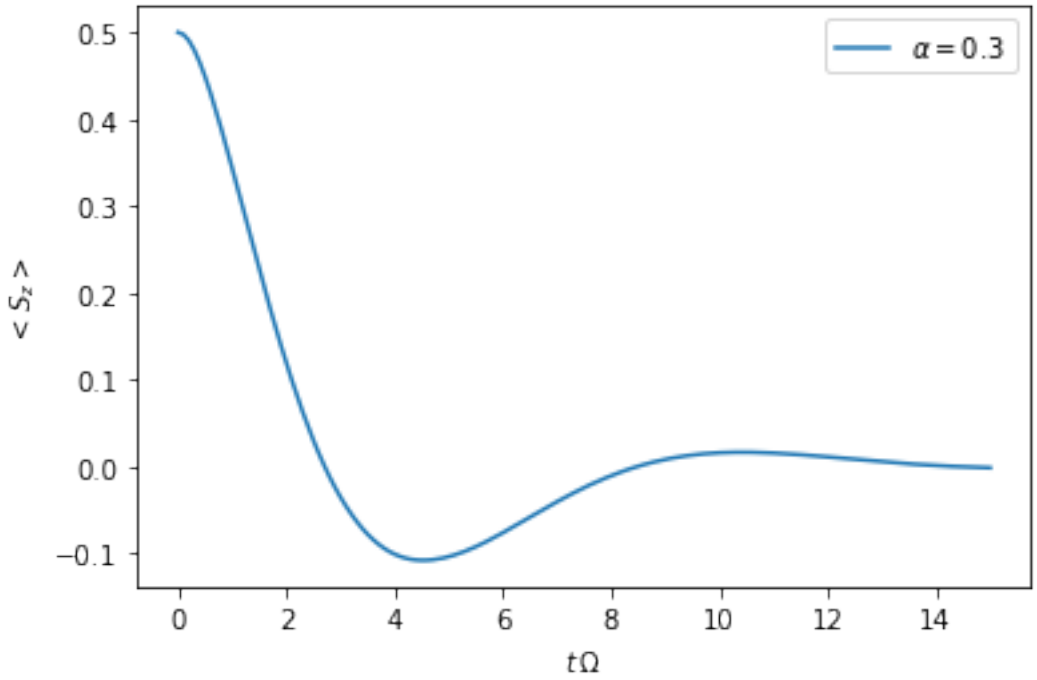
```
tempo_A.compute(end_time=15.0)
```

```
100.0%  100 of  100 [########################################] 00:00:12
Elapsed time: 12.3s
```

and then again get and plot the dynamics of expecatation values.

```
dynamics_A_2 = tempo_A.get_dynamics()
plt.plot(*dynamics_A_2.expectations(0.5*tempo.operators.sigma("z"),real=True), label=r
↪'$\alpha=0.3$')
plt.xlabel(r'$t\,\Omega$')
plt.ylabel(r'$<S_z>$')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fbcfc13dd30>
```



Finally, we note: to validate the accuracy the result **it vital to check the convergence of such a simulation by varying all three computational parameters!** For this we recommend repeating the same simulation with slightly "better" parameters (smaller `dt`, larger `dkmax`, smaller `epsrel`) and to consider the difference of the result as an estimate of the upper bound of the accuracy of the simulation.

CHAPTER

FOUR

API OUTLINE

The TEMPO Collaboration is continuously developing extensions to the original TEMPO algorithm to make this package applicable to a wider set of scenarios (the process tensor approach is one such extension). This calls for a flexible API design to allow to reuse the same objects with different algorithms. We therefore choose an almost fully object oriented approach and separate the code into three layers:

1. **The physical layer**: Consists of objects that describe physical quantities, like a system (with a specific Hamiltonian) or the spectral density of an environment.
2. **The algorithms layer**: Gathers the information from the physical layer and feeds it (with particular simulation parameters) into the backend. It has the opportunity to make use of specific, extra information from the physical layer.
3. **The backend layer**: Is the part of the code where the task has been reduced to a mathematically well defined computation, such as a specific tensor network or an integral. This not only allows to compare the performance of different implementations, it also allows to write hardware specialised implementations (single CPU, cluster, GPU) while keeping the other two layers untouched.

Also, we try to supply some handy utilities such as shortcuts for the Pauli operators for example.

4.1 The physical layer

class ***time_evolving_mpo.system.BaseSystem*** Abstract class representing a quantum system of interest.

class ***time_evolving_mpo.system.System*** Encodes system Hamiltonian and possibly some additional Markovian decay.

class ***time_evolving_mpo.system.TimeDependentSystem*** Encodes a time dependent system Hamiltonian and possibly some additional time dependent Markovian decay.

class ***time_evolving_mpo.correlations.BaseCorrelations*** Abstract class representing the environments auto-correlations.

class ***time_evolving_mpo.correlations.CustomCorrelations*** Encode an explicitly given environment auto-correlation function.

class ***time_evolving_mpo.correlations.CustomSD*** Encodes the auto-correlations for a given spectral density.

class ***time_evolving_mpo.correlations.PowerLawSD*** Encodes the auto-correlations for a given spectral density of a power law form.

class ***time_evolving_mpo.bath.Bath*** Bundles a *time_evolving_mpo.correlations.BaseCorrelations* object together with a coupling operator.

4.2 The algorithms layer

4.2.1 TEMPO

class ***time_evolving_mpo.tempo.TempoParameters*** Stores a set of parameters for a TEMPO computation.

class ***time_evolving_mpo.tempo.Tempo*** Class to facilitate a TEMPO computation.

method ***time_evolving_mpo.tempo.Tempo.compute()*** Method that carries out a TEMPO computation.

class ***time_evolving_mpo.dynamics.Dynamics*** Object that encodes the time evolution of a system (with discrete time steps).

function ***time_evolving_mpo.tempo.guess_tempo_parameters()*** Function that chooses an appropriate set of parameters for a particular TEMPO computation.

4.2.2 PT-TEMPO

class ***time_evolving_mpo.pt_tempo.PtTempoParameters*** Stores a set of parameters for a PT-TEMPO computation.

class ***time_evolving_mpo.pt_tempo.PtTempo*** Class to facilitate a PT-TEMPO computation.

method ***time_evolving_mpo.pt_tempo.PtTempo.compute()*** Method that carries out a PT-TEMPO computation.

class ***time_evolving_mpo.process_tensor.ProcessTensor*** Object that encodes a so called process tensor (which captures all possible Markovian and non-Markovian interactions between some system and an environment).

4.3 The backend layer

Currently the only backend available is the `'tensor-network'` backend, makes use of the external python package TensorNetwork to carry out the heavy lifting of the tensor network computations. This package itself can, however, be configured to use different tensor network backends (such as "numpy", "tensorflow" and "pytorch"). All the classes belonging to the algorithm layer allow you to choose the backend and its configuration (with the parameters `backend` and `backend_config`).

The default uses:

```
backend = 'tensor-network'
backend_config = {'backend':'numpy'}
```

4.4 Utillities

module *time_evolving_mpo.operators* Supplies several commonly used operators, such as the Pauli matrices and spin density matrices.

function *time_evolving_mpo.helpers.plot_correlations_with_parameters()* A helper function to plot an auto-correlation function and the sampling points given by a set of parameters for a TEMPO computation.

CHAPTER

FIVE

ALL MODULES

5.1 tempo.base_api

Module for base classes of API objects.

class time_evolving_mpo.base_api.**BaseAPIClass**(*name: Optional[str] = None*, *description: Optional[str] = None*, *description_dict: Optional[Dict] = None*)

Base class for API objects

Parameters

- **name** (*str*) – An optional name for the object.
- **description** (*str*) – An optional description of the object.
- **description_dict** (*dict*) – An optional dictionary with descriptive data.

property description
Detailed description of the object.

property description_dict
A dictionary for descriptive data.

property name
Name of the object.

5.2 tempo.bath

Module on physical information on the bath and its coupling to the system.

class time_evolving_mpo.bath.**Bath**(*coupling_operator: numpy.ndarray*, *correlations: time_evolving_mpo.correlations.BaseCorrelations*, *name: Optional[str] = None*, *description: Optional[str] = None*, *description_dict: Optional[Dict] = None*)

Bases: *time_evolving_mpo.base_api.BaseAPIClass*

Represents the bath degrees of freedom with a specific coupling operator (to the system) and a specific auto-correlation function.

Parameters

- **coupling_operator** (*np.ndarray*) – The system operator to which the bath couples.
- **correlations** (BaseCorrelations) – The bath's auto correlation function.
- **name** (*str*) – An optional name for the bath.

- **description** (*str*) – An optional description of the bath.
- **description_dict** (*dict*) – An optional dictionary with descriptive data.

Raises **ValueError:** – If the temperature T is smaller then 0.

property correlations
The correlations of the bath.

property coupling_operator
The diagonalised system operator to which the bath couples.

property dimension
Hilbert space dimension of the coupling operator.

property unitary_transform
The unitary that makes the coupling operator diagonal.

