Numerical ansatz for solving integro-differential equations with increasingly smooth memory kernels: spin-boson model and beyond

Michael Zwolak[1]

Institute for Quantum Information, California Institute of Technology, Pasadena, CA 91125, USA

E-mail: zwolak@theory.caltech.edu

Abstract. We present an efficient and stable numerical ansatz for solving a class of integro-differential equations. We define the class as integro-differential equations with *increasingly smooth* memory kernels. The resulting algorithm reduces the computational cost from the usual T^2 to $TC(T)$, where T is the total simulation time and $C(T)$ is some function. For instance, $C(T)$ is equal to $\ln T$ for polynomially decaying memory kernels. Due to the common occurrence of increasingly smooth memory kernels in physical, chemical and biological systems, the algorithm can be applied in quite a wide variety of situations. We demonstrate the performance of the algorithm by examining two cases. Firstly, we compare the algorithm to a typical numerical procedure for a simple integro-differential equation. Secondly, we solve the non-interacting blip approximation equations for the spin-boson model in real time.

[1] Present address: Theoretical Division MS-B213, Los Alamos National Laboratory, Los Alamos, NM 87545, USA.

Contents

1. Introduction

A major problem in the study of quantum systems is the influence of the environment on its dynamics. A variety of techniques, such as the influence functional [1], non-equilibrium perturbation theory [2, 3], and the weak-coupling approximation [4], have been developed to address this issue. The general approach is to derive equations of motion which only involve the degrees of freedom of the system. When working with systems weakly coupled to a rapidly relaxing environment, one can make the Markov approximation, which results in a local in time, first-order differential equation. This Markovian master equation is relatively easy to solve either analytically or numerically. As soon as the restrictions of weak coupling or fast environmental relaxation are removed, however, integro-differential equations appear, which reflects the fact that the environment retains a memory of the system at past times.

In particular, a number of interesting physical systems have integro-differential equations of motion with polynomially decaying memory kernels. For instance, the spin-boson model [5] and Josephson junctions with quasi-particle dissipation or resistive shunts [6] have $1/t^2$ asymptotic kernels when going through a dissipatively driven phase transition. The occurrence of polynomially decaying kernels is common in physics and chemistry because many open systems are connected to an Ohmic environment. Polynomially decaying kernels appear in biological applications as well [7]. Due to the frequent appearance of this type of kernel and also others which satisfy a requirement we call *increasingly smooth* below, it would be beneficial to have efficient numerical methods to handle the corresponding integro-differential equations. Such a method will be useful in applications in many disciplines. In addition, the method will enable the simulation of memory-dependent dissipation when used in conjunction with other computational methodologies, such as matrix product state algorithms for open systems [8–11].

In this paper, we give an efficient and stable numerical ansatz for solving integro-differential equations with increasingly smooth memory kernels. Using typical techniques, one incurs a computational cost of $AT^2 \propto N^2$, where A is some constant, T is the total simulation time and N is the number of time steps in the simulation. Thus, there is a considerable advantage in using a high-order approach to reduce the required value of N as much as possible. However, the computational cost of the algorithm described herein scales as $TC(T)$, where $C(T)$ is some function that depends on the form of the memory kernel. For example, $C(T)$ is equal to $\ln T$ for polynomially decaying kernels. Such a reduction represents a substantial advancement in numerically solving integro-differential equations since it allows for the efficient calculation of long-time behaviour.

This paper is organized as follows: in section 2, we introduce the types of equations under consideration, define *increasingly smooth*, and present the numerical ansatz with a discussion of errors. We demonstrate the

performance of the algorithm using two example integro-differential equations in section 3. Firstly, we consider an integro-differential equation composed of a polynomially decaying kernel and an oscillating function. Comparing directly to a two stage Runge–Kutta method, we see a large improvement in the scaling of the error as a function of computational cost. Secondly, we solve the non-interacting blip approximation (NIBA) equations for the spin-boson model in real time. We show that one can rapidly obtain the full real-time solution. In appendix A, we discuss how to extend the algorithm to higher orders. In appendix B, we outline the Runge–Kutta method we use as a comparison to the numerical ansatz. In appendix C, we give a simple derivation of the NIBA equations, showing the physical situation behind the appearance of an increasingly smooth memory kernel.

2. Algorithm

We want to be able to numerically solve linear integro-differential equations of the form

$$\frac{\partial \rho(t)}{\partial t} = -\mathcal{K} \int_0^t \mathrm{d}t' \alpha(t - t') \mathrm{e}^{-\mathcal{L}(t-t')} \mathcal{K}' \rho(t') = -\mathcal{K} \int_0^t \mathrm{d}\tau \alpha(\tau) \mathrm{e}^{-\mathcal{L}(\tau)} \mathcal{K}' \rho(t - \tau), \tag{1}$$

where $\rho(t)$ is some quantity of interest (like a density matrix) and $\alpha(\tau)$ is the memory kernel. $\mathcal{K}, \mathcal{K}'$ and $\mathcal{L}$ are here taken as time-independent operators. An equation of the form (1) appears in open quantum mechanical systems in the Born approximation to the full master equation [4], some exact non-Markovian master equations [12], and in phenomenological memory kernel master equations [13–15]. For an arbitrary form of the memory kernel, it is necessary to compute the integral on the right-hand side of (1) at each time step in the numerical simulation. Thus, ignoring the error of the simulation, the computational cost scales as T^2, where T is the total simulation time. This is prohibitively expensive in all but the shortest time simulations.

On the opposite extreme is when the memory kernel has an exponential form

$$\alpha(t - t') = \gamma \mathrm{e}^{-\gamma(t-t')}. \tag{2}$$

In this case, both functions responsible for evolving the integrand $(\alpha(t - t'), \mathrm{e}^{\mathcal{L}(t-t')})$ obey simple differential equations in t. This allows us to define a 'history'

$$H(t) = \int_0^t \mathrm{d}t' \alpha(t - t') \mathrm{e}^{\mathcal{L}(t-t')} \mathcal{K}' \rho(t'), \tag{3}$$

which obeys the differential equation

$$\dot{H}(t) = \gamma \mathcal{K}' \rho(t) - \gamma H(t) - \mathcal{L} H(t). \tag{4}$$

By solving this local in time differential equation together with

$$\dot{\rho}(t) = -\mathcal{K} H(t), \tag{5}$$

we can solve the integro-differential equation with a computational cost scaling as T. Cases in between these two extremes have a spectrum of computational costs scaling from T to T^2.

