Semi-global approach for propagation of the time-dependent Schrödinger equation for time-dependent and nonlinear problems

Ido Schaefer, Hillel Tal-Ezer and Ronnie Kosloff

May 1, 2017

Abstract

A detailed exposition of highly efficient and accurate method for the propagation of the time-dependent Schrödinger equation [50] is presented. The method is readily generalized to solve an arbitrary set of ODE's. The propagation is based on a global approach, in which large time-intervals are treated as a whole, replacing the local considerations of the common propagators. The new method is suitable for various classes of problems, including problems with a time-dependent Hamiltonian, nonlinear problems, non-Hermitian problems and problems with an inhomogeneous source term. In this paper, a thorough presentation of the basic principles of the propagator is given. We give also a special emphasis on the details of the numerical implementation of the method. For the first time, we present the application for a non-Hermitian problem by a numerical example of a one-dimensional atom under the influence of an intense laser field. The efficiency of the method is demonstrated by a comparison with the common Runge-Kutta approach.

Contents

1 Introduction

The time-dependent Schrödinger equation (TDSE),

$$\frac{d\psi(t)}{dt} = -\frac{i}{\hbar}\hat{\mathbf{H}}(t)\psi(t) \tag{1}$$

is a central pillar in quantum dynamics. Solution of the equation supplies insight on fundamental quantum processes. For the majority of problems closed form solutions do not exist. An alternative is to develop numerical schemes able to simulate from first principle quantum processes. In the present paper we concentrate on the issue of time-propagation.

We assume that the formal operation $\phi = \hat{\mathbf{H}}\psi$ can be carried out in a matrix vector representation, $\vec{\mathbf{v}} = H\vec{\mathbf{u}}$. Hence, the problem of solving Eq. (1) becomes a special case of the problem of solving a general set of ordinary differential equations (ODE's).

Historically, the common practice in early studies was either to solve Eq. (1) by general methods for solving a set of ODE's, or by methods which were developed particularly for Eq. (1). General solvers for a set of ODE's rely on approximations to a low order Taylor expansion of $\psi(t)$, derived from Eq. (1). This leads to the necessity of a time-step propagation scheme (see Sec. 2.2). The most popular methods for general applications are the Runge-Kutta methods [7]. Another method, which became very popular in early quantum studies, is second order differencing [23].

Commonly, early researchers preferred other methods, which were intended specifically for quantum applications. These methods have conservation properties of certain physical quantities, such as norm or energy. The popular methods are Crank-Nicolson implicit scheme [40] and split operator exponentiation [12] (it is noteworthy that the second order differencing method has also special conservation properties). These methods are also equivalent to an approximation of a low order Taylor expansion of $\psi(t)$, and lead to a time-step scheme. The advantage of these methods over the general methods is questionable, since the overall quality of the obtained solution is not improved over the general methods. The convergence becomes non-uniform—the error is accumulated in the physical quantities which are not conserved by the propagation scheme. In particular, the norm conservation leads to larger accumulation of errors in phase.

All the methods that were mentioned can be classified as *local methods*. They all share the common property of being equivalent to a low order Taylor expansion in time. The Taylor expansion has slow convergence properties, which limit its application for approximation purposes to low orders. This leads to the locality of the solution, and consequently, to the time-step integration scheme. The drawback of a time-step scheme is the accumulation of the errors in each time step, which limits the accuracy for realistic times. The time step Δt is limited by the spectral range of $\hat{\mathbf{H}}$, ΔE. Typically, the propagation process is numerically stable only for time-steps which do not exceed $\Delta t \sim \frac{1}{10}\hbar/\Delta E$. The final accuracy of a Taylor propagation method scales as $O(\Delta t^n)$ where n is the order of the method.

A breakthrough in solving the TDSE was the development of the global Chebyshev propagator. The global Chebyshev propagator solves Eq. (1) for a *time-independent Hamil-*

tonian, $\hat{\mathbf{H}}(t) \equiv \hat{\mathbf{H}}$, without the necessity of a time-step scheme. The whole propagation interval is treated *globally* in a single step. The development of the method was led by the insight that a local time-step integration scheme is unnecessary when integration can be performed analytically. With a time-independent Hamiltonian, Eq. (1) can be directly integrated to yield

$$\psi(t) = \exp\left(-\frac{i}{\hbar}\hat{\mathbf{H}}t\right)\psi(0) \tag{2}$$

The direct computation of this expression becomes highly demanding for large-scale problems (see Sec. 2.3.1); the computation in the Chebyshev propagator is based on a polynomial Chebyshev expansion of the evolution operator, $\hat{\mathbf{U}} = \exp(-\frac{i}{\hbar}\hat{\mathbf{H}}t)$ [48]. The method has uniform convergence and does not accumulate errors. The computational effort scales as $\sim \frac{\Delta E\, t}{2\hbar}$. The Chebyshev scheme outperforms all other methods for problems with time independent Hamiltonian operators [27]. More than thirty years after it was introduced it is still the method of choice for efficient and high accuracy solution to large scale problems with a time-independent Hamiltonian [53].

The case of a non-Hermitian Hamiltonian requires a special care. The global Chebyshev propagator was developed for a Hermitian Hamiltonian. Without modification it is not suitable for non-Hermitian operators. A generalization of Chebyshev are Faber polynomials which enable to enter the complex plane [16, 20, 21]. A different approach for non-Hermitian operators led to the development of Newtonian propagators. The Chebyshev approximation on the real axis is replaced by a Newton interpolation in the complex plane. The first application was the solution of the Liouville–von-Neumann equation [5]. The Newtonian scheme was later implemented also to the Schrödinger equation with absorbing boundary conditions [1]. A comparison of the different schemes and the relation to other propagators [9, 37, 51, 54] has been reviewed [15, 24].

The global schemes mentioned above assume a previous knowledge on the eigenvalue domain of the Hamiltonian. Commonly, such a knowledge is missing, in particular in non-Hermitian problems. In such a case, the global approach can be implemented by the Arnoldi approach, using a restarted Arnoldi algorithm (see, for example, [47]). A paper on this topic has not been published to date.

The focus of the present study is solving the Schrödinger equation for problems in which the Hamiltonian is explicitly time-dependent. Such problems are common in ultrafast spectroscopy, coherent control and high harmonic generation. Another typical complication arises when the Hamiltonian becomes nonlinear, i. e. explicitly depends on the state $\psi(t)$. Mean field approximation typically lead to such equations. Examples are the time-dependent Gross-Pitaevskii approximation [4], time-dependent Hartree [26, 29] and time-dependent DFT [14]. In general, Eq. (1) cannot be integrated analytically for a Hamiltonian with time-dependence or nonlinearity. A less common complication arises when a source term is added to the Schrödinger equation. Such equations can be found in scattering theory [34] and in particular problems in coherent control [35, 41, 42].

The common practice to overcome the explicit time dependence, or nonlinearity, is to resort again to a time-step scheme, which relies on Eq. (2). The propagation interval is

divided into small time-steps, where the Hamiltonian is assumed to be stationary within the time-step. This becomes equivalent to a first order method in time. The result is either a significant increase in computational cost or low accuracy. In our opinion, this is a misuse of the global Chebyshev propagator, which is intended to overcome this very problem. A better scheme is based on the Magnus expansion [28] to correct for time ordering, and a low order polynomial approximation for the exponent of an operator [2, 6, 10, 49]. This leads again to a local scheme, since the radius of convergence of the Magnus expansion is limited [6]. However, the scaling of the error with Δt is improved over the common naive practice. Another local approach with improved scaling is to use a high order splitting method [44].

Attempts have been made to implement the global approach in problems with time-dependent or nonlinear Hamiltonians. A very accurate scheme was developed, based on embedding in a larger space, which includes also time. In the extended space, the problem can be formulated by global means [38]. The drawback was the very high computational cost which scaled as $\Delta E\,\Delta t\,\Delta\omega$ where ΔE is the eigenvalue range of the Hamiltonian, Δt is the time step and $\Delta\omega$ is the bandwidth of the explicit time-dependent function. In addition, the method was not applicable to nonlinear problems.

Another attempt was proposed in [3]. Like in the global Chebyshev propagator, the propagation method is based on the idea of global integration of Eq. (1). A direct integration leads to an integral equation:

$$\psi(t) = \psi(0) - \frac{i}{\hbar}\int_0^t \hat{\mathbf{H}}\left[\psi(\tau),\tau\right]\psi(\tau)\,d\tau \tag{3}$$

The integral is approximated by the expansion of the integrand in time in a truncated Chebyshev series, which can be integrated analytically. This results in a system of equations of $\psi(t)$ in multiple time-points. In the nonlinear case, the system of equations becomes also nonlinear. Seemingly, this replaces the problem of time-propagation with the even more difficult problem of optimization. However, the system can be solved by a relatively simple iterative scheme when the time-interval is sufficiently small. This leads again to a time-step scheme. The hope was that it will be possible to use larger time-steps than other propagation schemes, thus leading to reduction of the error accumulation during propagation. Later it was found that this approach led to extremely small time-steps and a large number of iterations, thus becoming highly inefficient (from our experience, and private communications with the author). The failure of the method clearly lies in the necessity of solving a system of equations, a task which was found to be much more demanding than a local propagation scheme.

The introduction of source terms was followed by the development of a global propagation scheme for inhomogeneous problems [22, 32]. This led to new insight to the problem of the time-dependence or nonlinearity of the Hamiltonian. A new global scheme, based on integration in large time-steps, was first introduced in [33]. The method is based on another integrated version of Eq. (1), in which the $\psi(t)$ dependence in the integral expression is minimized in comparison to (3). Here again, an iterative scheme is used to solve the resulting system of equations. This scheme was a significant improvement to that

introduced in [3]. However, we found that it gave inferior results in comparison to a 4'th order Runge-Kutta scheme (RK4). A drastic improvement was achieved by new insights on the propagation scheme [50]. The improved scheme was demonstrated to be significantly more efficient than the Taylor methods, particularly when high accuracy is required. Quite importantly, the scheme was generalized to solve an arbitrary set of ODE's.

The propagation approach of the new scheme, as well as Ref. [3]), combines global and local elements. Hence, it can be classified as a *semi-global propagation approach*.

The original global Chebyshev propagator [48] was easy to program. This led to fast proliferation with many applications. The new algorithm for explicit time dependence became more involved with three user defined parameters which control the accuracy and efficiency. However, we believe that the vast increase in efficiency is worth the effort of learning and computing the algorithm.

