Quantum algorithm for time-dependent differential equations using Dyson series

Dominic W. Berry and Pedro C. S. Costa

School of Mathematical and Physical Sciences, Macquarie University, Sydney, New South Wales 2109, Australia

Time-dependent linear differential equations are a common type of problem that needs to be solved in classical physics. Here we provide a quantum algorithm for solving time-dependent linear differential equations with logarithmic dependence of the complexity on the error and derivative. As usual, there is an exponential improvement over classical approaches in the scaling of the complexity with the dimension, with the caveat that the solution is encoded in the amplitudes of a quantum state. Our method is to encode the Dyson series in a system of linear equations, then solve via the optimal quantum linear equation solver. Our method also provides a simplified approach in the case of time-independent differential equations.

Quantum computers offer tremendous potential advantages in computing speed, but it is a long-standing challenge to find speedups to important computational tasks. A major advance for quantum algorithms was the development of a way of solving systems of linear equations by Harrow, Hassidim, and Lloyd [11]. This algorithm provides an exponential improvement in complexity in terms of the dimension as compared to classical solution, with some caveats. In particular, the matrix needs to be given in an oracular way (rather than as classical data), and the solution is encoded in amplitudes of a quantum state.

There has therefore been a great deal of follow-up work on applications of solving linear equations where the result is potentially useful despite these limitations. For example, in discretised partial differential equations (PDEs), the discretisation yields a large set of simultaneous equations. Then one may aim to obtain some global feature of the solution which may be obtained by sampling from the prepared quantum state. This was the principle used in [8], and there have been considerable further advances since then [7, 16].

There is also the question of how to solve an *ordinary* differential equation (ODE). That is, spatial dimensions are not given explicitly (though the set of equations may have been obtained by a spatial discretization of a PDE), and the task is to solve the time evolution. The original algorithm for this task used linear multistep methods [1], and a further improvement was to use a Taylor series encoded in a larger system of linear equations to obtain complexity logarithmic in the allowable precision [3]. For this and other quantum algorithms for solving differential equations, it is required that the equations are dissipative in order to provide an efficient solution. Krovi [14] showed how to solve the time-independent case with an alternative dissipative condition based on the logarithmic norm rather than the eigenvalues of the matrix having non-positive real part as in [3].

That then leaves open the question of how to solve time-dependent differential equations. Algorithms for time-dependent Hamiltonian evolution have been given based on a Dyson series in [12, 15]. A method for time-dependent differential equations was given in

Ref. [5]. That used a spectral method to provide near-linear dependence on T, with poly-logarithmic factors in T and the allowable error. The poly-logarithmic factors were not explicitly given in [5], but they are quite large. In particular, n is chosen logarithmically in Eq. (8.6) of that work, then there is a factor of n^4 for the complexity in Eq. (8.15) of Ref. [5]. Another limitation of the approach of Ref. [5] is that it requires smoothness of the differential equation, with a complexity depending on arbitrary-order derivatives of the parameters. Reference [10] used a time-marching strategy for time-dependent differential equations, but gave quadratic dependence of the complexity on the overall evolution time T, making it considerably more costly. Moreover, it only analyses the homogeneous case.

Here we provide an algorithm for inhomogeneous time-dependent differential equations with significantly improved logarithmic dependence on the error and linear dependence on T up to logarithmic factors. Our logarithmic factors are significantly improved over those in [5]. Moreover, we need no smoothness condition as in Ref. [5]. Our oracle complexity is entirely independent of derivatives of the parameters, and the complexity in terms of elementary gates depends only on the log of the first derivatives of parameters. We combine methods from both [3] and [12, 15]. We use a block matrix similar to [3], but we do *not* use extra lines of the block matrix to implement terms in the series as in that work. Instead we construct this matrix via a block encoding using a Dyson series, in an analogous way as the block encodings in [12, 15].

In Section 1 we describe the form of the solution in terms of the sum of a solution to the homogeneous equation and a particular solution. Then in Section 2 we express the method in terms of a set of linear equations and determine the condition number. In Section 3 we describe the method to block encode the linear equations. Lastly we use these results to give the overall complexity in Section 4, and conclude in Section 5.

1 Form of solution

In this section we summarise standard methods of solving a linear differential equation via a Dyson series. A general ordinary linear differential equation has form

$$\dot{\boldsymbol{x}}(t) = A(t)\boldsymbol{x}(t) + \boldsymbol{b}(t), \tag{1}$$

where $\boldsymbol{b}(t) \in \mathbb{C}^N$ is a vector function of t, $A(t) \in \mathbb{C}^{N\times N}$ is a coefficient matrix and $\boldsymbol{x}(t) \in \mathbb{C}^N$ is the solution vector as a function of variable t. We are also given an initial condition $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$. Note that we can always express a higher-order ODE as a system of first-order differential equations by defining new parameters.

We will write the general solution to (1) as

$$\boldsymbol{x}(t) = \boldsymbol{x}_H(t) + \boldsymbol{x}_P(t), \tag{2}$$

where $\boldsymbol{x}_H(t)$ is the solution to the homogeneous equation $\dot{\boldsymbol{x}}_H = A(t)\boldsymbol{x}_H$ and $\boldsymbol{x}_P(t)$ is a particular solution. In the next two subsections we show how to compute them.

1.1 Solution to the homogeneous equation

First we solve a differential equation of the form

$$\dot{\boldsymbol{x}}_H = A(t)\boldsymbol{x}_H. \tag{3}$$

A general solution for $\boldsymbol{x}_H$ can be expressed in the form of a Dyson series $\boldsymbol{x}_H(t) = W(t, t_0)\boldsymbol{x}_H(t_0)$ with

$$W(t, t_0) := \sum_{k=0}^{\infty} \frac{1}{k!} \int_{t_0}^{t} dt_1 \int_{t_0}^{t} dt_2 \cdots \int_{t_0}^{t} dt_k \, \mathcal{T} A(t_1)A(t_2)\cdots A(t_k), \tag{4}$$

where $\mathcal{T}$ indicates time ordering. We will briefly review the algorithm for the first segment. We can explicitly order the solution and truncate the infinite sum at a cutoff K to give

$$W_K(t,t_0) \coloneqq \sum_{k=0}^{K} \int_{t_0}^{t} dt_1 \int_{t_0}^{t_1} dt_2 \cdots \int_{t_0}^{t_{k-1}} dt_k \, A(t_1)A(t_2)\cdots A(t_k). \tag{5}$$

Note that the $k!$ disappears because the times without ordering have $k!$ permutations, so sorting them gives $k!$ multiplicity. This expression is just equivalent to the Dyson series solution for Hamiltonian evolution used in Refs. [12, 15], and the fact that A is a more general matrix does not affect the form of the equations. Then, using $A_{\max} = \max_t \|A(t)\|$,

$$\begin{aligned} \|W_K(t,t_0) - W(t,t_0)\| &\leq \sum_{k=K+1}^{\infty} \frac{(A_{\max}\Delta t)^k}{k!} \\ &\leq \frac{(A_{\max}\Delta t)^{K+1}}{(K+1)!} \exp(A_{\max}\Delta t) \\ &= \mathcal{O}\left(\frac{(A_{\max}\Delta t)^{K+1}}{(K+1)!}\right), \end{aligned} \tag{6}$$

with $\Delta t = t - t_0$, provided Δt is chosen so $A_{\max}\Delta t$ is $\mathcal{O}(1)$. Here and throughout the paper we are taking $\|\cdot\|$ to be the spectral norm for operators, or the 2-norm for vectors.

If the goal is to give error no larger than ϵ due to the truncation, then we can choose $K \propto \log(1/\epsilon)/\log\log(1/\epsilon)$. To solve for a longer time we break the time into r segments and use the truncated Dyson series on each segment. Therefore the error in each segment should be no larger than ϵ/r. The choice of K should then be

$$K = \mathcal{O}\left(\frac{\log(r/\epsilon)}{\log\log(r/\epsilon)}\right). \tag{7}$$

1.2 Particular solution

Generalising to an inhomogeneous equation

$$\dot{\boldsymbol{x}} = A(t)\boldsymbol{x} + \boldsymbol{b}(t), \tag{8}$$

the particular solution (i.e. with initial condition of zero) is of the form $\boldsymbol{x}_P(t) = \boldsymbol{v}(t,t_0)$ with

$$\boldsymbol{v}(t,t_0) \coloneqq \sum_{k=1}^{\infty} \int_{t_0}^{t} dt_1 \int_{t_0}^{t_1} dt_2 \cdots \int_{t_0}^{t_{k-1}} dt_k A(t_1)A(t_2)\cdots A(t_{k-1})\boldsymbol{b}(t_k). \tag{9}$$

Note that this is very similar to $W(t,t_0)$ for the homogeneous solution, except we have replaced $A(t_k)$ with $\boldsymbol{b}(t_k)$. Note also that, in $W(t,t_0)$ we have $k=0$ giving a product of *none* of the A, so that term gives the identity. Here the sum starts from $k=1$, and $k=1$ gives the integral of $\boldsymbol{b}(t_1)$, so we can rewrite the solution as

$$\boldsymbol{v}(t,t_0) = \sum_{k=2}^{\infty} \int_{t_0}^{t} dt_1 \int_{t_0}^{t_1} dt_2 \cdots \int_{t_0}^{t_{k-1}} dt_k \, A(t_1)A(t_2)\cdots A(t_{k-1})\boldsymbol{b}(t_k) + \int_{t_0}^{t} dt_1 \, \boldsymbol{b}(t_1). \tag{10}$$

To see that this is the correct form of the solution, one can use

$$\begin{aligned} \dot{\boldsymbol{x}}_P(t) &= \dot{\boldsymbol{v}}(t,t_0) \\ &= \sum_{k=2}^{\infty} \int_{t_0}^{t} dt_2 \cdots \int_{t_0}^{t_{k-1}} dt_k \, A(t)A(t_2)\cdots A(t_{k-1})\boldsymbol{b}(t_k) + \boldsymbol{b}(t) \end{aligned}$$

$$= A(t)\sum_{k=1}^{\infty}\int_{t_0}^{t} dt_1 \int_{t_0}^{t_1} dt_2 \cdots \int_{t_0}^{t_{k-1}} dt_k\, A(t_1)\cdots A(t_{k-1})\boldsymbol{b}(t_k) + \boldsymbol{b}(t)$$
$$= A(t)\boldsymbol{x}_P + \boldsymbol{b}(t)\,, \tag{11}$$

which is the form of the differential equation. Truncating the solution, we have

$$\boldsymbol{v}_K(t,t_0) \coloneqq \sum_{k=1}^{K}\int_{t_0}^{t} dt_1 \int_{t_0}^{t_1} dt_2 \cdots \int_{t_0}^{t_{k-1}} dt_k\, A(t_1)A(t_2)\cdots A(t_{k-1})\boldsymbol{b}(t_k)\,. \tag{12}$$

The complete approximate solution, as the sum of the homogeneous and particular solutions with truncations, is then

$$\boldsymbol{x}_K(t) = W_K(t,t_0)\boldsymbol{x}(t_0) + \boldsymbol{v}_K(t,t_0). \tag{13}$$

Defining $b_{\max} = \max_t \|\boldsymbol{b}(t)\|$, then the error due the truncation of the Dyson series is

$$\begin{aligned}\|\boldsymbol{x}_K(t) - \boldsymbol{x}(t)\| &\leq \|W_K(t,t_0) - W(t,t_0)\| \times \|\boldsymbol{x}(t_0)\| \\ &\quad + \left\| \sum_{k=K+1}^{\infty}\int_{t_0}^{t} dt_1 \int_{t_0}^{t_1} dt_2 \cdots \int_{t_0}^{t_{k-1}} dt_k\, A(t_1)A(t_2)\cdots A(t_{k-1})\boldsymbol{b}(t_k)\right\| \\ &\leq \mathcal{O}\left(\frac{(A_{\max}\Delta t)^{K+1}}{(K+1)!}\|\boldsymbol{x}(t_0)\|\right) + \sum_{k=K+1}^{\infty}\frac{A_{\max}^{k-1}\Delta t^k b_{\max}}{k!} \\ &= \mathcal{O}\left(\frac{(A_{\max}\Delta t)^{K+1}}{(K+1)!}\|\boldsymbol{x}(t_0)\| + \frac{A_{\max}^{K}\Delta t^{K+1} b_{\max}}{(K+1)!}\right),\end{aligned} \tag{14}$$

where the second norm in the first line results from $\|\boldsymbol{v}(t,t_0) - \boldsymbol{v}_K(t,t_0)\|$. Again we require that $A_{\max}\Delta t$ is $\mathcal{O}(1)$. In our solution we will discretise the integrals, and to bound the overall error we also bound the error due to the discretisation; that is discussed in Section 4.