5.3 tempo.correlations

Module for environment correlations.

class time_evolving_mpo.correlations.**BaseCorrelations**(*name: Optional[str] = None*, *description: Optional[str] = None*, *description_dict: Optional[Dict] = None*)

Bases: *time_evolving_mpo.base_api.BaseAPIClass*

Base class for environment auto-correlations.

correlation(*tau: Any*, *epsrel: Optional[float] = 1.4901161193847656e-08*, *subdiv_limit: Optional[int] = 256*) → Any
Auto-correlation function.

$$C(\tau) = C(t, t-\tau) = \langle F(t)F(t-\tau)\rangle_{\mathrm{env}}$$

where τ is the time difference *tau* and $F(t)$ is the the environment part of the coupling operator in Heisenberg picture with respect to the environment Hamiltonian.

Parameters

- **tau** (*ndarray*) – Time difference τ
- **epsrel** (*float*) – Relative error tollerance.
- **subdiv_limit** (*int*) – Maximal number of interval subdivisions for numerical integration.

Returns **correlation** – The auto-correlation function $C(\tau)$ at time τ.

Return type ndarray

correlation_2d_integral(*delta: float*, *time_1: float*, *time_2: Optional[float] = None*, *shape: Optional[str] = 'square'*, *epsrel: Optional[float] = 1.4901161193847656e-08*, *subdiv_limit: Optional[int] = 256*) → complex

2D integrals of the correlation function

$$\eta_{\text{square}} = \int_{t_1}^{t_1+\Delta} \int_0^{\Delta} C(t'-t'')dt''dt'$$
$$\eta_{\text{upper-triangle}} = \int_{t_1}^{t_1+\Delta} \int_0^{t'-t_1} C(t'-t'')dt''dt'$$
$$\eta_{\text{lower-triangle}} = \int_{t_1}^{t_1+\Delta} \int_{t'-t_1}^{\Delta} C(t'-t'')dt''dt'$$
$$\eta_{\text{rectangle}} = \int_{t_1}^{t_2} \int_0^{\Delta} C(t'-t'')dt''dt'$$

for *shape* either `'square'`, `'upper-triangle'`, `'lower-triangle'`, or `'rectangle'`.

Parameters

- **delta** (*float*) – Length of integration intevals.
- **time_1** (*float*) – Lower bound of integration interval of dt'.
- **time_2** (*float*) – Upper bound of integration interval of dt' for *shape* = `'rectangle'`.
- **shape** (str (default = `'square'`)) – The shape of the 2D integral. Shapes are: {`'square'`, `'upper-triangle'`, `'lower-triangle'`, `'lower-triangle'`}
- **epsrel** (*float*) – Relative error tollerance.
- **subdiv_limit** (*int*) – Maximal number of interval subdivisions for numerical integration.

Returns **integral** – The numerical value for the two dimensional integral η_{shape}.

Return type float

property max_correlation_time
Maximal correlation time.

class time_evolving_mpo.correlations.**CustomCorrelations**(*correlation_function: Callable[[float], float], max_correlation_time: Optional[float] = None, name: Optional[str] = None, description: Optional[str] = None, description_dict: Optional[Dict] = None*)

Bases: *time_evolving_mpo.correlations.BaseCorrelations*

Encodes a custom auto-correlation function

$$C(\tau) = C(t, t-\tau) = \langle F(t)F(t-\tau)\rangle_{\text{env}}$$

with time difference *tau* τ and $F(t)$ is the the environment part of the coupling operator in Heisenberg picture with respect to the environment Hamiltonian. We assume that $C(\tau) = 0$ for all $\tau > \tau_{\max}$.

Parameters

- **correlation_function** (*callable*) – The correlation function C.
- **max_correlation_time** (*float*) – The maximal occuring correlation time $\tau_{\max}$.

- **name** (*str*) – An optional name for the correlations.
- **description** (*str*) – An optional description of the correlations.
- **description_dict** (*dict*) – An optional dictionary with descriptive data.

correlation(*tau: Any*, *epsrel: Optional[float] = None*, *subdiv_limit: Optional[int] = None*) → Any

Auto-correlation function.

$$C(\tau) = C(t, t-\tau) \ = \langle F(t)F(t-\tau)\rangle_{\mathrm{env}}$$

with time difference *tau* τ and $F(t)$ is the the environment part of the coupling operator in Heisenberg picture with respect to the environment Hamiltonian.

Parameters

- **tau** (*ndarray*) – Time difference τ
- **epsrel** (*float*) – Relative error tollerance (has no effect here).
- **subdiv_limit** (*int*) – Maximal number of interval subdivisions for numerical integration (has no effect here).

Returns **correlation** – The auto-correlation function $C(\tau)$ at time τ.

Return type ndarray

correlation_2d_integral

2D integrals of the correlation function

$$\eta_{\mathrm{square}} = \int_{t_1}^{t_1+\Delta}\int_{0}^{\Delta} C(t'-t'')dt''dt'$$
$$\eta_{\mathrm{upper-triangle}} = \int_{t_1}^{t_1+\Delta}\int_{0}^{t'-t_1} C(t'-t'')dt''dt'$$
$$\eta_{\mathrm{lower-triangle}} = \int_{t_1}^{t_1+\Delta}\int_{t'-t_1}^{\Delta} C(t'-t'')dt''dt'$$
$$\eta_{\mathrm{rectangle}} = \int_{t_1}^{t_2}\int_{0}^{\Delta} C(t'-t'')dt''dt'$$

for *shape* either `'square'`, `'upper-triangle'`, `'lower-triangle'`, or `'rectangle'`.

Parameters

- **delta** (*float*) – Length of integration intevals.
- **time_1** (*float*) – Lower bound of integration interval of dt'.
- **time_2** (*float*) – Upper bound of integration interval of dt' for *shape* = `'rectangle'`.
- **shape** (str (default = `'square'`)) – The shape of the 2D integral. Shapes are: {`'square'`, `'upper-triangle'`, `'lower-triangle'`, `'lower-triangle'`}
- **epsrel** (*float*) – Relative error tollerance.
- **subdiv_limit** (*int*) – Maximal number of interval subdivisions for numerical integration.

Returns **integral** – The numerical value for the two dimensional integral η_{shape}.

Return type float

class time_evolving_mpo.correlations.**CustomSD**(*j_function: Callable[[float], float], cutoff: float, cutoff_type: Optional[str] = 'exponential', temperature: Optional[float] = 0.0, max_correlation_time: Optional[float] = None, name: Optional[str] = None, description: Optional[str] = None, description_dict: Optional[Dict] = None*)

Bases: *time_evolving_mpo.correlations.BaseCorrelations*

Correlations corresponding to a custom spectral density. The resulting spectral density is

$$J(\omega) = j(\omega)X(\omega, \omega_c),$$

with *j_function* j, *cutoff* ω_c and a cutoff type X.

If *cutoff_type* is

- `'hard'` then $X(\omega, \omega_c) = \Theta(\omega_c - \omega)$, where Θ is the Heaviside step function.
- `'exponential'` then $X(\omega, \omega_c) = \exp(-\omega/\omega_c)$.
- `'gaussian'` then $X(\omega, \omega_c) = \exp(-\omega^2/\omega_c^2)$.

Parameters

- **j_function** (*callable*) – The spectral density j without the cutoff.
- **cutoff** (*float*) – The cutoff frequency ω_c.
- **cutoff_type** (str (default = `'exponential'`)) – The cutoff type. Types are: {`'hard'`, `'exponential'`, `'gaussian'`}
- **temperature** (*float*) – The environment's temperature.
- **max_correlation_time** (*float*) – The maximal occuring correlation time $\tau_{\max}$.
- **name** (*str*) – An optional name for the correlations.
- **description** (*str*) – An optional description of the correlations.
- **description_dict** (*dict*) – An optional dictionary with descriptive data.

correlation(*tau: Any, epsrel: Optional[float] = 1.4901161193847656e-08, subdiv_limit: Optional[int] = 256*) → Any

Auto-correlation function associated to the spectral density at the given temperature T

$$C(\tau) = \int_0^\infty J(\omega) \left[\cos(\omega\tau) \coth\left(\frac{\omega}{2T}\right) - i\sin(\omega\tau)\right] \mathrm{d}\omega.$$

with time difference *tau* τ.

Parameters

- **tau** (*ndarray*) – Time difference τ
- **epsrel** (*float*) – Relative error tollerance.
- **epsrel** – Relative error tollerance.
- **subdiv_limit** (*int*) – Maximal number of interval subdivisions for numerical integration.

Returns **correlation** – The auto-correlation function $C(\tau)$ at time τ.

