Computer Physics Communications 327 (2026) 110256

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Computer Programs in Physics

H-NESSi: The hierarchical non-equilibrium systems simulation package

Thomas Blommel [a,b,c,*], Jeremija Kovačević [d], Jason Kaye [e,f], Emanuel Gull [a,g], Jakša Vučičević [d], Denis Golež [h,i]

[a] Department of Physics, University of Michigan, Ann Arbor, Michigan 48109, USA
[b] Department of Chemistry, University of California, Santa Barbara, California, USA
[c] Materials Department, University of California, Santa Barbara, California, USA
[d] Scientific Computing Laboratory, Center for the Study of Complex Systems, Institute of Physics Belgrade, University of Belgrade, Pregrevica 118, 11080 Belgrade, Serbia
[e] Center for Computational Quantum Physics, Flatiron Institute, 162 5th Avenue, New York, New York 10010, USA
[f] Center for Computational Mathematics, Flatiron Institute, 162 5th Avenue, New York, NY 10010, USA
[g] Institute of Theoretical Physics, Faculty of Physics, University of Warsaw, Warsaw, Poland
[h] Jožef Stefan Institute, SI-1000 Ljubljana, Slovenia
[i] Faculty of Mathematics and Physics, University of Ljubljana, 1000 Ljubljana, Slovenia

ARTICLE INFO

The review of this paper was arranged by Prof. Blum Volker

Keywords:
Numerical simulations
Nonequilibrium dynamics of quantum many-body problems
Keldysh formalism
Kadanoff-Baym equations
Hierarchical low-rank compression

ABSTRACT

We present H-NESSi (The Hierarchical Non-Equilibrium Systems Simulation package), an open-source software package for solving the Kadanoff-Baym equations (KBE) of nonequilibrium Green's function (NEGF) theory using hierarchical low-rank compression techniques. The simulation of correlated quantum systems out of equilibrium is severely limited by the cubic scaling in propagation time and quadratic memory growth associated with conventional two-time formulations. H-NESSi overcomes these limitations by combining high-order time-stepping schemes with hierarchical off-diagonal low-rank (HODLR) representations of the retarded and lesser Green's functions, enabling controllable accuracy at substantially reduced computational cost and memory usage. Imaginary-time quantities are efficiently represented using the discrete Lehmann representation (DLR), enabling compact and accurate treatment of thermal initial states. The implementation supports multiorbital systems, adaptive singular value truncation, and both shared-memory (OpenMP) and distributed-memory (MPI) parallelization strategies suitable for large-scale lattice calculations. The workflow closely mirrors established NEGF frameworks while introducing compression transparently into the propagation procedure. Benchmark applications to driven superconductors within dynamical mean-field theory and to the two-dimensional Hubbard model demonstrate favorable scaling compared to conventional implementations, with asymptotic time complexity significantly below the cubic scaling of uncompressed approaches. H-NESSi thus enables long-time and large-system nonequilibrium simulations of correlated quantum materials, which were previously computationally prohibitive.

PROGRAM SUMMARY

Program Title: H-NESSi
CPC Library link to program files: https://doi.org/10.17632/9hf5ryb8yv.1
Licensing provisions: MIT
Programming language: C++
External routines/libraries: cmake, eigen3, hdf5, fftw, libdlr, mpi, openmp
Nature of problem: Solves equations of motion of time-dependent Green's functions on the Konstantinov-Perel' contour within a compressed representation
Solution method: High-order timestepping method for integro-differential equations within the compressed representation on the Konstantinov-Perel' contour.

* Corresponding author.
E-mail address: tblommel@ucsb.edu (T. Blommel).

https://doi.org/10.1016/j.cpc.2026.110256
Received 7 April 2026; Received in revised form 28 May 2026; Accepted 4 June 2026
Available online 12 June 2026
0010-4655/© 2026 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

Table 2
get, set, and map functions for the retarded and lesser Keldysh components in the red and green regions of Fig. 2. These functions will fail if used outside of the red or green regions of Fig. 2.

function	description
`get_les_curr` `set_les_curr`	Copy $N_o \times N_o$ lesser data $C^<_{ij}(t,t')$ at given (t,t') to/from provided `Eigen` matrix.
`map_les_curr`	Return $N_o \times N_o$ `Eigen` matrix map of $C^<_{ij}(t,t')$ at given (t,t')
`get_ret_curr` `set_ret_curr`	Copy $N_o \times N_o$ retarded data $C^R_{ij}(t,t')$ at given (t,t') to/from provided `Eigen` matrix.
`map_ret_curr`	Return $N_o \times N_o$ `Eigen` matrix map of $C^R_{ij}(t,t')$ at given (t,t')

Table 3
General get functions for retarded and lesser components that are valid for any (t,t') point up to the current timestep. For a point in the blue region of Fig. 2, this involves reconstructing the value from the TSVD representation. Points in the yellow, green, and red regions simply involve copying data.

function	description
`get_les`	Evaluate compressed representation of $C^<_{ij}(t,t')$ at given (t,t') into provided `Eigen` matrix.
`get_ret`	Evaluate compressed representation of $C^R_{ij}(t,t')$ at given (t,t') into provided `Eigen` matrix.

-compressed format. The retarded and lesser components are stored in the HODLR-compressed representation; see Fig. 2. A core responsibility of the `herm_matrix_hodlr` class is managing the HODLR representation and updating the TSVD of each block as new rows are computed. This is done by calling `update_blocks` once per timestep.

Because of the high-order integration routines, the compressed representation lags the timestepping by $k+1$ steps, where k is the integration order. At timestep T, the data $G^R(T-k \le t \le T, 0 \le t' \le t)$ and $G^<(0 \le t \le t', T-k \le t' \le T)$—corresponding to the red and green regions in Fig. 2—has been computed but not yet compressed. These regions are stored directly, allowing efficient access and modification via the functions in Table 2, which provide `Eigen` maps or copy data to/from `Eigen` matrices. These functions will fail if used to access data outside the red and green regions.

The blue and yellow regions of Fig. 2 correspond to data stored in TSVDs and directly stored triangles, respectively; these are read-only and must be accessed via the functions in Table 3, valid for any (t,t') up to the current timestep T. For points in the yellow, green, and red regions, these functions simply copy directly stored data. Accessing data in the TSVD (blue) region requires reconstruction of the function values via the TSVD, which requires $\mathcal{O}(N_S^\varepsilon)$ operations. If used often, the cost of this reconstruction procedure can become significant compared to cases in which data is only accessed while in the directly stored regions. Solving the KBE only requires the user to modify the self-energy at the current timestep (red region), meaning that for almost all cases, reconstruction of Green's function and self-energy values from the TSVD can be avoided.

As discussed in Section 2.5, imaginary-time functions are stored on the sparse DLR grid $\tau_i \in (0,\beta)$. The functions in Table 4 allow retrieval and manipulation of this data. It is often necessary, however, to evaluate the Matsubara and mixed components at points off the DLR grid—for example, to compute the equilibrium density matrix $-G^M(\beta^-)$ (the DLR grid does not include endpoints). The functions in Table 5 evaluate imaginary-time functions at arbitrary $0 \le \tau \le \beta$ using the DLR expansion. One can also obtain imaginary-time quantities on the "reversed" DLR grid $\tau_i \to \beta - \tau_i$, which is needed for self-energy diagrams involving quantities such as $G^M(-\tau) = \xi G^M(\beta - \tau)$.

Table 4
Functions to access imaginary time functions at DLR nodes.

function	description
`get_mat` `set_mat`	Copy $N_o \times N_o$ Matsubara data $C^M_{ij}(\tau)$ at given DLR node τ_i to/from provided `Eigen` matrix.
`map_mat`	Return $N_o \times N_o$ `Eigen` matrix map of $C^M_{ij}(\tau)$ at given DLR node τ_i
`get_tv` `set_tv`	Copy $N_o \times N_o$ mixed data $C^\rceil_{ij}(t,\tau)$ at given DLR node τ_i and timestep t to/from provided `Eigen` matrix.
`map_tv`	Return $N_o \times N_o$ `Eigen` matrix map of $C^\rceil_{ij}(t,\tau)$ at given DLR node τ_i and timestep t

Table 5
Functions to evaluate DLR representations at points not on DLR nodes.

function	description
`get_mat_tau`	Evaluate DLR expansion of $C^M_{ij}(\tau)$ for arbitrary $0 \le \tau \le \beta$ into provided `Eigen` matrix.
`get_tv_tau`	Evaluate DLR expansion of $C^\rceil_{ij}(t,\tau)$ for arbitrary $0 \le \tau \le \beta$ and given timestep into provided `Eigen` matrix.
`get_mat_reversed`	Evaluate $C^M_{ij}(\beta - \tau_l)$ into provided $N_\tau \times N_o^2$ `Eigen` matrix, where τ_l are DLR nodes
`get_tv_reversed`	Evaluate $C^\rceil_{ij}(t, \beta - \tau_l)$ into provided $N_\tau \times N_o^2$ `Eigen` matrix, where τ_l are DLR nodes

Table 6
Matsubara functions in the `dyson` class.

function	description
`dyson_mat`	Perform a Matsubara iteration.

3.2. *The* `dyson` *class*

The `dyson` class solves the KBE in three steps: (1) solving the Matsubara problem, (2) bootstrapping, and (3) timestepping. In this section, we will discuss the functions provided that solve the KBE for the respective steps. In all three cases, these functions will return the norm $|G^{(i)} - G^{(i-1)}|$ between successive self-consistent iterations, which can be used to terminate the loop once the desired accuracy is reached.

First, the Matsubara problem is solved to obtain the initial thermal state. This step is optional: systems may alternatively be initialized via adiabatic switching on the two-legged contour or with an uncorrelated density matrix, in which case the user begins directly at bootstrapping.

The bootstrap process initializes the k^{th}-order accurate multistep timestepper by handling the first k timesteps. The first k timesteps are solved simultaneously, meaning that at each iteration the user must update h^{MF} and Σ for all $0 \le t \le k$ before calling one of the bootstrap functions listed in Table 7. To initialize the self-consistency loop, we must provide an initial guess for G. This can be done using the `green_from_H` function, which evaluates the mean-field real-time Green's function for the first k timesteps.

If the system is driven out of equilibrium in the first k timesteps, for example, by an interaction quench or an electric field pulse, the user should use the non-time-translation invariant (ntti) solver `dyson_start_ntti`, which makes no assumptions regarding the dynamics of the system, and directly solves the KBE. This function implements the same bootstrap procedure as laid out in Ref. [17]. If the system remains in the translationally invariant state relative to the thermal preparation for at least the first k timesteps, we provide the function `dyson_start`, which only solves the mixed component equation and uses time-translation invariance to obtain G^R and $G^<$. Lastly, the function `dyson_start_2leg` is for cases when the system is not prepared in

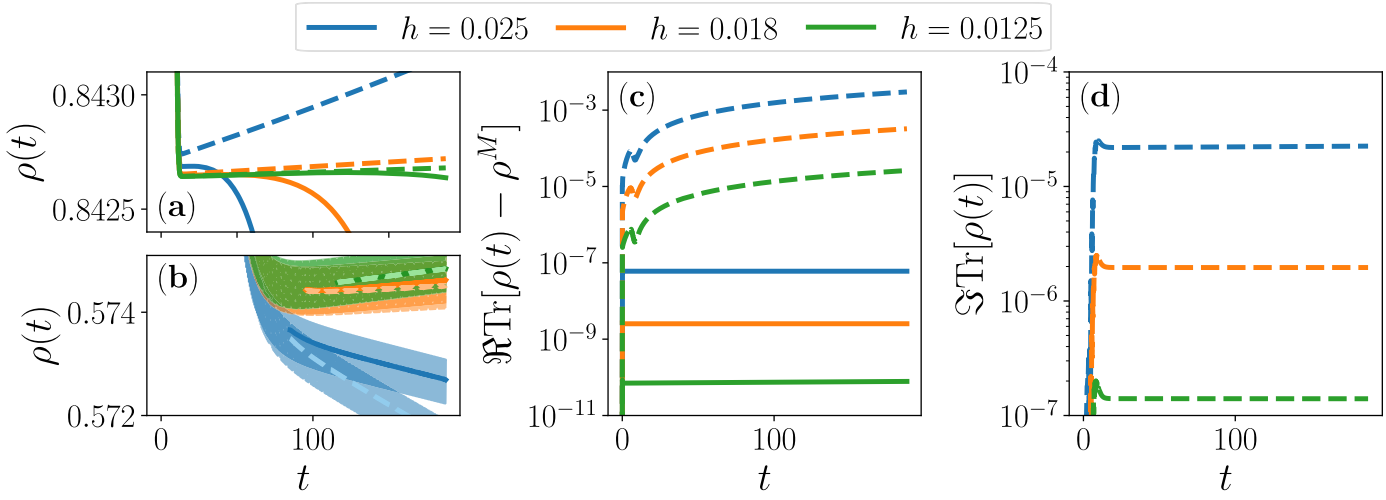

Fig. 3. Comparison of the dynamics for a diagonal element in the density matrix for `RHO_DIAGONAL` (solid lines) and `RHO_HORIZONTAL` (dashed lines) after (a) a short pulse excitation and (b) continuous periodic driving. The solid dark lines and light dashed lines overlaid in (b) are rolling averages meant to guide the eye and demonstrate convergence, as the underlying data is highly oscillatory. Real (c) and imaginary (d) parts of the density matrix trace for continuous driving. All calculations use $\epsilon_{SVD} = 10^{-10}$.

Table 7
Bootstrap functions in the `dyson` class.

function	description
`dyson_start`	Perform a Bootstrap iteration. Σ and h^{MF} must be TTI for first k timesteps.
`dyson_start_ntti`	Perform a Bootstrap iteration. No requirements on Σ or h^{MF}.
`dyson_start_2leg`	Perform a Bootstrap iteration. Solve without $\rceil$ or M component.

Table 8
Timestepping functions in the `dyson` class.

function	description
`dyson_timestep`	Perform a timestep iteration. No requirements on Σ or h^{MF}.
`dyson_timestep_2leg`	Perform a timestep iteration. Solve without $\rceil$ or M component.

