A Direct Time-Sliced Derivation of the Feynman–Vernon Influence Functional

From Harmonic-Bath Elimination to QUAPI, Persistent MPS, TEMPO, and Process Tensors

Victor S. Batista

Abstract

This tutorial develops the Feynman–Vernon influence functional in a deliberately step-by-step, time-sliced form for a quantum system linearly coupled to a harmonic bath. The derivation begins with the reduced density matrix, introduces the forward and backward system histories required by density-matrix propagation, and applies a symmetric Trotter splitting. Keeping the standard quadratic counterterm inside the conditional bath Hamiltonian makes every bath mode a displaced harmonic oscillator with unchanged curvature and minimum energy. Because both the conditional propagators and the initial thermal density matrix are Gaussian, all bath coordinates can then be eliminated exactly for the chosen discretization. The result is a quadratic temporal interaction between different system slices, organized as a causal lower-triangular product of pair factors. Truncating this triangle to a finite memory band gives the augmented-density-tensor structure used by QUAPI and the temporal network compressed by TEMPO.

The derivation is followed by a worked Ohmic-qubit example that connects each mathematical object to an executable implementation at three levels: an explicit dense finite-memory ADT, a pedagogical TT–SVD calculation that compresses and reconstructs the ADT at every step, and a persistent matrix-product-state implementation that never constructs the global dense ADT. The persistent update is derived locally by threading the newest forward/backward path-pair value through MPS virtual bonds, applying lagged influence factors, summing the expired history site at the boundary, and performing QR/SVD canonicalization and compression. The numerical section retains the full reduced-state and compression diagnostics and separate convergence sweeps in memory depth, Trotter step, and MPS bond dimension. A final main-text part then shows how the same finite-memory influence network can be reorganized into a process tensor with open intervention slots, allowing different preparations, pulses, measurements, and multitime spectroscopy protocols to be contracted with the same stored environmental influence. The appendices retain the foundational real- and imaginary-time propagator derivations, the harmonic-oscillator thermal kernel, the bath correlation function, the complete code map, the convergence tables, and the full executable source.

Contents

How to use this tutorial

The tutorial is intentionally complete: the main text develops the influence functional from the Hamiltonian through finite-memory QUAPI/TEMPO propagation and then to the reusable process-tensor construction, while the appendices retain the foundational propagator derivations and the complete executable implementation. It is therefore useful to distinguish four reading paths.

> **Learning goal.** By the end of the tutorial, the reader should be able to explain why reduced-density-matrix propagation requires two histories; derive the discrete influence functional by explicitly eliminating a harmonic bath; identify the origin of the lower-triangular time-pair tensor; implement the moving finite-memory QUAPI update; and distinguish a dense ADT, an MPS-compressed representation, a persistent MPS, and a process tensor.

First reading.
Follow Parts I–IV and skip the appendices on the first pass. This route gives the physical logic, the full influence-functional derivation, the numerical QUAPI/TEMPO example, and the extension to process tensors without interrupting the main argument.

Derivation-first.
Read the main text together with Appendices A–C. These appendices supply the real-time propagator, imaginary-time harmonic-oscillator kernel, and thermal bath correlation function used in the main derivation.

Implementation-first.
Read through the finite-memory construction and the worked example, especially Sec. 15.5, then use Appendix D to map each mathematical operation onto the dense ADT, TT–SVD baseline, and persistent-MPS routines. Use Appendix E for the complete convergence values and Appendix G for the full executable source.

Process-tensor path.
Read Sec. 13, Sec. 15.5, and then Sec. 17. This route follows the temporal-memory idea from the banded influence functional, through MPS/MPO compression, to the open input–output slots of the process tensor.

Notation for the derivation

Symbol	Meaning
N	Number of system propagation steps; the system times are $t_k = k\Delta t$, $k = 0, \ldots, N$.
k, k'	Discrete time-slice indices. The causal influence tensor is organized with $0 \le k' \le k \le N$.
K	Memory depth in time steps; the retained lag satisfies $k - k' \le K$, with physical memory time $\tau_c \simeq K\Delta t$.
τ_k	Width of the conditional bath interval: $\Delta t/2$ at the two endpoints and Δt in the interior.
$s_k^{\pm}$	System coupling-coordinate values on the forward (+) and backward (−) density-matrix branches.
$x_{\alpha,k}^{\pm}$	Coordinate of bath mode α at slice k on the two branches.
Λ	Counterterm coefficient, $\Lambda = \sum_\alpha c_\alpha^2/(2m_\alpha\omega_\alpha^2)$.
$C(t)$	Thermal bath-force correlation function, $\langle \hat{B}(t)\hat{B}(0)\rangle_B$.
$\alpha(t)$	Correlation kernel defined by $C(t) = \hbar\alpha(t)$. This function is unrelated to the bath-mode index α.
$\eta_{kk'}$	Cell-integrated complex influence coefficient coupling slices k and k'.
$I_{kk'}$	Pair influence factor associated with $\eta_{kk'}$.

Where the approximations enter

A useful way to read the derivation is to separate exact algebra from controlled numerical approximations. Once the time grid and the initial factorized state have been chosen, the elimination of a linearly coupled harmonic bath is Gaussian and is carried out exactly. The approximations tested later in the tutorial are instead

$$\begin{array}{rcl} \text{time discretization} & : & \Delta t, \\ \text{finite bath memory} & : & K, \\ \text{numerical integration} & : & \text{frequency and cell quadratures}, \\ \text{temporal compression} & : & \varepsilon_{\mathrm{MPS}},\ \chi_{\max}. \end{array} \tag{1}$$

These errors are conceptually distinct and must be converged separately. The numerical example returns to this hierarchy explicitly.

Part I
Constructing the discrete reduced dynamics

1 What we will derive

> **Learning goal.** Keep two destination formulas in view while reading this part: the reduced-state expression in Eq. (3) and the pair-factor form of the influence functional in Eq. (4). Every intermediate step is introduced only to explain where one of the ingredients in these two formulas comes from.

Suppose a small quantum system is coupled to a large harmonic environment. We are interested only in the reduced system density matrix

$$\hat{\rho}_S(t) = \mathrm{Tr}_B\, \hat{\rho}(t), \tag{2}$$

not in the detailed state of every bath oscillator. The central goal of this tutorial is to derive an explicit time-sliced expression for the *time-evolved reduced density matrix* after the bath has been integrated out.

More precisely, for a final time $t_N = N\Delta t$, we will derive an expression of the form

$$\boxed{\begin{aligned}\rho_S(s_N^+, s_N^-; t_N) = \sum_{s_0^+, s_0^-} \rho_S(s_0^+, s_0^-; 0) \sum_{\{s_k^+\}_{k=1}^{N-1}} \sum_{\{s_k^-\}_{k=1}^{N-1}} \\ \times \prod_{k=0}^{N-1} K_k(s_{k+1}^+, s_k^+) K_k^*(s_{k+1}^-, s_k^-)\, \mathcal{F}[\mathbf{s}^+, \mathbf{s}^-]\end{aligned}} \tag{3}$$

Here and below, equality for the time-sliced reduced state is understood up to the global second-order error of the symmetric Trotter splitting at fixed t_N. The factors K_k are the short-time propagators of the isolated system, while all effects of the environment are contained in the Feynman–Vernon influence functional $\mathcal{F}[\mathbf{s}^+, \mathbf{s}^-]$.

It is helpful to read Eq. (3) as three nested pieces. The initial matrix element $\rho_S(s_0^+, s_0^-; 0)$ specifies how the two histories start; the product of K_k and K_k^* propagates those histories locally from one time slice to the next; and $\mathcal{F}$ assigns the nonlocal environmental weight to the complete pair of histories. The rest of Parts I–II is a derivation of that last object.

This separation is the main conceptual result. The isolated-system propagators are local in time: the factor K_k connects only two neighboring system coordinates, s_k and s_{k+1}. By contrast, after the bath is eliminated, the influence functional couples system coordinates belonging to *different* time slices. For the harmonic bath considered here, we will show that it can be written as

$$\boxed{\mathcal{F}[\mathbf{s}^+, \mathbf{s}^-] = \prod_{k=0}^{N} \prod_{k'=0}^{k} I_{kk'},} \tag{4}$$

where each factor $I_{kk'}$ describes the bath-mediated coupling between the system at the two discrete times associated with k and k'. Thus the bath does not merely modify the instantaneous propagation of the system: it generates an effective *memory interaction in time*.

The pair structure in Eq. (4) is lower triangular because the interaction can be organized causally with $k' \le k$. If the bath correlation function has decayed sufficiently after K time steps, the influence

functional can then be approximated by retaining only

$$\boxed{k - k' \leq K,} \tag{5}$$

which turns the full triangular interaction tensor into a finite-width memory band. This is the central structure exploited explicitly by QUAPI and compressed as a temporal tensor network by TEMPO.

The logical flow of the derivation is therefore

$$\begin{array}{c} \hat{\rho}_S(0) \longrightarrow \text{forward/backward system histories} \longrightarrow \text{symmetric Trotter splitting} \\ \downarrow \\ \text{conditional displaced bath oscillators} \longrightarrow \text{exact Gaussian bath elimination} \\ \downarrow \\ \mathcal{F}[\mathbf{s}^+, \mathbf{s}^-] = \prod_{k \geq k'} I_{kk'} \longrightarrow \text{temporal memory couplings} \\ \downarrow \\ \rho_S(s_N^+, s_N^-; t_N) \text{ in terms of } \mathcal{F} \longrightarrow \text{finite-memory QUAPI/TEMPO form.} \end{array} \tag{6}$$

Stage	**Question answered**
Parts I–II	How does the microscopic system–bath Hamiltonian become a discrete influence functional coupling different system times?
Part III, first half	How does truncating those temporal couplings produce the finite-memory augmented density tensor used by QUAPI?
Part III, second half	How can the same finite-memory object be stored and propagated persistently as an MPS without reconstructing the dense ADT?
Appendices A–C	Where do the propagator, thermal-kernel, and bath-correlation formulas used in the main derivation come from?
Appendices D–G	How are the mathematical operations implemented numerically, and what are the complete convergence data and executable source?

Two points will be emphasized throughout.

1. The counterterm is kept *inside* the conditional bath Hamiltonian. It is therefore already present in the bath propagators and must not be added again later.

2. Once the short-time Trotter discretization has been chosen, the integration over the linearly coupled harmonic bath is Gaussian and can be carried out exactly. The nonlocal coupling between different system times is therefore generated explicitly by eliminating the bath degrees of freedom.

2 Model: system plus a harmonic bath

Physical picture. The bath is a collection of independent harmonic coordinates x_α. The system does not need to "know" each coordinate separately: it couples through the collective bath force $\hat{B} = \sum_\alpha c_\alpha \hat{x}_\alpha$. The harmonic and linear structure is what will ultimately make the bath elimination Gaussian and therefore analytically tractable.

We consider a model Hamiltonian of a system linearly coupled to a harmonic bath,

$$\boxed{\hat{H} = \hat{H}_S + \hat{H}_{BC},} \tag{7}$$

with the bath-plus-coupling to the system

$$\hat{H}_{BC} = \sum_\alpha \left[\frac{\hat{p}_\alpha^2}{2m_\alpha} + \frac{1}{2} m_\alpha \omega_\alpha^2 \hat{x}_\alpha^2 - c_\alpha \hat{s}\, \hat{x}_\alpha + \frac{c_\alpha^2}{2m_\alpha \omega_\alpha^2} \hat{s}^2 \right]. \tag{8}$$

It is useful to define

$$\hat{H}_B = \sum_\alpha \left(\frac{\hat{p}_\alpha^2}{2m_\alpha} + \frac{1}{2} m_\alpha \omega_\alpha^2 \hat{x}_\alpha^2 \right), \tag{9}$$

$$\hat{B} = \sum_\alpha c_\alpha \hat{x}_\alpha, \qquad \Lambda = \sum_\alpha \frac{c_\alpha^2}{2m_\alpha \omega_\alpha^2}, \tag{10}$$

so that

$$\boxed{\hat{H} = \hat{H}_S + \hat{H}_B - \hat{s}\hat{B} + \Lambda \hat{s}^2.} \tag{11}$$

Note that $\hat{H}_S$ is the *bare physical system Hamiltonian*, which does not include the counterterm Λs^2.

2.1 Why the counterterm is present

The purpose of the counterterm becomes obvious by examining one oscillator. For a fixed numerical value s of the system coupling coordinate,

$$\begin{aligned} &\frac{1}{2} m_\alpha \omega_\alpha^2 x_\alpha^2 - c_\alpha s x_\alpha + \frac{c_\alpha^2 s^2}{2m_\alpha \omega_\alpha^2} \\ &\qquad = \frac{1}{2} m_\alpha \omega_\alpha^2 \left(x_\alpha - \frac{c_\alpha s}{m_\alpha \omega_\alpha^2} \right)^2. \end{aligned} \tag{12}$$

Define

$$d_\alpha(s) = \frac{c_\alpha s}{m_\alpha \omega_\alpha^2}. \tag{13}$$

Then the oscillator potential is simply shifted from $x_\alpha = 0$ to $x_\alpha = d_\alpha(s)$.

Without the counterterm,

$$\frac{1}{2} m_\alpha \omega_\alpha^2 x_\alpha^2 - c_\alpha s x_\alpha = \frac{1}{2} m_\alpha \omega_\alpha^2 (x_\alpha - d_\alpha)^2 - \frac{c_\alpha^2 s^2}{2m_\alpha \omega_\alpha^2}, \tag{14}$$

so the coupling would also lower the minimum energy by an s-dependent amount. The counterterm cancels precisely this static lowering.

> **Key idea.** With the counterterm included, the system does not change the curvature or minimum energy of a bath oscillator. It changes only the oscillator's equilibrium position. Different system histories therefore drive the bath through different sequences of *displacements*.

> **Physical picture.** A useful mental picture is a spring whose stiffness is unchanged while its equilibrium position is moved by the instantaneous system value s. The bath records the system history because different values of s_k shift the spring to different centers at different times.

3 Reduced dynamics requires two histories

Assume an initially factorized state,

$$\hat{\rho}(0) = \hat{\rho}_S(0) \otimes \hat{\rho}_B^{\mathrm{eq}}, \qquad \hat{\rho}_B^{\mathrm{eq}} = \frac{e^{-\beta \hat{H}_B}}{Z_B}, \qquad Z_B = \mathrm{Tr}_B\, e^{-\beta \hat{H}_B}. \tag{15}$$

The reduced state is [1, 5]

$$\boxed{\hat{\rho}_S(t) = \mathrm{Tr}_B \left[e^{-\mathrm{i}\hat{H}t/\hbar} \hat{\rho}(0) e^{+\mathrm{i}\hat{H}t/\hbar} \right].} \tag{16}$$

Equation (16) contains a forward propagator and a backward propagator. This is the origin of the two paths. A matrix element $\rho_S(s^+, s^-)$ has a ket label and a bra label, so propagating the density matrix necessarily propagates one history on each side of $\hat{\rho}(0)$. The two histories are therefore not an additional assumption; they are bookkeeping forced by density-matrix evolution. Working in an eigenbasis of the coupling operator,

$$\hat{s}|s\rangle = s|s\rangle, \tag{17}$$

we will eventually sum over

$$\mathbf{s}^+ = (s_0^+, s_1^+, \ldots, s_N^+), \qquad \mathbf{s}^- = (s_0^-, s_1^-, \ldots, s_N^-). \tag{18}$$

The + history belongs to the ket branch and the − history to the bra branch. For a discrete coupling spectrum the path measures are sums, as written below; for a continuous spectrum, each sum over $s_k^{\pm}$ is replaced by the corresponding integral. The factorized preparation in Eq. (15) is an explicit assumption: initial system–bath correlations require an additional imaginary-time preparation branch and are outside the scope of this derivation.

4 Symmetric time slicing

Divide the interval $[0, t_N]$ into N system time steps,

$$t_k = k\Delta t, \qquad k = 0, \ldots, N, \qquad t_N = N\Delta t. \tag{19}$$

For one short step, use the symmetric Trotter factorization

$$\boxed{e^{-\mathrm{i}\hat{H}\Delta t/\hbar} = e^{-\mathrm{i}\hat{H}_{BC}\Delta t/(2\hbar)} e^{-\mathrm{i}\hat{H}_S \Delta t/\hbar} e^{-\mathrm{i}\hat{H}_{BC}\Delta t/(2\hbar)} + \mathcal{O}(\Delta t^3).} \tag{20}$$

The local splitting error is $\mathcal{O}(\Delta t^3)$; over a fixed total propagation time, the accumulated Trotter error is second order in Δt. The coordinate-space foundation of this construction—from the free-particle short-time kernel through the symmetric Trotter propagator and the finite-time path-integral limit—is derived in Appendix A. That appendix is not required for the bath elimination below, but it provides the underlying propagator derivation for readers who want to see where the short-time kernels come from.

> **Physical picture.** The symmetric splitting is useful here for a structural reason as well as for its second-order global accuracy. Each system propagation step is bracketed by two half bath steps. When consecutive steps are multiplied, the adjacent half bath steps join into a full bath interval centered on the intermediate system value. This is why the discretized bath naturally lives on $N + 1$ cells while the system propagates through only N steps.

When $\hat{H}_{BC}$ acts next to an eigenstate $|s_k\rangle$, the operator $\hat{s}$ inside $\hat{H}_{BC}$ becomes the number s_k. The bath therefore evolves under the conditional Hamiltonian

$$\boxed{\hat{H}_{BC}(s_k) = \hat{H}_B - s_k\hat{B} + \Lambda s_k^2.} \tag{21}$$

Equivalently,

$$\boxed{\hat{H}_{BC}(s_k) = \sum_\alpha \left[\frac{\hat{p}_\alpha^2}{2m_\alpha} + \frac{1}{2}m_\alpha\omega_\alpha^2\left(\hat{x}_\alpha - \frac{c_\alpha s_k}{m_\alpha\omega_\alpha^2}\right)^2\right].} \tag{22}$$

Equation (22) is the central physical simplification of the model: at each time slice, the bath is a set of ordinary harmonic oscillators with system-dependent equilibrium positions.

4.1 Why there are $N+1$ conditional bath intervals

The symmetric factorization places a half bath step on each side of every system step. After multiplying the N short-time propagators, neighboring half steps with the same intermediate value s_k combine. Consequently the bath experiences

$$\tau_k = \begin{cases} \Delta t/2, & k = 0, N, \\ \Delta t, & k = 1, \ldots, N-1. \end{cases} \tag{23}$$

There are therefore $N+1$ conditional bath intervals, even though there are only N system propagation steps. For example, if $N = 2$, the bath sequence is a half interval at s_0, a full interval at s_1, and a final half interval at s_2: three conditional bath intervals for two system steps. Defining

$$\hat{U}_k(s_k;\tau_k) = \exp\left[-\frac{\mathrm{i}\tau_k}{\hbar}\hat{H}_{BC}(s_k)\right], \tag{24}$$

the complete conditional bath propagator on the forward branch is

$$\hat{\mathcal{U}}_B[\mathbf{s}^+] = \hat{U}_N(s_N^+;\tau_N)\cdots\hat{U}_1(s_1^+;\tau_1)\hat{U}_0(s_0^+;\tau_0). \tag{25}$$

The isolated-system short-time propagator is

$$K_k(s_{k+1}, s_k) = \left\langle s_{k+1}\middle| e^{-\mathrm{i}\hat{H}_S\Delta t/\hbar}\middle| s_k\right\rangle. \tag{26}$$

For coordinate-space Hamiltonians, Appendix A derives the corresponding short-time kernel explicitly and shows how products of such kernels build the finite-time propagator.

5 The discrete influence functional

After inserting complete system resolutions between the Trotter factors, the reduced density matrix takes the form

$$\begin{aligned}\rho_S(s_N^+, s_N^-; t_N) = &\sum_{s_0^+, s_0^-} \rho_S(s_0^+, s_0^-; 0) \\ &\times \sum_{s_1^+,\ldots,s_{N-1}^+} \sum_{s_1^-,\ldots,s_{N-1}^-} \\ &\times \prod_{k=0}^{N-1} K_k(s_{k+1}^+, s_k^+)K_k^*(s_{k+1}^-, s_k^-) \\ &\times \mathcal{F}[\mathbf{s}^+, \mathbf{s}^-],\end{aligned} \tag{27}$$

Checkpoint. The system propagation is now separated into two pieces: neighboring time slices are connected locally by the isolated-system kernels, while every nonlocal environmental effect is contained in a single object, $\mathcal{F}[\mathbf{s}^+, \mathbf{s}^-]$.

where all bath effects are contained in

$$\boxed{\mathcal{F}[\mathbf{s}^+, \mathbf{s}^-] = \mathrm{Tr}_B \left[\hat{\mathcal{U}}_B[\mathbf{s}^+]\hat{\rho}_B^{\mathrm{eq}}\hat{\mathcal{U}}_B^\dagger[\mathbf{s}^-]\right]} \tag{28}$$

If the two histories coincide,

$$\mathbf{s}^+ = \mathbf{s}^- = \mathbf{s}, \tag{29}$$

then unitarity immediately gives

$$\boxed{\mathcal{F}[\mathbf{s}, \mathbf{s}] = 1.} \tag{30}$$

This is an important implementation check.

Physical picture. For a fixed pair of system histories, $\mathcal{F}$ is a single complex number. Its magnitude changes the relative weight of that pair of histories and its phase changes how that pair interferes with others. The bath coordinates no longer appear explicitly once this number has been evaluated.

Part II

Exact elimination of the harmonic bath

Part I established the forward–backward system histories and isolated all environmental effects in the discrete influence functional. We now eliminate the bath coordinates explicitly. The central point is that, for the linear harmonic environment considered here, this elimination is Gaussian. No new physical approximation is introduced in this step beyond the time slicing already chosen.

> **Learning goal.** The goal of Part II is to start from the operator trace in Eq. (28) and end with a bath-free quadratic functional of the discrete variables $s_k^{\pm}$. The bath elimination itself is exact for the chosen time grid; the important task is therefore to keep the forward/backward coordinates, endpoint trace, and source terms organized.

6 Bath trace in coordinates

Let $x = \{x_\alpha\}$ denote all bath coordinates. For each conditional interval define

$$\boxed{U_k^+(x_{k+1}, x_k; s_k^+) = \left\langle x_{k+1} \middle| \exp\left[-\frac{\mathrm{i}\tau_k}{\hbar}\left(\hat{H}_B - s_k^+ \hat{B} + \Lambda (s_k^+)^2\right)\right] \middle| x_k \right\rangle,} \tag{31}$$

and

$$\boxed{U_k^-(x_k, x_{k+1}; s_k^-) = \left\langle x_k \middle| \exp\left[+\frac{\mathrm{i}\tau_k}{\hbar}\left(\hat{H}_B - s_k^- \hat{B} + \Lambda (s_k^-)^2\right)\right] \middle| x_{k+1} \right\rangle.} \tag{32}$$

For the same value of s_k,

$$U_k^-(x_k, x_{k+1}; s_k) = \left[U_k^+(x_{k+1}, x_k; s_k)\right]^*. \tag{33}$$

Inserting coordinate resolutions between all $N+1$ conditional bath propagators gives

$$\begin{aligned} \mathcal{F}[\mathbf{s}^+, \mathbf{s}^-] = & \int dx_0^+\, dx_0^-\, \rho_B^{\mathrm{eq}}(x_0^+, x_0^-) \\ & \times \left[\prod_{j=1}^{N} \int dx_j^+\, dx_j^-\right] \int dx_{N+1} \\ & \times \prod_{k=0}^{N} U_k^+(x_{k+1}^+, x_k^+; s_k^+) U_k^-(x_k^-, x_{k+1}^-; s_k^-), \end{aligned} \tag{34}$$

with the trace condition

$$\boxed{x_{N+1}^+ = x_{N+1}^- \equiv x_{N+1}.} \tag{35}$$

The coordinate topology is therefore

$$\begin{array}{ccccccccc} x_0^+ & \xrightarrow{U_0^+} & x_1^+ & \longrightarrow & \cdots & \longrightarrow & x_N^+ & \xrightarrow{U_N^+} & x_{N+1} \\ x_0^- & \xleftarrow{U_0^-} & x_1^- & \longleftarrow & \cdots & \longleftarrow & x_N^- & \xleftarrow{U_N^-} & x_{N+1}. \end{array} \tag{36}$$

The initial coordinates are independent because the thermal density matrix can be off-diagonal. The final coordinate is common because a trace identifies the final bra and ket bath states.

Checkpoint. Count the variables before proceeding. For one bath mode there are $N+1$ forward coordinates before the common endpoint, $N+1$ backward coordinates before the common endpoint, and one shared final coordinate. After trace closure this gives the $2N+3$ variables assembled into $\mathbf{X}_\alpha$ below.

7 Conditional displaced oscillator: exact propagator of one bath mode

Learning goal. This section supplies the one ingredient needed for the Gaussian bath trace: the exact coordinate kernel for one oscillator during one conditional interval. Once that kernel is written as a Gaussian in its endpoint coordinates, the multimode bath follows by multiplication over independent modes.

Because the bath modes are independent, consider one mode α. For fixed s_k,

$$\hat{H}_{BC,\alpha}(s_k) = \frac{\hat{p}_\alpha^2}{2m_\alpha} + \frac{1}{2}m_\alpha\omega_\alpha^2\hat{x}_\alpha^2 - c_\alpha s_k\hat{x}_\alpha + \frac{c_\alpha^2 s_k^2}{2m_\alpha\omega_\alpha^2}. \tag{37}$$

Define

$$d_{\alpha k} = \frac{c_\alpha s_k}{m_\alpha\omega_\alpha^2}. \tag{38}$$

Then [5]

$$\boxed{\hat{H}_{BC,\alpha}(s_k) = \frac{\hat{p}_\alpha^2}{2m_\alpha} + \frac{1}{2}m_\alpha\omega_\alpha^2(\hat{x}_\alpha - d_{\alpha k})^2.} \tag{39}$$

Let $K_{\alpha,0}(x_f, x_i; \tau)$ be the ordinary real-time harmonic-oscillator kernel. Equation (39) gives immediately

$$\boxed{U_{\alpha,k}^+(x_f, x_i; s_k) = K_{\alpha,0}(x_f - d_{\alpha k}, x_i - d_{\alpha k}; \tau_k).} \tag{40}$$

The real-time harmonic-oscillator kernel required here is obtained by analytic continuation in Appendix B, specifically Sec. B.7; see Eq. (288). Substituting that kernel gives

$$\begin{aligned} U_{\alpha,k}^+(x_f, x_i; s_k) = &\sqrt{\frac{m_\alpha\omega_\alpha}{2\pi\mathrm{i}\hbar\sin(\omega_\alpha\tau_k)}} \\ &\times \exp\left\{\frac{\mathrm{i}m_\alpha\omega_\alpha}{2\hbar\sin(\omega_\alpha\tau_k)}\Big[\big((x_f - d_{\alpha k})^2 + (x_i - d_{\alpha k})^2\big)\cos(\omega_\alpha\tau_k)\right. \\ &\left.\qquad - 2(x_f - d_{\alpha k})(x_i - d_{\alpha k})\Big]\right\}. \end{aligned} \tag{41}$$

7.1 Expanded form: the exact Gaussian source

For Gaussian integration it is useful to re-expand the shifted kernel in the original coordinates. This step separates three roles cleanly: the free oscillator kernel supplies the source-independent quadratic form, the term linear in $x_f + x_i$ carries the system history as a Gaussian source, and the remaining

s_k^2 term is a coordinate-independent phase. One finds exactly

$$U^+_{\alpha,k}(x_f, x_i; s_k) = K_{\alpha,0}(x_f, x_i; \tau_k) \times \exp\left[\frac{\mathrm{i}c_\alpha s_k}{\hbar\omega_\alpha}\tan\left(\frac{\omega_\alpha\tau_k}{2}\right)(x_f + x_i)\right] \times \exp\left[-\frac{\mathrm{i}c_\alpha^2 s_k^2}{m_\alpha\hbar\omega_\alpha^3}\tan\left(\frac{\omega_\alpha\tau_k}{2}\right)\right]. \tag{42}$$

The short-time limit is instructive. Since

$$\tan\left(\frac{\omega_\alpha\tau_k}{2}\right) = \frac{\omega_\alpha\tau_k}{2} + \mathcal{O}(\tau_k^3), \tag{43}$$

the exact linear source becomes

$$\frac{\mathrm{i}c_\alpha s_k\tau_k}{2\hbar}(x_f + x_i) + \mathcal{O}(\tau_k^3), \tag{44}$$

which is precisely the trapezoidal/endpoint average expected for a constant force acting over a short interval.

7.2 Thermal density matrix of one free bath mode

The initial thermal density matrix of one oscillator is also Gaussian. Its normalized coordinate-space form is derived in Appendix B, Sec. B.10; see Eq. (304):

$$\rho_{\beta,\alpha}(x_0^+, x_0^-) = \sqrt{\frac{m_\alpha\omega_\alpha}{\pi\hbar}\tanh\left(\frac{\beta\hbar\omega_\alpha}{2}\right)} \times \exp\left\{-\frac{m_\alpha\omega_\alpha}{4\hbar}\left[(x_0^+ - x_0^-)^2\coth\left(\frac{\beta\hbar\omega_\alpha}{2}\right) + (x_0^+ + x_0^-)^2\tanh\left(\frac{\beta\hbar\omega_\alpha}{2}\right)\right]\right\}. \tag{45}$$

Thus every bath-coordinate dependence in Eq. (34) is Gaussian. Appendix B also derives the oscillator partition function and its extension to the full multimode bath, completing the construction of the normalized initial bath state used here.

8 Why the bath elimination is Gaussian

This section is the algebraic bottleneck of the derivation. The objective is not to write every entry of a large Gaussian matrix explicitly, but to expose the structure that matters: all bath coordinates appear quadratically, while the system history enters only through a linear source and a local quadratic term. Once this structure is recognized, the bath can be removed by one completion of the square.

> **Physical picture.** The matrix notation below is simply the multidimensional version of the elementary Gaussian integral $\int dx\, \exp[-ax^2/2 + jx]$. The vector $\mathbf{X}_\alpha$ collects the coordinates being integrated, A_α contains all terms quadratic in those coordinates, $\mathbf{J}_\alpha$ contains the linear "force" generated by the system history, and Q_α contains history-dependent terms with no remaining bath coordinate.

For one mode, impose the trace closure from the start and define

$$\mathbf{X}_\alpha = \begin{pmatrix} x^+_{\alpha,0} \\ x^+_{\alpha,1} \\ \vdots \\ x^+_{\alpha,N} \\ x_{\alpha,N+1} \\ x^-_{\alpha,0} \\ x^-_{\alpha,1} \\ \vdots \\ x^-_{\alpha,N} \end{pmatrix}, \qquad \dim \mathbf{X}_\alpha = 2N+3. \tag{46}$$

All source-independent quadratic terms can be collected into a complex symmetric matrix A_α. The exact linear source is most transparently written through

$$\begin{aligned} \mathbf{J}_\alpha^{\mathsf{T}} \mathbf{X}_\alpha = \sum_{k=0}^{N} & \frac{\mathrm{i}c_\alpha}{\hbar\omega_\alpha} \tan\left(\frac{\omega_\alpha \tau_k}{2}\right) \\ & \times \left[s_k^+ (x^+_{\alpha,k} + x^+_{\alpha,k+1}) - s_k^- (x^-_{\alpha,k} + x^-_{\alpha,k+1}) \right], \end{aligned} \tag{47}$$

where $x^+_{\alpha,N+1} = x^-_{\alpha,N+1} = x_{\alpha,N+1}$.

The source-dependent scalar term from Eq. (42) is

$$Q_\alpha[\mathbf{s}^+, \mathbf{s}^-] = -\sum_{k=0}^{N} \frac{\mathrm{i}c_\alpha^2}{m_\alpha \hbar \omega_\alpha^3} \tan\left(\frac{\omega_\alpha \tau_k}{2}\right) \left[(s_k^+)^2 - (s_k^-)^2 \right]. \tag{48}$$

The normalized one-mode influence functional therefore has the exact form

$$\boxed{\mathcal{F}_\alpha = \frac{\displaystyle\int \mathrm{d}\mathbf{X}_\alpha \, \exp\left[-\frac{1}{2} \mathbf{X}_\alpha^{\mathsf{T}} A_\alpha \mathbf{X}_\alpha + \mathbf{J}_\alpha^{\mathsf{T}} \mathbf{X}_\alpha + Q_\alpha \right]}{\displaystyle\int \mathrm{d}\mathbf{X}_\alpha \, \exp\left[-\frac{1}{2} \mathbf{X}_\alpha^{\mathsf{T}} A_\alpha \mathbf{X}_\alpha \right]}.} \tag{49}$$

Real-time kernels give oscillatory Fresnel Gaussians. Equation (49) is understood with the usual infinitesimal convergence prescription, or equivalently by analytic continuation from a convergent Gaussian.

9 Eliminating all bath coordinates exactly

The payoff of the previous bookkeeping is that the remaining integration is a standard matrix Gaussian. The same algebra that shifts a one-dimensional Gaussian by j/a now shifts the coordinate vector by $A_\alpha^{-1}\mathbf{J}_\alpha$. Complete the square:

$$\begin{aligned} -\frac{1}{2} \mathbf{X}_\alpha^{\mathsf{T}} A_\alpha \mathbf{X}_\alpha + \mathbf{J}_\alpha^{\mathsf{T}} \mathbf{X}_\alpha = & -\frac{1}{2} (\mathbf{X}_\alpha - A_\alpha^{-1} \mathbf{J}_\alpha)^{\mathsf{T}} A_\alpha (\mathbf{X}_\alpha - A_\alpha^{-1} \mathbf{J}_\alpha) \\ & + \frac{1}{2} \mathbf{J}_\alpha^{\mathsf{T}} A_\alpha^{-1} \mathbf{J}_\alpha. \end{aligned} \tag{50}$$

The shift of integration variables leaves the Gaussian normalization unchanged, so

$$\boxed{\mathcal{F}_\alpha = \exp\left[Q_\alpha + \frac{1}{2}\mathbf{J}_\alpha^{\mathsf{T}} A_\alpha^{-1}\mathbf{J}_\alpha\right].} \tag{51}$$

Checkpoint. All bath coordinates have now disappeared. The environment survives only through $Q_\alpha + \frac{1}{2}\mathbf{J}_\alpha^{\mathsf{T}} A_\alpha^{-1}\mathbf{J}_\alpha$, which is a quadratic functional of the discrete forward and backward system histories.