The present paper consolidates the numerical scheme. In addition, the application of the algorithm is extended to non-Hermitian Hamiltonians. Our purpose is to give explicit description of all steps and considerations in the scheme to enable the potential user either to program from scratch or to be able to tailor an existing program to the problem of choice. Although the scheme is more involved than the basic Chebyshev scheme, we hope that the explicit description will lead to proliferation of the method.

2 Theory

2.1 Definition of the problem

Let us rewrite the time-dependent Schrödinger equation in a matrix-vector notation:

$$\frac{d\vec{\mathbf{u}}(t)}{dt} = -iH(t)\vec{\mathbf{u}}(t) \tag{4}$$

where $\vec{\mathbf{u}}(t)$ represents the state, and $H(t)$ is a matrix representing the time-dependent Hamiltonian of the system. (Atomic units are used throughout, so we set $\hbar = 1$.)

In our discussion, we shall consider a generalization of Eq. (4). First, we let H include a dependence on the state vector, $\vec{\mathbf{u}}(t)$, i. e. $H \equiv H(\vec{\mathbf{u}}(t), t)$. This results in a nonlinear equation of motion. In addition, we include an inhomogeneous *source term* $\vec{\mathbf{s}}(t)$. The time-dependent nonlinear inhomogeneous Schrödinger equation reads:

$$\frac{d\vec{\mathbf{u}}(t)}{dt} = -iH(\vec{\mathbf{u}}(t), t)\vec{\mathbf{u}}(t) + \vec{\mathbf{s}}(t) \tag{5}$$

Actually, Eq. (5) has the form of a much more general problem. A general set of ODE's is equivalent to an equation of the following form:

$$\frac{d\vec{\mathbf{u}}(t)}{dt} = \vec{\mathbf{g}}(\vec{\mathbf{u}}(t), t) \tag{6}$$

where $\vec{\mathbf{g}}(\vec{\mathbf{u}}(t),t)$ is an arbitrary vector function of $\vec{\mathbf{u}}(t)$ and t. This can be always rewritten as:

$$\frac{d\vec{\mathbf{u}}(t)}{dt} = G(\vec{\mathbf{u}}(t),t)\vec{\mathbf{u}}(t) + \vec{\mathbf{s}}(t) \tag{7}$$

where $G(\vec{\mathbf{u}}(t),t)$ is a matrix. Hence, the problem of solving Eq. (5) is equivalent to the problem of solving Eq. (7), by setting $G(\vec{\mathbf{u}}(t),t) = -iH(\vec{\mathbf{u}}(t),t)$. As a matter of convenience, we shall use the form of Eq. (7) in our discussion.

The initial condition for the vector state is:

$$\vec{\mathbf{u}}(0) = \vec{\mathbf{u}}_0 \tag{8}$$

We require the solution, $\vec{\mathbf{u}}(t)$, at an arbitrary time t.

2.2 Local approach—Taylor methods

The popular algorithms for solving a general set of ODE's are based on Taylor expansion considerations. In order to illustrate this approach, we will consider the *Euler method* which is the simplest Taylor method.

The Euler method is based on a first order Taylor expansion for approximation of the solution at a close point. The solution at $t = \Delta t$ is approximated by:

$$\vec{\mathbf{u}}(\Delta t) \approx \vec{\mathbf{u}}(0) + \Delta t \frac{d\vec{\mathbf{u}}(0)}{dt} \tag{9}$$

$\vec{\mathbf{u}}(0)$ is given by Eq. (8). $d\vec{\mathbf{u}}(0)/dt$ can be computed by plugging Eq. (8) into Eq. (7). Using a first order approximation, the solution will be of low accuracy, unless Δt is sufficiently small. In order to get an accurate solution far from $t = 0$, we have to march in small time-steps. The solution at $t = 2\Delta t$ is computed in the same way, using $\vec{\mathbf{u}}(\Delta t)$ from Eq. (9), and Eq. (7) for obtaining $d\vec{\mathbf{u}}(\Delta t)/dt$. We continue by repeating this propagation technique until we reach the solution at the final time, which will be denoted as $t = T$. If desired, the accuracy of the solution can be improved by choosing a smaller time-step Δt. Of course, this requires more computational effort.

The Euler method is rarely used, because of its slow convergence properties with the decrement of Δt. The error of the solution in T scales as $O(\Delta t)$. Other Taylor methods are based on higher order expansions. The error of a Taylor method of order n scales as $O(\Delta t^n)$.

The most popular Taylor methods are the *Runge-Kutta methods*. The idea underlying the Runge-Kutta methods is to approximate the Taylor expansion without a direct evaluation of high-order derivatives of $\vec{\mathbf{u}}(t)$. The Taylor expansion is approximated by first order derivative evaluations, using Eq. (7). This approximation preserves the scaling of the error with Δt. For instance, we consider the Runge-Kutta method of the 4'th order (RK4). The

solution at $t = \Delta t$ is approximated by:

$$\begin{aligned}
&\vec{\mathbf{u}}(\Delta t) \approx \vec{\mathbf{u}}(0) + \frac{1}{6}(\vec{\mathbf{k}}_1 + 2\vec{\mathbf{k}}_2 + 2\vec{\mathbf{k}}_3 + \vec{\mathbf{k}}_4) \qquad (10)\\
&\vec{\mathbf{k}}_1 = \Delta t\, \vec{\mathbf{g}}(\vec{\mathbf{u}}(0), 0)\\
&\vec{\mathbf{k}}_2 = \Delta t\, \vec{\mathbf{g}}\left(\vec{\mathbf{u}}(0) + \frac{\vec{\mathbf{k}}_1}{2}, \frac{\Delta t}{2}\right)\\
&\vec{\mathbf{k}}_3 = \Delta t\, \vec{\mathbf{g}}\left(\vec{\mathbf{u}}(0) + \frac{\vec{\mathbf{k}}_2}{2}, \frac{\Delta t}{2}\right)\\
&\vec{\mathbf{k}}_4 = \Delta t\, \vec{\mathbf{g}}(\vec{\mathbf{u}}(0) + \vec{\mathbf{k}}_3, \Delta t)
\end{aligned}$$

Eq. (10) approximates a fourth order Taylor expansion. In our numerical example (Sec. 4) we shall use RK4 as a reference method.

The Taylor approach is based on local considerations—in each time-step, the solution $\vec{\mathbf{u}}(t)$ is approximated using our knowledge on the local behaviour of $\vec{\mathbf{u}}(t)$ at the previous time-point. The information on the behaviour of $\vec{\mathbf{u}}(t)$ is deduced from its derivatives *at the time-point*. For this information to be accurate, it is essential that the time-point in which the solution is to be evaluated is close enough. Hence, it is necessary to propagate in small time-steps. The many time-step propagation scheme results in a large computational effort. Moreover, the error is accumulated during the propagation process. These drawbacks are direct consequences of the locality of the Taylor approach.

Another drawback of the Taylor approach lies in the slow convergence properties of the Taylor series. These reduce the efficiency of this approach when using high order Taylor expansions. The popular Runge-Kutta methods are based on 4'th or 5'th order expansions.

In what follows, we shall accommodate with these problems by developing a more global approach for the task of solving Eq. (7).

2.3 Global approach using closed integrated forms

In order to approach the problem in a global manner, we seek a way to treat the whole time interval of the problem in a single stage. Indeed, Eq. (7) can be solved in a single step in the special cases that it can be integrated analytically. The closed integrated forms in these special cases constitute the basis for the present approach.

2.3.1 Time-independent Hamiltonian

We start from the simplest case with a closed integrated form. The Hamiltonian is time-independent:

$$H(t) \equiv H_0$$

or, equivalently:

$$G(t) \equiv G_0$$

In addition, there is no inhomogeneous term:

$$\vec{\mathbf{s}}(t) \equiv 0$$

Eq. (7) becomes:

$$\frac{d\vec{\mathbf{u}}(t)}{dt} = G_0\vec{\mathbf{u}}(t) \tag{11}$$

This equation, with the initial condition (8), can be integrated directly to yield:

$$\vec{\mathbf{u}}(t) = \exp(G_0 t)\vec{\mathbf{u}}_0 \tag{12}$$

for an arbitrary t. In the special case of the Schrödinger equation, we have the well known result of the situation of stationary dynamics:

$$\vec{\mathbf{u}}(t) = \exp(-iH_0 t)\vec{\mathbf{u}}_0 \tag{13}$$

The problem that arises is that the exponent of the matrix $G_0 t$ cannot be computed directly (unless G_0 is diagonal). One immediate approach is to diagonalize G_0 and compute the function of the matrix in the basis of the eigenvectors of G_0. Then we can write:

$$\vec{\mathbf{u}}(t) = S\exp(Dt)S^{-1}\vec{\mathbf{u}}_0 \tag{14}$$

where D is the diagonalized G_0, and S is the transformation matrix from the eigenvector basis to the original basis. The problem with this approach is that when the dimension of G_0 is large, it becomes infeasible to diagonalize it, because of the high numerical cost of this operation—diagonalization scales as $O(N^3)$, where N is the dimension of the problem.

Usually, Eq. (13) is solved by another approach, which is less demanding numerically. We expand the RHS of Eq. (12) in a polynomial series in G_0. First, we define a function $f(x) = \exp(xt)$, where t is treated as a parameter. Then we approximate it by a truncated polynomial series:

$$f(x) \approx \sum_{n=0}^{K-1} a_n P_n(x) \tag{15}$$

where $P_n(x)$ is a polynomial of degree n, and a_n is the corresponding expansion coefficient. This requires the choice of the set of expansion polynomials $P_n(x)$, and the computation of the corresponding a_n's. Then, we approximate Eq. (12) as:

$$\vec{\mathbf{u}}(t) \approx \sum_{n=0}^{K-1} a_n P_n(G_0)\vec{\mathbf{u}}_0 \tag{16}$$

The expansion (15) has to be accurate in the eigenvalue domain of G_0 in order that the form (16) will be useful (see Appendix B.1).

The RHS of Eq. (16) can be computed by successive matrix-vector multiplications. Matrix-vector multiplications scale just as $O(N^2)$. In many cases, the computational effort

can be reduced further. The direct multiplication of $\vec{\mathbf{u}}_0$ by G_0 can be replaced by the operation of an equivalent linear operator. The operation of the linear operator can be defined by a computational procedure, which may have a lower scaling with N. For instance, in the Fourier grid method (see [23]) the Hamiltonian operation scales as $O(N \ln N)$ only.