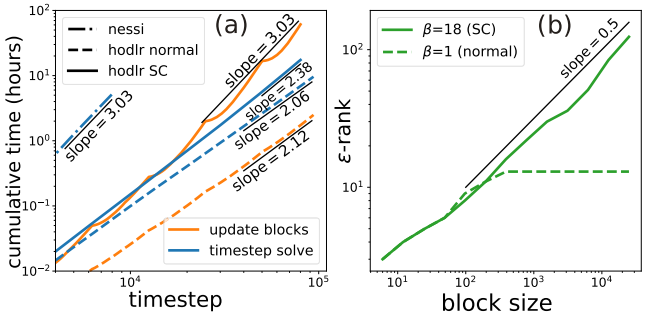

Fig. 4. (a) Wall clock time for superconducting (solid lines) and disordered (dashed lines) attractive Hubbard model with asymptotic scaling fits (black lines). Blue and orange lines show the contribution of solving timesteps and updating SVD blocks, respectively. The timing of the NESSi solver [17] (dot-dashed line) is shown for comparison. (b) Scaling of the ϵ-ranks versus the block size in the superconducting state at $\beta = 18$ (SC) and the disordered state at $\beta = 1$ (normal). Hamiltonian parameters are $U = 2$, $E_0 = 0.02$, $\sigma = 6.5$, $t_c = 16$, $\omega = 1.3$. H-NESSi data obtained using $\epsilon_{SVD} = 10^{-6}$.

an initial thermal state and therefore has no imaginary-time leg. In such cases, only the retarded and lesser components are solved.

After the bootstrap is complete, the system is integrated using the timestepping functions. Before each timestep, an initial guess for the self-consistency loop may be obtained by extrapolating from the previous k timesteps using the `extrapolate` function. After this, the timestep may be solved self-consistently using one of the two timestep functions shown in Table 8.

The density matrix, $\rho(t) = \xi i G^<(t,t)$, describes the particle distribution, and enters the KBE via the mean-field Hamiltonian, making its accurate propagation essential. When constructing the dyson class, one of two arguments must be provided: either `RHO_HORIZONTAL` or `RHO_DIAGONAL`. The former provides no special treatment of the density matrix and simply integrates the lesser Green's function as laid out in Ref. [17]. The latter treats the density matrix in a special manner by integrating the equation

$$\partial_t \rho(t)|_{t=T} = \xi i \partial_t G^<(t,T) + \xi i \partial_t G^<(T,t)$$

along the $t = t'$ diagonal, which incurs no extra computational expense. While both schemes solve the KBE, the diagonal integration provided by `RHO_DIAGONAL` often results in more strictly preserved conservation laws.

To compare the two schemes, we plot population dynamics for an initially occupied site in a simple two-band, two-site Hubbard model in Fig. 3. The state is initially fully occupied before a driving field is turned on at $t = 10$ to excite a portion of the population into the conduction band. We consider two variations of this setup: a short pulse excitation where the field is quickly turned off, Fig. 3a, and continuous periodic driving leading to sustained population oscillations, Fig. 3b. In both instances, the two methods converge to the same physical result as the timestep size h is reduced. In the continuous driving case, this convergence is demonstrated by the rolling averages for $h = 0.0125$, for which the diagonal method (solid dark line) and the horizontal method (dashed light line) perfectly overlap.

Advantages of the diagonal integration are shown in (c), where we demonstrate that the diagonal integration very accurately preserves the total number of particles in the system, whereas the horizontal integration is worse by several orders of magnitude and has a continually compounding loss of particle number. Furthermore, (d) illustrates that the horizontal integration method causes a non-zero imaginary

component in the density matrix, which in turn leads to a small non-Hermitian part of the mean-field Hamiltonian. While one could manually force the mean-field Hamiltonian to be Hermitian at each timestep, this ad-hoc intervention only corrects the time-evolution operator; it does not fix the underlying non-Hermiticity of the density matrix itself, meaning physical observables will still be corrupted. The diagonal method naturally avoids this flaw entirely.

A notable caveat to and failure of the diagonal method occurs in steady-state regimes; in situations where the system remains in a steady-state for long periods of time, as in panel (a), the diagonal integration may fail and rapidly diverge. Therefore, in simulation regimes where the system is expected to remain in a steady state for long times, the diagonal method should be avoided, and `RHO_HORIZONTAL`, with a manual Hermitian correction to the mean-field Hamiltonian, should be used instead.

3.3. Self-energy helper functions

To aid users in implementing their own self-energy evaluation routines, we have included several functions that aid in evaluating scalar products of two-time Keldysh functions. These products often appear when evaluating self-energy diagrams, especially in the context of polarization bubbles. The function `Bubble1` evaluates the Keldysh components of the product $C(z,z') = iA(z,z')B(z',z)$ Applying the standard Langreth rules alongside our established Keldysh-to-Matsubara mapping convention, this function explicitly computes the following quantities:

$$
\begin{aligned}
C^<(t,t') &= iA^<(t,t')B^>(t',t)\\
C^R(t,t') &= i\left[A^R(t,t')B^<(t',t) + A^<(t,t')B^A(t',t)\right]\\
C^\rceil(t,\tau) &= iA^\rceil(t,\tau)B^\lceil(\tau,t)\\
C^M(\tau) &= -A^M(\tau)B^M(-\tau).
\end{aligned}
$$

The function `Bubble2` evaluates the product $C(z,z') = iA(z,z')B(z,z')$

$$
\begin{aligned}
C^<(t,t') &= iA^<(t,t')B^<(t,t')\\
C^R(t,t') &= i\left[A^<(t,t')B^R(t,t') + A^R(t,t')B^<(t,t')\right.\\
&\qquad \left. + A^R(t,t')B^R(t,t')\right]\\
C^\rceil(t,\tau) &= iA^\rceil(t,\tau)B^\rceil(t,\tau)\\
C^M(\tau) &= -A^M(\tau)B^M(\tau).
\end{aligned}
$$

4. Example program: Driven superconductor

Here we provide an example implementation that showcases how to use `H-NESSi` to integrate the KBE. We solve a driven superconducting system modeled using the attractive Hubbard model, which is solved self-consistently using the DMFT approximation and a second-Born self-energy. We provide a brief overview of the problem setup; a more in-depth discussion is available in Ref. [71].

The attractive Hubbard Hamiltonian is given by

$$
H = -t_0 \sum_{\langle j,k\rangle\sigma} e^{iqA(t)} d^\dagger_{j\sigma} d_{k\sigma} - U\sum_j (n_{j\uparrow} - 1/2)(n_{j\downarrow} - 1/2), \tag{9}
$$

where $d_{i\sigma}$ is the annihilation operator at site i and spin σ, $n_{i\sigma}$ is the spin-dependent density operator and q is the charge. To describe the dynamics in the superconducting state, it is convenient to rewrite the problem using Nambu spinors

$$
\psi_i = \begin{pmatrix} d_{i\uparrow} \\ d^\dagger_{i\downarrow} \end{pmatrix}. \tag{10}
$$

The Hamiltonian now takes the form

$$
\begin{aligned}
H(t) = &-t_0 \sum_{\langle j,k\rangle\alpha} \alpha e^{iqA(t)\alpha} \psi^\dagger_{j\alpha}\psi_{k\alpha} \\
&+ U(t)\sum_i \psi^\dagger_{i+}\psi_{i+}\psi^\dagger_{i-}\psi_{i-} - \frac{U(t)}{2}\sum_{i\alpha}\psi^\dagger_{i\alpha}\psi_{i\alpha},
\end{aligned} \tag{11}
$$

where $\alpha \in \{-1, 1\}$ runs through the first and second components of the Nambu spinor. We solve the KBE for the normal and anomalous Green's functions

$$
G(t,t') = \langle \psi(t)\psi^\dagger(t')\rangle = \begin{pmatrix} \langle d_\uparrow d^\dagger_\uparrow\rangle(t,t') & \langle d_\uparrow d_\downarrow\rangle(t,t') \\ \langle d^\dagger_\downarrow d^\dagger_\uparrow\rangle(t,t') & \langle d^\dagger_\downarrow d_\downarrow\rangle(t,t') \end{pmatrix}. \tag{12}
$$

The system is then mapped to a single site using DMFT on a Bethe lattice, which introduces the hybridization function for the driven situation [85],

$$
\begin{aligned}
\Delta(t,t') &= \Delta_R(t,t') + \Delta_L(t,t')\\
\Delta_R(t,t') &= \frac{1}{2}\bar{t}_0(t)\sigma_z G(t,t')\sigma_z \bar{t}^*_0(t')\\
\Delta_L(t,t') &= \frac{1}{2}\bar{t}^*_0(t)\sigma_z G(t,t')\sigma_z \bar{t}_0(t'),
\end{aligned} \tag{13}
$$

with hopping matrix elements given by

$$
\bar{t}_0(t) = \begin{pmatrix} e^{iA(t)} & 0 \\ 0 & e^{-iA(t)} \end{pmatrix}, \tag{14}
$$

$$
A(t) = -\int_0^t \mathcal{E}(s)\,ds, \tag{15}
$$

due to the Peierls substitution. Here $\mathcal{E}$ is the driving electric field $\mathcal{E}(t) = \mathcal{E}_0 \exp\left(-\frac{(t-t_c)^2}{\sigma^2}\right)\sin(\omega(t-t_c))$, with pulse center t_c, pulse width σ, driving frequency ω, and amplitude $\mathcal{E}_0$. The hybridization function enters the KBE in the same manner as the self-energy, so the KBE in Eqs. (5)–(8) are solved with $\Sigma \to \Sigma + \Delta$. This introduces no algorithmic complications: in the example below, we simply define `DeltaPlusSigma` and compress the sum as a single object. The impurity problem is solved using the fully self-consistent second Born approximation. In this approximation, the Fock term is given by

$$
\Sigma^F_{ij}(t) = -U(t)\rho_{ij}(t)\delta_{i\bar{j}}, \tag{16}
$$

where $\bar{i} = 1 - i$. We work at half-filling, so that the Hartree term cancels the chemical potential. For the first several iterations of the self-consistency loop in equilibrium, we apply a small source field $h^B_{ij} = \eta\delta_{i\bar{j}}$ to break symmetry and possibly induce a phase transition into the superconducting state. The time-dependent bubble and exchange diagrams are given by

$$
\begin{aligned}
\Sigma^{B,\gtrless}_{ij}(t,t') &= U(t)U(t')G^{\gtrless}_{ij}(t,t')G^{\gtrless}_{\bar{i}\bar{j}}(t,t')G^{\lessgtr}_{\bar{j}i}(t',t),\\
\Sigma^{E,\gtrless}_{ij}(t,t') &= -U(t)U(t')G^{\gtrless}_{ij}(t,t')G^{\gtrless}_{i\bar{j}}(t,t')G^{\lessgtr}_{\bar{j}i}(t',t).
\end{aligned} \tag{17}
$$

Appendix B contains code implementing these expressions.

In pump-probe simulations, we can measure the induced current and the energy deposited into the system. The macroscopic current is given by

$$
j(t) = \mathrm{Im}\big(\mathrm{Tr}[\sigma_z(\Gamma_L(t) - \Gamma_R(t))]\big), \tag{18}
$$

$$
\Gamma_{R/L}(t) = -i[G * \Delta_{L/R}]^<(t,t), \tag{19}
$$

where $*$ denotes the convolution operator on the Konstantinov-Perel' contour. The energy in the system is described by the Galitskii-Migdal equation with an additional term arising from the DMFT hybridization,

$$
E^{\Delta * G} = \frac{1}{2}\mathrm{Im}\mathrm{Tr}[\Delta * G]^<(t,t). \tag{20}
$$

Furthermore, the pump pulse deposits energy into the system by inducing the current $j(t)$. The total energy deposited by the pulse can be evaluated by integrating the product of the pump electric field $\mathcal{E}(t)$ and the induced current,

$$
E^j(t) = \int_0^t d\bar{t}\mathcal{E}(\bar{t})j(\bar{t}). \tag{21}
$$

As the quantity $[A * B]^<(t,t)$ often appears in expressions for physical observables, it is implemented in the function `gamma_integral` found in the `dyson` class. We forego any analysis of these physical observables,

as they are detailed extensively in Refs. [71,82]. Instead, we mention them as examples of quantities which can be extracted from the Green's function and self-energy obtained from H-NESSi.

4.1. Initialization of objects

To begin the program, we define the scalar parameters for the calculation. The number of timesteps nt and the number of HODLR levels nlvl define the geometry of the hierarchical decomposition, while svdtol is the truncation tolerance of the TSVD of each block. The parameters h and k are the timestep size and integration order, respectively. For systems initialized in thermal equilibrium, we must provide the inverse temperature beta, as well as the DLR parameters epsdlr, and lambda discussed in Section 2.5. The self-consistent iteration is controlled by the MaxIter and MaxErr parameters, which set the maximum number of iterations and the tolerance at which solutions are considered converged. Lastly, the matrix dimension of the Green's function and self-energy is set by nao, and the particle statistics (+1 for bosons, −1 for fermions) is set by xi.

```
// Hodlr parameters
int nlvl, nt, k;
double svdtol, h;
// Matsubara parameters
double epsdlr, lambda, beta;
int ntau;
// Self-Consistency Parameters
int BootMaxIter, StepMaxIter, MatsMaxIter;
double BootMaxErr, StepMaxErr, MatsMaxErr;
// Number of orbitals and particle statistics
int nao, xi;
```

These parameters can be read from an input file, which is provided as a command-line argument:

```
char* flin;
flin=argv[1];
find_param(flin,"__nlvl=",nlvl);
```

We then initialize the required objects using classes defined in H-NESSi, namely the DLR, integration weights, Dyson solver, Green's function, self-energy, self-energy evaluator, and quadratic Hamiltonian. Note that the dlr_info class takes the argument ntau by reference and updates it to the number of DLR nodes required to represent imaginary-time functions to the accuracy set by lambda and epsdlr. The self-energy and hybridization functions, Eqs. (13), (16), and (17), are implemented in the class SC_gf2; a demonstration of their implementation is shown in Appendix B.

```
// Dyson
hodlr::dlr_info dlr(ntau, lambda, epsdlr, beta,
nao, xi);
Integration::Integrator I(k);
int r_v = RHO_DIAGONAL;
hodlr::dyson dyson_sol(nt, nao, k, dlr, r_v);
// Green's function storage
hodlr::herm_matrix_hodlr G(nt, ntau, nlvl, svdtol,
nao, nao, xi, k);
hodlr::herm_matrix_hodlr DeltaPlusSigma(nt, ntau,
nlvl, svdtol, nao, nao, xi, k);
// Self Energy evaluator
hodlr::SC_gf2 SC_gf2_solver(dlr);
// Time dependent Hamiltonian & chem. pot.
double mu = 0;
hodlr::function Hmf(nt, nao, nao);
hodlr::function U_func(nt, 1, 1);
hodlr::function t0_func(nt, nao, nao);
```