2.1. Increasing smoothness

We are interested in integro-differential equations of the form (1) with memory kernels which are *increasingly smooth*

$$\frac{\partial}{\partial t} \left| \frac{\alpha'(t)}{\alpha(t)} \right| < 0. \tag{6}$$

The idea behind increasing smoothness will become clear below. More specifically in this paper, we will look at examples where the memory kernel decays polynomially outside some cut-off time

$$\alpha(t) = \begin{cases} \alpha_{\Delta T}(t), & t < \Delta T, \\ \propto \frac{1}{t^p}, & t \geqslant \Delta T, \end{cases} \tag{7}$$

where $\alpha_{\Delta T}(t)$ is a bounded but otherwise arbitrary function and p is greater than one.[2] ΔT is a cut-off which allows us to ignore any transient behaviour in the memory kernel which does not satisfy the increasingly smooth condition (6). There will generally be some natural cut-off to the integral. For the overall scaling of the computational cost for large simulation times, we can ignore the resulting problem-dependent, constant computational cost due to the integral at times $t < \Delta T$.

Below, we will give an algorithm that is specific to the polynomial kernel (7). Similar algorithms are possible for any kernel which gets smoother as the time argument gets larger. The computational speedup will depend on how the function gets smoother. For instance, the polynomially decaying kernel gets smoother as

$$\left| \frac{(\partial 1/t^p)/\partial t}{1/t^p} \right| = \frac{p}{t}, \tag{8}$$

which allows one to take larger integration blocks as t gets larger. In this case, one can take a grid with spacing $\delta \propto t$ to cover the integration region. We call the partition around a grid point a *block*. In this case, the number of blocks scales logarithmically with the total time of the simulation. Consider another form of the kernel, $\beta(t)$, that gets smoother even faster than polynomial, for example

$$\left| \frac{\beta'(t)}{\beta(t)} \right| = b e^{-ct}. \tag{9}$$

This will give an even smaller number of blocks (that have exponential size in t) needed to cover a given range of integration and thus a faster speedup.[3] In this case, though, a simple truncation of the integral is sufficient, which is not the case for a polynomially decaying kernel.

2.2. Blocking algorithm for a polynomially decaying kernel

The algorithm is designed to take advantage of the fact that at each new time step in solving the integro-differential equation (1), we almost already have the integral on the right-hand side. Thus, if we have $\rho(t)$ at all the previous time steps and we group them together in a particular way, we should be able to do away with both storing all these previous values and the need for a full integration of the right-hand side. In more concrete terms, we want to be able to evaluate the history integral

$$I(T, \Delta T) = \int_{\Delta T}^{T} d\tau \, \alpha(\tau) F(\tau, T) \tag{10}$$

with

$$F(\tau, T) = e^{\mathcal{L}\tau} \mathcal{K}' \rho(T - \tau), \tag{11}$$

in such a way that when $T \to T + h$ in the integro-differential equation (1), all the pieces used to compute the integral can be recycled by evolving for a time step h and then added back together to get the new integral. This is just what is achieved for the exponential memory kernel (3), but in that case it is very extreme, the old integral simply has to be multiplied by a factor and added to a new piece. Below we show how to evaluate the integral with a blocking scheme that requires only $\ln T$ number of blocks to cover the integration region, can achieve any desired accuracy and has blocks that can all be recycled when $T \to T + h$. Again, we emphasize

---

[2] For the algorithm, p need not be greater than one. However, physical equations of motion, such as the NIBA equations at $\alpha = 1/2$, have an additional factor in the kernel to ensure its integral is bounded.

[3] Equation (9) has two solutions. Only the one with bounded integral would be physically relevant.

that the scheme we give now is specific for polynomially decaying memory kernels. The basic scheme can be constructed for other memory kernels as well simply by choosing a block size such that the block size times the smoothness stays constant.

Lets first rewrite the integral (10) as a sum over K blocks of width δ_i and Taylor expand the memory kernel:

$$
\begin{aligned}
I(T, \Delta T) &= \sum_{i=1}^{K} \int_{-\delta_i/2}^{+\delta_i/2} \mathrm{d}\epsilon \, F(\tau_i + \epsilon, T) \alpha(\tau_i + \epsilon) \\
&\approx \sum_{i=1}^{K} \int_{-\delta_i/2}^{+\delta_i/2} \mathrm{d}\epsilon \, F(\tau_i + \epsilon, T) \left\{ \alpha(\tau_i) + \alpha'(\tau_i)\epsilon + \mathcal{O}(\epsilon^2) \right\}.
\end{aligned} \tag{12}
$$

Here, only $\alpha(\tau_i + \epsilon)$ is expanded in order to make use of its increasing smoothness, which makes the change of $\alpha(\tau_i + \epsilon)$ over the block less relevant as the time argument gets larger. Given the complete integration of $F(\tau_i + \epsilon, T)$ over the block, the lowest order approximation to this integral is

$$
I^0(T, \Delta T) = \sum_{i=1}^{K} \alpha(\tau_i) \int_{-\delta_i/2}^{+\delta_i/2} \mathrm{d}\epsilon \, F(\tau_i + \epsilon, T). \tag{13}
$$

This equation represents the fact that we will take some step size, h, to compute the integral of F over the block, but use some other variable step size to calculate the integral of the product of α and F. For the polynomial kernel, the whole procedure is based on choosing a block size that grows with increasing distance from the current time,

$$
\delta_i = b\tau_i, \tag{14}
$$

or close to it, as shown in figure 1. We call b the *block parameter*. This function for the block size gives a constant value when multiplied by the smoothness (8) of the kernel. Assuming the function F is bounded by 1, the highest order error on each block is bounded by

$$
\left| \alpha'(\tau_i) \int_{-\delta_i/2}^{+\delta_i/2} \mathrm{d}\epsilon \, F(\tau_i + \epsilon, T)\epsilon \right| \leqslant \frac{p}{4} \frac{b^2}{\tau_i^{p-1}}. \tag{15}
$$

Thus, when choosing the blocks in this way, there is a constant ratio, $pb/4$, of the bound on the error to the bound on the integral. The bound on the highest order error of the whole integral is

$$
\left| \sum_{i=1}^{K} \alpha'(\tau_i) \int_{-\delta_i/2}^{+\delta_i/2} \mathrm{d}\epsilon \, F(\tau_i + \epsilon, T)\epsilon \right| \lesssim \frac{pb}{4(p-1)} \frac{1}{\Delta T^{p-1}}. \tag{16}
$$

Roughly, we then expect the error to decrease at least linearly in the block parameter b. We discuss the errors in more detail below.