Return type ndarray

correlation_2d_integral
2D integrals of the correlation function

$$\eta_{\text{square}} = \int_{t_1}^{t_1+\Delta} \int_{0}^{\Delta} C(t'-t'')dt''dt'$$
$$\eta_{\text{upper-triangle}} = \int_{t_1}^{t_1+\Delta} \int_{0}^{t'-t_1} C(t'-t'')dt''dt'$$
$$\eta_{\text{lower-triangle}} = \int_{t_1}^{t_1+\Delta} \int_{t'-t_1}^{\Delta} C(t'-t'')dt''dt'$$
$$\eta_{\text{rectangle}} = \int_{t_1}^{t_2} \int_{0}^{\Delta} C(t'-t'')dt''dt'$$

for *shape* either `'square'`, `'upper-triangle'`, `'lower-triangle'`, or `'rectangle'`.

Parameters

- **delta** (*float*) – Length of integration intevals.
- **time_1** (*float*) – Lower bound of integration interval of dt'.
- **time_2** (*float*) – Upper bound of integration interval of dt' for *shape* = `'rectangle'`.
- **shape** (str (default = `'square'`)) – The shape of the 2D integral. Shapes are: {`'square'`, `'upper-triangle'`, `'lower-triangle'`, `'lower-triangle'`}
- **epsrel** (*float*) – Relative error tollerance.
- **subdiv_limit** (*int*) – Maximal number of interval subdivisions for numerical integration.

Returns **integral** – The numerical value for the two dimensional integral η_{shape}.

Return type float

spectral_density(*omega: Any*) → Any
The resulting spectral density (including the cutoff).

Parameters **omega** (*ndarray*) – The frequency ω for which we want to know the spectral density.

Returns **spectral_density** – The resulting spectral density $J(\omega)$ at the frequency ω.

Return type ndarray

class time_evolving_mpo.correlations.**PowerLawSD**(*alpha: float*, *zeta: float*, *cutoff: float*, *cutoff_type: str = 'exponential'*, *temperature: Optional[float] = 0.0*, *max_correlation_time: Optional[float] = None*, *name: Optional[str] = None*, *description: Optional[str] = None*, *description_dict: Optional[Dict] = None*)

Bases: *time_evolving_mpo.correlations.CustomSD*

Correlations corresponding to the spectral density of the standard form

$$J(\omega) = 2\alpha \frac{\omega^{\zeta}}{\omega_c^{\zeta-1}} X(\omega, \omega_c)$$

with *alpha* α, *zeta* ζ and a cutoff type X.

If *cutoff_type* is

- `'hard'` then $X(\omega, \omega_c) = \Theta(\omega_c - \omega)$, where Θ is the Heaviside step function.
- `'exponential'` then $X(\omega, \omega_c) = \exp(-\omega/\omega_c)$.
- `'gaussian'` then $X(\omega, \omega_c) = \exp(-\omega^2/\omega_c^2)$.

Parameters

- **alpha** (*float*) – The coupling strenght α.
- **zeta** (*float*) – The exponent ζ (corresponds to the dimensionality of the environment). The environment is called *ohmic* if $\zeta = 1$, *superohmic* if $\zeta > 1$ and *subohmic* if $\zeta < 1$
- **cutoff** (*float*) – The cutoff frequency ω_c.
- **cutoff_type** (str (default = `'exponential'`)) – The cutoff type. Types are: {`'hard'`, `'exponential'`, `'gaussian'`}
- **temperature** (*float*) – The environment's temperature.
- **max_correlation_time** (*float*) – The maximal occuring correlation time $\tau_{\max}$.
- **name** (*str*) – An optional name for the correlations.
- **description** (*str*) – An optional description of the correlations.
- **description_dict** (*dict*) – An optional dictionary with descriptive data.

5.4 tempo.dynamics

Module on the discrete time evolution of a density matrix.

class time_evolving_mpo.dynamics.**Dynamics**(*times: Optional[List[float]] = None*, *states: Optional[List[numpy.ndarray]] = None*, *name: Optional[str] = None*, *description: Optional[str] = None*, *description_dict: Optional[Dict] = None*)

Bases: *time_evolving_mpo.base_api.BaseAPIClass*

Represents a specific time evolution of a density matrix.

Parameters

- **times** (*List[float] (default = None)*) – A list of points in time.
- **states** (*List[ndarray] (default = None)*) – A list of states at the times *times*.
- **name** (*str*) – An optional name for the dynamics.
- **description** (*str*) – An optional description of the dynamics.
- **description_dict** (*dict*) – An optional dictionary with descriptive data.

add(*time: float*, *state: numpy.ndarray*) → None

Append a state at a specific time to the time evolution.

Parameters

- **time** (*float*) – The point in time.
- **state** (*ndarray*) – The state at the time *time*.

expectations (*operator: Optional[numpy.ndarray] = None*, *real: Optional[bool] = False*) → Tuple[numpy.ndarray, numpy.ndarray]

Return the time evolution of the expectation value of specific operator. The expectation for t is

$$\langle \hat{O}(t) \rangle = \mathrm{Tr}\{\hat{O}\rho(t)\}$$

with *operator* $\hat{O}$.

Parameters

- **operator** (*ndarray (default = None)*) – The operator $\hat{O}$. If *operator* is *None* then the trace of $\rho(t)$ is returned.
- **real** (*bool (default = False)*) – If set True then only the real part of the expectation is returned.

Returns

- **times** (*ndarray*) – The points in time t.
- **expectations** (*ndarray*) – Expectation values $\langle \hat{O}(t) \rangle$.

export (*filename: str*, *overwrite: bool = False*) → None

Save dynamics to a file (format TempoDynamicsFormat version 1.0).

Parameters

- **filename** (*str*) – Path and filename to file that should be created.
- **overwrite** (*bool (default = False)*) – If set *True* then file is overwritten in case it already exists.

property shape

Numpy shape of the states.

property states

States of the dynamics.

property times

Times of the dynamics.

time_evolving_mpo.dynamics.**import_dynamics** (*filename: str*) → time_evolving_mpo.dynamics.Dynamics

Load dynamics from a file (format TempoDynamicsFormat version 1.0).

Parameters **filename** (*str*) – Path and filename to file that should read in.

Returns **dynamics** – The time evolution stored in the file *filename*.

Return type *Dynamics*

5.5 tempo.exceptions

Custom TEMPO Warings and Errors.

exception time_evolving_mpo.exceptions.**NumericsError**

Bases: Exception

Custom TEMPO Error class to indicate numerical issues.

exception time_evolving_mpo.exceptions.**NumericsWarning**

Bases: UserWarning

Custom TEMPO Warning class to indicate numerical issues.

5.6 tempo.file_formats

Module to define/check the export/import file formats.

time_evolving_mpo.file_formats.**assert_process_tensor_dict**(*p_t_dict: Dict*) → None

Assert that the data is of correct .processTensor form, which is:

```
Dict[ "version"          : "1.0",
      "name"             : None / Text,
      "description"      : None / Text,
      "description_dict" : None / Dict,
      "times"            : None / float / ndarray,
      "initial_tensor"   : None / ndarray,
      "tensors"          : List[ndarray] ]
```

If the field *times* is *None* this amounts to no information on the time slots of the process tensor (only "initial step", "first step", etc). If the field *times* is a *float* it signals that the time steps are uniformly spaced. If the field *times* is an *numpy.ndarray* it has to be a vector of the time slots considered in this process tensor in ascending order. In this case the length of *times* must be the length of *tensors* plus 1.

If the field *initial_tensor* is *None* this amounts to no given initial state. If the field *initial_tensor* is an *numpy.ndarray* it must be a 2-legged tensor (i.e. a matrix) where the first leg is the internal leg connecting to the next part of the array of tensors that represent the process tensor. The second leg is vectorised initial state (in fact the first slot in the process tensor).

The field *tensors* is list of three or four legged tensors. The first and second legs are the internal legs that connect to the previous and next tensor. If *initial_tensor* is *None* the first leg of the first tensor must be a dummy leg of dimension 1. The second leg of the last tensor must always be a dummy leg of dimension 1. The third leg is the "incoming" leg of the previous time slot, while the fourth leg is the "resulting" leg of the following time slot. If the tensor has only three legs, a Kronecker-delta between the third and fourth leg is assumed.

Parameters **p_t_dict** (*dict*) – Data dictionary of the process tensor.

Raises **AssertionError** – If *p_t_dict* is not of the correct process tensor form.

time_evolving_mpo.file_formats.**assert_process_tensor_file**(*filename: str*) → None

Assert that the file is of correct .processTensor form [see tempo.assert_process_tensor_file() for more details].

Parameters **filename** (*str*) – Path to the file.

Raises **AssertionError** – If the data found in the file is not of the correct .processTensor form.

time_evolving_mpo.file_formats.**assert_tempo_dynamics_dict**(*t_dyn_dict: Dict*) → None

Assert that the data is of correct .tempoDynamics form, which is:

```
Dict[ "version"          : "1.0",
      "name"             : None / Text,
      "description"      : None / Text,
      "description_dict" : None / Dict,
      "times"            : ndarray,
      "states"           : ndarray ]
```

where *times* is a 1D array of floats and *states* is a 1D array of square matrices of the same length.