An immediate question that arises is how to choose the set of expansion polynomials $P_n(x)$. One might suggest to use the Taylor polynomials, i. e. $P_n(x) = x^n$, and expand $f(x)$ in a Taylor series. However, this would be a poor choice, because of the slow convergence properties of a Taylor series. The reason for the slow convergence lies in the low quality of the Taylor polynomials as expansion functions—as n increases, they are getting closer to be parallel in the function space. In order to attain a fast convergence of the polynomial series with K, an orthogonal set of polynomials should be used. The expansion coefficients a_n are given by a scalar product of the $P_n(x)$'s with $f(x)$.

Usually, $f(x)$ is expanded in a *Chebyshev polynomial* series, or equivalently, by a *Newton interpolation polynomial* at the *Chebyshev points* of the eigenvalue domain. When the Hamiltonian is non-Hermitian, the eigenvalue domain becomes complex, and the Chebyshev approach is not appropriate anymore. Then, the *Arnoldi approach* should be used instead. In Appendix A we present the approximation methods of a function by a Newton interpolation polynomial or a Chebyshev polynomial series. In Appendix B we describe the different approximation methods for the multiplication of a vector by a function of matrix, by Chebyshev or Newton series, or by the Arnoldi approach.

In the Chebyshev or Newton methods, an approximation of degree $K-1$, with K expansion terms, requires $K-1$ matrix-vector multiplications. This is due to the recurrence relations between the expansion polynomials in both methods, as will be described in Appendix B. Similarly, in the Arnoldi approach, K matrix-vector multiplications are required for K expansion terms.

Note that using Eq. (13), the solution is given only at the chosen t, and not at intermediate time points. Usually, it is desirable to follow the whole physical process which leads to the result at the final time, and the intermediate times are also of interest. Actually, the solution at the intermediate time-points can be obtained with a negligible additional computational effort. Let us rewrite Eq. (15) for each of the time-points to be computed:

$$\begin{aligned} f_j(x) &= \exp(xt_j) \qquad j = 1, \ldots, N_{tp} \\ f_j(x) &= \sum_{n=0}^{K-1} a_{n,j} P_n(x) \end{aligned} \tag{17}$$

where N_{tp} is the number of time points, and t_j is the j'th time-point. The only difference between the t_j's is in the definition of the $f_j(x)$'s, and the corresponding expansion coefficients, $a_{n,j}$. The $P_n(x)$'s remain the same. Hence, it is sufficient to compute the $P_n(G_0)\vec{\mathbf{u}}_0$ just once for all the desired time points. The $a_{n,j}$'s are computed for each time-point t_j. The computational effort of the matrix-vector multiplications (or the equivalent linear operations) is much greater than that of the computation of the $a_{n,j}$'s, unless N is very small.

2.3.2 Addition of a source term with a polynomial time-dependence

Now let us add to Eq. (11) a source term:

$$\frac{d\vec{\mathbf{u}}(t)}{dt} = G_0\vec{\mathbf{u}}(t) + \vec{\mathbf{s}}(t) \tag{18}$$

A source term is not very common in quantum applications. Nevertheless, the treatment of the common cases of a time-dependent or nonlinear Hamiltonian relies on the results that will be derived in the present section.

Eq. (18) can be integrated using the *Duhamel principle* which relates the solution for the inhomogeneous equation to that of the corresponding homogeneous equation. Let us denote the evolution matrix for the homogeneous equation (11) by:

$$U_0(t) = \exp(G_0 t) \tag{19}$$

The Duhamel principle states that the solution of Eq. (11) can be written by the means of $U_0(t)$ in the following way:

$$\begin{aligned}\vec{\mathbf{u}}(t) &= U_0(t)\vec{\mathbf{u}}_0 + \int_0^t U_0(t-\tau)\vec{\mathbf{s}}(\tau)\,d\tau \\ &= \exp(G_0 t)\vec{\mathbf{u}}_0 + \int_0^t \exp[G_0(t-\tau)]\vec{\mathbf{s}}(\tau)\,d\tau \\ &= \exp(G_0 t)\vec{\mathbf{u}}_0 + \exp(G_0 t)\int_0^t \exp(-G_0\tau)\vec{\mathbf{s}}(\tau)\,d\tau\end{aligned} \tag{20}$$

Eq. (20) assumes a closed form when the integral in the RHS of Eq. (20),

$$\int_0^t \exp(-G_0\tau)\vec{\mathbf{s}}(\tau)\,d\tau \tag{21}$$

can be performed analytically.

We shall focus on a family of source terms for which (21) assumes a closed form—source terms with a polynomial time-dependence:

$$\vec{\mathbf{s}}(t) = \sum_{m=0}^{M-1} \frac{t^m}{m!}\vec{\mathbf{s}}_m \tag{22}$$

First, we need a closed expression for (21) in this particular case. For convenience, we will discuss the scalar version of (21), without loss of generality:

$$\int_0^t \exp(-z\tau)s(\tau)\,d\tau \tag{23}$$

where z is a complex variable, and $s(t)$ is a scalar function of the form

$$s(t) = \sum_{m=0}^{M-1} \frac{t^m}{m!}s_m \tag{24}$$

After we obtain a closed expression, we will be able to write the RHS of Eq. (20) by the means of multiplication of vectors by functions of the matrix G_0. Finally, we will show that the solution can be written in a modified form, in which the computational effort is much reduced.

First, we discuss a source term of the form:

$$s(t) = \frac{t^m}{m!} s_m \tag{25}$$

Let us define:

$$J_{m+1}(z,t) \equiv \int_0^t \exp(-z\tau)\tau^m \, d\tau, \qquad\qquad m = 0, 1, \ldots \tag{26}$$

which are the integrals that need to be evaluated. We start with the simplest situation, when $m = 0$, and $s(t)$ becomes a constant. In the case that $z \neq 0$ we obtain:

$$J_1(z,t) \equiv \int_0^t \exp(-z\tau) \, d\tau = \frac{1 - \exp(-zt)}{z} \tag{27}$$

When $z = 0$, we obtain:

$$J_1(0,t) = t \tag{28}$$

Now let us consider the case that $m > 0$. If $z \neq 0$, we can evaluate the integral using integration by parts. A simple calculation yields:

$$J_{m+1}(z,t) = -\frac{\exp(-zt)t^m}{z} + \frac{m}{z}\int_0^t \exp(-z\tau)\tau^{m-1} \, d\tau = -\frac{\exp(-zt)t^m}{z} + \frac{m}{z} J_m(z,t) \tag{29}$$

or, equivalently:

$$J_m(z,t) = -\frac{\exp(-zt)t^{m-1}}{z} + \frac{m-1}{z} J_{m-1}(z,t), \qquad\qquad m = 2, 3, \ldots \tag{30}$$

Successive operations of the resulting recursion formula lead to the following expression:

$$J_m(z,t) = \frac{(m-1)!}{z^m}\left[1 - \exp(-zt)\sum_{j=0}^{m-1}\frac{(zt)^j}{j!}\right], \qquad\qquad m = 1, 2, \ldots \tag{31}$$

Note that Eq. (31) applies also for $J_1(z,t)$. In the case that $z = 0$ we have:

$$J_m(0,t) = \frac{t^m}{m}, \qquad\qquad m = 1, 2, \ldots \tag{32}$$

We proceed to the evaluation of the scalar form of Eq. (20):

$$u(t) = \exp(zt)u_0 + \exp(zt)\int_0^t \exp(-z\tau)s(\tau) \, d\tau \tag{33}$$

for a source term of the form (24). We begin with the treatment of the second term in the RHS of Eq. (33). Plugging (24) into this term, we obtain:

$$\exp(zt)\sum_{m=0}^{M-1}\frac{1}{m!}\int_0^t \exp(-z\tau)t^m\,d\tau\,s_m = \exp(zt)\sum_{m=0}^{M-1}\frac{1}{m!}J_{m+1}(z,t)s_m = \sum_{m=0}^{M-1} f_{m+1}(z,t)s_m \tag{34}$$

where we defined:

$$f_m(z,t) \equiv \begin{cases} \frac{1}{z^m}\left[\exp(zt) - \sum_{j=0}^{m-1}\frac{(zt)^j}{j!}\right] & z \neq 0 \\ \frac{t^m}{m!} & z = 0 \end{cases} \qquad m = 1, 2, \ldots \tag{35}$$

The corresponding scalar form of Eq. (20) becomes:

$$u(t) = \exp(zt)u_0 + \sum_{m=0}^{M-1} f_{m+1}(z,t)s_m \tag{36}$$

Let us write Eq. (36) in a prettier way. First, we define the following set of constants:

$$w_m \equiv \begin{cases} u_0 & m = 0 \\ s_{m-1} & 0 < m \leq M \end{cases} \tag{37}$$

Second, we note that the definition (35) can be extended to the case of $m = 0$, using the convention that

$$\sum_{j=L}^{N} b_j = 0, \qquad N < L \tag{38}$$

for arbitrary b_j's. Using this extension of definition, we have:

$$f_0(z,t) = \exp(zt) \tag{39}$$

Then, Eq. (36) becomes:

$$u(t) = \sum_{m=0}^{M} f_m(z,t)w_m \tag{40}$$

We can use the form of Eq. (40) to write an analogous vector solution for Eq. (18):

$$\vec{\mathbf{u}}(t) = \sum_{m=0}^{M} f_m(G_0,t)\vec{\mathbf{w}}_m \tag{41}$$

where the $\vec{\mathbf{w}}_m$'s are defined in an analogous manner to the scalar w_m's.

When we compare Eq. (41) to Eq. (12), it seems that the addition of the source term is quite expensive numerically. In Eq. (12) it is necessary to evaluate just one multiplication of a vector by a function of a matrix. In Eq. (41), it is necessary to perform the same kind

of operation $M+1$ times. Actually, the computational effort can be much reduced, if we rewrite Eq. (41) in a modified form.

Let us return to the corresponding scalar equation, Eq. (40). We are going to show that it can be rewritten using just one of the $f_m(z,t)$ functions. Observing the definition (35), it can be easily seen that

$$f_m(z,t) = z f_{m+1}(z,t) + \frac{t^m}{m!} \tag{42}$$

Eq. (42) implies that a function $f_m(z,t)$ can be expressed using any of the other $f_k(z,t)$ functions. If $k > m$, we need $k-m$ successive operations of Eq. (42) in order to write $f_m(z,t)$ in the terms of $f_k(z,t)$. We obtain:

$$f_m(z,t) = z^{k-m} f_k(z,t) + \sum_{j=m}^{k-1} \frac{t^j}{j!} z^{j-m} \tag{43}$$

Eq. (43) can be applied also to the case of $k = m$, with the summation convention (38).