This complete expression in Eq. (14) has an extra factor of $\|\boldsymbol{x}(t_0)\|$ because we are considering the error in the state. The error in W being no greater than ϵ implies that the error in the state is no larger than ϵ times the norm of $\boldsymbol{x}$, which is the form we require the bound on the error for our theorem. Appropriately bounding the error in the second term in Eq. (14) can require taking K somewhat larger than in Eq. (7). The choice of K is discussed further in Section 4.

1.3 The time-independent solution

We are also going to look at the case where $A \in \mathbb{C}^{N\times N}$ is time-independent, which gives our usual ODE to solve

$$\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + \boldsymbol{b}. \tag{15}$$

In [3] this was solved by using the Taylor series. Here we apply a similar principle, except that we will encode into the block matrix in a simpler way. Here we summarise the Taylor series solution, then give the encoding into the matrix in the next section.

The solution can easily be seen by substituting a constant A into the solution for the time-varying A. We obtain

$$W_K(t,t_0) = \sum_{k=0}^{K}\frac{(A\Delta t)^k}{k!}, \tag{16}$$

and

$$\boldsymbol{v}_K(t,t_0) = \sum_{k=1}^{K}\frac{(A\Delta t)^{k-1}}{k!}\boldsymbol{b}\,. \tag{17}$$

The error in the solution can be bounded as

$$\|\boldsymbol{x}_K(t) - \boldsymbol{x}(t)\| = \mathcal{O}\left(\frac{(\|A\|\Delta t)^{K+1}}{(K+1)!}\|\boldsymbol{x}(t_0)\| + \frac{\|A\|^K \Delta t^{K+1}\|\boldsymbol{b}\|}{(K+1)!}\right). \tag{18}$$

2 Encoding in linear equations

Next we describe how the Dyson series (or Taylor series) for the solution may be encoded into block matrices which can then be solved via the quantum algorithm for solving linear systems. The preceding section described how the series may be applied to accurately approximate the solution over a short time. In the usual way, for solving the solution over a longer time we divide the total time T into r steps of length $\Delta t = T/r$. The solution for the shorter times is obtained by a series, and all time steps are encoded in a block matrix.

2.1 Time-dependent equations

2.1.1 Encoding

First we describe the more complicated case of the block encoding of the Dyson series solution. This step shares many similarities with techniques for time-independent equations [3] but elements of the block matrix $\mathcal{A}^{-1}$ will need to be computed through integration described in the preceding section. The initial time for the multiple steps will be taken to be zero without loss of generality, since it is always possible to apply a time shift in the definitions. Then we can use notation t_0 for the starting time for individual steps of the Dyson series.

To illustrate the method, we first give the example of three time steps, where the encoding is

$$\begin{bmatrix} \mathbb{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ -V_1 & \mathbb{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & -V_2 & \mathbb{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & -V_3 & \mathbb{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\mathbb{1} & \mathbb{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & -\mathbb{1} & \mathbb{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\mathbb{1} & \mathbb{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(0) \\ \tilde{\boldsymbol{x}}(\Delta t) \\ \tilde{\boldsymbol{x}}(2\Delta t) \\ \tilde{\boldsymbol{x}}(3\Delta t) \\ \tilde{\boldsymbol{x}}(3\Delta t) \\ \tilde{\boldsymbol{x}}(3\Delta t) \\ \tilde{\boldsymbol{x}}(3\Delta t) \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}(0) \\ \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \boldsymbol{v}_3 \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix} \tag{19}$$

where

$$\boldsymbol{v}_m \coloneqq \boldsymbol{v}_K(m\Delta t, (m-1)\Delta t), \tag{20}$$
$$V_m \coloneqq W_K(m\Delta t, (m-1)\Delta t), \tag{21}$$

and $\tilde{\boldsymbol{x}}$ indicates the approximate solution for $\boldsymbol{x}$. The top row (numbered as 0 here) sets the initial condition and rows 1 to 3 give steps of the solution for time Δt as described by Eq. (13). These rows all describe forward evolution as

$$\tilde{\boldsymbol{x}}(m\Delta t) = V_m \boldsymbol{x}((m-1)\Delta t) + \boldsymbol{v}_m \tag{22}$$

for $m \in \{1, 2, 3\}$. In rows 4 to 6, the system is constant in time, $\tilde{\boldsymbol{x}}(m\Delta t) = \boldsymbol{x}((m-1)\Delta t)$. Including rows after the final evolution steps might appear unnecessary, but these additional rows will boost the success probability of the resulting quantum algorithm [2]. Note that $\boldsymbol{v}_m$ and V_m involve multiple integrals over times, which would need to be addressed via the block encoding which is described in Section 3. This is a major departure from

the method for time-independent differential equations from [3], where it was possible to encode the terms of the sum via extra lines in the block matrix.

In general, we can define the system of linear equations as

$$\mathcal{A}\mathcal{X} = \mathcal{B}, \tag{23}$$

where $\mathcal{A}$ is a block matrix, $\mathcal{A} \in \mathbb{C}^{NR\times NR}$ and $\mathcal{X}$ and $\mathcal{B}$ are vectors $\mathcal{X}, \mathcal{B} \in \mathbb{C}^{NR}$. Here R is the number of time steps, including those steps at the end where the solution is held constant. Taking $R - r \propto r$ will give a success probability roughly corresponding to the square of the amplitude of the solution. If the solution has not decayed too much, then this success probability will give an acceptable overhead to the complexity.

The individual blocks in $\mathcal{A}, \mathcal{X}, \mathcal{B}$ can be given explicitly as

$$\mathcal{B}_m = \begin{cases} \boldsymbol{x}(0), & m = 0 \\ \boldsymbol{v}_m, & 0 < m \leq r \\ 0, & m > r \end{cases} \tag{24}$$

$$\mathcal{X}_m = \begin{cases} \boldsymbol{x}(0), & m = 0 \\ \tilde{\boldsymbol{x}}(m\Delta t), & 0 < m \leq r \\ \tilde{\boldsymbol{x}}(r\Delta t), & m > r \end{cases} \tag{25}$$

$$\mathcal{A}_{mn} = \begin{cases} \mathbb{1}, & m = n \\ -V_n, & (m = n+1) \wedge (n \leq r) \\ -\mathbb{1}, & (m = n+1) \wedge (n > r) \\ 0, & \text{otherwise.} \end{cases} \tag{26}$$

It is easily checked that this is a general form of the example given above. By construction, this a lower bidiagonal matrix which will be be useful for computing its properties.

2.1.2 Condition number

The complexity of solving the system of linear equations is proportional to the condition number of $\mathcal{A}$. To bound the condition number of $\mathcal{A}$, we just need to bound the norm of $\mathcal{A}$ and its inverse. The norm of $\mathcal{A}$ is easily bounded as $\mathcal{O}(1)$, but the norm of the inverse will depend on how much the approximate solution $\tilde{\boldsymbol{x}}$ can grow as compared to the initial condition $\boldsymbol{x}(0)$ and driving $\boldsymbol{v}_m$.

This means that the condition number can be well-bounded provided the differential equations are stable. That can be obtained provided the real parts of the eigenvalues of $A(t)$ are non-positive. This was the approach applied in [3], where a diagonalisation was used so the complexity was also dependent on the condition number of the matrix that diagonalises A. It has been pointed out in [14] that the complexity of this approach can be bounded in a number of other ways which do not depend on the diagonalisability of A.

As discussed in [14], a useful way to describe the stability of the differential equation is in terms of the logarithmic norm of $A(t)$, which can be given as the maximum eigenvalue of $[A(t) + A^\dagger(t)]/2$. A standard property of the logarithmic norm $\mu(A(t))$ is that

$$\|W(t, t_0)\| \leq \exp\left(\int_{t_0}^{t} \mu(A(t))\right). \tag{27}$$

This means that, if the logarithmic norm is non-positive, then $\|W(t, t_0)\| \leq 1$. That is, this approach can be used to bound the norm of the time-ordered exponential, in a similar

way as the exponential is bounded in the time-independent case in [14]. In the following we will require that the logarithmic norm is non-positive for stability, though other conditions bounding the norm of the time-ordered exponential can be used.

Starting with $\mathcal{A}$ from Eq. (26), it can be separated into a sum of the identity (the main diagonal) and a matrix which is just the $-V_n$ and $\mathbb{1}$ on the offdiagonal. Using the triangle inequality, the norm can therefore be upper bounded as

$$\|\mathcal{A}\| \leq 1 + \max_m \|V_m\|. \tag{28}$$

Here $\max_m \|V_m\|$ is obtained because the spectral norm of the block-diagonal matrix can be given as the maximum of the spectral norms of the blocks. The maximum is over the values of m from 1 to M. Now, V_m is defined in terms of W_K which is an approximation of the exact evolution over time Δt. We will require that the overall solution is given to accuracy $\epsilon < 1$, so the norm of W_K cannot deviate by more than ϵ from the norm for W, which is upper bounded by 1 according to our stability requirement on the logarithmic norm. That means $\max_m \|V_m\| = \mathcal{O}(1)$, and so $\|\mathcal{A}\| = \mathcal{O}(1)$. An alternative bound that does not depend on the stability is

$$\begin{aligned} \|V_m\| &= \|W_K(m\,\Delta t, (m-1)\Delta t)\| \\ &\leq \left\| \sum_{k=0}^{K} \frac{(A_{\max}\Delta t)^k}{k!} \right\| \\ &< \|\exp(A_{\max}\Delta t)\|. \end{aligned} \tag{29}$$

We choose the number of steps such that $A_{\max}\Delta t = \mathcal{O}(1)$, which again gives the upper bound $\|\mathcal{A}\| = \mathcal{O}(1)$.

Next, we bound the norm of the inverse. Since $\mathcal{A}$ is a lower bidiagonal matrix, it has the explicit form of the inverse [13]

$$(\mathcal{A}^{-1})_{mn} = \begin{cases} \mathbb{1}, & m = n \\ \prod_{\ell=n}^{m-1} V_\ell, & (m > n) \wedge (m \leq r+1) \wedge (n \leq r) \\ \prod_{\ell=n}^{r} V_\ell, & (m > n) \wedge (m > r+1) \wedge (n \leq r) \\ \mathbb{1}, & (m > n) \wedge (n > r) \\ 0, & m < n. \end{cases} \tag{30}$$

In our example for $r = 3$, $\mathcal{A}^{-1}$ takes the form

$$\mathcal{A}^{-1} = \begin{bmatrix} \mathbb{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ V_1 & \mathbb{1} & 0 & 0 & 0 & 0 & 0 \\ V_2V_1 & V_2 & \mathbb{1} & 0 & 0 & 0 & 0 \\ V_3V_2V_1 & V_3V_2 & V_3 & \mathbb{1} & 0 & 0 & 0 \\ V_3V_2V_1 & V_3V_2 & V_3 & \mathbb{1} & \mathbb{1} & 0 & 0 \\ V_3V_2V_1 & V_3V_2 & V_3 & \mathbb{1} & \mathbb{1} & \mathbb{1} & 0 \\ V_3V_2V_1 & V_3V_2 & V_3 & \mathbb{1} & \mathbb{1} & \mathbb{1} & \mathbb{1} \end{bmatrix}. \tag{31}$$

Again we can use the triangle inequality. We express $\mathcal{A}^{-1}$ as a sum of matrices, each of which contains one of the diagonals. For each of the block diagonal matrices, the spectral norm is given by the maximum spectral norm of the blocks on the diagonal. More explicitly, we expand $\mathcal{A}^{-1}$ in the sum of block-diagonal matrices

$$\mathcal{A}^{-1} = \sum_{k=0}^{R-1} \mathcal{A}_k^{\text{inv}} \tag{32}$$

where

$$(\mathcal{A}_k^{\text{inv}})_{mn} = \begin{cases} \prod_{\ell=n}^{\min(m-1,r)} V_\ell, & m = n+k \\ 0, & m < n, \end{cases} \tag{33}$$

and we take the convention that a product with no factors gives the identity. We have by the triangle inequality

$$\|\mathcal{A}^{-1}\| \leq \sum_{k=0}^{R-1} \|\mathcal{A}_k^{\text{inv}}\|. \tag{34}$$