4.2. Matsubara problem

Once all objects are initialized, we solve the Matsubara problem self-consistently. At each iteration, the Hartree-Fock self-energy, second-Born self-energy, and hybridization function are evaluated, and the Dyson equation is solved in the DLR basis [83,86]. The difference between the previous and updated G^M is returned, and iterations continue until convergence. After the Matsubara problem is solved, we must call the initGMConvTensor function of the Green's function object to calculate the imaginary-time convolution in the equation of the mixed component. Note that the value of a function object on the equilibrium branch $f(t = 0_|)$ can be accessed by setting tstp=-1.

```
// Do Matsubara iterations
double MatErr;
double eta = 0.0001;
int eta_steps = 5;
int tstp = -1;
for(int iter = 0; iter < MatsMaxIter; iter++) {
  // Reset Self-Energy
  DeltaPlusSigma.set_tstp_zero(-1);

  // Evaluate Delta via increment
  SC_gf2_solver.solve_Delta(tstp, t0_func, G,
  DeltaPlusSigma);

  // Evaluate Sigma via increment
  SC_gf2_solver.solve_Sigma(tstp, U_func, G,
  DeltaPlusSigma);

  // Evaluate Hmf
  SC_gf2_solver.solve_Sigma_Fock(tstp, U_func, G,
  Hmf);

  // For first several iters, add small field to
  // break symmetry
  if(iter < eta_steps) {
    Hmf(tstp,0,1) += eta;
    Hmf(tstp,1,0) += eta;
  }

  // Solve Dyson Equation for G Matsubara
  MatErr = dyson_sol.dyson_mat(G, mu, Hmf,
  DeltaPlusSigma);

  // Exit if iteration difference is within tolerance
  // Only exit after magnetic field is off
  if(MatErr < MatsMaxErr && iter > eta_steps) break;
}
// Matsubara is converged, we set the convolution
// tensor
G.initGMConvTensor(dlr);
```

4.3. Bootstrapping

After solving the Matsubara problem, we begin the bootstrapping portion of the integration. To obtain a starting point for the self-consistency loop, we evaluate the mean-field Green's functions as

$$G^{R,HF}(t,t') = -i\theta(t-t')\exp\left[-i\epsilon(0_|)(t-t')\right] \tag{22}$$

$$G^{<,HF}(t,t') = i\exp\left[-i\epsilon(0_|)t\right]\rho^M\exp\left[i\epsilon(0_|)t'\right] \tag{23}$$

$$G^{\rceil,HF}(t,\tau) = \xi i\exp\left[-i\epsilon(0_|)t\right]\rho^M\exp\left[\epsilon(0_|)\tau\right] \tag{24}$$

using the mean-field Matsubara Hamiltonian. At each iteration, we first evaluate the self-energy and hybridization, Eqs. (13), (16), and (17), for

$0 \le t \le k$ (with k the order of accuracy), before solving the KBE in this same region. Just as in the Matsubara case, the `dyson` integrator returns the difference between the previous and current iterations of G, which is used to monitor convergence.

```
// Self-consistency for bootstrapping
dyson_sol.green_from_H(G, mu, Hmf, h, k);
for(int iter = 0; iter < BootMaxIter; iter++) {

  // Evaluate Hmf
  for(int tstp = 0; tstp <= k; tstp++) {
    SC_gf2_solver.solve_Sigma_Fock(tstp, U_func, G,
    Hmf);
  }

  // Evaluate Self-Energy
  for(int tstp = 0; tstp <= k; tstp++) {
    DeltaPlusSigma.set_tstp_zero(tstp);
    SC_gf2_solver.solve_Delta(tstp, t0_func, G,
    DeltaPlusSigma);
    SC_gf2_solver.solve_Sigma(tstp, U_func, G,
    DeltaPlusSigma);
  }

  // Solve Dyson
  double err = dyson_sol.dyson_start_ntti(G, mu, Hmf,
  DeltaPlusSigma, I, h);

  if(err < BootMaxErr) break;
}
```

4.4. Timestepping and SVD updates

After the bootstrapping procedure has converged, we proceed to timestepping. To begin each step, we update the HODLR representations of the Green's function and self-energy. We then extrapolate the Green's function to get an initial guess for the self-consistent iteration. Within the self-consistency loop, we first evaluate the mean-field and correlated self-energies before calling the timestep function provided by the `dyson` class, which again returns the difference between G and its previous iteration.

```
for(int tstp = k+1; tstp < nt; tstp++) {
  G.update_blocks(I);
  DeltaPlusSigma.update_blocks(I);

  dyson_sol.Extrapolate(G, I);

  for(int iter = 0;  iter < StepMaxIter; iter++) {
    // HF
    SC_gf2_solver.solve_Sigma_Fock(tstp, U_func, G,
    Hmf);

    // 2B
    DeltaPlusSigma.set_tstp_zero(tstp);
    SC_gf2_solver.solve_Delta(tstp, t0_func, G,
    DeltaPlusSigma);
    SC_gf2_solver.solve_Sigma(tstp, U_func, G,
    DeltaPlusSigma);

    // Dyson
    double err = dyson_sol.dyson_timestep(tstp, G,
    mu, Hmf, DeltaPlusSigma, I, h);

    if(err < CorrectorMaxErr) break;
  }
}
```

4.5. Outputting results

Once the calculations are complete, the data may be printed to an `hdf5` file using the following functions. A Python notebook is included in the `Plotting` directory, which reads in G^R and $G^<$ printed using the `write_to_hdf5` function.

```
// Open output file
h5e::File out_file("out.hdf5", h5e::File::Overwrite
| h5e::File::ReadWrite | h5e::File::Create);
// Output for Green's function and self-energy
G.write_to_hdf5(out_file, "G");
DeltaPlusSigma.write_to_hdf5(out_file, "S");
// Output single-time functions
Hmf.write_to_hdf5(out_file, "H");
// Write timing information is profiling if enabled
dyson_sol.write_timing(out_file, "dyson");
```

4.6. Checkpoint files

Calculations that must be interrupted and restarted can make use of `hdf5` checkpoint files, which store the full simulation state. These files can initialize objects via specialized constructors, and are supported for the `herm_matrix_hodlr`, `function`, and `dyson` classes. An example of writing a checkpoint file is as follows:

```
// Open checkpoint file
h5e::File check_file("check.hdf5", h5e::File::
Overwrite | h5e::File::ReadWrite |
h5e::File::Create);
// Checkpoints for Green's function and self-energy
G.write_checkpoint_hdf5(check_file, "G");
DeltaPlusSigma.write_checkpoint_hdf5(check_file,
"S");
// Checkpoints for single-time functions
Hmf.write_checkpoint_hdf5(check_file, "H");
// Checkpoints for dyson integrator only
// necessary for timing information
dyson_sol.write_checkpoint_hdf5(check_file, "dyson");
```

A calculation can then be restarted by reading from this file:

```
h5e::File check_file("check.hdf5",
h5e::File::ReadOnly);
hodlr::herm_matrix_hodlr G(check_file, "G");
hodlr::herm_matrix_hodlr DeltaPlusSigma(check_file,
"S");
hodlr::function Hmf(check_file, "H");
hodlr::dyson dyson_sol(check_file, "dyson", dlr);
```

4.7. Timing results

To demonstrate the efficiency of the `H-NESSi` code, we plot the wall clock time required to reach a given timestep and the ε-rank for growing block sizes in Fig. 4(a) and (b), respectively. We compare three different calculations: a reference implementation in the NESSi package, a calculation from `H-NESSi` in the superconducting state, and a calculation from `H-NESSi` in the normal state. From (b), we can see that the ε-rank of the Green's function has very different behavior in the two different phases. The normal state is highly compressible, and the ranks

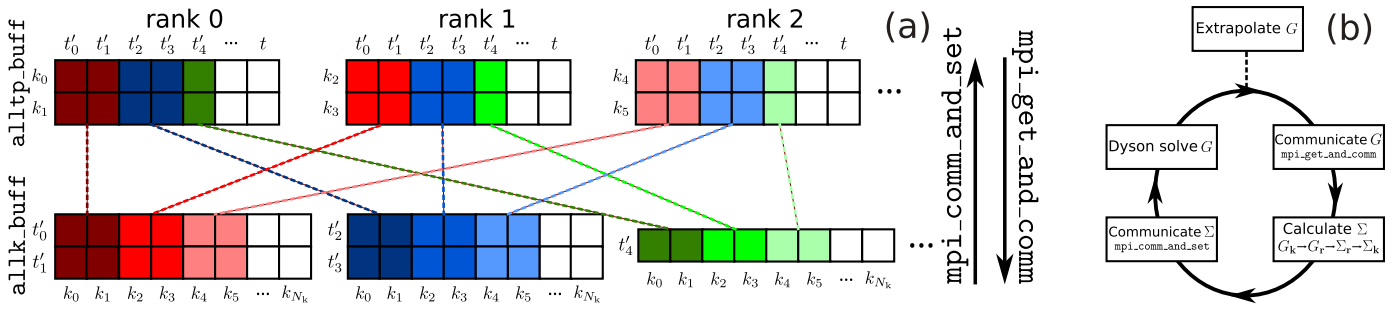

Fig. 5. (a) Schematic of the MPI communication performed by the `mpi_comm` class between the `alltp_buff` and `allk_buff` data buffers; see the main text for details. Squares of the same color represent the same data, and colored dashed lines indicate their distribution between the two buffers. White squares denote allocated buffer space for future time points or additional momentum points beyond the active block. For the specific parameters shown ($N_k = 6$, $N_{\text{mpi}} = 3$, and communication up to t'_4), the colored blocks account for the entirety of the actively communicated data, illustrating the complete data transposition. (b) Typical self-consistency loop for lattice calculations.

of the blocks saturate at small block sizes, whereas in the superconducting state, the ranks grow as $N^{1/2}$, where N is the size of the block. The behavior of the ranks affects the efficiency of our integration algorithm, as shown in (a). For the Dyson solver, both the superconducting and normal states scale approximately as $T^{2+\alpha}$, where α is the exponent characterizing the growth of the ϵ-rank with block size (e.g., $\alpha \approx 0.5$ for the superconducting state and $\alpha \approx 0$ for the normal state). This is significantly better than the T^3 scaling of the NESSi solver (dot-dashed line in (a)). Furthermore, the scaling prefactors are observed to be significantly smaller in `H-NESSi`. We also show timings for the TSVD update step in orange. The observed scaling matches the expected scaling of $T^{2+2\alpha}$ for both the normal and superconducting phases. Crucially, the Green's functions remain highly compressible across diverse physical regimes, a property consistently demonstrated in previous studies [70,71,82]. Consequently, while the exact asymptotic scaling varies between physical systems, it has proven highly advantageous in all cases we have examined.

5. MPI parallelization for generic lattice

Many systems of interest possess translational invariance, so that the Green's functions $G_{\mathbf{k},ij}$ and self-energies $\Sigma_{\mathbf{k},ij}$ are block-diagonal in momentum $\mathbf{k}$, with ij indexing non-diagonalizable orbital degrees of freedom. The Dyson equation then decouples across $\mathbf{k}$-points, enabling straightforward parallelization. However, the self-energy is typically a functional of the Green's function over the entire Brillouin zone, not just at the local $\mathbf{k}$-point. In a distributed-memory setting, inter-process communication of $G_{\mathbf{k},ij}$ is therefore required so that each process can evaluate $\Sigma_{\mathbf{k},ij}$ for its assigned $\mathbf{k}$-points.

This section is organized as follows. Section 5.1 introduces the `mpi_comm` class, which manages internode communication with minimal user effort and an interface mirroring that of `herm_matrix_hodlr`. Section 5.2 presents example calculations for the two-dimensional Hubbard model using a massively parallel implementation, comparing performance with NESSi and analyzing Green's function compression. Section 5.3 provides implementation details of our hybrid MPI + OpenMP parallelization, and Section 5.4 discusses scaling behavior.

5.1. *The* `mpi_comm` *class*

The `mpi_comm` class in `H-NESSi` manages workload distribution across MPI processes, communication of Green's function and self-energy data, and intraprocess parallelization via OpenMP threading. Upon initialization, the `mpi_comm` object exposes `my_Nk`, the number of Brillouin zone points assigned to the local process, and `my_global_k_list`, the corresponding global indices. These quantities are then used to initialize the Green's functions, self-energies, and any auxiliary containers such as a `function` object storing the $\mathbf{k}$-dependent Hamiltonian:

```
// Using SolverOrder instead of k to avoid
// clashing with k for momentum points
mpi_comm comm(global_Nk, nt, ntau, nao);
int my_Nk = comm.my_Nk;
std::vector<herm_matrix_hodlr> G_vec;
std::vector<herm_matrix_hodlr> S_vec;
std::vector<function> H_vec;
G_vec.reserve(my_Nk);
S_vec.reserve(my_Nk);
H_vec.reserve(my_Nk);
for(int k = 0; k < my_Nk; k++){
  G_vec.emplace_back(nt, ntau, nlvl, svdtol,
  nao, nao, xi, SolverOrder);
  S_vec.emplace_back(nt, ntau, nlvl, svdtol,
  nao, nao, xi, SolverOrder);
  H_vec.emplace_back(nt, nao, nao);
}
```

At each timestep, the self-consistency loop shown in Fig. 5 iterates the self-energy and Green's function until convergence. Evaluating the self-energy at a given $\mathbf{k}$-point requires knowledge of $G_{\mathbf{k}'}$ for all $\mathbf{k}'$ in the Brillouin zone. This communication is handled by two functions: `mpi_get_and_comm`, which loads a vector of the locally stored Green's functions into MPI communication buffers and distributes them to all processes, and `mpi_comm_and_set`, which communicates the evaluated self-energies back and extracts them into a vector of locally stored self-energies. For both functions, only the Matsubara component is communicated when `tstp==-1`; otherwise, the lesser, retarded, and mixed components at the current timestep are all communicated. For imaginary-time functions, the "reversed" values $G^M(\beta - \tau_i)$ and $G^{\rceil}(t, \beta - \tau_i)$ are also communicated, as they are often needed in the self-energy evaluation. Each communication function comes in two variants: a `_spawn` variant, which creates OpenMP threads at each call via `#pragma omp parallel` to accelerate buffer loading and unloading, and a `_nospawn` variant, which assumes it is called from within an existing OpenMP parallel region.