Independent bath modes multiply:

$$\mathcal{F} = \prod_\alpha \mathcal{F}_\alpha, \qquad \ln\mathcal{F} = \sum_\alpha \ln\mathcal{F}_\alpha. \tag{52}$$

This is the central algebraic result. Because $\mathbf{J}_\alpha$ is linear in the forward/backward system history and Q_α is already quadratic in that history, $\ln\mathcal{F}$ must be a quadratic form in the discrete values $s_k^\pm$.

9.1 What is A_α^{-1}?

Define the normalized zero-source closed-contour Gaussian average

$$\langle\mathcal{O}(\mathbf{X}_\alpha)\rangle_{0,C} = \frac{\int \mathrm{d}\mathbf{X}_\alpha\, \mathcal{O}(\mathbf{X}_\alpha) e^{-\frac{1}{2}\mathbf{X}_\alpha^{\mathsf{T}} A_\alpha \mathbf{X}_\alpha}}{\int \mathrm{d}\mathbf{X}_\alpha\, e^{-\frac{1}{2}\mathbf{X}_\alpha^{\mathsf{T}} A_\alpha \mathbf{X}_\alpha}}. \tag{53}$$

Differentiating its generating function gives

$$\boxed{(A_\alpha^{-1})_{ij} = \langle X_{\alpha,i} X_{\alpha,j}\rangle_{0,C}.} \tag{54}$$

Thus A_α^{-1} is the free oscillator coordinate covariance on the closed forward–backward contour, with the thermal initial condition and final trace closure already built in.

It is important not to identify the final slice–slice influence coefficient with a bare entry of A_α^{-1}. The physical time-slice kernel is obtained after A_α^{-1} is contracted with the finite-interval source map in Eq. (47), together with the local term Q_α.

More explicitly, the four branch blocks of this closed-contour covariance are the ordinary thermal two-point functions with the appropriate ordering:

$$\begin{aligned} G_\alpha^{++}(t,t') &= \langle \mathcal{T}\, x_\alpha(t) x_\alpha(t')\rangle_\beta, & G_\alpha^{--}(t,t') &= \langle \widetilde{\mathcal{T}}\, x_\alpha(t) x_\alpha(t')\rangle_\beta, \\ G_\alpha^{-+}(t,t') &= \langle x_\alpha(t) x_\alpha(t')\rangle_\beta, & G_\alpha^{+-}(t,t') &= \langle x_\alpha(t') x_\alpha(t)\rangle_\beta, \end{aligned} \tag{55}$$

where $\mathcal{T}$ and $\widetilde{\mathcal{T}}$ denote time and anti-time ordering. Contracting these four blocks with the two branch sources in Eq. (47) produces the retarded lower-triangular combination in Eq. (69). Thus the passage from the finite-dimensional Gaussian integral to the usual bath-correlation coefficients is an evaluation of A_α^{-1}, not an additional approximation.

Physical picture. This is the bridge from linear algebra to open-system physics. Instead of carrying the large inverse matrix element by element, we recognize its branch blocks as thermal correlation functions. Stationarity then packages the same information into a function of time differences, which is much more natural for constructing the discrete coefficients $\eta_{kk'}$.

10 How the bath correlation function emerges

Define the thermal bath-force correlation function

$$C(t) = \langle \hat{B}(t)\hat{B}(0)\rangle_B = \sum_\alpha c_\alpha^2 \langle \hat{x}_\alpha(t)\hat{x}_\alpha(0)\rangle_\beta. \tag{56}$$

Appendix C derives this function explicitly from the thermal oscillator occupations, converts the discrete mode sum to the spectral-density form used below, and relates its real and imaginary parts to bath fluctuations and linear response; the final spectral representation is Eq. (323). For definiteness, use the spectral-density convention

$$J(\omega) = \frac{\pi}{2}\sum_\alpha \frac{c_\alpha^2}{m_\alpha \omega_\alpha}\delta(\omega - \omega_\alpha). \tag{57}$$

Then

$$\boxed{\alpha(t) = \frac{1}{\pi}\int_0^\infty \mathrm{d}\omega\, J(\omega)\left[\coth\left(\frac{\beta\hbar\omega}{2}\right)\cos(\omega t) - \mathrm{i}\sin(\omega t)\right],} \tag{58}$$

and

$$\boxed{C(t) = \hbar\alpha(t).} \tag{59}$$

Stationarity implies

$$\alpha(-t) = \alpha^*(t). \tag{60}$$

With the same convention,

$$\boxed{\Lambda = \frac{1}{\pi}\int_0^\infty \mathrm{d}\omega\, \frac{J(\omega)}{\omega}.} \tag{61}$$

11 From the bath correlation function to the discrete influence coefficients

The $N+1$ conditional bath intervals can be represented as time cells. This is the symmetric-cell form of the discretized influence functional used in QUAPI [2, 3]:

$$\mathcal{C}_k = \begin{cases} [0, \Delta t/2], & k = 0, \\ [t_k - \Delta t/2, t_k + \Delta t/2], & k = 1, \ldots, N-1, \\ [t_N - \Delta t/2, t_N], & k = N. \end{cases} \tag{62}$$

Their widths are exactly τ_k from Eq. (23); they are contiguous, nonoverlapping, and cover $[0, t_N]$.

> **Physical picture.** The continuous correlation function tells us how strongly two infinitesimal times are coupled. The coefficient $\eta_{kk'}$ is the corresponding coupling after those infinitesimal times have been integrated over the two finite Trotter cells associated with slices k and k'. This is why $\eta_{kk'}$ is the natural object for a time-sliced algorithm.

It is useful first to define the contribution arising from the fluctuating bath force. For $k > k'$,

$$\boxed{\eta_{kk'}^{(B)} = \int_{\mathcal{C}_k} \mathrm{d}t_1 \int_{\mathcal{C}_{k'}} \mathrm{d}t_2\, \alpha(t_1 - t_2).} \tag{63}$$

For the same cell, causality keeps only the time-ordered triangle:

$$\boxed{\eta_{kk}^{(B)} = \int_{\mathcal{C}_k} \mathrm{d}t_1 \int_{\mathcal{C}_k \cap (-\infty, t_1]} \mathrm{d}t_2 \, \alpha(t_1 - t_2).} \tag{64}$$

If $\mathcal{C}_k = [a_k, b_k]$, this is simply

$$\eta_{kk}^{(B)} = \int_{a_k}^{b_k} \mathrm{d}t_1 \int_{a_k}^{t_1} \mathrm{d}t_2 \, \alpha(t_1 - t_2). \tag{65}$$

The endpoint half cells need no ad hoc correction: their smaller widths are already built into the cell definitions.

11.1 Where the counterterm appears in the discrete tensor

Because the counterterm was included in $\hat{H}_{BC}$ before the Trotter splitting, its forward/backward contribution over the conditional intervals is

$$\exp\left\{ -\frac{\mathrm{i}\Lambda}{\hbar} \sum_{k=0}^{N} \tau_k \left[(s_k^+)^2 - (s_k^-)^2 \right] \right\}. \tag{66}$$

This is *not* an additional factor to be appended after the calculation. It is a way of identifying, inside the already-computed influence functional, the local part originating from the counterterm.

For compact QUAPI/TEMPO notation, absorb this local phase into the diagonal coefficient:

$$\boxed{\eta_{kk'} = \begin{cases} \eta_{kk'}^{(B)}, & k > k', \\ \eta_{kk}^{(B)} + \mathrm{i}\Lambda\tau_k, & k = k'. \end{cases}} \tag{67}$$

Indeed,

$$\begin{aligned} &-\frac{1}{\hbar}(s_k^+ - s_k^-) \left[(\mathrm{i}\Lambda\tau_k) s_k^+ - (-\mathrm{i}\Lambda\tau_k) s_k^- \right] \\ &\quad = -\frac{\mathrm{i}\Lambda\tau_k}{\hbar} \left[(s_k^+)^2 - (s_k^-)^2 \right], \end{aligned} \tag{68}$$

which is exactly the exponent in Eq. (66).

> **Key idea.** The counterterm has not disappeared. At the oscillator level it converts the forced oscillator into a pure displacement, Eq. (39). At the final discrete influence-functional level the same physics appears as the local imaginary shift $\mathrm{i}\Lambda\tau_k$ in the diagonal coefficient, Eq. (67). These are two representations of the same contribution and must not be counted twice.

12 The lower-triangular influence functional

The complete influence functional can now be written

$$\boxed{\mathcal{F}[\mathbf{s}^+, \mathbf{s}^-] = \exp\left\{ -\frac{1}{\hbar} \sum_{k=0}^{N} \sum_{k'=0}^{k} (s_k^+ - s_k^-) \left[\eta_{kk'} s_{k'}^+ - \eta_{kk'}^* s_{k'}^- \right] \right\}.} \tag{69}$$

Checkpoint. The microscopic bath has now become a temporal interaction network: each coefficient $\eta_{kk'}$ couples two system time slices, and causality organizes those couplings into a lower triangle with $k' \le k$.

Equivalently,

$$\boxed{\mathcal{F} = \prod_{k=0}^{N} \prod_{k'=0}^{k} I_{kk'},} \tag{70}$$

with

$$I_{kk'} = \exp\left\{ -\frac{1}{\hbar}(s_k^+ - s_k^-)\left[\eta_{kk'} s_{k'}^+ - \eta_{kk'}^* s_{k'}^-\right] \right\}. \tag{71}$$

The pair domain is

$$\boxed{\mathcal{T}_N = \{(k,k') : 0 \le k' \le k \le N\}.} \tag{72}$$

The lower triangle is not imposed by hand. It is the causal reorganization of the quadratic covariance produced by the Gaussian bath elimination.

For example, through slice $k = 2$ the allowed pairs are $(0,0)$; $(1,1), (1,0)$; and $(2,2), (2,1), (2,0)$. Writing the pairs row by row in the newer index k makes the causal triangular structure visible before any memory approximation is introduced.

13 Finite-memory truncation

If the integrated influence coefficients have become negligible beyond a correlation time

$$\tau_c \simeq K\Delta t, \tag{73}$$

then interactions with longer lags can be neglected. Decay of $\alpha(t)$ is a useful guide, but convergence must be checked with respect to K, especially at low temperature or for structured spectral densities. The retained set is

$$\boxed{\mathcal{T}_{N,K} = \{(k,k') \in \mathcal{T}_N : k - k' \le K\}.} \tag{74}$$

The truncated influence functional is

$$\boxed{\mathcal{F}^{(K)} = \prod_{k=0}^{N} \prod_{k'=\max(0,k-K)}^{k} I_{kk'}.} \tag{75}$$

Checkpoint. Finite bath memory is therefore a geometric truncation of the temporal interaction network: the full lower triangle is replaced by a moving band of width $K+1$. No absolute time index is frozen; the retained window advances with the current slice k.

Equivalently, with the lag $j = k - k'$,

$$\boxed{\mathcal{F}^{(K)} = \prod_{k=0}^{N} \prod_{j=0}^{\min(K,k)} I_{k,k-j}.} \tag{76}$$

Common pitfall. K limits a *lag*, not an absolute time index. At every new slice the retained window moves forward. Confusing these two statements freezes the memory window at the beginning of the trajectory and gives the wrong QUAPI tensor.

The distinction is important: K is a *maximum separation in time*, not a fixed upper bound on the absolute earlier index. Thus

$$\prod_{k} \prod_{k'=0}^{K} I_{kk'} \tag{77}$$

is generally wrong. The memory window must move forward with the current time k.

For a stationary bath on a uniform grid, away from the two endpoint cells,

$$\eta_{kk'} \simeq \eta_{k-k'}, \tag{78}$$

so the finite-memory approximation simply means retaining lags $j = 0, 1, \ldots, K$.

Part III
QUAPI, TEMPO, and numerical convergence

The bath has now been reduced to the discrete coefficients $\eta_{kk'}$ and the associated pair factors $I_{kk'}$. The remaining problem is computational: how should a finite band of temporal interactions be propagated without storing the entire system history? QUAPI and temporal tensor-network methods answer this question with the same finite-memory physics but different tensor representations. The notebook below makes the distinction explicit by comparing a dense ADT, an MPS used only as an intermediate compression, and a persistent MPS that remains the propagated history state.

> **Learning goal.** Part III separates two ideas that are easy to conflate: *finite memory* is a physical approximation controlled by K, whereas *tensor compression* is a representation choice controlled by singular values and bond dimensions. Dense QUAPI, reconstructing TT–SVD, and persistent MPS use the same retained influence factors; they differ only in how the history tensor is stored and updated.

14 Why finite memory leads to QUAPI and TEMPO

Define the path-pair index

$$a_k = (s_k^+, s_k^-). \tag{79}$$

With memory depth K, the newest pair a_k interacts only with

$$a_k, a_{k-1}, \ldots, a_{k-K}. \tag{80}$$

Once an older index lies more than K steps in the past, it cannot appear in any future influence factor and can be summed out. This is the basic reason QUAPI can propagate an augmented density tensor with fixed memory depth rather than storing the entire history.

The same structure has a tensor-network interpretation. Each $I_{kk'}$ is an edge connecting two time slices. Before the memory approximation, the graph is a complete lower triangle (Fig. 1). After $k - k' \le K$, it is a finite-width band. TEMPO contracts this banded temporal interaction structure as a matrix-product operator and compresses the resulting temporal correlations [4].

Thus

$$\boxed{\begin{array}{c}\text{harmonic Gaussian bath} \longrightarrow \eta_{kk'} \longrightarrow \text{lower-triangular pair tensor} \longrightarrow k - k' \le K \\ \qquad\qquad\qquad\qquad\qquad\qquad \swarrow \qquad\qquad \searrow \\ \text{QUAPI: explicit finite history} \qquad\qquad \text{TEMPO: compressed temporal network}\end{array}} \tag{81}$$

without ever needing to introduce a continuous-time action.

Figure 1 should be read column-by-column in the newer time index k. Within a fixed column, moving farther below the diagonal increases the lag $k - k'$ and therefore reaches farther into the past. The memory approximation keeps only the strip adjacent to the diagonal, so the retained region moves with k rather than remaining attached to the origin.

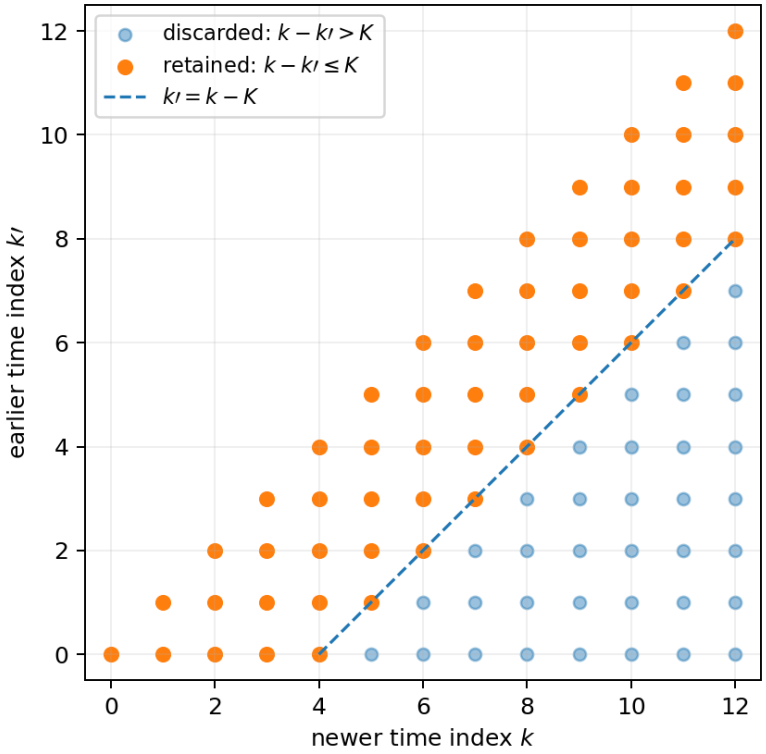

Figure 1: **Lower-triangular time-pair structure of the influence functional.** Each point represents an influence factor $I_{kk'}$ coupling the system at time steps k and k', with $k' \le k$. Under the finite-memory approximation, only factors within the diagonal band $k - k' \le K$ are retained (blue), whereas factors with $k - k' > K$ are discarded (gray). The dashed line $k' = k - K$ marks the memory cutoff.

15 Worked numerical example: an Ohmic spin–boson model

Learning goal. The numerical example is not a second derivation of the influence functional. Its purpose is to show how the already-derived objects become arrays and tensor contractions. For each algorithm, keep asking three questions: what object is stored, how one new time slice is added, and how the expired history is removed.

Numerical and MPS notation

The implementation introduces a small set of symbols that were deliberately omitted from the opening notation table because they are not needed for the influence-functional derivation itself.

Symbol	Meaning
$a_k = (s_k^+, s_k^-)$	Forward/backward path-pair index at time slice k; for the qubit it has local dimension $d = 4$.
d	Local physical dimension of one temporal path-pair site; $d = 4$ for the qubit example.
$G^{[n]}$	Persistent-MPS core at temporal site n, with shape (χ_n, d, χ_{n+1}).
χ_n	MPS bond dimension on bond n; $\chi_0 = \chi_L = 1$ at the boundaries.
$\chi_{\max}$	Optional upper bound on the bond dimension retained after local SVD truncation.
$\varepsilon_{\mathrm{MPS}}$	Relative singular-value cutoff used for temporal compression.
α_{SB}	Dimensionless Ohmic spin–boson coupling parameter, distinct from the bath-mode index α and correlation kernel $\alpha(t)$.

Three representations of the same finite-memory physics

Before entering the code details, it is useful to separate the physical approximation (finite bath memory) from the representation used to store the resulting history tensor. The notebook uses three levels:

Method	**Stored object**	**Dense reconstruction?**	**Role in the tutorial**
Dense QUAPI	Full ADT $A_k(a_k, \ldots, a_{k-K})$	Always dense	Finite-memory reference for validating the tensor operations.
TT–SVD baseline	MPS representation of the ADT	Yes, after every compression	Makes truncation and reconstruction transparent and provides a pedagogical compression check.
Persistent MPS	MPS cores $G^{[n]}$	**Never**	Scalable realization in which the compressed temporal state itself is propagated.

Key idea. All three calculations use the same finite-memory influence structure. They differ in how the augmented history tensor is represented and updated, not in the definition of the pair influence factors I_j.

We now turn the discrete influence functional into an executable finite-memory calculation. The accompanying notebook deliberately contains *three* levels of representation: (i) an explicit dense augmented density tensor (ADT), which is the finite-memory reference; (ii) a pedagogical TT–SVD calculation that compresses the ADT and then reconstructs it before the next update; and (iii) a

genuinely *persistent* MPS propagation in which the ADT remains a matrix-product state throughout the calculation. Keeping these three levels separate is useful because it distinguishes the physics of finite-memory QUAPI from the representation used to store and update its history tensor.

15.1 Qubit Hamiltonian, bath, and numerical parameters

We use units $\hbar = k_{\mathrm{B}} = 1$ and the spin–boson Hamiltonian

$$\hat{H}_S = \frac{\epsilon}{2}\hat{\sigma}_z + \frac{\Delta}{2}\hat{\sigma}_x, \qquad \hat{H}_{SB} = \hat{\sigma}_z \otimes \hat{B}, \tag{82}$$

with exponentially cut off Ohmic spectral density

$$J(\omega) = 2\pi\alpha_{\mathrm{SB}}\,\omega e^{-\omega/\omega_c}. \tag{83}$$

The notebook uses

$$\epsilon = \Delta = 1, \qquad \alpha_{\mathrm{SB}} = 0.08, \qquad \omega_c = 5, \qquad T = 0.5, \tag{84}$$

and, for the main $t = 8$ demonstration,

$$\Delta t = 0.10, \qquad N = 80, \qquad K = 6, \qquad \varepsilon_{\mathrm{MPS}} = 10^{-6}, \qquad \chi_{\max} = 32. \tag{85}$$

The routine-by-routine implementation of the quantities introduced in this subsection is described in Appendix D; the complete source is preserved in Appendix G.

The initial qubit state is

$$\rho_S(0) = |+x\rangle\langle+x| = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}. \tag{86}$$

With the spectral-density convention used throughout this tutorial, the bath correlation kernel evaluated in the notebook is

$$\alpha_B(t) = \frac{1}{\pi}\int_0^\infty \mathrm{d}\omega\, J(\omega)\left[\coth\left(\frac{\beta\omega}{2}\right)\cos(\omega t) - \mathrm{i}\sin(\omega t)\right]. \tag{87}$$

The baseline quadrature uses 5000 frequency points on $10^{-7} \le \omega \le 12\omega_c$ and 41 points per time cell. For $j \ge 1$,

$$\eta_j = \int_{-\Delta t/2}^{\Delta t/2}\mathrm{d}u \int_{-\Delta t/2}^{\Delta t/2}\mathrm{d}v\,\alpha_B(j\Delta t + u - v), \tag{88}$$

whereas the same-cell coefficient keeps only the causal triangle,

$$\eta_0 = \int_{-\Delta t/2}^{\Delta t/2}\mathrm{d}u \int_{-\Delta t/2}^{u}\mathrm{d}v\,\alpha_B(u - v). \tag{89}$$

Because the coupling operator is σ_z and $\sigma_z^2 = I$, the quadratic counterterm is proportional to the identity. Its forward and backward phases cancel, so the notebook does not introduce an additional counterterm factor for this example.

For the parameters above, the notebook obtains

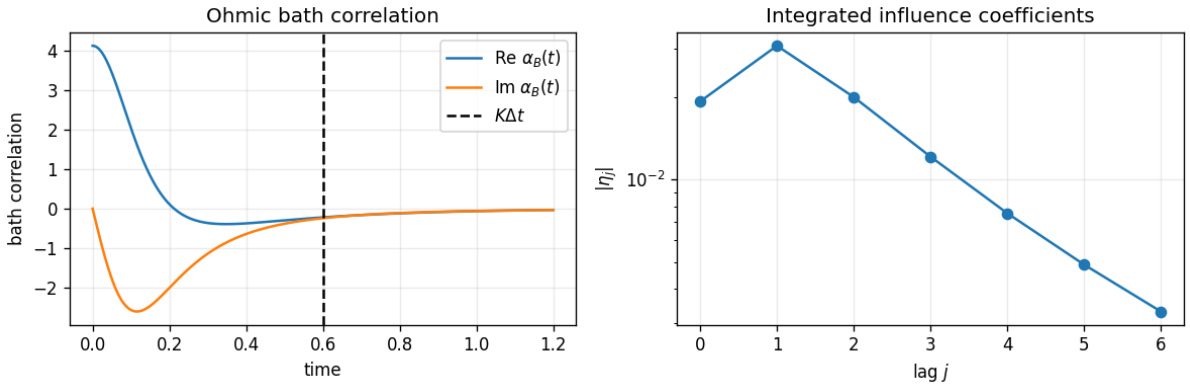

Figure 2: **Ohmic bath correlation and integrated influence coefficients, reproduced from the notebook.** Left: real and imaginary parts of $\alpha_B(t)$; the dashed line marks $K\Delta t = 0.60$. Right: $|\eta_j|$ for the retained lags $j = 0, \ldots, 6$. The same-cell coefficient uses the causal triangular cell integral, while $j \geq 1$ uses complete cells.

j	$\mathrm{Re}\,\eta_j$	$\mathrm{Im}\,\eta_j$	$\lvert\eta_j\rvert$
0	1.8422255×10^{-2}	$-5.8164857 \times 10^{-3}$	1.9318669×10^{-2}
1	2.0890374×10^{-2}	$-2.2700268 \times 10^{-2}$	3.0849796×10^{-2}
2	2.3699684×10^{-3}	$-1.9896477 \times 10^{-2}$	2.0037129×10^{-2}
3	$-3.2709661 \times 10^{-3}$	$-1.1686873 \times 10^{-2}$	1.2135989×10^{-2}
4	$-3.6600305 \times 10^{-3}$	$-6.5944494 \times 10^{-3}$	7.5420545×10^{-3}
5	$-2.9565997 \times 10^{-3}$	$-3.9017907 \times 10^{-3}$	4.8954522×10^{-3}
6	$-2.2129107 \times 10^{-3}$	$-2.4488522 \times 10^{-3}$	3.3005834×10^{-3}

Figure 2 shows the same information graphically. Notice that $|\eta_6|$ is small but not zero; $K = 6$ is therefore a chosen memory approximation, not evidence by itself that the memory limit has converged.

> **Checkpoint.** The correlation function $\alpha_B(t)$ and the discrete coefficients η_j answer different questions. $\alpha_B(t)$ diagnoses the continuous bath time scale, whereas η_j is the actual cell-integrated quantity used by the discrete update. Convergence must ultimately be judged from the propagated reduced state, not from either plot alone.

15.2 The path-pair index and the QUAPI update

For a qubit, one temporal physical index is the forward/backward pair

$$a_k = (s_k^+, s_k^-), \tag{90}$$

with the notebook ordering

$$(+,+), \quad (+,-), \quad (-,+), \quad (-,-). \tag{91}$$

Thus every temporal leg has local dimension $d = 4$. For $U = e^{-\mathrm{i}H_S\Delta t}$, the isolated-system path-pair propagator is

$$K_S(a_{k+1}, a_k) = U_{s_{k+1}^+, s_k^+} U^*_{s_{k+1}^-, s_k^-}. \tag{92}$$

Stationarity allows the influence factor to be labeled only by its lag,

$$I_j(a_k, a_{k-j}) = \exp\left[-(s_k^+ - s_k^-)\left(\eta_j s_{k-j}^+ - \eta_j^* s_{k-j}^-\right)\right]. \tag{93}$$

The factor inserted when the newest slice k is added is therefore

$$\prod_{j=0}^{\min(K,k)} I_j(a_k, a_{k-j}). \tag{94}$$

15.3 Dense finite-memory ADT: the reference calculation

The explicit QUAPI implementation stores

$$A_k(a_k, a_{k-1}, \ldots, a_{k-K}), \tag{95}$$

with fewer legs before the memory window has filled. One step first creates a new leading index and applies the system kernel,

$$\widetilde{A}_{k+1}(a_{k+1}, a_k, \ldots) = K_S(a_{k+1}, a_k) A_k(a_k, \ldots), \tag{96}$$

then multiplies the same tensor by $I_0(a_{k+1})$ and the pair factors $I_j(a_{k+1}, a_{k+1-j})$. Once more than $K+1$ temporal sites are present, the oldest physical index is summed out. The reduced state is obtained by summing every retained history leg except the newest one.

For a qubit, the maximum dense storage is

$$N_{\text{dense}} = d^{K+1} = 4^{K+1}. \tag{97}$$

At $K = 6$ this is only $4^7 = 16384$ complex numbers, so the dense ADT is still practical and provides a useful finite-memory reference.

15.4 Pedagogical TT–SVD baseline: compress, then reconstruct

The notebook next compresses the same dense ADT by TT–SVD. At each temporal cut the tensor is reshaped into a matrix and decomposed as

$$M = U \Sigma V^\dagger. \tag{98}$$

A singular value is retained when

$$\sigma_r > \varepsilon_{\text{MPS}} \sigma_1, \tag{99}$$

subject to the optional cap $\chi_{\max}$. Repeating the SVD from the newest site toward the oldest produces

$$A(a_k, \ldots, a_{k-L+1}) \simeq \sum_{\alpha_1 \cdots \alpha_{L-1}} G^{[0]}_{1, a_k, \alpha_1} G^{[1]}_{\alpha_1, a_{k-1}, \alpha_2} \cdots G^{[L-1]}_{\alpha_{L-1}, a_{k-L+1}, 1}. \tag{100}$$

In this first implementation the MPS is immediately contracted back into a dense ADT before the next QUAPI update. This is deliberate: it makes the effect of TT–SVD truncation easy to inspect, but it does *not* provide the memory scaling of a persistent MPS algorithm.

With $K = 6$, $\varepsilon_{\text{MPS}} = 10^{-6}$, and $\chi_{\max} = 32$, the largest retained bond is 18. Figure 3 shows that the compressed and uncompressed reduced-density-matrix trajectories are visually indistinguishable over $t = 0$–8. The maximum elementwise difference over the complete trajectory is

$$\max_{t,i,j} |\rho_{ij}^{\text{MPS}}(t) - \rho_{ij}^{\text{ADT}}(t)| = 3.304 \times 10^{-6}. \tag{101}$$

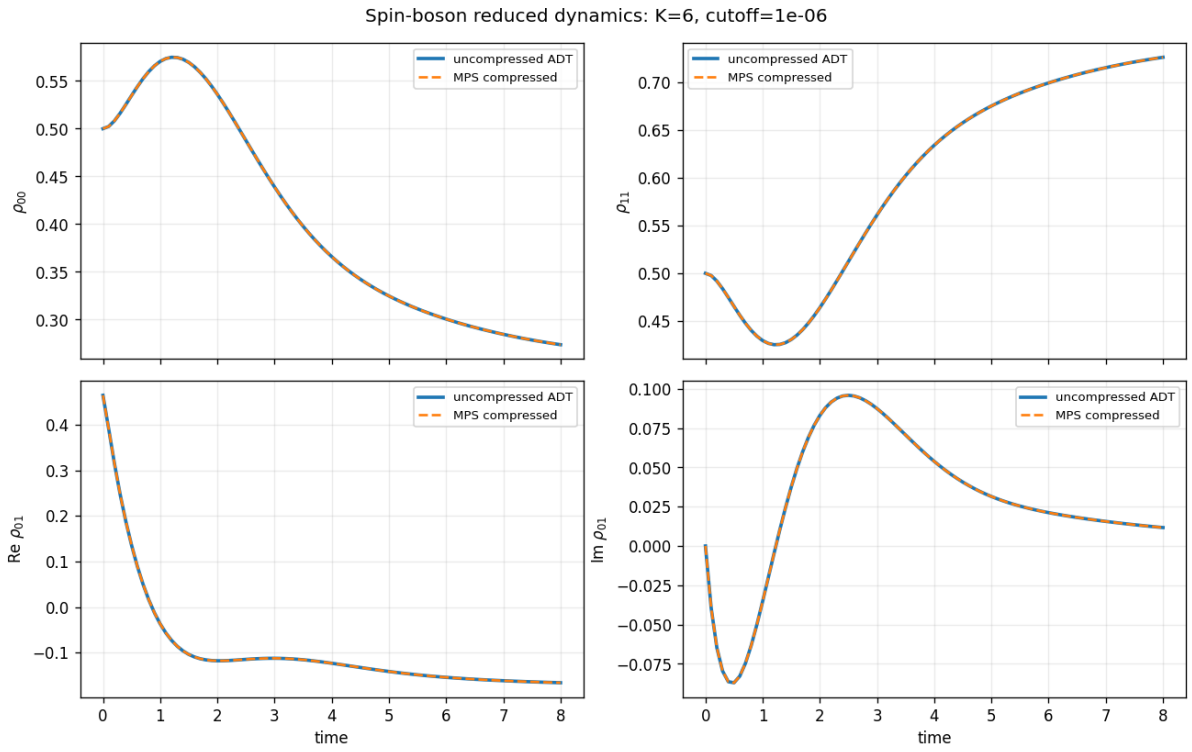

Figure 3: **Dense ADT versus the reconstruct-after-TT–SVD calculation.** The four panels show ρ_{00}, ρ_{11}, $\mathrm{Re}\,\rho_{01}$, and $\mathrm{Im}\,\rho_{01}$ for $K = 6$ and relative cutoff 10^{-6}. The dashed compressed curves lie essentially on top of the uncompressed finite-memory reference.

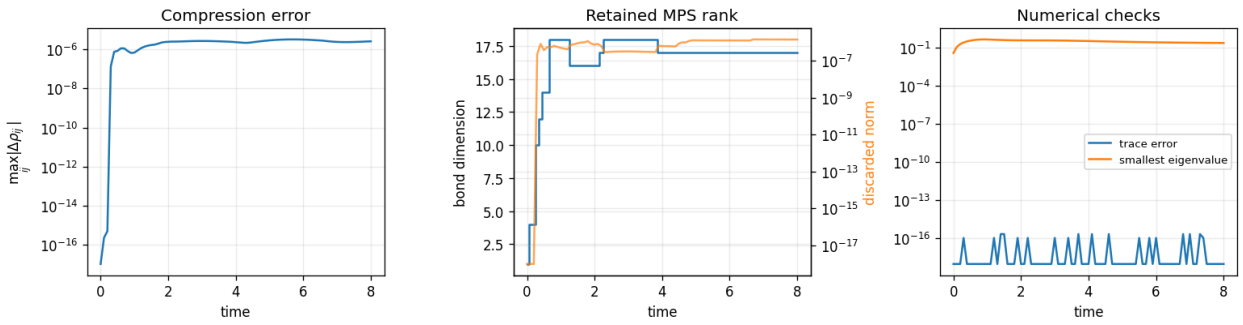

Figure 4: **Diagnostics for reconstruct-after-TT–SVD compression.** Left: maximum element-wise reduced-state error relative to the dense ADT. Center: retained maximum MPS bond dimension together with the discarded TT–SVD norm. Right: trace error after the notebook's cleanup and the smallest eigenvalue of the reduced state.

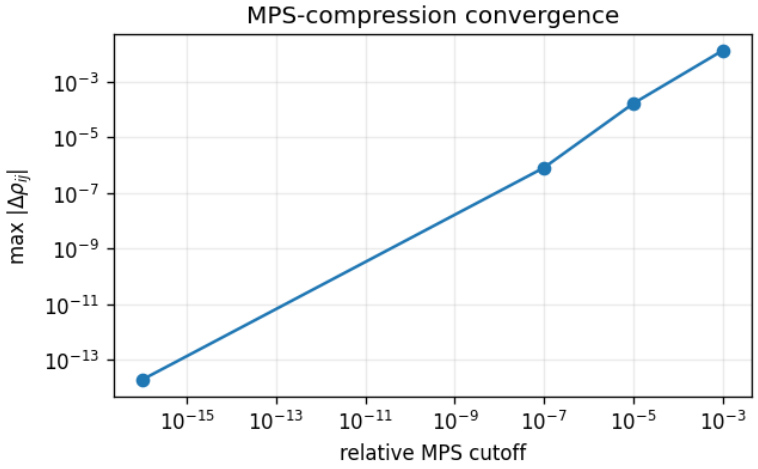

Figure 5: **TT–SVD cutoff convergence for the pedagogical reconstructing implementation.** The error decreases monotonically as the relative singular-value cutoff is tightened; at zero cutoff and without a bond cap the result agrees with the dense ADT to approximately 2×10^{-14}.

The diagnostics in Fig. 4 quantify the same result. The maximum trace error after the notebook's small Hermiticity/trace cleanup is 2.220×10^{-16}, and the smallest eigenvalue encountered is 3.552×10^{-2}. The cleanup symmetrizes and normalizes ρ_S only after the history tensor has been reduced; it does not impose positivity and is not fed back into the ADT propagation.