Then

$$\|\mathcal{A}_k^{\text{inv}}\| \leq \max_n \left\| \prod_{\ell=n}^{\min(n+k-1,r)} V_\ell \right\|, \tag{35}$$

so

$$\begin{aligned} \|\mathcal{A}^{-1}\| &\leq R \max_k \|\mathcal{A}_k^{\text{inv}}\| \\ &\leq R \max_{m \geq n} \left\| \prod_{\ell=n}^{m-1} V_\ell \right\|. \end{aligned} \tag{36}$$

The bound in (36) can be interpreted as R times the norm of the largest block of $\mathcal{A}^{-1}$. This will be an identity or some products of V_ℓs corresponding to an approximation of the time-evolution operator over a period of time within the interval $[t_0, T]$. We will choose the parameters such that the approximation is within ϵ of the exact operator, and so the norm is within ϵ of the exact operator. According to our condition on the stability in terms of the logarithmic norm, the time-ordered exponential has its spectral norm upper bounded by 1. The product of V_ℓ gives the solution of the homogeneous equation approximated using a Dyson series, and we will choose the parameters such that the error in this solution is no more than ϵ. This means that the error in the spectral norm of the product is also no more than ϵ, and is upper bounded by $1+\epsilon$. Hence the spectral norm of the inverse is upper bounded as

$$\|\mathcal{A}^{-1}\| \leq R(1+\epsilon) = \mathcal{O}(R), \tag{37}$$

where we have used the fact that we are considering complexity for small ϵ. Therefore the condition number of $\mathcal{A}$ is

$$\kappa_{\mathcal{A}} = \|\mathcal{A}\| \times \|\mathcal{A}^{-1}\| = \mathcal{O}(R). \tag{38}$$

2.2 Time-independent equations

The encoding of the time-dependent case is exactly the same of the independent case, except that now the V_m are independent of m, so we can write for example

$$\mathcal{A} = \begin{bmatrix} \mathbb{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ -V & \mathbb{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & -V & \mathbb{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & -V & \mathbb{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\mathbb{1} & \mathbb{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & -\mathbb{1} & \mathbb{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\mathbb{1} & \mathbb{1} \end{bmatrix}, \tag{39}$$

where now we have

$$V = W_K(m\Delta t, (m-1)\Delta t) = \sum_{k=0}^{K} \frac{(A\Delta t)^k}{k!}, \tag{40}$$

which is independent of m. Similarly, the $\boldsymbol{v}_m$ are replaced with

$$\boldsymbol{v} = \sum_{k=1}^{K} \frac{A^{k-1}\Delta t^k}{k!}\boldsymbol{b}\,. \tag{41}$$

We can write the general form in a similar way as for the time-dependent case, except that the $V_m, \boldsymbol{v}_m$ are replaced with the time-independent $V, \boldsymbol{v}$.

In exactly the same way as for the time-dependent case, the spectral norm can be upper bounded by expressing the matrix as a sum of block-diagonal matrices and using the triangle inequality

$$\|\mathcal{A}\| \leq 1 + \|V\|. \tag{42}$$

Again $\|V\|$ will be an accurate approximation of the exponential, which is upper bounded by 1 according to the stability criterion, and so $\|\mathcal{A}\| = \mathcal{O}(1)$. Similarly to the time-dependent case, the spectral norm of $\mathcal{A}^{-1}$ can be upper bounded by using the triangle inequality on the explicit expression for the inverse. Again the stability condition guarantees that the norm of the exponential is no larger than 1, and the condition on the approximation of the solution being with error ϵ implies that the norm of the powers of V is within ϵ of 1. Since there is a sum over R of these norms, we again have $\mathcal{A}^{-1} = \mathcal{O}(R)$ and so

$$\kappa_{\mathcal{A}} = \|\mathcal{A}\| \times \|\mathcal{A}^{-1}\| = \mathcal{O}(R)\,. \tag{43}$$

3 Block encoding

3.1 Time-dependent block encoding

Here we address how to give a block encoding of the matrix in order to apply the quantum linear equation solver. The block encoding is not trivial, because the blocks we have given above are composed of multiple integrals. For our result we just assume that the matrix $A(t)$ is given by a block encoding, rather than considering how it would specifically be done for a particular encoding (such as sparse matrix oracles). Similarly, we will assume that there is a block encoding for a preparation of the state corresponding to $\boldsymbol{b}(t)$ as well as the initial state $\boldsymbol{x}_0$. We assume that the block encoding can be given the time register as a quantum input. The overall complexity is then in terms of the number of calls to these block encodings.

More specifically, given the target system of dimension N, a time register of dimension N_t and ancillas of dimension N_A, N_b, there are unitaries U_A, U_b, and U_x such that

$$_A\langle 0|\, U_A\, |0\rangle_A\, |n_t\rangle = \frac{1}{\lambda_A} A(t)\, |n_t\rangle\,, \tag{44}$$

$$_b\langle 0|\, U_b\, |0\rangle_b\, |n_t\rangle\, |0\rangle_s = \frac{1}{\lambda_b}\, |n_t\rangle\, |\boldsymbol{b}(t)\rangle_s\,, \tag{45}$$

$$U_x\, |0\rangle_s = \frac{1}{\lambda_x}\, |\boldsymbol{x}_0\rangle_s\,, \tag{46}$$

where n_t is a binary encoding of the time t, the subscripts A, b on the state indicate the ancillas for the block encodings of $A(t)$ and $\boldsymbol{b}(t)$, and the subscript s indicates the target system. The operator $A(t)$ acts upon the target system, which is omitted in the first line for simplicity. The quantities $\lambda_A, \lambda_b, \lambda_x$ are the factors for the block encoding and are assumed to be known. The value of λ_x is just that needed to ensure normalisation. The value of λ_b also ensures normalisation, but since the norm of $\boldsymbol{b}(t)$ can vary over time we

define the oracle in terms of a block encoding so that λ_b can be taken to be independent of time. It would also be possible to define an oracle with a unitary preparation and time-dependent λ_b, but that would make the later algorithms much more complicated. We apply the standard encoding of the vector in amplitudes of the state so, for example,

$$|\boldsymbol{x}(t)\rangle = \sum_{n=0}^{N-1} x_n(t) |n\rangle , \tag{47}$$

for computational basis states $|n\rangle$. Note that this state is not defined in a normalised way, which is why we have, for example, division by λ_x above. We also use the standard assumption that the block-encoding unitaries can be applied in a controlled way and we can also apply their inverses.

There are standard methods for combining block encodings of matrices we will use. When we are multiplying operations in a block encoding, the general procedure is that if matrices A and B are block encoded as

$$\langle 0| U |0\rangle = \frac{A}{\lambda_A}, \qquad \langle 0| V |0\rangle = \frac{B}{\lambda_B}, \tag{48}$$

then we have

$$\langle 0| \langle 0| VU |0\rangle |0\rangle = \frac{BA}{\lambda_A \lambda_B}. \tag{49}$$

The understanding is that U and V are acting on different ancillas. That is, we need the ancilla space for both when multiplying. This means that if A and B are block encoded with respective λ-values of λ_A and λ_B then the λ-value for the product is $\lambda_A \lambda_B$.

In adding operations, we would build a controlled operation that performs

$$C = |0\rangle \langle 0| \otimes U + |1\rangle \langle 1| \otimes V \tag{50}$$

controlled by a qubit ancilla. If we aim to block encode $aA + bB$ (with positive a, b), then we can use a state preparation operation P such that

$$P |0\rangle = \frac{1}{\sqrt{a\lambda_A + b\lambda_B}} \left(\sqrt{a\lambda_A} |0\rangle + \sqrt{b\lambda_B} |1\rangle \right). \tag{51}$$

Then we obtain

$$\langle 0| \langle 0| P^\dagger C P |0\rangle |0\rangle = \frac{1}{a\lambda_A + b\lambda_B} (aA + bB). \tag{52}$$

The first P puts the qubit ancilla in a superposition, then C applies either U or V controlled on that ancilla, and $P^\dagger$ inverts the preparation. The resulting λ-value is then $a\lambda_A + b\lambda_B$, so we are just applying the same arithmetic to determine the λ-values as for the operators. If we want negative weights in a or b then the sign can be applied using C, and the resulting λ-value is $|a|\lambda_A + |b|\lambda_B$.

These primitives mean that whenever we have a polynomial in something that is block encoded we can construct a block encoding for the polynomial. The value of λ is determined by using exactly the same arithmetic as in the polynomial for the operators, except we take the absolute values of any weights. This reasoning also holds for discretised integrals. The integral can be discretised into a sum, and the value of λ for the discretised integral can be determined using the same arithmetic for the λ-values. In particular, we use ancilla registers with times to give an approximation of the time integral in the Dyson series, in a similar way as was done for Hamiltonian simulation.

Here we want to block encode an operation of the following form, given for the example of three time steps

$$\mathcal{A} = \begin{bmatrix} \mathbb{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ -V_1 & \mathbb{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & -V_2 & \mathbb{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & -V_3 & \mathbb{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\mathbb{1} & \mathbb{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & -\mathbb{1} & \mathbb{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\mathbb{1} & \mathbb{1} \end{bmatrix}. \tag{53}$$

This matrix can be written in the form

$$\mathcal{A} = \begin{bmatrix} \mathbb{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbb{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbb{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbb{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbb{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbb{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbb{1} \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbb{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbb{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbb{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbb{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbb{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbb{1} & 0 \end{bmatrix} \begin{bmatrix} V_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & V_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & V_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbb{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbb{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbb{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbb{1} \end{bmatrix}. \tag{54}$$

Here there are two operations that are trivial to implement. One is the identity, and the other is an increment on the time register. It is not unitary, because the top row is zero. It can easily be block encoded by incrementing the register in a non-modular way, and using the carry qubit. That is, recall that the block encoding involves a projection onto the $|0\rangle$ state, so if the carry qubit is flipped to $|1\rangle$ then that part is eliminated in the block encoding.

The difficult operation to block encode is the one with V_m on the diagonal. To block encode that matrix, we use the intermediate matrix

$$\mathcal{A}(\delta t) = \begin{bmatrix} A(\delta t) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & A(\Delta t + \delta t) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & A(2\Delta t + \delta t) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{55}$$

where δt is used to index an offset between 0 and Δt. This matrix can be block encoded using the block encoding of A with a time register as input. This may be achieved simply by using a number of time intervals for each Δt that is a power of 2. Then we may use the qubits encoding δt, and the qubits encoding the line in the block matrix, together to give the time input for the block encoding of $A(t)$. For the zeros in the lower-right of $A(\delta t)$ we can flip an ancilla qubit to eliminate that part in the block encoding. That is most easily performed when M is a power of 2 as well, so a single qubit will flag the lower-right part of the matrix.

Then we use this matrix as input to a truncated Dyson series of the form

$$\sum_{k=0}^{K} \int_{t_0}^{t_0+\Delta t} dt_1 \int_{t_0}^{t_1} dt_2 \cdots \int_{t_0}^{t_{k-1}} dt_k \, \mathcal{A}(t_1)\mathcal{A}(t_2)\cdots\mathcal{A}(t_k). \tag{56}$$

In order to block encode the Dyson series, the same procedure as in [12] or [15] can be used. To be more specific on the complexity, there are K time registers which are generated in equal superposition states and then sorted via a sorting network (as is needed for quantum algorithms). The sort has an optimal complexity of $\mathcal{O}(K \log K)$ steps, each of which has a complexity corresponding to the number of bits used to store δt. Using M for the number of these time steps, the number of bits is $\log M$, and so the gate complexity is $\mathcal{O}(K \log K \log M)$. The block encoding of each of the K applications of $\mathcal{A}(\delta t)$ is used, which has complexity of K calls to the block encoding of $A(t)$.

If the inequality testing approach of [15] is used then the complexity of the preparation of the time register is $\mathcal{O}(K \log M)$, because there are $K-1$ inequality tests on registers of size $\log M$. Despite this, in practice it is preferable to use the sorting approach because it provides an improved constant factor in the complexity (which is ignored here in the order scaling).

The preparation of the register with values of k can be performed simply using a unary representation and controlled rotations. The precision of the truncated Dyson series needs to be $\mathcal{O}(\epsilon/r)$, and since there are K rotations each needs precision $\mathcal{O}(\epsilon/(Kr))$, and therefore has complexity $\mathcal{O}(\log(Kr/\epsilon))$. That complexity is no larger than $\mathcal{O}(\log(K)\log(r/\epsilon))$, so multiplying by K for the K rotations gives $\mathcal{O}(K \log(K)\log(r/\epsilon))$.

It is also possible to perform the preparation of k using the improved approach of [18] based on inequality testing. That yields similar complexity for the order scaling because the number of bits needed is $\mathcal{O}(K \log K)$. In the method of [15] for preparing time registers, the preparation over k only needs an equal superposition, which may be prepared with trivial $\mathcal{O}(\log K)$ complexity [17].