The left-hand side of Fig. 5 illustrates the communication scheme. Before communication, each rank stores all time points (and DLR nodes, although only the real-time buffers are depicted) at the current timestep for its locally assigned $\mathbf{k}$-points, in the buffer `alltp_buff`. After communication, each rank instead holds every $\mathbf{k}$-point but only a subset of time points, in the buffer `allk_buff`. In the illustrated example, at time $t = 4$, rank 1 receives $G^R_{\mathbf{k}}(t = 4, t')$ and $G^<_{\mathbf{k}}(t', t = 4)$ for $t' = 2, 3$ and every $\mathbf{k}$. Each rank is then responsible for evaluating the self-energy at its assigned subset of time points (and DLR nodes, for the mixed

Table 9
Functions for reading and writing to the communication buffers internal to `mpi_comm`.

function	description
`map_ret`	Return $N_o \times N_o$ `Eigen` matrix map of $C^R_{ij}(t,t')$ at given t' and $\mathbf{k}$ for communicated timestep t
`map_les`	Return $N_o \times N_o$ `Eigen` matrix map of $C^<_{ij}(t',t)$ at given t' and $\mathbf{k}$ for communicated timestep t
`map_tv`	Return $N_o \times N_o$ `Eigen` matrix map of $C^\rceil_{ij}(t,\tau)$ at given DLR node τ_i and $\mathbf{k}$ for communicated timestep t
`map_tv_rev`	Return $N_o \times N_o$ `Eigen` matrix map of $C^\rceil_{ij}(t,\beta-\tau)$ at given DLR node τ_i and $\mathbf{k}$ for communicated timestep t
`map_mat`	Return $N_o \times N_o$ `Eigen` matrix map of $C^M_{ij}(\tau)$ at given DLR node τ_i and $\mathbf{k}$
`map_mat_rev`	Return $N_o \times N_o$ `Eigen` matrix map of $C^M_{ij}(\beta-\tau)$ at given DLR node τ_i and $\mathbf{k}$

component) using the communicated Green's function data. The number of assigned time points is given by `my_Nt`, and the first assigned time point by `my_first_t`. The mixed and Matsubara components are split analogously, with `my_Ntau` and `my_first_tau` specifying the local imaginary-time nodes. The communicated data is accessed through an interface that returns `Eigen` matrix maps of the internal buffers; these functions, listed in Table 9, closely mirror those of the `herm_matrix_hodlr` class.

Since the `mpi_comm` class handles all communication and buffer management, the user's only responsibility is to evaluate the self-energy $\Sigma_{\mathbf{k}}$ from $G_{\mathbf{k}}$. In practice, this amounts to using the map functions in Table 9 to *replace* the communicated $G_{\mathbf{k}}$ data with the corresponding $\Sigma_{\mathbf{k}}$ data. A typical implementation has the following structure:

```
// Communicate G
comm.mpi_get_and_comm_spawn(tstp, G_vec, dlr);
// mpi_comm assigns each mpi rank a range for t'
int init_t = comm.my_first_t;
int last_t = init_t + comm.my_Nt;
// each rank must do assigned t' and all k
for(int t = init_t; t < last_t; t++) {
  for(int k = 0; k < global_Nk; k++) {
    // Read in G ret,les before overwriting
    ZMatrix GR = comm.map_ret(k,t);
    ZMatrix GL = comm.map_les(k,t);
    // Evaluate Sigma into comm buffers
    comm.map_ret(k,t) = ...
    comm.map_les(k,t) = ...
    }
}
// mpi_comm assigned range for dlr points
int init_tau = comm.my_first_tau;
int last_tau = init_tau + comm.my_Ntau;
// each rank must do assigned tau and all k
for(int tau = init_tau; tau < last_tau; tau++) {
  for(int k = 0; k < global_Nk; k++) {
    // Read in G mix before overwriting
    ZMatrix GTV = comm.map_tv(k,tau);
    // Evaluate Sigma into comm buffers
    comm.map_tv(k,tau) = ...
  }
}
// Communicate Sigma
comm.mpi_comm_and_set_spawn(tstp, S_vec);
```

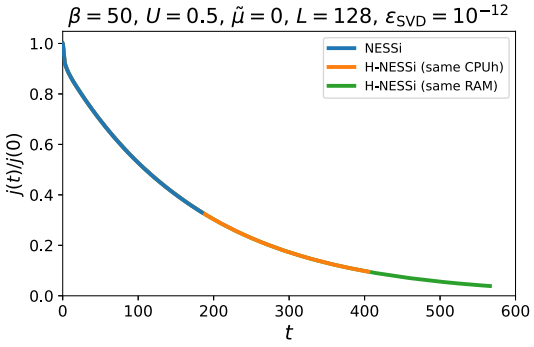

Fig. 6. Current obtained from the solution of the KB equations via Eq. (26). The NESSi calculation is limited to $N_t = 2048$ time points, whereas `H-NESSi` reaches $N_t = 6289$ time points under the same memory constraints (green curve). For the same number of CPU hours, `H-NESSi` propagates to longer physical times, as shown by the orange curve. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

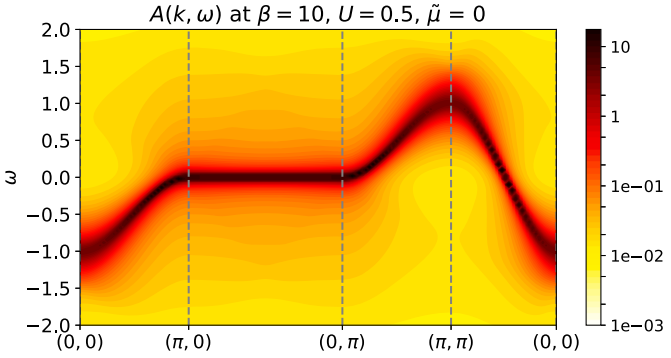

Fig. 7. Equilibrium spectral function $A(k,\omega)$ for the Hubbard Hamiltonian at half-filling, with $U = 0.5$ and $\beta = 10$.

This default interface should suffice for nearly all use cases. For situations requiring communication of additional Keldysh components or only a subset of orbital indices, a low-level interface to the underlying communication buffers is described in Appendix C.

5.2. Driven 2D hubbard example

We demonstrate the parallel implementation by computing the optical conductivity of the Hubbard model. The system is probed with a short electric field pulse, and the optical conductivity is extracted by inverting the linear-response relation. The pulse preserves translational invariance, enabling large-scale parallelization of the `Dyson` solver. Such parallelization is essential: the lattice must be large enough to suppress finite-size effects and the propagation long enough for the current response to fully decay, so that in physically relevant parameter regimes RAM usage easily exceeds 10 TB even with HODLR compression. This example is therefore well-suited to illustrate how hierarchical compression enables otherwise infeasible calculations. We require a highly parallel implementation for this example: for the largest lattices and lowest temperatures considered here, we require more than 15 thousand CPU cores across 60 nodes.

Below, we summarize the physical setup, then outline the hybrid MPI + OpenMP parallelization scheme, which combines our distributed data containers with efficient global communication and shared-memory parallelism on each MPI rank. Finally, we compare memory usage and runtimes with NESSi and study the scaling behavior of the algorithm. We explicitly define the physical observables below to demonstrate the types

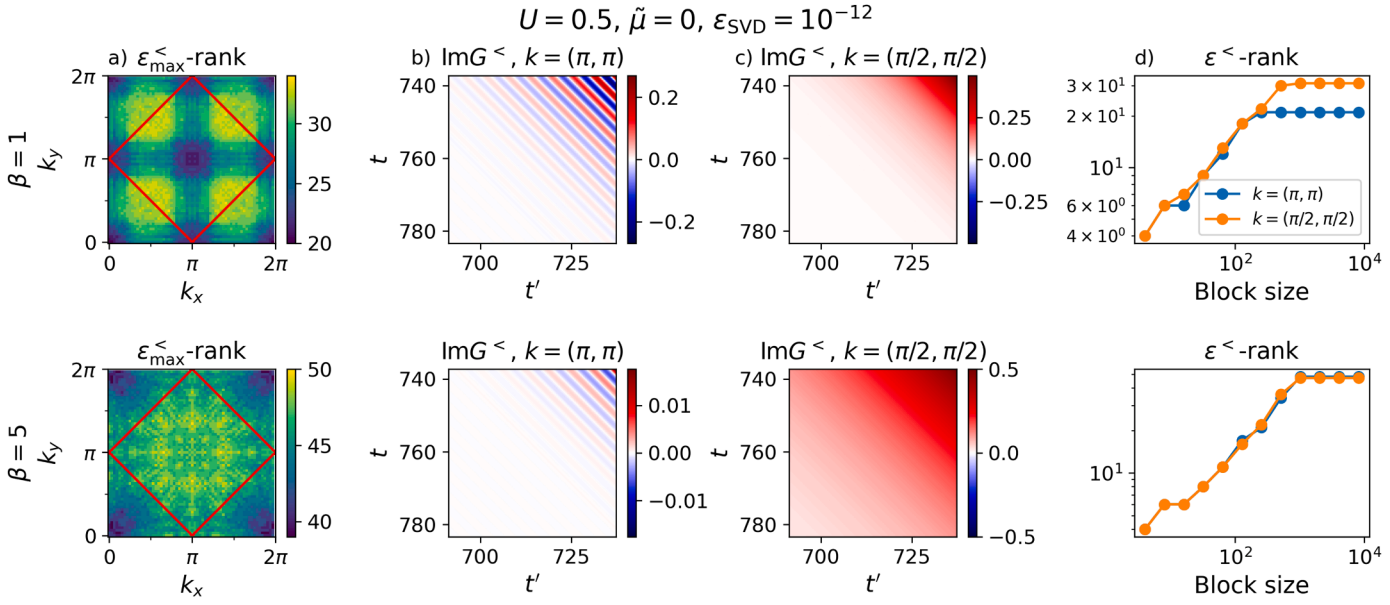

Fig. 8. (a) Maximum ε-rank across different $\mathbf{k}$-points. The red line indicates the Fermi surface. (b, c) Largest block of the imaginary part of the lesser Green's function $G^{<}$ at $\mathbf{k} = (\pi, \pi)$ and $\mathbf{k} = (\pi/2, \pi/2)$. (d) ε-rank of $G^{<}$ as a function of block size. All results are shown for two temperatures, with lattice size $L = 64$ and $N_t = 16384$ timesteps. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

of quantities that can be extracted from these calculations. The analysis presented here, however, focuses exclusively on the computational scaling and memory efficiency of the implementation. The physical insights derived from computing these specific observables are detailed extensively in Ref. [32].

We consider the square lattice Hubbard model, described by the Hamiltonian

$$H = -J \sum_{\langle ij \rangle, \sigma} c^{\dagger}_{\sigma,i} c_{\sigma,j} - \mu \sum_{\sigma,i} n_{\sigma,i} + U \sum_{i} n_{\uparrow,i} n_{\downarrow,i} \tag{25}$$

where i, j enumerate lattice sites, $c^{\dagger}$ and c are creation and annihilation operators, $\sigma \in \{\uparrow, \downarrow\}$ denotes spin, J is the nearest-neighbor hopping amplitude, $n_{\sigma,i}$ is the particle-number operator, μ is the chemical potential, and U is the on-site interaction strength. In practice, the Hartree shift is absorbed into the chemical potential, $\tilde{\mu} = \mu - U\langle n_{\sigma,i} \rangle$. The system is probed by a spatially uniform electric field in the x-direction, introduced through the vector potential $A(t)$ via the Peierls substitution $\varepsilon_{\mathbf{k}} \to \varepsilon_{\mathbf{k}-\mathbf{A}(t)}$, where $\varepsilon_{\mathbf{k}}$ is the non-interacting dispersion relation. For a given self-energy approximation, we solve the KB equations for the Green's function and compute the current, kinetic energy, and average occupancy per spin as

$$\langle j(t) \rangle = -\frac{2i}{N_{\mathrm{k}}} \sum_{\mathbf{k}} v^{x}_{\mathbf{k}-\mathbf{A}(t)} G^{<}_{\mathbf{k}}(t,t) \tag{26}$$

$$E_{\mathrm{kin}} = -2i/N_k \sum_{\mathbf{k}} \varepsilon_{\mathbf{k}} G^{<}_{\mathbf{k}}(t,t) \tag{27}$$

$$\langle n \rangle = -i/N_k \sum_{\mathbf{k}} G^{<}_{\mathbf{k}}(t,t), \tag{28}$$

where $N_{\mathrm{k}} = L^2$ is the number of $\mathbf{k}$ points, L is the linear lattice size, and $\mathbf{v} = \nabla_{\mathbf{k}} \varepsilon_{\mathbf{k}-\mathbf{A}(t)}$ is the time-dependent velocity. We choose a step-function vector potential,

$$A(t) = -A_{\mathrm{max}} \theta(t) \implies E(t) = A_{\mathrm{max}} \delta(t) \tag{29}$$

where A_{max} is the amplitude of the abrupt shift at $t = 0$ and $E(t) = -\partial_t A(t)$. The δ-function form of the electric field allows us to invert the linear-response relation,

$$\langle j(t) \rangle = \int_0^t dt' \sigma(t - t') E(t') \implies \sigma(t) = \frac{\langle j(t) \rangle}{A_{\mathrm{max}}}, \tag{30}$$

and extract the optical conductivity $\sigma(t)$ directly from the computed current. For a short pulse of the electric field in the x-direction, the Green's functions and the self-energy have the reflection symmetry

$$G_{(k_x,k_y)}(t,t') = G_{(k_x,-k_y)}(t,t'), \tag{31}$$

allowing us to consider a smaller number of $\mathbf{k}$-points, $L^2 \to L\left(\frac{L}{2} + 1\right)$.

To close the system of KB equations, we approximate the self-energy with the self-consistent second-Born approximation, which for the Hubbard Hamiltonian reads

$$\Sigma^{M}_{\mathbf{r}}(\tau) = U^2 G^{M}_{\mathbf{r}}(\tau) G^{M}_{\mathbf{r}}(\tau) G^{M}_{-\mathbf{r}}(-\tau) \tag{32}$$

$$\Sigma^{\gtrless}_{\mathbf{r}}(t,t') = U^2 G^{\gtrless}_{\mathbf{r}}(t,t') G^{\gtrless}_{\mathbf{r}}(t,t') G^{\lessgtr}_{-\mathbf{r}}(t',t) \tag{33}$$

$$\Sigma^{\lceil}_{\mathbf{r}}(t,\tau) = U^2 G^{\lceil}_{\mathbf{r}}(t,\tau) G^{\lceil}_{\mathbf{r}}(t,\tau) G^{\rceil}_{-\mathbf{r}}(\tau,t), \tag{34}$$

where $\mathbf{r}$ denotes real-space lattice sites. We note that these expressions are much more expensive to evaluate when written in the momentum basis. We therefore use `FFTW` [87] to Fourier transform $G_{\mathbf{k}} \to G_{\mathbf{r}}$ and inverse-transform $\Sigma_{\mathbf{r}} \to \Sigma_{\mathbf{k}}$. Performing these transforms requires every $\mathbf{k}$-point on each MPI process, which is precisely the communication protocol implemented in the `mpi_comm` class.