Parameters **t_dyn_dict** (*dict*) – Data dictionary of the .tempoDynamics file.

Raises **AssertionError** – If *t_dyn_dict* is not of the correct .tempoDynamics form.

time_evolving_mpo.file_formats.**assert_tempo_dynamics_file**(*filename: str*) → None
Assert that the file is of correct .tempoDynamics form [see tempo.assert_tempo_dynamics_file() for more details].

Parameters **filename** (*str*) – Path to the file.

Raises **AssertionError** – If the data found in the file is not of the correct .tempoDynamics form.

time_evolving_mpo.file_formats.**check_process_tensor_file**(*filename: str*) → bool
Check if file is of correct .processTensor form [see tempo.assert_process_tensor_file() for more details].

Parameters **filename** (*str*) – Path to the file.

Returns **True or False** – True if the file has the correct format.

Return type bool

time_evolving_mpo.file_formats.**check_tempo_dynamics_file**(*filename: str*) → bool
Check if file is of correct .tempoDynamics form. [see tempo.assert_tempo_dynamics_file() for more details].

Parameters **filename** (*str*) – Path to the file.

Returns **True or False** – True if the file has the correct format.

Return type bool

time_evolving_mpo.file_formats.**print_process_tensor_dict**(*p_t_dict: Dict*) → None
Print header of .processTensor data.

time_evolving_mpo.file_formats.**print_process_tensor_file**(*filename: str*) → None
Print header of .processTensor file.

time_evolving_mpo.file_formats.**print_tempo_dynamics_dict**(*t_dyn_dict: Dict*) → None
Print header of .processTensor data.

time_evolving_mpo.file_formats.**print_tempo_dynamics_file**(*filename: str*) → None
Print header of .processTensor file.

5.7 tempo.helpers

Handy helper functions.

time_evolving_mpo.helpers.**plot_correlations_with_parameters**(*correlations: time_evolving_mpo.correlations.BaseCorrela parameters: time_evolving_mpo.tempo.TempoParameters, ax: matplotlib.axes._axes.Axes = None*) → matplotlib.axes._axes.Axes
Plot the correlation function on a grid that corresponds to some tempo parameters. For comparison, it also draws a solid line that is 10% longer and has two more sampling points per interval.

Parameters

- **correlations** (BaseCorrelations) – The correlation obeject we are interested in.
- **parameters** (TempoParameters) – The tempo parameters that determine the grid.

5.8 tempo.operators

Shorthand for commonly used operators.

time_evolving_mpo.operators.**create**(*n: int*)
Bosonic creation operator of dimension *n* x *n*.

Parameters **n** (*int*) – Dimension of the Hilbert space.

Returns **create** – Creation operator matrix of dimension *n* x *n*.

Return type ndarray

time_evolving_mpo.operators.**destroy**(*n: int*)
Bosonic annihilation operator of dimension *n* x *n*.

Parameters **n** (*int*) – Dimension of the Hilbert space.

Returns **create** – Annihilation operator matrix of dimension *n* x *n*.

Return type ndarray

time_evolving_mpo.operators.**identity**(*n: int*)
Identity matrix of dimension *n* x *n*.

Parameters **n** (*int*) – Dimension of the square matrix.

Returns **identity** – Identity matrix of dimension *n* x *n*.

Return type ndarray

time_evolving_mpo.operators.**sigma**(*name: str*)
Spin matrix sigma of type *name*.

Parameters **name** (str{ 'id', 'x', 'y', 'z', '+', '-'}) –

Returns **sigma** – Spin matrix of type *name*.

Return type ndarray

time_evolving_mpo.operators.**spin_dm**(*name: str*)
Spin 1/2 state of type *name*.

Parameters **name** (str{ 'up'/'z+', 'down'/'z-', 'x+', 'x-', 'y+', 'y-', mixed}) –

Returns **density_matrix** – Spin density matrix.

Return type ndarray

5.9 tempo.process_tensor

Module for the process tensor (PT) object. This code is based on [Pollock2018].

[Pollock2018] F. A. Pollock, C. Rodriguez-Rosario, T. Frauenheim, M. Paternostro, and K. Modi, *Non-Markovian quantumprocesses: Complete framework and efficient characterization*, Phys. Rev. A 97, 012127 (2018).

class time_evolving_mpo.process_tensor.**ProcessTensor**(*times: Union[float, numpy.ndarray], tensors: List[numpy.ndarray], initial_tensor: Optional[numpy.ndarray] = None, backend: Optional[str] = None, backend_config: Optional[Dict] = None, name: Optional[str] = None, description: Optional[str] = None, description_dict: Optional[Dict] = None*)

Bases: *time_evolving_mpo.base_api.BaseAPIClass*

Represents a specific process tensor.

If the field *times* is *None* this amounts to no information on the time slots of the process tensor (only "initial step", "first step", etc). If the field *times* is a *float* it signals that the time steps are uniformly spaced. If the field *times* is an *numpy.ndarray* it has to be a vector of the time slots considered in this process tensor in ascending order. In this case the length of *times* must be the length of *tensors* plus 1.

If the field *initial_tensor* is *None* this amounts to no given initial state. If the field *initial_tensor* is an *numpy.ndarray* it must be a 2-legged tensor (i.e. a matrix) where the first leg is the internal leg connecting to the next part of the array of tensors that represent the process tensor. The second leg is vectorised initial state (in fact the first slot in the process tensor).

The field *tensors* is list of three or four legged tensors. The first and second legs are the internal legs that connect to the previous and next tensor. If *initial_tensor* is *None* the first leg of the first tensor must be a dummy leg of dimension 1. The second leg of the last tensor must always be a dummy leg of dimension 1. The third leg is the "incoming" leg of the previous time slot, while the fourth leg is the "resulting" leg of the following time slot. If the tensor has only three legs, a Kronecker-delta between the third and fourth leg is assumed.

Parameters

- **times** (*ndarray / float / None*) – Time slots of process tensor. See description above.
- **tensors** (*list(ndarray)*) – Process tensor tensors in MPS form. See description above.
- **initial_tensor** (*ndarray*) – Initial tensor of process tensor in MPS form. See description above.
- **backend** (*str (default = None)*) – The name of the backend to use foe computations. If *backend* is `None` then the default backend is used.
- **backend_config** (*dict (default = None)*) – The configuration of the backend. If *backend_config* is `None` then the default backend configuration is used.
- **name** (*str*) – An optional name for the process tensor.
- **description** (*str*) – An optional description of the process tensor.
- **description_dict** (*dict*) – An optional dictionary with descriptive data.

compute_dynamics_from_system(*system: time_evolving_mpo.system.BaseSystem, initial_state: Optional[numpy.ndarray] = None*) → time_evolving_mpo.dynamics.Dynamics

Compute the system dynamics for a given system Hamiltonian.

Parameters **system** (*BaseSystem*) – Object containing the system Hamiltonian information.

Returns **dynamics** – The system dynamics for the given system Hamiltonian (accounting for the interaction with the environment).

Return type *Dynamics*

compute_final_state_from_system(*system: time_evolving_mpo.system.BaseSystem*, *initial_state: Optional[numpy.ndarray] = None*) → time_evolving_mpo.dynamics.Dynamics

Compute final state for a given system Hamiltonian.

Parameters **system** (BaseSystem) – Object containing the system Hamiltonian information.

Returns **final_state** – The final system system state for the given system Hamiltonian (accounting for the interaction with the environment).

Return type ndarray

export(*filename: str*, *overwrite: bool = False*) → None

Save process tensor to a file (format ProcessTensorFormat version 1.0).

Parameters

- **filename** (*str*) – Path and filename to file that should be created.
- **overwrite** (*bool (default = False)*) – If set *True* then file is overwritten in case it already exists.

get_bond_dimensions() → numpy.ndarray

Return the bond dimensions of the MPS form of the process tensor.

property times

Times of the dynamics.

time_evolving_mpo.process_tensor.**import_process_tensor**(*filename: str*) → time_evolving_mpo.process_tensor.ProcessTensor

Load process tensor from a file (format ProcessTensorFormat version 1.0).

Parameters **filename** (*str*) – Path and filename to file that should read in.

Returns **process_tensor** – The process tensor stored in the file *filename*.

Return type *ProcessTensor*

5.10 tempo.pt_tempo

Module for the process tensor time evolving matrix product operator algorithm (PT-TEMPO). This code is based on [Strathearn2018, Pollock2018, Jorgensen2019, Fux2021].

[Strathearn2018] A. Strathearn, P. Kirton, D. Kilda, J. Keeling and B. W. Lovett, *Efficient non-Markovian quantum dynamics using time-evolving matrix product operators*, Nat. Commun. 9, 3322 (2018).