Using Eq. (43), we can express all the $f_m(z,t)$ functions in Eq. (40) by the function with the largest m, i. e. $f_M(z,t)$. Eq. (40) becomes:

$$\begin{aligned} u(t) &= \sum_{m=0}^{M} \left[z^{M-m} f_M(z,t) w_m + \sum_{j=m}^{M-1} \frac{t^j}{j!} z^{j-m} w_m \right] \\ &= f_M(z,t) \sum_{m=0}^{M} z^{M-m} w_m + \sum_{j=0}^{M-1} \frac{t^j}{j!} \sum_{m=0}^{j} z^{j-m} w_m \\ &= f_M(z,t) v_M + \sum_{j=0}^{M-1} \frac{t^j}{j!} v_j \end{aligned} \tag{44}$$

where we defined:

$$v_j \equiv \sum_{m=0}^{j} z^{j-m} w_m \tag{45}$$

Returning to the vector solution of Eq. (18), we can write an analogous expression:

$$\vec{\mathbf{u}}(t) = f_M(G_0,t) \vec{\mathbf{v}}_M + \sum_{j=0}^{M-1} \frac{t^j}{j!} \vec{\mathbf{v}}_j \tag{46}$$

where

$$\vec{\mathbf{v}}_j \equiv \sum_{m=0}^{j} G_0^{j-m} \vec{\mathbf{w}}_m, \qquad j = 0, 1, \ldots \tag{47}$$

Now, only one function of G_0 appears in the solution. However, the computation of the $\vec{\mathbf{v}}_j$ vectors is still an expensive operation—the computation of the $M+1$ vectors involves M

sums, which require $O(M^2)$ matrix-vector multiplications. The computational effort can be much reduced when we notice that the $\vec{\mathbf{v}}_j$'s satisfy a recursion relation:

$$\vec{\mathbf{v}}_j = G_0 \vec{\mathbf{v}}_{j-1} + \vec{\mathbf{w}}_j, \qquad j = 1, 2, \ldots \tag{48}$$

Using Eq. (48), all the $\vec{\mathbf{v}}_j$'s can be computed by M matrix-vector multiplications only, starting from

$$\vec{\mathbf{v}}_0 = \vec{\mathbf{w}}_0 = \vec{\mathbf{u}}_0 \tag{49}$$

Taking into account also the evaluation of the first term in Eq. (46), we can conclude that the overall computational cost is reduced to $M + K - 1$ matrix-vector multiplications for the Chebyshev or Newton series approximation methods (see Sec. 2.3.1). Similarly, $M + K$ matrix-vector multiplications are required for the Arnoldi approach.

2.4 Approximated solutions based on closed integrated forms

2.4.1 Source term with an arbitrary time-dependence

Let us consider the case of Eq. (18) with a source term $\vec{\mathbf{s}}(t)$ with an arbitrary time-dependence. In general, the integral (21) does not assume a closed form, so a closed solution for Eq. (18) cannot be obtained. In the present approach, we utilize the closed solution for the case of (22) in order to approximate the solution in the general case. The idea is to approximate the general source term by a truncated polynomial series of the form of (22). Then, the solution is approximated by a direct application of Eq. (46).

The approximation of $\vec{\mathbf{s}}(t)$ by the form of (22) requires the computation of the $\vec{\mathbf{s}}_m$ coefficients. The form of Eq. (22) might suggest that we should set $\vec{\mathbf{s}}_m = d^m \vec{\mathbf{s}}(0)/dt^m$, to yield a truncated Taylor series of $\vec{\mathbf{s}}(t)$. However, as has already been mentioned, a Taylor series is a poor tool for approximation purposes. Thus, this approach is not recommended.

A better approach is to approximate $\vec{\mathbf{s}}(t)$ by an orthogonal polynomial set at the first stage:

$$\vec{\mathbf{s}}(t) \approx \sum_{n=0}^{M-1} \vec{\mathbf{b}}_n P_n(t) \tag{50}$$

where the $\vec{\mathbf{b}}_n$'s are computed by a scalar product of the $P_n(t)$'s with $\vec{\mathbf{s}}(t)$. The orthogonal expansion polynomials can be expressed in the terms of the Taylor polynomials:

$$P_n(t) = \sum_{m=0}^{n} q_{n,m} \frac{t^m}{m!} \tag{51}$$

Plugging Eq. (51) into Eq. (50) we obtain:

$$\vec{\mathbf{s}}(t) \approx \sum_{n=0}^{M-1} \sum_{m=0}^{n} q_{n,m} \vec{\mathbf{b}}_n \frac{t^m}{m!} = \sum_{m=0}^{M-1} \left(\sum_{n=m}^{M-1} q_{n,m} \vec{\mathbf{b}}_n \right) \frac{t^m}{m!} \tag{52}$$

Then, the result is equated to the Taylor form,

$$\sum_{m=0}^{M-1}\left(\sum_{n=m}^{M-1} q_{n,m}\vec{\mathbf{b}}_n\right)\frac{t^m}{m!} = \sum_{m=0}^{M-1}\frac{t^m}{m!}\vec{\mathbf{s}}_m \tag{53}$$

to yield the $\vec{\mathbf{s}}_m$ Taylor polynomial coefficients:

$$\vec{\mathbf{s}}_m = \sum_{n=m}^{M-1} q_{n,m}\vec{\mathbf{b}}_n \tag{54}$$

These are in general different from the Taylor expansion coefficients. In this way, we preserve the Taylor polynomial form of Eq. (22), but with the advantage of the fast convergence of an orthogonal polynomial set.

It is recommended to use the Chebyshev polynomials as the $P_n(t)$ set. An equivalent option is to use a Newton interpolation expansion in the Chebyshev points.

We still need a procedure for a systematic computation of the $q_{n,m}$ coefficients. Recursive algorithms can be derived from recursive definitions of different polynomial sets. In appendix C we develop recursive conversion algorithms from Chebyshev and Newton expansions to a Taylor form.

Of course, the expansion (50) should not be confused with the similar expansion (15). The first approximates the function $\vec{\mathbf{s}}(t)$ in *time*, within the time interval of the solution, while the second approximates a function of the matrix G_0 in the *eigenvalue domain of* G_0.

2.4.2 Time-dependent Hamiltonian

Now we shall consider the case of a time-dependent Hamiltonian, $H = H(t)$. For the sake of generality, a source term is included in the equation. We have:

$$\frac{d\vec{\mathbf{u}}(t)}{dt} = -iH(t)\vec{\mathbf{u}}(t) + \vec{\mathbf{s}}(t) \tag{55}$$

or,

$$\frac{d\vec{\mathbf{u}}(t)}{dt} = G(t)\vec{\mathbf{u}}(t) + \vec{\mathbf{s}}(t) \tag{56}$$

In this case, the Duhamel principle cannot be applied directly for obtaining a closed form solution, as in the previous cases (see Sec. 2.3.2). However, we shall see that the results from the previous cases can be utilized for obtaining a procedure which approximates the solution in the present case.

First, it is always possible to split $G(t)$ into a sum of time-dependent and time-independent parts:

$$G(t) = \widetilde{G} + \bar{G}(t) \tag{57}$$

where $\widetilde{G}$ is arbitrary, and

$$\bar{G}(t) \equiv G(t) - \widetilde{G} \tag{58}$$

Let us define:

$$\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t),t) = \vec{\mathbf{s}}(t) + \bar{G}(t)\vec{\mathbf{u}}(t) \tag{59}$$

$\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t),t)$ is a new, extended "source term". Now, Eq. (56) can be written as

$$\frac{d\vec{\mathbf{u}}(t)}{dt} = \widetilde{G}\vec{\mathbf{u}}(t) + \vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t),t) \tag{60}$$

which resembles the form of Eq. (18). The Duhamel principle can be applied to yield:

$$\vec{\mathbf{u}}(t) = \exp(\widetilde{G}t)\vec{\mathbf{u}}_0 + \exp(\widetilde{G}t)\int_0^t \exp(-\widetilde{G}\tau)\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(\tau),\tau)\,d\tau \tag{61}$$

As in the case of Sec. 2.4.1, we can write an approximation of this equality in the form of Eq. (46):

$$\vec{\mathbf{u}}(t) \approx f_M(\widetilde{G},t)\vec{\mathbf{v}}_M + \sum_{j=0}^{M-1}\frac{t^j}{j!}\vec{\mathbf{v}}_j \tag{62}$$

The $\vec{\mathbf{v}}_j$ vectors are computed by expanding $\vec{\mathbf{s}}_{ext}$ in time in the form of (22), and using the resulting coefficients, as in Sec. 2.4.1.

Apparently, this gives nothing—Eq. (61) is an integral equation, and the RHS includes a dependence on $\vec{\mathbf{u}}(t)$ itself, which is still unknown. Consequently, the $\vec{\mathbf{v}}_j$'s also depend on $\vec{\mathbf{u}}(t)$. However, it is possible to utilize this form for obtaining a solution by an *iterative procedure*. First, we *guess* a solution $\vec{\mathbf{u}}_g(t)$ in the desired time interval. Then, we use $\vec{\mathbf{u}}_g(t)$ for the computation of the RHS of the equation. We obtain for the LHS a new approximated solution. It seems reasonable that it should be closer to the actual solution than $\vec{\mathbf{u}}_g(t)$. We can use the improved solution for obtaining a better one by inserting it into the RHS, and so on. This procedure can be continued until the solution converges with a desired accuracy.

This iterative scheme sounds reasonable. However, we have not given a rigorous justification to it. Thus, one might suspect if it should actually work. Experience shows that this iterative process does converge to the solution, given that *the time-interval is sufficiently small*. Thus, the iterative procedure has a *convergence radius*. The size of the convergence radius is problem dependent. When the time interval is larger than the convergence radius, the solution diverges, i. e. it tends to infinity with the number of iterations.

A more rigorous justification to the iterative procedure can be obtained by a convergence analysis. This topic is left for a future paper.