If we choose the block positions perfectly, we can calculate the number of blocks required to cover the integration region. The first block has position (its midpoint)

$$
t_1 = \Delta T + \frac{b}{2}t_1 = \frac{\Delta T}{1 - (b/2)} \tag{17}
$$

and given the position of block i, block $i + 1$ is at

$$
t_{i+1} = t_i + \frac{b}{2}t_i + \frac{b}{2}t_{i+1} = \frac{1 + (b/2)}{1 - (b/2)}t_i. \tag{18}
$$

The nth block is then at

$$
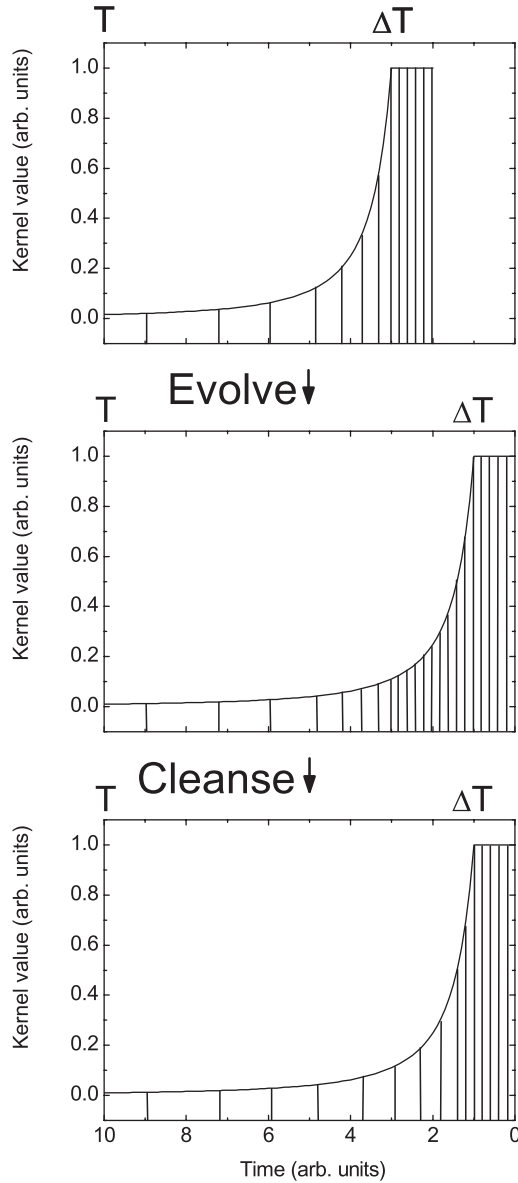t_n = \frac{(1 + (b/2))^{n-1}}{(1 - (b/2))^n} \Delta T. \tag{19}
$$

Figure 1. Procedure for evolving the integral (10) in time. We start with the integral represented as a sum over variable size blocks, with short times having constant block sizes and longer times having larger block sizes. We evolve for some amount of time (here shown as a large amount of time, but in practice this whole procedure is done discretely in the small time steps of the differential equation). Then we cleanse the blocks, grouping them together so long as their size is not too big for their location in time.

Since T is arbitrary, the nth block does not necessarily cover the remaining area. But approximately,

$$t_n \approx T - \frac{b}{2} T + \mathcal{O}(b^2) \tag{20}$$

and the total number of blocks for the integral is

$$K = \lceil n \rceil = \left\lceil \frac{\ln\left(\Delta T / T\right)}{\ln\left(1 - (b/2)\right) - \ln\left(1 + (b/2)\right)} \right\rceil . \tag{21}$$

In practice, K will always be bigger than (21) because of two factors. One is the finite step size, h, which forces the blocks to be an integral multiple of h. Two is that we are not able to take the complete integral and

divide it up perfectly. The division into blocks has to happen as we solve the integro-differential equation. We will see in section 2.3 that neither of these factors poses a serious problem and the true value of K will be of the same order as (21). For small b (or large K), we can simplify to

$$K \approx \frac{1}{b} \ln \left(\frac{T}{\Delta T} \right).$$ (22)

Within the algorithm, then, we need to keep track of the K integrals

$$I_i = \int_{-\delta_i/2}^{+\delta_i/2} d\epsilon \, F(\tau_i + \epsilon, T), \qquad \delta_i = b\tau_i,$$ (23)

which can be summed up with the weight $\alpha(\tau_i)$ to get the full integral. Putting in the explicit form of F, the K integrals are

$$\int_{-\delta_i/2}^{+\delta_i/2} d\epsilon \, e^{-\mathcal{L}(\tau_i + \epsilon)} \mathcal{K}' \rho(T - \tau_i - \epsilon).$$ (24)

When we increment $T \to T + h$, we first need to fix the block size, $\delta_i \to B_i$. Thus, the blocks will no longer be exactly $b\tau_i$ in width. As $T \to T + h$, $\tau_i \to \tau_i + h$ and the integrals are easily updated by

$$\int_{-B_i/2}^{+B_i/2} d\epsilon \, F(\tau_i + \epsilon, T) \to e^{-\mathcal{L}(h)} \left\{ \int_{-B_i/2}^{+B_i/2} d\epsilon \, F(\tau_i + \epsilon, T) \right\}.$$ (25)

After evolving, $B_i < b\tau_i$, which is acceptable since the smaller the blocks the smaller the error. The block sizes have to grow eventually or else we will not get the logarithmic coverage. Each time we increment T we can check whether nearest neighbour blocks can be grouped. We can group them so long as

$$B^{\text{new}} \leqslant b\tau^{\text{new}},$$ (26)

where τ^{new} is the midpoint of the new block. This is the 'cleansing' stage of figure 1, and is discussed in more detail in section 2.3. When the condition on the new block size is met, we can simply add the blocks together

$$\int_{-B^{\text{new}}/2}^{+B^{\text{new}}/2} d\epsilon \, F(\tau^{\text{new}} + \epsilon, T) = \int_{-B_i/2}^{+B_i/2} d\epsilon \, F(\tau_i + \epsilon, T) + \int_{-B_{i+1}/2}^{+B_{i+1}/2} d\epsilon \, F(\tau_{i+1} + \epsilon, T).$$ (27)

To summarize, we defined a logarithmic covering of the integration region in terms of growing blocks. As we solve the integro-differential equation, we can evolve the blocks and perform further groupings to ensure approximately the same covering. This covering allows for a logarithmic speedup in computing the history and a logarithmic reduction in the memory necessary to store the history. At the same time, due to the increasing smoothness of the polynomial memory kernel, it has a controllable error. The algorithm can be extended to higher orders as shown in appendix A.

2.3. Growing blocks in real time

The above description of the blocking algorithm presented the idea of using a logarithmic covering of the history integral. The formula (21) gives the number of blocks needed for the covering if we could choose their positions and widths perfectly. However, when solving the integro-differential equation, growing pains exist: the blocks have to be formed from the smaller blocks of the integration step size h. The block covering will thus be suboptimal.