Choosing $\lambda_A \Delta t \leq 1$ (and the sorting approach for time registers), the Dyson series in Eq. (56) is block encoded with a λ-value no more than e. In particular, in Ref. [12] the normalisation factor for the Dyson series sum with times prepared by a sort is given in Eq. (55) of that work. That is for a Hermitian Hamiltonian, but the analysis is unchanged for the general matrix. In the notation of [12], λ is equivalent to λ_A here, and T/r is equivalent to Δt here. That means the normalisation factor (λ-value) is no larger than $e^{\lambda_A \Delta t} \leq e$ with $\lambda_A \Delta t \leq 1$.

This result can also be seen using the principles described above for determining λ-values for block-encodings of polynomials. Equation (56) can be described by using weights of $1/k!$ then a time-ordering of the values of t_j. Using the same arithmetic for the λ-values as for the operators, we obtain the value of λ

$$\sum_{k=0}^{K} \frac{1}{k!} \int_{t_0}^{t_0+\Delta t} dt_1 \int_{t_0}^{t_0+\Delta t} dt_2 \cdots \int_{t_0}^{t_0+\Delta t} dt_k \, \lambda_A^k = \sum_{k=0}^{K} \frac{(\lambda_A \Delta t)^k}{k!} < e^{\lambda_A \Delta t} \leq e. \tag{57}$$

There is a further constant factor in the block encoding of the sum for $\mathcal{A}$ in Eq. (54), which may be ignored in our analysis because we are providing the order scaling. That is, in Eq. (54) the matrix with V_j is the Dyson series, which is block encoded with λ-value no larger than e. The identity has λ-value of 1, as does the increment operator (the matrix with identities in the off-diagonal). In the usual way for describing the block encoding of sums and products of operators, the overall λ-value is no larger than $e+1$.

The linear equations to solve are of the form $\mathcal{A}\mathcal{X} = \mathcal{B}$, or

$$\begin{bmatrix} \mathbb{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ -V_1 & \mathbb{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & -V_2 & \mathbb{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & -V_3 & \mathbb{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\mathbb{1} & \mathbb{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & -\mathbb{1} & \mathbb{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\mathbb{1} & \mathbb{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(0) \\ \boldsymbol{x}(\Delta t) \\ \boldsymbol{x}(2\Delta t) \\ \boldsymbol{x}(3\Delta t) \\ \boldsymbol{x}(3\Delta t) \\ \boldsymbol{x}(3\Delta t) \\ \boldsymbol{x}(3\Delta t) \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \boldsymbol{v}_3 \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}. \tag{58}$$

We therefore need to consider the complexity of preparing the vector on the right $\mathcal{B}$, which is composed of $\boldsymbol{x}_0$ which we are given an oracle for, as well as $\boldsymbol{v}_j$ which needs to be constructed by block encoding integrals of $\boldsymbol{b}(t)$. The integral to encode is of the form

$$\boldsymbol{v}_K(t_0 + \Delta t, t_0) = \sum_{k=1}^{K} \int_{t_0}^{t_0+\Delta t} dt_1 \int_{t_0}^{t_1} dt_2 \cdots \int_{t_0}^{t_{k-1}} dt_k \, A(t_1)A(t_2)\cdots A(t_{k-1})\boldsymbol{b}(t_k)\,. \tag{59}$$

The block encoding of this vector can be applied in almost an identical way as for the block encoding of the truncated Dyson series, except the initial block encoding of $A(t_k)$ is replaced with the block encoding of the preparation of $|\boldsymbol{b}(t_k)\rangle$. Therefore, the gate complexity of preparing and sorting the time registers is again $\mathcal{O}\left(K \log K \log M\right)$, or $\mathcal{O}\left(K \log M\right)$ if one were simply to perform inequality testing as in [15]. There are $K-1$ calls to block encodings of $A(t)$, as well as a single call to the block encoding of $\boldsymbol{b}(t_k)$.

For the block encoding of Eq. (59), the choice $\lambda_A \Delta t \leq 1$ means that λ-value of the block encoding is no larger than $(e-1)\lambda_b \Delta t$. As before, the λ-value can be determined using the same arithmetic, and we can use a factor of $1/k!$ followed by a time ordering of t_j. That gives the λ value as

$$\begin{aligned} \sum_{k=1}^{K} \frac{1}{k!} \int_{t_0}^{t_0+\Delta t} dt_1 \int_{t_0}^{t_0+\Delta t} dt_2 \cdots \int_{t_0}^{t_0+\Delta t} dt_k \, \lambda_A^{k-1}\lambda_b &= \frac{\lambda_b}{\lambda_A} \sum_{k=1}^{K} \frac{(\lambda_A \Delta t)^k}{k!} \\ &< \frac{\lambda_b}{\lambda_A}(e^{\lambda_A \Delta t} - 1) \\ &\leq (e-1)\lambda_b \Delta t\,, \end{aligned} \tag{60}$$

for $\lambda_A \Delta t \leq 1$.

For the complete preparation, it is necessary to prepare $\boldsymbol{x}(0)$ as well. We would initially prepare a state on the time registers

$$\frac{1}{\sqrt{\lambda_x^2 + (e-1)^2 r \lambda_b^2 \Delta t^2}} \left[\lambda_x |0\rangle + (e-1)\lambda_b \Delta t \sum_{m=1}^{r} |m\rangle \right]. \tag{61}$$

Then a preparation of $|\boldsymbol{x}(t_0)\rangle$ controlled on $|0\rangle$ and a preparation of $|\boldsymbol{v}_m\rangle$ controlled on $|m\rangle$ gives a state of the form

$$\frac{1}{\sqrt{\lambda_x^2 + (e-1)^2 r \lambda_b^2 \Delta t^2}} \left[|0\rangle |\boldsymbol{x}(t_0)\rangle + \sum_{m=1}^{r} |m\rangle |\boldsymbol{v}_m\rangle \right]. \tag{62}$$

The amplitude for success is then

$$\sqrt{\frac{\lambda_x^2 + \sum_{m=1}^{r} \|\boldsymbol{v}_m\|^2}{\lambda_x^2 + (e-1)^2 r \lambda_b^2 \Delta t^2}} \geq \frac{\min_m \|\boldsymbol{v}_m\|}{(e-1)\lambda_b \Delta t}. \tag{63}$$

The expression here is in terms of $\boldsymbol{v}_m$, which will be constructed with the truncated series and discretised integrals, but the parameters will be chosen so that the error in each is no more than $\epsilon x_{\max}/r$. As a result

$$\|\boldsymbol{v}_m\| \geq \|\boldsymbol{v}(m\Delta t, (m-1)\Delta t)\| - \epsilon x_{\max}/r\,. \tag{64}$$

It would be expected that the correction here would typically be small enough to ignore. If the norm of $\boldsymbol{b}$ does not vary too much and there are not cancellations in the integral, then the expression is proportional to $b_{\max}\Delta t$. Since it would be expected that λ_b is comparable to $b_{\max}$, in such a case the amplitude for success would be at least a constant. For full generality we use the full expression rather than considering typical cases.

The preparation of $\mathcal{B}$ needs to be performed twice in the walk step used in the solution of the linear equations using the approach in Ref. [9]. That would lead to the square of this amplitude for success in the overall amplitude for this walk step. To avoid this square factor, we can perform amplitude amplification on the state preparation to boost its amplitude to $\Theta(1)$. The number of steps of amplitude amplification will be the inverse of this amplitude, or

$$\mathcal{O}\left(\frac{\lambda_b \Delta t}{\min_m \|\boldsymbol{v}(m\Delta t, (m-1)\Delta t)\| - \epsilon x_{\max}/r}\right). \tag{65}$$

Usually the amplitude will be unknown, but it can be determined at the beginning of the procedure with a logarithmic overhead. That amplitude estimation is considerably less costly than the cost of solving the linear equations, which has an extra factor of the condition number $\kappa_{\mathcal{A}}$.

There is also the initial preparation needed for the time register for the state. This can be performed as follows. First perform a rotation on an ancilla qubit to obtain the appropriate weighting between $m = 0$ and $m = 1, \ldots, r$. If r is a power of 2, then controlled on this qubit being $|0\rangle$ we perform Hadamards on the $\log r$ qubits for the time, giving values from 0 to $r-1$. In the more general case where r is not a power of 2, one can perform a controlled preparation over a number of basis states that is not a power of 2, which has complexity $\mathcal{O}(\log r)$ [17].

We can perform CNOTs from the ancilla qubit to the remaining qubits for the time, so that if this control qubit was $|1\rangle$ then we have all ones. Then adding 1 in a modular way with the ancilla as the most significant bit, if the ancilla was $|1\rangle$ then we get all zeros, and if it was $|0\rangle$ we get values from 1 to r. The complexity of this procedure is $\mathcal{O}(\log(1/\epsilon))$ for the initial rotation, and $\mathcal{O}(\log r)$ for the remaining steps. This complexity can be disregarded because it is smaller than many other parts of the procedure.

3.2 Time-independent block encoding

The time-independent case is substantially simplified over the time-dependent case. First, the block encodings are simplified to

$${}_A\langle 0|\, U_A\, |0\rangle_A = \frac{1}{\lambda_A}, \tag{66}$$

$$U_b\, |0\rangle = \frac{1}{\lambda_b}\, |\boldsymbol{b}\rangle\,, \tag{67}$$

$$U_x\, |0\rangle = \frac{1}{\lambda_x}\, |\boldsymbol{x}_0\rangle\,, \tag{68}$$

where there is now no time register for the input and the preparation of $|\boldsymbol{b}\rangle$ is just via a unitary operation. We now omit the s subscript because there is just the one register.

This means that now $\lambda_b = \|\boldsymbol{b}\|$. Again the block encoding can use Eq. (54), except with V_m replaced with the Taylor series V as given in Eq. (40), and $\boldsymbol{v}_m$ replaced with $\boldsymbol{v}$ given in Eq. (41).

The block encoding proceeds in exactly the same way as before, except that there is no need for the time integrals. This means that we need to prepare a register with k for both the Taylor series and the series for $\boldsymbol{v}$. Again, if $\lambda_A \Delta t \leq 1$, then the Taylor series is block encoded with a factor of at least $1/e$.

We can also circumvent the problem we have in the time-dependent case that there are possible cancellations in $\boldsymbol{v}_m$. That is, we have

$$\begin{aligned}
\|\boldsymbol{v}\| &\geq \|\boldsymbol{b}\|\Delta t \left\{ 1 - \sum_{k=2}^{K} \frac{(\|A\|\Delta t)^{k-1}}{k!} \right\} \\
&> \|\boldsymbol{b}\|\Delta t \left\{ 1 - \sum_{k=2}^{\infty} \frac{(\|A\|\Delta t)^{k-1}}{k!} \right\} \\
&= \|\boldsymbol{b}\|\Delta t \left\{ 1 - \frac{[\exp(\|A\|\Delta t) - (1 + \|A\|\Delta t)]}{\|A\|\Delta t} \right\} \\
&\geq (3 - e)\|\boldsymbol{b}\|\Delta t \,,
\end{aligned} \tag{69}$$

where we have used $\|A\|\Delta t \leq \lambda_A \Delta t \leq 1$. We would then find that the amplitude for success of the state preparation would, using Eq. (63), be given by

$$\sqrt{\frac{\lambda_x^2 + \sum_{m=1}^{r} \|\boldsymbol{v}\|^2}{\lambda_x^2 + (e-1)^2 r \lambda_b^2 \Delta t^2}} \geq \frac{(3-e)\|\boldsymbol{b}\|\Delta t}{(e-1)\lambda_b \Delta t} = \frac{3-e}{e-1} \,. \tag{70}$$

Thus the amplitude for success of the state preparation is at least a constant here, so cannot affect the scaling of the complexity and can be omitted in the $\mathcal{O}$ expressions.

4 Complexity of solution

4.1 Time-dependent complexity

Before giving the overall complexity, we give further discussion of the choice of K. As explained above, the error due to the truncation is given in Eq. (14), and the error in the first term (due to the homogeneous component of the solution) can be appropriately bounded by choosing K as in Eq. (7).