Fig. 6 shows an example current obtained with both NESSi and `H-NESSi`. Memory constraints limit the NESSi calculation to $N_{\mathrm{t}} = 2048$ time points, whereas `H-NESSi` reaches $N_{\mathrm{t}} = 6289$ time points with the same memory budget (green curve in Fig. 6). For an equal number of CPU hours, `H-NESSi` propagates to longer physical times than NESSi (orange curve). In this example, `H-NESSi` reaches a propagation time roughly three times larger than NESSi under identical memory constraints. For larger N_{t} the advantage is even more pronounced, reflecting the improved memory scaling: $\mathcal{O}(N_{\mathrm{t}}^2)$ for NESSi versus $\mathcal{O}(N_{\mathrm{t}} N_S^{\epsilon})$ for `H-NESSi`. At lower temperatures, the current decays substantially more slowly, requiring longer propagation times. Simultaneously, larger lattices are needed to suppress finite-size effects. Both factors increase memory requirements and make the calculations more demanding.

Within this setup, equilibrium single-particle correlators in the real-time domain can be obtained by simply setting the external field to zero. The self-consistent Born approximation is less expensive when formulated in real time, scaling as $\mathcal{O}(N_t)$, than in real frequency, where a straightforward implementation scales as $\mathcal{O}(N_\omega^2)$. Although the Dyson solver is typically more expensive in real time, the overall calculation

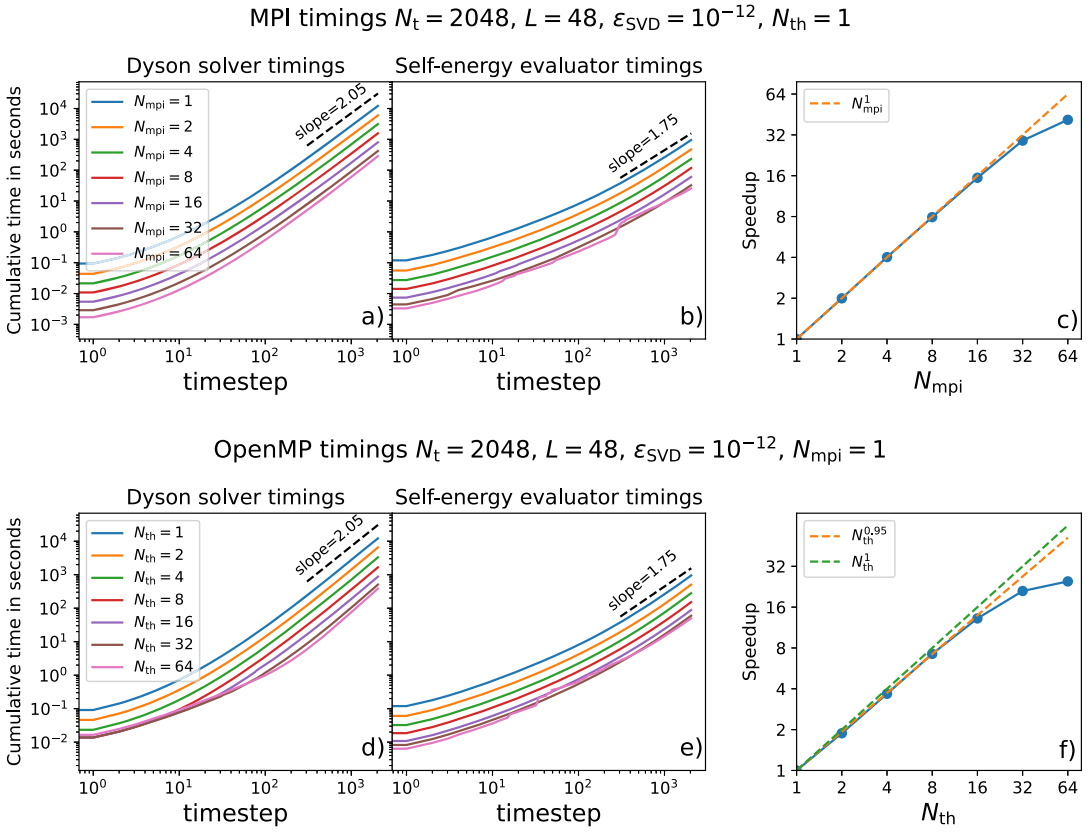

Fig. 9. Scaling of the Dyson solver (a, d) and the self-energy evaluator (b, e) for different numbers of MPI ranks and OpenMP threads, together with the relative speedup with the number of CPUs (c, f). Upper panels: scaling with MPI ranks at $N_{th} = 1$. Lower panels: scaling with OpenMP threads at $N_{mpi} = 1$. The speedup is nearly linear in both cases.

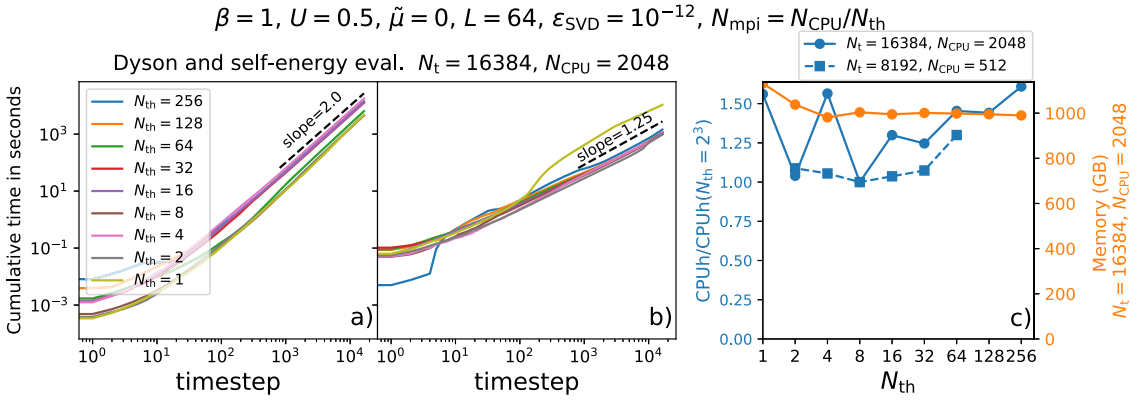

Fig. 10. Performance scaling of the hybrid MPI/OpenMP implementation for a fixed total number of CPUs ($N_{CPU} = 2048$). Cumulative computation time as a function of simulated timestep is shown for (a) the Dyson solver and (b) the self-energy evaluator across various OpenMP thread counts per MPI rank (N_{th}). (c) Computational metrics as a function of N_{th}. The left axis (blue) shows the total CPU hours normalized to the optimal case ($N_{th} = 8$) for two different simulation lengths. The right axis (orange) shows the total memory footprint for the $N_t = 16384$ calculation. For these hardware and calculation parameters, the optimal configuration that minimizes computational cost is $N_{th} = 8$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

remains inexpensive and is straightforward to set up with our code. As an example, the equilibrium spectral function is shown in Fig. 7.

Fig. 8(a) shows that the compressibility of the Green's function varies significantly across **k**-points: the least compressible points require nearly twice as many singular values as the most compressible ones. At lower temperatures, this spread appears to narrow, but the overall number of required singular values increases, reflecting the richer structure of low-temperature correlators. Despite these differences, the ϵ-rank at all **k**-points saturates at relatively small block sizes, consistent with optimal quadratic scaling of the integration routines. At block sizes of 8129, the memory savings relative to dense storage reach nearly a factor of 100 at $\beta = 5$.

5.3. Implementation details

The computation of the self-energy is made drastically cheaper by translation symmetry (even in the presence of a uniform electric field encoded in the time-dependent vector potential). As seen from Eqs. (32)–(34), the self-energy is diagonal in the position basis, unlike in the momentum representation. Moreover, the expression is time-local, enabling efficient parallelization through the communication protocol of `mpi_comm` described in Section 5.1.

The self-energy is most efficiently evaluated in real space, whereas the KB equations decouple in the **k**-space representation. We bridge the two domains as follows:

1. *Distribution of Green's functions*: `mpi_comm` distributes the Green's functions via `mpi_get_and_comm`, providing each MPI process with all **k**-points for its assigned range of time points.
2. *Real-space self-energy evaluation*: Each MPI process loops over its assigned time (and DLR) points and performs the following steps:
   - Inverse 2D Fourier transform $G_{\mathbf{k}} \to G_{\mathbf{r}}$ using `FFTW`.
   - Evaluate the self-energy in the real-space basis via Eqs. (32)–(34).
   - Forward 2D Fourier transform $\Sigma_{\mathbf{r}} \to \Sigma_{\mathbf{k}}$ using `FFTW`.

   Since these operations are independent at each time (and DLR) point, they can be parallelized over threads with OpenMP.
3. *Distribution of self-energies*: `mpi_comm` redistributes the self-energies via `mpi_comm_and_set`, returning each process's local **k**-points.

This self-energy evaluation scheme follows exactly the example implementation in Section 5.1. The most expensive step in the procedure (aside from internode communication) is the Fourier transforms, which scale as $\mathcal{O}(N_k \log N_k)$, compared to the $\mathcal{O}(N_k^2)$ cost of evaluating the self-energy directly in momentum space. This communication and evaluation process is encapsulated in the `born_approx_se` class, which also exploits the time-locality of the self-energy to parallelize over time and DLR points using OpenMP:

```
born_approx_se se_eval(U, L, global_Nk,
nao, nthreads);
```

The `lattice` object stores information about all global **k**-points in the Brillouin zone, accounting for the reflection symmetry of Eq. (31), and provides the time-dependent non-interacting dispersion relation $\varepsilon_{\mathbf{k}}(t)$:

```
lattice_2d_ysymm lattice(L, nt, h, J, Amax,
mu, nao);
```

The Dyson solver is parallelized over local **k**-points using OpenMP. Because the `dyson` class uses internal buffers, it is not thread-safe; each thread must therefore have its own solver instance. The same applies to the `dlr_info` class, whose off-node evaluation functions (Table 5) are likewise not thread-safe:

```
// dlr vector
std::vector<hodlr::dlr_info> dlr_vec;
dlr_vec.reserve(nthreads);
for(int i = 0; i < nthreads; i++) {
  dlr_vec.emplace_back
  (ntau, lambda, epsdlr, beta, nao, xi);
}
std::vector<hodlr::dyson> dyson_sol_vec;
dyson_sol_vec.reserve(nthreads);
for(int i = 0; i < nthreads; i++) {
  dyson_sol_vec.emplace_back
  (nt, nao, SolverOrder, dlr_vec[i], r_v);
}
```

Each **k**-point is represented by an instance of the `kpoint` class, which stores two `herm_matrix_hodlr` objects for the Green's function and self-energy, the non-interacting dispersion $\varepsilon_{\mathbf{k}}(t)$, and methods that perform the Dyson propagation at that **k**-point. Each MPI rank holds its assigned **k**-points in a vector `corrK_rank` of `kpoint` objects:

```
// kpoint contains Gs and Sigmas
std::vector<std::unique_ptr<kpoint>> corrK_rank;
corrK_rank.resize(my_Nk);

// create kpoints either from scratch or from
// checkpoint
if(!checkpoint_exists){
  for(int k = 0; k < my_Nk; k++){
    int global_k = comm.my_global_k_list[k];
    corrK_rank[k] = std::make_unique<kpoint>
    (nt, ntau, nlvl, svdtol, nao, beta, h,
     SolverOrder, lattice.kpoints_[global_k],
     lattice, mu, Ainitx);
  }
}
else{
  for(int k = 0; k < my_Nk; k++){
    int global_k = comm.my_global_k_list[k];
    h5e::File checkpoint_file(checkpoint_dir +
    "GSigma" + std::to_string(k) + ".h5",
    h5e::File::ReadOnly);
    corrK_rank[k] = std::make_unique<kpoint>
    (nt, ntau, nlvl, svdtol, nao, beta, h,
     SolverOrder, lattice.kpoints_[global_k],
     lattice, mu, Ainitx, checkpoint_file);
  }
}
```

Observables are computed on each MPI rank for their local **k**-blocks during timestepping and accumulated across all ranks via `MPI_Reduce` at the end of the calculation.

The Dyson integration is carried out by the `step_dyson` method of the `kpoint` class, which calls `dyson_mat`, `dyson_start_ntti`, or `dyson_timestep` for the Matsubara, bootstrapping, and timestepping stages, respectively. Since each **k**-point is independent, this loop is parallelized with OpenMP; as noted above, each thread must use its own `dyson` and `dlr_info` instances:

```
std::vector<double> errk(my_Nk);
#pragma omp parallel
{
  int thread_id = omp_get_thread_num();
  #pragma omp for schedule(static)
  for(int k = 0; k < my_Nk; k++){
    errk[k] = corrK_rank[k]->step_dyson(tstp,
    SolverOrder, lattice, I,
    dyson_sol_vec[thread_id], dlr_vec[thread_id]);
  }
}
// gather all errors from each k point
double err;
double tot_err;
err = std::reduce(errk.begin(),errk.end());
MPI_Allreduce(&err, &tot_err, 1, MPI_DOUBLE,
MPI_SUM, MPI_COMM_WORLD);
```

5.4. Performance

In this section, we benchmark the hybrid MPI + OpenMP implementation. To isolate the contributions of OpenMP and MPI, we first consider a smaller system with $L = 48$ and $N_t = 2048$, which can still be solved without parallelization. These results are shown in Fig. 9. We

then consider a large calculation with $L = 64$ and $N_t = 16384$, which requires the full hybrid implementation. These results are shown in Fig. 10 and correspond to the calculations in Fig. 8. In both cases, the integration scales as $N_t^{2.05}$, consistent with the expected scaling when the ϵ-ranks saturate at a maximal value, as confirmed in Fig. 8. This behavior closely mirrors that of the single-site calculations in Section 4.