[Pollock2018] F. A. Pollock, C. Rodriguez-Rosario, T. Frauenheim, M. Paternostro, and K. Modi, *Non-Markovian quantumprocesses: Complete framework and efficient characterization*, Phys. Rev. A 97, 012127 (2018).

[Jorgensen2019] M. R. Jørgensen and F. A. Pollock, *Exploiting the causal tensor network structure of quantum processes to efficiently simulate non-markovian path integrals*, Phys. Rev. Lett. 123, 240602 (2019)

[Fux2021] G. E. Fux, E. Butler, P. R. Eastham, B. W. Lovett, and J. Keeling, *Efficient exploration of Hamiltonian parameter space for optimal control of non-Markovian open quantum systems*, arXiv2101.03071 (2021).

class time_evolving_mpo.pt_tempo.**PtTempo**(*bath: time_evolving_mpo.bath.Bath*, *start_time: float*, *end_time: float*, *parameters: time_evolving_mpo.pt_tempo.PtTempoParameters*, *backend: Optional[str] = None*, *backend_config: Optional[Dict] = None*, *name: Optional[str] = None*, *description: Optional[str] = None*, *description_dict: Optional[Dict] = None*)

Bases: *time_evolving_mpo.base_api.BaseAPIClass*

Class to facilitate a PT-TEMPO computation.

Parameters

- **bath** (Bath) – The Bath (includes the coupling operator to the sytem).
- **parameters** (PtTempoParameters) – The parameters for the PT-TEMPO computation.
- **start_time** (*float*) – The start time.
- **backend** (*str (default = None)*) – The name of the backend to use for the computation. If *backend* is None then the default backend is used.
- **backend_config** (*dict (default = None)*) – The configuration of the backend. If *backend_config* is None then the default backend configuration is used.
- **name** (*str (default = None)*) – An optional name for the tempo object.
- **description** (*str (default = None)*) – An optional description of the tempo object.
- **description_dict** (*dict (default = None)*) – An optional dictionary with descriptive data.

compute(*progress_type: Optional[str] = None*) → None

Propagate (or continue to propagete) the TEMPO tensor network to time *end_time*.

Parameters **progress_type** (*str (default = None)*) – The progress report type during the computation. Types are: {silent, simple, bar}. If *None* then the default progress type is used.

property dimension

Hilbert space dimension.

get_process_tensor(*progress_type: Optional[str] = None*, *backend: Optional[str] = None*, *backend_config: Optional[Dict] = None*) → time_evolving_mpo.process_tensor.ProcessTensor

Returns a the computed process tensor. It performs the computation if it hasn't been already done.

Parameters

- **backend** (*str (default = None)*) – The name of the backend for the following process tensor computations. If *backend* is None then the default backend is used.
- **backend_config** (*dict (default = None)*) – The configuration of the backend. If *backend_config* is None then the default backend configuration is used.

Returns **process_tensor** – The computed process tensor.

Return type *ProcessTensor*

class time_evolving_mpo.pt_tempo.**PtTempoParameters**(*dt: float*, *dkmax: int*, *epsrel: float*, *name: Optional[str] = None*, *description: Optional[str] = None*, *description_dict: Optional[Dict] = None*)

Bases: *time_evolving_mpo.tempo.TempoParameters*

Parameters for the PT-TEMPO computation.

Parameters

- **dt** (*float*) – Length of a time step δt. - It should be small enough such that a trotterisation between the system Hamiltonian and the environment it valid, and the environment auto-correlation function is reasonably well sampled.
- **dkmax** (*int*) – Number of time steps $K \in \mathbb{N}$ that should be included in the non-Markovian memory. - It must be large enough such that $\delta t \times K$ is larger than the neccessary memory time τ_{cut}.
- **epsrel** (*float*) – The maximal relative error in the singular value truncation (done in the underlying tensor network algorithm). - It must be small enough such that the numerical compression (using tensor network algorithms) does not truncate relevant correlations.

time_evolving_mpo.pt_tempo.**guess_pt_tempo_parameters**(*bath: time_evolving_mpo.bath.Bath*, *start_time: float*, *end_time: float*, *tollerance: Optional[float] = 0.0039*) → time_evolving_mpo.pt_tempo.PtTempoParameters

Function to roughly estimate appropriate parameters for a PT-TEMPO computation.

Warning: No guarantie that resulting PT-TEMPO calculation converges towards the correct dynamics! Please refere to the TEMPO documentation and check convergence by varying the parameters for PT-TEMPO manually.

Parameters

- **bath** (Bath) – The bath.
- **start_time** (*float*) – The start time.
- **end_time** (*float*) – The time to which the TEMPO should be computed.
- **tollerance** (*float*) – Tollerance for the parameter estimation.

Returns **pt_tempo_parameters** – Estimate of appropropriate tempo parameters.

Return type *TempoParameters*

time_evolving_mpo.pt_tempo.**pt_tempo_compute**(*bath: time_evolving_mpo.bath.Bath*, *start_time: float*, *end_time: float*, *parameters: Optional[time_evolving_mpo.pt_tempo.PtTempoParameters] = None*, *tollerance: Optional[float] = 0.0039*, *backend: Optional[str] = None*, *backend_config: Optional[Dict] = None*, *progress_type: Optional[str] = None*, *name: Optional[str] = None*, *description: Optional[str] = None*, *description_dict: Optional[Dict] = None*) → time_evolving_mpo.process_tensor.ProcessTensor

Shortcut for creating a process tensor by performing a PT-TEMPO computation.

Parameters

- **bath** (Bath) – The Bath (includes the coupling operator to the sytem).
- **start_time** (*float*) – The start time.
- **end_time** (*float*) – The time to which the PT-TEMPO should be computed.
- **parameters** (PtTempoParameters) – The parameters for the PT-TEMPO computation.
- **tollerance** (*float*) – Tollerance for the parameter estimation (only applicable if *parameters* is None).
- **backend** (*str (default = None)*) – The name of the backend to use for the computation. If *backend* is None then the default backend is used.
- **backend_config** (*dict (default = None)*) – The configuration of the backend. If *backend_config* is None then the default backend configuration is used.
- **progress_type** (*str (default = None)*) – The progress report type during the computation. Types are: {'silent', 'simple', 'bar'}. If *None* then the default progress type is used.
- **name** (*str (default = None)*) – An optional name for the tempo object.
- **description** (*str (default = None)*) – An optional description of the tempo object.
- **description_dict** (*dict (default = None)*) – An optional dictionary with descriptive data.

5.11 tempo.system

Module on physical information of the system.

class time_evolving_mpo.system.**BaseSystem**(*dimension: int*, *name: Optional[str] = None*, *description: Optional[str] = None*, *description_dict: Optional[Dict] = None*)

Bases: *time_evolving_mpo.base_api.BaseAPIClass*

Base class for systems.

property dimension

Hilbert space dimension of the system.

liouvillian (*t: Optional[float] = None*) → numpy.ndarray
Returns the Liouvillian super-operator $\mathcal{L}(t)$ with

$$\mathcal{L}(t)\rho = -i[\hat{H}(t), \rho] + \sum_n^N \gamma_n \left(\hat{A}_n(t)\rho\hat{A}_n^\dagger(t) - \frac{1}{2}\hat{A}_n^\dagger(t)\hat{A}_n(t)\rho - \frac{1}{2}\rho\hat{A}_n^\dagger(t)\hat{A}_n(t) \right),$$

with time t.

Parameters **t** (*float (default = None)*) – time t.

Returns **liouvillian** – Liouvillian $\mathcal{L}(t)$ at time t.

Return type ndarray

class time_evolving_mpo.system.**System**(*hamiltonian: numpy.ndarray, gammas: Optional[List[float]] = None, lindblad_operators: Optional[List[numpy.ndarray]] = None, name: Optional[str] = None, description: Optional[str] = None, description_dict: Optional[Dict] = None*)
Bases: *time_evolving_mpo.system.BaseSystem*

Represents a system (without any coupling to a non-Markovian bath). It is possible to include Lindblad terms in the master equation. The equations of motion for a system density matrix (without any coupling to a non-Markovian bath) is then:

$$\begin{aligned}\frac{d}{dt}\rho(t) =& -i[\hat{H}, \rho(t)] \\ &+ \sum_n^N \gamma_n \left(\hat{A}_n\rho(t)\hat{A}_n^\dagger - \frac{1}{2}\hat{A}_n^\dagger\hat{A}_n\rho(t) - \frac{1}{2}\rho(t)\hat{A}_n^\dagger\hat{A}_n \right)\end{aligned}$$

with *hamiltionian* $\hat{H}$, the rates *gammas* γ_n and *linblad_operators* $\hat{A}_n$.