The relative-cutoff sweep provides an internal check of the compression machinery. The maximum reduced-state error falls from 1.312×10^{-2} at $\varepsilon_{\mathrm{MPS}} = 10^{-3}$ to 8.073×10^{-7} at 10^{-7}; with zero cutoff and no bond cap, the dense finite-memory result is recovered to 1.910×10^{-14}. The complete numerical values, including retained bond dimensions and notebook runtimes, are collected in Appendix E, Table 1.

The zero-cutoff calculation also removes the bond cap, recovering the dense finite-memory trajectory to roundoff. The wall-clock values are notebook-run timings and should not be interpreted as hardware-independent performance benchmarks.

15.5 Persistent MPS QUAPI: keeping the history tensor compressed

The final notebook implementation removes the dense reconstruction step. Conceptually, the algorithm now carries a compressed "rolling memory" of the recent path-pair history: new information enters at the newest boundary, bath couplings are applied only across the retained lags, expired information leaves at the oldest boundary, and local factorizations keep the representation compact. The retained history tensor is always stored as MPS cores,

$$A_k(a_k, a_{k-1}, \ldots, a_{k-L+1}) = \sum_{\alpha_1 \cdots \alpha_{L-1}} G^{[0]}_{1,a_k,\alpha_1} G^{[1]}_{\alpha_1,a_{k-1},\alpha_2} \cdots G^{[L-1]}_{\alpha_{L-1},a_{k-L+1},1}, \tag{102}$$

where $L \leq K+1$ and each physical index has dimension $d = 4$. The storage is now

$$N_{\mathrm{MPS}} = \sum_{n=0}^{L-1} \chi_n \, d \, \chi_{n+1}, \qquad \chi_0 = \chi_L = 1, \tag{103}$$

rather than d^L.

Algorithm at a glance

The persistent update is easiest to understand in two passes. First ignore the tensor indices and view one time step as the six operations

$$\boxed{\begin{array}{c} \text{add newest site} \longrightarrow \text{apply } K_S \text{ and } I_j \longrightarrow \text{discard expired memory} \\ \downarrow \\ \rho_S \longleftarrow \text{SVD-compress} \longleftarrow \text{right-canonicalize} \end{array}} \tag{104}$$

The first three operations implement the exact finite-memory QUAPI update for the chosen K; the next two change only its MPS representation (up to controlled SVD truncation); and the final contraction reads out the reduced system state. The central bookkeeping trick is that the newest path-pair value $p = a_{k+1}$ is carried consistently through the virtual bonds so that every lagged factor $I_j(p, a_{k+1-j})$ sees the *same* p. Appendix D maps each of these six operations to the corresponding notebook routine, while Appendix G contains the complete executable source.

We now derive these operations one by one.

Step 1: introduce the new path-pair value as both a physical and virtual label

Let $p = a_{k+1}$ denote the newly introduced path-pair value. The exact finite-memory update can be written

$$\begin{aligned} B(p, a_k, a_{k-1}, \ldots) = I_0(p)\, A_k(a_k, a_{k-1}, \ldots)\, [K_S(p, a_k) I_1(p, a_k)] \\ \times \prod_{j=2}^{K} I_j(p, a_{k-j+1}) \,, \end{aligned} \tag{105}$$

with factors omitted when the corresponding history site does not yet exist. The key implementation idea is to *thread the same value p through the virtual bonds.* The newly prepended core is

$$G^{[\mathrm{new}]}_{1,p,q} = I_0(p)\, \delta_{pq}, \tag{106}$$

so its right virtual index q explicitly carries the four-valued label p into the history chain.

Step 2: insert each lagged influence factor locally

At the first old site, the local multiplier is

$$F_1(p, a_k) = K_S(p, a_k) I_1(p, a_k), \tag{107}$$

whereas at an older retained site of lag j it is simply

$$F_j(p, a) = I_j(p, a), \qquad 2 \le j \le K. \tag{108}$$

For each possible p, the old core is copied into the corresponding p block of the expanded virtual bonds and multiplied by $F_j(p, a)$. In other words, the virtual bond is used as a classical carrier for the newest path-pair value while the physical index of that core supplies the older path-pair value. Because p has only four possible values, this exact pre-compression update enlarges an affected virtual dimension by at most the local physical dimension $d = 4$; it never requires formation of the full rank-$(K + 2)$ dense ADT.

At the right boundary there is no future core to which p must be passed. The final updated core therefore has right bond dimension one. This is the tensor-network version of completing the influence factors associated with the newest slice.

Step 3: discard expired memory directly at the right boundary

After the new site is inserted, there can temporarily be $K + 2$ physical sites. The oldest site is then beyond the permitted lag and cannot enter any future influence factor. The notebook removes it without reconstructing the ADT by forming the boundary vector

$$b_\alpha = \sum_a G^{[L-1]}_{\alpha,a,1} \tag{109}$$

and contracting b_α into the right bond of the neighboring core. This operation is exactly the MPS realization of the QUAPI sum over the expired history index.

Step 4: right-canonicalize using only local QR factorizations

Before truncation, the cores are swept from right to left. For core n with dimensions (χ_L, d, χ_R), the notebook reshapes

$$M_n \in \mathbb{C}^{\chi_L \times (d\chi_R)} \tag{110}$$

and performs a reduced QR decomposition of M_n^{T}. The resulting orthonormal factor is reshaped back into the current core, while the triangular factor is absorbed into the core on its left. The purpose of this canonicalization is not cosmetic: it puts the subsequent singular values into a well-defined local Schmidt-like form so that truncation can be performed bond by bond.

Step 5: compress with a left-to-right local SVD sweep

For the truncation sweep, core n is reshaped instead as

$$M_n \in \mathbb{C}^{(\chi_L d) \times \chi_R} = U \Sigma V^\dagger. \tag{111}$$

The retained rank is

$$\chi_n^{\text{keep}} = \min[\chi_{\max}, \#\{r : \sigma_r > \varepsilon_{\text{MPS}} \sigma_1\}], \tag{112}$$

with at least one singular value kept. U becomes the new left-isometric core and $\Sigma V^\dagger$ is absorbed into the next core. The discarded diagnostic recorded by the notebook is the square root of the sum of discarded squared singular values over the sweep. Finally, the last core is divided by its norm to prevent numerical underflow or overflow over long propagations. This changes only a global scale of the ADT, which drops out when the reduced density matrix is normalized.

Step 6: obtain ρ_S directly from the MPS

The reduced density matrix is also evaluated without forming the history tensor. Starting from the right boundary, each history core is summed over its physical index and contracted into an environment vector. Contracting this environment with the newest core leaves only the four values of a_k, which are reshaped into the 2×2 reduced density matrix. Thus the entire propagation–update, memory removal, compression, and readout–uses only MPS cores and local matrix factorizations.

> **Key idea.** In the persistent algorithm, "MPS compression" is not an after-the-fact representation of a dense QUAPI tensor. The MPS *is* the propagated augmented state. The new path value is transported through virtual bonds so that every I_j is applied locally, and the oldest history value is eliminated at the right boundary before the next compression. At no point does the code allocate or reconstruct an object of size 4^{K+1}.

15.6 Persistent-MPS demonstration at $K = 6$

The persistent demonstration uses the same physical parameters, $\Delta t = 0.10$, $K = 6$, $t_{final} = 8$, $\varepsilon_{\mathrm{MPS}} = 10^{-6}$, and $\chi_{\max} = 32$. Its cached bath routine evaluates the same integrals with 2500 frequency points and 31 time-cell points. The notebook reports

largest retained bond dimension	18,
peak stored MPS coefficients	3376,
hypothetical dense ADT elements	16384,
propagation runtime	0.683 s.

(113)

The stored-core count is about 20.6% of the dense-element count, a reduction by a factor of approximately 4.85 for this modest memory depth. The principal point is not this particular factor—which depends on the attained bond dimensions—but that the persistent algorithm does not require dense storage at all.

Figure 6 shows the reduced-state trajectory produced by this persistent propagation. Because the persistent demonstration uses its own cached quadrature resolution, the figure should be read as the output of that implementation, not as a direct point-by-point comparison with Fig. 3.

15.7 Convergence tests for the persistent MPS

The notebook then performs three separate convergence sweeps over the shorter interval $0 \le t \le 1$. For speed, these sweeps use 25 time-cell quadrature points and 1800 frequency points.

> **Implementation note.** A useful convergence order is: first stabilize the bath quadratures, then increase the physical memory time $K\Delta t$, then reduce Δt while holding that physical memory time fixed, and finally tighten the MPS truncation. In practice some iteration is usually needed, but this ordering prevents one error source from being mistaken for another.

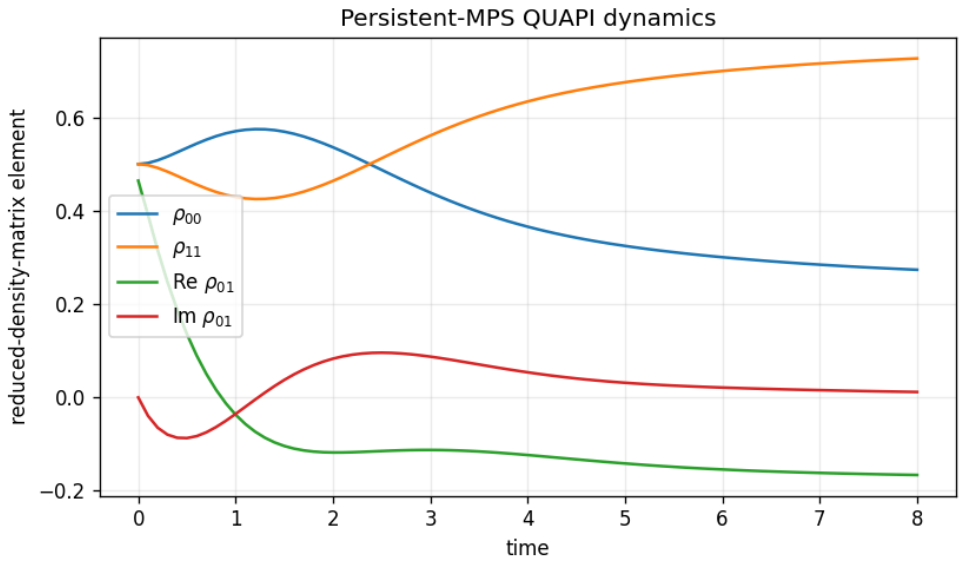

Figure 6: **Reduced-density-matrix dynamics from persistent-MPS QUAPI.** The calculation uses $\Delta t = 0.10$, $K = 6$, $t_{\mathrm{final}} = 8$, $\varepsilon_{\mathrm{MPS}} = 10^{-6}$, and $\chi_{\max} = 32$. The ADT is never constructed or reconstructed densely.

The error metric is

$$\epsilon_\rho = \max_{t,i,j} \left| \rho_{ij}(t) - \rho_{ij}^{\mathrm{ref}}(t) \right|, \tag{114}$$

where the reference is the finest calculation *within that particular sweep*. Consequently, the zero reported for the reference row means "difference from itself"; it does not mean that every other numerical approximation has been removed.

How much bath memory must we retain?

Varied. Memory depth K.

Held fixed. $\Delta t = 0.10$, relative MPS cutoff 10^{-11}, and $\chi_{\max} = 48$.

Reference. The $K = 10$ trajectory.

For $\Delta t = 0.10$, the notebook uses $K = 2, 4, 6, 8, 10$, a tight relative cutoff 10^{-11}, and $\chi_{\max} = 48$. Relative to the $K = 10$ trajectory, the error decreases from 4.434×10^{-2} at $K = 2$ to 5.126×10^{-3} at $K = 6$ and 1.277×10^{-3} at $K = 8$. The full table is given in Appendix E, Table 2.

The monotonic decrease shows that extending the physical memory time materially changes the trajectory over this parameter range; in particular, the default $K = 6$ result is not yet converged to the $K = 10$ reference at the 10^{-3} level.

Is the Trotter time step converged?

Varied. Time step Δt.

Held fixed. Physical memory time $K\Delta t = 0.60$.

Reference. The $\Delta t = 0.05$, $K = 12$ trajectory.

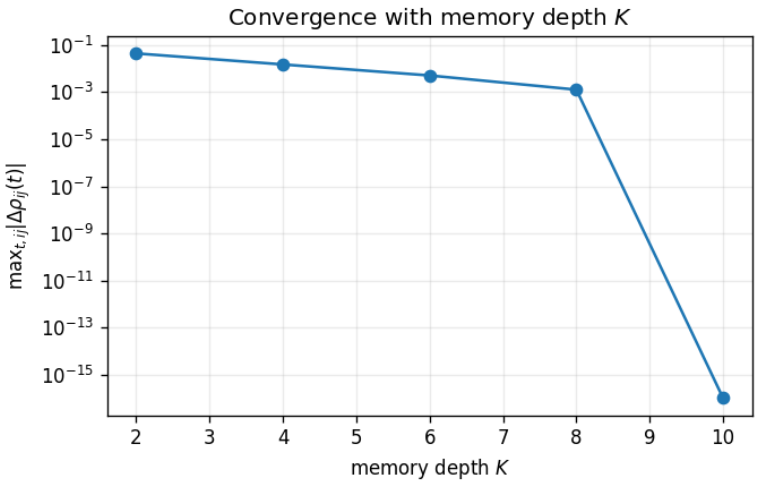

Figure 7: **Persistent-MPS convergence with memory depth.** The metric is the maximum reduced-density-matrix element error over $0 \leq t \leq 1$ relative to the $K = 10$ calculation at fixed $\Delta t = 0.10$. The final plotted point is at the plotting floor because the reference trajectory has zero self-error.

A time-step sweep must not inadvertently change the bath memory time. The notebook therefore fixes

$$\tau_{\mathrm{mem}} = K\Delta t = 0.60 \tag{115}$$

and uses $(\Delta t, K) = (0.20, 3), (0.10, 6), (0.05, 12)$. With the $\Delta t = 0.05$ calculation as the reference, the maximum trajectory error is 9.315×10^{-2} at $\Delta t = 0.20$ and 3.645×10^{-2} at $\Delta t = 0.10$. The complete values are reported in Appendix E, Table 3.

The substantial change on halving Δt again demonstrates why memory convergence and Trotter convergence must be tested independently.

How large must the MPS bond dimension be?

Varied. Maximum retained bond dimension $\chi_{\max}$.

Held fixed. $\Delta t = 0.10$, $K = 6$, and relative cutoff 10^{-13}.

Reference. The $\chi_{\max} = 64$ run, whose largest actually used bond is 60.

Finally, the notebook isolates the tensor-compression limit at fixed $\Delta t = 0.10$ and $K = 6$ using the tight relative cutoff 10^{-13}. The $\chi_{\max} = 64$ run is the reference; its largest actually used bond is 60. The compression error drops from 1.097×10^{-3} at $\chi_{\max} = 4$ to 5.626×10^{-7} at 16, 2.352×10^{-9} at 32, and 1.332×10^{-12} at 48. The complete bond and storage data are given in Appendix E, Table 4.

This sweep is particularly useful pedagogically because the error falls by almost nine orders of magnitude between $\chi_{\max} = 4$ and 48. It also illustrates that the bond cap and singular-value cutoff are distinct controls: once the cap is sufficiently large, the singular-value spectrum rather than $\chi_{\max}$ determines the actual retained rank.

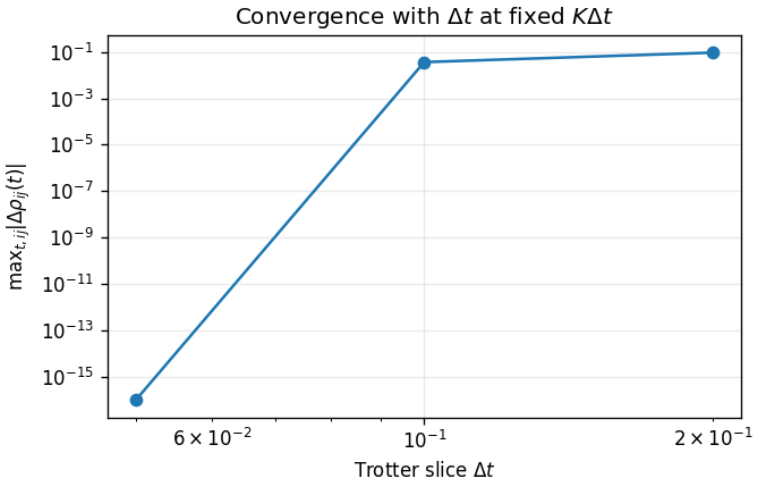

Figure 8: **Persistent-MPS convergence with the Trotter slice at fixed physical memory time.** Here $K\Delta t = 0.60$ is held constant while Δt is reduced. Errors are measured relative to the $\Delta t = 0.05$, $K = 12$ trajectory.

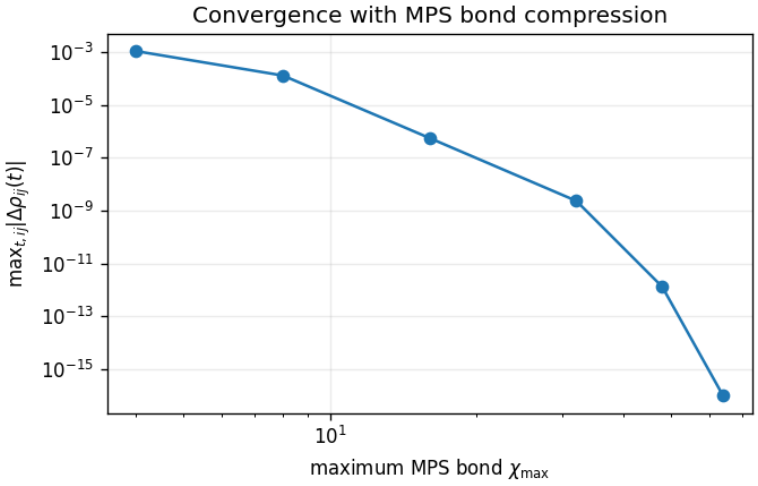

Figure 9: **Persistent-MPS convergence with maximum bond dimension.** The calculation uses $\Delta t = 0.10$, $K = 6$, and relative cutoff 10^{-13}; errors are reported relative to the $\chi_{\max} = 64$ run, whose largest used bond is 60.

15.8 What the notebook establishes—and what it does not

The three implementations answer different numerical questions. Agreement between the dense ADT and zero-cutoff reconstruct-after-TT–SVD calculation verifies the path-pair ordering, TT–SVD factorization, and reconstruction. The persistent algorithm then shows how the same finite-memory structure can be propagated using only MPS cores. Its bond sweep demonstrates systematic convergence of the tensor compression.

The notebook also makes clear that tensor compression is only one error source. For these particular test parameters, both the K sweep and the Δt sweep change the trajectory more strongly than a well-converged MPS bond dimension. A production calculation must therefore converge, separately, (i) bath quadrature, (ii) memory depth K or physical memory time $K\Delta t$, (iii) Trotter slice Δt, and (iv) MPS truncation parameters. A small MPS compression error does not imply that the finite-memory or time-discretization errors are small.

16 Implementation checklist and convergence hierarchy

A numerical implementation should satisfy all of the following.

1. **Trace closure.** The final bath coordinate is common to the two branches:

$$x^+_{N+1} = x^-_{N+1}. \tag{116}$$

   Without this identification, one has not taken the bath trace.

2. **Normalization.** For identical system histories,

$$\mathcal{F}[\mathbf{s}, \mathbf{s}] = 1. \tag{117}$$

   In Eq. (69), this is manifest because every exponent contains $s^+_k - s^-_k$.

3. **Counterterm appears exactly once.** In this tutorial it belongs to

$$\hat{H}_{BC} = \hat{H}_B - \hat{s}\hat{B} + \Lambda \hat{s}^2. \tag{118}$$

   Therefore it is already present in the conditional propagators and in the full influence functional. Do not add a second factor $\mathcal{F}_{\mathrm{ct}}$ and do not also absorb $\Lambda\hat{s}^2$ into $\hat{H}_S$.

4. **Correct exact source.** For one forward conditional interval, the exact linear source is

$$\frac{\mathrm{i}c_\alpha s^+_k}{\hbar\omega_\alpha} \tan\left(\frac{\omega_\alpha \tau_k}{2}\right) (x^+_{\alpha,k} + x^+_{\alpha,k+1}), \tag{119}$$

   with the opposite sign on the backward branch. A point-source replacement proportional to $\Delta t\, s_k$ is only a short-time approximation and misses the endpoint half-step structure.

5. **Diagonal cell.** The bath-correlation part of η_{kk} is a triangular same-cell integral, not a full square. The full diagonal coefficient is

$$\eta_{kk} = \eta^{(B)}_{kk} + \mathrm{i}\Lambda\tau_k. \tag{120}$$

6. **No extra factor of** i **in the pair factor.** With the convention $C(t) = \hbar\alpha(t)$ used here, $\eta_{kk'}$ is already complex. The pair exponent therefore begins with $-1/\hbar$, not $-\mathrm{i}/\hbar$.

7. **Memory is a lag cutoff.** The retained earlier indices are

$$k' = \max(0, k - K), \ldots, k. \tag{121}$$

8. **A persistent MPS must never reconstruct the global ADT.** All update, memory-removal, canonicalization, compression, and reduced-state operations should act on local cores. Dense QR/SVD operations are permitted only on local core matricizations, not on a tensor of size d^{K+1}.

9. **The newest path-pair label must be the same at every lag factor.** In the persistent update, the new value $p = a_{k+1}$ is threaded through the virtual bonds. Using independent virtual labels at different history sites would apply $I_j(p, a_{k+1-j})$ with inconsistent values of p and would no longer represent the QUAPI influence product.

10. **The expired site is removed at the oldest boundary.** Once $L > K + 1$, sum the physical index of the rightmost MPS core and absorb the resulting boundary vector into its neighbor. Removing any other site would destroy the moving lag window.

Part IV
From TEMPO to process tensors

The finite-memory construction has so far been used to answer a forward-propagation question: given an initial reduced state and a specified system evolution, what reduced state is obtained at a later time? The same temporal bath-memory network can be used more generally. If the operations performed on the system at intermediate times are not contracted immediately, those times become open operation slots. The resulting multitime object is a *process tensor*.

This part develops that construction directly from the influence factors already used in QUAPI and TEMPO. The important conceptual step is therefore not the introduction of a new bath model, but a reorganization of the same path-integral tensor network.

17 Process tensors: from a fixed propagation to a reusable multitime object

Learning goal. The preceding sections used the influence functional to propagate one specified reduced-state dynamics. We now keep the same temporal memory network but leave system operations at intermediate times open. By the end of this part, the reader should be able to explain what an intervention is, why a process-tensor slot needs separate input and output Liouville-space indices, how finite bath memory is compressed into a matrix-product representation, how the fixed free-system propagator is incorporated in the pedagogical implementation used below, and how one stored process tensor can be reused for many pulse and measurement protocols.

A concrete spectroscopy picture is useful before introducing the notation. Imagine preparing the system, waiting, applying a pulse, waiting again, and measuring an observable,

$$\boxed{\text{prepare} \to \text{wait} \to \text{pulse} \to \text{wait} \to \text{measure.}} \tag{122}$$

Ordinary propagation evaluates the dynamics for this specified experiment. A process tensor instead leaves the pulse slot open while the expensive temporal memory network is constructed. Different pulses can then be inserted afterward.

17.1 What is an intervention?

An *intervention* is any controlled operation applied to the system at a specified time during its evolution. Examples include a unitary pulse, a measurement, a preparation or reset, a quantum gate, or a conditional operation chosen according to an earlier measurement outcome.

We denote the operation at time t_k by

$$\mathcal{A}_k. \tag{123}$$

For a deterministic physical operation, $\mathcal{A}_k$ is a completely positive trace-preserving map. A map corresponding to one selected measurement outcome is completely positive and trace nonincreasing.

In this tutorial we keep the initial reduced density matrix as an explicit boundary condition rather than treating the initial preparation as another process-tensor slot. With this convention,

$$\boxed{\rho_S(t_N) = \mathcal{T}_{N:0}\left[\mathcal{A}_{N-1}, \ldots, \mathcal{A}_1; \rho_S(0)\right].} \tag{124}$$

Equivalently, one may absorb the initial preparation into the first intervention. The two conventions differ only in bookkeeping.

17.2 Starting point: the finite-memory influence functional

For the qubit linearly coupled to a Gaussian bath considered in the preceding sections, discretize time as

$$t_k = k\Delta t, \qquad k = 0, \ldots, N. \tag{125}$$

After truncating the bath memory to K time steps, write

$$\boxed{\mathcal{F}^{(K)} = \prod_{k=0}^{N} \prod_{j=0}^{\min(K,k)} I_{k,k-j}.} \tag{126}$$

Introduce the Liouville-space path-pair index

$$q_k = (s_k^+, s_k^-). \tag{127}$$

For a qubit, q_k has four values. The influence functional becomes

$$\boxed{\mathcal{F}^{(K)}(q_0, \ldots, q_N) = \prod_{k=0}^{N} \prod_{j=0}^{\min(K,k)} I_{k,k-j}(q_k, q_{k-j}).} \tag{128}$$

Each pair factor records a bath-mediated correlation between two time slices. Because only $k - k' \le K$ is retained, the tensor has a finite-width interaction band along the time direction.

17.3 Step 1: interpret the influence functional as a temporal network

For example, with $K = 2$,

$$\begin{aligned} \mathcal{F}^{(2)} = {} & I_{0,0}\,(I_{1,1}I_{1,0})\,(I_{2,2}I_{2,1}I_{2,0}) \\ & \times (I_{3,3}I_{3,2}I_{3,1}) \cdots . \end{aligned} \tag{129}$$

Thus q_k interacts only with

$$q_k, q_{k-1}, \ldots, q_{k-K}. \tag{130}$$

This finite-range temporal interaction is the structure exploited by TEMPO.

17.4 Step 2: compress the bath influence into a matrix-product representation

The object

$$\mathcal{F}^{(K)}(q_0, \ldots, q_N) \tag{131}$$

is a tensor with one Liouville-space index for every time slice. Storing it explicitly therefore scales exponentially with N. TEMPO instead compresses the temporal tensor into a matrix-product form,

$$\boxed{\mathcal{F}^{(K)}(q_0, \ldots, q_N) \simeq \sum_{a_0, \ldots, a_{N-1}} W^{[0]}_{q_0,a_0} W^{[1]}_{a_0,q_1,a_1} \cdots W^{[N]}_{a_{N-1},q_N}.} \tag{132}$$

At this stage each time site has one fused physical index q_k. It is therefore most transparent to call Eq. (132) a temporal matrix-product state, or more generally a matrix-product representation, of the influence functional. The matrix-product-*operator* structure becomes explicit when each intervention time is split into separate input and output legs.

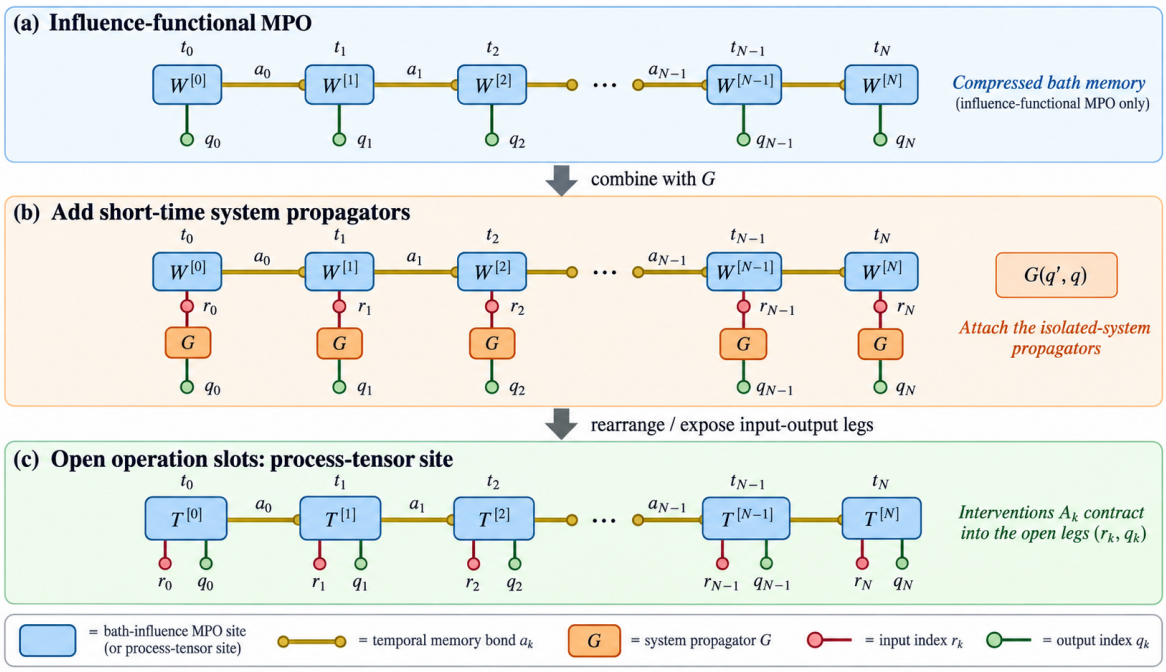

Figure 10: Construction of the system-dressed process tensor used in the pedagogical implementation below. **(a)** The finite-memory influence functional is compressed into a temporal matrix-product representation with physical Liouville-space indices q_k and bond indices a_k carrying the compressed bath memory. **(b)** A fixed short-time free-system kernel G is attached between successive time slices, but the externally chosen intervention maps are not contracted. **(c)** Each intervention time is resolved into an incoming index r_k and an outgoing index q_k. The pair (q_k, r_k) is an open operation slot that can later be contracted with a pulse, measurement, or other map $\mathcal{A}_k$. Splitting the grouped physical leg into its input and output factors gives the usual process-tensor MPO picture.

The bond indices a_k compress the bath information that must pass from the past to the future. Cutting the temporal network between t_k and t_{k+1} gives the schematic interpretation

$$\boxed{\underbrace{q_0, \ldots, q_k}_{\text{past}} \longrightarrow a_k \longrightarrow \underbrace{q_{k+1}, \ldots, q_N}_{\text{future}} .} \tag{133}$$

Sequential compression. When a new time slice t_k is introduced, only the factors

$$I_{k,k}, I_{k,k-1}, \ldots, I_{k,k-\min(K,k)} \tag{134}$$

must be added. The enlarged tensor is reshaped across an appropriate temporal cut,

$$M_{\alpha\beta}, \tag{135}$$

and decomposed as

$$\boxed{M = USV^\dagger.} \tag{136}$$

Keeping only the significant singular values replaces the explicit collection of past histories by a smaller bond index.

Two approximations must be distinguished. First, the memory cutoff neglects pair factors beyond

$$\tau_{\mathrm{mem}} = K\Delta t. \tag{137}$$

Second, SVD compression discards weak tensor components inside the retained memory window. Schematically,

memory truncation	matrix-product compression
K	ϵ
discard long-lag correlations	discard weak singular components.

(138)

These limits must be converged independently.

17.5 Two useful conventions: bath-only and system-dressed process tensors

A point of terminology is worth making explicit before proceeding because both conventions occur in the literature and in software implementations.

In a *bath-only PT-TEMPO factorization*, one first compresses only the environmental influence. Denoting this object by $\mathcal{T}^{(B)}_{N:0}$, the system propagators are supplied later together with any control operations. This factorization is particularly useful when one wants to reuse the same bath tensor for different time-dependent system Hamiltonians.

In the pedagogical notebook analyzed below, we use a slightly more dressed object. The fixed free-system short-time propagator G is incorporated during construction, while the interventions $\mathcal{A}_k$ remain open. We denote this object simply by $\mathcal{T}_{N:0}$. It is the operational process tensor for that fixed uncontrolled system evolution. It can be reused for different pulses, pulse times, measurements, observables, and initial system states, but changing H_S changes G and therefore requires rebuilding this particular stored tensor.

Thus,

bath-only PT-TEMPO	notebook convention
$\{I_{k,k-j}\}$	$\{I_{k,k-j}\} + G$
$\Downarrow$	$\Downarrow$
$\mathcal{T}^{(B)}_{N:0}$	$\mathcal{T}_{N:0}$
$H_S(t)$ inserted later	H_S fixed; interventions inserted later.

(139)

Both are legitimate tensor-network factorizations. The second is used here because it follows directly from the persistent-MPS QUAPI implementation and makes the relation between ordinary propagation and open intervention slots especially transparent. The bath-only form is preferable when repeated scans over different system Hamiltonians are required [6].

17.6 Step 3: incorporate the fixed free-system propagation

Let

$$U_{\Delta t} = e^{-iH_S\Delta t/\hbar} \tag{140}$$

be the free-system propagator. In Liouville space,

$$\boxed{G(q', q) = \langle s'^{+}|U_{\Delta t}|s^{+}\rangle\langle s'^{-}|U_{\Delta t}|s^{-}\rangle^{*}.} \tag{141}$$

For uninterrupted evolution, ordinary QUAPI contracts these kernels while the influence network is propagated,

$$\rho_S(q_N, t_N) = \sum_{q_0,\ldots,q_{N-1}} \rho_S(q_0, 0) \left[\prod_{k=0}^{N-1} G(q_{k+1}, q_k)\right] \mathcal{F}^{(K)}(q_0, \ldots, q_N). \tag{142}$$

The system-dressed process tensor changes the contraction at the points where controlled operations may later be inserted. The free step remains fixed, but the intervention itself is not chosen yet.

17.7 Step 4: split an intervention time into an input and an output

Immediately before the intervention at t_k, define

$$r_k = (r_k^+, r_k^-), \tag{143}$$

and immediately afterward define

$$q_k = (s_k^+, s_k^-). \tag{144}$$

The local sequence is

$$\boxed{q_{k-1} \xrightarrow{G} r_k \xrightarrow{\mathcal{A}_k} q_k.} \tag{145}$$

The intervention matrix elements are

$$\boxed{[\mathcal{A}_k]_{q_k r_k} = \langle s_k^+ | \mathcal{A}_k \left(|r_k^+\rangle\langle r_k^-| \right) |s_k^-\rangle.} \tag{146}$$

For a d-dimensional system, q_k and r_k each have dimension d^2. The open operation slot therefore has dimension d^4. For a qubit,

$$\dim q_k = \dim r_k = 4, \qquad \dim(q_k, r_k) = 16. \tag{147}$$

It is convenient in the code to group the two legs into one physical index,

$$x_k = (q_k, r_k), \qquad x_k = 4q_k + r_k, \qquad x_k = 0, \ldots, 15. \tag{148}$$

The grouped tensor is stored as a temporal MPS; ungrouping x_k into (q_k, r_k) exposes the process-tensor MPO input/output structure.