The complete expression for the error in Eq. (14) accounts for the inhomogeneity in the second term. The expression for the inhomogeneous error can also be bounded with λ_A replacing $A_{\max}$ because $\lambda_A \geq A_{\max}$, which gives this term as

$$\mathcal{O}\left(\frac{\lambda_A^K \Delta t^{K+1} b_{\max}}{(K+1)!} \right). \tag{71}$$

For our algorithm we will be choosing Δt such that $\lambda_A \Delta t \leq 1$, which means that this error is bounded as

$$\mathcal{O}\left(\frac{b_{\max}}{(K+1)! \lambda_A} \right). \tag{72}$$

Then we can choose K to make this error no larger than $\epsilon x_{\max}/r$ by taking

$$K = \mathcal{O}\left(\frac{\log(r b_{\max}/(\lambda_A x_{\max} \epsilon))}{\log\log(r b_{\max}/(\lambda_A x_{\max} \epsilon))} \right). \tag{73}$$

With $\Delta t \approx 1/\lambda_A$, we have $r = \mathcal{O}(\lambda_A T)$, which gives

$$K = \mathcal{O}\left(\frac{\log(Tb_{\max}/(x_{\max}\epsilon))}{\log\log(Tb_{\max}/(x_{\max}\epsilon))}\right). \tag{74}$$

Using $r = \mathcal{O}(\lambda_A T)$ in Eq. (7) for the choice of K for the homogeneous error gives

$$K = \mathcal{O}\left(\frac{\log(\lambda_A T/\epsilon)}{\log\log(\lambda_A T/\epsilon)}\right). \tag{75}$$

Since we need to ensure that both error terms in Eq. (14) are appropriately bounded, we should take the maximum of these two scalings of K, which can be expressed as

$$K = \mathcal{O}\left(\frac{\log(\lambda_{Ax} T/\epsilon)}{\log\log(\lambda_{Ax} T/\epsilon)}\right), \tag{76}$$

where

$$\lambda_{Ax} = \max\left(\lambda_A, b_{\max}/x_{\max}\right). \tag{77}$$

We will be measuring the error between pure states via the 2-norm distance, but also need to account for cases where the approximate state generated is not pure. We therefore use the Bures-Wasserstein metric [4]

$$d_{BW}(\rho,\sigma) = \sqrt{\mathrm{Tr}(\rho) + \mathrm{Tr}(\sigma) - 2\mathrm{Tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}}. \tag{78}$$

This measure is for states that are not normalised, which is why it includes the traces. In the case that they are normalised this measure is the Bures distance. For the case where the two states are pure, so $\rho = |\psi\rangle\langle\psi|$ and $\sigma = |\phi\rangle\langle\phi|$, then provided $\langle\psi|\phi\rangle$ is real (and positive) we obtain

$$d_{BW}(\rho,\sigma) = \| \, |\psi\rangle - |\phi\rangle \, \|. \tag{79}$$

Therefore this measure can be regarded as a generalisation to mixed states of the 2-norm distance. Another feature of this measure is that d_{BW}^2 is convex in ρ. This property follows from concavity of the fidelity and square root.

The overall complexity of the quantum algorithm for the time-dependent differential equations can be described as in the following theorem.

Theorem 4.1 *We are given an ordinary linear differential equation of the form*

$$\dot{\boldsymbol{x}}(t) = A(t)\boldsymbol{x}(t) + \boldsymbol{b}(t), \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{80}$$

where $\boldsymbol{b}(t) \in \mathbb{C}^N$ is a vector function of t, $A(t) \in \mathbb{C}^{N\times N}$ is a coefficient matrix with non-positive logarithmic norm, and $\boldsymbol{x}(t) \in \mathbb{C}^N$ is the solution vector as a function of t. The parameters of the differential equation are provided via unitaries U_A, U_b, and U_x with known $\lambda_A, \lambda_b, \lambda_x$ such that

$$_A\langle 0| U_A |0\rangle_A |n_t\rangle = \frac{1}{\lambda_A} A(t) |n_t\rangle, \tag{81}$$

$$_b\langle 0| U_b |0\rangle_b |n_t\rangle |0\rangle_s = \frac{1}{\lambda_b} |n_t\rangle |\boldsymbol{b}(t)\rangle_s, \tag{82}$$

$$U_x |0\rangle_s = \frac{1}{\lambda_x} |\boldsymbol{x}_0\rangle_s. \tag{83}$$

A quantum algorithm can provide an approximation $\hat{\boldsymbol{x}}$ *of the solution* $|\boldsymbol{x}(T)\rangle$ *satisfying* $d_{BW}\left(\hat{\boldsymbol{x}}, |\boldsymbol{x}(T)\rangle\langle\boldsymbol{x}(T)|\right) \leq \epsilon x_{\max}$ *using an average number*

$$\mathcal{O}\left(\mathcal{R}\lambda_A T \log\left(1/\epsilon\right)\right) \qquad \text{calls to } U_b, U_x, \tag{84}$$

$$\mathcal{O}\left(\mathcal{R}\lambda_A T \log\left(1/\epsilon\right) \log\left(\frac{\lambda_{Ax} T}{\epsilon}\right)\right) \qquad \text{calls to } U_A, \tag{85}$$

$$\mathcal{O}\left(\mathcal{R}\lambda_A T \log\left(1/\epsilon\right) \log\left(\frac{\lambda_{Ax} T}{\epsilon}\right)\left[\log\left(\frac{T\mathcal{D}}{\lambda_A \epsilon}\right) + \log\left(\frac{\lambda_A T}{\epsilon}\right)\right]\right) \qquad \text{additional gates,} \tag{86}$$

where $\lambda_{Ax} = \max\left(\lambda_A, b_{\max}/x_{\max}\right)$*, given constants satisfying*

$$\mathcal{R} \geq \frac{x_{\max}}{\|\boldsymbol{x}(T)\| \min_m \|\boldsymbol{v}(m\Delta t, (m-1)\Delta t)\| - \epsilon x_{\max}/(\lambda_A T)} \frac{\lambda_b/\lambda_A}{}, \tag{87}$$

$$\mathcal{D} \geq \left(1 + \frac{T b_{\max}}{\lambda_A x_{\max}}\right) \max_{t\in[0,T]} \|A'(t)\| + \frac{\max_{t\in[0,T]} \|\boldsymbol{b}'(t)\|}{x_{\max}}, \tag{88}$$

$$x_{\max} \geq \max_{t\in[0,T]} \|\boldsymbol{x}(t)\|, \tag{89}$$

$$b_{\max} \geq \max_{t\in[0,T]} \|\boldsymbol{b}(t)\|. \tag{90}$$

Here

$$\boldsymbol{v}(t, t_0) := \sum_{k=1}^{\infty} \int_{t_0}^{t} dt_1 \int_{t_0}^{t_1} dt_2 \cdots \int_{t_0}^{t_{k-1}} dt_k \, A(t_1)A(t_2)\cdots A(t_{k-1})\boldsymbol{b}(t_k). \tag{91}$$

and

$$\Delta t = \frac{T}{\lceil \lambda_A T \rceil}. \tag{92}$$

The condition that the logarithmic norm is non-positive ensures that the differential equation is dissipative. We use the convention that state vectors such as $|\boldsymbol{x}_0\rangle_s$ are the unnormalised form with amplitudes exactly equal to the coefficients of the corresponding complex vector. For the unnormalised solution state $|\hat{\boldsymbol{x}}\rangle$, this is defined in the sense that for the normalised quantum state, there exists some constant that we can multiply it by to give $|\hat{\boldsymbol{x}}\rangle$ that satisfies the accuracy constraint.

We require that we are given upper bounds $\mathcal{R}$, $\mathcal{D}$, and so on, because it may be too difficult to determine the maxima and minima required exactly. For completeness we have given a complicated expression for $\mathcal{R}$ to account for the steps of amplitude amplification needed, but in practice if the solution does not decay significantly, and $\boldsymbol{b}(t)$ does not vary such that it cancels and gives small $\boldsymbol{v}$, then it can be expected that $\mathcal{R} = \mathcal{O}(1)$ and can be ignored in the scaling of the complexity.

For the count of additional gates, we are not allowing arbitrary precision rotations. Instead it would be for a fixed set of gates, such as Toffolis (or T gates) plus Clifford gates. The complexity would be equivalent (up to a constant factor) to the non-Clifford count that is often used in quantifying complexities for algorithms intended for error-corrected quantum computers with magic-state factories.

Proof. The principle we use for the solution is to express the solution in terms of a Dyson series over many time steps encoded in a system of linear equations as shown in Eq. (58). We then use the linear equation solver of Ref. [9] which has complexity $\mathcal{O}\left(\kappa_{\mathcal{A}} \log(1/\epsilon)\right)$, in terms of calls to the block-encoded matrix $\mathcal{A}$ and vector $\mathcal{B}$. We then need to perform amplitude amplification to obtain the solution at the final time from the vector with the solution over all times. In order to determine the complexity, we therefore

need to determine $\kappa_{\mathcal{A}}$, the complexity of block encoding $\mathcal{A}, \mathcal{B}$, and the amplitude of the component of the solution at the final time. We will show that the factor of $\mathcal{R}$ comes from the complexity of that amplitude amplification, as well as amplitude amplification as part of preparing $\mathcal{B}$. Moreover, we will show that $\lambda_A T$ comes from the factor of $\kappa_{\mathcal{A}}$, and $\log(\lambda_{Ax} T/\epsilon)$ comes from the order K used in the Dyson series for block encoding $\mathcal{A}, \mathcal{B}$.

The simplest part is the condition number. The condition number is $\kappa_{\mathcal{A}} = \mathcal{O}(R)$ as in Eq. (38). Here, R is the total number of time steps, including those at the end where the solution is held fixed at the final time. This is the standard approach to ensure that there is a reasonable amplitude for the solution at the final time. We use an equal number of time steps for the time evolution r as those where the solution is held fixed, so that $R = 2r$ and $\kappa_{\mathcal{A}} = \mathcal{O}(r)$. Because $r = T/\Delta t$, to minimise the complexity we should choose Δt as large as possible. The restriction limiting how large Δt can be is that $\lambda_A \Delta t \leq 1$, which we use to ensure that the λ-value for the block encoding of $\mathcal{A}$ is $\mathcal{O}(1)$. That means we should take $\Delta t \approx 1/\lambda_A$ so $r = \lambda_A T$, and therefore $\kappa_{\mathcal{A}} = \mathcal{O}(\lambda_A T)$ as claimed.

The complexity of solving the system of linear equations is $\mathcal{O}(\lambda_A T \log(1/\epsilon_{\text{QLSP}}))$ in terms of calls to block encodings of $\mathcal{A}, \mathcal{B}$, where ϵ_{QLSP} is the allowable error in the solution of the system of linear equations. For the complexity we need to relate that error to the allowable error in the solution.

To achieve that, let us denote the approximate solution obtained at time t by $|\tilde{\boldsymbol{x}}(t)\rangle$, to be compared with the correct solution of the linear equations $|\boldsymbol{x}(t)\rangle$. Because the solution of the linear equations is approximate, the components of the approximate solution vector $\widetilde{\mathcal{X}}$ need not be equal in the second half as they are for the exact solution. We are using $|\tilde{\boldsymbol{x}}(t)\rangle$ for the components of $\widetilde{\mathcal{X}}$, so to account for the unequal components in the notation, we use $|\tilde{\boldsymbol{x}}(t)\rangle$ with values of $t > T$, and similarly for $\boldsymbol{x}$.

When the solution of the linear equations is accurate to within error ϵ_{QLSP}, it means that

$$\begin{aligned}
\sqrt{\sum_{j=r+1}^{R} \| |\tilde{\boldsymbol{x}}(j\Delta t)\rangle - |\boldsymbol{x}(T)\rangle \|^2} &\leq \sqrt{\sum_{j=0}^{R} \| |\tilde{\boldsymbol{x}}(j\Delta t)\rangle - |\boldsymbol{x}(j\Delta t)\rangle \|^2} \\
&\leq \epsilon_{\text{QLSP}} \sqrt{\sum_{j=0}^{R} \| |\boldsymbol{x}(j\Delta t)\rangle \|^2} \\
&\leq \epsilon_{\text{QLSP}}\, x_{\max} \sqrt{R+1}. \qquad (93)
\end{aligned}$$

Therefore, the approximate solution $\hat{\boldsymbol{x}}$ will satisfy

$$\begin{aligned}
d_{BW}\left(\hat{\boldsymbol{x}}, |\boldsymbol{x}(T)\rangle \langle \boldsymbol{x}(T)|\right) &\leq \sqrt{\frac{1}{r} \sum_{j=r+1}^{R} d_{BW}^2 \left(|\tilde{\boldsymbol{x}}(j\Delta t)\rangle \langle \tilde{\boldsymbol{x}}(j\Delta t)|, |\boldsymbol{x}(T)\rangle \langle \boldsymbol{x}(T)|\right)} \\
&= \frac{1}{\sqrt{r}} \sqrt{\sum_{j=r+1}^{R} \| |\tilde{\boldsymbol{x}}(j\Delta t)\rangle - |\boldsymbol{x}(T)\rangle \|^2} \\
&\leq \frac{1}{\sqrt{r}} \epsilon_{\text{QLSP}}\, x_{\max} \sqrt{R+1}. \qquad (94)
\end{aligned}$$

Because we take $R = 2r$ we can take $\epsilon_{\text{QLSP}} \propto \epsilon$ and the desired precision will be obtained. Note that there will be error in other steps in the algorithm so the error in solving the linear equations will be taken to be some fraction of the total allowable error ϵ. As usual in analysis of this type, the constant factor can be omitted when giving complexity scalings. Therefore the complexity of solving the linear equations can be given as $\mathcal{O}(\lambda_A T \log(1/\epsilon))$ in terms of calls to block encodings of $\mathcal{A}, \mathcal{B}$.