To determine how close to optimal the number of blocks will be, we perform a simulation. Here, we choose a polynomially decaying function but with a more natural cut-off, i.e.,

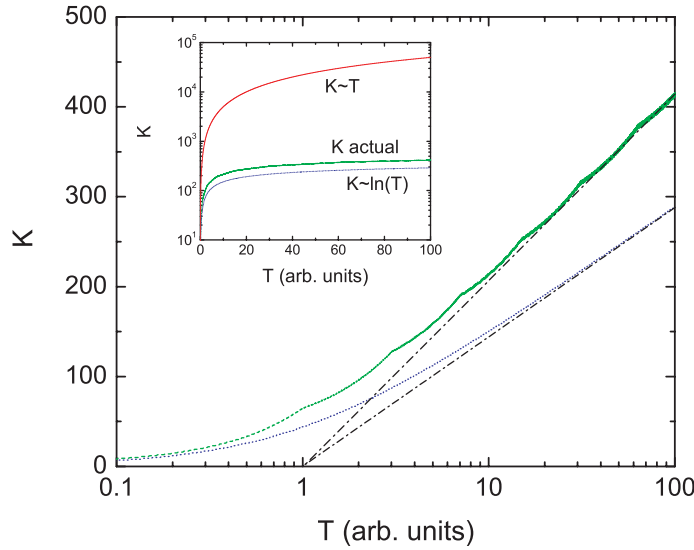$$\alpha(t) = \frac{1}{(t+1)^p}.$$ (28)

Figure 2. The number of blocks K versus T. The dotted (blue) curve shows the optimal case given by (21). The dashed (green) curve shows the number of blocks in practice. The dot-dashed curves represent the asymptotic behaviour of the number of blocks versus time. These curves intersect the axis at $T = 1$ because of the cut-off $\Delta T = 1$. The inset shows the two latter quantities plotted together with the linear growth of K for a typical numerical solution. The parameters in the evolution are the following: $\Delta T = 1$, $d = 0.016$ and $h = 0.002$.

This will be fully equivalent to $1/(t')^p$ with a cut-off $\Delta T = 1$ and a shifted time $t' = t + 1$. Because of this, the optimal formula (21) will only hold for large T (or with $\Delta T = 1$ and T replaced with $T + 1$). Within the simulation, we start at $T = 0$ and create blocks of size h. At each step we check if neighbouring blocks can be grouped by satisfying the condition (26). If they satisfy that condition, we group them and check again with the next block. We perform this grouping from the smallest to the largest blocks, but the directionality of growth does not matter. Figure 2 shows how the number of blocks grow in practice. Although suboptimal, it is still on the same order as the optimal K. How close it is to optimal is dependent on the step size and block parameter.

2.4. Errors

Given the blocking construction above, we can analyse how we expect the errors to behave versus both the number of blocks K and the computational cost. The first question that is natural to ask is what error do we make by the replacement of the integral (10) with the approximation (13). This gives us an idea of the error incurred by keeping K blocks of history in a logarithmic covering.

To answer this first we consider just the integral

$$\int_0^{T_f} \frac{e^{i\omega t}}{(t+1)^p} dt = \int_{\Delta T=1}^{T_f+1} \frac{e^{i\omega(t-1)}}{t^p} dt, \qquad (29)$$

where we have some frequency of oscillation ω and some power of the polynomial decay p. In the integration below, we take $p = 2$. It is important to note that in demonstrating the algorithm with just the integral (29), and not an integro-differential equation, it is not possible to show the computational speedup. However, it can show the accuracy of the integration as a function of the logarithmic coverage of the integration region when using the algorithm. We can also use it to get an idea of how the error changes when we vary ω or p without having to solve the more complicated integro-differential equations. The form of this integral was chosen because of its relevance to the real-time simulation of quantum systems, where the integral of the integro-differential equation will be a sum over oscillating terms times a polynomial memory kernel.

We gave a simple error bound above. To get a better idea of errors, we can examine the behaviour of the integral (29) as a function of the blocking parameter, frequency, and the integration range (e.g. simulation

time). If one takes the highest order error of the expansion (12), the worse case error in a block will occur when there is approximately one oscillation in the block,

$$b\tau_i = \frac{2\pi}{\omega}, \tag{30}$$

as this is when the asymmetry contributes most. However, this contribution will only occur in some finite region of the integral because the block sizes vary. When one changes the block parameter, this error will be shifted further back in time, e.g. to the left in figure 1. The memory kernel has a smaller value at this past time, and thus the error will be decreased. If one has many frequencies contributing to the integrand, then each of their error contributions will be shifted to a later time as one decreases the block parameter. Thus, the error will be systematically reduced. The same conclusion can be drawn if one expands both the functions in the integrand around their midpoint for each block. The fractional error (the highest order error term as a fraction of the zeroth order term) in this case is

$$\frac{b^2}{24}(p(p+1) - 2\imath\omega p\tau_i - \omega^2\tau_i^2). \tag{31}$$

The expansion will be invalid at approximately

$$\frac{\omega^2 b^2 \tau_i^2}{24} \approx 1, \tag{32}$$

which gives $b\tau_i \sim 5/\omega$.

We compare the logarithmic covering to the midpoint method with step sizes $h = 10^{-3} * 2^j$ with $j = 1, \ldots, 9$. This gives a number of blocks $K_{\text{mid}} = T_f/h$. For the algorithm, we use a step size $h = 10^{-3}$ to compute the integrals (23) and then use block parameters $b = 10^{-3} * 2^j$ with $j = 1, \ldots, 8$ to compute (13). The number of blocks K_{alg} is then given approximately by (21). It is only approximate because each block width has to be rounded down to a multiple of h, this gives a larger number of blocks. We use this larger number of blocks as the value for K_{alg}.

If we are just looking at the integral (29), we expect the error due to the algorithm to be

$$E_{\text{alg}} \propto K_{\text{alg}} b^3 \approx \frac{1}{K_{\text{alg}}^2} \left(\ln \frac{T_f}{\Delta T} \right)^3, \tag{33}$$

so long as $bT \lessgtr 2\pi/\omega$. This can be compared to the error of the midpoint method

$$E_{\text{mid}} \propto K_{\text{mid}} h^3 \approx \frac{T_f^3}{K_{\text{mid}}^2}. \tag{34}$$

Thus, if we want a fixed error, E, and end time, we get a difference in the number of blocks of

$$K_{\text{mid}} \propto \left(\frac{T_f^3}{E} \right)^{1/2} \tag{35}$$

compared to

$$K_{\text{alg}} \propto \left(\frac{(\ln(T_f/\Delta T))^3}{E} \right)^{1/2}, \tag{36}$$

which is quite large for long times.