17.8 Step 5: leave the intervention tensors and initial state uncontracted

During construction we combine the bath influence factors with the fixed free-system propagators G, but we do *not* contract the operations $\mathcal{A}_k$. The initial state is also retained as an external boundary condition.

The resulting temporal tensor has the schematic structure

$$\boxed{q_N - (q_{N-1}, r_{N-1}) - \cdots - (q_1, r_1) - q_0.} \tag{149}$$

Only after this object has been built do we choose $\rho_S(0)$ and the operations $\mathcal{A}_k$. The final state is then

$$\boxed{\rho_S(t_N) = \mathcal{T}_{N:0} \left[\mathcal{A}_{N-1}, \ldots, \mathcal{A}_1; \rho_S(0)\right].} \tag{150}$$

The essential point is that the operation tensors are not fixed during construction of $\mathcal{T}_{N:0}$.

17.9 A one-pulse spectroscopy protocol

Consider

$$\boxed{\text{prepare} \to \tau_1 \to \mathcal{P} \to \tau_2 \to \text{measure},} \tag{151}$$

with

$$\tau_1 = m_1 \Delta t, \qquad \tau_2 = t_N - \tau_1. \tag{152}$$

Let the prepared state be ρ_{prep}. All unused process-tensor slots are contracted with the identity map $\mathcal{I}$, while the pulse slot is contracted with $\mathcal{P}$. For a unitary pulse,

$$\mathcal{P}[\rho] = U_P \rho U_P^\dagger, \tag{153}$$

and, for an x rotation through angle θ,

$$U_P(\theta) = \exp\left(-i\frac{\theta}{2}\sigma_x\right). \tag{154}$$

The reduced state is therefore

$$\boxed{\rho_S(t_N) = \mathcal{T}_{N:0}\left[\mathcal{I}, \ldots, \mathcal{P}, \ldots, \mathcal{I}; \rho_{\text{prep}}\right].} \tag{155}$$

If the measured observable is M,

$$\boxed{S(\tau_1, \tau_2) = \text{Tr}\left[M \rho_S(t_N)\right].} \tag{156}$$

17.10 What can be changed without rebuilding this process tensor?

Once the system-dressed tensor $\mathcal{T}_{N:0}$ has been constructed, the same tensor can be contracted repeatedly with different intervention maps. For example,

$$\mathcal{P}_{\pi/2} \longrightarrow \mathcal{P}_\pi \tag{157}$$

changes the pulse angle without changing the bath-memory network or the stored free-system propagation.

Similarly, moving the pulse from t_{m_1} to $t_{m_1'}$ only changes which open slot contains $\mathcal{P}$. Additional pulses can be inserted into other slots,

$$\mathcal{T}_{N:0}\left[\ldots, \mathcal{P}_3, \ldots, \mathcal{P}_2, \ldots, \mathcal{P}_1, \ldots; \rho_{\text{prep}}\right]. \tag{158}$$

The same tensor may also be contracted with different initial states or final observables. In the notebook convention, however, G is already contained in $\mathcal{T}_{N:0}$. A change in the free-system Hamiltonian therefore requires rebuilding the tensor. If instead one stores the bath-only object $\mathcal{T}^{(B)}_{N:0}$, different $H_S(t)$ can be supplied during the later contraction without recomputing the environmental influence.

17.11 Multipulse spectroscopy

A multipulse sequence,

$$\text{prepare} \to \tau_1 \to \mathcal{P}_1 \to \tau_2 \to \mathcal{P}_2 \to \tau_3 \to \mathcal{P}_3 \to \text{measure}, \tag{159}$$

requires no new bath construction. If the pulses occur at $t_{m_1}, t_{m_2}, t_{m_3}$, one inserts

$$\mathcal{A}_{m_1} = \mathcal{P}_1, \qquad \mathcal{A}_{m_2} = \mathcal{P}_2, \qquad \mathcal{A}_{m_3} = \mathcal{P}_3, \tag{160}$$

and contracts every other intervention slot with $\mathcal{I}$.

This is the basic reason process tensors are useful for multidimensional spectroscopy and control: one expensive temporal-memory object can support many different multitime contractions.

17.12 Ordinary TEMPO, the present process tensor, and bath-only PT-TEMPO

The three contractions can be summarized as

$$\begin{array}{|ccc|}\hline \text{ordinary TEMPO} & \text{system-dressed PT} & \text{bath-only PT-TEMPO} \\ \{I\}+G+\{\mathcal{A}_k\} & \{I\}+G & \{I\} \\ \Downarrow & \Downarrow & \Downarrow \\ \rho_S(t_N) & \mathcal{T}_{N:0} & \mathcal{T}^{(B)}_{N:0} \\ & +\{\mathcal{A}_k\} & +\{G_k,\mathcal{A}_k\} \\ & \Downarrow & \Downarrow \\ & \rho_S(t_N) & \rho_S(t_N). \\ \hline \end{array} \tag{161}$$

The notebook below implements the middle column.

17.13 Where the bath memory appears in the process tensor

The factor

$$I_{k,k-j} \tag{162}$$

means that a system configuration at time t_{k-j} can modify the bath-mediated response at the later time t_k. With finite memory, only $j \le K$ survives, so the process tensor carries explicit environmental correlations over approximately

$$\tau_{\mathrm{mem}} = K\Delta t. \tag{163}$$

The temporal MPS bond indices are the compressed representation of this memory.

Schematically,

$$\begin{array}{|c|}\hline \alpha_B(t) \\ \Downarrow \\ \eta_j \\ \Downarrow \\ I_j \\ \Downarrow \\ \text{finite-memory temporal tensor} \\ \Downarrow \quad \text{SVD compression} \\ \text{compressed bath-memory representation} \\ \Downarrow \quad +G \\ \mathcal{T}_{N:0} \text{ with open intervention slots} \\ \Downarrow \quad +\{\mathcal{A}_k\},\ \rho_S(0) \\ \rho_S(t_N) \\ \Downarrow \\ \text{spectroscopic signal.} \\ \hline \end{array} \tag{164}$$

17.14 Pedagogical walkthrough of the process-tensor spectroscopy code

The notebook `process_tensor_spectroscopy.ipynb` implements the construction above directly using the persistent-MPS machinery developed for QUAPI. It is intentionally written for transparency rather than maximum performance. This makes it useful for seeing exactly which tensor indices are contracted and which are left open.

The computational flow is

$$\boxed{\alpha_B(t) \to \eta_j \to I_j \to \text{persistent temporal MPS} \to \mathcal{T}_{N:0} \to \{\mathcal{A}_k\} \to S.} \tag{165}$$

17.14.1 Model and numerical parameters

The code uses the spin-boson Hamiltonian

$$H_S = \frac{\epsilon}{2}\sigma_z + \frac{\Delta}{2}\sigma_x, \qquad H_{SB} = \sigma_z \otimes B, \tag{166}$$

with an Ohmic spectral density with exponential cutoff,

$$J(\omega) = 2\pi\alpha_{\mathrm{SB}}\,\omega e^{-\omega/\omega_c}, \tag{167}$$

and $\hbar = k_B = 1$.

The numerical values used in the stored notebook output are

parameter	value	meaning
ϵ	1.0	bias energy
Δ	1.0	tunneling matrix element
α_{SB}	0.08	system–bath coupling strength
ω_c	5.0	Ohmic cutoff frequency
T	0.5	bath temperature
Δt	0.10	time step
K	6	memory depth
$K\Delta t$	0.60	retained memory time
N	18	number of propagation intervals
$t_{\max}$	1.80	process-tensor horizon
ϵ_{rel}	10^{-10}	relative SVD cutoff
$\chi_{\max}$	48	maximum retained MPS bond
n_{cell}	21	quadrature points per time cell
n_ω	1400	frequency-grid points

There are $N - 1 = 17$ open intervention slots. The endpoint q_0 is reserved for the initial density matrix, and the endpoint q_N remains open as the final reduced state.

17.14.2 From the bath correlation function to the influence coefficients

The routine `_bath_alpha_for_mps` evaluates

$$\alpha_B(t) = \frac{1}{\pi}\int_0^\infty d\omega\, J(\omega)\left[\coth\left(\frac{\beta\omega}{2}\right)\cos(\omega t) - i\sin(\omega t)\right] \tag{168}$$

by direct numerical quadrature. The helper `coth_stable` replaces the numerically ill-conditioned small-argument evaluation of $\coth x$ by its series expansion.

The routine `influence_coefficients_mps` then integrates $\alpha_B(t)$ over pairs of time cells to obtain the finite-cell coefficients η_j. For $j = 0$, only the causal triangle of the double time-cell integral is retained.

For the parameters above, the notebook obtains

$$|\eta_0| = 1.93 \times 10^{-2}, \qquad |\eta_1| = 3.08 \times 10^{-2}, \qquad |\eta_6| = 3.30 \times 10^{-3}. \tag{169}$$

The lag-one coefficient is the largest because the two complete neighboring time cells contain a strong short-time bath correlation, whereas the $j = 0$ coefficient is integrated only over the causal triangle.

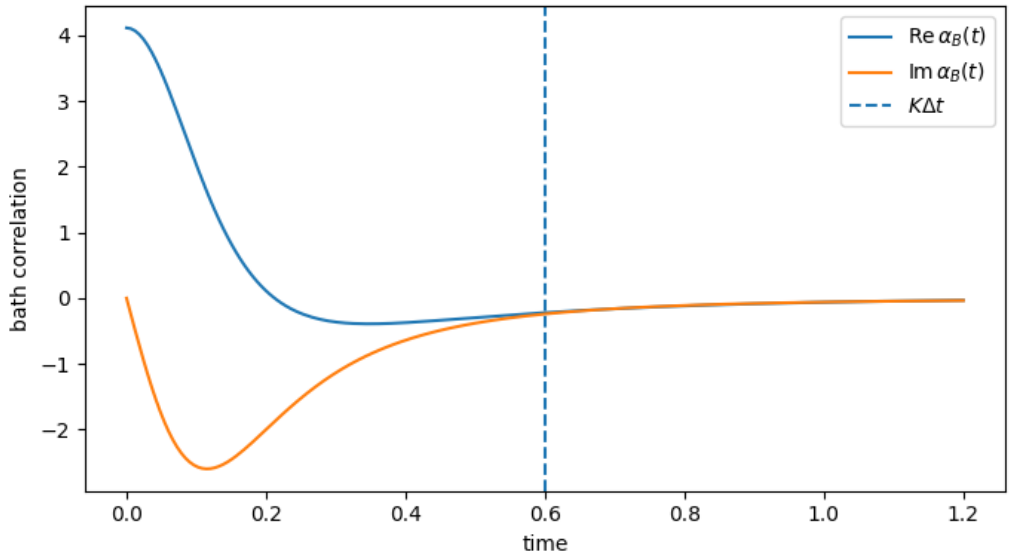

Figure 11: Bath correlation function used by the process-tensor calculation. The real and imaginary components of $\alpha_B(t)$ decay rapidly from their short-time values. The vertical dashed line marks the chosen memory time $K\Delta t = 0.60$. The correlations are already much smaller there than at $t = 0$, but they are not exactly zero; therefore $K = 6$ must be regarded as a numerical choice to be tested by a memory-convergence calculation rather than as an exact cutoff.

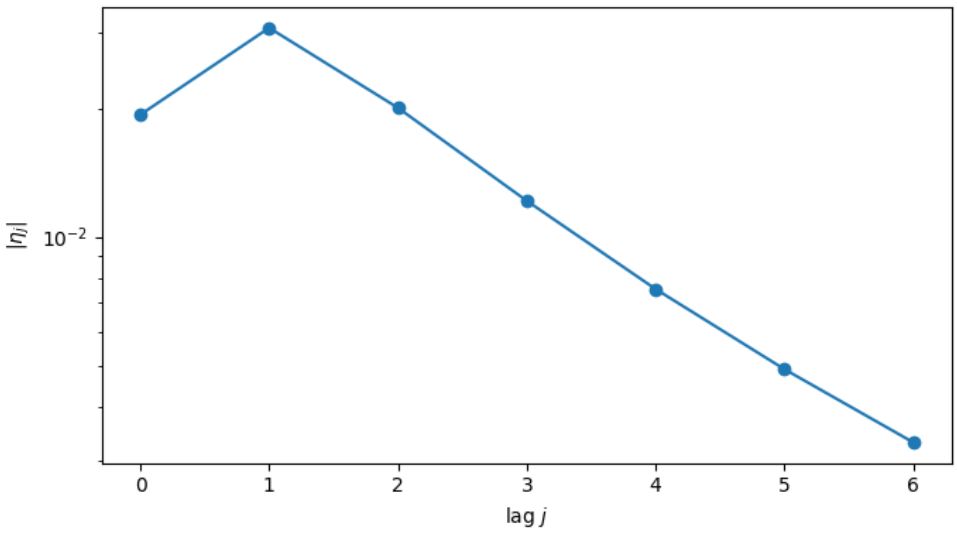

Figure 12: Magnitude of the finite-cell influence coefficients η_j. The decay with lag j shows why a finite-memory approximation is plausible for this example. The nonzero value at $j = 6$, however, also emphasizes that the selected memory depth must be checked for convergence.

17.14.3 Liouville-space system kernel and influence tables

The code orders the forward/backward qubit path pair as

$$a = 2s^+ + s^-, \qquad a = 0, \ldots, 3. \tag{170}$$

The function `system_kernel_dt` constructs the 4×4 Liouville kernel

$$G(a', a) = \langle s'^+ | U_{\Delta t} | s^+ \rangle \langle s'^- | U_{\Delta t} | s^- \rangle^*. \tag{171}$$

The function `influence_tables_mps` converts each η_j into a 4×4 pair-influence table,

$$I_j(a_k, a_{k-j}) = \exp\left[-(s_k^+ - s_k^-) \left(\eta_j s_{k-j}^+ - \eta_j^* s_{k-j}^- \right) \right]. \tag{172}$$

Keeping these two objects conceptually separate is useful: G carries the fixed isolated-system evolution, whereas the I_j carry the environmental memory.

17.14.4 Persistent-MPS primitives and why the global scale is retained

Each temporal MPS core has the shape

$$(\chi_{\mathrm{L}}, d_{\mathrm{phys}}, \chi_{\mathrm{R}}). \tag{173}$$

The routine `right_canonicalize_mps` performs a right-to-left QR canonicalization, and `compress_process_tensor_` subsequently performs local SVD truncations.

A subtle but important implementation detail is the variable `global_scale`. During ordinary state propagation, one might be tempted to renormalize the state after a compression step. That is not allowed for a reusable process tensor because an inserted operation may be trace nonpreserving. Renormalization would destroy linearity. The code therefore extracts numerical scale factors for conditioning but multiplies them back into the final contraction.

17.14.5 How an open intervention slot is created

The key routine is `add_open_intervention_slot`. It introduces the grouped physical index

$$x_k = (q_k, r_k), \qquad x_k = 4q_k + r_k, \tag{174}$$

but does not insert a particular $[\mathcal{A}_k]_{q_k r_k}$.

The local sequence represented in the tensor is

$$q_{k-1} \xrightarrow{G} r_k \xrightarrow{\mathcal{A}_k} q_k. \tag{175}$$

Only the first arrow is contracted during construction. The second arrow is left open. At the same time, the post-intervention index q_k is threaded backward through at most K temporal bonds so that the newly introduced factors

$$I_0(q_k), I_1(q_k, q_{k-1}), \ldots, I_K(q_k, q_{k-K}) \tag{176}$$

can be attached.

The function `add_final_state_leg` performs the last propagation step without adding another intervention. Consequently, after construction the physical dimensions, ordered from newest to oldest, are

$$\boxed{[4, 16, 16, \ldots, 16, 4].} \tag{177}$$

For this notebook there are seventeen 16-dimensional operation slots between the two four-dimensional endpoint legs.

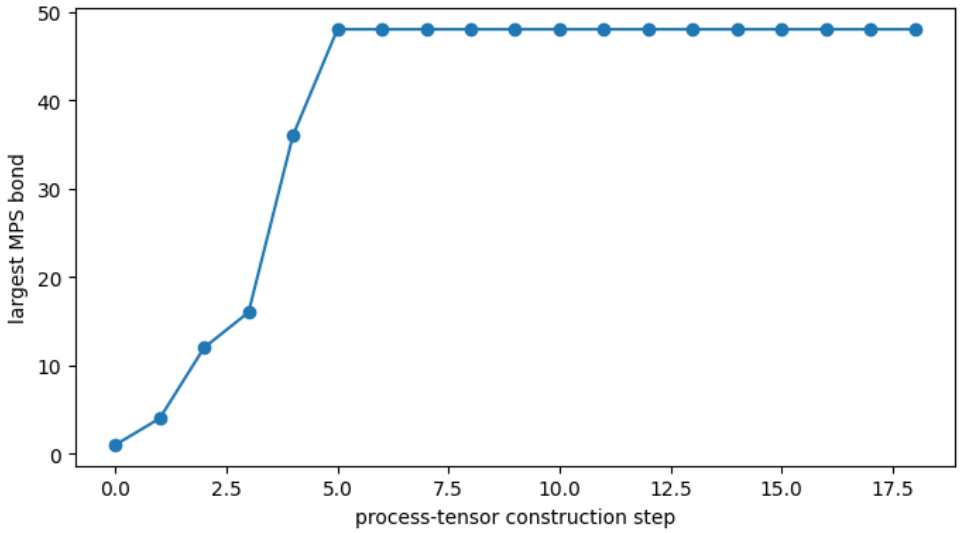

Figure 13: Largest retained MPS bond during process-tensor construction. The bond grows rapidly as temporal correlations accumulate and reaches $\chi_{\max} = 48$ after approximately five construction steps. The subsequent plateau is caused by the imposed bond-dimension cap and must not be interpreted as evidence that the exact process tensor naturally saturates at dimension 48.

17.14.6 Building the stored process tensor

The function `build_process_tensor_mps` performs the complete construction. At each step it

1. adds one open intervention slot;
2. attaches the bath factors that lie inside the memory window;
3. incorporates the fixed kernel G;
4. canonicalizes and compresses the enlarged temporal MPS;
5. stores diagnostics for bond dimension, memory use, and discarded weight.

The stored notebook output reports a build time of 18.010 s, a largest retained bond dimension of 48, and a peak of 522 528 stored MPS coefficients. The runtime is machine dependent; the tensor dimensions are the more meaningful algorithmic diagnostics.

This last point is important. Because the calculation hits $\chi_{\max} = 48$, the displayed results are a pedagogical demonstration, not by themselves a proof of bond-dimension convergence.

17.14.7 Contracting an experiment with the stored tensor

The routine `contract_process_tensor` first contracts the oldest four-dimensional leg with ρ_0. It then moves forward through the 16-dimensional slots. If the user has supplied a map $\mathcal{A}_k$, its 4×4 Liouville matrix is flattened to match the grouped physical leg; otherwise the identity superoperator is inserted. Finally, the newest four-dimensional leg is left as the vectorized final density matrix.

For the initial state

$$\rho_{+x} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \tag{178}$$

the no-pulse contraction gives

$$\rho_S(t_N) \simeq \begin{pmatrix} 0.551731 & -0.116027 + 0.069395\,i \\ -0.116027 - 0.069395\,i & 0.448179 \end{pmatrix}. \tag{179}$$

The reported trace is 0.9999107, while the Hermiticity error is 2.65×10^{-14}. The essentially machine-precision Hermiticity is a useful algebraic check. The trace error of order 10^{-4} is instead a numerical approximation error and should be monitored in the convergence study.

17.14.8 Exact small-network validation

Before interpreting any spectroscopy result, the notebook validates the open-slot construction against ordinary finite-memory QUAPI for a small network with no SVD truncation. At a pulse time,

$$q_{k-1} \xrightarrow{G} r_k \xrightarrow{P} q_k, \tag{180}$$

so direct propagation can use

$$G_{\text{eff}} = PG. \tag{181}$$

The process-tensor contraction and direct controlled-QUAPI propagation agree with

$$\max |\rho_{\text{PT}} - \rho_{\text{QUAPI}}| = 5.7 \times 10^{-15}. \tag{182}$$

This roundoff-level agreement is an important validation of the index ordering, the placement of G, and the open intervention contraction.

17.14.9 One-pulse spectroscopy

The first spectroscopy example prepares ρ_{+x}, applies a $\pi/2$ pulse about x at

$$\tau_1 = 0.60, \qquad \tau_2 = 1.20, \tag{183}$$

and measures σ_z at $t_N = 1.80$. The notebook obtains

$$S(\tau_1, \tau_2) = \langle \sigma_z(t_N) \rangle = -0.02238411, \tag{184}$$

with

$$\operatorname{Tr} \rho_S(t_N) = 0.9999174. \tag{185}$$

The numerical value itself is less important than the workflow: the pulse is inserted only during contraction of the already stored process tensor.

17.14.10 Reuse 1: scanning the pulse angle

The notebook next replaces a single pulse superoperator $\mathcal{P}_{\pi/2}$ by $\mathcal{P}_\theta$ while keeping every other tensor unchanged. Thirty-one pulse angles are evaluated in 0.137 s in the stored run, compared with 18.010 s for constructing the reusable tensor.

For this grid the signal ranges from approximately -0.1049 near $\theta \simeq 0.93\pi$ to 0.1061 near $\theta \simeq 1.93\pi$. The scan is a direct demonstration of process-tensor reuse: no bath coefficient, influence table, or temporal MPS core is rebuilt.

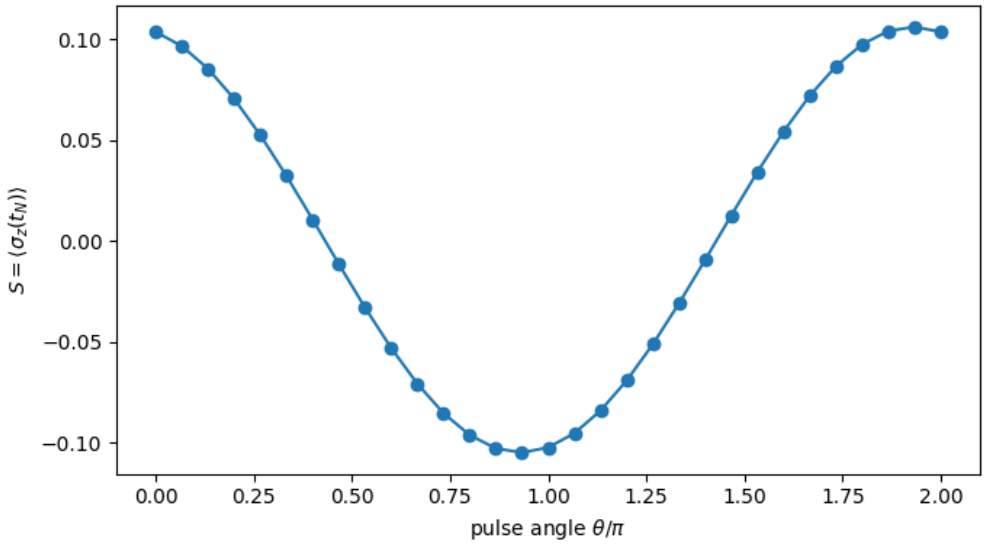

Figure 14: Final signal $S = \langle \sigma_z(t_N) \rangle$ as a function of the x-pulse rotation angle. The same stored process tensor is contracted for every point; only the 4×4 pulse superoperator changes. The nearly sinusoidal response illustrates the continuous dependence of the final state on the intervention.

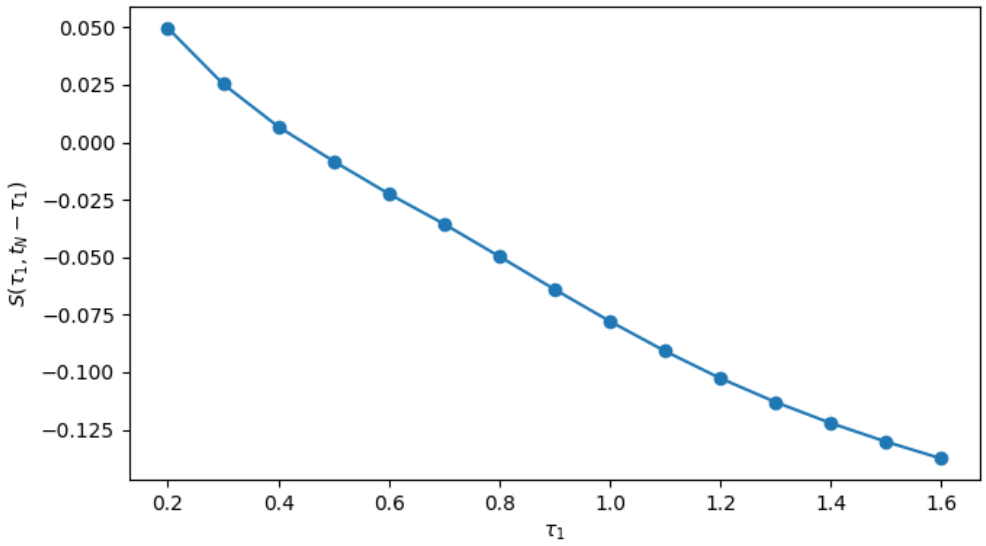

Figure 15: Final σ_z signal for a fixed $\pi/2$ x pulse inserted at different times τ_1, with the final time held at $t_N = 1.80$ and $\tau_2 = t_N - \tau_1$. Moving the pulse changes only which process-tensor slot contains the pulse superoperator.

17.14.11 Reuse 2: moving the pulse in time

Changing τ_1 is equally simple. The pulse matrix is unchanged; it is contracted into a different open slot.

The signal decreases from approximately 0.0497 for $\tau_1 = 0.20$ to -0.1376 for $\tau_1 = 1.60$. Physically, this reflects the fact that the state and its bath correlations at the instant of the pulse depend on the preceding evolution. Computationally, it demonstrates why the multitime object is more useful than a single stored trajectory.

17.14.12 Extracting an earlier-time observable from the longest tensor

To read an observable at an intermediate time t_m without constructing a shorter process tensor, the notebook inserts the linear map

$$\boxed{\mathcal{R}_M[\rho] = \mathrm{Tr}(M\rho)\rho_{\mathrm{reset}}, \qquad \mathrm{Tr}\,\rho_{\mathrm{reset}} = 1.} \tag{186}$$

Trace-preserving identity operations are placed after this slot. Causality then gives

$$\mathrm{Tr}\,\rho(t_N) = \mathrm{Tr}[M\rho(t_m)]. \tag{187}$$

When M is not a positive operator—as for $M = \sigma_z$—the map $\mathcal{R}_M$ is not a physical measurement channel. It is an algebraic tensor-contraction device that exploits the multilinearity and causality of the process tensor. The notebook explicitly uses it only in this computational sense.

A separate uncompressed causality test compares a process tensor terminating at t_m with the longer tensor plus Eq. (186). The two signals differ by only

$$4.52 \times 10^{-15}, \tag{188}$$

again confirming the contraction logic to roundoff.

17.14.13 Reuse 3: a two-delay spectroscopy map

The previous trick allows a two-delay quantity to be extracted from one longest tensor. For each pair

$$\tau_1 = m_1\Delta t, \qquad \tau_2 = m_2\Delta t, \tag{189}$$

the pulse is inserted at m_1 and the observable contraction at $m_1 + m_2$.

On the displayed 4×4 grid, the largest value is approximately 0.1905 at $(\tau_1, \tau_2) = (0.40, 0.20)$, whereas the value at $(0.80, 0.80)$ is slightly negative, approximately -0.0046. The nonseparable dependence on the two delays reflects the combined effect of coherent system evolution and environmental memory.

17.14.14 Reuse 4: a multipulse sequence

The notebook also inserts three pulses into the same tensor,

$$\mathcal{P}_x(\pi/2) \quad \text{at step 3}, \qquad \mathcal{P}_y(\pi/2) \quad \text{at step 7}, \qquad \mathcal{P}_x(\pi) \quad \text{at step 12}. \tag{190}$$

The resulting final signal is

$$\langle\sigma_z(t_N)\rangle = 0.2455781, \tag{191}$$

with final trace 0.9999283. No new process tensor is built. This is exactly the operation-slot picture required for more elaborate multipulse protocols.

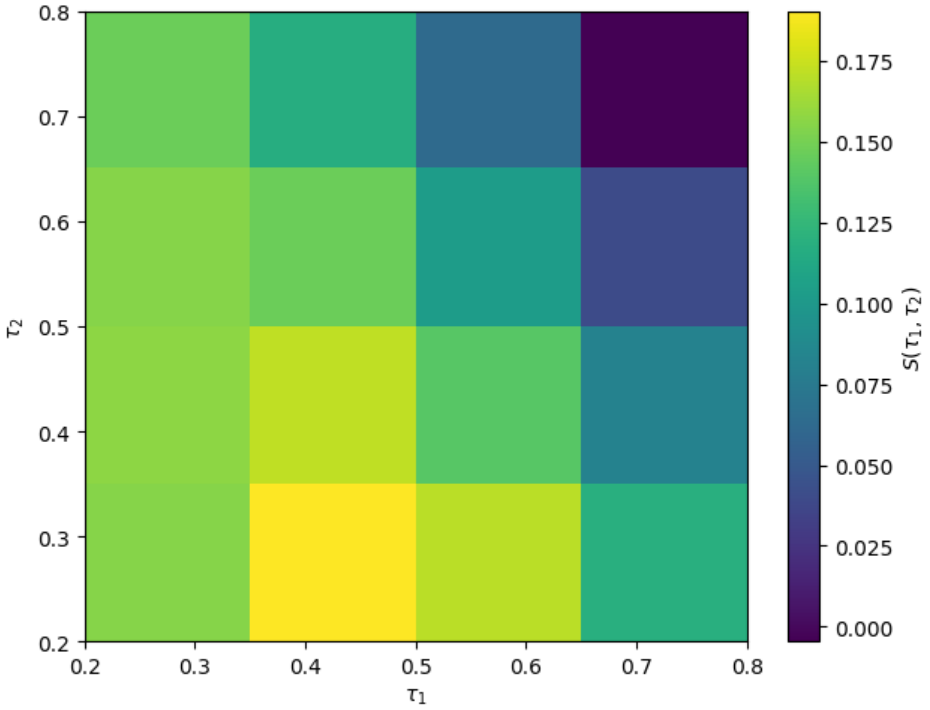

Figure 16: Two-delay signal $S(\tau_1, \tau_2)$ obtained from a single stored process tensor. For every grid point the pulse and readout maps are contracted into different temporal slots; the bath-memory tensor is unchanged. The map therefore illustrates directly how a process tensor converts one expensive multitime calculation into many inexpensive spectroscopy contractions.

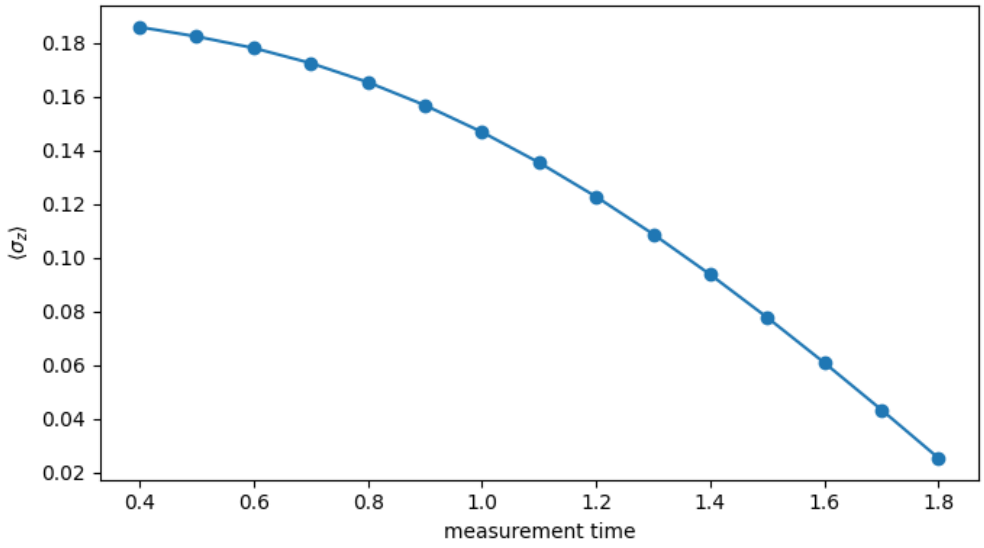

Figure 17: Post-pulse time-domain signal extracted from the longest process tensor. The observable is evaluated at different times without rebuilding shorter tensors. For the present parameters, $\langle\sigma_z\rangle$ decreases monotonically from approximately 0.186 at $t = 0.40$ to 0.0254 at $t = 1.80$.

17.14.15 Time-domain signal and illustrative Fourier transform

After a $\pi/2$ x pulse at step 3, the notebook extracts $\langle\sigma_z(t)\rangle$ at many later times by moving the observable contraction through the existing slots.

The signal is then mean-subtracted, multiplied by a Hann window, and Fourier transformed.

This distinction is essential. A formal optical absorption or multidimensional spectrum additionally requires a specified dipole operator, pulse convention, appropriate response function, and, when relevant, Liouville pathways or phase cycling. The Fourier transform here demonstrates only how a time-dependent observable can be extracted and postprocessed from a reusable process tensor.

17.14.16 What has been validated, and what still requires convergence

The notebook provides two strong structural checks:

1. the open-slot process tensor agrees with direct uncompressed controlled QUAPI to approximately 5.7×10^{-15} for a small test problem;
2. the intermediate-time readout obtained from the long tensor agrees with a short tensor ending at that time to approximately 4.5×10^{-15}.

These checks validate the tensor contractions and index conventions. They do *not* establish numerical convergence of the production-size example.

Three limits should be tested independently:

1. increase K at fixed Δt until observables stop changing;
2. reduce Δt while keeping the physical memory time $K\Delta t$ fixed or larger;
3. tighten the SVD threshold and increase $\chi_{\max}$ until the selected signals and trace errors are stable.