The complexity of block encoding $\mathcal{A}$ can be determined from the complexity of block encoding the Dyson series in Eq. (56), with other costs being trivial. That Dyson series uses calls to the oracles U_A, but not U_b, U_x. Because the Dyson series is truncated at order K, it can be block encoded with K calls to U_A, with K as given in Eq. (76). That gives the factor of $\log(\lambda_{Ax} T/\epsilon)$ in the complexity for calls to U_A.

The complexity of block encoding $\mathcal{B}$ is similar, except we also need to account for calls to U_b, U_x, and we also need to account for the complexity of amplitude amplification. The form of the Dyson series used to prepare $\mathcal{B}$ is given in Eq. (59). The block encoding of this Dyson series uses $K-1$ calls to U_A, and a single call to U_b. Moreover, a single call to U_x is used to prepare the component of $\mathcal{B}$ with the initial state. Therefore we have the factor of K for calls to U_A, but *not* to U_b, U_x.

Next, the number of steps of amplitude amplification for preparing the state for $\mathcal{B}$ is given in Eq. (65). Using $\Delta t \propto 1/\lambda_A$ gives the number of steps as

$$\mathcal{O}\left(\frac{\lambda_b/\lambda_A}{\min_m \|\boldsymbol{v}(m\Delta t,(m-1)\Delta t)\| - \epsilon x_{\max}/(\lambda_A T)}\right). \tag{95}$$

That is the second factor in the expression for $\mathcal{R}$.

Next we consider the number of steps of amplitude amplification needed to obtain the solution at the final time. This amplitude amplification is performed on the state obtained from the linear equation solver. The amplitude of the state at the final time is given by

$$\sqrt{\frac{\sum_{j=r+1}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle \, \|^2}{\sum_{j=0}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle \, \|^2}}, \tag{96}$$

which implies an average number of steps of amplitude amplification

$$\mathcal{O}\left(\sqrt{\frac{\sum_{j=0}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle \, \|^2}{\sum_{j=r+1}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle \, \|^2}}\right). \tag{97}$$

For the numerator we obtain

$$\begin{aligned}
\sum_{j=0}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle \, \|^2 &\leq \sum_{j=0}^{R} (\| \, |\boldsymbol{x}(j\Delta t)\rangle \, \| + \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle - |\boldsymbol{x}(j\Delta t)\rangle \, \|)^2 \\
&\leq (R+1)x_{\max}^2 + 2x_{\max} \sum_{j=0}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle - |\boldsymbol{x}(j\Delta t)\rangle \, \| \\
&\quad + \sum_{j=0}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle - |\boldsymbol{x}(j\Delta t)\rangle \, \|^2 \\
&\leq (R+1)x_{\max}^2 + 2x_{\max}\sqrt{R+1} \\
&\quad \times \sqrt{\sum_{j=0}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle - |\boldsymbol{x}(j\Delta t)\rangle \, \|^2 + \epsilon_{\mathrm{QLSP}}^2 \, x_{\max}^2 (R+1)} \\
&\leq (R+1)x_{\max}^2 + 2x_{\max}\epsilon_{\mathrm{QLSP}} \, x_{\max}(R+1) + \epsilon_{\mathrm{QLSP}}^2 \, x_{\max}^2 (R+1) \\
&= (R+1)x_{\max}^2 (1+\epsilon_{\mathrm{QLSP}})^2 \\
&= \mathcal{O}\left(r x_{\max}^2\right).
\end{aligned} \tag{98}$$

For the denominator of Eq. (97) we similarly obtain

$$\sum_{j=r+1}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle \, \|^2 + \sum_{j=r+1}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle - |\boldsymbol{x}(j\Delta t)\rangle \, \|^2 \geq \sum_{j=r+1}^{R} \| \, |\boldsymbol{x}(j\Delta t)\rangle \, \|^2$$

$$\sqrt{\sum_{j=r+1}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle \, \|^2} + \sqrt{\sum_{j=r+1}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle - |\boldsymbol{x}(j\Delta t)\rangle \, \|^2} \geq \sqrt{\sum_{j=r+1}^{R} \| \, |\boldsymbol{x}(j\Delta t)\rangle \, \|^2}$$

$$\sqrt{\sum_{j=r+1}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle \, \|^2} + \sqrt{R+1}\epsilon_{\mathrm{QLSP}} x_{\max} \geq \sqrt{r}\|\boldsymbol{x}(T)\|$$

$$\sqrt{\sum_{j=r+1}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle \, \|^2} \geq \sqrt{r}\left(\|\boldsymbol{x}(T)\| - \epsilon x_{\max}/2\right), \quad (99)$$

where in the last line we have used ϵ_{QLSP} as suitable fraction of ϵ. Now, provided $\|\boldsymbol{x}(T)\| \geq \epsilon x_{\max}$ we obtain

$$\sum_{j=r+1}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle \, \|^2 \geq \frac{r}{4}\|\boldsymbol{x}(T)\|^2 . \quad (100)$$

If $\|\boldsymbol{x}(T)\| < \epsilon x_{\max}$ then this is a pathological case where the size of the solution is smaller than the precision required in the algorithm. For any output state we can give $\hat{\boldsymbol{x}} = 0$ and the solution will have the desired precision. Therefore, excluding that pathological case we obtain

$$\sqrt{\frac{\sum_{j=0}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle \, \|^2}{\sum_{j=r+1}^{R} \| \, |\tilde{\boldsymbol{x}}(j\Delta t)\rangle \, \|^2}} = \mathcal{O}\left(\frac{\sqrt{r x_{\max}^2}}{\sqrt{r\|\boldsymbol{x}(T)\|^2}}\right)$$

$$= \mathcal{O}\left(\frac{x_{\max}}{\|\boldsymbol{x}(T)\|}\right). \quad (101)$$

The overall factor in the complexity required is this number of steps of amplitude amplification for obtaining the final time, multiplied by the number of steps of amplitude amplification for preparing $\mathcal{B}$ given in Eq. (95). That product is the lower bound for $\mathcal{R}$ given in the statement of the theorem.

A minor subtlety in the use of amplitude amplification on the result of the quantum linear equations algorithm is that the algorithm as given in Ref. [9] uses measurements to ensure the correct result of filtering, which would not be compatible with amplitude amplification. However, it is trivial to just use omit that measurement, and perform the amplitude amplification jointly on the successful result of the filtering and obtaining the final time in the solution state. That introduces no further factors in the complexity.

Hence, we have proven the factor $\mathcal{R}$ in the complexities for U_A and U_b, U_x, with the factor $\lambda_A T$ arising from the factor of $\kappa_{\mathcal{A}}$ in the complexity for solving linear equations, $\log(1/\epsilon)$ coming from the factor in the complexity of solving linear equations, and the factor of $\log(\lambda_{Ax} T/\epsilon)$ for the U_A complexity coming from the order K used in the Dyson series. It just remains to account for the number of additional gates.

The majority of the complexity for the additional gates is in constructing the registers for the time. We need to construct K registers for the time, and sort in order to obtain the correctly ordered time registers. Similarly to the case for Hamiltonian simulation in Ref. [12], the complexity of these steps is then $\mathcal{O}(K \log K \log M)$, where the factor of $K \log K$ is for the number of steps in the sort, and the factor of $\log M$ is for the number of qubits to store each time. Taking K as in Eq. (76), we obtain $K \log K = \mathcal{O}(\log(\lambda_{Ax} T/\epsilon))$.

That accounts for the factor of $\log(\lambda_{Ax} T/\epsilon)$ before the square brackets given for the number of additional gates. Next we consider the contribution to the complexity from the factor of $\log M$. The value of M needs to be chosen to be large enough such that the error from discretisation of the integrals in the Dyson series is no larger than ϵ/r for each of the r time segments. Exactly the same derivation as for the time-dependent Hamiltonian in

Ref. [12] holds, where the result is given in Eq. (58) and the following text of that work. For completeness we give the derivation of the error in Appendix A, which gives

$$\left\| W_K\left(t_\beta, t_\alpha\right) - \widetilde{W}_K\left(t_\beta, t_\alpha\right) \right\| \in \mathcal{O}\left(\frac{(T/r)^2}{M} \max_{t\in[0,T]} \|A'(t)\| \right) \tag{102}$$

where $W_K(t_\alpha, t_\beta)$ is given by Eq. (5), $t_\alpha - t_\beta = T/r$ and

$$\widetilde{W}_K = \sum_{k=0}^{K} \frac{(T/r)^k}{M^k k!} \sum_{j_1,\cdots,j_k=0}^{M-1} \mathcal{T} A(t_{j_k}) \cdots A(t_{j_1}). \tag{103}$$

This expression corresponds to discretising the integral over each time variable to approximate it by a sum with M terms. In Eq. (102) we also have $A'(t)$ which is the time derivative of $A(t)$.

Similarly, it is easy to see that

$$\|\boldsymbol{v}_K\left(t_\beta, t_\alpha\right) - \tilde{\boldsymbol{v}}_K\left(t_\beta, t_\alpha\right)\| \in \mathcal{O}\left(\frac{(T/r)^2}{M} \max_{t\in[0,T]} \|\boldsymbol{b}'(t)\| + \frac{(T/r)^3 b_{\max}}{M} \max_{t\in[0,T]} \|A'(t)\| \right), \tag{104}$$

for the error in the integral for the driving term due to time discretisation of Eq. (9). The first term in Eq. (104) comes from $k = 1$, and the second comes from $k = 2$. See Appendix A for details of the derivation.

Now, because the error in each evolution operator W_K should be no larger than ϵ/r, the expression for the error given in Eq. (102) implies that we should choose

$$M = \Omega\left(\frac{T^2}{r\epsilon} \max_{t\in[0,T]} \|A'(t)\| \right) = \Omega\left(\frac{T}{\lambda_A \epsilon} \max_{t\in[0,T]} \|A'(t)\| \right). \tag{105}$$

Recall that the error in W_K needs to be no larger than ϵ/r, because we multiply this operator with the vector $\boldsymbol{x}$ to give a final error in the vector no larger than $\epsilon x_{\max}$. The error in $\boldsymbol{v}_K$ should be no larger than $\epsilon x_{\max}/r$, because it directly translates to error in $\boldsymbol{x}$. The requirement to bound the first term in Eq. (104) implies that we should take

$$M = \Omega\left(\frac{T}{\lambda_A \epsilon\, x_{\max}} \max_{t\in[0,T]} \|\boldsymbol{b}'(t)\| \right). \tag{106}$$

In order to bound the second term in Eq. (104), we should take

$$M = \Omega\left(\frac{T^2 b_{\max}}{\lambda_A^2 \epsilon\, x_{\max}} \max_{t\in[0,T]} \|A'(t)\| \right). \tag{107}$$

The smallest M satisfying all three conditions (105), (106), and (107), is then

$$M = \mathcal{O}\left(\frac{T\mathcal{D}}{\lambda\epsilon} \right), \tag{108}$$

with our bound on $\mathcal{D}$. Taking the log of this expression gives the first term in square brackets for the number of additional gates in the theorem.

Lastly, we consider the complexity of the rotations used for preparing the k register. As explained above, these give complexity $\mathcal{O}\left(K \log K \log(r/\epsilon)\right)$. Using Eq. (76) for K gives $K \log K = \mathcal{O}\left(\log(\lambda_{Ax} T/\epsilon)\right)$, then $\mathcal{O}\left(\log(r/\epsilon)\right)$ gives another factor of $\mathcal{O}\left(\log(\lambda_A T/\epsilon)\right)$,

which is given as the second term in the square brackets for the gate complexity. In practice it would be expected that this term is smaller.