In the case that the time of the desired solution is outside the convergence interval of the algorithm, this procedure cannot be used directly. Instead, we can use a *time-step algorithm*, in a similar manner to the Taylor approach. We divide the time-interval into smaller time-steps, in which the iterative procedure converges. In each time-step we solve the sub-problem of obtaining the solution within the time-step. We use the solution obtained in order to compute $\vec{\mathbf{u}}_g(t)$ for the next time-step, as will be described later.

We see that at the end of the day we still need a time-step propagation, as in the Taylor approach. The advantage of the present approach is that we can use much larger

time-steps, which means that the accumulation of errors and the computational effort can be much reduced. The approach for the computation of each time-step is global, replacing the local considerations of the Taylor approach. However, the algorithm still contains an obvious local element, in the sense that the solution is computed separately in each local time-step. Hence, we can call this approach a *"semi-global approach"*.

Note that the definition of the $\vec{\mathbf{s}}_j$'s, the $\vec{\mathbf{w}}_j$'s and corresponding $\vec{\mathbf{v}}_j$'s is different for each time-step. Let us denote the k'th time-point by t_k. The time-interval of the k'th time-step is $[t_k, t_{k+1}]$. The $\vec{\mathbf{s}}_j$'s, the $\vec{\mathbf{w}}_j$'s and the $\vec{\mathbf{v}}_j$'s in the k'th time-step will be denoted by $\vec{\mathbf{s}}_{k,j}$, $\vec{\mathbf{w}}_{k,j}$ and $\vec{\mathbf{v}}_{k,j}$, accordingly. The $\vec{\mathbf{s}}_{k,j}$'s are computed using the expansion (50) of $\vec{\mathbf{s}}_{ext}(t)$ *within the k'th time-interval*. The $\vec{\mathbf{w}}_{k,j}$'s are defined as:

$$\vec{\mathbf{w}}_{k,j} \equiv \begin{cases} \vec{\mathbf{u}}(t_k) & j = 0 \\ \vec{\mathbf{s}}_{k,j-1} & 0 < j \leq M \end{cases} \tag{63}$$

The $\vec{\mathbf{v}}_{k,j}$'s are computed accordingly by the recursion

$$\begin{aligned} \vec{\mathbf{v}}_{k,0} &= \vec{\mathbf{u}}(t_k) \\ \vec{\mathbf{v}}_{k,j} &= \widetilde{G}\vec{\mathbf{v}}_{k,j-1} + \vec{\mathbf{w}}_{k,j}, \qquad\qquad j = 1, 2, \ldots \end{aligned} \tag{64}$$

The solution within the k'th time-step is:

$$\vec{\mathbf{u}}(t) = f_M(\widetilde{G}, t - t_k)\vec{\mathbf{v}}_{k,M} + \sum_{j=0}^{M-1} \frac{(t-t_k)^j}{j!}\vec{\mathbf{v}}_{k,j}, \qquad\qquad t \in [t_k, t_{k+1}] \tag{65}$$

Two questions remained open: How $G(t)$ should be split (see Eq. (57)), and how the guess solution $\vec{\mathbf{u}}_g(t)$ should be chosen. We begin with the first question. From a physical point of view, it seems that a natural choice in many problems is to split the Hamiltonian in the following way:

$$H(t) = H_0 + V(t) \tag{66}$$

where H_0 is the unperturbed Hamiltonian and $V(t)$ is a time-dependent perturbation. If, in addition, $\vec{\mathbf{s}}(t) \equiv 0$, Eq. (61) becomes

$$\vec{\mathbf{u}}(t) = \exp(-iH_0 t)\vec{\mathbf{u}}_0 - i\int_0^t \exp[-iH_0(t-\tau)]V(\tau)\vec{\mathbf{u}}(\tau)\,d\tau \tag{67}$$

which has a striking resemblance to the well-known expression of the first-order time-dependent perturbation theory (see, for example, [11, Chapter XIII]). Indeed, the expressions become identical by the replacement of $\vec{\mathbf{u}}(\tau)$ in the integral by $\vec{\mathbf{u}}_0$. However, although this option is appealing in the sense of the directness of the physical interpretation, it needn't be the best option from a numerical point of view. The result may converge faster with other choices of splitting.

A more educated choice of splitting comes to us when we realize that the weak point of the algorithm lies in the point where we "cheat". This point is the treatment of the

$\vec{\mathbf{u}}(t)$ dependent "source term", $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t),t)$, as an inhomogeneous, $\vec{\mathbf{u}}(t)$ independent term. We should choose the splitting in a way that minimizes the size of the $\vec{\mathbf{u}}(t)$ dependence in $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t),t)$. Hence, $\bar{G}(t)$ should be as small as possible. Consider the k'th time-step, in the time-interval $[t_k, t_{k+1}]$. For the sake of generality, we consider also a non-equidistant time-grid. Let us denote: $\Delta t_k = t_{k+1} - t_k$. Usually, Δt_k can be assumed to be small, by the requirements of the convergence radius of the algorithm. Hence, we can assume that $G(t)$ does not change much during the time-interval. Then it becomes reasonable to choose the following splitting:

$$\widetilde{G} = G\left(t_k + \frac{\Delta t_k}{2}\right) \tag{68}$$

$$\bar{G}(t) \equiv G(t) - G\left(t_k + \frac{\Delta t_k}{2}\right) \qquad t \in [t_k, t_{k+1}] \tag{69}$$

Obviously, the splitting of Eqs. (68)-(69) is time-step dependent, unlike the splitting of Eq. (66).

The choice of the guess solution $\vec{\mathbf{u}}_g(t)$ is lead by two contradicting considerations: On one hand, we need a sufficiently accurate starting point for the iterative process. On the other hand, we require that it can be obtained with minimal amount of extra numerical effort. One obvious choice, which requires no extra numerical effort, is the zero'th order approximation—within the k'th time-step interval, $[t_k, t_{k+1}]$, the guess solution is

$$\vec{\mathbf{u}}_g(t) \equiv \vec{\mathbf{u}}(t_k) \qquad t \in [t_k, t_{k+1}] \tag{70}$$

However, this approximation is of low accuracy. Actually, a very accurate approximation can be obtained from an *extrapolation* of the solution in the previous time-step. All we need to do is to use Eq. (65) for the previous time-step to compute the solution in the current time-step. We obtain the following approximated guess solution:

$$\vec{\mathbf{u}}_g(t) = f_M(\widetilde{G}, t - t_{k-1})\vec{\mathbf{v}}_{k-1,M} + \sum_{j=0}^{M-1} \frac{(t-t_{k-1})^j}{j!}\vec{\mathbf{v}}_{k-1,j}, \qquad t \in [t_k, t_{k+1}] \tag{71}$$

This solution approximates the solution in the interval $[t_k, t_{k+1}]$, using information from the previous interval $[t_{k-1}, t_k]$. Note that the second argument of the function $f_M(z,t)$ in Eq. (71) represents a different time interval from that of Eq. (65). As in Eq. (17), the function is expanded in z, and t serves as a parameter. Hence, the functions to be computed are different in the two equations, and new expansion coefficients have to be computed in the new interval. The numerical effort of this operation is negligible in comparison to the matrix-vector multiplications (or the equivalent linear operations), unless the dimension N of the problem is very small. Thus, by Eq. (71) we obtain an accurate guess with a relatively low computational cost.

In the first time-step, $\vec{\mathbf{u}}_g(t)$ can be computed by Eq. (70). More iterations will be required in comparison with the other time-steps, but the overall additional computational effort in a many time-point grid is negligible.

2.4.3 Nonlinear Hamiltonian

Let the Hamiltonian include also a dependence on $\vec{\mathbf{u}}(t)$, i. e. $H \equiv H(\vec{\mathbf{u}}(t), t)$. Now we get to the general case of Eq. (5), or equivalently, Eq. (7). The treatment of this case is completely analogous to that of the linear time-dependent Hamiltonian.

Let us split $G(\vec{\mathbf{u}}(t), t)$ in the following way:

$$G(\vec{\mathbf{u}}(t), t) = \widetilde{G} + \bar{G}(\vec{\mathbf{u}}(t), t) \tag{72}$$

where $\widetilde{G}$ is linear and time-independent. Let us define:

$$\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t) = \vec{\mathbf{s}}(t) + \bar{G}(\vec{\mathbf{u}}(t), t)\vec{\mathbf{u}}(t) \tag{73}$$

The rest of the algorithm is identical to that of Sec. 2.4.2, for the same considerations.

In an analogous manner to the time-dependent linear case, it is recommended to choose the following splitting in the time-step algorithm:

$$\widetilde{G} = G\left[\vec{\mathbf{u}}\left(t_k + \frac{\Delta t_k}{2}\right), t_k + \frac{\Delta t_k}{2}\right] \tag{74}$$

$$\bar{G}(\vec{\mathbf{u}}(t), t) \equiv G(\vec{\mathbf{u}}(t), t) - G\left[\vec{\mathbf{u}}\left(t_k + \frac{\Delta t_k}{2}\right), t_k + \frac{\Delta t_k}{2}\right] \qquad t \in [t_k, t_{k+1}] \tag{75}$$

3 Implementation

3.1 The propagation time-grid

In order to obtain the $\vec{\mathbf{s}}_m$ Taylor like coefficients, we have to know the total time-dependence of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t)$ (see Sec. 2.4.1). Of course, we have no explicit expression for the total time-dependence of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t)$, because of its dependence on $\vec{\mathbf{u}}(t)$. Hence, the time-dependence of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t)$ has to be approximated from several *sampling points* within the time interval. In each sampling point t_l, we need the values of $\vec{\mathbf{u}}(t_l)$, $G(\vec{\mathbf{u}}(t_l), t_l)$ and $\vec{\mathbf{s}}(t_l)$ in order to compute $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t_l), t_l)$. The $\vec{\mathbf{b}}_n$'s from Eq. (50) are obtained from the samplings of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t)$ within the time-interval. Then, the $\vec{\mathbf{s}}_m$'s can be computed from the $\vec{\mathbf{b}}_n$'s as described in Sec. 2.4.1.

It is recommended to choose the *Chebyshev points* within the time-interval as the sampling points. Then, $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t)$ can be expanded in time either by a Chebyshev polynomial expansion, or by a Newton interpolation at the Chebyshev points (see Appendix A). When a Chebyshev polynomial expansion is used, the $\vec{\mathbf{b}}_n$'s from Eq. (50) correspond to the Chebyshev coefficients, that will be denoted by $\vec{\mathbf{c}}_n$, and the polynomials are the Chebyshev polynomials. When a Newton interpolation is used, the $\vec{\mathbf{b}}_n$'s are the divided differences, that will be denoted by $\vec{\mathbf{a}}_n$, and the polynomials are the Newton basis polynomials. In appendix A we describe how the $\vec{\mathbf{a}}_n$'s or the $\vec{\mathbf{c}}_n$'s can be obtained from the samplings at the Chebyshev points.