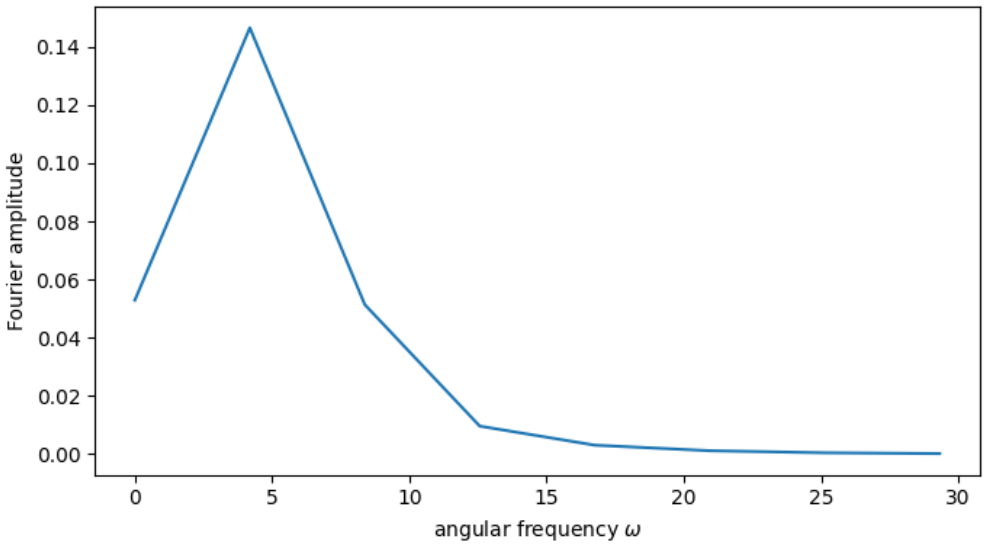

Figure 18: Magnitude of the discrete Fourier transform of the windowed time-domain signal. The largest nonzero Fourier bin occurs at $\omega \simeq 4.19$ in the chosen units. This spectrum is intentionally illustrative: the record contains only fifteen points separated by $\Delta t = 0.10$, so the frequency spacing is $\Delta\omega = 2\pi/(15\Delta t) \simeq 4.19$. The plot therefore does not resolve a spectroscopic line shape.

The present calculation reaches the imposed cap $\chi_{\max} = 48$, so the bond-dimension limit is active. Moreover, the optional convergence cell in the uploaded notebook is disabled. Consequently, the figures above should be described as a pedagogical demonstration of the process-tensor workflow rather than as fully converged benchmark data. For a publication-quality numerical result, the convergence scan should be enabled and the resulting observables reported as functions of K, Δt, the SVD tolerance, and $\chi_{\max}$.

17.14.17 What the code demonstrates

The notebook makes the process-tensor idea concrete:

$$\boxed{\begin{array}{c|c} \text{ordinary persistent-MPS QUAPI} & \text{process-tensor persistent MPS} \\ \{I_j\} + G + \text{chosen interventions} & \{I_j\} + G \\ \Downarrow & \Downarrow \\ \rho_S(t) & \mathcal{T}_{N:0} \text{ with open } (q_k, r_k) \text{ slots} \\ & +\{\mathcal{A}_k\},\ \rho_S(0) \\ & \Downarrow \\ & \rho_S(t). \end{array}} \tag{192}$$

The computational saving appears when many experiments share the same bath and the same fixed free-system propagator G. The relatively expensive temporal memory construction is performed once; pulse angles, pulse times, pulse sequences, readout times, observables, and initial states are then changed by small contractions with the stored tensor.

Key idea. The process tensor is obtained by deciding *not* to contract selected system-operation legs while the temporal memory network is built. In the pedagogical implementation used here, the Gaussian bath influence and the fixed free-system propagator G

are incorporated into a persistent temporal MPS, whereas each intervention remains as an open input–output pair (r_k, q_k). This produces a reusable operational process tensor for the fixed uncontrolled dynamics.

A bath-only PT-TEMPO factorization goes one step further and also leaves the system propagators outside the stored environmental tensor. That variant can be reused across different time-dependent system Hamiltonians. Keeping these two conventions explicit prevents a common source of confusion and makes clear exactly what is, and is not, being reused in the code.

Appendices: foundations and implementation

The appendices are written so that each can be read as a self-contained derivation, but each also ends by identifying the exact result returned to the main text. Readers who are comfortable with a particular foundation can skip its derivation and use the boxed result as an entry point back into Parts I–IV.

Appendix A derives the real-time coordinate propagator and its path-integral limit. Appendix B derives the imaginary-time harmonic-oscillator kernel, the normalized thermal density matrix used for the initial bath, and the corresponding real-time harmonic-oscillator propagator. Appendix C derives the thermal bath correlation function $C(t)$ from oscillator occupation numbers and the spectral density. Appendix D maps the finite-memory propagation onto the Python implementation; Appendix E collects the complete convergence values; and Appendix G contains the full executable source.

A Foundation: from short-time propagators to the path integral

> **Checkpoint. Why this appendix is here.** The main text uses symmetric short-time propagators in Sec. 4. This appendix derives those coordinate-space kernels, shows how finite-time propagation is assembled from them, and identifies the continuum action. It can be skipped on a first reading without interrupting the discrete bath elimination.

This appendix supplies the coordinate-space construction underlying the short-time system kernels used in the main text. It is not needed for the direct Gaussian elimination of the bath, but it makes explicit how a finite-time propagator is assembled from short-time factors. We first derive the free-particle kernel, then include a local potential with the same symmetric Trotter factorization used in Eq. (20), and finally take the continuum limit. Throughout this appendix, real-time Gaussian integrals are defined with the usual prescription $\tau \to \tau - \mathrm{i}0^+$.

A.1 Short-time real-time propagator for a free particle

We begin with the free-particle Hamiltonian

$$\hat{H}_0 = \frac{\hat{p}^2}{2m}. \tag{193}$$

For a short time interval τ, the propagator is

$$K_0(x', \tau; x, 0) = \langle x' | e^{-i\hat{p}^2\tau/(2m\hbar)} | x \rangle. \tag{194}$$

Because the free Hamiltonian is diagonal in the momentum basis, it is natural to insert the momentum-space resolution of the identity,

$$\hat{I} = \int_{-\infty}^{\infty} dp \, |p\rangle\langle p|. \tag{195}$$

Equation (194) becomes

$$K_0(x', \tau; x, 0) = \int_{-\infty}^{\infty} dp \, \langle x'|p\rangle e^{-ip^2\tau/(2m\hbar)} \langle p|x\rangle. \tag{196}$$

Using

$$\langle x|p\rangle = \frac{1}{\sqrt{2\pi\hbar}} e^{ipx/\hbar}, \tag{197}$$

we obtain

$$K_0(x',\tau;x,0) = \frac{1}{2\pi\hbar}\int_{-\infty}^{\infty} dp\, \exp\left[-\frac{i\tau}{2m\hbar}p^2 + \frac{i}{\hbar}p(x'-x)\right]. \tag{198}$$

This is a Gaussian integral. The only nontrivial step is to identify the coefficient of p^2 and the linear term in p; the integration then has the standard form obtained by shifting the integration variable to the center of the quadratic form.

Completing the square gives

$$-\frac{i\tau}{2m\hbar}p^2 + \frac{i}{\hbar}p(x'-x) = -\frac{i\tau}{2m\hbar}\left[p - \frac{m(x'-x)}{\tau}\right]^2 + \frac{im(x'-x)^2}{2\hbar\tau}. \tag{199}$$

Therefore,

$$\begin{aligned} K_0(x',\tau;x,0) = {} & \frac{1}{2\pi\hbar}\exp\left[\frac{im(x'-x)^2}{2\hbar\tau}\right] \\ & \times \int_{-\infty}^{\infty} dp\, \exp\left[-\frac{i\tau}{2m\hbar}\left(p - \frac{m(x'-x)}{\tau}\right)^2\right]. \end{aligned} \tag{200}$$

Evaluating the Gaussian integral gives

$$\boxed{K_0(x',\tau;x,0) = \sqrt{\frac{m}{2\pi i\hbar\tau}}\exp\left[\frac{i}{\hbar}\frac{m(x'-x)^2}{2\tau}\right].} \tag{201}$$

An important observation is that

$$\frac{x'-x}{\tau} \tag{202}$$

has the form of a finite-difference velocity. Thus the exponent can be written as

$$\frac{i}{\hbar}\left[\frac{1}{2}m\left(\frac{x'-x}{\tau}\right)^2\right]\tau. \tag{203}$$

The kinetic contribution to the classical action has therefore already appeared naturally in the free-particle propagator.

A.2 Short-time propagator for a general potential

We now consider

$$\hat{H} = \hat{T} + \hat{V}, \tag{204}$$

where

$$\hat{T} = \frac{\hat{p}^2}{2m}, \qquad \hat{V} = V(\hat{x}). \tag{205}$$

Since $\hat{T}$ and $\hat{V}$ generally do not commute,

$$[\hat{T},\hat{V}] \neq 0, \tag{206}$$

we cannot simply write

$$e^{-i\tau(\hat{T}+\hat{V})/\hbar} = e^{-i\tau\hat{T}/\hbar}e^{-i\tau\hat{V}/\hbar} \tag{207}$$

as an exact identity.

For a sufficiently short time step τ, however, we can use the symmetric Trotter–Suzuki factorization,

$$\boxed{e^{-i\tau(\hat{T}+\hat{V})/\hbar} = e^{-i\tau\hat{V}/(2\hbar)}e^{-i\tau\hat{T}/\hbar}e^{-i\tau\hat{V}/(2\hbar)} + \mathcal{O}(\tau^3).} \tag{208}$$

Taking the position-space matrix element gives

$$\langle x'|e^{-i\tau\hat{H}/\hbar}|x\rangle \simeq \langle x'|e^{-i\tau\hat{V}/(2\hbar)}e^{-i\tau\hat{T}/\hbar}e^{-i\tau\hat{V}/(2\hbar)}|x\rangle. \tag{209}$$

Because position eigenstates are eigenstates of $\hat{V}$,

$$e^{-i\tau\hat{V}/(2\hbar)}|x\rangle = e^{-i\tau V(x)/(2\hbar)}|x\rangle, \tag{210}$$

$$\langle x'|e^{-i\tau\hat{V}/(2\hbar)} = e^{-i\tau V(x')/(2\hbar)}\langle x'|. \tag{211}$$

Therefore,

$$\langle x'|e^{-i\tau\hat{H}/\hbar}|x\rangle \simeq e^{-i\tau V(x')/(2\hbar)}\langle x'|e^{-i\tau\hat{T}/\hbar}|x\rangle e^{-i\tau V(x)/(2\hbar)}. \tag{212}$$

We can now insert the free-particle result, Eq. (201):

$$\begin{aligned}\langle x'|e^{-i\tau\hat{H}/\hbar}|x\rangle \simeq \sqrt{\frac{m}{2\pi i\hbar\tau}}\exp\left[\frac{i}{\hbar}\frac{m(x'-x)^2}{2\tau}\right] \\ \times\exp\left[-\frac{i\tau}{2\hbar}\left(V(x')+V(x)\right)\right].\end{aligned} \tag{213}$$

Combining the two exponentials gives

$$\boxed{\begin{aligned}\langle x'|e^{-i\tau\hat{H}/\hbar}|x\rangle \simeq \sqrt{\frac{m}{2\pi i\hbar\tau}} \\ \times\exp\left\{\frac{i}{\hbar}\left[\frac{1}{2}m\left(\frac{x'-x}{\tau}\right)^2 - \frac{1}{2}\left(V(x')+V(x)\right)\right]\tau\right\}.\end{aligned}} \tag{214}$$

This is the short-time propagator for a particle moving in an arbitrary potential $V(x)$.

The quantity inside square brackets,

$$\frac{1}{2}m\left(\frac{x'-x}{\tau}\right)^2 - \frac{1}{2}\left[V(x')+V(x)\right], \tag{215}$$

is already recognizable as a discrete approximation to the classical Lagrangian

$$L(x,\dot{x}) = \frac{1}{2}m\dot{x}^2 - V(x). \tag{216}$$

A.3 From short-time to finite-time propagation

We now divide the total propagation time t into n equal intervals,

$$t = n\tau, \qquad \tau = \frac{t}{n}. \tag{217}$$

The evolution operator satisfies

$$e^{-i\hat{H}t/\hbar} = \left(e^{-i\hat{H}\tau/\hbar}\right)^n. \tag{218}$$

Introduce the coordinates

$$x_0 = x_i, \qquad x_n = x_f, \tag{219}$$

and insert $n-1$ resolutions of the identity,

$$\hat{I} = \int dx_j \, |x_j\rangle\langle x_j|, \tag{220}$$

between the short-time evolution operators.

The finite-time propagator becomes

$$K(x_n, t; x_0, 0) = \int dx_1 \cdots dx_{n-1} \times \prod_{j=1}^{n} \langle x_j | e^{-i\hat{H}\tau/\hbar} | x_{j-1} \rangle. \tag{221}$$

Substituting the short-time propagator Eq. (214) gives

$$K(x_n, t; x_0, 0) \simeq \int dx_1 \cdots dx_{n-1} \left(\frac{m}{2\pi i\hbar\tau}\right)^{n/2} \times \exp\left\{\frac{i}{\hbar}\sum_{j=1}^{n}\left[\frac{1}{2}m\left(\frac{x_j - x_{j-1}}{\tau}\right)^2 - \frac{1}{2}\left(V(x_j) + V(x_{j-1})\right)\right]\tau\right\}. \tag{222}$$

Thus the finite-time propagator is written as an integral over all intermediate positions

$$x_1, x_2, \ldots, x_{n-1}. \tag{223}$$

Each particular set

$$\{x_0, x_1, \ldots, x_{n-1}, x_n\} \tag{224}$$

defines a discrete "trajectory" joining x_0 to x_n. The integral does not select one trajectory; it sums the amplitudes associated with every possible choice of the intermediate coordinates. The continuum path integral is simply the limit in which this piecewise-defined history is refined indefinitely.

A.4 Recognizing the classical action

Define the discrete velocity on the jth time interval as

$$\dot{x}_j \approx \frac{x_j - x_{j-1}}{\tau}. \tag{225}$$

Equation (222) contains the discrete quantity

$$S_n[x] = \sum_{j=1}^{n} \left[\frac{1}{2} m \left(\frac{x_j - x_{j-1}}{\tau} \right)^2 - \frac{1}{2} \left(V(x_j) + V(x_{j-1}) \right) \right] \tau. \tag{226}$$

This is a discretized approximation to

$$\int_0^t dt' \, \left[\frac{1}{2} m \dot{x}^2(t') - V(x(t')) \right]. \tag{227}$$

Indeed, in the limit

$$n \to \infty, \qquad \tau \to 0, \qquad n\tau = t, \tag{228}$$

the finite differences become derivatives,

$$\frac{x_j - x_{j-1}}{\tau} \longrightarrow \dot{x}(t'), \tag{229}$$

and the sum becomes an integral,

$$\sum_{j=1}^{n} (\cdots)\tau \longrightarrow \int_0^t dt' \, (\cdots). \tag{230}$$

Consequently,

$$\boxed{S[x] = \int_0^t dt' \, \left[\frac{1}{2} m \dot{x}^2(t') - V(x(t')) \right].} \tag{231}$$

This is precisely the classical action associated with the trajectory $x(t')$.

It is important to emphasize that $x(t')$ is *not* required to be a classical trajectory satisfying Newton's equations. Rather, $S[x]$ is the classical action functional evaluated on an arbitrary path.

Equation (222) therefore takes the form

$$K(x_n, t; x_0, 0) = \lim_{\substack{n\to\infty \\ \tau\to 0}} \int dx_1 \cdots dx_{n-1} \left(\frac{m}{2\pi i \hbar \tau} \right)^{n/2} e^{iS_n[x]/\hbar}. \tag{232}$$

This motivates the definition of the path-integral measure

$$\boxed{\int \mathcal{D}x(t') \, F[x] \equiv \lim_{\substack{n\to\infty \\ \tau\to 0}} \left(\frac{m}{2\pi i \hbar \tau} \right)^{n/2} \int dx_1 \cdots dx_{n-1} \, F[x].} \tag{233}$$

The propagator can therefore be written compactly as

$$\boxed{K(x_f, t; x_i, 0) = \int_{x(0)=x_i}^{x(t)=x_f} \mathcal{D}x(t') \, \exp\left[\frac{i}{\hbar} S[x] \right].} \tag{234}$$

Equation (234) is Feynman's path-integral representation of the quantum propagator.

A.5 Physical meaning of the path integral

The path integral expresses the quantum propagator as a coherent sum over all possible trajectories joining the initial and final positions:

$$x(0) = x_i, \qquad x(t) = x_f. \tag{235}$$

Each path contributes the phase

$$\exp\left[\frac{i}{\hbar}S[x]\right]. \tag{236}$$

Thus paths with different actions interfere with one another.

In the classical limit, where the action is large compared with $\hbar$, the phase oscillates rapidly from one path to another. Most contributions cancel through destructive interference.

The dominant contributions come from paths for which the action is stationary,

$$\delta S = 0. \tag{237}$$

This condition leads to the Euler–Lagrange equation

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} = 0, \tag{238}$$

which, for

$$L = \frac{1}{2}m\dot{x}^2 - V(x), \tag{239}$$

gives

$$m\ddot{x} = -\frac{dV}{dx}. \tag{240}$$

Thus classical mechanics emerges as the stationary-phase limit of the quantum path integral.

> **Key idea. Result returned to the main text.** A finite-time real-time propagator is built from short-time kernels, and the symmetric short-time factorization used in Sec. 4 is the coordinate-space starting point of the discrete forward/backward construction. No continuous-time action is required for the bath elimination carried out in Parts I–II.

B Foundation: imaginary time and the harmonic-oscillator thermal kernel

> **Checkpoint. Why this appendix is here.** The main bath elimination uses two exact ingredients: the thermal density matrix of an uncoupled oscillator and its real-time propagator. This appendix derives both from the harmonic oscillator, including normalization through the partition function. The formulas returned to the main text are Eqs. (304) and (288).

The conceptual chain in this appendix is

$$\text{real-time propagation} \longrightarrow \text{imaginary-time propagation} \longrightarrow \langle x|e^{-\beta H}|x'\rangle \longrightarrow \rho_{\rm th}(x,x').$$

The distinction between the unnormalized kernel and the normalized thermal density matrix will be kept explicit throughout.

This appendix develops an exact result that is useful as a benchmark for imaginary-time path-integral methods. Starting from the analytic continuation of the real-time propagator, we construct the discretized Euclidean path integral and then evaluate the harmonic-oscillator thermal kernel exactly by using its spectral representation and Mehler's formula. The result provides a convenient reference for checking short-time approximations, path-integral discretizations, and path-integral Monte Carlo implementations.

B.1 From real time to imaginary time

The Boltzmann thermal operator is

$$e^{-\beta \hat{H}}, \tag{241}$$

where $\beta = (k_B T)^{-1}$. It is related to the real-time evolution operator by analytic continuation. Writing

$$t = -i\tau_E, \tag{242}$$

and setting the final imaginary time to

$$\tau_E = \beta\hbar, \tag{243}$$

gives

$$e^{-i\hat{H}t/\hbar} \longrightarrow e^{-\beta \hat{H}}. \tag{244}$$

Equivalently, at the end of the interval one may write

$$t = -i\hbar\beta. \tag{245}$$

Under this continuation,

$$dt = -i\, d\tau_E, \tag{246}$$

and the oscillatory Feynman weight

$$e^{iS[x]/\hbar} \tag{247}$$

is converted into the Euclidean weight

$$e^{-S_E[x]/\hbar}, \tag{248}$$

where

$$\boxed{S_E[x] = \int_0^{\beta\hbar} d\tau_E \left[\frac{m}{2}\left(\frac{dx}{d\tau_E}\right)^2 + V(x)\right].} \tag{249}$$

The important sign change is therefore

$$\frac{m}{2}\dot{x}^2 - V(x) \quad \longrightarrow \quad \frac{m}{2}\left(\frac{dx}{d\tau_E}\right)^2 + V(x). \tag{250}$$

The thermal density-matrix kernel has the imaginary-time path-integral representation

$$\boxed{\left\langle x_f \middle| e^{-\beta\hat{H}} \middle| x_i \right\rangle = \int_{x(0)=x_i}^{x(\beta\hbar)=x_f} \mathcal{D}x(\tau_E)\, \exp\left[-\frac{1}{\hbar} S_E[x]\right].} \tag{251}$$

Unlike the real-time phase factor, the Euclidean weight is nonoscillatory. For the usual class of real, bounded-below potentials, this is what makes importance-sampling approaches such as path-integral Monte Carlo possible.

B.2 Discretized imaginary-time path integral

Divide the interval $[0, \beta\hbar]$ into n slices. Define

$$\Delta\tau_E = \frac{\beta\hbar}{n}, \qquad \epsilon = \frac{\beta}{n}, \tag{252}$$

so that

$$\Delta\tau_E = \hbar\epsilon. \tag{253}$$

For $\hat{H} = \hat{T} + \hat{V}$, the symmetric Trotter factorization gives

$$e^{-\epsilon(\hat{T}+\hat{V})} = e^{-\epsilon\hat{V}/2} e^{-\epsilon\hat{T}} e^{-\epsilon\hat{V}/2} + \mathcal{O}(\epsilon^3). \tag{254}$$

The corresponding short-imaginary-time coordinate kernel is

$$\boxed{\begin{aligned}\left\langle x_j \middle| e^{-\epsilon\hat{H}} \middle| x_{j-1} \right\rangle \simeq \sqrt{\frac{m}{2\pi\hbar^2\epsilon}} \\ \times \exp\left[-\frac{m(x_j - x_{j-1})^2}{2\hbar^2\epsilon} - \frac{\epsilon}{2}\left(V(x_j) + V(x_{j-1})\right) \right].\end{aligned}} \tag{255}$$

Inserting $n-1$ coordinate resolutions of the identity yields

$$\begin{aligned}\left\langle x_n \middle| e^{-\beta\hat{H}} \middle| x_0 \right\rangle \simeq \left(\frac{m}{2\pi\hbar^2\epsilon}\right)^{n/2} \int dx_1 \cdots dx_{n-1} \\ \times \exp\left[-\sum_{j=1}^{n} \frac{m(x_j - x_{j-1})^2}{2\hbar^2\epsilon} \right. \\ \left. -\frac{\epsilon}{2}\sum_{j=1}^{n}\left(V(x_j) + V(x_{j-1})\right) \right].\end{aligned} \tag{256}$$

The continuum limit $n \to \infty$ recovers Eq. (251).

B.3 Specialization to the harmonic oscillator

Consider

$$\boxed{\hat{H} = \frac{\hat{p}^2}{2m} + \frac{1}{2}m\omega_0^2\hat{x}^2.} \tag{257}$$

We seek the unnormalized thermal density-matrix kernel

$$\rho(x, x'; \beta) \equiv \left\langle x \middle| e^{-\beta\hat{H}} \middle| x' \right\rangle. \tag{258}$$

The Euclidean action becomes

$$S_E[x] = \int_0^{\beta\hbar} d\tau_E \left[\frac{m}{2}\left(\frac{dx}{d\tau_E}\right)^2 + \frac{m\omega_0^2}{2}x^2 \right]. \tag{259}$$

Because this action is quadratic in the entire path $x(\tau_E)$, the functional integral is Gaussian. One can evaluate it directly by expanding around the classical Euclidean path and integrating the quadratic fluctuations.

B.4 Energy eigenstates and spectral representation

Here, we obtain the same exact kernel introduced by Eq. (259) by a particularly compact spectral calculation, which also makes the thermal-state interpretation transparent.

We introduce the ladder operators

$$\hat{a} = \sqrt{\frac{m\omega_0}{2\hbar}}\,\hat{x} + \frac{i}{\sqrt{2m\hbar\omega_0}}\,\hat{p}, \tag{260}$$

$$\hat{a}^\dagger = \sqrt{\frac{m\omega_0}{2\hbar}}\,\hat{x} - \frac{i}{\sqrt{2m\hbar\omega_0}}\,\hat{p}, \tag{261}$$

which satisfy

$$[\hat{a}, \hat{a}^\dagger] = 1. \tag{262}$$

Then

$$\hat{H} = \hbar\omega_0 \left(\hat{a}^\dagger\hat{a} + \frac{1}{2}\right), \tag{263}$$

so that

$$\boxed{E_n = \hbar\omega_0 \left(n + \frac{1}{2}\right), \qquad n = 0, 1, 2, \ldots} \tag{264}$$

and

$$e^{-\beta\hat{H}}|n\rangle = e^{-\beta E_n}|n\rangle. \tag{265}$$

Using completeness,

$$\hat{I} = \sum_{n=0}^{\infty} |n\rangle\langle n|, \tag{266}$$

we obtain

$$\rho(x, x'; \beta) = \sum_{n=0}^{\infty} e^{-\beta E_n}\langle x|n\rangle\langle n|x'\rangle \tag{267}$$

$$= \sum_{n=0}^{\infty} e^{-\beta\hbar\omega_0(n+1/2)}\psi_n(x)\psi_n(x'), \tag{268}$$

where the harmonic-oscillator eigenfunctions can be chosen real. Thus

$$\boxed{\rho(x, x'; \beta) = \sum_{n=0}^{\infty} e^{-\beta\hbar\omega_0(n+1/2)}\psi_n(x)\psi_n(x').} \tag{269}$$

The normalized coordinate-space eigenfunctions are

$$\psi_n(x) = \left(\frac{\alpha}{\pi}\right)^{1/4} \frac{1}{\sqrt{2^n n!}} H_n(\sqrt{\alpha}\,x) e^{-\alpha x^2/2}, \tag{270}$$

with

$$\alpha = \frac{m\omega_0}{\hbar}. \tag{271}$$

Define

$$u = \sqrt{\alpha}\,x, \qquad v = \sqrt{\alpha}\,x', \qquad q = e^{-\beta\hbar\omega_0}. \tag{272}$$

Substitution into Eq. (269) gives

$$\rho(x, x'; \beta) = \sqrt{\frac{\alpha}{\pi}}\, q^{1/2} e^{-(u^2+v^2)/2} \sum_{n=0}^{\infty} \frac{q^n}{2^n n!} H_n(u) H_n(v). \tag{273}$$

B.5 Summing the spectrum with Mehler's formula

For $|q| < 1$, Mehler's formula is

$$\boxed{\sum_{n=0}^{\infty} \frac{q^n}{2^n n!} H_n(u) H_n(v) = \frac{1}{\sqrt{1-q^2}} \exp\left[\frac{2quv - q^2(u^2+v^2)}{1-q^2}\right].} \tag{274}$$

Therefore

$$\rho(x, x'; \beta) = \sqrt{\frac{\alpha}{\pi}} \frac{q^{1/2}}{\sqrt{1-q^2}} \exp\left[-\frac{u^2+v^2}{2} + \frac{2quv - q^2(u^2+v^2)}{1-q^2}\right]. \tag{275}$$

To simplify the prefactor, define

$$b = \beta\hbar\omega_0, \qquad q = e^{-b}. \tag{276}$$

Then

$$1 - q^2 = 1 - e^{-2b} = 2e^{-b}\sinh b, \tag{277}$$

so that

$$\frac{q^{1/2}}{\sqrt{1-q^2}} = \frac{1}{\sqrt{2\sinh b}}. \tag{278}$$

Using $\alpha = m\omega_0/\hbar$, the prefactor is therefore

$$\boxed{\sqrt{\frac{m\omega_0}{2\pi\hbar\sinh(\beta\hbar\omega_0)}}.} \tag{279}$$

For the exponent,

$$-\frac{u^2+v^2}{2} + \frac{2quv - q^2(u^2+v^2)}{1-q^2} \tag{280}$$

$$= -\frac{(1+q^2)(u^2+v^2) - 4quv}{2(1-q^2)}. \tag{281}$$

The identities

$$\frac{1+e^{-2b}}{1-e^{-2b}} = \coth b, \qquad \frac{2e^{-b}}{1-e^{-2b}} = \frac{1}{\sinh b} \tag{282}$$

give

$$-\frac{(u^2+v^2)\cosh b - 2uv}{2\sinh b}. \tag{283}$$

Since

$$u^2 = \frac{m\omega_0}{\hbar}x^2, \qquad v^2 = \frac{m\omega_0}{\hbar}x'^2, \qquad uv = \frac{m\omega_0}{\hbar}xx', \tag{284}$$

the exponent becomes

$$\boxed{-\frac{m\omega_0}{2\hbar\sinh(\beta\hbar\omega_0)}\left[(x^2+x'^2)\cosh(\beta\hbar\omega_0) - 2xx'\right].} \tag{285}$$

B.6 Exact thermal kernel

Combining Eqs. (279) and (285) gives

$$\boxed{\left\langle x\middle|e^{-\beta\hat{H}}\middle|x'\right\rangle = \sqrt{\frac{m\omega_0}{2\pi\hbar\sinh(\beta\hbar\omega_0)}} \times \exp\left\{-\frac{m\omega_0}{2\hbar\sinh(\beta\hbar\omega_0)}\left[(x^2+x'^2)\cosh(\beta\hbar\omega_0)-2xx'\right]\right\}.} \tag{286}$$

This is the exact imaginary-time propagator, equivalently the *unnormalized* thermal density-matrix kernel, of the quantum harmonic oscillator. The normalized coordinate-space thermal density matrix is

$$\rho_{\rm th}(x,x';\beta) = \frac{1}{Z(\beta)}\left\langle x\middle|e^{-\beta\hat{H}}\middle|x'\right\rangle. \tag{287}$$

B.7 Exact propagator

Analytically continuing the imaginary-time kernel back to real time, $\beta\hbar \to \mathrm{i}t$, gives the exact real-time propagator of the harmonic oscillator:

$$\left\langle x\middle|e^{-\frac{i}{\hbar}\hat{H}t}\middle|x'\right\rangle = \sqrt{\frac{m\omega_0}{2\pi i\hbar\sin(\omega_0 t)}} \times \exp\left\{\frac{im\omega_0}{2\hbar\sin(\omega_0 t)}\left[(x^2+x'^2)\cos(\omega_0 t)-2xx'\right]\right\}. \tag{288}$$

This is the unshifted harmonic-oscillator kernel used in Eq. (41) after translating both endpoint coordinates by $d_{\alpha k}$.

B.8 Consistency checks

B.8.1 Short-imaginary-time limit

For

$$\beta\hbar\omega_0 \ll 1, \tag{289}$$

we have

$$\sinh(\beta\hbar\omega_0) \simeq \beta\hbar\omega_0, \qquad \cosh(\beta\hbar\omega_0) \simeq 1. \tag{290}$$

Keeping the leading singular contribution as $\beta \to 0$ gives

$$\rho(x,x';\beta) \simeq \sqrt{\frac{m}{2\pi\beta\hbar^2}}\exp\left[-\frac{m(x-x')^2}{2\beta\hbar^2}\right]. \tag{291}$$

This is the free-particle imaginary-time propagator. The physical reason is that over an infinitesimal imaginary-time interval the kinetic contribution, which scales as $(x-x')^2/\beta$, controls the leading short-time behavior; the potential enters at higher order in β.

B.8.2 Low-temperature limit

As $\beta \to \infty$, the spectral sum is dominated by the ground state:

$$\rho(x,x';\beta) \simeq e^{-\beta E_0}\psi_0(x)\psi_0(x'). \tag{292}$$

Since

$$E_0 = \frac{1}{2}\hbar\omega_0 \tag{293}$$

and

$$\psi_0(x) = \left(\frac{m\omega_0}{\pi\hbar}\right)^{1/4} \exp\left[-\frac{m\omega_0 x^2}{2\hbar}\right], \tag{294}$$

we obtain

$$\rho(x, x'; \beta) \simeq e^{-\beta\hbar\omega_0/2} \left(\frac{m\omega_0}{\pi\hbar}\right)^{1/2} \exp\left[-\frac{m\omega_0}{2\hbar}(x^2 + x'^2)\right], \tag{295}$$

which is exactly the expected ground-state contribution.

B.9 Partition function from the diagonal kernel

Taking the trace gives the canonical partition function,

$$Z(\beta) = \mathrm{Tr}(e^{-\beta\hat{H}}) = \int_{-\infty}^{\infty} dx\, \rho(x, x; \beta). \tag{296}$$

Setting $x' = x$ in Eq. (286),

$$\rho(x, x; \beta) = \sqrt{\frac{m\omega_0}{2\pi\hbar \sinh(\beta\hbar\omega_0)}} \exp\left[-\frac{m\omega_0}{\hbar} x^2 \tanh\left(\frac{\beta\hbar\omega_0}{2}\right)\right]. \tag{297}$$

The Gaussian integral then yields

$$\boxed{Z(\beta) = \frac{1}{2\sinh(\beta\hbar\omega_0/2)}.} \tag{298}$$

This agrees with the direct energy-level sum,

$$Z(\beta) = \sum_{n=0}^{\infty} e^{-\beta\hbar\omega_0(n+1/2)} \tag{299}$$

$$= \frac{e^{-\beta\hbar\omega_0/2}}{1 - e^{-\beta\hbar\omega_0}} \tag{300}$$

$$= \frac{1}{2\sinh(\beta\hbar\omega_0/2)}. \tag{301}$$

B.10 Normalized kernel and extension to the full harmonic bath

Dividing Eq. (286) by Eq. (298) gives the normalized one-mode density matrix. With $b = \beta\hbar\omega_0$ and the identities

$$\frac{(x^2 + x'^2)\cosh b - 2xx'}{\sinh b} = \frac{1}{2}(x - x')^2 \coth\left(\frac{b}{2}\right) + \frac{1}{2}(x + x')^2 \tanh\left(\frac{b}{2}\right), \tag{302}$$

and

$$2\sinh\left(\frac{b}{2}\right)\sqrt{\frac{1}{2\sinh b}} = \sqrt{\tanh\left(\frac{b}{2}\right)}, \tag{303}$$

one obtains

$$\rho_{\rm th}(x,x';\beta) = \sqrt{\frac{m\omega_0}{\pi\hbar}\tanh\left(\frac{b}{2}\right)} \times \exp\left\{-\frac{m\omega_0}{4\hbar}\left[(x-x')^2\coth\left(\frac{b}{2}\right)+(x+x')^2\tanh\left(\frac{b}{2}\right)\right]\right\}. \tag{304}$$

This is precisely Eq. (45) after the replacements $m \to m_\alpha$, $\omega_0 \to \omega_\alpha$, $x \to x^+_{\alpha,0}$, and $x' \to x^-_{\alpha,0}$.