Figure 3 shows how the error behaves versus the number of blocks used to compute the integral. The fast decrease of error as a function of K_{alg} shows that we can block together integrals as we evolve the integro-differential equation and have a controlled error. Further, the longer the simulation time, T_f, the better performing the algorithm should be when compared to the midpoint method. In addition, the overall
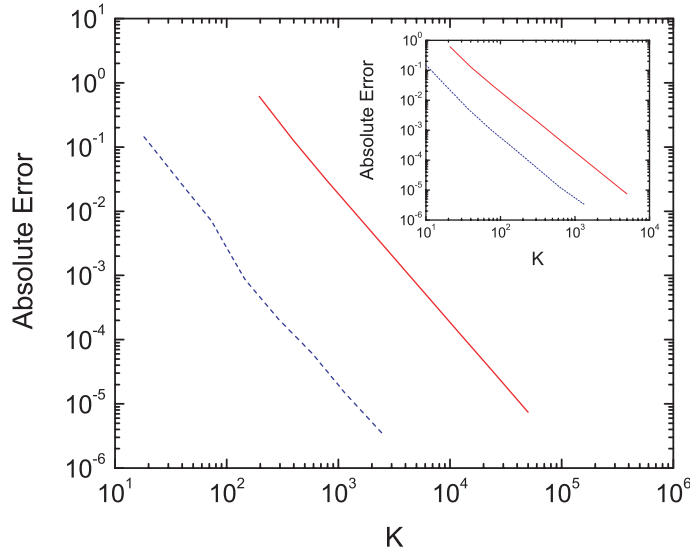
Figure 3. Absolute error versus number of blocks for the two methods of calculating the integral (29) with $\omega = 2\pi$ and $p = 2$ (similar curves were found for other ω). The solid (red) curves are for the midpoint method and the dashed (blue) curves are for the first-order blocking algorithm. The main figure shows the error for $T_f = 100$ and the inset shows it for $T_f = 10$. Since the midpoint method is second order, we find what is expected, $E \propto K^{-2}$. The blocking algorithm has the same dependence on K. The difference in error for the same number of blocks with the two methods is dependent on $T/\ln T$, reflecting the fact that the algorithm chooses blocks at optimal places and only covers the integration area with $\ln T$ blocks. We take as the exact value the one computed with a step size $h = 10^{-4}$.

performance of the algorithm should not be significantly affected by the frequency of integrand oscillation ω or the power p. One interesting feature of the error versus K_{alg} figure is the slight dip in the error at approximately $bT \sim 2\pi/\omega$, which represents the error discussed above being pushed out of the integration region.

The second question we can ask is how the error depends on the computational cost compared to more traditional methods. If we use the method for integro-differential equations outlined in appendix B, we have an error at each step of h^3 and we have N steps. Thus

$$E_{\mathrm{mid}} \propto N h^3 \propto \frac{1}{C_{\mathrm{pu}}}, \tag{37}$$

where $C_{\mathrm{pu}} \propto N^2$ is the computational cost and we hold the simulation time fixed. For the algorithm

$$E_{\mathrm{alg}} \propto \frac{1}{C_{\mathrm{pu}}^2}, \tag{38}$$

where we hold the step size h fixed and $C_{\mathrm{pu}} = NK$. That is, the error goes down with a higher power in the computational cost.

Of course, using the algorithm is not the only way to get the error to scale better with the computational cost. One can also just block the history with constant blocks larger than h. Although the error does scale better, the error can never be lower than just using the brute force method, e.g. the error versus cost curve will never cross the brute force curve. For the algorithm these two curves do cross, as we will see below.

3. Examples

We consider two example problems to demonstrate the performance of the algorithm: (1) solving an integro-differential equation that has both an oscillating function and a polynomially decaying function in the integral and (2) the NIBA equations for the spin-boson model.
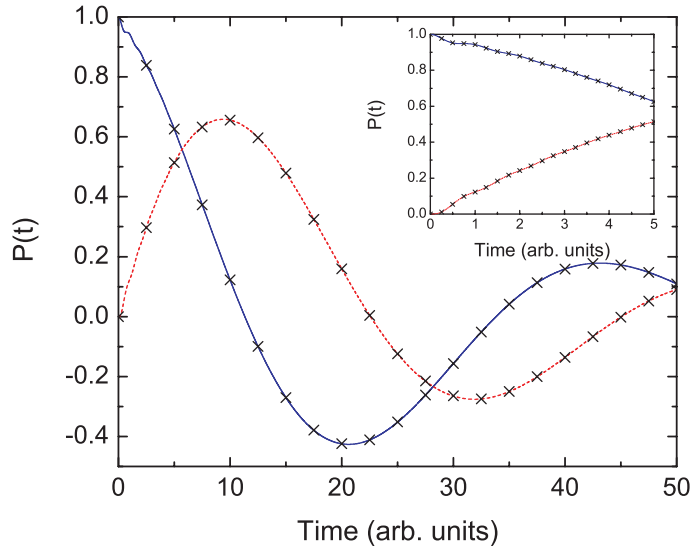
Figure 4. Real time evolution of (42) with $\omega = 2\pi$. The solid (blue) curve shows the real part of $P(t)$ and the dashed (red) curve the imaginary part. These curves were calculate with the algorithm with $h = 0.002$ and $d = 0.016$. The crosses represent the solution using the midpoint procedure of appendix B with a step size of $h = 0.001$. The inset shows a blow up of the initial time where there is some transient structure in the solution.

3.1. Oscillating integro-differential equation

The oscillating integro-differential equation we use is

$$\dot{n}(t) = \imath \omega n(t) - \int_0^t \mathrm{d}t' \alpha(t - t') n(t') \tag{39}$$

with

$$\alpha(t) = \frac{1}{(t + 1)^2} \, . \tag{40}$$

We get rid of the oscillation in the differential equation by setting

$$P(t) = \mathrm{e}^{-\imath \omega t} n(t), \tag{41}$$

which gives

$$\dot{P}(t) = -\int_0^t \mathrm{d}t' \alpha(t - t') \mathrm{e}^{-\imath \omega(t - t')} P(t'). \tag{42}$$

This integro-differential equation involves the computation of an integral similar to (29), with $p = 2$, at each time step. With the algorithm, we will be able to recycle the integral with the logarithmic covering at each step, rather than computing it from scratch. Note that we choose blocks by $b(t + 1)$ here because we have the cut-off included in the polynomial decay. The simulation results for one set of parameters are shown in figure 4.