Parameters

- **hamiltonian** (*ndarray*) – System-only Hamiltonian $\hat{H}$.
- **gammas** (*List(float)*) – The rates γ_n.
- **lindblad_operators** (*list(ndarray)*) – The Lindblad operators $\hat{A}_n$.
- **name** (*str*) – An optional name for the system.
- **description** (*str*) – An optional description of the system.
- **description_dict** (*dict*) – An optional dictionary with descriptive data.

property gammas
List of gammas.

property hamiltonian
The system Hamiltonian.

property lindblad_operators
List of lindblad operators.

liouvillian
Returns the Liouvillian super-operator $\mathcal{L}$ with

$$\mathcal{L}\rho = -i[\hat{H}, \rho] + \sum_n^N \gamma_n \left(\hat{A}_n\rho\hat{A}_n^\dagger - \frac{1}{2}\hat{A}_n^\dagger\hat{A}_n\rho - \frac{1}{2}\rho\hat{A}_n^\dagger\hat{A}_n \right).$$

Returns **liouvillian** – Liouvillian $\mathcal{L}$.

Return type ndarray

class time_evolving_mpo.system.**TimeDependentSystem**(*hamiltonian: Callable[[float], numpy.ndarray], gammas: Optional[List[Callable[[float], float]]] = None, lindblad_operators: Optional[List[Callable[[float], numpy.ndarray]]] = None, name: Optional[str] = None, description: Optional[str] = None, description_dict: Optional[Dict] = None*)

Bases: *time_evolving_mpo.system.BaseSystem*

Represents an explicitly time dependent system (without any coupling to a non-Markovian bath). It is possible to include (also explicitly time dependent) Lindblad terms in the master equation. The equations of motion for a system density matrix (without any coupling to a non-Markovian bath) is then:

$$\frac{d}{dt}\rho(t) = -i[\hat{H}(t), \rho(t)] + \sum_n^N \gamma_n(t) \left(\hat{A}_n(t)\rho(t)\hat{A}_n(t)^\dagger - \frac{1}{2}\hat{A}_n^\dagger(t)\hat{A}_n(t)\rho(t) - \frac{1}{2}\rho(t)\hat{A}_n^\dagger(t)\hat{A}_n(t) \right)$$

with the time dependent *hamiltionian* $\hat{H}(t)$, the time dependent rates *gammas* $\gamma_n(t)$ and the time dependent *linblad_operators* $\hat{A}_n(t)$.

Parameters

- **hamiltonian** (*callable*) – System-only Hamiltonian $\hat{H}(t)$.
- **gammas** (*List(callable)*) – The rates $\gamma_n(t)$.
- **lindblad_operators** (*list(callable)*) – The Lindblad operators $\hat{A}_n(t)$.
- **name** (*str*) – An optional name for the system.
- **description** (*str*) – An optional description of the system.
- **description_dict** (*dict*) – An optional dictionary with descriptive data.

property gammas
List of gammas.

property hamiltonian
The system Hamiltonian.

property lindblad_operators
List of lindblad operators.

liouvillian(*t: Optional[float] = None*) → numpy.ndarray
Returns the Liouvillian super-operator $\mathcal{L}(t)$ with

$$\mathcal{L}(t)\rho = -i[\hat{H}(t), \rho] + \sum_n^N \gamma_n \left(\hat{A}_n(t)\rho\hat{A}_n^\dagger(t) - \frac{1}{2}\hat{A}_n^\dagger(t)\hat{A}_n(t)\rho - \frac{1}{2}\rho\hat{A}_n^\dagger(t)\hat{A}_n(t) \right),$$

with time t.

Parameters **t** (*float (default = None)*) – time t.

Returns **liouvillian** – Liouvillian $\mathcal{L}(t)$ at time t.

Return type ndarray

Raises **ValueError** – If *t* = *None*

5.12 tempo.tempo

Module on for the original time evolving matrix product operator (TEMPO) algorithm. This code is based on [Strathearn2018].

[Strathearn2018] A. Strathearn, P. Kirton, D. Kilda, J. Keeling and B. W. Lovett, *Efficient non-Markovian quantum dynamics using time-evolving matrix product operators*, Nat. Commun. 9, 3322 (2018).

class time_evolving_mpo.tempo.**Tempo**(*system: time_evolving_mpo.system.BaseSystem*, *bath: time_evolving_mpo.bath.Bath*, *parameters: time_evolving_mpo.tempo.TempoParameters*, *initial_state: numpy.ndarray*, *start_time: float*, *backend: Optional[str] = None*, *backend_config: Optional[Dict] = None*, *name: Optional[str] = None*, *description: Optional[str] = None*, *description_dict: Optional[Dict] = None*)

Bases: *time_evolving_mpo.base_api.BaseAPIClass*

Class representing the entire TEMPO tensornetwork as introduced in [Strathearn2018].

Parameters

- **system** (BaseSystem) – The system.
- **bath** (Bath) – The Bath (includes the coupling operator to the sytem).
- **parameters** (TempoParameters) – The parameters for the TEMPO computation.
- **initial_state** (*ndarray*) – The initial density matrix of the sytem.
- **start_time** (*float*) – The start time.
- **backend** (*str (default = None)*) – The name of the backend to use for the computation. If *backend* is None then the default backend is used.
- **backend_config** (*dict (default = None)*) – The configuration of the backend. If *backend_config* is None then the default backend configuration is used.
- **name** (*str (default = None)*) – An optional name for the tempo object.
- **description** (*str (default = None)*) – An optional description of the tempo object.
- **description_dict** (*dict (default = None)*) – An optional dictionary with descriptive data.

compute(*end_time: float*, *progress_type: str = None*) → time_evolving_mpo.dynamics.Dynamics

Propagate (or continue to propagete) the TEMPO tensor network to time *end_time*.

Parameters

- **end_time** (*float*) – The time to which the TEMPO should be computed.
- **progress_type** (*str (default = None)*) – The progress report type during the computation. Types are: {'silent', 'simple', 'bar'}. If *None* then the default progress type is used.

Returns **dynamics** – The instance of Dynamics associated with the TEMPO object.

Return type *Dynamics*

property dimension
Hilbert space dimension.

get_dynamics() → time_evolving_mpo.dynamics.Dynamics
Returns the instance of Dynamics associated with the Tempo object.

class time_evolving_mpo.tempo.**TempoParameters**(*dt: float*, *dkmax: int*, *epsrel: float*, *name: Optional[str] = None*, *description: Optional[str] = None*, *description_dict: Optional[Dict] = None*)
Bases: *time_evolving_mpo.base_api.BaseAPIClass*

Parameters for the TEMPO computation.

Parameters

- **dt** (*float*) – Length of a time step δt. - It should be small enough such that a trotterisation between the system Hamiltonian and the environment it valid, and the environment auto-correlation function is reasonably well sampled.
- **dkmax** (*int*) – Number of time steps $K \in \mathbb{N}$ that should be included in the non-Markovian memory. - It must be large enough such that $\delta t \times K$ is larger than the neccessary memory time τ_{cut}.
- **epsrel** (*float*) – The maximal relative error in the singular value truncation (done in the underlying tensor network algorithm). - It must be small enough such that the numerical compression (using tensor network algorithms) does not truncate relevant correlations.

property dkmax
Number of time steps that should be included in the non-Markovian memory.

property dt
Length of a time step.

property epsrel
The maximal relative error in the singular value truncation.

time_evolving_mpo.tempo.**guess_tempo_parameters**(*bath: time_evolving_mpo.bath.Bath*, *start_time: float*, *end_time: float*, *system: Optional[time_evolving_mpo.system.BaseSystem] = None*, *tollerance: Optional[float] = 0.0039*) → time_evolving_mpo.tempo.TempoParameters
Function to roughly estimate appropriate parameters for a TEMPO computation.

Warning: No guarantie that resulting TEMPO calculation converges towards the correct dynamics! Please refere to the TEMPO documentation and check convergence by varying the parameters for TEMPO manually.

Parameters

- **bath** (Bath) – The bath.
- **start_time** (*float*) – The start time.
- **end_time** (*float*) – The time to which the TEMPO should be computed.
- **system** (BaseSystem) – The system.

- **tollerance** (*float*) – Tollerance for the parameter estimation.

Returns **tempo_parameters** – Estimate of appropropriate tempo parameters.

Return type *TempoParameters*

time_evolving_mpo.tempo.**tempo_compute**(*system: time_evolving_mpo.system.BaseSystem*, *bath: time_evolving_mpo.bath.Bath*, *initial_state: numpy.ndarray*, *start_time: float*, *end_time: float*, *parameters: Optional[time_evolving_mpo.tempo.TempoParameters] = None*, *tollerance: Optional[float] = 0.0039*, *backend: Optional[str] = None*, *backend_config: Optional[Dict] = None*, *progress_type: Optional[str] = None*, *name: Optional[str] = None*, *description: Optional[str] = None*, *description_dict: Optional[Dict] = None*) → time_evolving_mpo.dynamics.Dynamics

Shortcut for creating a Tempo object and runing the computation.