For the free harmonic bath

$$\hat{H}_B = \sum_\alpha \hat{H}_{B,\alpha}, \qquad [\hat{H}_{B,\alpha}, \hat{H}_{B,\gamma}] = 0, \tag{305}$$

the Boltzmann operator, trace, and partition function factorize:

$$e^{-\beta\hat{H}_B} = \bigotimes_\alpha e^{-\beta\hat{H}_{B,\alpha}}. \tag{306}$$

$$\boxed{\begin{aligned} Z_B &= \mathrm{Tr}_B\, e^{-\beta\hat{H}_B} = \prod_\alpha Z_\alpha \\ &= \prod_\alpha \frac{1}{2\sinh(\beta\hbar\omega_\alpha/2)}. \end{aligned}} \tag{307}$$

Consequently, the normalized initial bath state used in Eq. (15) is

$$\boxed{\rho_B^{\rm eq}(x^+,x^-) = \prod_\alpha \rho_{\beta,\alpha}(x^+_\alpha, x^-_\alpha),} \tag{308}$$

where each factor is given by Eq. (304). The factorization is over bath modes; it does not imply a classical mixture in the coordinate basis, because each one-mode kernel is generally off-diagonal in x^+_α and x^-_α.

> **Key idea. Result returned to the main text.** The normalized thermal kernel Eq. (304) supplies the initial oscillator density matrix, and the real-time kernel Eq. (288) supplies each conditional harmonic propagator used in the exact Gaussian bath trace.

C Derivation of the thermal bath correlation function

> **Checkpoint. Why this appendix is here.** The main text reorganizes the Gaussian bath result in terms of the stationary thermal correlation function $C(t)$. This appendix derives $C(t)$ from a single oscillator, sums over independent bath modes, and converts the mode sum to the spectral-density representation used in Sec. 10.

We now derive the equilibrium correlation function of the independent harmonic bath. This is the quantity that enters the Feynman–Vernon influence functional and whose cell integrals produce the coefficients $\eta_{kk'}$ typically used in QUAPI and TEMPO. The derivation proceeds in the same order as the physics: first evaluate one oscillator, then sum independent modes, and finally replace the discrete mode sum by the spectral density $J(\omega)$.

The coupling operator and its equilibrium correlation function are

$$\hat{B} = \sum_{\alpha} c_\alpha \hat{x}_\alpha, \qquad C(t) = \langle \hat{B}(t)\hat{B}(0)\rangle_\beta, \tag{309}$$

where

$$\langle \hat{O}\rangle_\beta = \frac{\mathrm{Tr}_B(e^{-\beta \hat{H}_B}\hat{O})}{Z_B}, \qquad Z_B = \mathrm{Tr}_B\, e^{-\beta \hat{H}_B}, \qquad \beta = (k_B T)^{-1}. \tag{310}$$

Because the thermal state factorizes over modes and $\langle \hat{x}_\alpha\rangle_\beta = 0$, cross-mode terms vanish:

$$\langle \hat{x}_\alpha(t)\hat{x}_\gamma(0)\rangle_\beta = 0 \quad (\alpha \neq \gamma). \tag{311}$$

It is therefore sufficient to evaluate one oscillator.

C.1 Single-oscillator correlation function

For mode α,

$$\hat{x}_\alpha(t) = \sqrt{\frac{\hbar}{2m_\alpha\omega_\alpha}}\left(\hat{a}_\alpha e^{-\mathrm{i}\omega_\alpha t} + \hat{a}_\alpha^\dagger e^{+\mathrm{i}\omega_\alpha t}\right). \tag{312}$$

Consequently,

$$\begin{aligned}\langle \hat{x}_\alpha(t)\hat{x}_\alpha(0)\rangle_\beta = \frac{\hbar}{2m_\alpha\omega_\alpha}\Big[&e^{-\mathrm{i}\omega_\alpha t}\langle \hat{a}_\alpha\hat{a}_\alpha\rangle_\beta + e^{-\mathrm{i}\omega_\alpha t}\langle \hat{a}_\alpha\hat{a}_\alpha^\dagger\rangle_\beta \\ &+ e^{+\mathrm{i}\omega_\alpha t}\langle \hat{a}_\alpha^\dagger\hat{a}_\alpha\rangle_\beta + e^{+\mathrm{i}\omega_\alpha t}\langle \hat{a}_\alpha^\dagger\hat{a}_\alpha^\dagger\rangle_\beta\Big].\end{aligned} \tag{313}$$

In a number-diagonal thermal state,

$$\langle \hat{a}_\alpha\hat{a}_\alpha\rangle_\beta = \langle \hat{a}_\alpha^\dagger\hat{a}_\alpha^\dagger\rangle_\beta = 0, \tag{314}$$

whereas

$$\langle \hat{a}_\alpha^\dagger\hat{a}_\alpha\rangle_\beta = \bar{n}_\alpha, \qquad \langle \hat{a}_\alpha\hat{a}_\alpha^\dagger\rangle_\beta = \bar{n}_\alpha + 1, \tag{315}$$

with the Bose–Einstein occupation

$$\bar{n}_\alpha = \frac{1}{e^{\beta\hbar\omega_\alpha} - 1}. \tag{316}$$

Thus

$$\langle \hat{x}_\alpha(t)\hat{x}_\alpha(0)\rangle_\beta = \frac{\hbar}{2m_\alpha\omega_\alpha}\left[(\bar{n}_\alpha + 1)e^{-\mathrm{i}\omega_\alpha t} + \bar{n}_\alpha e^{+\mathrm{i}\omega_\alpha t}\right]. \tag{317}$$

Using

$$2\bar{n}_\alpha + 1 = \coth\left(\frac{\beta\hbar\omega_\alpha}{2}\right), \tag{318}$$

we obtain

$$\boxed{\langle \hat{x}_\alpha(t)\hat{x}_\alpha(0)\rangle_\beta = \frac{\hbar}{2m_\alpha\omega_\alpha}\left[\coth\left(\frac{\beta\hbar\omega_\alpha}{2}\right)\cos(\omega_\alpha t) - \mathrm{i}\sin(\omega_\alpha t)\right].} \tag{319}$$

The contribution of this mode to the force correlation is therefore

$$\boxed{C_\alpha(t) = \frac{\hbar c_\alpha^2}{2m_\alpha\omega_\alpha}\left[\coth\left(\frac{\beta\hbar\omega_\alpha}{2}\right)\cos(\omega_\alpha t) - \mathrm{i}\sin(\omega_\alpha t)\right].} \tag{320}$$

C.2 From discrete modes to the spectral density

Summing the independent contributions gives $C(t) = \sum_\alpha C_\alpha(t)$. Introduce the convention used in Eq. (57),

$$J(\omega) = \frac{\pi}{2} \sum_\alpha \frac{c_\alpha^2}{m_\alpha \omega_\alpha} \delta(\omega - \omega_\alpha). \tag{321}$$

For any test function f,

$$\frac{1}{\pi} \int_0^\infty \mathrm{d}\omega \, J(\omega) f(\omega) = \frac{1}{2} \sum_\alpha \frac{c_\alpha^2}{m_\alpha \omega_\alpha} f(\omega_\alpha). \tag{322}$$

Applying Eq. (322) to the two trigonometric terms yields

$$\boxed{C(t) = \frac{\hbar}{\pi} \int_0^\infty \mathrm{d}\omega \, J(\omega) \left[\coth\left(\frac{\beta\hbar\omega}{2}\right) \cos(\omega t) - \mathrm{i} \sin(\omega t) \right].} \tag{323}$$

Comparison with Eq. (58) proves $C(t) = \hbar\alpha(t)$ for the spectral-density convention adopted in this tutorial.

C.3 Fluctuation and response components

Write $C(t) = C_R(t) + \mathrm{i}C_I(t)$. Equation (323) gives

$$\boxed{C_R(t) = \frac{\hbar}{\pi} \int_0^\infty \mathrm{d}\omega \, J(\omega) \coth\left(\frac{\beta\hbar\omega}{2}\right) \cos(\omega t),} \tag{324}$$

$$\boxed{C_I(t) = -\frac{\hbar}{\pi} \int_0^\infty \mathrm{d}\omega \, J(\omega) \sin(\omega t).} \tag{325}$$

Equivalently,

$$C_R(t) = \frac{1}{2} \langle \{\hat{B}(t), \hat{B}(0)\} \rangle_\beta, \qquad \mathrm{i}C_I(t) = \frac{1}{2} \langle [\hat{B}(t), \hat{B}(0)] \rangle_\beta. \tag{326}$$

The real, even part is the symmetrized fluctuation correlation and contains both thermal and zero-point fluctuations. The imaginary, odd part is temperature independent for this harmonic bath and determines the retarded susceptibility

$$\chi_R(t) = \frac{\mathrm{i}}{\hbar} \theta(t) \langle [\hat{B}(t), \hat{B}(0)] \rangle_\beta = -\frac{2}{\hbar} \theta(t) C_I(t). \tag{327}$$

The Hermiticity and stationarity check

$$C(-t) = C^*(t) \tag{328}$$

is immediate: C_R is even and C_I is odd. The discretized cell integrals of $C(t)/\hbar = \alpha(t)$ are precisely the coefficients introduced in Eqs. (63) and (64).

> **Key idea. Result returned to the main text.** The spectral representation Eq. (323) is the bath correlation function used to define the cell-integrated coefficients $\eta_{kk'}$ in Part II and the stationary lag coefficients η_j in the numerical example.

D Complete notebook implementation and code map

> **Checkpoint. Why this appendix is here.** The main text explains the algorithms mathematically. This appendix gives the one-to-one map from those operations to the notebook functions, array shapes, broadcasting conventions, and convergence-sweep machinery. The complete executable listing is separated into Appendix G so that code does not interrupt the implementation guide.

This appendix maps the numerical discussion in Sec. 15 onto the actual routines in `Qubit_Ohmic_Persistent.ipynb`. The bundle accompanying this tutorial contains both the original notebook and a plain-Python extraction of all executable cells. The complete executable source is preserved separately in Appendix G; the present appendix is a guided map of the same code.

> **Implementation note.** When reading the code, track the axis meaning rather than only the array size. In the dense ADT the leftmost axis is always the newest path-pair value, while in the persistent MPS the newest temporal site is the left boundary and the oldest retained site is the right boundary. This convention explains both the direction of the lag factors and why expired memory is removed at the right edge.

D.1 Program structure

The notebook is intentionally cumulative. Its executable blocks are organized as follows:

1. define the spin–boson and numerical parameters;
2. evaluate $\alpha_B(t)$ and the finite-cell influence coefficients η_j;
3. construct the dense path-pair system kernel and lag tables I_j;
4. propagate the dense ADT and the reconstruct-after-TT–SVD baseline;
5. plot the reduced dynamics, compression diagnostics, and relative-cutoff sweep;
6. define a second set of cached bath/influence routines for the persistent MPS;
7. propagate the ADT directly as MPS cores, with no global dense reconstruction;
8. perform the K, Δt, and $\chi_{\max}$ convergence sweeps.

This order matters because later cells reuse the physical parameters and the stable `coth_stable` routine defined earlier.

D.2 Baseline bath and dense-ADT routines

The function `bath_alpha` evaluates Eq. (87) by trapezoidal frequency integration. The low-argument thermal factor is stabilized through

```
small = np.abs(x) < 1e-5
out[small] = 1/x[small] + x[small]/3
out[~small] = 1/np.tanh(x[~small])
```

so that numerical evaluation does not rely on subtracting nearly equal exponentials near $x = 0$.

The routine `influence_coefficients` then evaluates the two-cell integrals directly. For $j = 0$, the notebook multiplies the square grid by

```
mask = np.tril(np.ones_like(vals, dtype=float), k=-1)
mask += 0.5*np.eye(n_cell)
```

before the two trapezoidal integrations. The strict lower triangle enforces time ordering, while the half-weight diagonal is the trapezoidal representation of the boundary of the causal triangle.

The four-state pair ordering is encoded by

```
sval = np.array([+1.0, -1.0])
sp = np.repeat(sval, 2)
sm = np.tile(sval, 2)
```

which yields $(+,+),(+,-),(-,+),(-,-)$. Every operation that flattens or reshapes a 2×2 reduced density matrix must use this same ordering.

D.3 Dense QUAPI update

For an old tensor with axes $(a_k, a_{k-1}, \ldots)$, the code creates the new leading path pair by broadcasting the 4×4 system kernel,

```
B = A[None, ...] * Ksys.reshape(
    (4, 4) + (1,)*(A.ndim-1)
)
```

then multiplies the self factor and each retained lag table. If the new tensor exceeds $K+1$ temporal legs, the oldest axis is summed:

```
if B.ndim > memory_depth + 1:
    B = B.sum(axis=-1)
```

This is the explicit array form of the moving QUAPI memory window.

D.4 Reconstruct-after-TT–SVD baseline

The routine `tt_svd_compress` performs a conventional left-to-right tensor-train factorization. At each cut it reshapes the remaining work tensor, computes an economy-size SVD, and keeps

```
threshold = rel_cutoff*s[0] if s.size else 0.0
keep = int(np.count_nonzero(s > threshold))
keep = max(1, keep)
if max_bond is not None:
    keep = min(keep, max_bond)
```

The retained left singular vectors become the current core and $\Sigma V^\dagger$ is passed to the next cut. The routine then contracts all cores back together. This last operation is the defining distinction between the pedagogical baseline and the persistent algorithm below.

The variable named `rho_exact` in the notebook should therefore be interpreted carefully: it is the dense reference *for the chosen K, Δt, and quadrature*, not an exact solution of the continuum open-system problem.

D.5 Persistent-MPS routines

The persistent section begins by defining cached versions of the bath and influence quadratures. The optimized cell integration notices that the matrix

$$j\Delta t + x_m - x_n \tag{329}$$

contains only $2n_{\text{cell}} - 1$ distinct time differences. The routine evaluates α_B only at those unique differences and reconstructs the matrix by indexing. This optimization changes how often the correlation function is evaluated, not the definition of η_j.

D.5.1 Direct reduced-state contraction

`reduced_from_mps_cores` starts with a one-component right environment. Each history core is summed over its physical index and contracted into that environment. The newest core is left un-summed, producing the four path-pair components of ρ_S. This is why readout itself never requires dense ADT reconstruction.

D.5.2 Right canonicalization

For core `cores[i]` with shape (χ_L, d, χ_R), `right_canonicalize_mps` reshapes it to $(\chi_L, d\chi_R)$ and applies QR to its transpose:

```
Q, R = np.linalg.qr(M.T, mode='reduced')
cores[i] = Q.T.reshape(chi_new, dphys, chi_r)
cores[i-1] = np.tensordot(cores[i-1], R.T, axes=(2, 0))
```

The sweep proceeds from the oldest site toward the newest, transferring the non-isometric factor leftward.

D.5.3 Local SVD compression

`compress_persistent_mps` then sweeps in the opposite direction. The current core is reshaped to $(\chi_L d, \chi_R)$, decomposed by SVD, and truncated with exactly the relative criterion in Eq. (112). The retained U is reshaped back into the current core, while $\Sigma V^\dagger$ is contracted into the next core. No matrix in this sweep has the dimensions of the global ADT.

D.5.4 Adding the newest slice

`add_slice_persistent_mps` implements Eq. (105). The new core has shape $(1, 4, 4)$ and places $I_0(p)$ on the diagonal connecting the physical index p to a four-dimensional virtual carrier. At each old core, this carrier is propagated block-diagonally. For lag one the multiplier is `Ksys_local*Ilag_local[1]`; for later retained lags it is `Ilag_local[lag]`. At the final core the right bond remains one because the carrier need not be passed farther.

The intermediate virtual dimensions can be larger before compression because each old bond is replicated in four p blocks. This temporary growth is local. The code never calls `reshape` on, or allocates, an array with the full shape $(4, 4, \ldots, 4)$ for all retained temporal sites.

D.5.5 Removing the oldest history site

If there are more than $K + 1$ MPS sites after insertion, the code executes

```
boundary_vec = cores[-1].sum(axis=1)[:, 0]
new_last = np.tensordot(
    cores[-2], boundary_vec, axes=(2, 0)
)[:, :, None]
```

which sums the oldest physical path-pair index and absorbs the result into the preceding core. This is algebraically the same operation as summing the last axis of the dense ADT.

D.6 Persistent propagation loop and diagnostics

The main persistent routine is `persistent_mps_quapi`. Its loop has exactly four logical stages:

1. `add_slice_persistent_mps`: insert the new system step and all influence factors within memory;
2. `discard_oldest_mps_site`: remove the expired history site if necessary;
3. `compress_persistent_mps`: canonicalize and truncate local bonds;
4. `reduced_from_mps_cores`: contract the history to record $\rho_S(t)$.

The diagnostic

```
sum(core.size for core in cores)
```

is the actual number of complex MPS coefficients stored at that step. The code also records the largest retained bond and the discarded singular-value norm.

D.7 Convergence-sweep conventions

The function `cached_persistent_run` reuses calculations with identical $(\Delta t, K, t_{\text{final}}, \varepsilon_{\text{MPS}}, \chi_{\max})$. The convergence cells use `n_cell=25` and `n_omega=1800` for speed. When trajectories use different Δt, the reference reduced-state components are linearly interpolated onto the target time grid before Eq. (114) is evaluated.

Two details are especially important when interpreting the plots:

- the memory sweep changes $K\Delta t$ while holding Δt fixed;
- the time-step sweep deliberately changes K so that $K\Delta t = 0.60$ remains fixed.

Thus the latter isolates time discretization much more cleanly than simply reducing Δt at fixed K.

D.8 Array-shape guide

Object	Shape	Meaning
`U`	$(2, 2)$	one-step Hilbert-space system propagator
`Ksys`	$(4, 4)$	one-step forward/backward path-pair propagator
`eta`	$(K+1,)$	lag coefficients $\eta_0, \ldots, \eta_K$
`Ilag[j]`	$(4, 4)$	pair factor $I_j(p, a)$
Dense `A`	$(4,)^{\otimes L}$	explicit ADT, $L \leq K+1$
Persistent core	$(\chi_L, 4, \chi_R)$	one temporal MPS site
`rho_pmps`	$(N+1, 2, 2)$	persistent-MPS reduced-state trajectory
`bond_history`	$(N+1,)$	largest retained bond at each step
`storage_history`	$(N+1,)$	stored MPS coefficients at each step

E Numerical values underlying the convergence figures

Checkpoint. Why this appendix is here. The main results section emphasizes the convergence question, experimental design, and trend in each sweep. The complete notebook values are collected here so that no numerical information is lost while the main narrative remains readable.

E.1 Reconstruct-after-TT–SVD cutoff sweep

E.2 Persistent-MPS memory-depth sweep

$\varepsilon_{\mathrm{MPS}}$	maximum bond	$\max \lvert\Delta\rho\rvert$	runtime (s)
10^{-3}	4	1.312×10^{-2}	0.728
10^{-5}	12	1.639×10^{-4}	0.873
10^{-7}	22	8.073×10^{-7}	1.190
0	64	1.910×10^{-14}	1.580

Table 1: Complete relative-cutoff sweep for the pedagogical reconstruct-after-TT–SVD implementation. The zero-cutoff calculation also removes the bond cap. Runtimes are notebook-run timings and are not hardware-independent benchmarks.

K	$K\Delta t$	maximum bond	ϵ_ρ
2	0.20	4	4.434×10^{-2}
4	0.40	16	1.507×10^{-2}
6	0.60	48	5.126×10^{-3}
8	0.80	48	1.277×10^{-3}
10	1.00	48	0 (reference)

Table 2: Persistent-MPS convergence with memory depth at fixed $\Delta t = 0.10$. The $K = 10$ trajectory is the reference within this sweep.

E.3 Persistent-MPS time-step sweep at fixed physical memory

Δt	K	$K\Delta t$	maximum bond	ϵ_ρ
0.200	3	0.60	12	9.315×10^{-2}
0.100	6	0.60	48	3.645×10^{-2}
0.050	12	0.60	48	0 (reference)

Table 3: Persistent-MPS time-step convergence with $K\Delta t = 0.60$ held fixed. The $\Delta t = 0.05$, $K = 12$ trajectory is the reference within this sweep.

E.4 Persistent-MPS bond-dimension sweep

Key idea. These tables should be read together with the convergence figures in Sec. 15.7. A zero in a reference row is a zero self-difference, not a claim that all other numerical approximations have vanished.

F Complete executable source

Checkpoint. Why this appendix is here. Appendix D explains the program structure and individual routines. This final appendix preserves the full executable source in notebook order so that every figure and numerical value in the tutorial can be reproduced from one listing.

The following listing contains all executable notebook cells in their original order. The Markdown cells have been converted only to Python comments so that the file remains a standalone, readable script. Running it reproduces all eight numerical figures included in Sec. 15.

$\chi_{\max}$	used bond	peak MPS coefficients	ϵ_ρ
4	4	352	1.097×10^{-3}
8	8	1056	1.310×10^{-4}
16	16	3296	5.626×10^{-7}
32	32	8160	2.352×10^{-9}
48	48	12192	1.332×10^{-12}
64	60	14688	0 (reference)

Table 4: Persistent-MPS convergence with maximum bond dimension at fixed $\Delta t = 0.10$, $K = 6$, and relative cutoff 10^{-13}. The $\chi_{\max} = 64$ run is the reference within this sweep.

```
# Extracted verbatim from the executable cells of Qubit_Ohmic_Persistent.ipynb
# Markdown cells are retained below as comments for orientation.


# ============================================================================
# NOTEBOOK MARKDOWN CELL 0
# ============================================================================
# # Qubit propagation with and without MPS compression
#
# This Colab-ready test implements a finite-memory QUAPI propagation for the spin-boson model and compares:
#
# 1. **Uncompressed augmented density tensor (ADT)** -- the finite-memory reference.
# 2. **MPS-compressed ADT** -- after every time step the temporal tensor is factorized by TT-SVD and singular values are truncated.
#
# We use units $\hbar=k_B=1$,
# $$
# H_S=\frac{\epsilon}{2}\sigma_z+\frac{\Delta}{2}\sigma_x,
# \qquad H_{SB}=\sigma_z\otimes B,
# $$
# and the Ohmic spectral density
# $$
# J(\omega)=2\pi\alpha_{\rm SB}\,\omega e^{-\omega/\omega_c}.
# $$
# With the spectral-density convention in the attached tutorial,
# $$
# \alpha_B(t)=\frac{1}{\pi}\int_0^\infty d\omega\,J(\omega)
# \left[\coth\!\left(\frac{\beta\omega}{2}\right)\cos\omega t-i\sin\omega t\right].
# $$
#
# The causal influence factor added at time slice $k$ is
# $$
# \prod_{j=0}^{\min(K,k)}
# \exp\!\left[-(s_k^+-s_k^-)
# \left(\eta_j s_{k-j}^+-\eta_j^*s_{k-j}^-\right)\right].
# $$
#
# This is a transparent pedagogical implementation, not an optimized production TEMPO package. The compressed propagation reconstructs the small ADT after TT-SVD so that the update remains easy to inspect; nevertheless, its retained tensor-train ranks and truncation are exactly those of an MPS representation.
#
# > **Added in this revision.** The final section implements a genuinely **persistent MPS QUAPI** propagation. It never reconstructs or allocates the global dense augmented density tensor (ADT); only local MPS-core matricizations are used for QR/SVD bond compression. The original dense and reconstruct-after-TT-SVD implementations are retained above as transparent pedagogical baselines.

# ============================================================================
# NOTEBOOK CODE CELL 1
# ============================================================================
import time
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.linalg import expm

np.set_printoptions(precision=5, suppress=True)
plt.rcParams.update({"figure.dpi": 120, "axes.grid": True, "grid.alpha": 0.25})

# ------------------------- physical parameters -------------------------
epsilon = 1.0
Delta = 1.0
alpha_SB = 0.08
omega_c = 5.0
temperature = 0.5
beta = 1.0 / temperature

# ------------------------- numerical parameters ------------------------
dt = 0.10
n_steps = 80
memory_depth = 6       # retained lags j = 0,...,K; ADT has at most K+1 legs
mps_rel_cutoff = 1e-6  # relative discarded singular-value threshold
mps_max_bond = 32

assert memory_depth >= 1
print(f"Maximum dense ADT size: 4^{memory_depth+1} = {4**(memory_depth+1):,} complex numbers")

# ============================================================================
# NOTEBOOK MARKDOWN CELL 2
# ============================================================================
# ## Bath correlation function and influence coefficients
#
# For $j\ge 1$, $\eta_j$ is integrated over two complete time cells separated by $j\Delta t$. For $j=0$, only
    the time-ordered triangle in a single cell is retained. The quadrature below directly implements these
    definitions. For coupling through $\sigma_z$, the usual quadratic counterterm is proportional to $\
    sigma_z^2=I$ and its forward/backward phase cancels, so no separate counterterm factor is required.

# ============================================================================
# NOTEBOOK CODE CELL 3
# ============================================================================
def coth_stable(x):
    x = np.asarray(x)
    out = np.empty_like(x, dtype=float)
    small = np.abs(x) < 1e-5
    out[small] = 1/x[small] + x[small]/3
    out[~small] = 1/np.tanh(x[~small])
    return out


def bath_alpha(t, n_omega=5000, omega_max_factor=12.0):
    '''Bath correlation alpha_B(t)=C(t)/hbar for the stated J(omega).'''
    t = np.atleast_1d(t).astype(float)
    w = np.linspace(1e-7, omega_max_factor*omega_c, n_omega)
    J = 2*np.pi*alpha_SB*w*np.exp(-w/omega_c)
    thermal = coth_stable(0.5*beta*w)
    phase = np.outer(t, w)
    integrand = (J/np.pi)[None, :] * (
        thermal[None, :]*np.cos(phase) - 1j*np.sin(phase)
    )
    ans = np.trapezoid(integrand, w, axis=1)
    return ans[0] if ans.size == 1 else ans


def influence_coefficients(K, dt, n_cell=41):
    '''Uniform-cell eta_j, with the causal triangle used at j=0.'''
    x = np.linspace(-dt/2, dt/2, n_cell)
    eta = np.zeros(K+1, dtype=complex)
    for j in range(K+1):
        tau = j*dt + x[:, None] - x[None, :]
        vals = bath_alpha(tau.ravel()).reshape(tau.shape)
        if j == 0:
```

```
            # t1 >= t2. Half-weight on the diagonal gives a trapezoidal
            # approximation to the causal triangle.
            mask = np.tril(np.ones_like(vals, dtype=float), k=-1)
            mask += 0.5*np.eye(n_cell)
            vals = vals*mask
        eta[j] = np.trapezoid(np.trapezoid(vals, x, axis=1), x, axis=0)
    return eta


eta = influence_coefficients(memory_depth, dt)
print("lag j     Re(eta_j)       Im(eta_j)       |eta_j|")
for j, z in enumerate(eta):
    print(f"{j:3d}    {z.real: .7e}  {z.imag: .7e}  {abs(z):.7e}")

tau_plot = np.linspace(0, memory_depth*dt*2, 300)
c_plot = bath_alpha(tau_plot)
fig, ax = plt.subplots(1, 2, figsize=(10, 3.4))
ax[0].plot(tau_plot, c_plot.real, label=r"Re $\alpha_B(t)$")
ax[0].plot(tau_plot, c_plot.imag, label=r"Im $\alpha_B(t)$")
ax[0].axvline(memory_depth*dt, color="k", ls="--", label=r"$K\Delta t$")
ax[0].set(xlabel="time", ylabel="bath correlation", title="Ohmic bath correlation")
ax[0].legend()
ax[1].semilogy(range(memory_depth+1), np.maximum(abs(eta), 1e-18), "o-")
ax[1].set(xlabel="lag $j$", ylabel=r"$|\eta_j|$", title="Integrated influence coefficients")
plt.tight_layout(); plt.show()

# ============================================================================
# NOTEBOOK MARKDOWN CELL 4
# ============================================================================
# ## QUAPI update and MPS/TT-SVD compression
#
# Each temporal leg is the path-pair index $a_k=(s_k^+,s_k^-)$, of dimension four. The first axis is always
    the newest time. After adding a new slice, the oldest leg is summed out once the tensor exceeds $K+1$
    legs.
#
# `tt_svd_compress` converts the ADT to an MPS, truncates Schmidt singular values at every temporal bond, and
    reconstructs the tensor for the next explicit update. Set `mps_rel_cutoff=0` and a sufficiently large
    bond dimension to recover the uncompressed result to roundoff.

# ============================================================================
# NOTEBOOK CODE CELL 5
# ============================================================================
sx = np.array([[0, 1], [1, 0]], complex)
sz = np.array([[1, 0], [0, -1]], complex)
HS = 0.5*epsilon*sz + 0.5*Delta*sx
U = expm(-1j*HS*dt)

# Pair ordering a = 2*s_plus_index + s_minus_index; sigma_z eigenvalues.
sval = np.array([+1.0, -1.0])
sp = np.repeat(sval, 2)
sm = np.tile(sval, 2)

# Ksys[a_new,a_old] maps rho_{old} to U rho U^dagger.
Ksys = np.empty((4, 4), complex)
for anew in range(4):
    ip, im = divmod(anew, 2)
    for aold in range(4):
        jp, jm = divmod(aold, 2)
        Ksys[anew, aold] = U[ip, jp] * U[im, jm].conjugate()


def influence_table(eta):
    tables = []
    for z in eta:
        # first index: newest pair; second: pair at the requested lag
        tables.append(np.exp(-(sp[:, None]-sm[:, None]) *
                             (z*sp[None, :] - z.conjugate()*sm[None, :])))
    return tables
```

```
175
176
177 Ilag = influence_table(eta)
178
179
180 def reduced_from_adt(A):
181     v = A.sum(axis=tuple(range(1, A.ndim))) if A.ndim > 1 else A.copy()
182     return v.reshape(2, 2)
183
184
185 def physical_cleanup(rho):
186     '''Only remove tiny numerical trace/Hermiticity drift; do not enforce positivity.'''
187     rho = 0.5*(rho + rho.conj().T)
188     return rho/np.trace(rho)
189
190
191 def tt_svd_compress(A, rel_cutoff=1e-8, max_bond=None):
192     '''TT-SVD/MPS compression. Returns reconstructed tensor, ranks, discarded weight.'''
193     dims = A.shape
194     if len(dims) == 1:
195         return A.copy(), [], 0.0
196     cores, ranks = [], []
197     discarded_sq = 0.0
198     rleft = 1
199     work = A.reshape(dims)
200     for site in range(len(dims)-1):
201         M = work.reshape(rleft*dims[site], -1)
202         u, s, vh = np.linalg.svd(M, full_matrices=False)
203         threshold = rel_cutoff*s[0] if s.size else 0.0
204         keep = int(np.count_nonzero(s > threshold))
205         keep = max(1, keep)
206         if max_bond is not None:
207             keep = min(keep, max_bond)
208         discarded_sq += float(np.sum(s[keep:]**2))
209         cores.append(u[:, :keep].reshape(rleft, dims[site], keep))
210         ranks.append(keep)
211         work = s[:keep, None]*vh[:keep, :]
212         rleft = keep
213     cores.append(work.reshape(rleft, dims[-1], 1))
214
215     rec = cores[0]
216     for core in cores[1:]:
217         rec = np.tensordot(rec, core, axes=([-1], [0]))
218     rec = np.squeeze(rec, axis=(0, -1))
219     return rec, ranks, np.sqrt(discarded_sq)
220
221
222 def add_time_slice(A):
223     '''Add a newest path-pair leg and all influence factors within memory.'''
224     # B[new, old0, old1,...] = Ksys[new,old0] A[old0,old1,...]
225     B = A[None, ...] * Ksys.reshape((4, 4) + (1,)*(A.ndim-1))
226     # self interaction j=0
227     shape_new = (4,) + (1,)*A.ndim
228     B *= np.diag(Ilag[0]).reshape(shape_new)
229     # interactions with retained older slices
230     for j in range(1, min(memory_depth, A.ndim)+1):
231         shape = [1]*B.ndim
232         shape[0] = 4
233         shape[j] = 4
234         B *= Ilag[j].reshape(shape)
235     # Keep newest plus K older indices: K+1 temporal legs total.
236     if B.ndim > memory_depth + 1:
237         B = B.sum(axis=-1)
238     return B
239
240
241 def propagate(compress=False, rel_cutoff=1e-8, max_bond=None):
242     # Initial state |+x><+x|. Include the diagonal influence factor at t=0.
```

```
    rho0 = 0.5*np.array([[1, 1], [1, 1]], complex)
    A = rho0.reshape(4)*np.diag(Ilag[0])
    rhos = [physical_cleanup(reduced_from_adt(A))]
    max_ranks, discarded = [1], [0.0]
    tic = time.perf_counter()
    for _ in range(n_steps):
        A = add_time_slice(A)
        if compress:
            A, ranks, disc = tt_svd_compress(A, rel_cutoff, max_bond)
            max_ranks.append(max(ranks, default=1))
            discarded.append(disc)
        else:
            max_ranks.append(np.nan)
            discarded.append(0.0)
        rhos.append(physical_cleanup(reduced_from_adt(A)))
    return np.asarray(rhos), np.asarray(max_ranks), np.asarray(discarded), time.perf_counter()-tic


rho_exact, _, _, time_exact = propagate(compress=False)
rho_mps, ranks, discarded, time_mps = propagate(
    compress=True, rel_cutoff=mps_rel_cutoff, max_bond=mps_max_bond
)
print(f"Uncompressed runtime: {time_exact:.3f} s")
print(f"Compressed runtime:   {time_mps:.3f} s (pedagogical reconstruct-after-SVD version)")
print(f"Largest retained MPS bond: {int(np.max(ranks))}")

# ============================================================================
# NOTEBOOK MARKDOWN CELL 6
# ============================================================================
# ## Compare reduced-density-matrix elements
#
# The top row overlays the reference and compressed results. The lower panels show elementwise absolute errors,
     MPS bond dimensions, discarded TT-SVD norm, trace drift, and the smallest eigenvalue. A converged
    calculation should also be repeated for larger $K$, smaller $\Delta t$, denser bath quadrature, and
    tighter MPS cutoffs.

# ============================================================================
# NOTEBOOK CODE CELL 7
# ============================================================================
times = dt*np.arange(n_steps+1)
labels = [r"$\rho_{00}$", r"$\rho_{11}$", r"Re $\rho_{01}$", r"Im $\rho_{01}$"]
series_e = [rho_exact[:,0,0].real, rho_exact[:,1,1].real,
            rho_exact[:,0,1].real, rho_exact[:,0,1].imag]
series_m = [rho_mps[:,0,0].real, rho_mps[:,1,1].real,
            rho_mps[:,0,1].real, rho_mps[:,0,1].imag]

fig, axes = plt.subplots(2, 2, figsize=(11, 7), sharex=True)
for ax, ye, ym, lab in zip(axes.ravel(), series_e, series_m, labels):
    ax.plot(times, ye, lw=2.2, label="uncompressed ADT")
    ax.plot(times, ym, "--", lw=1.6, label="MPS compressed")
    ax.set(ylabel=lab)
    ax.legend(fontsize=8)
for ax in axes[-1]: ax.set_xlabel("time")
fig.suptitle(f"Spin-boson reduced dynamics: K={memory_depth}, cutoff={mps_rel_cutoff:g}")
plt.tight_layout(); plt.show()

element_error = np.max(np.abs(rho_mps-rho_exact), axis=(1,2))
trace_error = np.abs(np.trace(rho_mps, axis1=1, axis2=2)-1)
min_eval = np.array([np.linalg.eigvalsh(r).min() for r in rho_mps])

fig, ax = plt.subplots(1, 3, figsize=(13, 3.5))
ax[0].semilogy(times, np.maximum(element_error, 1e-17))
ax[0].set(xlabel="time", ylabel=r"$\max_{ij}|\Delta\rho_{ij}|$", title="Compression error")
ax[1].step(times, ranks, where="mid", label="max MPS bond")
ax[1].set(xlabel="time", ylabel="bond dimension", title="Retained MPS rank")
ax2 = ax[1].twinx()
ax2.semilogy(times, np.maximum(discarded, 1e-18), color="C1", alpha=.7, label="discarded norm")
ax2.set_ylabel("discarded norm", color="C1")
```

```
ax[2].semilogy(times, np.maximum(trace_error, 1e-18), label="trace error")
ax[2].plot(times, min_eval, label="smallest eigenvalue")
ax[2].axhline(0, color="k", lw=.8)
ax[2].set(xlabel="time", title="Numerical checks")
ax[2].legend(fontsize=8)
plt.tight_layout(); plt.show()

print(f"Maximum elementwise compression error: {element_error.max():.3e}")
print(f"Maximum trace error after cleanup:      {trace_error.max():.3e}")
print(f"Minimum eigenvalue (compressed rho):    {min_eval.min():.3e}")

# ============================================================================
# NOTEBOOK MARKDOWN CELL 8
# ============================================================================
# ## Compression-cutoff convergence test
#
# This final cell repeats only the compressed calculation. Tightening the cutoff should make the MPS result approach the uncompressed finite-memory reference while generally increasing the retained bond dimension.

# ============================================================================
# NOTEBOOK CODE CELL 9
# ============================================================================
cutoffs = [1e-3, 1e-5, 1e-7, 0.0]
rows = []
for cutoff in cutoffs:
    r, rk, disc, runtime = propagate(
        compress=True, rel_cutoff=cutoff,
        max_bond=None if cutoff == 0 else mps_max_bond
    )
    err = np.max(np.abs(r-rho_exact))
    rows.append((cutoff, int(np.max(rk)), err, runtime))

print(" rel cutoff     max bond     max |Delta rho|     runtime (s)")
for cutoff, bond, err, runtime in rows:
    print(f" {cutoff:9.1e}     {bond:8d}       {err:11.3e}       {runtime:8.3f}")

fig, ax = plt.subplots(figsize=(5.5, 3.5))
x = [max(r[0], 1e-16) for r in rows]
y = [max(r[2], 1e-17) for r in rows]
ax.loglog(x, y, "o-")
ax.set(xlabel="relative MPS cutoff", ylabel=r"max $|\Delta\rho_{ij}|$",
       title="MPS-compression convergence")
plt.tight_layout(); plt.show()

# ============================================================================
# NOTEBOOK MARKDOWN CELL 10
# ============================================================================
# ## Persistent-MPS QUAPI: the ADT is never reconstructed densely
#
# The implementation above is useful pedagogically because the full finite-memory ADT is explicit. However, after TT-SVD it reconstructs that tensor before the next QUAPI update. The following implementation instead keeps the ADT **persistently as an MPS**,
# $$
# A(a_k,a_{k-1},\ldots)=
# \sum_{\{\alpha\}}
# G^{[0]}_{1,a_k,\alpha_1}
# G^{[1]}_{\alpha_1,a_{k-1},\alpha_2}\cdots
# G^{[L-1]}_{\alpha_{L-1},a_{k-L+1},1},
# $$
# where every physical index is the four-state forward/backward path-pair index
# $a_j=(s_j^+,s_j^-)$.
#
# When a new slice with path-pair index $p=a_{k+1}$ is introduced, its value is threaded through the MPS virtual bonds. This exactly represents
# $$
# B(p,a_k,a_{k-1},\ldots)
# =I_0(p)\,A(a_k,a_{k-1},\ldots)
# \,[K_S(p,a_k)I_1(p,a_k)]
```

```
# \prod_{j=2}^{K} I_j(p,a_{k-j+1}).
# $$
# Before compression this threading multiplies the relevant virtual dimensions by at most the local physical dimension $d=4$. The MPS is then right-canonicalized and truncated by local SVDs. If more than $K+1$ temporal sites are present, the oldest physical index is summed directly at the **right boundary** and absorbed into the neighboring core.
#
# **Important:** no object of size $4^{K+1}$ is allocated or reconstructed by this propagation. QR and SVD do form small dense **local core matricizations**, but never the global dense ADT.

# ============================================================================
# NOTEBOOK CODE CELL 11
# ============================================================================
# ============================================================================
# Persistent-MPS QUAPI: no global dense ADT is ever constructed
# ============================================================================

from functools import lru_cache

_PAIR_SP = np.repeat(np.array([+1.0, -1.0]), 2)
_PAIR_SM = np.tile(np.array([+1.0, -1.0]), 2)

# We cache the bath quadrature because convergence studies reuse (K, dt) pairs.
_bath_grid_cache = {}
_eta_mps_cache = {}


def _bath_alpha_for_mps(t, n_omega=2500, omega_max_factor=12.0):
    """Same Ohmic bath correlation used above, but with a cached frequency grid."""
    key = (int(n_omega), float(omega_max_factor), float(alpha_SB),
           float(omega_c), float(beta))
    if key not in _bath_grid_cache:
        w = np.linspace(1e-7, omega_max_factor*omega_c, n_omega)
        J = 2*np.pi*alpha_SB*w*np.exp(-w/omega_c)
        thermal = coth_stable(0.5*beta*w)
        _bath_grid_cache[key] = (w, J/np.pi, thermal)
    w, pref, thermal = _bath_grid_cache[key]

    t = np.atleast_1d(t).astype(float)
    phase = np.outer(t, w)
    values = np.trapezoid(
        pref[None, :]*(thermal[None, :]*np.cos(phase) - 1j*np.sin(phase)),
        w, axis=1
    )
    return values


def influence_coefficients_mps(K, dt_local, n_cell=31, n_omega=2500):
    """Finite-cell eta_j with the same causal j=0 triangle, cached for sweeps.

    The 2D cell integral is evaluated from only 2*n_cell-1 unique time
    differences, rather than re-evaluating alpha_B on every grid pair.
    """
    key = (int(K), float(dt_local), int(n_cell), int(n_omega))
    if key in _eta_mps_cache:
        return _eta_mps_cache[key].copy()

    x = np.linspace(-dt_local/2, dt_local/2, n_cell)
    h = x[1] - x[0]
    q = np.arange(-(n_cell-1), n_cell)
    diff_index = (np.arange(n_cell)[:, None] - np.arange(n_cell)[None, :]) + (n_cell-1)

    eta_local = np.zeros(K+1, dtype=complex)
    for j in range(K+1):
        tau_unique = j*dt_local + q*h
        alpha_unique = _bath_alpha_for_mps(tau_unique, n_omega=n_omega)
        vals = alpha_unique[diff_index]
        if j == 0:
```

```
            mask = np.tril(np.ones((n_cell, n_cell)), k=-1) + 0.5*np.eye(n_cell)
            vals = vals*mask
        eta_local[j] = np.trapezoid(np.trapezoid(vals, x, axis=1), x, axis=0)

    _eta_mps_cache[key] = eta_local.copy()
    return eta_local


def system_kernel_dt(dt_local):
    """K_S[a_new,a_old] for one symmetric system time slice."""
    sx_local = np.array([[0, 1], [1, 0]], complex)
    sz_local = np.array([[1, 0], [0, -1]], complex)
    H_local = 0.5*epsilon*sz_local + 0.5*Delta*sx_local
    U_local = expm(-1j*H_local*dt_local)

    K_local = np.empty((4, 4), complex)
    for anew in range(4):
        ip, im = divmod(anew, 2)
        for aold in range(4):
            jp, jm = divmod(aold, 2)
            K_local[anew, aold] = U_local[ip, jp]*U_local[im, jm].conjugate()
    return K_local


def influence_tables_mps(eta_local):
    return [
        np.exp(-(_PAIR_SP[:, None]-_PAIR_SM[:, None]) *
               (z*_PAIR_SP[None, :] - z.conjugate()*_PAIR_SM[None, :]))
        for z in eta_local
    ]


def _cleanup_rho(rho):
    rho = 0.5*(rho + rho.conj().T)
    return rho/np.trace(rho)


def reduced_from_mps_cores(cores):
    """Trace/sum all retained history legs directly in the MPS."""
    env = np.ones(1, dtype=complex)
    for core in cores[:0:-1]:
        env = core.sum(axis=1) @ env
    newest = cores[0][0] @ env
    return newest.reshape(2, 2)


def right_canonicalize_mps(cores):
    """Right-canonicalize using QR on local core matricizations only."""
    cores = [c.copy() for c in cores]
    for i in range(len(cores)-1, 0, -1):
        chi_l, dphys, chi_r = cores[i].shape
        M = cores[i].reshape(chi_l, dphys*chi_r)
        Q, R = np.linalg.qr(M.T, mode='reduced')
        chi_new = Q.shape[1]
        cores[i] = Q.T.reshape(chi_new, dphys, chi_r)
        cores[i-1] = np.tensordot(cores[i-1], R.T, axes=(2, 0))
    return cores


def compress_persistent_mps(cores, rel_cutoff=1e-10, max_bond=64):
    """Canonicalize and truncate each MPS bond without global reconstruction."""
    if len(cores) <= 1:
        return [c.copy() for c in cores], [], 0.0

    cores = right_canonicalize_mps(cores)
    ranks = []
    discarded_sq = 0.0