The speedup plots in Fig. 9 show the wall time required to reach a given timestep with the Dyson solver and the self-energy evaluator, as a function of CPU count, for an example where memory requirements do not necessitate parallelization. The speedup relative to a single CPU is nearly perfectly linear up to 32 MPI processes on a single node. The OpenMP scaling is slightly less efficient, but still reaches $N^{0.95}$ up to 16 threads before saturating.

Fig. 10 shows results for a system size and propagation time that require a large number of nodes and a hybrid parallelization strategy. The efficiency depends sensitively on the total number of MPI ranks N_{mpi}, the number of threads per rank N_{th}, and their ratio. At lower temperatures, for which the Green's functions decay slowly, a large number of time points ($N_t > 10000$) is needed for the current to fully decay. Finite-size effects are also more prominent in this regime, requiring large lattice sizes ($N_k > 8000$), and memory constraints correspondingly demand more computational nodes. For a fixed total number of CPUs, the ratio $N_{\text{mpi}}/N_{\text{th}}$ can be tuned to minimize computational cost. Fig. 10(c) shows the CPU hours and memory usage for different ratios at a fixed CPU count. Memory grows slowly with N_{mpi} because each rank maintains its own copy of several auxiliary communication buffers (see Table C.2). The scalings of the Dyson solver and self-energy evaluator are shown in Fig. 10(a) and (b), respectively. The best performance is achieved with $N_{\text{th}} = 8$ threads per rank, a finding we have reproduced across two different examples and two computing clusters.

6. Conclusions

We have presented `H-NESSi`, an open-source implementation of nonequilibrium Green's function propagation based on hierarchical low-rank compression of two-time objects. By combining high-order integration schemes with HODLR representations of the retarded and lesser components, the method reduces the cubic cost scaling and quadratic memory growth that traditionally limit KBE simulations. The discrete Lehmann representation provides an efficient treatment of imaginary-time quantities and thermal initial states. The resulting framework maintains controllable accuracy while substantially reducing computational cost, extending the accessible simulation times and system sizes for nonequilibrium correlated-electron problems.

We have demonstrated these capabilities in two representative applications: a driven superconducting system treated within dynamical mean-field theory, and the two-dimensional Hubbard model solved with the second Born approximation. In both cases, the compressed propagation exhibits favorable cost and memory requirements compared to conventional approaches while preserving the dynamical features of the underlying many-body problem. The modular design supports multiorbital systems, adaptive singular-value truncation, and hybrid MPI + OpenMP parallelization, making the code suitable for large-scale simulations on modern computing architectures.

The implementation provides a foundation for further extensions, including more advanced self-energy approximations, embedding schemes, and coupling to additional degrees of freedom. By reducing the computational barrier to two-time nonequilibrium simulations, `H-NESSi` opens the door to systematic studies of long-time dynamics, strong driving protocols, and complex multiorbital materials within a controlled many-body framework.

CRediT authorship contribution statement

Thomas Blommel: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Jeremija Kovačević:** Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation; **Jason Kaye:** Writing – review & editing, Writing – original draft, Supervision, Software, Project administration, Methodology, Conceptualization; **Emanuel Gull:** Writing – review & editing, Writing – original draft, Supervision, Resources, Project administration, Funding acquisition, Conceptualization; **Jakša Vučičević:** Writing – review & editing, Writing – original draft, Supervision, Resources, Project administration, Funding acquisition, Conceptualization; **Denis Golež:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

T.B. was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Department of Energy Computational Science Graduate Fellowship under Award No. DE-SC0020347 until Aug. 2023. T.B. (after Aug 2023), and E.G. were supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research and Office of Basic Energy Sciences, Scientific Discovery through Advanced Computing (SciDAC) program under Award Number DE-SC0022088. As of June 1, 2025, E.G. was supported by the European Research Council, grant ERC-2023-AdG: 101142136 (Quantum Algorithms). The Flatiron Institute is a division of the Simons Foundation. J. V. and J. Kov. acknowledge funding provided by the Institute of Physics Belgrade, through the grant by the Ministry of Science, Technological Development and Innovation of the Republic of Serbia. J. V. and J. Kov. acknowledge funding by the European Research Council, grant ERC-2022-StG: 101076100. D.G. is supported by the Slovenian Research and Innovation Agency (ARIS) under Programs No. P1-0044, No. J1-2455, and No. MN-0016-106. The authors gratefully acknowledge the HPC RIVR consortium (www.hpc-rivr.si) and EuroHPC JU (eurohpc-ju.europa.eu) for funding this research by providing computing resources of the HPC system Vega at the Institute of Information Science (www.izum.si). Computations were also performed on the PARADOX supercomputing facility (Scientific Computing Laboratory, Center for the Study of Complex Systems, Institute of Physics Belgrade).

Data availability

Data will be made available on request.

Appendix A. Integral and derivative approximations

The numerical evaluation of derivatives, $\frac{d}{dt}y(t)|_T$, and integrals, $\int_0^T dt\, y(t)$, is split into two cases: $T \leq hk$ and $T > hk$. In the former, we use k^{th}-order polynomial differentiation and integration weights; in the latter, backward differentiation and Gregory integration [88]. To obtain the polynomial differentiation and integration weights, we begin with the polynomial interpolant of $y(t)$ that takes the values $y_j \equiv y(jh)$ on the equidistant grid:

$$\mathcal{P}[y_0, \ldots, y_k](t) = \sum_{a,l=0}^{k} h^{-a} t^a P_{al} y_l \tag{A.1}$$

$$P_{al} = (M^{-1})_{al} \text{ for } M_{ja} = j^a \tag{A.2}$$

for $0 \leq j \leq k$. The polynomial differentiation weights D_{ml} are obtained by explicitly differentiating this interpolant:

$$\left.\frac{dy}{dt}\right|_{t=mh} \approx h^{-1} \sum_{l=0}^{k} D_{ml} y_l, \text{ with} \tag{A.3}$$

$$D_{ml} = \sum_{a=1}^{k} P_{al} a m^{a-1}. \tag{A.4}$$

The polynomial integration weights I_{mnl} are similarly obtained by explicitly integrating this interpolant:

$$\int_{mh}^{nh} dt\, y(t) \approx h \sum_{l=0}^{k} I_{mnl} y_l, \text{ with} \tag{A.5}$$

$$I_{mnl} = \sum_{a=0}^{k} P_{al} \frac{n^{a+1} - m^{a+1}}{a+1}. \tag{A.6}$$

These weights are then used for $T \leq hk$ to set up the so-called bootstrapping procedure, which is discussed in Section 3.2 and implemented in the functions shown in Table 7.

For the $T > hk$ case, we use the backward differentiation weights and Gregory weights for approximating the derivatives and integrals, respectively. The backward differentiation weights a_j are obtained from the polynomial differentiation weights via

$$\left.\frac{dy}{dt}\right|_{nh} \approx h^{-1} \sum_{j=0}^{k} a_j y_{n-j}, \text{ where} \tag{A.7}$$

$$a_j = -D_{0j}. \tag{A.8}$$

Lastly, the Gregory weights ω_i are best thought of as high-order edge corrections to the Riemann sum, where the corrections occur for $k+1$ points on each end of the sum

$$\int_0^{nh} dt\, y(t) \approx h \sum_{i=0}^{k} \omega_i y_i + h \sum_{i=k+1}^{n-k-1} y_i + h \sum_{i=0}^{k} \omega_i y_{n-i}. \tag{A.9}$$

Since the integration weights differ from unity only near the endpoints, the evaluation of history integrals using the SVD representation is greatly simplified. These weights are given by the sum

$$\omega_i = \sum_{j=0}^{i} b_{k+1-i} \tag{A.10}$$

where b_i are the weights defining the $k+1$ order Adams–Moulton method [88]. These weights are then used for $T > hk$ for timestepping. The details of this procedure are described in Ref. [17].

Appendix B. Superconducting self-energy evaluation

Here we provide an example self-energy and hybridization evaluation class for the DMFT superconducting problem of Section 4. The Fock diagram evaluation (Eq. (16)) requires the density matrix, obtained from the Green's function. In general, the interaction U may vary in time, and the implementation supports this case, although U is constant in the example of Section 4. Because U is a scalar and the `function` class stores matrix-valued functions, we access $U(t)$ as `U(tstp,0,0)`.

```
void SC_gf2::solve_Sigma_Fock(int tstp, function &U, herm_matrix_hodlr &G, function &H) {
  ZMatrix rho(2,2);
  G.density_matrix(tstp, dlr_, rho);

  for(int i = 0; i < 2; i++) {
    H(tstp,i,1-i) = -U(tstp,0,0) * rho(i,1-i);
  }
}
```

Next, we implement the second-order diagrams of Eq. (17). When `tstp==-1`, only the Matsubara component is obtained; otherwise, the retarded, lesser, and mixed components are computed.

```
void SC_gf2::solve_Sigma(int tstp, function &U, herm_matrix_hodlr &G, herm_matrix_hodlr &Sigma) {
  if(tstp == -1) { Sigma_mat(U, G, Sigma); }
  else { Sigma_tstp(tstp, U, G, Sigma); }
}

void SC_gf2::Sigma_mat(function &U, herm_matrix_hodlr &G, herm_matrix_hodlr &Sigma) {
  DMatrix GM_reversed(G.ntau(), 4);
  G.get_mat_reversed(dlr_, GM_reversed);

  for(int tau = 0; tau < G.ntau(); tau++) {
    auto SM_map = Sigma.map_mat(tau);
    auto GM_map = G.map_mat(tau);
    for(int i = 0; i < 2; i++) {
      for(int j = 0; j < 2; j++) {
        // For reversed we index (i,j) using row-major ordering
        SM_map(i,j) += (U(-1,0,0) * U(-1,0,0) * GM_map(i,j) * GM_map(1-i,1-j)
                                              * GM_reversed(tau, (1-j)*2 + (1-i))).real();
        SM_map(i,j) -= (U(-1,0,0) * U(-1,0,0) * GM_map(i,1-j) * GM_map(1-i,j)
                                              * GM_reversed(tau, (1-j)*2 + (1-i))).real();
      }
    }
  }
}

void SC_gf2::Sigma_tstp(int tstp, function &U, herm_matrix_hodlr &G, herm_matrix_hodlr &Sigma) {
  for(int t = 0; t <= tstp; t++) {
    auto SR = Sigma.map_ret_curr(tstp, t);
    auto GR = G.map_ret_curr(tstp, t);
    auto SL = Sigma.map_les_curr(t, tstp);
    auto GL = G.map_les_curr(t, tstp);
    auto GG = GR - GL.adjoint();

    for(int i = 0; i < 2; i++) {
      for(int j = 0; j < 2; j++) {
        SL(i,j) += U(tstp,0,0) * U(t,0,0) * GL(i,j) * GL(1-i,1-j) * GG(1-j,1-i);
        SL(i,j) -= U(tstp,0,0) * U(t,0,0) * GL(i,1-j) * GG(1-j,1-i) * GL(1-i,j);
        // R(t,t') = >(t,t') - <(t,t') =  >(t,t') + <(t',t)^\dagger
        SR(i,j) += U(tstp,0,0) * U(t,0,0) * (GG(i,j) * GG(1-i,1-j) * GL(1-j,1-i)
                                + std::conj(GL(j,i) * GL(1-j,1-i) * GG(1-i,1-j)));
        SR(i,j) -= U(tstp,0,0) * U(t,0,0) * (GG(i,1-j) * GL(1-j,1-i) * GG(1-i,j)
                                + std::conj(GL(1-j,i) * GG(1-i,1-j) * GL(j,1-i)));
      }
    }
  }

  ZMatrix GVT(G.ntau(), 4);
```

```
  G.get_vt(tstp, dlr_, GVT);
  for(int tau = 0; tau < G.ntau(); tau++) {
    auto GTV_map = G.map_tv(tstp, tau);
    auto STV_map = Sigma.map_tv(tstp, tau);
    for(int i = 0; i < 2; i++) {
      for(int j = 0; j < 2; j++) {
        // For reversed we index (i,j) using row-major ordering
        STV_map(i,j) += U(tstp,0,0) * U(-1,0,0) * GTV_map(i,j) * GTV_map(1-i,1-j) * GVT(tau,(1-j)*2 + (1-i));
        STV_map(i,j) -= U(tstp,0,0) * U(-1,0,0) * GTV_map(i,1-j) * GVT(tau, (1-j)*2 + (1-i)) * GTV_map(1-i,j);
      }
    }
  }
}
```

Finally, we implement the hybridization function of Eq. (13). `sigma3_` denotes the third Pauli matrix.

```
void SC_gf2::solve_Delta(int tstp, function &t0, herm_matrix_hodlr &G, herm_matrix_hodlr &Sigma) {
  if(tstp == -1) { Delta_mat(t0, G, Sigma); }
  else { Delta_tstp(tstp, t0, G, Sigma); }
}

void SC_gf2::Delta_mat(function &t0, herm_matrix_hodlr &G, herm_matrix_hodlr &Sigma) {
  for(int tau = 0; tau < G.ntau(); tau++) {
    Sigma.map_mat(tau).noalias() += (t0.get_map(-1) * sigma3_ * G.map_mat(tau)
                                                    * sigma3_ * t0.get_map(-1).conjugate()).real();
  }
}

void SC_gf2::Delta_tstp(int tstp, function &t0, herm_matrix_hodlr &G, herm_matrix_hodlr &Sigma) {
  ZMatrix sGs = ZMatrix::Zero(2,2);
  for(int t = 0; t <= tstp; t++) {
    sGs.noalias() = sigma3_ * G.map_ret_curr(tstp, t) * sigma3_;
    Sigma.map_ret_curr(tstp, t).noalias() += 0.5 * t0.get_map(tstp) * sGs * t0.get_map(t).conjugate();
    Sigma.map_ret_curr(tstp, t).noalias() += 0.5 * t0.get_map(tstp).conjugate() * sGs * t0.get_map(t);

    sGs.noalias() = sigma3_ * G.map_les_curr(t, tstp) * sigma3_;
    Sigma.map_les_curr(t, tstp).noalias() += 0.5 * t0.get_map(t) * sGs * t0.get_map(tstp).conjugate();
    Sigma.map_les_curr(t, tstp).noalias() += 0.5 * t0.get_map(t).conjugate() * sGs * t0.get_map(tstp);
  }

  for(int tau = 0; tau < G.ntau(); tau++) {
    sGs.noalias() = sigma3_ * G.map_tv(tstp, tau) * sigma3_;
    Sigma.map_tv(tstp, tau).noalias() += 0.5 * t0.get_map(tstp) * sGs * t0.get_map(-1).conjugate();
    Sigma.map_tv(tstp, tau).noalias() += 0.5 * t0.get_map(tstp).conjugate() * sGs * t0.get_map(-1);
  }
}
```

Appendix C. Low-level `mpi_comm` interface

By default, `mpi_comm` handles loading, unloading, and communication of Green's functions and self-energies automatically. In cases where only a subset of orbital or Keldysh components needs to be communicated, a low-level interface allows users to implement custom protocols, as described below.