Parameters

- **system** (BaseSystem) – The system.
- **bath** (Bath) – The Bath (includes the coupling operator to the sytem).
- **initial_state** (*ndarray*) – The initial density matrix of the sytem.
- **start_time** (*float*) – The start time.
- **end_time** (*float*) – The time to which the TEMPO should be computed.
- **parameters** (TempoParameters) – The parameters for the TEMPO computation.
- **tollerance** (*float*) – Tollerance for the parameter estimation (only applicable if *parameters* is None).
- **backend** (*str (default = None)*) – The name of the backend to use for the computation. If *backend* is None then the default backend is used.
- **backend_config** (*dict (default = None)*) – The configuration of the backend. If *backend_config* is None then the default backend configuration is used.
- **progress_type** (*str (default = None)*) – The progress report type during the computation. Types are: {'silent', 'simple', 'bar'}. If *None* then the default progress type is used.
- **name** (*str (default = None)*) – An optional name for the tempo object.
- **description** (*str (default = None)*) – An optional description of the tempo object.
- **description_dict** (*dict (default = None)*) – An optional dictionary with descriptive data.

5.13 tempo.util

Module for utillities.

class time_evolving_mpo.util.**BaseProgress**
Base class to display computation progress.

update(*step=None*)
Update the progress.

class time_evolving_mpo.util.**ProgressBar**(*max_value*)
Class to display the computation progress with a nice progress bar.

update(*step=None*)
Update the progress.

class time_evolving_mpo.util.**ProgressSilent**(*max_value*)
Class NOT to display the computation progress.

update(*step=None*)
Update the progress.

class time_evolving_mpo.util.**ProgressSimple**(*max_value*)
Class to display the computation progress step by step.

update(*step=None*)
Update the progress.

time_evolving_mpo.util.**acommutator**(*operator: numpy.ndarray*) → numpy.ndarray
Construct anti-commutator superoperator from operator.

time_evolving_mpo.util.**commutator**(*operator: numpy.ndarray*) → numpy.ndarray
Construct commutator superoperator from operator.

time_evolving_mpo.util.**get_progress**(*progress_type: str = None*) → time_evolving_mpo.util.BaseProgress
Get a progress class from the progress_type.

time_evolving_mpo.util.**left_right_super**(*left_operator: numpy.ndarray*, *right_operator: numpy.ndarray*) → numpy.ndarray
Construct left and right acting superoperator from operators.

time_evolving_mpo.util.**left_super**(*operator: numpy.ndarray*) → numpy.ndarray
Construct left acting superoperator from operator.

time_evolving_mpo.util.**load_object**(*filename: str*) → Any
Load an object from a file using pickle.

time_evolving_mpo.util.**right_super**(*operator: numpy.ndarray*) → numpy.ndarray
Construct right acting superoperator from operator.

time_evolving_mpo.util.**save_object**(*obj: Any*, *filename: str*, *overwrite: bool*) → None
Save an object to a file using pickle.

5.14 tempo.backends

5.14.1 base_backends

Module for base classes of backends.

class time_evolving_mpo.backends.base_backends.**BaseProcessTensorBackend**(*tensors: List[numpy.ndarray], initial_tensor: Optional[numpy.ndarray], config: Dict*)

Base class for process tensor backends.

If *initial_tensor* is *None* this amounts to no given initial state. If *initial_tensor* is an *numpy.ndarray* it must be a 2-legged tensor (i.e. a matrix) where the first leg is the internal leg connecting to the next part of the array of tensors that represent the process tensor. The second leg is vectorised initial state (in fact the first slot in the process tensor).

The parameter *tensors* is list of three or four legged tensors. The first and second legs are the internal legs that connect to the previous and next tensor. If *initial_tensor* is *None* the first leg of the first tensor must be a dummy leg of dimension 1. The second leg of the last tensor must always be a dummy leg of dimension 1. The third leg is the "incoming" leg of the previous time slot, while the fourth leg is the "resulting" leg of the following time slot. If the tensor has only three legs, a Kronecker-delta between the third and fourth leg is assumed.

Parameters

- **tensors** (*list(ndarray)*) – The tensors that make up the MPO representation of the process tensor.
- **initial_tensor** (*ndarray*) – The zeroth/initial tensor of the process tensor.

compute_dynamics(*controls: Callable[[int], Tuple[numpy.ndarray, numpy.ndarray]], initial_state: Optional[numpy.ndarray] = None*) → time_evolving_mpo.dynamics.Dynamics

Compute the Dynamics for a given set of controls.

Parameters

- **controls** (*callable(int) -> ndarray, ndarray*) – Callable that takes an integer *step* and returns the pre and post control at that time step.
- **initial_state** (*ndarray*) – Initial state vector.

Returns **dynamics** – The dynamics of the system.

Return type *Dynamics*

compute_final_state(*controls: Callable[[int], Tuple[numpy.ndarray, numpy.ndarray]], initial_state: Optional[numpy.ndarray] = None*) → numpy.ndarray

Compute final state for a given set of controls.

Parameters

- **controls** (*callable(int) -> ndarray, ndarray*) – Callable that takes an integer *step* and returns the pre and post control at that time step.
- **initial_state** (*ndarray*) – Initial state vector.

Returns **final_state** – The final state of the system.

Return type ndarray

export_tensors() → Tuple[numpy.ndarray, numpy.ndarray]
Export process tensor tensors.

Returns

- **tensors** (*list(ndarray)*) – The tensors that make up the MPO representation of the process tensor.
- **initial_tensor** (*ndarray*) – The zeroth/initial tensor of the process tensor.

get_bond_dimensions() → numpy.ndarray
Return MPS bond dimensions.

class time_evolving_mpo.backends.base_backends.**BasePtTempoBackend**(*dimension: int, influence: Callable[[int], numpy.ndarray], unitary_transform: numpy.ndarray, sum_north: numpy.ndarray, sum_west: numpy.ndarray, num_steps: int, dkmax: int, epsrel: float, config: Dict*)

Base class for process tensor tempo backends.

Parameters

- **influence** (*callable(int) -> ndarray*) – Callable that takes an integer *step* and returns the influence super operator of that *step*.
- **unitary_transform** (*ndarray*) – ToDo
- **sum_north** (*ndarray*) – The summing vector for the north leggs.
- **sum_west** (*ndarray*) – The summing vector for the west leggs.
- **dkmax** (*int*) – Number of influences to include. If `dkmax == -1` then all influences are included.
- **epsrel** (*float*) – Maximal relative SVD truncation error.

compute_step() → None
Take a step in the PT-TEMPO tensor network computation.

get_tensors() → List[numpy.ndarray]
Return the computed tensors.

initialize() → None
Initializes the PT-TEMPO tensor network.

property num_steps
The current step in the PT-TEMPO computation.

property step
The current step in the PT-TEMPO computation.

class time_evolving_mpo.backends.base_backends.**BaseTempoBackend**(*initial_state: numpy.ndarray, influence: Callable[[int], numpy.ndarray], unitary_transform: numpy.ndarray, propagators: Callable[[int], Tuple[numpy.ndarray, numpy.ndarray]], sum_north: numpy.ndarray, sum_west: numpy.ndarray, dkmax: int, epsrel: float, config: Dict*)

Base class for tempo backends.

Parameters

- **initial_state** (*ndarray*) – The initial density matrix (as a vector).
- **influence** (*callable(int) -> ndarray*) – Callable that takes an integer *step* and returns the influence super operator of that *step*.
- **unitary_transform** (*ndarray*) – Unitary that transforms the coupling operator into a diagonal form.
- **propagators** (*callable(int) -> ndarray, ndarray*) – Callable that takes an integer *step* and returns the first and second half of the system propagator of that *step*.
- **sum_north** (*ndarray*) – The summing vector for the north leggs.
- **sum_west** (*ndarray*) – The summing vector for the west leggs.
- **dkmax** (*int*) – Number of influences to include. If `dkmax == -1` then all influences are included.
- **epsrel** (*float*) – Maximal relative SVD truncation error.

compute_step() → Tuple[int, numpy.ndarray]
Takes a step in the TEMPO tensor network computation.

Returns

- **step** (*int*) – The current step count.
- **state** (*ndarray*) – Density matrix at the current step.

initialize() → Tuple[int, numpy.ndarray]
Initializes the TEMPO tensor network.

Returns

- **step** (*int = 0*) – The current step count, which after initialization, is 0 .

- **state** (*ndarray*) – Density matrix (as a vector) at the current step, which after initialization, is just the initial state.

property step

The current step in the TEMPO computation.

CHAPTER

SIX

CONTRIBUTING

Contributions of all kinds are welcome! Get in touch if you ...

- ... found a bug.
- ... have a question on how to use the code.
- ... have a suggestion, on how to improve the code or documentation.
- ... would like to get involved in writing code or documentation.
- ... have some other thoughts or suggestions.