```

```
    for i in range(len(cores)-1):
        chi_l, dphys, chi_r = cores[i].shape
        M = cores[i].reshape(chi_l*dphys, chi_r)
        Uloc, s, Vh = np.linalg.svd(M, full_matrices=False)

        threshold = rel_cutoff*s[0] if s.size else 0.0
        keep = max(1, int(np.count_nonzero(s > threshold)))
        if max_bond is not None:
            keep = min(keep, int(max_bond))

        discarded_sq += float(np.sum(s[keep:]**2))
        cores[i] = Uloc[:, :keep].reshape(chi_l, dphys, keep)
        transfer = s[:keep, None]*Vh[:keep, :]
        cores[i+1] = np.tensordot(transfer, cores[i+1], axes=(1, 0))
        ranks.append(keep)

    # After the left-to-right SVD sweep, all preceding cores are left-isometric.
    # Rescaling the final core changes only an irrelevant global ADT factor and
    # avoids under/overflow in long propagations.
    scale = np.linalg.norm(cores[-1])
    if scale > 0:
        cores[-1] = cores[-1]/scale

    return cores, ranks, np.sqrt(discarded_sq)


def add_slice_persistent_mps(cores, Ksys_local, Ilag_local, K):
    """Add the newest QUAPI time slice directly to a persistent MPS.

    The new physical value p is carried through the virtual bonds, so the
    pairwise influence factors I_j[p,a_old] are inserted without constructing
    the rank-(K+2) dense ADT.
    """
    dphys = 4
    n_old = len(cores)

    # Newest site: right virtual index carries p into the history chain.
    new0 = np.zeros((1, dphys, dphys), dtype=complex)
    new0[0, np.arange(dphys), np.arange(dphys)] = np.diag(Ilag_local[0])
    out = [new0]

    for i, G in enumerate(cores):
        chi_l, d_old, chi_r = G.shape
        lag = i + 1

        if lag == 1:
            F = Ksys_local.copy()
            if K >= 1:
                F *= Ilag_local[1]
        elif lag <= K:
            F = Ilag_local[lag]
        else:
            # This can occur only for the oldest site immediately before it
            # is removed; QUAPI contains no interaction beyond lag K.
            F = np.ones((dphys, d_old), dtype=complex)

        if i == n_old - 1:
            # Right boundary: old chi_r is 1; no need to carry p farther.
            C = np.zeros((dphys*chi_l, d_old, 1), dtype=complex)
            for p in range(dphys):
                C[p*chi_l:(p+1)*chi_l, :, 0] = G[:, :, 0]*F[p, :][None, :]
        else:
            # Block-diagonal propagation of the same p through left/right bonds.
            C = np.zeros((dphys*chi_l, d_old, dphys*chi_r), dtype=complex)
            for p in range(dphys):
                C[p*chi_l:(p+1)*chi_l, :, p*chi_r:(p+1)*chi_r] = \
                    G*F[p, :][None, :, None]
        out.append(C)
```

```
572
573     return out
574
575
576 def discard_oldest_mps_site(cores):
577     """Sum the oldest physical index at the right MPS boundary."""
578     if len(cores) <= 1:
579         return cores
580     boundary_vec = cores[-1].sum(axis=1)[:, 0]
581     new_last = np.tensordot(cores[-2], boundary_vec, axes=(2, 0))[:, :, None]
582     return cores[:-2] + [new_last]
583
584
585 def mps_storage_elements(cores):
586     return int(sum(core.size for core in cores))
587
588
589 def max_mps_bond(cores):
590     if len(cores) <= 1:
591         return 1
592     return int(max(max(c.shape[0], c.shape[2]) for c in cores))
593
594
595 def persistent_mps_quapi(dt_local, K, t_final, rel_cutoff=1e-10,
596                          max_bond=64, n_cell=31, n_omega=2500,
597                          rho0=None):
598     """Finite-memory QUAPI propagation with a persistent MPS ADT.
599
600     At no point is the global dense ADT constructed or reconstructed.
601     """
602     assert K >= 1
603     n_steps_local = int(round(t_final/dt_local))
604     t_final = n_steps_local*dt_local
605
606     eta_local = influence_coefficients_mps(K, dt_local, n_cell, n_omega)
607     Ilag_local = influence_tables_mps(eta_local)
608     Ksys_local = system_kernel_dt(dt_local)
609
610     if rho0 is None:
611         rho0 = 0.5*np.array([[1, 1], [1, 1]], complex)  # |+x><+x|
612
613     A0 = rho0.reshape(4)*np.diag(Ilag_local[0])
614     cores = [A0.reshape(1, 4, 1)]
615
616     rhos = [_cleanup_rho(reduced_from_mps_cores(cores))]
617     bond_history = [1]
618     storage_history = [mps_storage_elements(cores)]
619     discarded_history = [0.0]
620
621     tic = time.perf_counter()
622     for _ in range(n_steps_local):
623         cores = add_slice_persistent_mps(cores, Ksys_local, Ilag_local, K)
624
625         # Retain newest + K older path-pair sites.
626         if len(cores) > K + 1:
627             cores = discard_oldest_mps_site(cores)
628
629         cores, ranks, disc = compress_persistent_mps(
630             cores, rel_cutoff=rel_cutoff, max_bond=max_bond
631         )
632         rhos.append(_cleanup_rho(reduced_from_mps_cores(cores)))
633         bond_history.append(max(ranks, default=1))
634         storage_history.append(mps_storage_elements(cores))
635         discarded_history.append(disc)
636
637     diagnostics = {
638         'runtime_s': time.perf_counter() - tic,
639         'max_bond': int(np.max(bond_history)),
```

```
        'peak_mps_elements': int(np.max(storage_history)),
        'bond_history': np.asarray(bond_history),
        'storage_history': np.asarray(storage_history),
        'discarded_history': np.asarray(discarded_history),
        'K': int(K),
        'dt': float(dt_local),
        't_final': float(t_final),
    }
    return dt_local*np.arange(n_steps_local+1), np.asarray(rhos), diagnostics


# Demonstration using the notebook's original numerical parameters.
t_pmps, rho_pmps, diag_pmps = persistent_mps_quapi(
    dt, memory_depth, n_steps*dt,
    rel_cutoff=mps_rel_cutoff, max_bond=mps_max_bond
)

hypothetical_dense_elements = 4**(memory_depth+1)  # count only; nothing is allocated
print('Persistent-MPS QUAPI completed')
print(f'  K = {memory_depth}, dt = {dt:g}, t_final = {t_pmps[-1]:g}')
print(f'  largest retained bond dimension = {diag_pmps["max_bond"]}')
print(f'  peak stored MPS coefficients     = {diag_pmps["peak_mps_elements"]:,}')
print(f'  hypothetical dense ADT elements  = {hypothetical_dense_elements:,}')
print(f'  propagation runtime              = {diag_pmps["runtime_s"]:.3f} s')

plt.figure(figsize=(7.0, 4.2))
plt.plot(t_pmps, rho_pmps[:, 0, 0].real, label=r'$\rho_{00}$')
plt.plot(t_pmps, rho_pmps[:, 1, 1].real, label=r'$\rho_{11}$')
plt.plot(t_pmps, rho_pmps[:, 0, 1].real, label=r'Re $\rho_{01}$')
plt.plot(t_pmps, rho_pmps[:, 0, 1].imag, label=r'Im $\rho_{01}$')
plt.xlabel('time')
plt.ylabel('reduced-density-matrix element')
plt.title('Persistent-MPS QUAPI dynamics')
plt.legend()
plt.tight_layout()
plt.show()

# ============================================================================
# NOTEBOOK MARKDOWN CELL 12
# ============================================================================
# ## Convergence of the persistent-MPS result
#
# Three independent numerical limits should be checked:
#
# 1. **Memory depth $K$** at fixed $\Delta t$: increasing $K$ increases the physical memory time $\tau_{\rm
    mem}=K\Delta t$.
# 2. **Trotter time slice $\Delta t$**: here $\tau_{\rm mem}$ is held fixed, so $K$ is increased as $\Delta t$
     is reduced. This avoids confusing Trotter convergence with a simultaneous change in bath-memory time.
# 3. **MPS bond compression**: at fixed $K$ and $\Delta t$, increase the maximum bond dimension $\chi_{\max}$
    while using a tight singular-value cutoff.
#
# The metric below is the maximum absolute error of any reduced-density-matrix element over the propagated
    trajectory relative to the finest calculation in the corresponding sweep. These are pedagogical
    convergence tests, not universal production tolerances.

# ============================================================================
# NOTEBOOK CODE CELL 13
# ============================================================================
# ============================================================================
# Persistent-MPS convergence: K, dt, and maximum bond dimension chi_max
# ============================================================================

_run_cache = {}


def cached_persistent_run(dt_local, K, t_final, rel_cutoff, max_bond):
    key = (float(dt_local), int(K), float(t_final), float(rel_cutoff), int(max_bond))
    if key not in _run_cache:
```

```
        _run_cache[key] = persistent_mps_quapi(
            dt_local, K, t_final,
            rel_cutoff=rel_cutoff, max_bond=max_bond,
            n_cell=25, n_omega=1800
        )
    return _run_cache[key]


def interpolate_rho(t_target, t_ref, rho_ref):
    out = np.empty((len(t_target), 2, 2), dtype=complex)
    for i in range(2):
        for j in range(2):
            out[:, i, j] = (
                np.interp(t_target, t_ref, rho_ref[:, i, j].real)
                + 1j*np.interp(t_target, t_ref, rho_ref[:, i, j].imag)
            )
    return out


def trajectory_error(t, rho, t_ref, rho_ref):
    rho_ref_on_t = interpolate_rho(t, t_ref, rho_ref)
    return float(np.max(np.abs(rho - rho_ref_on_t)))


Tconv = 1.0
chi_tight = 48
cut_tight = 1e-11

# ------------------------------ K convergence ------------------------------
dt_K = 0.10
K_values = [2, 4, 6, 8, 10]
K_reference = K_values[-1]
tK_ref, rK_ref, dK_ref = cached_persistent_run(
    dt_K, K_reference, Tconv, cut_tight, chi_tight
)

K_rows = []
for Kval in K_values:
    tt, rr, dd = cached_persistent_run(dt_K, Kval, Tconv, cut_tight, chi_tight)
    err = trajectory_error(tt, rr, tK_ref, rK_ref)
    K_rows.append((Kval, Kval*dt_K, dd['max_bond'], err))

print('K convergence at fixed dt = 0.10')
print('   K    tau_mem    max bond      max trajectory error')
for Kval, tau, bond, err in K_rows:
    print(f' {Kval:3d}    {tau:6.2f}       {bond:4d}            {err:10.3e}')

plt.figure(figsize=(5.8, 3.8))
plt.semilogy([r[0] for r in K_rows],
             [max(r[3], 1e-16) for r in K_rows], 'o-')
plt.xlabel(r'memory depth $K$')
plt.ylabel(r'max$_{t,ij}|\Delta\rho_{ij}(t)|$')
plt.title(r'Convergence with memory depth $K$')
plt.tight_layout()
plt.show()


# ---------------------------- dt convergence ----------------------------
# Keep the physical memory time fixed while refining dt.
tau_mem_target = 0.60
dt_values = [0.20, 0.10, 0.05]
dt_reference = dt_values[-1]
K_dt = {dti: int(round(tau_mem_target/dti)) for dti in dt_values}

tdt_ref, rdt_ref, ddt_ref = cached_persistent_run(
    dt_reference, K_dt[dt_reference], Tconv, cut_tight, chi_tight
)

```

```
dt_rows = []
for dti in dt_values:
    Kval = K_dt[dti]
    tt, rr, dd = cached_persistent_run(dti, Kval, Tconv, cut_tight, chi_tight)
    err = trajectory_error(tt, rr, tdt_ref, rdt_ref)
    dt_rows.append((dti, Kval, Kval*dti, dd['max_bond'], err))

print('\ndt convergence at fixed physical memory time tau_mem = 0.60')
print('   dt       K    tau_mem    max bond      max trajectory error')
for dti, Kval, tau, bond, err in dt_rows:
    print(f' {dti:6.3f}   {Kval:3d}     {tau:6.2f}       {bond:4d}            {err:10.3e}')

plt.figure(figsize=(5.8, 3.8))
# Sort by increasing dt for a conventional horizontal axis.
dt_plot = sorted(dt_rows, key=lambda x: x[0])
plt.loglog([r[0] for r in dt_plot],
           [max(r[4], 1e-16) for r in dt_plot], 'o-')
plt.xlabel(r'Trotter slice $\Delta t$')
plt.ylabel(r'max$_{t,ij}|\Delta\rho_{ij}(t)|$')
plt.title(r'Convergence with $\Delta t$ at fixed $K\Delta t$')
plt.tight_layout()
plt.show()


# --------------------------- bond-dimension convergence ----------------------
dt_chi = 0.10
K_chi = 6
chi_values = [4, 8, 16, 32, 48, 64]
chi_reference = chi_values[-1]
cut_chi = 1e-13

tchi_ref, rchi_ref, dchi_ref = cached_persistent_run(
    dt_chi, K_chi, Tconv, cut_chi, chi_reference
)

chi_rows = []
for chi in chi_values:
    tt, rr, dd = cached_persistent_run(dt_chi, K_chi, Tconv, cut_chi, chi)
    err = trajectory_error(tt, rr, tchi_ref, rchi_ref)
    chi_rows.append((chi, dd['max_bond'], dd['peak_mps_elements'], err))

print('\nBond-dimension convergence at dt = 0.10, K = 6')
print(' chi_max   used bond    peak MPS coeffs    max trajectory error')
for chi, used, nelem, err in chi_rows:
    print(f' {chi:7d}      {used:4d}          {nelem:8d}            {err:10.3e}')

plt.figure(figsize=(5.8, 3.8))
plt.loglog([r[0] for r in chi_rows],
           [max(r[3], 1e-16) for r in chi_rows], 'o-')
plt.xlabel(r'maximum MPS bond $\chi_{\max}$')
plt.ylabel(r'max$_{t,ij}|\Delta\rho_{ij}(t)|$')
plt.title('Convergence with MPS bond compression')
plt.tight_layout()
plt.show()
```

Listing 1: Complete dense-ADT, reconstruct-after-TT–SVD, persistent-MPS, and convergence code extracted from `Qubit_Ohmic_Persistent.ipynb`.

G Complete executable process tensor source

Checkpoint. Why this appendix is here. Section 17.14 explains the structure of the process-tensor spectroscopy program, the role of the individual routines, and the interpretation of the numerical results. This appendix preserves the full executable

source in notebook order so that every figure and numerical value discussed there can be reproduced from a single listing.

The following listing contains all executable cells from `process_tensor_spectroscopy.ipynb` in their original order. Markdown cells have been converted only to Python comments so that the file remains a standalone, readable script. Running the program reproduces the eight numerical figures discussed in Sec. 17.14, together with the reported process-tensor construction diagnostics, validation tests, pulse scans, two-delay map, multipulse example, and illustrative Fourier analysis.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Standalone source extracted from process_tensor_spectroscopy(1).ipynb.

Notebook cells are preserved in their original order.
Markdown cells are converted to Python comments.
"""

# %% [markdown] Notebook cell 1
# # Process-tensor spectroscopy with a native persistent-MPS implementation
#
# This notebook implements the process-tensor spectroscopy methodology from the accompanying QUAPI/TEMPO tutorial.
#
# The numerical building blocks are taken from the uploaded notebook `Qubit_Ohmic_Persistent(2).ipynb`:
#
# - direct quadrature of the Ohmic bath correlation function;
# - finite-cell influence coefficients $\eta_j$;
# - influence tables $I_j$;
# - the Liouville-space short-time system kernel $G$;
# - persistent-MPS canonicalization and SVD compression.
#
# We extend those routines in one essential way: instead of contracting a predetermined system trajectory while the augmented density tensor is propagated, we leave the intermediate system operations open.
#
# For the spectroscopy protocol
#
# $$
# \text{prepare}\rightarrow \tau_1\rightarrow P\rightarrow \tau_2\rightarrow \text{measure}
# $$
#
# the process tensor is constructed first,
#
# $$
# \{I_{k,k-j}\}\longrightarrow T_{N:0},
# $$
#
# and only afterward do we insert
#
# $$
# I,\ldots, P,\ldots, I.
# $$
#
# The same stored tensor can therefore be reused for different pulse angles, pulse times, multipulse sequences, observables, and initial states.
#
# ## Open-slot convention
#
# At an intervention time $t_k$,
#
# $$
# q_{k-1}\xrightarrow{\;G\;}r_k\xrightarrow{\;A_k\;}q_k.
# $$
#
# Here $r_k$ is the Liouville index immediately before the intervention and $q_k$ is the index immediately after it. The bath influence at slice $k$ is evaluated using the post-intervention path pair $q_k$.
```

```
#
# For a qubit, $\dim q_k=\dim r_k=4$, so one grouped process-tensor slot has physical dimension $4\times4=16$.

# %% [markdown] Notebook cell 2
# ## 1. Imports and model parameters
#
# We use the same spin-boson model as the persistent-MPS notebook,
#
# $$
# H_S=\frac{\epsilon}{2}\sigma_z+\frac{\Delta}{2}\sigma_x,
# \qquad H_{SB}=\sigma_z\otimes B,
# $$
#
# with
#
# $$
# J(\omega)=2\pi\alpha_{\rm SB}\omega e^{-\omega/\omega_c},
# $$
#
# and units $\hbar=k_B=1$.

# %% Notebook cell 3
import time
from dataclasses import dataclass

import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import expm

np.set_printoptions(precision=6, suppress=True)

# Physical parameters from Qubit_Ohmic_Persistent(2).ipynb
epsilon = 1.0
Delta = 1.0
alpha_SB = 0.08
omega_c = 5.0
temperature = 0.5
beta = 1.0 / temperature

# Process-tensor grid
dt = 0.10
memory_depth = 6

# A full open-slot process tensor is more demanding than one fixed trajectory.
# N=18 keeps this pedagogical notebook responsive; increase after convergence tests.
n_steps_pt = 18
t_max = n_steps_pt * dt

pt_rel_cutoff = 1.0e-10
pt_max_bond = 48

# Bath quadrature
n_cell = 21
n_omega = 1400

print(f"Process-tensor horizon: t_max = {t_max:.2f}")
print(f"Memory time: K*dt = {memory_depth*dt:.2f}")
print(f"Open intervention slots: {n_steps_pt-1}")

# %% [markdown] Notebook cell 4
# ## 2. Bath correlation and influence coefficients
#
# The native quadrature routines implement
#
# $$
# \alpha_B(t)=\frac{1}{\pi}\int_0^\infty d\omega\,J(\omega)
# \left[\coth\!\left(\frac{\beta\omega}{2}\right)\cos\omega t-i\sin\omega t\right].
# $$
```

```
#
# For $j\ge1$, $\eta_j$ is integrated over two complete time cells separated by $j\Delta t$. For $j=0$, the
    causal triangle is retained.

# %% Notebook cell 5
def coth_stable(x):
    x = np.asarray(x)
    out = np.empty_like(x, dtype=float)
    small = np.abs(x) < 1e-5
    out[small] = 1/x[small] + x[small]/3
    out[~small] = 1/np.tanh(x[~small])
    return out


_bath_grid_cache = {}
_eta_mps_cache = {}


def _bath_alpha_for_mps(t, n_omega=2500, omega_max_factor=12.0):
    """Ohmic bath correlation with a cached frequency grid."""
    key = (int(n_omega), float(omega_max_factor), float(alpha_SB),
           float(omega_c), float(beta))
    if key not in _bath_grid_cache:
        w = np.linspace(1e-7, omega_max_factor*omega_c, n_omega)
        J = 2*np.pi*alpha_SB*w*np.exp(-w/omega_c)
        thermal = coth_stable(0.5*beta*w)
        _bath_grid_cache[key] = (w, J/np.pi, thermal)
    w, pref, thermal = _bath_grid_cache[key]

    t = np.atleast_1d(t).astype(float)
    phase = np.outer(t, w)
    values = np.trapezoid(
        pref[None, :]*(thermal[None, :]*np.cos(phase) - 1j*np.sin(phase)),
        w, axis=1
    )
    return values


def influence_coefficients_mps(K, dt_local, n_cell=31, n_omega=2500):
    """Finite-cell eta_j with the causal j=0 triangle, cached for sweeps."""
    key = (int(K), float(dt_local), int(n_cell), int(n_omega))
    if key in _eta_mps_cache:
        return _eta_mps_cache[key].copy()

    x = np.linspace(-dt_local/2, dt_local/2, n_cell)
    h = x[1] - x[0]
    q = np.arange(-(n_cell-1), n_cell)
    diff_index = (np.arange(n_cell)[:, None] - np.arange(n_cell)[None, :]) + (n_cell-1)

    eta_local = np.zeros(K+1, dtype=complex)
    for j in range(K+1):
        tau_unique = j*dt_local + q*h
        alpha_unique = _bath_alpha_for_mps(tau_unique, n_omega=n_omega)
        vals = alpha_unique[diff_index]
        if j == 0:
            mask = np.tril(np.ones((n_cell, n_cell)), k=-1) + 0.5*np.eye(n_cell)
            vals = vals*mask
        eta_local[j] = np.trapezoid(np.trapezoid(vals, x, axis=1), x, axis=0)

    _eta_mps_cache[key] = eta_local.copy()
    return eta_local

# %% Notebook cell 6
eta = influence_coefficients_mps(
    memory_depth, dt, n_cell=n_cell, n_omega=n_omega
)

print(" lag j        Re(eta_j)          Im(eta_j)          |eta_j|")
```

```
for j, z in enumerate(eta):
    print(f" {j:3d}      {z.real: .8e}    {z.imag: .8e}    {abs(z): .8e}")

tau_plot = np.linspace(0.0, 2.0*memory_depth*dt, 200)
alpha_plot = _bath_alpha_for_mps(tau_plot, n_omega=n_omega)

plt.figure(figsize=(7, 4))
plt.plot(tau_plot, alpha_plot.real, label=r"$\mathrm{Re}\,\alpha_B(t)$")
plt.plot(tau_plot, alpha_plot.imag, label=r"$\mathrm{Im}\,\alpha_B(t)$")
plt.axvline(memory_depth*dt, linestyle="--", label=r"$K\Delta t$")
plt.xlabel("time")
plt.ylabel("bath correlation")
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(7, 4))
plt.semilogy(np.arange(memory_depth+1), np.maximum(np.abs(eta), 1e-18), marker="o")
plt.xlabel("lag $j$")
plt.ylabel(r"$|\eta_j|$")
plt.tight_layout()
plt.show()

# %% [markdown] Notebook cell 7
# ## 3. Liouville-space kernels and influence tables
#
# For the row-major path-pair ordering $a=2s^+ + s^-$,
#
# $$
# G(a',a)=\langle s^{\prime +}|U_{\Delta t}|s^+\rangle
# \langle s^{\prime -}|U_{\Delta t}|s^-\rangle^*.
# $$
#
# The finite-memory bath factor is
#
# $$
# I_j(a_k,a_{k-j})=
# \exp\!\left[-(s_k^+-s_k^-)
# \left(\eta_j s_{k-j}^+-\eta_j^*s_{k-j}^-\right)\right].
# $$

# %% Notebook cell 8
sx = np.array([[0, 1], [1, 0]], dtype=complex)
sy = np.array([[0, -1j], [1j, 0]], dtype=complex)
sz = np.array([[1, 0], [0, -1]], dtype=complex)

_PAIR_SP = np.repeat(np.array([+1.0, -1.0]), 2)
_PAIR_SM = np.tile(np.array([+1.0, -1.0]), 2)


def system_kernel_dt(dt_local):
    """G[a_new,a_old] for one isolated-system interval."""
    H_local = 0.5*epsilon*sz + 0.5*Delta*sx
    U_local = expm(-1j*H_local*dt_local)
    G = np.empty((4, 4), dtype=complex)
    for a_new in range(4):
        ip, im = divmod(a_new, 2)
        for a_old in range(4):
            jp, jm = divmod(a_old, 2)
            G[a_new, a_old] = U_local[ip, jp]*U_local[im, jm].conjugate()
    return G


def influence_tables_mps(eta_local):
    return [
        np.exp(-(_PAIR_SP[:, None]-_PAIR_SM[:, None]) *
               (z*_PAIR_SP[None, :] - z.conjugate()*_PAIR_SM[None, :]))
        for z in eta_local
```

```
    ]


Ilag = influence_tables_mps(eta)
Gdt = system_kernel_dt(dt)
print("G shape:", Gdt.shape)
print("number of influence tables:", len(Ilag))
print("each I_j shape:", Ilag[0].shape)

# %% [markdown] Notebook cell 9
# ## 4. Persistent-MPS primitives
#
# An MPS core has shape
#
# $$
# (\chi_{\rm left},d_{\rm phys},\chi_{\rm right}).
# $$
#
# For a reusable process tensor we must preserve linearity. An arbitrary intervention need not be trace preserving, so any scalar removed from the MPS for numerical conditioning is accumulated explicitly in `global_scale`; it is never discarded by renormalizing the output state.