In the time-step algorithm, the time interval of each time-step is sampled at the Chebyshev points of the interval. Thus, the structure of the time-grid necessary for the propagation is complex; it consists of adjacent time-intervals, each with an internal Chebyshev grid. In order to refer also to the internal grid of each interval, we shall replace the single index notation of the time-grid from Sec. 2.4.2, t_k, by a double index notation, $t_{k,l}$. The first index k refers to the k'th time-interval, where $k = 1, 2, \ldots, N_t$. The second index l indexes the points in the internal Chebyshev grid of each interval, as will be readily seen.

The length of the k'th time-interval is denoted by Δt_k, as in Sec. 2.4.2. For the sake of generality, we will consider also the possibility that the number of Chebyshev points is different for each time-step. The number of Chebyshev points in the k'th interval is denoted by M_k. M_k is also the number of expansion terms for the approximation of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t)$ (see Eq. (50)).

We use the set of boundary including Chebyshev points. In the Chebyshev domain, $[-1, 1]$, the Chebyshev grid is defined as follows (Cf. Appendix A.1.2, Eq. (117); note that the equations of Appendix A are formulated in the terms of the order of the polynomial approximation, which is equivalent to $M_k - 1$):

$$y_{k,l} \equiv -\cos\left(\frac{l\pi}{M_k - 1}\right), \qquad l = 0, 1, \ldots, M_k - 1 \tag{76}$$

In the k'th time-step domain, the Chebyshev points become (Cf. Eq. (118)):

$$t_{k,l} = t_{k,0} + \frac{\Delta t_k}{2}(1 + y_{k,l}) \tag{77}$$

Note that we have:

$$t_{k,M_k-1} = t_{k,0} + \Delta t_k = t_{k+1,0} \tag{78}$$

Eqs. (77), (78) define together the entire time grid, where $t_{1,0}$ and the Δt_k's are given.

3.2 Algorithm

We assume that the initial condition, $\vec{\mathbf{u}}(t_{1,0})$, is given. In addition, it is assumed that the structure of the propagation time-grid (i. e. Δt_k and M_k for each time-step) is known in advance. Alternatively, it is possible to choose it adaptively during the propagation, by an internal procedure. The number of expansion terms for the approximation of $f_{M_k}(\widetilde{G}, t - t_{k,0})\vec{\mathbf{v}}_{M_k}$ (see Eq. (65)) is also supplied by the user. It may depend on k.

The accuracy of the solution is determined by a tolerance parameter, which will be denoted by ϵ. It represents the order of the accepted relative error of the solution. The tolerance parameter is supplied by the user.

The scheme of propagation goes as follows:

1. Set the guess solution of the first time-step in the internal Chebyshev grid (Cf. Eq. (70)):
$$\vec{\mathbf{u}}(t_{1,l}) = \vec{\mathbf{u}}(t_{1,0}), \qquad l = 0, 1, \ldots, M_1 - 1$$

2. for $k = 1$ to N_t

   (a) Set the middle point of the internal grid, $t_{mid} = t_{k,M_k \backslash 2}$, where $\backslash$ denotes integer division.

   (b) $(l = 0, 1, \ldots, M_k - 1, \qquad n = 0, 1, \ldots, M_k - 1, \qquad j = 0, 1, \ldots, M_k)$

   (c) do

      i. Set $\vec{\mathbf{s}}'_{ext} = \vec{\mathbf{s}}(t_{k,l}) + [G(\vec{\mathbf{u}}(t_{k,l}), t_{k,l}) - G(\vec{\mathbf{u}}(t_{mid}), t_{mid})]\vec{\mathbf{u}}(t_{k,l})$ (Cf. Eqs. (73), (75)).

      ii. Use the $\vec{\mathbf{s}}'_{ext}$'s to compute the expansion coefficients of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t)$ in the time-step.

      - *For a Newton interpolation*: Compute the divided differences $\vec{\mathbf{a}}_n$ recursively, as described in Appendix A.1 (relevant equations: (109), (110),(113)). Use $4t_{k,l}/\Delta t_k$ as the sampling points (see Sec. A.1.3), and the corresponding $\vec{\mathbf{s}}'_{ext}$'s as the function values.
      - *For a Chebyshev expansion*: Compute the Chebyshev coefficients $\vec{\mathbf{c}}_n$, as described in Appendix A.2 (relevant equation: (148)). Use the $\vec{\mathbf{s}}'_{ext}$'s as the function values.

      iii. Compute the $\vec{\mathbf{s}}_n$ Taylor-like coefficients recursively from the $\vec{\mathbf{a}}_n$'s or the $\vec{\mathbf{c}}_n$'s, using the conversion schemes described in Appendix C (relevant equations: (223)-(225), (215) for the $\vec{\mathbf{a}}_n$'s, (246)-(250) for the $\vec{\mathbf{c}}_n$'s).

      iv. Compute the $\vec{\mathbf{v}}_j$ vectors recursively from $\vec{\mathbf{u}}_{k,0}$ and the $\vec{\mathbf{s}}_n$'s (see Eqs. (64), (63)), where $\widetilde{G} = G(\vec{\mathbf{u}}(t_{mid}), t_{mid})$ (Cf. Eq. (74)).

      v. Store the current solution at the time-step edge for convergence check, $\vec{\mathbf{u}}_{old} = \vec{\mathbf{u}}(t_{k,M_k-1})$.

      vi. Compute a new solution from the $\vec{\mathbf{v}}_j$'s by the expression (Cf. Eq. (65)):

$$\vec{\mathbf{u}}(t_{k,l}) = f_{M_k}(\widetilde{G}, t_{k,l} - t_{k,0})\vec{\mathbf{v}}_{M_k} + \sum_{j=0}^{M_k-1} \frac{(t_{k,l} - t_{k,0})^j}{j!}\vec{\mathbf{v}}_j$$

      $f_{M_k}(\widetilde{G}, t_{k,l} - t_{k,0})\vec{\mathbf{v}}_{M_k}$ is approximated by one of the methods described in Appendix B (note that the expansion vectors required for the approximation are computed just once for all time points—see Sec. 2.3.1).

      vii. Repeat from step 2(c)i while

$$\frac{\|\vec{\mathbf{u}}(t_{k,M_k-1}) - \vec{\mathbf{u}}_{old}\|}{\|\vec{\mathbf{u}}_{old}\|} > \epsilon$$

   (d) end do

(e) Compute the solution at any desired point $t_p \in [t_{k,0}, t_{k,M_k-1}]$ by

$$\vec{\mathbf{u}}(t_p) = f_{M_k}(\widetilde{G}, t_p - t_{k,0})\vec{\mathbf{v}}_{M_k} + \sum_{j=0}^{M_k-1} \frac{(t_p - t_{k,0})^j}{j!}\vec{\mathbf{v}}_j$$

(f) Set the guess solution for the next time-step; by definition: $\vec{\mathbf{u}}(t_{k+1,0}) = \vec{\mathbf{u}}(t_{k,M_k-1})$ (see Eq. (78)). The guess solution at the rest of the Chebyshev internal points is computed by (Cf. Eq. (71)):

$$\vec{\mathbf{u}}(t_{k+1,m}) = f_{M_k}(\widetilde{G}, t_{k+1,m} - t_{k,0})\vec{\mathbf{v}}_{M_k} + \sum_{j=0}^{M_k-1} \frac{(t_{k+1,m} - t_{k,0})^j}{j!}\vec{\mathbf{v}}_j,$$

$$m = 1, \ldots, M_{k+1} - 1$$

3. end for

The algorithm is sketched schematically in Fig. 1.

3.3 Programming

3.3.1 Numerical stability of the time polynomial expansion

In Eqs. (22), (51), the coefficients of the t polynomials are defined as the coefficients of $t^m/m!$, in analogy to the Taylor expansion form. Accordingly, we obtained in Eq. (46) the $\vec{\mathbf{v}}_m$'s as the coefficients of $t^m/m!$. The $1/m!$ factor decreases very fast as m grows. Consequently, the $\vec{\mathbf{s}}_m$'s, the $q_{n,m}$'s and the $\vec{\mathbf{v}}_m$'s tend to attain huge values. This may lead to numerical instability.

The problem can be solved by the definition of alternative polynomial expansions, in which the coefficients absorb the $1/m!$ factor. The alternative expansions will lead to expressions which are more stable numerically. The source term $\vec{\mathbf{s}}(t)$ is expanded as

$$\vec{\mathbf{s}}(t) \approx \sum_{m=0}^{M-1} t^m \tilde{\vec{\mathbf{s}}}_m \tag{79}$$

where $\tilde{\vec{\mathbf{s}}}_m = \vec{\mathbf{s}}_m/m!$. Accordingly, Eq. (51) is replaced by

$$P_n(t) = \sum_{m=0}^{n} \tilde{q}_{n,m} t^m \tag{80}$$

where $\tilde{q}_{n,m} = q_{n,m}/m!$.