Please, feel free to file an issue in the Issues section on GitHub for this. Also, have a look at CONTRIBUTING.md if you want to get involved in the development.

CHAPTER

SEVEN

LIST OF AUTHORS

This open source project has been started by Gerald E. Fux in 2020 as a member of the TEMPO collaboration.

7.1 Members of the TEMPO collaboration

- Kristín Arnardóttir (*University of St Andrews*) <kristinbjorga@gmail.com>
- Gerald E. Fux (*University of St Andrews*) <gf52@st-andrews.ac.uk>
- Erik Gauger (*Heriot-Watt University*) <e.gauger@hw.ac.uk>
- Jonathan Keeling (*University of St Andrews*) <jmjk@st-andrews.ac.uk>
- Peter Kirton (*University of Strathclyde*) <peter.kirton@strath.ac.uk>
- Thibaut Lacroix (*University of St Andrews*) <tfml1@st-andrews.ac.uk>
- Brendon W. Lovett (*University of St Andrews*) <bwl4@st-andrews.ac.uk>

7.2 Project administrators

- Gerald E. Fux (*University of St Andrews*) <gf52@st-andrews.ac.uk>
- Jonathan Keeling (*University of St Andrews*) <jmjk@st-andrews.ac.uk>
- Brendon W. Lovett (*University of St Andrews*) <bwl4@st-andrews.ac.uk>

7.3 Project maintainers

- Gerald E. Fux (*University of St Andrews*) <gf52@st-andrews.ac.uk>
- Peter Kirton (*University of Strathclyde*) <peter.kirton@strath.ac.uk>

7.4 Major code contributions

Version 0.1.0	Main Contributors
Setup Project (CI, API design, project planning, etc.)	Gerald E. Fux
Implement core TEMPO functionality (API and backend)	Gerald E. Fux
Implement process tensor TEMPO (API and backend)	Gerald E. Fux

For a complete list of *all* code contributors see the contributors list on GitHub.

CHAPTER

EIGHT

HOW TO CITE THIS PROJECT

8.1 Citing the Code

- **[TimeEvolvingMPO]** The TEMPO collaboration, TimeEvolvingMPO: A Python 3 package to efficiently compute non-Markovianopen quantum systems, (2020).

8.2 Please consider citing

TEMPO algorithm:

- **[Strathearn2018]** A. Strathearn, P. Kirton, D. Kilda, J. Keeling and B. W. Lovett, *Efficient non-Markovian quantum dynamics using time-evolving matrix product operators*, Nat. Commun. 9, 3322 (2018).
- **[Strathearn2019]** A. Strathearn, *Modelling Non-Markovian Quantum Sys-tems Using Tensor Networks*, Springer Theses (2020)

Process Tensor approach:

- **[Pollock2018]** F. A. Pollock, C. Rodriguez-Rosario, T. Frauenheim, M. Paternostro, and K. Modi, *Non-Markovian quantumprocesses: Complete framework and efficient characterization*, Phys. Rev. A 97, 012127 (2018).
- **[Jorgensen2020]** M. R. Jørgensen and F. A. Pollock, *Exploiting the causal tensor network structure of quantum processes to efficiently simulate non-markovian path integrals*, Phys. Rev. Lett. 123, 240602 (2019)

Process Tensor - TEMPO algorithm:

- **[Fux2021]** G. E. Fux, E. Butler, P. R. Eastham, B. W. Lovett, and J. Keeling, *Efficient exploration of Hamiltonian parameter space for optimal control of non-Markovian open quantum systems*, arXiv2101.03071 (2021).

8.3 BibTeX

See *BibTeX* for the corresponding bibTeX.

CHAPTER

NINE

BIBLIOGRAPHY

The code in this project is based on ideas from the following publications:

[Strathearn2017] A. Strathearn, B.W. Lovett, and P. Kirton, *Efficient real-time path integrals for non-Markovian spin-boson models*. New Journal of Physics, 19(9), p.093009 (2017).

[Strathearn2018] A. Strathearn, P. Kirton, D. Kilda, J. Keeling and B. W. Lovett, *Efficient non-Markovian quantum dynamics using time-evolving matrix product operators*, Nat. Commun. 9, 3322 (2018).

[Pollock2018] F. A. Pollock, C. Rodriguez-Rosario, T. Frauenheim, M. Paternostro, and K. Modi, *Non-Markovian quantumprocesses: Complete framework and efficient characterization*, Phys. Rev. A 97, 012127 (2018).

[Jorgensen2019] M. R. Jørgensen and F. A. Pollock, *Exploiting the causaltensor network structure of quantum processes to efficiently simulate non-markovian path integrals*, Phys. Rev. Lett. 123, 240602 (2019)

[Fux2021] G. E. Fux, E. Butler, P. R. Eastham, B. W. Lovett, and J. Keeling, *Efficient exploration of Hamiltonian parameter space for optimal control of non-Markovian open quantum systems*, arXiv2101.03071 (2021).

9.1 BibTeX

```
@article{Fux2021,
  archivePrefix = {arXiv},
  arxivId = {2101.03071},
  author = {Fux, Gerald E. and Butler, Eoin and Eastham, Paul R. and Lovett,
            Brendon W. and Keeling, Jonathan},
  eprint = {2101.03071},
  pages = {1--6},
  title = {{Efficient exploration of Hamiltonian parameter space for optimal
            control of non-Markovian open quantum systems}},
  url = {http://arxiv.org/abs/2101.03071},
  year = {2021}
}

@article{Jorgensen2019,
  title = {Exploiting the Causal Tensor Network Structure of Quantum
           Processes to Efficiently Simulate Non-Markovian Path Integrals},
  author = {J\o{}rgensen, Mathias R. and Pollock, Felix A.},
  journal = {Phys. Rev. Lett.},
  volume = {123},
  issue = {24},
  pages = {240602},
  numpages = {7},
  year = {2019},
  month = {Dec},
```

(continues on next page)

(continued from previous page)

```
  publisher = {American Physical Society},
  doi = {10.1103/PhysRevLett.123.240602},
  url = {https://link.aps.org/doi/10.1103/PhysRevLett.123.240602}
}

@article{Pollock2018,
  author = {Pollock, Felix A. and Rodr{\'{i}}guez-Rosario, C{\'{e}}sar and
            Frauenheim, Thomas and Paternostro, Mauro and Modi, Kavan},
  doi = {10.1103/PhysRevA.97.012127},
  issn = {24699934},
  journal = {Phys. Rev. A},
  month = {jan},
  number = {1},
  pages = {012127},
  title = {{Non-Markovian quantum processes: Complete framework and
            efficient characterization}},
  url = {https://link.aps.org/doi/10.1103/PhysRevA.97.012127
         http://arxiv.org/abs/1512.00589
         http://dx.doi.org/10.1103/PhysRevA.97.012127},
  volume = {97},
  year = {2018}
}

@article{Strathearn_2017,
      doi = {10.1088/1367-2630/aa8744},
      url = {https://doi.org/10.1088/1367-2630/aa8744},
      year = 2017,
      month = {sep},
      publisher = {{IOP} Publishing},
      volume = {19},
      number = {9},
      pages = {093009},
      author = {A Strathearn and B W Lovett and P Kirton},
      title = {Efficient real-time path integrals for non-Markovian spin-boson models}
↪,
      journal = {New Journal of Physics},
}

@article{Strathearn2018,
  author = {Strathearn, A. and Kirton, P. and Kilda, D. and Keeling, J. and
            Lovett, B. W.},
  doi = {10.1038/s41467-018-05617-3},
  issn = {20411723},
  journal = {Nat. Commun.},
  month = {dec},
  number = {1},
  pages = {3322},
  pmid = {30127490},
  title = {{Efficient non-Markovian quantum dynamics using time-evolving
            matrix product operators}},
  url = {https://doi.org/10.1038/s41467-018-05617-3},
  volume = {9},
  year = {2018}
}

@book{Strathearn2019,
  address = {Cham},
```

(continues on next page)

(continued from previous page)

```
  author = {Strathearn, Aidan},
  doi = {10.1007/978-3-030-54975-6},
  isbn = {978-3-030-54974-9},
  publisher = {Springer International Publishing},
  series = {Springer Theses},
  title = {{Modelling Non-Markovian Quantum Systems Using Tensor Networks}},
  url = {http://link.springer.com/10.1007/978-3-030-54975-6},
  year = {2020}
}

@misc{TimeEvolvingMPO,
  author={{The TEMPO collaboration}},
  title={{TimeEvolvingMPO: A Python 3 package to efficiently compute
          non-Markovian open quantum systems.}},
  year=2020,
  publisher={GitHub},
  doi={10.5281/zenodo.4428316}
  url={https://github.com/tempoCollaboration/TimeEvolvingMPO}
}
```

CHAPTER

TEN

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

t

INDEX

N

P

R

S

T

U