The workload distribution and communication details are managed by the `mpi_comm` class, whose main components are summarized in Tables C.1–C.3. At the start of the program, each MPI rank initializes an instance with

```
mpi_comm comm(global_Nk, nt, ntau, nao, max_component_size);
```

where `Nk` $= N_{\mathrm{k}}$ is the total number of **k**-points, `Nt` $= N_{\mathrm{t}}$ is the total number of time points, `r` is the number of DLR imaginary-time nodes used by the `dyson` class, `nao` is the number of orbitals, and `max_component_size` is the maximum number of Keldysh and/or orbital components sent per communication (default 2). During initialization, the communicator assigns blocks of **k**-points and τ points to each rank (Table C.1), whereas real-time t' points are distributed on the fly at each timestep t, since the number of τ points is fixed but the number of t' points grows with each timestep. Several buffers are also allocated for communication and self-energy evaluation (Table C.2). In the self-consistent loop, the Green's functions are extracted from the `herm_matrix_hodlr` objects into `alltp_buff` and communicated to `allk_buff` in the downward direction shown in Fig. 5; both steps are handled by `mpi_get_and_comm`. The self-energy is then evaluated inside `allk_buff` and communicated back to `alltp_buff` in the upward direction (Fig. 5), after which the results are written into the `herm_matrix_hodlr` objects; these steps are handled by `mpi_comm_and_set`. Because

Table C.1
Vectors inside `mpi_comm` describing the workload distribution among MPI ranks.

object	description
`k_rank_map`	Vector of size N_k that maps a global k index to the MPI rank responsible for that k-point.
`my_global_k_list`	Vector of size N_k^{loc} that maps local k-point indices at each MPI rank to the global k-point indices.
`Nk_per_rank`	Vector of size N_{mpi} that contains the number of k points handled by each MPI rank
`init_tau_per_rank`	Vector of size N_{mpi} that contains the initial τ index at each rank.
`Ntau_per_rank`	Vector of size N_{mpi} that contains the number of τ points handled by each MPI rank

Table C.2
Intermediate communication buffers inside `mpi_comm`.

object	description
`tau_alltp_buff`	Buffer that contains the data at all τ points and a set of **k**-points local to the MPI rank.
`t_alltp_buff`	Buffer that contains the data at all t' points and a set of **k**-points local to the MPI rank.
`tau_allk_buff`	Buffer that contains the data at all **k**-points and a set of τ points local to the MPI rank.
`t_allk_buff`	Buffer that contains the data at all **k**-points and a set of t' points local to the MPI rank.

Table C.3
Helper indexing functions inside `mpi_comm`.

function	description
`alltp_buff_index`	Computes the index in the `alltp_buff` buffers.
`t_allk_buff_index`	Computes the index in the `t_allk_buff` buffer.
`tau_allk_buff_index`	Computes the index in the `tau_allk_buff` buffer.

`MPI_Alltoallv` requires one-dimensional arrays, the indexing into the communication buffers is nontrivial; the helper functions in Table C.3 perform the necessary index calculations.

The MPI communicator is agnostic to the internal layout of the Green's functions and self-energies. The user therefore supplies lambda expressions specifying how to read and write each component; these lambdas are collected into vectors and passed to the communicator methods, which iterate over them to perform the extraction or writing. The user may define any number of lambdas to access any Keldysh component or orbital of interest. The only constraint is that `max_component_size` must be set to at least the largest number of components sent in a single communication. Therefore, if the user wants to send n Keldysh components of the Green's function in one communication, `max_component_size` must be at least n. If fewer components are sent in a subsequent call, the existing buffer space suffices; if more are sent than `max_component_size`, an error is thrown.

For example, computing the lesser self-energy from Eq. (33) requires both $G^>$ and $G^<$. The corresponding get lambdas are

```
// returns G^<(t,t') at local k index and real times t,t'
auto getLess = [&corrK_rank, &nao]
(int local_ki, int ti, int tpi)
{
  hodlr::ZMatrix tmp(nao, nao);
  corrK_rank[local_ki]->G_.get_les(ti, tpi, tmp);
  return tmp(0,0);
};

// returns G^>(t,t') at local k index and real times t,t'
auto getGreat = [&corrK_rank, &nao](int local_ki, int ti, int tpi){
  hodlr::ZMatrix tmpRet(nao, nao), tmpLess(nao, nao), tmpGreat(nao, nao);

  corrK_rank[local_ki]->G_.get_les(ti, tpi, tmpLess);
  corrK_rank[local_ki]->G_.get_ret(ti, tpi, tmpRet);
  tmpGreat(0,0) = tmpRet(0,0) + tmpLess(0,0);

  return tmpGreat(0,0);
};
```

```
// vector of lambdas for Lesser and Greater components
std::vector<std::function<std::complex<double>(int, int, int)>> getsLG = {getLess,getGreat};
```

After computing and communicating the self-energy, the communicator needs to write the results back into the self-energy objects. The following set lambdas write $\Sigma^{\mathrm{R}}_{\mathbf{k}}(t,t')$ and $\Sigma^{<}_{\mathbf{k}}(t,t')$:

```
// sets Sigma^<(t,t') at local k index and real time t for all t'<=t
auto setLess = [&corrK_rank, &nao](int k, int ti, std::vector<std::complex<double>> &Sigma){
  std::vector<std::complex<double>> Sigmac(ti+1);
  for(int j = 0; j<=ti; j++){
          Sigmac[j] = -std::conj(Sigma[j]);
  }
  hodlr::ZMatrixMap(corrK_rank[k]->Sigma_.curr_timestep_les_ptr(0,ti), (ti + 1) * nao, nao).noalias()
  = hodlr::ZMatrixMap(Sigmac.data(), (ti + 1) * nao, nao);
};

// sets Sigma^R(t,t') at local k index and real time t for all t'>=t
auto setRet = [&corrK_rank, &nao](int k, int ti, std::vector<std::complex<double>> &Sigma)
{
  hodlr::ZMatrixMap(corrK_rank[k]->Sigma_.curr_timestep_ret_ptr(ti,0), (ti + 1) * nao, nao).noalias()
  = hodlr::ZMatrixMap(Sigma.data(), (ti + 1) * nao, nao);
};

std::vector<std::function<void(int, int, std::vector<std::complex<double>>&)>> setsLR = {setLess,setRet};
```

The vectors `getsLG` and `setsLR` are passed to the communicator methods, which iterate over their elements and perform the extraction or writing in the appropriate communication direction (Fig. 5). In this example, `max_component_size` must be at least 2, since two components of each quantity are communicated.

For general complex time indices t and t' and Keldysh component index η, the get and set lambdas must satisfy the following requirements:

- A get lambda returns $G^{\eta}_{\mathbf{k}}(t,t')$ at a local $\mathbf{k}$ index, with signature `std::complex<double>(int local_ki, int ti, int tpi)`.
- A set lambda writes $\Sigma^{\eta}_{\mathbf{k}}(t,t')$ at a local $\mathbf{k}$ index for an array of $t' \leq t$ points, with signature `void(int local_ki, int ti, std::vector<std::complex<double>> &Sigma_tp)`.
- Both get and set lambdas must use the timestep t as their first time index, including for the Matsubara case where $t = -1$.

References

[1] D. Fausti, R.I. Tobey, N. Dean, S. Kaiser, A. Dienst, M.C. Hoffmann, S. Pyon, T. Takayama, H. Takagi, A. Cavalleri, Light-induced superconductivity in a stripe-ordered cuprate, Science 331 (6014) (2011) 189–191.

[2] M. Mitrano, A. Cantaluppi, D. Nicoletti, S. Kaiser, A. Perucchi, S. Lupi, P. Di Pietro, D. Pontiroli, M. Riccò, S.R. Clark, et al., Possible light-induced superconductivity in K 3 C 60 at high temperature, Nature 530 (7591) (2016) 461–464.

[3] C. Giannetti, M. Capone, D. Fausti, M. Fabrizio, F. Parmigiani, D. Mihailovic, Ultrafast optical spectroscopy of strongly correlated materials and high-temperature superconductors: a non-equilibrium approach, Adv. Phys. 65 (2) (2016) 58–238.

[4] A. de la Torre, D.M. Kennes, M. Claassen, S. Gerber, J.W. McIver, M.A. Sentef, Colloquium: nonthermal pathways to ultrafast control in quantum materials, Rev. Mod. Phys. 93 (2021) 041002. https://doi.org/10.1103/RevModPhys.93.041002

[5] F. Zhou, J. Williams, C.D. Malliakas, M.G. Kanatzidis, A.F. Kemper, C.-Y. Ruan, Nonequilibrium dynamics of spontaneous symmetry breaking into a hidden state of charge-density wave, (2019). arXiv:1904.07120

[6] Y. Murakami, D. Golež, M. Eckstein, P. Werner, Photoinduced nonequilibrium states in Mott insulators, Rev. Mod. Phys. 97 (2025) 035001. https://doi.org/10.1103/tkjh-lr83

[7] D.N. Basov, R.D. Averitt, D. Hsieh, Towards properties on demand in quantum materials, Nat. Mater. 16 (11) (2017) 1077–1088.

[8] M. Ligges, I. Avigo, D. Golež, H. Strand, Y. Beyazit, K. Hanff, F. Diekmann, L. Stojchevska, M. Kalläne, P. Zhou, et al., Ultrafast doublon dynamics in photoexcited 1 T-TaS 2, Phys. Rev. Lett. 120 (16) (2018) 166401.

[9] M.A.L. Marques, N.T. Maitra, F.M.S. Nogueira, E.K.U. Gross, A. Rubio, Fundamentals of Time-Dependent Density Functional Theory, Springer, 2012.

[10] U. Schollwöck, The density-matrix renormalization group in the age of matrix product states, Ann. Phys. 326 (1) (2011) 96–192.

[11] H. Haug, A.-P. Jauho, Quantum Kinetics in Transport and Optics of Semiconductors, Springer, 2 edition, 2008.

[12] P. Werner, T. Oka, M. Eckstein, A.J. Millis, Weak-coupling quantum Monte Carlo calculations on the Keldysh contour: theory and application to the current-voltage characteristics of the Anderson model, Phys. Rev. B 81 (2010) 035108.

[13] G. Cohen, E. Gull, D.R. Reichman, A.J. Millis, Taming the dynamical sign problem in real-time evolution of quantum many-body problems, Phys. Rev. Lett. 115 (2015) 266802.

[14] E. Gull, D.R. Reichman, A.J. Millis, Numerically exact long-time behavior of nonequilibrium quantum impurity models, Phys. Rev. B 84 (2011) 085134.

[15] L.P. Kadanoff, G.A. Baym, Quantum Statistical Mechanics, W.A. Benjamin, 1962.

[16] G. Stefanucci, R. Van Leeuwen, Nonequilibrium Many-Body Theory of Quantum Systems: A Modern Introduction, Cambridge University Press, 2013.

[17] M. Schüler, D. Golež, Y. Murakami, N. Bittner, A. Herrmann, H.U.R. Strand, P. Werner, M. Eckstein, NESSi: the non-equilibrium systems simulation package, Comput. Phys. Commun. 257 (2020) 107484.

[18] H. Aoki, N. Tsuji, M. Eckstein, M. Kollar, T. Oka, P. Werner, Nonequilibrium dynamical mean-field theory and its applications, Rev. Mod. Phys. 86 (2) (2014) 779.

[19] M. Sentef, A.F. Kemper, B. Moritz, J.K. Freericks, Z.-X. Shen, T.P. Devereaux, Examining electron-boson coupling using time-resolved spectroscopy, Phys. Rev. X 3 (2013) 041033. https://doi.org/10.1103/PhysRevX.3.041033

[20] J.K. Freericks, V.M. Turkowski, V. Zlatić, Nonequilibrium dynamical mean-field theory, Phys. Rev. Lett. 97 (2006) 266408. https://doi.org/10.1103/PhysRevLett.97.266408

[21] M. Jarrell, J.E. Gubernatis, Bayesian inference and the analytic continuation of imaginary-time quantum Monte Carlo data, Phys. Rep. 269 (3) (1996) 133–195. https://doi.org/10.1016/0370-1573(95)00074-7

[22] A.W. Sandvik, Stochastic method for analytic continuation of quantum Monte Carlo data, Phys. Rev. B 57 (1998) 10287–10290. https://doi.org/10.1103/PhysRevB.57.10287

[23] K. Yoshimi, J. Otsuki, Y. Motoyama, M. Ohzeki, H. Shinaoka, SpM: Sparse modeling tool for analytic continuation of imaginary-time Green's function, Comput. Phys. Commun. 244 (2019) 319–323. https://doi.org/10.1016/j.cpc.2019.07.001

[24] R. Fournier, L. Wang, O.V. Yazyev, Q. Wu, Artificial neural network approach to the analytic continuation problem, Phys. Rev. Lett. 124 (2020) 056401. https://doi.org/10.1103/PhysRevLett.124.056401

[25] H. Yoon, J.-H. Sim, M.J. Han, Analytic continuation via domain knowledge free machine learning, Phys. Rev. B 98 (2018) 245101. https://doi.org/10.1103/PhysRevB.98.245101

[26] L. Ying, Analytic continuation from limited noisy Matsubara data, J. Comput. Phys. 469 (2022) 111549. https://doi.org/10.1016/j.jcp.2022.111549

[27] E.W. Huang, R. Sheppard, B. Moritz, T.P. Devereaux, Strange metallicity in the doped hubbard model, Science 366 (6468) (2019) 987–990. https://doi.org/10.1126/science.aau7063

[28] J. Vučičević, J. Kokalj, R. Žitko, N. Wentzell, D. Tanasković, J. Mravlje, Conductivity in the square lattice hubbard model at high temperatures: importance of vertex corrections, Phys. Rev. Lett. 123 (2019) 036601. https://doi.org/10.1103/PhysRevLett.123.036601

[29] L. Zhang, Y. Yu, E. Gull, Minimal pole representation and analytic continuation of matrix-valued correlation functions, Phys. Rev. B 110 (2024) 235131. https://doi.org/10.1103/PhysRevB.110.235131

[30] A. Erpenbeck, E. Gull, G. Cohen, Quantum Monte Carlo method in the steady state, Phys. Rev. Lett. 130 (18) (2023) 186301. https://doi.org/10.1103/PhysRevLett.130.186301

[31] X. Dong, E. Gull, H.U.R. Strand, Excitations and spectra from equilibrium real-time Green's functions, Phys. Rev. B 106 (12) (2022) 125153. https://doi.org/10.1103/PhysRevB.106.125153

[32] J. Kovačević, M. Ferrero, J.c.v. Vučičević, Toward numerically exact computation of conductivity in the thermodynamic limit of interacting lattice models, Phys. Rev. Lett. 135 (2025) 016502. https://doi.org/10.1103/mm38-zttx

[33] E. Perfetto, D. Sangalli, A. Marini, G. Stefanucci, Ultrafast charge migration in xuv photoexcited phenylalanine: a first-principles study based on real-time nonequilibrium green's functions, J. Phys. Chem. Lett. 9 (6) (2018) 1353–1358.