Now we examine the error versus computational cost. Figure 5 is one of the main results of this paper. It shows three key results. (i) The algorithm has significantly better error for the same computational cost compared to a brute force method in a large range of parameter space. (ii) As the simulation time is increased, the efficiency of the algorithm drastically increases compared to the brute force method. (iii) The figure also suggests how to use the whole ansatz. Rather than have two parameters b and h, one should set $b \propto h$ (for polynomial decay). This will give a similar convergence to the exact answer as the second-order method, but a smaller factor of proportionality.
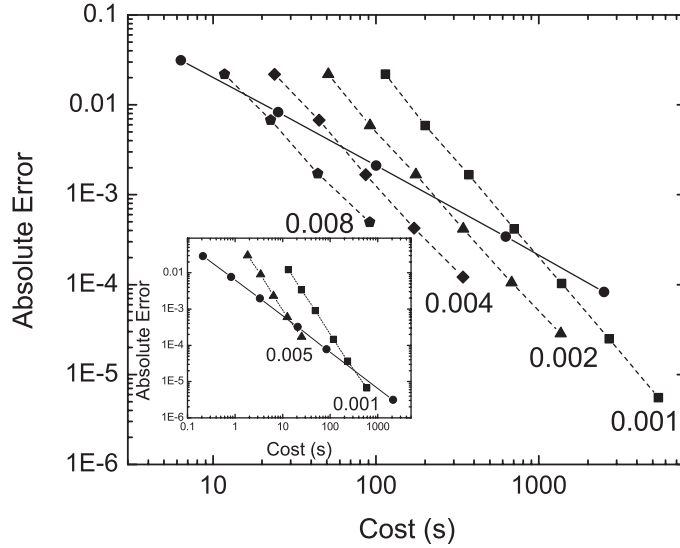
Figure 5. Absolute error versus computational cost. The error is computed by comparing the simulation results with those the brute force method with $h = 0.001$ for $T_f = 50$ (in the inset, the 'exact' solution is taken as the brute method with $h = 0.0002$ and $T_f = 10$). Circles represent the brute force method for step sizes 0.1, 0.05, 0.025, 0.01 and 0.005 (in the inset, 0.1, 0.05, 0.025, 0.01, 0.005 and 0.001). The other data represent using the algorithm with the step size as shown and each data point has a different block parameter starting with $2h$ and increasing by factors of 2. For the longer simulation time of $T_f = 50$, the algorithm has quite a substantial decrease in error for the same computational cost (for the shorter time in the inset, one can see that there is barely a gain. This is due to the overhead cost in implementing the algorithm). Also, the curves have the behaviour discussed in section 2.4. We perform all computations on a P4, 3 GHz processor.

As mentioned in section 2.4, one can also consider another method that just uses a constant grouping to reduce the number of blocks in the history. However, this method cannot achieve better results as determined by the error versus computational cost. For a given step size, the highest accuracy will be achieved by using the brute procedure. A lower computational cost can be achieved by grouping the history into constant blocks, but the reduction in the computational cost will yield an even stronger loss in accuracy because $E \propto C^{-2}$, when one step size is fixed. Thus, a similar figure to figure 5 would show curves moving only upward from the brute force line and not crossing it.

3.2. Non-interacting blip approximation equations

The NIBA for the spin-boson model gives a good physical example for a system with a polynomially decaying kernel. The NIBA equations are derived by Leggett *et al* [5], see also appendix C for a simple derivation, from which we can get the equation of motion

$$\dot{P}(t) = -\int_0^t \mathrm{d}t' f(t - t') P(t'), \tag{43}$$

where

$$f(t) = \Delta^2 \cos\left[2\alpha \tan^{-1} t\right] \left[\frac{(\pi t/\beta\omega_\mathrm{c}) \operatorname{csch}(\pi t/\beta\omega_\mathrm{c})}{(1 + t^2)^{1/2}}\right]^{2\alpha}. \tag{44}$$

Time is in units of the cut-off frequency ω_c^{-1}. At zero temperature ($\beta \to \infty$), the kernel becomes

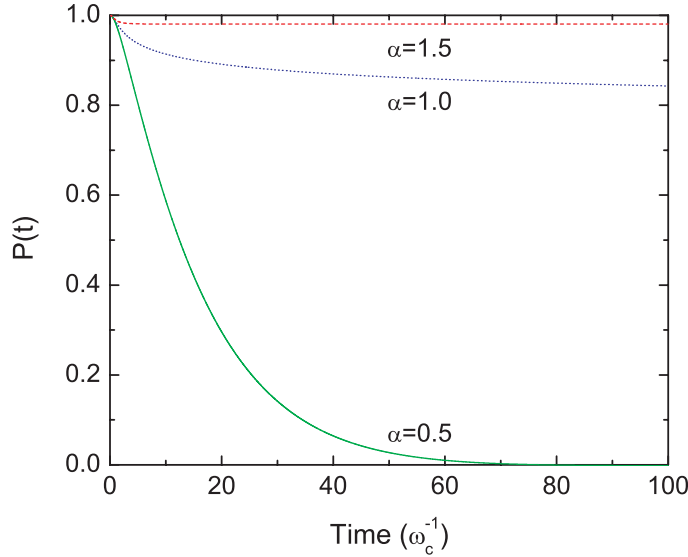$$f(t) = \frac{\Delta^2 \cos\left[2\alpha \tan^{-1} t\right]}{(1 + t^2)^\alpha}. \tag{45}$$

12

Figure 6. NIBA equations simulated in real time for $\alpha = 0.5$, 1.0 and 1.5. The simulations were done with $h = 0.008$ and $d = 0.016$. They took approximately 2 min of CPU time on a P4, 3 GHz processor. This is orders of magnitude faster than a brute force approach would take to get the same error. From the figure, we can clearly observe the physics of the different regimes. For weak dissipation, one gets decay to $P = 0$. For the transition, $\alpha = 1$, one gets polynomial decay of $P(t)$. Above the transition, $P(t)$ gets locked into a non-zero value.

The $1 + t^2$ has a physical cut-off to the low time behaviour of the polynomial decay. There are only two parameters of interest to us. We set $\Delta = 0.2$ as it is required to be small (see appendix C) and we vary the dissipation strength α to see the behaviour of $P(t)$ on different sides of the dissipative transition. Varying α also shows the algorithm at work for different powers of polynomial decay.

Depending on the value of the dissipation strength α, we have to use a different cut-off ΔT. For $\alpha = 0.5$, 1.0 and 1.5, we use $\Delta T = 1$, 3, and 2, respectively. After these cut-offs, $f(t)$ is increasingly smooth (it is also smoother than bare polynomial decay). We want to point out that due to the form of the polynomial decay, the most efficient way to choose block sizes is by $b(1 + t^2)/t$, which just comes from the inverse of the increasing smoothness. Despite this, we still use the less efficient bt.

The simulation results for the NIBA equations are shown in figure 6. Using the algorithm allows one to have long simulation times for negligible computational cost. Further, simulations of NIBA-like equations on a lattice or with time-dependent Hamiltonians can be done efficiently with the method.

We want to emphasize that the simulation of the NIBA equations at finite temperature (44) can also use the algorithm presented in this paper. The finite temperature, however, introduces a long-time cut-off into the kernel, beyond which the kernel may not be increasingly smooth. The contribution beyond this cut-off, though, is negligible. Thus, the algorithm can be used for times less than $\sim \beta$ exactly how it is used for the zero temperature case, and there can be a truncation of the kernel at times beyond times $\sim \beta$.

4. Conclusions

We have given an efficient and stable numerical algorithm for solving integro-differential equations with increasingly smooth memory kernels. The general computational speedup is $T^2 \rightarrow TC(T)$, where $C(T)$ depends on how fast the kernel gets smooth. For example, the computational cost of the algorithm for polynomially decaying kernels is $T \ln T$ rather than the usual T^2.