There are also operations needed on the approximately $\log r$ qubits for preparing the time registers, but this complexity is no more than $\mathcal{O}(K)$ for each time step. That is smaller than the other complexities so can be omitted in the order scaling. □

A similar form of scaling of complexity was given for the spectral method in [5]. The statement of the complexity in that work gave the log factors in Theorem 1 just as polylog, but the true scaling can be seen in their proof in Eq. (8.15). There the complexity is given as proportional to n^4 times a polylog factor coming from solving systems of linear equations. Their quantity n scales as the log of $1/\epsilon$, T, and the derivatives of the solution.

Therefore the approach of [5] gives scaling as the fourth power of these logs, times whatever factor is obtained from the solution of linear systems. In that work the linear equation solver of Childs, Kothari, and Somma [6] was used, which gives a complexity superlinear in $\log(1/\epsilon)$. The approach of [5] can also be used with the optimal solver of [9] with just linear complexity in $\log(1/\epsilon)$, as we do here.

Assuming that optimal solver for a fair comparison, the scaling of the number of calls to a block encoding of $A(t)$ in terms of T and ϵ in [5] is as $\log^4(T/\epsilon)\log(1/\epsilon)$ (with the second log from the solver). In contrast our complexity is $\log(T/\epsilon)\log(1/\epsilon)$. Moreover, [5] has the same $\log^4(T/\epsilon)\log(1/\epsilon)$ complexity in calls to block encodings of $\boldsymbol{b}(t)$, whereas our complexity is greatly improved to just the $\log(1/\epsilon)$ factor from the linear equation solver. That is because $\boldsymbol{b}(t)$ is only needed once in the Dyson series solution.

A further improvement in our approach is that the number of oracle calls is independent of derivatives of $A(t)$ and $\boldsymbol{b}(t)$. In [5] the complexity scales as the fourth power of the log of arbitrary-order derivatives through their variable g'. In our approach there is dependence on derivatives only in the number of additional gates (needed for addressing time registers), and that complexity only depends on the first-order derivatives.

4.2 Time-independent complexity

In the time-independent case the complexity becomes as follows.

Theorem 4.2 *We are given an ordinary linear differential equation of the form*

$$\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + \boldsymbol{b}, \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{109}$$

where $\boldsymbol{b} \in \mathbb{C}^N$ is a vector function of t, $A \in \mathbb{C}^{N\times N}$ is a coefficient matrix with non-positive logarithmic norm, and $\boldsymbol{x}(t) \in \mathbb{C}^N$ is the solution vector as a function of t. The parameters of the differential equation are provided via unitaries U_A, U_b, and U_x such that

$$ {}_A\langle 0| U_A |0\rangle_A = \frac{1}{\lambda_A} A, \tag{110}$$

$$U_b |0\rangle = \frac{1}{\lambda_b} |\boldsymbol{b}\rangle, \tag{111}$$

$$U_x |0\rangle = \frac{1}{\lambda_x} |\boldsymbol{x}_0\rangle. \tag{112}$$

A quantum algorithm can provide an approximation $\hat{\boldsymbol{x}}$ of the solution $|\boldsymbol{x}(T)\rangle$ satisfying $d_{BW}(\hat{\boldsymbol{x}}, |\boldsymbol{x}(T)\rangle\langle\boldsymbol{x}(T)|) \leq \epsilon x_{\max}$ using an average number

$$\mathcal{O}(\mathcal{R}\lambda T \log(1/\epsilon)) \qquad \text{calls to } U_b, U_x, \tag{113}$$

$$\mathcal{O}\left(\mathcal{R}\lambda T \log(1/\epsilon) \log\left(\frac{\lambda_{Ax} T}{\epsilon}\right)\right) \qquad \text{calls to } U_A, \tag{114}$$

$$\mathcal{O}\left(\mathcal{R}\lambda T \log\left(1/\epsilon\right) \log\left(\frac{\lambda_{Ax} T}{\epsilon}\right) \log\left(\frac{\lambda_A T}{\epsilon}\right)\right) \qquad \text{additional gates,} \tag{115}$$

where $\lambda_{Ax} = \max\left(\lambda_A, \|\boldsymbol{b}\|/x_{\max}\right)$*, given constants satisfying*

$$\mathcal{R} \geq \frac{x_{\max}}{\|\boldsymbol{x}(T)\|}, \tag{116}$$

$$x_{\max} \geq \max_{t\in[0,T]} \|\boldsymbol{x}(t)\| . \tag{117}$$

Proof. The proof proceeds in exactly the same way as for the time-dependent case, except for a few minor amendments. First, the state preparation succeeds with probability at least a constant, so the factor from this amplitude does not appear in $\mathcal{R}$. The other difference in the proof is that the number of additional gates is greatly reduced. We no longer need to perform any arithmetic on time registers, so the $\log\left(T\mathcal{D}/\lambda\epsilon\right)$ factor from the time-dependent cost is removed. However, we do still need to perform rotations and controlled rotations to prepare the register with k. As a result, we still have the extra factor of $\log\left(\lambda T/\epsilon\right)$. □

5 Conclusion

We have given a quantum algorithm to solve a time-dependent differential equation in the sense that it outputs a quantum state with amplitudes encoding the solution vector. This is a distinct method than that used for the time-independent case in [3], where different orders of the sum were encoded in successive lines of the block matrix. In our approach the block matrix has each successive line encoding a new time step, and employs a block encoding of the Dyson series in a similar way as for Hamiltonian simulation in [12].

This approach makes the analysis of the complexity simpler than in [3], because it is not necessary to account for the extra lines in the encoding. As in [14], the expression for the complexity is simplified by using a condition on the logarithmic norm of $A(t)$. This approach avoids needing $A(t)$ to be diagonalisable. These techniques also give a significantly simplified result for the complexity in the time-independent case.

The complexity is, up to logarithmic factors, linear in the total evolution time T and the constant λ in the block encoding of $A(t)$. By applying the optimal quantum linear equation solver of [9], we obtain excellent scaling of the complexity in $\log(1/\epsilon)$. The number of calls to encodings of the driving and initial state is linear in $\log(1/\epsilon)$, whereas the number of calls to the block encoding of $A(t)$ is quadratic in $\log(1/\epsilon)$.

The complexity only depends logarithmically on the first derivatives of $A(t)$ and the driving, and only the additional gates depend on these derivatives not the calls to the block encoding. We have expressed the complexity in terms of maxima of these quantities. It is also possible to express the complexity in terms of averages via the approach in [15], though the derivation is considerably more complicated. A bound could also be given in terms of the total variational distance as in [10]. That is useful when there are discontinuities. Those bounds are just from a more careful analysis of the error in the discretisation of the integrals, rather than a different algorithm.

The way the result for the complexity is expressed is complicated by the need to account for the number of steps of amplitude amplification, which in turn depends on the norms of the $\boldsymbol{v}$ vectors. This full expression is to account for pathological cases, which could potentially be ruled out by a more careful analysis to give a simplified expression.

Our result gives a significant improvement in the log factors over the spectral method from [5]. This raises the question of whether our complexity is optimal, or whether further

improvements are possible. We have log factors appearing from the optimal linear equation solver and from the order of the Dyson series, so any further improvement would require a radically different approach.

Acknowledgements

We thank Maria Kieferova for helpful discussions. DWB worked on this project under a sponsored research agreement with Google Quantum AI. DWB is supported by Australian Research Council Discovery Projects DP190102633, DP210101367, and DP220101602.

References

[1] Dominic W. Berry. High-order quantum algorithm for solving linear differential equations. *Journal of Physics A: Mathematical and Theoretical*, 47(10):105301, 2014. DOI: 10.1088/1751-8113/47/10/105301.

[2] Dominic W Berry. High-order quantum algorithm for solving linear differential equations. *Journal of Physics A: Mathematical and Theoretical*, 47(10):105301, 2014. DOI: 10.1088/1751-8113/47/10/105301.

[3] Dominic W. Berry, Andrew M. Childs, Aaron Ostrander, and Guoming Wang. Quantum algorithm for linear differential equations with exponentially improved dependence on precision. *Communications in Mathematical Physics*, 356(3):1057–1081, Dec 2017. ISSN 1432-0916. DOI: 10.1007/s00220-017-3002-y.

[4] Rajendra Bhatia, Tanvi Jain, and Yongdo Lim. On the bures–wasserstein distance between positive definite matrices. *Expositiones Mathematicae*, 37(2):165–191, 2019. ISSN 0723-0869. DOI: https://doi.org/10.1016/j.exmath.2018.01.002.

[5] Andrew M. Childs and Jin-Peng Liu. Quantum spectral methods for differential equations. *Communications in Mathematical Physics*, 375(2):1427–1457, February 2020. DOI: 10.1007/s00220-020-03699-z.

[6] Andrew M. Childs, Robin Kothari, and Rolando D. Somma. Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. *SIAM Journal on Computing*, 46(6):1920–1950, 2017. DOI: 10.1137/16M1087072.

[7] Andrew M. Childs, Jin-Peng Liu, and Aaron Ostrander. High-precision quantum algorithms for partial differential equations. *Quantum*, 5:574, November 2021. DOI: 10.22331/q-2021-11-10-574.

[8] B. D. Clader, B. C. Jacobs, and C. R. Sprouse. Preconditioned quantum linear system algorithm. *Physical Review Letters*, 110:250504, Jun 2013. DOI: 10.1103/PhysRevLett.110.250504.

[9] Pedro C.S. Costa, Dong An, Yuval R. Sanders, Yuan Su, Ryan Babbush, and Dominic W. Berry. Optimal scaling quantum linear-systems solver via discrete adiabatic theorem. *PRX Quantum*, 3:040303, Oct 2022. DOI: 10.1103/PRXQuantum.3.040303.

[10] Di Fang, Lin Lin, and Yu Tong. Time-marching based quantum solvers for time-dependent linear differential equations. *Quantum*, 7:955, March 2023. ISSN 2521-327X. DOI: 10.22331/q-2023-03-20-955.

[11] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15):150502, Oct 2009. ISSN 1079-7114. DOI: 10.1103/physrevlett.103.150502.

[12] Mária Kieferová, Artur Scherer, and Dominic W. Berry. Simulating the dynamics of time-dependent Hamiltonians with a truncated Dyson series. *Physical Review A*, 99 (4):042314, 4 2019. ISSN 2469-9926. DOI: 10.1103/PhysRevA.99.042314.

[13] Emrah Kılıç and Pantelimon Stanica. The inverse of banded matrices. *Journal of Computational and Applied Mathematics*, 237(1):126–135, 2013. DOI: 10.1016/j.cam.2012.07.018.

[14] Hari Krovi. Improved quantum algorithms for linear and nonlinear differential equations. *Quantum*, 7:913, February 2023. ISSN 2521-327X. DOI: 10.22331/q-2023-02-02-913.

[15] Guang Hao Low and Nathan Wiebe. Hamiltonian Simulation in the Interaction Picture. *arXiv:1805.00675*, 2018. DOI: 10.48550/arXiv.1805.00675.

[16] Ashley Montanaro and Sam Pallister. Quantum algorithms and the finite element method. *Physical Review A*, 93:032324, Mar 2016. DOI: 10.1103/PhysRevA.93.032324.

[17] Yuval R. Sanders, Dominic W. Berry, Pedro C. S. Costa, Louis W. Tessler, Nathan Wiebe, Craig Gidney, Hartmut Neven, and Ryan Babbush. Compilation of fault-tolerant quantum heuristics for combinatorial optimization. *PRX Quantum*, 1(2): 020312–020382, 7 2020. DOI: 10.1103/PRXQuantum.1.020312.

[18] Yuan Su, Dominic W. Berry, Nathan Wiebe, Nicholas Rubin, and Ryan Babbush. Fault-tolerant quantum simulations of chemistry in first quantization. *PRX Quantum*, 2:040332, Nov 2021. DOI: 10.1103/PRXQuantum.2.040332.