[34] A. Molina-Sánchez, D. Sangalli, L. Wirtz, A. Marini, Ab initio calculations of ultrashort carrier dynamics in two-dimensional materials: valley depolarization in single-layer wse2, Nano Lett. 17 (8) (2017) 4549–4555.

[35] D. Sangalli, A. Ferretti, H. Miranda, C. Attaccalite, I. Marri, E. Cannuccia, P. Melo, M. Marsili, F. Paleari, A. Marrazzo, et al., Many-body perturbation theory calculations using the yambo code, J. Phys. Condens. Matter 31 (32) (2019) 325902.

[36] A. Marini, C. Hogan, M. Grüning, D. Varsano, Yambo: an ab initio tool for excited state calculations, Comput. Phys. Commun. 180 (8) (2009) 1392–1403.

[37] E. Perfetto, G. Stefanucci, CHEERS: a tool for correlated hole-electron evolution from real-time simulations, J. Phys. Condens. Matter 30 (46) (2018) 465901.

[38] K. Balzer, M. Bonitz, Nonequilibrium Green's Functions Approach to Inhomogeneous Systems, Springer, 2012.

[39] K. Balzer, Solving the Two-Time Kadanoff-Baym Equations: Application to Model Atoms and Molecules, Ph.D. Thesis, 2011.

[40] A. Erpenbeck, T. Blommel, L. Zhang, W.-T. Lin, G. Cohen, E. Gull, Steady-state properties of multi-orbital systems using quantum Monte Carlo, J. Chem. Phys. 161 (9) (2024) 094104. https://doi.org/10.1063/5.0226253

[41] P. Danielewicz, Quantum theory of nonequilibrium processes II. application to nuclear collisions, Ann. Phys. 152 (2) (1984) 305–326. https://doi.org/10.1016/0003-4916(84)90093-9

[42] M. Bonitz, D. Kremp, D.C. Scott, R. Binder, W.D. Kraeft, H.S. Köhler, Numerical analysis of non-Markovian effects in charge-carrier scattering: one-time versus two-time kinetic equations, J. Phys. Condens. Matter 8 (33) (1996) 6057. https://doi.org/10.1088/0953-8984/8/33/012

[43] H.S. Köhler, N.H. Kwong, H.A. Yousif, A Fortran code for solving the Kadanoff–Baym equations for a homogeneous fermion system, Comput. Phys. Commun. 123 (1) (1999) 123–142. https://doi.org/10.1016/S0010-4655(99)00260-X

[44] K. Balzer, S. Bauch, M. Bonitz, Efficient grid-based method in nonequilibrium Green's function calculations: application to model atoms and molecules, Phys. Rev. A 81 (2010) 022510.

[45] P. Lipavskỳ, V. Špička, B. Velickỳ, Generalized Kadanoff-Baym ansatz for deriving quantum transport equations, Phys. Rev. B 34 (10) (1986) 6933.

[46] A. Kalvová, B. Velickỳ, V. Špička, Beyond the generalized Kadanoff–Baym Ansatz, Phys. Status Solidi B 256 (7) (2019) 1800594.

[47] N. Schlünzen, J.-P. Joost, M. Bonitz, Achieving the scaling limit for nonequilibrium green functions simulations, Phys. Rev. Lett. 124 (2020) 076601. https://doi.org/10.1103/PhysRevLett.124.076601

[48] J.-P. Joost, N. Schlünzen, M. Bonitz, G1-G2 scheme: dramatic acceleration of nonequilibrium green functions simulations within the Hartree-Fock generalized Kadanoff-Baym ansatz, Phys. Rev. B 101 (2020) 245101. https://doi.org/10.1103/PhysRevB.101.245101

[49] R. Tuovinen, D. Golež, M. Eckstein, M.A. Sentef, Comparing the generalized Kadanoff-Baym ansatz with the full Kadanoff-Baym equations for an excitonic insulator out of equilibrium, Phys. Rev. B 102 (11) (2020) 115157.

[50] C.C. Reeves, Y. Zhu, C. Yang, V.c.v. Vlček, Unimportance of memory for the time nonlocal components of the Kadanoff-Baym equations, Phys. Rev. B 108 (2023) 115152. https://doi.org/10.1103/PhysRevB.108.115152

[51] D. Karlsson, R. van Leeuwen, Y. Pavlyukh, E. Perfetto, G. Stefanucci, Fast Green's function method for ultrafast electron-boson dynamics, Phys. Rev. Lett. 127 (2021) 036402. https://doi.org/10.1103/PhysRevLett.127.036402

[52] G. Stefanucci, E. Perfetto, Semiconductor electron-phonon equations: a rung above Boltzmann in the many-body ladder, 2023, arXiv:2311.03980 [cond-mat.mtrl-sci]

[53] J. Yin, Y.-h. Chan, F.H.d. Jornada, D.Y. Qiu, S.G. Louie, C. Yang, Using dynamic mode decomposition to predict the dynamics of a two-time non-equilibrium Green's function, J. Comput. Sci. 64 (2022) 101843. https://doi.org/10.1016/j.jocs.2022.101843

[54] J. Yin, Y.-h. Chan, F.H. da Jornada, D.Y. Qiu, C. Yang, S.G. Louie, Analyzing and predicting non-equilibrium many-body dynamics via dynamic mode decomposition, J. Comput. Phys. 477 (2023) 111909. https://doi.org/10.1016/j.jcp.2023.111909

[55] C.C. Reeves, J. Yin, Y. Zhu, K.Z. Ibrahim, C. Yang, V.c.v. Vlček, Dynamic mode decomposition for extrapolating nonequilibrium Green's-function dynamics, Phys. Rev. B 107 (2023) 075107. https://doi.org/10.1103/PhysRevB.107.075107

[56] H. Bassi, Y. Zhu, S. Liang, J. Yin, C.C. Reeves, V. Vlček, C. Yang, Learning nonlinear integral operators via recurrent neural networks and its application in solving integro-differential equations, Mach. Learn. Appl. 15 (2024) 100524.

[57] Y. Zhu, J. Yin, C.C. Reeves, C. Yang, V. Vlcek, Predicting nonequilibrium Green's function dynamics and photoemission spectra via nonlinear integral operator learning, 2024.arXiv:2407.09773 [cond-mat.str-el]

[58] A. Erpenbeck, Y. Zhu, Y. Yu, L. Zhang, R. Gerum, O. Goulko, C. Yang, G. Cohen, E. Gull, Compact representation and long-time extrapolation of real-time data for quantum systems (2025). https://doi.org/10.48550/arXiv.2506.13760

[59] G. Cohen, E. Rabani, Memory effects in nonequilibrium quantum impurity models, Phys. Rev. B 84 (7) (2011) 075150. https://doi.org/10.1103/PhysRevB.84.075150

[60] G. Cohen, E. Gull, D.R. Reichman, A.J. Millis, E. Rabani, Numerically exact long-time magnetization dynamics at the nonequilibrium Kondo crossover of the Anderson impurity model, Phys. Rev. B 87 (19) (2013) 195108. https://doi.org/10.1103/PhysRevB.87.195108

[61] M. Schüler, M. Eckstein, P. Werner, Truncating the memory time in nonequilibrium dynamical mean field theory calculations, Phys. Rev. B 97 (2018) 245129.

[62] C. Stahl, N. Dasari, J. Li, A. Picano, P. Werner, M. Eckstein, Memory truncated Kadanoff-Baym equations, Phys. Rev. B 105 (2022) 115146. https://doi.org/10.1103/PhysRevB.105.115146
[63] A. Picano, M. Eckstein, Accelerated gap collapse in a Slater antiferromagnet, Phys. Rev. B 103 (2021) 165118. https://doi.org/10.1103/PhysRevB.103.165118
[64] N. Dasari, J. Li, P. Werner, M. Eckstein, Photoinduced strange metal with electron and hole quasiparticles, Phys. Rev. B 103 (2021) L201116. https://doi.org/10.1103/PhysRevB.103.L201116
[65] H. Shinaoka, M. Wallerberger, Y. Murakami, K. Nogaki, R. Sakurai, P. Werner, A. Kauch, Multiscale space-time Ansatz for correlation functions of quantum systems based on quantics tensor trains, Phys. Rev. X 13 (2023) 021015. https://doi.org/10.1103/PhysRevX.13.021015
[66] M. Murray, H. Shinaoka, P. Werner, Nonequilibrium diagrammatic many-body simulations with quantics tensor trains, Phys. Rev. B 109 (2024) 165135. https://doi.org/10.1103/PhysRevB.109.165135
[67] M. Środa, K. Inayoshi, H. Shinaoka, P. Werner, High-resolution nonequilibrium GW calculations based on quantics tensor trains, 2024, arXiv:2412.14032 [cond-mat.str-el]
[68] K. Inayoshi, M. Środa, A. Kauch, P. Werner, H. Shinaoka, A causality-based divide-and-conquer algorithm for nonequilibrium Green's function calculations with quantics tensor trains, 2025. https://doi.org/10.48550/arXiv.2509.15028
[69] M. Środa, K. Inayoshi, M. Schüler, H. Shinaoka, P. Werner, Predictor-corrector method based on dynamic mode decomposition for tensor-train nonequilibrium Green's function calculations, 2025. https://doi.org/10.48550/arXiv.2509.22177
[70] J. Kaye, D. Golež, Low rank compression in the numerical solution of the nonequilibrium Dyson equation, SciPost Phys. 10 (2021) 091. https://doi.org/10.21468/SciPostPhys.10.4.091
[71] T. Blommel, J. Kaye, Y. Murakami, E. Gull, D. Golež, Chirped amplitude mode in photoexcited superconductors, Phys. Rev. B 111 (2025) 094502. https://doi.org/10.1103/PhysRevB.111.094502
[72] B. Lamic, Solving the transient dyson equation with quasilinear complexity via matrix compression, (2024). arXiv:2410.11057
[73] J. Gašperlin, D. Golež, J. Kaye, Stability and complexity of global iterative solvers for the Kadanoff-Baym equations, 2025. https://doi.org/10.48550/arXiv.2512.11371
[74] J. Ballani, D. Kressner, Matrices with Hierarchical Low-Rank Structures, Springer, 2016, pp. 161–209.
[75] A. Georges, G. Kotliar, W. Krauth, M.J. Rozenberg, Dynamical mean-field theory of strongly correlated fermion systems and the limit of infinite dimensions, Rev. Mod. Phys. 68 (1) (1996) 13.
[76] T. Blommel, D.J. Gardner, C.S. Woodward, E. Gull, Adaptive time stepping for the two-time integro-differential Kadanoff-Baym equations, Phys. Rev. B 110 (2024) 205134. https://doi.org/10.1103/PhysRevB.110.205134
[77] H. LaBollita, J. Lee-Hand, F.B. Kugler, L. Van Muñoz, S. Beck, A. Hampel, J. Kaye, A. Georges, C.E. Dreyer, Low-temperature transport in high-conductivity correlated metals: a density functional plus dynamical mean-field study of cubic perovskites, Phys. Rev. B 113 (2026) 085125. https://doi.org/10.1103/71c6-sb7v
[78] F.B. Kugler, J. Lee-Hand, H. LaBollita, L. Van Muñoz, J. Kaye, S. Beck, A. Hampel, A. Georges, C.E. Dreyer, Fermi-liquid T^2 resistivity: dynamical mean-field theory meets experiment, Phys. Rev. B 113 (2026) L081105. https://doi.org/10.1103/f4p8-5xb8
[79] O.V. Konstantinov, V.I. Perel', A diagram technique for evaluating transport quantities, Sov. Phys. JETP 12 (1) (1961) 142–149.
[80] Q. Dong, I. Krivenko, J. Kleinhenz, A.E. Antipov, G. Cohen, E. Gull, Quantum Monte Carlo solution of the dynamical mean field equations in real time, Phys. Rev. B 96 (15) (2017) 155126. https://doi.org/10.1103/PhysRevB.96.155126
[81] A. Picano, G. Biroli, M. Schirò, Quantum thermalization via travelling waves, Phys. Rev. Lett. 134 (2025) 116503. https://doi.org/10.1103/PhysRevLett.134.116503
[82] T. Blommel, Numerical Integration of the Kadanoff-Baym Equations, Ph.D. thesis, University of Michigan, 2024.
[83] J. Kaye, K. Chen, O. Parcollet, Discrete Lehmann representation of imaginary time Green's functions, Phys. Rev. B 105 (2022) 235115. https://doi.org/10.1103/PhysRevB.105.235115
[84] J. Kaye, K. Chen, H.U.R. Strand, libdlr: efficient imaginary time calculations using the discrete Lehmann representation, Comput. Phys. Commun. 280 (2022) 108458. https://doi.org/10.1016/j.cpc.2022.108458
[85] J. Li, D. Golež, P. Werner, M. Eckstein, Long-range η-pairing in photodoped Mott insulators, (2019). arXiv:1908.08693
[86] J. Kaye, H.U.R. Strand, A fast time domain solver for the equilibrium Dyson equation, Adv. Comput. Math. 49 (4) (2023). https://doi.org/10.1007/s10444-023-10067-7
[87] M. Frigo, S.G. Johnson, The design and implementation of FFTW3, Proc. IEEE 93 (2) (2005) 216–231. https://doi.org/10.1109/JPROC.2004.840301
[88] P.H.M. WOLKENFELT, The construction of reducible quadrature rules for volterra integral and integro-differential equations, IMA J. Numer. Anal. 2 (2) (1982) 131–152. https://doi.org/10.1093/imanum/2.2.131