Using a simple integro-differential equation, we demonstrated how well the algorithm performs compared to a second order, constant step size method. For long times, there is quite a substantial speedup in the computation cost to achieve a given error. The solution to the NIBA equations for the spin-boson model in real-time showed that one can get results and analyse the situation quite rapidly. Similar procedures can be

applied to other forms of memory kernels which satisfy the increasingly smooth condition (6). In these other cases, the computational speedup can be better or worse than the case presented here.

In practice, the usefulness of this algorithm is due to the possibility of its incorporation into other techniques, such as matrix product state algorithms, or its use with time-dependent Hamiltonian terms. Thus, one can simulate strongly-correlated lattice or driven systems subject to strong dissipation, which will be the subject of a future contribution. The algorithm can also be used for equations less restrictive than (1), such as integro-differential equations with memory kernels dependent on both time arguments.

Appendix A. Higher order blocking algorithms

The algorithm above can be extended to higher orders. For instance, if we want the next order method, we need to be able to evaluate and update

$$I^1(T, \Delta T) \quad = I^0(T, \Delta T) + \sum_{i=1}^{K} \alpha'(\tau_i) \int_{-\delta_i/2}^{+\delta_i/2} d\epsilon \, F(\tau_i + \epsilon, T)\epsilon \, . \tag{A.1}$$

The latter term picks out the asymmetric part of F inside of block i and then multiplies it by the derivative of α. Let us define this asymmetric part of the block as

$$A_i(T) \equiv \int_{-\delta_i/2}^{+\delta_i/2} d\epsilon \, F(\tau_i + \epsilon, T)\epsilon. \tag{A.2}$$

If we suppose we have these asymmetric integrals at a given time, and we want to increment T, we first need to fix the block size, as before. Then we can update them by

$$A_i(T + h) = e^{-\mathcal{L}(h)} A_i(T). \tag{A.3}$$

We also need to be able to add these blocks together. We can do this by

$$\begin{aligned} A^{\text{new}}(T) &= \int_{-B^{\text{new}}/2}^{+B^{\text{new}}/2} d\epsilon \, F(\tau^{\text{new}} + \epsilon, T)\epsilon \\ &= A_i(T) - B_{i+1} I_i(T) + A_{i+1}(T) + B_i I_{i+1}(T), \end{aligned} \tag{A.4}$$

where we use the asymmetric integrals from before we did the blocking and also the first-order integrals (23) for the two blocks. The error for an order z algorithm will have a bound proportional to b^z.

Appendix B. Two stage Runge–Kutta method

In this paper, we compare the proposed numerical ansatz to a two stage Runge–Kutta method. Since we are dealing with integro-differential equations, we give the details of the second-order technique we use. In the main text, we discuss how the errors scale with total simulation time, step size and computational cost.

For all the examples in this work, the basic integro-differential equation (1), is reduced to

$$\frac{\partial \rho(t)}{\partial t} = \int_0^t dt' \alpha(t - t') e^{-\mathcal{L}(t-t')} \rho(t'), \tag{B.1}$$

e.g. $K, K' = \mathrm{I}$. In a generic form, we can write

$$\frac{\partial \rho(t)}{\partial t} = f\left[t, \rho(t), \int_0^t \mathrm{d}t' \mathcal{F}(t - t')\rho(t')\right]. \tag{B.2}$$

Discretizing time as

$$t_n = t_0 + nh, \tag{B.3}$$

we can write a general two stage Runge–Kutta integration scheme

$$\rho_{n+1} = \rho_n + h\,\mathrm{d}\rho_n, \tag{B.4}$$

$$\mathrm{d}\rho_n = f[t, P_n, \tilde{z}_n], \tag{B.5}$$

$$P_n = \rho_n + \mathrm{d}P_n, \tag{B.6}$$

$$\mathrm{d}P_n = \frac{h}{2} f\left[t, \rho_n, z_n\right], \tag{B.7}$$

$$z_n = h\sum_{m=1}^{n-1} \bar{\mathcal{F}}_{nm}\left(\rho_m + \rho_{m+1}\right)/2, \tag{B.8}$$

and

$$\tilde{z}_n = h\sum_{m=1}^{n-1} \bar{\mathcal{F}}_{nm}\left(\rho_m + \rho_{m+1}\right)/2 + \frac{h}{2}\bar{\mathcal{F}}_0\left(\rho_n + P_n\right)/2, \tag{B.9}$$

where

$$\bar{\mathcal{F}}_{nm} = \frac{1}{h}\langle\mathcal{F}(t_n - t')\rangle_{t_m}^{t_m+h} = \frac{1}{h}\int_{t_m}^{t_m+h} \mathrm{d}t'\mathcal{F}(t_n - t'), \tag{B.10}$$

and

$$\bar{\mathcal{F}}_0 = \frac{2}{h}\int_{t_n-h/2}^{t_n} \mathrm{d}t'\mathcal{F}(t_n - t'). \tag{B.11}$$

Although using the average $\bar{\mathcal{F}}_{nm}$ over an interval does not increase the order of the method, it does preserve important properties of the kernel such as its total integration. This is very important in cases where the kernel integrates to zero and thus the transient behaviour completely determines the steady state. We use the average of the kernel over each block with the algorithm as well. The Runge–Kutta scheme can of course be generalized to higher stages and to kernels with two time arguments [16–19].

Appendix C. Simple derivation of the non-interacting blip approximation equations

In this appendix, we give a simple derivation of the NIBA equations for the spin-boson model to show how polynomially decaying memory kernels can arise physically in the case of strong dissipation. The derivation of the NIBA equations is based on the observation that they come from a Born approximation of a transformed Hamiltonian [20].

The spin-boson Hamiltonian is

$$H_{\mathrm{SB}} = -\frac{1}{2}\Delta\sigma_x + \frac{1}{2}\epsilon\sigma_z + \sum_k \omega_k a_k^\dagger a_k + \gamma\sigma_z\sum_k g_k(a_k^\dagger + a_k), \tag{C.1}$$

where we have a two level system with internal coupling constant Δ and bias ϵ. The two level system is coupled to a collection of bosons of frequencies $\{\omega_k\}$ with a coupling constant $g_k = c_k/\sqrt{2m_k\omega_k}$ and overall coupling factor γ. The spectral density of the bath is given by

$$J(\omega) = \pi \sum_k g_k^2 \delta(\omega - \omega_k). \tag{C.2}$$

The physical scenario we want to explore is one in which the two level system is initially held fixed in an eigenstate of σ_z while the bath equilibrates around this state. That is, the bath will equilibrate around the Hamiltonian

$$H_{\rm R} = \sum_k \omega_k a_k^\dagger a_k + \sum_k \gamma g_k (a_k^\dagger + a_k), \tag{C.3}$$

if we hold the two level system in the $+1$ eigenstate of σ_z. This gives the thermal starting state for the bath

$$R_0 = \frac{{\rm e}^{-\beta H_R}}{Z_R}. \tag{C.4}$$

Then, we want to release the two level system and follow its dynamics in real time. In particular, we look at the expectation value of σ_z

$$P(t) \equiv \langle \sigma_z \rangle_t = {\rm tr}\left\{ {\rm e}^{{\rm i}H_{\rm SB}t} \sigma_z {\rm e}^{-{\rm i}H_{\rm SB}t} |1\rangle\langle 1| \otimes R_0 \right\}. \tag{C.5}$$