# %% Notebook cell 10
def right_canonicalize_mps(cores):
    """Right-canonicalize using QR on local core matricizations only."""
    cores = [c.copy() for c in cores]
    for i in range(len(cores)-1, 0, -1):
        chi_l, dphys, chi_r = cores[i].shape
        M = cores[i].reshape(chi_l, dphys*chi_r)
        Q, R = np.linalg.qr(M.T, mode="reduced")
        chi_new = Q.shape[1]
        cores[i] = Q.T.reshape(chi_new, dphys, chi_r)
        cores[i-1] = np.tensordot(cores[i-1], R.T, axes=(2, 0))
    return cores


def compress_process_tensor_mps(cores, rel_cutoff=1e-10, max_bond=64):
    """Compress while preserving the represented tensor's overall scale."""
    if len(cores) <= 1:
        return [c.copy() for c in cores], [], 0.0, 1.0

    cores = right_canonicalize_mps(cores)
    ranks = []
    discarded_sq = 0.0

    for i in range(len(cores)-1):
        chi_l, dphys, chi_r = cores[i].shape
        M = cores[i].reshape(chi_l*dphys, chi_r)
        Uloc, s, Vh = np.linalg.svd(M, full_matrices=False)
        threshold = rel_cutoff*s[0] if s.size else 0.0
        keep = max(1, int(np.count_nonzero(s > threshold)))
        if max_bond is not None:
            keep = min(keep, int(max_bond))
        discarded_sq += float(np.sum(s[keep:]**2))
        cores[i] = Uloc[:, :keep].reshape(chi_l, dphys, keep)
        transfer = s[:keep, None]*Vh[:keep, :]
        cores[i+1] = np.tensordot(transfer, cores[i+1], axes=(1, 0))
        ranks.append(keep)

    extracted_scale = np.linalg.norm(cores[-1])
    if extracted_scale > 0.0:
        cores[-1] = cores[-1]/extracted_scale
    else:
        extracted_scale = 1.0

    return cores, ranks, np.sqrt(discarded_sq), extracted_scale


```

```
def mps_storage_elements(cores):
    return int(sum(core.size for core in cores))


def max_mps_bond(cores):
    if len(cores) <= 1:
        return 1
    return int(max(max(core.shape[0], core.shape[2]) for core in cores))

# %% [markdown] Notebook cell 11
# ## 5. Add an open intervention slot
#
# Define the grouped physical index
#
# $$
# x_k=(q_k,r_k),\qquad x_k=4q_k+r_k,
# $$
#
# with $x_k=0,\ldots,15$.
#
# The local network contains $G(r_k,q_{k-1})$ but **does not contain** $[A_k]_{q_k r_k}$. The latter remains open for later contraction. The bath factors depend on $q_k$:
#
# $$
# I_0(q_k)I_1(q_k,q_{k-1})\cdots I_K(q_k,q_{k-K}).
# $$
#
# The new $q_k$ label is threaded through at most $K$ temporal bonds.

# %% Notebook cell 12
def _q_values_for_site(dphys):
    if dphys == 4:
        return np.arange(4)
    if dphys == 16:
        return np.arange(16)//4
    raise ValueError(f"Unexpected physical dimension {dphys}")


def add_open_intervention_slot(cores, G_local, I_local, K):
    """Prepend one open x=(q,r) process-tensor slot."""
    d_liouville = 4
    d_slot = 16
    n_old = len(cores)

    # First virtual bond carries x=(q,r), because G needs r at lag 1.
    newest = np.zeros((1, d_slot, d_slot), dtype=complex)
    for x in range(d_slot):
        q, r = divmod(x, d_liouville)
        newest[0, x, x] = I_local[0][q, q]
    out = [newest]

    for i, Gcore in enumerate(cores):
        chi_l, d_old, chi_r = Gcore.shape
        lag = i + 1
        q_old = _q_values_for_site(d_old)
        is_oldest = (i == n_old-1)

        if lag == 1:
            terminate_q = (K <= 1) or is_oldest
            right_dim = chi_r if terminate_q else d_liouville*chi_r
            C = np.zeros((d_slot*chi_l, d_old, right_dim), dtype=complex)

            for x in range(d_slot):
                q, r = divmod(x, d_liouville)
                factor = G_local[r, q_old]
                if K >= 1:
                    factor = factor*I_local[1][q, q_old]
                left = x*chi_l
```

```
                if terminate_q:
                    C[left:left+chi_l, :, :] += Gcore*factor[None, :, None]
                else:
                    right = q*chi_r
                    C[left:left+chi_l, :, right:right+chi_r] += Gcore*factor[None, :, None]
            out.append(C)

        elif lag <= K:
            terminate_q = (lag == K) or is_oldest
            right_dim = chi_r if terminate_q else d_liouville*chi_r
            C = np.zeros((d_liouville*chi_l, d_old, right_dim), dtype=complex)

            for q in range(d_liouville):
                factor = I_local[lag][q, q_old]
                left = q*chi_l
                if terminate_q:
                    C[left:left+chi_l, :, :] += Gcore*factor[None, :, None]
                else:
                    right = q*chi_r
                    C[left:left+chi_l, :, right:right+chi_r] += Gcore*factor[None, :, None]
            out.append(C)

        else:
            out.append(Gcore.copy())

    return out

# %% [markdown] Notebook cell 13
# ## 6. Add the final open reduced-state leg
#
# At $t_N$ there is no additional intervention in this notebook. The final four-component Liouville index
    $q_N$ remains open and becomes the final reduced density matrix after all earlier slots are contracted.

# %% Notebook cell 14
def add_final_state_leg(cores, G_local, I_local, K):
    """Prepend the final open Liouville-state leg q_N."""
    d_liouville = 4
    n_old = len(cores)

    newest = np.zeros((1, d_liouville, d_liouville), dtype=complex)
    newest[0, np.arange(d_liouville), np.arange(d_liouville)] = np.diag(I_local[0])
    out = [newest]

    for i, Gcore in enumerate(cores):
        chi_l, d_old, chi_r = Gcore.shape
        lag = i + 1
        q_old = _q_values_for_site(d_old)
        is_oldest = (i == n_old-1)

        if lag == 1:
            terminate_q = (K <= 1) or is_oldest
            right_dim = chi_r if terminate_q else d_liouville*chi_r
            C = np.zeros((d_liouville*chi_l, d_old, right_dim), dtype=complex)
            for q in range(d_liouville):
                factor = G_local[q, q_old]
                if K >= 1:
                    factor = factor*I_local[1][q, q_old]
                left = q*chi_l
                if terminate_q:
                    C[left:left+chi_l, :, :] += Gcore*factor[None, :, None]
                else:
                    right = q*chi_r
                    C[left:left+chi_l, :, right:right+chi_r] += Gcore*factor[None, :, None]
            out.append(C)

        elif lag <= K:
            terminate_q = (lag == K) or is_oldest
            right_dim = chi_r if terminate_q else d_liouville*chi_r
```

```
            C = np.zeros((d_liouville*chi_l, d_old, right_dim), dtype=complex)
            for q in range(d_liouville):
                factor = I_local[lag][q, q_old]
                left = q*chi_l
                if terminate_q:
                    C[left:left+chi_l, :, :] += Gcore*factor[None, :, None]
                else:
                    right = q*chi_r
                    C[left:left+chi_l, :, right:right+chi_r] += Gcore*factor[None, :, None]
            out.append(C)

        else:
            out.append(Gcore.copy())

    return out

# %% [markdown] Notebook cell 15
# ## 7. Build $T_{N:0}$ once
#
# The initial density matrix is not inserted during construction. The final temporal MPS has the structure
#
# $$
# q_N-(q_{N-1},r_{N-1})-\cdots-(q_1,r_1)-q_0.
# $$
#
# Thus its physical dimensions, from newest to oldest, are
#
# $$
# [4,16,16,\ldots,16,4].
# $$
#
# Grouping each $(q_k,r_k)$ pair into one 16-dimensional physical leg gives an MPS representation along time; splitting that leg into input/output factors gives the usual process-tensor MPO picture.

# %% Notebook cell 16
@dataclass
class ProcessTensorMPS:
    cores: list
    global_scale: complex
    eta: np.ndarray
    influence_tables: list
    system_kernel: np.ndarray
    dt: float
    N: int
    K: int
    diagnostics: dict


def build_process_tensor_mps(dt_local, K, N, rel_cutoff=1e-10,
                             max_bond=64, n_cell=31, n_omega=2500):
    """Construct a reusable finite-memory process tensor as a temporal MPS."""
    assert K >= 1
    assert N >= 1

    eta_local = influence_coefficients_mps(K, dt_local, n_cell=n_cell, n_omega=n_omega)
    I_local = influence_tables_mps(eta_local)
    G_local = system_kernel_dt(dt_local)

    # q_0 is open; rho_0 is contracted later.
    cores = [np.diag(I_local[0]).reshape(1, 4, 1).copy()]
    global_scale = 1.0
    max_bond_history = [1]
    storage_history = [mps_storage_elements(cores)]
    discarded_history = [0.0]

    tic = time.perf_counter()

    for k in range(1, N):
```

```
        cores = add_open_intervention_slot(cores, G_local, I_local, K)
        cores, ranks, discarded, scale = compress_process_tensor_mps(
            cores, rel_cutoff=rel_cutoff, max_bond=max_bond
        )
        global_scale *= scale
        max_bond_history.append(max(ranks, default=1))
        storage_history.append(mps_storage_elements(cores))
        discarded_history.append(discarded)

    cores = add_final_state_leg(cores, G_local, I_local, K)
    cores, ranks, discarded, scale = compress_process_tensor_mps(
        cores, rel_cutoff=rel_cutoff, max_bond=max_bond
    )
    global_scale *= scale
    max_bond_history.append(max(ranks, default=1))
    storage_history.append(mps_storage_elements(cores))
    discarded_history.append(discarded)

    diagnostics = {
        'build_runtime_s': time.perf_counter()-tic,
        'max_bond': int(np.max(max_bond_history)),
        'peak_mps_elements': int(np.max(storage_history)),
        'max_bond_history': np.asarray(max_bond_history),
        'storage_history': np.asarray(storage_history),
        'discarded_history': np.asarray(discarded_history),
        'rel_cutoff': float(rel_cutoff),
        'max_bond_requested': max_bond,
    }

    return ProcessTensorMPS(
        cores=cores, global_scale=global_scale, eta=eta_local,
        influence_tables=I_local, system_kernel=G_local,
        dt=float(dt_local), N=int(N), K=int(K), diagnostics=diagnostics
    )

# %% Notebook cell 17
process_tensor = build_process_tensor_mps(
    dt_local=dt,
    K=memory_depth,
    N=n_steps_pt,
    rel_cutoff=pt_rel_cutoff,
    max_bond=pt_max_bond,
    n_cell=n_cell,
    n_omega=n_omega,
)

print("Process tensor built.")
print(f"  horizon                    = {process_tensor.N*process_tensor.dt:.2f}")
print(f"  intervention slots         = {process_tensor.N-1}")
print(f"  largest retained bond      = {process_tensor.diagnostics['max_bond']}")
print(f"  peak stored coefficients   = {process_tensor.diagnostics['peak_mps_elements']:,}")
print(f"  build runtime              = {process_tensor.diagnostics['build_runtime_s']:.3f} s")
print("\nTemporal physical dimensions, newest -> oldest:")
print([core.shape[1] for core in process_tensor.cores])

plt.figure(figsize=(7, 4))
plt.plot(np.arange(len(process_tensor.diagnostics['max_bond_history'])),
         process_tensor.diagnostics['max_bond_history'], marker="o")
plt.xlabel("process-tensor construction step")
plt.ylabel("largest MPS bond")
plt.tight_layout()
plt.show()

# %% [markdown] Notebook cell 18
# No pulse angle, pulse time, observable, or initial density matrix was used in the construction above. Those
    are experiment-specific objects and are inserted only during contraction.

# %% [markdown] Notebook cell 19
```

```
# ## 8. Define intervention superoperators
#
# For a unitary pulse
#
# $$
# U_{\hat n}(\theta)=e^{-i\theta\hat n\cdot\boldsymbol\sigma/2},
# $$
#
# the row-major Liouville superoperator is
#
# $$
# P_\theta=U\otimes U^*.
# $$

# %% Notebook cell 20
def unitary_pulse(theta, axis="x"):
    axis_operator = {"x": sx, "y": sy, "z": sz}[axis]
    return expm(-0.5j*theta*axis_operator)


def unitary_superoperator(U):
    """Row-major Liouville map rho -> U rho U^dagger."""
    return np.kron(U, U.conjugate())


def pulse_superoperator(theta, axis="x"):
    return unitary_superoperator(unitary_pulse(theta, axis))


identity_superoperator = np.eye(4, dtype=complex)

# Verify the vectorization convention.
rho_check = np.array([[0.7, 0.2+0.1j], [0.2-0.1j, 0.3]], dtype=complex)
U_check = unitary_pulse(np.pi/3, "x")
P_check = unitary_superoperator(U_check)
rho_direct = U_check @ rho_check @ U_check.conj().T
rho_liouville = (P_check @ rho_check.reshape(4)).reshape(2, 2)
print("maximum unitary-superoperator error =",
      np.max(np.abs(rho_direct-rho_liouville)))

# %% [markdown] Notebook cell 21
# ## 9. Contract an experiment with the stored process tensor
#
# For selected interventions $\{A_k\}$,
#
# $$
# \rho_S(t_N)=T_{N:0}[A_{N-1},\ldots,A_1,\rho_0].
# $$
#
# Any unspecified slot is filled with the identity map. The stored process tensor itself is not modified.

# %% Notebook cell 22
def contract_process_tensor(pt, rho0, operations=None):
    """Contract rho0 and selected 4x4 intervention maps with a stored PT-MPS."""
    if operations is None:
        operations = {}

    rho0 = np.asarray(rho0, dtype=complex)

    # Oldest core: q_0 input.
    G0 = pt.cores[-1]
    env = np.einsum(
        "lpr,p,r->l",
        G0,
        rho0.reshape(4),
        np.ones(G0.shape[2], dtype=complex),
    )

```

```
    # Cores are [q_N, slot_(N-1), ..., slot_1, q_0].
    for core_index in range(len(pt.cores)-2, 0, -1):
        Gslot = pt.cores[core_index]
        k = pt.N-core_index
        A = np.asarray(operations.get(k, identity_superoperator), dtype=complex)
        if A.shape != (4, 4):
            raise ValueError(f"Operation at step {k} must have shape (4,4).")
        env = np.einsum("lpr,p,r->l", Gslot, A.reshape(16), env)

    # Newest core leaves q_N open.
    Gf = pt.cores[0]
    rho_final_vec = np.einsum("lpr,r->p", Gf, env)
    return (pt.global_scale*rho_final_vec).reshape(2, 2)


def expectation(rho, observable):
    return np.trace(observable @ rho)


rho_plus_x = 0.5*np.array([[1, 1], [1, 1]], dtype=complex)
rho_identity = contract_process_tensor(process_tensor, rho_plus_x)

print("No-pulse final state:")
print(rho_identity)
print("trace =", np.trace(rho_identity))
print("Hermiticity error =",
      np.max(np.abs(rho_identity-rho_identity.conj().T)))

# %% [markdown] Notebook cell 23
# ## 10. Exact small-network check
#
# We verify the open-slot construction against ordinary finite-memory QUAPI with no SVD truncation. At pulse time $k$,
#
# $$
# q_{k-1}\xrightarrow{G}r_k\xrightarrow{P}q_k,
# $$
#
# so direct propagation uses
#
# $$
# G_{\rm eff}=P G.
# $$
#
# The direct and process-tensor contractions should agree to roundoff.

# %% Notebook cell 24
def add_slice_persistent_mps(cores, G_local, I_local, K):
    """Ordinary persistent-MPS QUAPI slice from the uploaded notebook."""
    dphys = 4
    n_old = len(cores)
    new0 = np.zeros((1, dphys, dphys), dtype=complex)
    new0[0, np.arange(dphys), np.arange(dphys)] = np.diag(I_local[0])
    out = [new0]

    for i, Gcore in enumerate(cores):
        chi_l, d_old, chi_r = Gcore.shape
        lag = i+1
        if lag == 1:
            F = G_local.copy()
            if K >= 1:
                F *= I_local[1]
        elif lag <= K:
            F = I_local[lag]
        else:
            F = np.ones((dphys, d_old), dtype=complex)

        if i == n_old-1:
```

```
            C = np.zeros((dphys*chi_l, d_old, 1), dtype=complex)
            for q in range(dphys):
                C[q*chi_l:(q+1)*chi_l, :, 0] = Gcore[:, :, 0]*F[q, :][None, :]
        else:
            C = np.zeros((dphys*chi_l, d_old, dphys*chi_r), dtype=complex)
            for q in range(dphys):
                C[q*chi_l:(q+1)*chi_l, :, q*chi_r:(q+1)*chi_r] = \
                    Gcore*F[q, :][None, :, None]
        out.append(C)
    return out


def discard_oldest_mps_site(cores):
    if len(cores) <= 1:
        return cores
    boundary_vec = cores[-1].sum(axis=1)[:, 0]
    new_last = np.tensordot(cores[-2], boundary_vec, axes=(2, 0))[:, :, None]
    return cores[:-2] + [new_last]


def reduced_from_mps_cores(cores):
    env = np.ones(1, dtype=complex)
    for core in cores[:0:-1]:
        env = core.sum(axis=1) @ env
    return (cores[0][0] @ env).reshape(2, 2)


def direct_controlled_quapi_final(dt_local, K, N, rho0, operations=None,
                                  n_cell=15, n_omega=800):
    if operations is None:
        operations = {}

    eta_local = influence_coefficients_mps(K, dt_local, n_cell=n_cell, n_omega=n_omega)
    I_local = influence_tables_mps(eta_local)
    G_local = system_kernel_dt(dt_local)

    A0 = np.asarray(rho0).reshape(4)*np.diag(I_local[0])
    cores = [A0.reshape(1, 4, 1)]
    global_scale = 1.0

    for k in range(1, N+1):
        G_eff = G_local
        if k < N:
            G_eff = np.asarray(operations.get(k, identity_superoperator)) @ G_local

        cores = add_slice_persistent_mps(cores, G_eff, I_local, K)
        if len(cores) > K+1:
            cores = discard_oldest_mps_site(cores)

        cores, _, _, scale = compress_process_tensor_mps(
            cores, rel_cutoff=0.0, max_bond=None
        )
        global_scale *= scale

    return global_scale*reduced_from_mps_cores(cores)


K_test = 3
N_test = 6
pulse_step_test = 3
P_test = pulse_superoperator(np.pi/2, "x")

pt_test = build_process_tensor_mps(
    dt_local=dt, K=K_test, N=N_test, rel_cutoff=0.0,
    max_bond=None, n_cell=15, n_omega=800
)

rho_from_pt = contract_process_tensor(
```

```
    pt_test, rho_plus_x, operations={pulse_step_test: P_test}
)
rho_from_direct = direct_controlled_quapi_final(
    dt, K_test, N_test, rho_plus_x,
    operations={pulse_step_test: P_test}, n_cell=15, n_omega=800
)

print("max |rho_PT - rho_direct_QUAPI| =",
      np.max(np.abs(rho_from_pt-rho_from_direct)))

# %% [markdown] Notebook cell 25
# ## 11. One-pulse spectroscopy
#
# Take $\rho_{\rm prep}=|+x\rangle\langle+x|$, insert a $\pi/2$ $x$-pulse at $\tau_1=m_1\Delta t$, and measure
      $M=\sigma_z$ at $t_N$. Then
#
# $$
# S(\tau_1,\tau_2)=\operatorname{Tr}\!\left[\sigma_z\,T_{N:0}(\ldots,P_{\pi/2},\ldots,\rho_{\rm prep})\right],
# $$
#
# with $\tau_2=t_N-\tau_1$.

# %% Notebook cell 26
m1 = 6
tau1 = m1*dt
tau2 = t_max-tau1

rho_final = contract_process_tensor(
    process_tensor,
    rho_plus_x,
    operations={m1: pulse_superoperator(np.pi/2, "x")},
)
signal = expectation(rho_final, sz)

print(f"tau1 = {tau1:.2f}")
print(f"tau2 = {tau2:.2f}")
print(f"S(tau1,tau2) = {signal.real:.8f}")
print(f"trace(final rho) = {np.trace(rho_final)}")

# %% [markdown] Notebook cell 27
# ## 12. Reuse the same process tensor: pulse-angle scan
#
# Only $P_{\pi/2}\to P_\theta$ changes. No bath coefficient or temporal tensor is rebuilt.

# %% Notebook cell 28
angles = np.linspace(0.0, 2.0*np.pi, 31)
angle_signal = []

tic = time.perf_counter()
for theta in angles:
    rho = contract_process_tensor(
        process_tensor, rho_plus_x,
        operations={m1: pulse_superoperator(theta, "x")}
    )
    angle_signal.append(expectation(rho, sz).real)
contraction_time = time.perf_counter()-tic
angle_signal = np.asarray(angle_signal)

print(f"{len(angles)} contractions with one stored PT: {contraction_time:.3f} s")

plt.figure(figsize=(7, 4))
plt.plot(angles/np.pi, angle_signal, marker="o")
plt.xlabel(r"pulse angle $\theta/\pi$")
plt.ylabel(r"$S=\langle\sigma_z(t_N)\rangle$")
plt.tight_layout()
plt.show()

# %% [markdown] Notebook cell 29
```

```
# ## 13. Reuse the same process tensor: move the pulse
#
# Changing $\tau_1$ only changes which open slot contains $P$. At fixed $t_N$, $\tau_2=t_N-\tau_1$.

# %% Notebook cell 30
pulse_steps = np.arange(2, n_steps_pt-1)
pulse_times = pulse_steps*dt
pulse_time_signal = []
P_half_pi = pulse_superoperator(np.pi/2, "x")

for step in pulse_steps:
    rho = contract_process_tensor(
        process_tensor, rho_plus_x, operations={int(step): P_half_pi}
    )
    pulse_time_signal.append(expectation(rho, sz).real)

plt.figure(figsize=(7, 4))
plt.plot(pulse_times, pulse_time_signal, marker="o")
plt.xlabel(r"$\tau_1$")
plt.ylabel(r"$S(\tau_1,t_N-\tau_1)$")
plt.tight_layout()
plt.show()

# %% [markdown] Notebook cell 31
# ## 14. Extract an earlier-time signal from the longest process tensor
#
# To obtain $S(t_m)=\operatorname{Tr}[M\rho(t_m)]$ for $m<N$ without rebuilding a shorter tensor, insert the
    linear map
#
# $$
# R_M[\rho]=\operatorname{Tr}(M\rho)\rho_{\rm reset},\qquad \operatorname{Tr}\rho_{\rm reset}=1.
# $$
#
# Trace-preserving identity operations are inserted afterward. By causality,
#
# $$
# \operatorname{Tr}\rho(t_N)=\operatorname{Tr}[M\rho(t_m)].
# $$
#
# This map is a contraction device, not a model of a physical detector.

# %% Notebook cell 32
rho_reset = np.array([[1.0, 0.0], [0.0, 0.0]], dtype=complex)


def observable_and_reset_superoperator(observable, reset_state=rho_reset):
    observable = np.asarray(observable, dtype=complex)
    reset_state = np.asarray(reset_state, dtype=complex)
    # Row-major vec: Tr(M rho) = vec(M^T) dot vec(rho).
    return np.outer(reset_state.reshape(4), observable.T.reshape(4))


def signal_at_intermediate_step(pt, rho0, observable, measurement_step,
                                operations_before_measurement=None):
    if not (1 <= measurement_step <= pt.N):
        raise ValueError("measurement_step must satisfy 1 <= m <= N")

    operations = {}
    if operations_before_measurement is not None:
        operations.update(operations_before_measurement)

    if measurement_step == pt.N:
        rho = contract_process_tensor(pt, rho0, operations=operations)
        return expectation(rho, observable)

    if measurement_step in operations:
        raise ValueError("Measurement slot must differ from pulse slots in this helper.")

```

```
    operations[measurement_step] = observable_and_reset_superoperator(observable)
    rho_end = contract_process_tensor(pt, rho0, operations=operations)
    return np.trace(rho_end)

# %% [markdown] Notebook cell 33
# ## 15. Two-delay map $S(\tau_1,\tau_2)$ from one process tensor
#
# For each pair $\tau_1=m_1\Delta t$ and $\tau_2=m_2\Delta t$, insert the pulse at $m_1$ and the observable
    contraction at $m_1+m_2$. Every other slot remains the identity.

# %% Notebook cell 34
tau1_steps = np.array([2, 4, 6, 8])
tau2_steps = np.array([2, 4, 6, 8])
S_map = np.empty((len(tau2_steps), len(tau1_steps)), dtype=float)

for i2, m2 in enumerate(tau2_steps):
    for i1, m1_scan in enumerate(tau1_steps):
        measurement_step = int(m1_scan+m2)
        if measurement_step >= n_steps_pt:
            S_map[i2, i1] = np.nan
            continue
        S = signal_at_intermediate_step(
            process_tensor, rho_plus_x, sz,
            measurement_step=measurement_step,
            operations_before_measurement={int(m1_scan): P_half_pi},
        )
        S_map[i2, i1] = S.real

plt.figure(figsize=(6.5, 5))
plt.imshow(
    S_map, origin="lower", aspect="auto",
    extent=[tau1_steps[0]*dt, tau1_steps[-1]*dt,
            tau2_steps[0]*dt, tau2_steps[-1]*dt]
)
plt.xlabel(r"$\tau_1$")
plt.ylabel(r"$\tau_2$")
plt.colorbar(label=r"$S(\tau_1,\tau_2)$")
plt.tight_layout()
plt.show()

# %% [markdown] Notebook cell 35
# ## 16. Multipulse sequence
#
# For
#
# $$
# \text{prepare}\to\tau_1\to P_1\to\tau_2\to P_2\to\tau_3\to P_3\to\text{measure},
# $$
#
# we simply place three different superoperators into three existing slots.

# %% Notebook cell 36
multipulse_operations = {
    3: pulse_superoperator(np.pi/2, "x"),
    7: pulse_superoperator(np.pi/2, "y"),
    12: pulse_superoperator(np.pi, "x"),
}

rho_multi = contract_process_tensor(
    process_tensor, rho_plus_x, operations=multipulse_operations
)
print("Three-pulse signal =", expectation(rho_multi, sz).real)
print("final trace =", np.trace(rho_multi))

# %% [markdown] Notebook cell 37
# ## 17. Time-domain signal and illustrative Fourier spectrum
#
# We can extract $\langle\sigma_z(t)\rangle$ at many post-pulse times from the same longest tensor and Fourier
```

```
     transform the resulting record.
#
# This is only an illustrative spectroscopy transform. A formal optical absorption or multidimensional
    response requires a specified dipole operator, pulse convention, Liouville pathways or phase cycling, and
     the appropriate response-function definition.

# %% Notebook cell 38
pulse_step = 3
measurement_steps = np.arange(pulse_step+1, n_steps_pt+1)
time_signal = []

for m in measurement_steps:
    S = signal_at_intermediate_step(
        process_tensor, rho_plus_x, sz,
        measurement_step=int(m),
        operations_before_measurement={pulse_step: P_half_pi},
    )
    time_signal.append(S.real)

time_signal = np.asarray(time_signal)
post_times = measurement_steps*dt

plt.figure(figsize=(7, 4))
plt.plot(post_times, time_signal, marker="o")
plt.xlabel("measurement time")
plt.ylabel(r"$\langle\sigma_z\rangle$")
plt.tight_layout()
plt.show()

y = time_signal-time_signal.mean()
window = np.hanning(len(y))
spectrum = np.fft.rfft(window*y)
omega = 2*np.pi*np.fft.rfftfreq(len(y), d=dt)

plt.figure(figsize=(7, 4))
plt.plot(omega, np.abs(spectrum))
plt.xlabel(r"angular frequency $\omega$")
plt.ylabel("Fourier amplitude")
plt.tight_layout()
plt.show()

# %% [markdown] Notebook cell 39
# ## 18. Causality check
#
# For a small uncompressed network, compare a tensor ending exactly at the measurement time with the longer
    tensor plus the observable-and-reset contraction. The signals should agree to roundoff.

# %% Notebook cell 40
measurement_test = 4
pulse_test = 2

pt_short = build_process_tensor_mps(
    dt_local=dt, K=3, N=measurement_test, rel_cutoff=0.0,
    max_bond=None, n_cell=15, n_omega=800
)

rho_short = contract_process_tensor(
    pt_short, rho_plus_x,
    operations={pulse_test: pulse_superoperator(np.pi/2, "x")}
)
S_short = expectation(rho_short, sz)

S_from_long = signal_at_intermediate_step(
    pt_test, rho_plus_x, sz,
    measurement_step=measurement_test,
    operations_before_measurement={pulse_test: pulse_superoperator(np.pi/2, "x")},
)

```

```
print("signal from short PT =", S_short)
print("signal from long PT  =", S_from_long)
print("absolute difference  =", abs(S_short-S_from_long))

# %% [markdown] Notebook cell 41
# ## 19. Process-tensor convergence
#
# Three numerical limits should be checked independently:
#
# 1. memory depth $K$ at fixed $\Delta t$;
# 2. time step $\Delta t$ at fixed physical memory time $K\Delta t$;
# 3. MPS compression, by tightening the singular-value threshold and increasing $\chi_{\max}$.
#
# Convergence of a process tensor is stricter than convergence of one trajectory because the reusable object must support many possible intervention sequences. The optional cell below varies $\chi_{\max}$ for one spectroscopy protocol.

# %% Notebook cell 42
RUN_PT_CONVERGENCE = False

if RUN_PT_CONVERGENCE:
    chi_values = [16, 24, 32, 48, 64]
    convergence_signal = []
    trace_errors = []

    for chi in chi_values:
        pt_chi = build_process_tensor_mps(
            dt_local=dt, K=memory_depth, N=n_steps_pt,
            rel_cutoff=1e-11, max_bond=chi,
            n_cell=n_cell, n_omega=n_omega,
        )
        rho_chi = contract_process_tensor(
            pt_chi, rho_plus_x,
            operations={m1: pulse_superoperator(np.pi/2, "x")},
        )
        convergence_signal.append(expectation(rho_chi, sz).real)
        trace_errors.append(abs(np.trace(rho_chi)-1.0))

    print(" chi_max        signal             |Tr rho - 1|")
    for chi, sig, terr in zip(chi_values, convergence_signal, trace_errors):
        print(f" {chi:7d}    {sig: .10f}       {terr:.3e}")

    plt.figure(figsize=(6, 4))
    plt.plot(chi_values, convergence_signal, marker="o")
    plt.xlabel(r"maximum MPS bond $\chi_{\max}$")
    plt.ylabel("spectroscopy signal")
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(6, 4))
    plt.semilogy(chi_values, np.maximum(trace_errors, 1e-18), marker="o")
    plt.xlabel(r"maximum MPS bond $\chi_{\max}$")
    plt.ylabel(r"$|\mathrm{Tr}\rho-1|$")
    plt.tight_layout()
    plt.show()
else:
    print("Set RUN_PT_CONVERGENCE = True to run the process-tensor convergence scan.")

# %% [markdown] Notebook cell 43
# # Summary
#
# The notebook implements
#
# $$
# \boxed{\alpha_B(t)\rightarrow\eta_j\rightarrow I_j\rightarrow\text{finite-memory temporal MPS}\rightarrow T_{N:0}\rightarrow\{ A_k\}\rightarrow S}
# $$
#
```

```
# using only the native QUAPI/persistent-MPS machinery developed in the uploaded notebook.
#
# The key distinction is now explicit in code.
#
# ### Ordinary persistent-MPS QUAPI
#
# The chosen system operation is contracted while the ADT is propagated:
#
# $$
# \{I_j\}+\text{chosen evolution}\longrightarrow\rho_S(t).
# $$
#
# ### Process-tensor persistent MPS
#
# The intermediate operation $[A_k]_{q_k r_k}$ is **not** inserted during construction. Instead, $(q_k,r_k)$
    remains an open 16-dimensional physical leg:
#
# $$
# \{I_j\}+G\longrightarrow T_{N:0}.
# $$
#
# Only afterward do we choose identity maps, pulses, preparation, and measurement contractions.
#
# This is the numerical meaning of reusing the same bath-memory object for different spectroscopy experiments.
#
# > **Practical note.** This implementation is deliberately transparent rather than production optimized. A
    reusable process tensor can require substantially larger temporal bond dimensions than an ADT propagated
    for one fixed experiment because it must retain information relevant to many possible future
    interventions.
```

Listing 2: Complete persistent-MPS process-tensor spectroscopy code extracted from `process_tensor_spectroscopy.ipynb`. The listing constructs the finite-memory influence tensor, incorporates the fixed free-system propagator, leaves the intervention slots open, contracts one- and multipulse protocols, performs direct-QUAPI validation tests, and reproduces all numerical figures discussed in Sec. 17.14.

References

[1] R. P. Feynman and F. L. Vernon, Jr., "The theory of a general quantum system interacting with a linear dissipative system," *Annals of Physics* **24**, 118–173 (1963). doi:10.1016/0003-4916(63)90068-X.

[2] N. Makri and D. E. Makarov, "Tensor propagator for iterative quantum time evolution of reduced density matrices. I. Theory," *Journal of Chemical Physics* **102**, 4600–4610 (1995). doi:10.1063/1.469508.

[3] N. Makri and D. E. Makarov, "Tensor propagator for iterative quantum time evolution of reduced density matrices. II. Numerical methodology," *Journal of Chemical Physics* **102**, 4611–4618 (1995). doi:10.1063/1.469509.

[4] A. Strathearn, P. Kirton, D. Kilda, J. Keeling, and B. W. Lovett, "Efficient non-Markovian quantum dynamics using time-evolving matrix product operators," *Nature Communications* **9**, 3322 (2018). doi:10.1038/s41467-018-05617-3.

[5] U. Weiss, *Quantum Dissipative Systems*, 4th ed. (World Scientific, Singapore, 2012). doi:10.1142/8334.

[6] 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). doi:10.1103/PhysRevLett.123.240602.

[7] J. Keeling, E. M. Stoudenmire, M.-C. Bañuls, and D. R. Reichman, "Process tensor approaches to non-Markovian quantum dynamics," *Phys. Rev. X* **16**, 020502 (2026). doi:10.1103/1ncg-11hz.