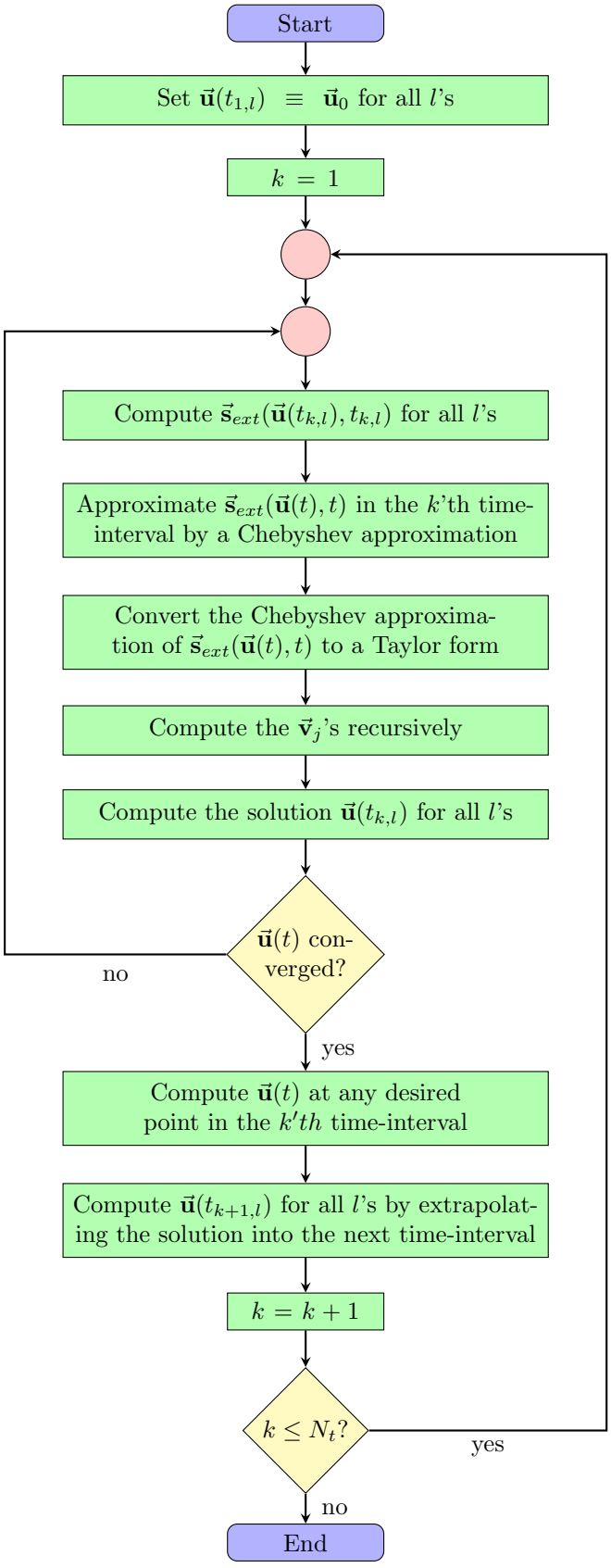

Figure 1: The semi-global propagator algorithm

First we derive the solution equation for a time-independent Hamiltonian. We plug the expansion (79) into Eq. (20). The derivation of the solution is similar to that of Sec. 2.3.2, but less appealing from an aesthetic point of view. We end with the following equation:

$$\vec{\mathbf{u}}(t) = \tilde{f}_M(G_0, t)\vec{\tilde{\mathbf{v}}}_M + \sum_{j=0}^{M-1} t^j \vec{\tilde{\mathbf{v}}}_j \tag{81}$$

where we defined

$$\tilde{f}_m(z,t) \equiv \begin{cases} \frac{m!}{z^m}\left[\exp(zt) - \sum_{j=0}^{m-1}\frac{(zt)^j}{j!}\right] & z \neq 0 \\ t^m & z = 0 \end{cases} \qquad m = 0, 1, \ldots \tag{82}$$

The $\vec{\tilde{\mathbf{v}}}_j$'s are defined recursively in the following way:

$$\begin{aligned} \vec{\tilde{\mathbf{v}}}_0 &= \vec{\mathbf{u}}_0 \\ \vec{\tilde{\mathbf{v}}}_j &= \frac{G_0\vec{\tilde{\mathbf{v}}}_{j-1} + \vec{\tilde{\mathbf{s}}}_{j-1}}{j}, \qquad j = 1, 2, \ldots \end{aligned} \tag{83}$$

Note that we have:

$$\tilde{f}_m(z,t) = m! f_m(z,t) \tag{84}$$

$$\vec{\tilde{\mathbf{v}}}_j = \frac{\vec{\mathbf{v}}_j}{j!} \tag{85}$$

Thus, it can be easily verified that Eq. (81) is equivalent to Eq. (46).

In the case of a time-dependent nonlinear Hamiltonian, we expand $\vec{\mathbf{s}}_{ext}(t)$ as in Eq. (79), and replace G_0 in Eqs. (81), (83), by $\widetilde{G}$.

The computation of the $\tilde{q}_{n,m}$'s is completely analogous to that of the $q_{n,m}$'s. The procedure is given in Appendix C.

In summary, in order to improve the stability of the program, the following changes should be made in the algorithm:

- In step 2(c)iii, the $\vec{\tilde{\mathbf{s}}}_n$'s are computed instead of the $\vec{\mathbf{s}}_n$'s, via the computation of the $\tilde{q}_{n,m}$'s (relevant equations: (226)-(228), (221) for the $\vec{\mathbf{a}}_n$'s, (254)-(258) for the $\vec{\mathbf{c}}_n$'s).

- In step 2(c)iv, the $\vec{\tilde{\mathbf{v}}}_j$'s are computed instead of the $\vec{\mathbf{v}}_j$'s, by the recursion

$$\begin{aligned} \vec{\tilde{\mathbf{v}}}_0 &= \vec{\mathbf{u}}_0 \\ \vec{\tilde{\mathbf{v}}}_j &= \frac{\widetilde{G}\vec{\tilde{\mathbf{v}}}_{j-1} + \vec{\tilde{\mathbf{s}}}_{j-1}}{j}, \qquad j = 1, 2, \ldots \end{aligned}$$

  where $\widetilde{G} = G(\vec{\mathbf{u}}(t_{mid}), t_{mid})$.

- In steps 2(c)vi, 2e and 2f, the solution at the relevant time-points is computed by

$$\vec{\mathbf{u}}(t) = \tilde{f}_M(\widetilde{G}, t - t_{k,0})\vec{\tilde{\mathbf{v}}}_M + \sum_{j=0}^{M-1}(t - t_{k,0})^j \vec{\tilde{\mathbf{v}}}_j \tag{86}$$

3.3.2 The computation of $\tilde{f}_m(z,t)$

The function $f_m(z,t)$, and its variant, $\tilde{f}_m(z,t)$, include the following expression (see Eqs. (35), (82)):

$$\exp(zt) - \sum_{j=0}^{m-1} \frac{(zt)^j}{j!} \tag{87}$$

The sum is just a truncated Taylor expansion of $\exp(zt)$. If zt is small, the difference between $\exp(zt)$ and its truncated expansion becomes extremely small. Often, this results in roundoff errors.

The problem can be solved by an alternative computation of $\tilde{f}_m(z,t)$ for small zt values. Let us expand $\exp(zt)$ from (87) by a Taylor expansion. (87) can be expressed as a "tail" of the Taylor expansion in the following way:

$$\exp(zt) - \sum_{j=0}^{m-1} \frac{(zt)^j}{j!} = \sum_{j=0}^{\infty} \frac{(zt)^j}{j!} - \sum_{j=0}^{m-1} \frac{(zt)^j}{j!} = \sum_{j=m}^{\infty} \frac{(zt)^j}{j!} \tag{88}$$

The expression for $\tilde{f}_m(z,t)$ becomes:

$$\tilde{f}_m(z,t) = m!\, t^m \sum_{j=0}^{\infty} \frac{(zt)^j}{(j+m)!} \tag{89}$$

$\tilde{f}_m(z,t)$ can be computed by truncating the sum in (89). The Taylor expansion converges very slowly. Hence, the expansion should be truncated only after achieving the machine accuracy in the summation procedure. The form (89) should not be used when zt is large enough to be computed directly.

3.3.3 Efficiency of the computation of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t),t)$

The number of required matrix-vector multiplications for the computation of the integrated form (46) is $M+K-1$ for a polynomial approximation of the function of matrix, and $M+K$ for the Arnoldi approach, Cf. Sec. 2.3.2. The computation of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t_{k,l}),t_{k,l})$ in the general case (step 2(c)i in the algorithm) seems to cost considerable amount of additional computational effort. It can be readily seen from step 2(c)i that a direct computation of each of the $\vec{\mathbf{s}}^{l}_{ext}$'s requires a subtraction of two matrices, and a matrix-vector multiplication. Subtraction of matrices has the same scaling as a matrix-vector multiplication, $O(N^2)$. An alternative, which is less time-consuming, is to perform the computation as $\vec{\mathbf{s}}^{l}_{ext} = \vec{\mathbf{s}}(t_{k,l}) + G(\vec{\mathbf{u}}(t_{k,l}),t_{k,l})\vec{\mathbf{u}}(t_{k,l}) - G(\vec{\mathbf{u}}(t_{mid}),t_{mid})\vec{\mathbf{u}}(t_{k,l})$. This requires two matrix-vector multiplications for each l. This is with the exception of $l = M_k \backslash 2$, which indexes the middle internal time-point, t_{mid}; the extended part of the inhomogeneous term vanishes, and we are left with $\vec{\mathbf{s}}^{M_k\backslash 2}_{ext} = \vec{\mathbf{s}}(t_{mid})$. Thus, the computation in the middle point does not involve additional expensive operations. The overall additional cost of step 2(c)i is $2(M_k-1)$ matrix-vector multiplications. This roughly doubles the computational effort.

Most frequently, the computational effort can be considerably reduced by a proper formulation of the calculation. First, in many problems, the operator represented by $G(\vec{\mathbf{u}}(t_{k,l}),t_{k,l})-G(\vec{\mathbf{u}}(t_{mid}),t_{mid})$ is diagonal in the basis of representation. Thus, the scaling of its operation on $\vec{\mathbf{u}}(t_{k,l})$ becomes linear, $O(N)$. The computational cost of this operation is negligible. A common example is a Hamiltonian which is composed from a stationary part and a time-dependent nonlinear potential,

$$H(\vec{\mathbf{u}}(t),t)=H_0+V(\vec{\mathbf{u}}(t),t) \tag{90}$$

In this case we have:

$$[G(\vec{\mathbf{u}}(t_{k,l}),t_{k,l})-G(\vec{\mathbf{u}}(t_{mid}),t_{mid})]\vec{\mathbf{u}}(t_{k,l})=-i[V(\vec{\mathbf{u}}(t_{k,l}),t_{k,l})-V(\vec{\mathbf{u}}(t_{mid}),t_{mid})]\vec{\mathbf{u}}(t_{k,l}) \tag{91}$$

When the problem is represented in the spatial basis, the potential becomes diagonal. Thus, the computation of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t_{k,l}),t_{k,l})$ does not require any additional expensive operation.