Since we are interested in strong dissipation, we can perform a canonical transformation on this Hamiltonian to incorporate all orders of the system–bath interaction. With

$$S = -\sum_k \frac{\gamma g_k}{\omega_k} (a_k - a_k^\dagger) \sigma_z, \tag{C.6}$$

we get

$$H = {\rm e}^S H_{\rm SB} {\rm e}^{-S} \tag{C.7}$$

$$= -\frac{1}{2}\Delta\,(\sigma_+ B_- + \sigma_- B_+) + \frac{1}{2}\epsilon\sigma_z + \sum_k \omega_k a_k^\dagger a_k, \tag{C.8}$$

where

$$B_\pm = \exp\left\{ \pm 2 \sum_k \frac{\gamma g_k}{\omega_k}(a_k - a_k^\dagger) \right\}. \tag{C.9}$$

For the unbiased case, $\epsilon = 0$, this gives the interaction picture Hamiltonian

$$H^{\rm I}(t) = -\tfrac{1}{2}\Delta(\sigma_+ B_-^{\rm I}(t) + \sigma_- B_+^{\rm I}(t)) \tag{C.10}$$

with

$$B_\pm^{\rm I}(t) = \exp\left\{ \pm 2 \sum_k \frac{\gamma g_k}{\omega_k}(a_k {\rm e}^{-{\rm i}\omega_k t} - a_k^\dagger {\rm e}^{{\rm i}\omega_k t}) \right\}. \tag{C.11}$$

We can then transform the equation for $P(t)$ to get

$$P(t) = {\rm tr}\left\{ {\rm e}^{{\rm i}Ht} \sigma_z {\rm e}^{-{\rm i}Ht} |1\rangle\langle 1| \otimes R_0' \right\}, \tag{C.12}$$

where

$$R_0' = {\rm e}^{-\beta \sum_k \omega_k a_k^\dagger a_k}/Z_R'. \tag{C.13}$$

We can find the master equation in the Born approximation for $P(t)$, also known as the NIBA, by performing perturbation theory on (C.12). To second order in Δ,

$$\dot{P}(t) = -\int_0^t dt' f(t - t') P(t') \tag{C.14}$$

with

$$f(t) = \frac{\Delta^2}{4} \left\{ \langle [B_+^I(t), B_-^I(0)] \rangle_{R_0'} + \langle [B_+^I(-t), B_-^I(0)] \rangle_{R_0'} \right\} \tag{C.15}$$

$$= \frac{\Delta^2}{2} \langle [B_+^I(t), B_-^I(0)] \rangle_{R_0'}, \tag{C.16}$$

where we have used that the correlation functions are equal. To compute $f(t)$ we can use the Feynman disentangling of operators [21] to get, in the notation of Leggett *et al* [5],

$$f(t) = \Delta^2 \cos\left\{ \frac{4\gamma^2}{\pi} Q_1(t) \right\} \exp\left\{ -\frac{4\gamma^2}{\pi} Q_2(t) \right\} \tag{C.17}$$

with

$$Q_1(t) = \int_0^\infty d\omega \frac{J(\omega)}{\omega^2} \sin(\omega t) \tag{C.18}$$

and

$$Q_2(t) = \int_0^\infty d\omega \frac{J(\omega)}{\omega^2} (1 - \cos \omega t) \coth\left(\frac{\beta\omega}{2}\right). \tag{C.19}$$

For Ohmic dissipation, $J(\omega) = \eta\omega \exp(-\omega/\omega_c)$, these quantities become

$$Q_1(t) = \eta \tan^{-1} \omega_c t \tag{C.20}$$

and

$$Q_2(t) = \frac{\eta}{2} \ln\left(1 + \omega_c^2 t^2\right) + \eta \ln\left(\frac{\beta}{\pi t} \sinh \frac{\pi t}{\beta}\right). \tag{C.21}$$

With $\alpha \equiv 2\eta\gamma^2/\pi$, this gives (44) for the NIBA memory kernel.

References

[1] Feynman R P and Vernon F L 1963 *Ann. Phys. (NY)* **24** 118
[2] Keldysh L V 1965 *Sov. Phys.—JETP* **20** 1018
[3] Kadanoff L P and Baym G 1962 *Quantum Statistical Mechanics* (New York: Benjamin)
[4] Breuer H-P and Petruccione F 2002 *The Theory of Open Quantum Systems* (Oxford: Oxford University Press)
[5] Leggett A J, Chakravarty S, Dorsey A T, Fisher M P A, Garg A and Zwerger W 1987 *Rev. Mod. Phys.* **59** 1
[6] Schön G and Zaikin A D 1990 *Phys. Rep.* **198** 237
[7] Cushing J M 1977 *Integrodifferential Equations and Delay Models in Population* (Berlin: Springer)
[8] Zwolak M and Vidal G 2004 *Phys. Rev. Lett.* **93** 207205
[9] Verstraete F, Garcia-Ripoll J J and Cirac J I 2004 *Phys. Rev. Lett.* **93** 207204
[10] Vidal G 2003 *Phys. Rev. Lett.* **91** 147902
[11] Vidal G 2004 *Phys. Rev. Lett.* **93** 040502
[12] Zwolak M and Refael G 2007 *in preparation*
[13] Shabani A and Lidar D A 2001 *Phys. Rev.* A **71** 020101
[14] Barnett S M and Stenholm S 2001 *Phys. Rev.* A **64** 033808
[15] Daffer S, Wódkiewicz K, Cresser J D and McIver J K 2004 *Phys. Rev.* A **70** 010304
[16] Leathers A S and Micha D A 2005 *Chem. Phys. Lett.* **415** 46
[17] Brunner H and van der Houwen P J 1986 *The Numerical Solution of Volterra Equations* (New York: North-Holland)
[18] Linz P 1985 *Analytical and Numerical Methods for Volterra Equations* (Philadelphia, PA: SIAM)
[19] Lubich Ch 1982 *Numer. Math.* **40** 119
[20] Aslangul C, Pottier N and Saint-James D 1986 *J. Physique* **47** 1657
[21] Mahan G 2000 *Many-Particle Physics* (New York: Kluwer Academic/Plenum)