A Error estimates for time discretisation

First we consider the error in the sum approximation of the integrals in the definition of W_K. We can write W_K as

$$W_K(t_\beta, t_\alpha) = \sum_{k=0}^{K} \frac{1}{k!} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \, \mathcal{T} A(t_1) A(t_2) \cdots A(t_k) \,, \tag{118}$$

where $\mathcal{T}$ reorders the A matrices in descending time order, and we are taking $t_\alpha \in \{0, \Delta t, 2\Delta t, \cdots, T - \Delta t\}$ and $t_\beta = t_\alpha + \Delta t$. The sum approximation of W_K is then

$$\widetilde{W}_K(t_\beta, t_\alpha) = \sum_{k=0}^{K} \frac{\delta t^k}{k!} \sum_{j_1, \cdots, j_k = 0}^{M-1} \mathcal{T} A(t_{j_1}) \cdots A(t_{j_{k-1}}) A(t_{j_k}) \,, \tag{119}$$

where $\delta t = \Delta t / M$. The sum can be rewritten as

$$\widetilde{W}_K(t_\beta, t_\alpha) = \sum_{k=0}^{K} \frac{1}{k!} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \, \mathcal{T} A(t_{j_1}) A(t_{j_2}) \cdots A(t_{j_k}) \,, \tag{120}$$

where t_{j_ℓ} is t_ℓ rounded down to the nearest multiple of δt as

$$t_{j_\ell} = t_\alpha + \delta t \lfloor (t_\ell - t_\alpha)/\delta t \rfloor. \tag{121}$$

This integral form of the sum is obvious, because each t_{j_ℓ} is constant over the interval δt.

Then we can write the error due to the time discretisation as

$$\begin{aligned}
\left\| W_K(t_\beta, t_\alpha) - \widetilde{W}_K(t_\beta, t_\alpha) \right\| &= \left\| \sum_{k=0}^{K} \frac{1}{k!} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \, \mathcal{T} \left[A(t_1) A(t_2) \cdots A(t_k) \right. \right. \\
&\quad \left. \left. - A(t_{j_1}) A(t_{j_2}) \cdots A(t_{j_k}) \right] \right\| \\
&\leq \sum_{k=0}^{K} \frac{1}{k!} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \, \mathcal{T} \left\| A(t_1) A(t_2) \cdots A(t_k) \right. \\
&\quad \left. - A(t_{j_1}) A(t_{j_2}) \cdots A(t_{j_k}) \right\| ,
\end{aligned} \tag{122}$$

where the convention is taken that the same ordering of t_{j_ℓ} is used as for t_ℓ. For each term for a given k, we can upper bound it as

$$\begin{aligned}
&\sum_{\ell=1}^{k} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \mathcal{T} \left\| A(t_1) \cdots A(t_\ell) A(t_{j_{\ell+1}}) \cdots A(t_{j_k}) \right. \\
&\quad \left. - A(t_1) \cdots A(t_{\ell-1}) A(t_{j_\ell}) \cdots A(t_{j_k}) \right\| \\
&\leq \sum_{\ell=1}^{k} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \mathcal{T} \left\| A(t_1) \cdots \left[A(t_\ell) - A(t_{j_\ell}) \right] A(t_{j_{\ell+1}}) \cdots A(t_{j_k}) \right\| \\
&= \sum_{\ell=1}^{k} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \mathcal{T} \left\| A(t_1) \cdots \left[\int_{t_{j_\ell}}^{t_\ell} ds \, A'(s) \right] A(t_{j_{\ell+1}}) \cdots A(t_{j_k}) \right\| \\
&\leq A_{\max}^{k-1} \sum_{\ell=1}^{k} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \int_{t_{j_\ell}}^{t_\ell} ds \left\| A'(s) \right\|
\end{aligned}$$

$$\leq A_{\max}^{k-1} \max_{s\in[t_\alpha,t_\beta]} \|A'(s)\| \sum_{\ell=1}^{k} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \int_{t_{j_\ell}}^{t_\ell} ds$$
$$\leq A_{\max}^{k-1} \frac{\delta t\, \Delta t^k}{2} \max_{s\in[t_\alpha,t_\beta]} \|A'(s)\|. \tag{123}$$

We can therefore upper bound the error as

$$\begin{aligned}\left\|W_K(t_\beta,t_\alpha) - \widetilde{W}_K(t_\beta,t_\alpha)\right\| &\leq \sum_{k=1}^{K} A_{\max}^{k-1} \frac{\delta t \Delta t^k}{k!} \max_{s\in[t_\alpha,t_\beta]} \|A'(s)\| \\ &< \delta t\, \Delta t \max_{s\in[t_\alpha,t_\beta]} \|A'(s)\| \sum_{k=1}^{\infty} \frac{[A_{\max}\Delta t]^k}{k!} \\ &= \delta t\, \Delta t \max_{s\in[t_\alpha,t_\beta]} \|A'(s)\| \left[\exp\left(A_{\max}\Delta t\right) - 1\right]. \end{aligned} \tag{124}$$

Because we choose $A_{\max}\Delta t \leq 1$, the error due to the discretisation can be upper bounded as

$$\left\|W_K(t_\beta,t_\alpha) - \widetilde{W}_K(t_\beta,t_\alpha)\right\| = \mathcal{O}\left(\frac{(T/r)^2}{M} \max_{s\in[t_\alpha,t_\beta]} \|A'(s)\|\right). \tag{125}$$

This gives the bound used in Eq. (102).

Next we bound the error of the particular solution Eq. (9) when the truncated Dyson series

$$\boldsymbol{v}_K(t_\beta,t_\alpha) = \sum_{k=1}^{K} \frac{1}{k!} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k\, \mathcal{T} A(t_1)A(t_2)\cdots A(t_{k-1})\boldsymbol{b}(t_k), \tag{126}$$

is approximated by the discretised integrals

$$\tilde{\boldsymbol{v}}_K(t_\beta,t_\alpha) = \sum_{k=2}^{K} \frac{\delta t^k}{k!} \sum_{j_1,\cdots,j_k=0}^{M-1} \mathcal{T} A(t_{j_1})\cdots A(t_{j_{k-1}})\boldsymbol{b}(t_{j_k}) + \frac{\Delta t}{M} \sum_{j=0}^{M-1} \boldsymbol{b}(t_j). \tag{127}$$

Here the time ordering operator $\mathcal{T}$ is used to mean that the times are sorted in ascending order, *not* that the order of A and $\boldsymbol{b}$ is changed, because $\boldsymbol{b}$ must always go on the right.

Again we can express the summation in the form of an integral

$$\tilde{\boldsymbol{v}}_K(t_\beta,t_\alpha) = \sum_{k=1}^{K} \frac{1}{k!} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k\, \mathcal{T} A(t_{j_1})A(t_{j_2})\cdots A(t_{j_{k-1}})\boldsymbol{b}(t_k), \tag{128}$$

with the times t_{j_ℓ} rounded down to the nearest multiple of δt. Now we can upper bound the error in the discretised integrals for each k as

$$\begin{aligned}&\int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k\, \mathcal{T} \|A(t_1)A(t_2)\cdots A(t_{k-1})\boldsymbol{b}(t_{j_k}) - A(t_{j_1})A(t_{j_2})\cdots A(t_{j_{k-1}})\boldsymbol{b}(t_{j_k})\| \\ &\leq \sum_{\ell=1}^{k-1} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k\, \mathcal{T} \|A(t_1)\cdots A(t_\ell)A(t_{j_{\ell+1}})\cdots A(t_{j_{k-1}})\boldsymbol{b}(t_{j_k}) \\ &\quad - A(t_1)\cdots A(t_{j_\ell})\cdots A(t_{j_{k-1}})\boldsymbol{b}(t_{j_k})\| \\ &\quad + \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k\, \mathcal{T} \|A(t_1)\cdots A(t_{k-1})\boldsymbol{b}(t_k) - A(t_1)\cdots A(t_{k-1})\boldsymbol{b}(t_{j_k})\| \end{aligned}$$

$$
\begin{aligned}
&= \sum_{\ell=1}^{k-1} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \, \mathcal{T} \left\| A(t_1) \cdots \int_{t_{j_\ell}}^{t_\ell} ds \, A'(s) A(t_{j_{\ell+1}}) \cdots A(t_{k-1}) \boldsymbol{b}(t_{j_k}) \right\| \\
&\quad + \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \, \mathcal{T} \left\| A(t_1) \cdots A(t_{k-1}) \int_{t_{j_k}}^{t_k} ds \, \boldsymbol{b}'(s) \right\| \\
&\leq A_{\max}^{k-2} b_{\max} \sum_{\ell=1}^{k-1} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \int_{t_{j_\ell}}^{t_\ell} ds \, \|A'(s)\| \\
&\quad + A_{\max}^{k-1} \int_{t_\alpha}^{t_\beta} dt_1 \int_{t_\alpha}^{t_\beta} dt_2 \cdots \int_{t_\alpha}^{t_\beta} dt_k \int_{t_{j_k}}^{t_k} ds \, \|\boldsymbol{b}'(s)\| \\
&\leq A_{\max}^{k-2} b_{\max} \sum_{\ell=1}^{k-1} \frac{\Delta t^k \delta t}{2} \max_{s \in [t_\alpha, t_\beta]} \|A'(s)\| + A_{\max}^{k-1} \frac{\Delta t^k \delta t}{2} \max_{s \in [t_\alpha, t_\beta]} \|\boldsymbol{b}'(s)\|. \qquad (129)
\end{aligned}
$$

Now summing over k we have

$$
\begin{aligned}
&\|\boldsymbol{v}_K(t_\beta, t_\alpha) - \tilde{\boldsymbol{v}}_K(t_\beta, t_\alpha)\| \\
&< \sum_{k=1}^{\infty} A_{\max}^{k-2} b_{\max} \frac{k-1}{k!} \frac{\Delta t^k \delta t}{2} \max_{s \in [t_\alpha, t_\beta]} \|A'(s)\| + \sum_{k=1}^{\infty} A_{\max}^{k-1} \frac{\Delta t^k \delta t}{2k!} \max_{s \in [t_\alpha, t_\beta]} \|\boldsymbol{b}'(s)\| \\
&= \sum_{k=1}^{\infty} A_{\max}^{k-2} b_{\max} \frac{\Delta t^k \delta t}{2(k-1)!} \max_{s \in [t_\alpha, t_\beta]} \|A'(s)\| - \sum_{k=1}^{\infty} A_{\max}^{k-2} b_{\max} \frac{\Delta t^k \delta t}{2k!} \max_{s \in [t_\alpha, t_\beta]} \|A'(s)\| \\
&\quad + \sum_{k=1}^{\infty} A_{\max}^{k-1} \frac{\Delta t^k \delta t}{2k!} \max_{s \in [t_\alpha, t_\beta]} \|\boldsymbol{b}'(s)\| \\
&= \sum_{k=0}^{\infty} A_{\max}^{k-1} b_{\max} \frac{\Delta t^{k+1} \delta t}{2k!} \max_{s \in [t_\alpha, t_\beta]} \|A'(s)\| - b_{\max} \frac{\delta t}{2A_{\max}^2} \max_{s \in [t_\alpha, t_\beta]} \|A'(s)\| \left[\exp\left(A_{\max} \Delta t\right) - 1\right] \\
&\quad + \frac{\delta t}{2A_{\max}} \max_{s \in [t_\alpha, t_\beta]} \|\boldsymbol{b}'(s)\| \left[\exp\left(A_{\max} \Delta t\right) - 1\right] \\
&= b_{\max} \frac{\Delta t \delta t}{2A_{\max}} \max_{s \in [t_\alpha, t_\beta]} \|A'(s)\| \exp\left(A_{\max} \Delta t\right) \\
&\quad - b_{\max} \frac{\delta t}{2A_{\max}^2} \max_{s \in [t_\alpha, t_\beta]} \|A'(s)\| \left[\exp\left(A_{\max} \Delta t\right) - 1\right] \\
&\quad + \frac{\delta t}{2A_{\max}} \max_{s \in [t_\alpha, t_\beta]} \|\boldsymbol{b}'(s)\| \left[\exp\left(A_{\max} \Delta t\right) - 1\right] \\
&= b_{\max} \frac{\delta t}{2A_{\max}^2} \max_{s \in [t_\alpha, t_\beta]} \|A'(s)\| \left[1 - (1 - A_{\max} \Delta t) \exp\left(A_{\max} \Delta t\right)\right] \\
&\quad + \frac{\delta t}{2A_{\max}} \max_{s \in [t_\alpha, t_\beta]} \|\boldsymbol{b}'(s)\| \left[\exp\left(A_{\max} \Delta t\right) - 1\right]. \qquad (130)
\end{aligned}
$$

Now with the choice that $A_{\max} \Delta t \leq 1$, the upper bound on the error becomes

$$
\mathcal{O}\left(b_{\max} \frac{\Delta t^3}{M} \max_{s \in [t_\alpha, t_\beta]} \|A'(s)\| + \frac{\Delta t^2}{M} \max_{s \in [t_\alpha, t_\beta]} \|\boldsymbol{b}'(s)\| \right), \qquad (131)
$$

which is equivalent to the upper bound on the error in Eq. (104).