Moreover, in many situations, the dependence of $G(\vec{\mathbf{u}}(t),t)$ on $\vec{\mathbf{u}}(t)$ and t is determined by a small number of parameters. This may reduce the required computational cost. For example, let us consider a Hamiltonian of the form of (90) with a potential

$$V(\vec{\mathbf{u}}(t),t)=\zeta(\vec{\mathbf{u}}(t),t)\mu \tag{92}$$

where $\zeta(\vec{\mathbf{u}}(t),t)$ is a scalar parameter and μ is a matrix. $\zeta(\vec{\mathbf{u}}(t),t)$ may represent an electric or magnetic field (up to a sign), and μ may represent the electric or magnetic moment operator, respectively (state dependent electric or magnetic fields occur, e. g. , in coherent control problems, when the Krotov algorithm is employed; see [52] for a review). We have:

$$[G(\vec{\mathbf{u}}(t_{k,l}),t_{k,l})-G(\vec{\mathbf{u}}(t_{mid}),t_{mid})]\vec{\mathbf{u}}(t_{k,l})=-i[\zeta(\vec{\mathbf{u}}(t_{k,l}),t_{k,l})-\zeta(\vec{\mathbf{u}}(t_{mid}),t_{mid})]\,\mu\,\vec{\mathbf{u}}(t_{k,l}) \tag{93}$$

We see that the computation of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t_{k,l}),t_{k,l})$ requires just one matrix-vector multiplication (when μ is a non-diagonal matrix). Thus, the additional computational cost of the $\vec{\mathbf{s}}^{l}_{ext}$'s is just M_k-1 matrix-vector multiplications.

More generally, let us consider $G(\vec{\mathbf{u}}(t),t)$ which can be represented in the following form:

$$G(\vec{\mathbf{u}}(t),t)=G_0+\sum_{j=1}^{L}\xi_j(\vec{\mathbf{u}}(t),t)G_j \tag{94}$$

where the $\xi_j(\vec{\mathbf{u}}(t),t)$'s are scalar parameters, the G_j's are matrices, and $L\ll N^2$. We have:

$$[G(\vec{\mathbf{u}}(t_{k,l}),t_{k,l})-G(\vec{\mathbf{u}}(t_{mid}),t_{mid})]\vec{\mathbf{u}}(t_{k,l})=\sum_{j=1}^{L}[\xi_j(\vec{\mathbf{u}}(t_{k,l}),t_{k,l})-\xi_j(\vec{\mathbf{u}}(t_{mid}),t_{mid})]G_j\vec{\mathbf{u}}(t_{k,l}) \tag{95}$$

The cost of this operation is less than a single multiplication of a vector by $G(\vec{\mathbf{u}}(t),t)$. Since $L\ll N^2$, the computational cost of the expressions $\xi_j(\vec{\mathbf{u}}(t_{k,l}),t_{k,l})-\xi_j(\vec{\mathbf{u}}(t_{mid}),t_{mid})$ becomes negligible in comparison to a matrix-vector multiplication (note that the condition for L becomes different when the multiplication by the matrix is represented by an equivalent linear operation procedure with lower scaling than $O(N^2)$).

3.4 Parameter choice

Several free parameters are involved in the propagation algorithm. They need to be supplied by the user in each problem. The choice of the parameters determines the efficiency and the accuracy of the algorithm. With an inappropriate choice, the algorithm might become inefficient, inaccurate, or even completely fail. Of course, the parameters which yield good results are problem dependent.

There are two main criteria for a successful choice of the parameters:

1. The accuracy of the results;
2. The efficiency of the algorithm.

The treatment of the first criterion is closely related to the ability to estimate the error of the different approximations involved in the algorithm. This important topic is left to Appendix D, due to its length. The present discussion mainly focuses on the second criterion.

In general, a successful choice of parameters can be achieved by trial and error. The choice may improve with experience with the algorithm, and after the treatment of similar problems. Here we give several recommendations which are based on our experience with the algorithm.

The choice of the ϵ tolerance parameter (see step 2(c)vii) is obvious: It should be determined by the desired accuracy of the solution. Note that the tolerance parameter is defined for a single time-step. The error of the final solution is expected to accumulate during the propagation, roughly as the sum of the errors of each time-step.

In addition, there are three free parameters which need to be specified in each time-step:

1. The length of the time-step interval, Δt_k;
2. The number of expansion terms for the approximation of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t)$, M_k;
3. The number of expansion terms for the approximation of $\tilde{f}_{M_k}(\widetilde{G}, t - t_{k,0})\vec{\tilde{\mathbf{v}}}_{M_k}$, K_k.

Our experience shows that the parameters should be chosen such that *a single iteration is required in each time-step*. In other words, the steps of the loop in stage 2c of the algorithm are performed just once. This is with the exception of the first time-step, in which the guess solution is of low accuracy (see Eq. (70)), and usually two or more iterations are required for a sufficient accuracy. The observation that the algorithm becomes most efficient with a minimal number of iterations is consistent with the reasoning that lead us to the choice of the $G(\vec{\mathbf{u}}(t), t)$ splitting (see Sec. 2.4.2)—*the weak point of the algorithm lies in the iterative process*. Hence, the part that the iterative process takes in the computation of an accurate solution should be minimized.

Typically, the values of M_k and K_k should be chosen to lie in the range $5 - 13$. For higher values, even though Δt_k can be increased, the algorithm usually becomes less efficient. The guess solution for the next time-step begins to become less accurate for large M. This is due to the high sensitivity of a high order extrapolation to roundoff errors. A

high order K is usually simply unnecessary. It should be noted that for high M orders, the algorithm may become numerically unstable. The source of the instability lies in the fact that both $(t-t_{k,0})^j$ and $\tilde{f}_j(\widetilde{G}, t-t_{k,0})$ in Eq. (86) typically become exceedingly small for high j's. Accordingly, the $\vec{\tilde{\mathbf{v}}}_j$'s attain very large values, and the computational process for obtaining them becomes unstable.

In the future, we plan to develop a version of the algorithm which is parameter free. In such an algorithm, the parameters are specified adaptively by the procedure during the propagation process, according to the accuracy requirements.

4 Numerical example

In the present section we test the efficiency of the semi-global propagator in solving a numerical problem of physical importance. Several numerical examples of relatively simple problems are already presented in Ref. [50]. In this paper, we test the performance of the propagator in a more realistic physical model problem. Moreover, for the first time, we demonstrate the capability of the algorithm to solve a problem with a *complex spectrum*, i. e. a problem in which the eigenvalue spectrum of G is distributed on the *complex plane*. This requires the use of the Arnoldi approach (see Sec. B.2) for the computation of $f_M(\widetilde{G}, t-t_k)\vec{\mathbf{v}}_{k,M}$ (see the procedure in Sec. 3.2, step 2(c)vi).

We choose a physical problem which is known to be challenging numerically—an atom subject to an intense laser field. This physical situation is characterized by extreme conditions for which an accurate numerical calculation becomes difficult. Under the influence of the intense field, a partial ionization of the atom occurs. The ionized part of the electron wave-function has the characteristics of an unbound particle, thus is spread to large spatial distances from the parent atom. Its dynamics is characterized by a strongly accelerated motion under the influence of the intense field. The central potential of the parent atom has a Coulomb character, which is steep in nature. These characteristics of the problem contribute to the difficulty of an accurate computation of the dynamics.

In order to give a reliable description of the problem, *absorbing boundary conditions* have to be employed. These are implemented here by a *complex absorbing potential*. This is the origin of the complex spectrum in our problem. The topic will be discussed in more detail in Sec. 4.2.

In Sec. 4.1 we present the physical details of the model problem. In Sec. 4.2 we present the details of the numerical implementation of the physical problem. In Sec. 4.3 the results are presented, and compared to a reference method.

4.1 The details of the physical problem

Remark: Atomic units are used throughout.

We use a one-dimensional model for the problem. The central potential of the parent

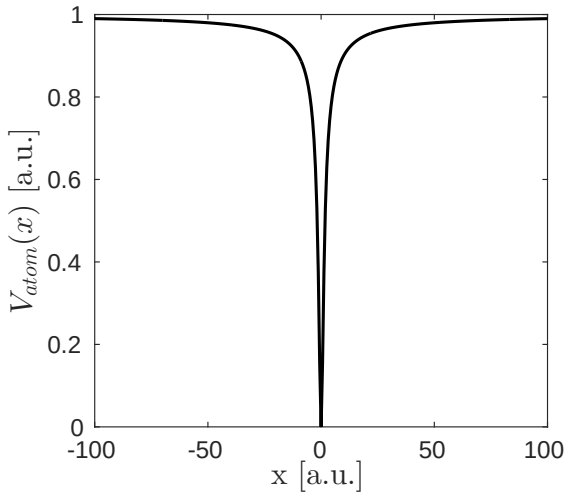

Figure 2: The central atom model potential

atom is represented by a truncated Coulomb potential (see Fig. 2):

$$V_{atom}(x) = 1 - \frac{1}{\sqrt{x^2+1}} \tag{96}$$

In this model, the singularity of the Coulomb potential at $x = 0$ is removed. The model has been extensively studied in the context of intense laser atomic physics. The fundamental energy difference, $\Delta E_1 = 0.395\, a.u.$, is similar to that of the hydrogen atom ($0.375\, a.u.$).

The laser pulse electric field has the following form (see Fig. 3):

$$\zeta(t) = 0.1\,\mathrm{sech}^2\left(\frac{t-500}{170}\right)\cos[0.06(t-500)] \tag{97}$$

The central frequency of the pulse is $\omega = 0.06\, a.u.$, which corresponds to a wavelength $\lambda = 760_{nm}$—similar to the central wavelength of the common Titanium-Sapphire laser. The envelope sech^2 form is known to be similar to the actual form of laser pulses. The peak amplitude of the field, $\zeta_{max} = 0.1\, a.u.$, corresponds to an intensity of $I_{max} = 3.52 \times 10^{14}\, W/cm^2$. The final time is $T = 1000\, a.u.$, which corresponds to a pulse duration of $24.2\, fs$.

The dipole approximation is employed for the potential induced by the laser field. The total potential of the *physical model* is

$$V_{phys}(x,t) = V_{atom}(x) + V_{field}(x,t) = 1 - \frac{1}{\sqrt{x^2+1}} - x\zeta(t) \tag{98}$$

However, the potential of the *numerical problem* must be modified in order to obtain a reliable description of the physical situation, as will be explained in Sec. 4.2.

The mass of the electron is $m = 1\, a.u.$, and the kinetic energy becomes $p^2/2$. The total time-dependent *physical Hamiltonian* is

$$H(t) = \frac{p^2}{2} + 1 - \frac{1}{\sqrt{x^2+1}} - x\zeta(t) \tag{99}$$

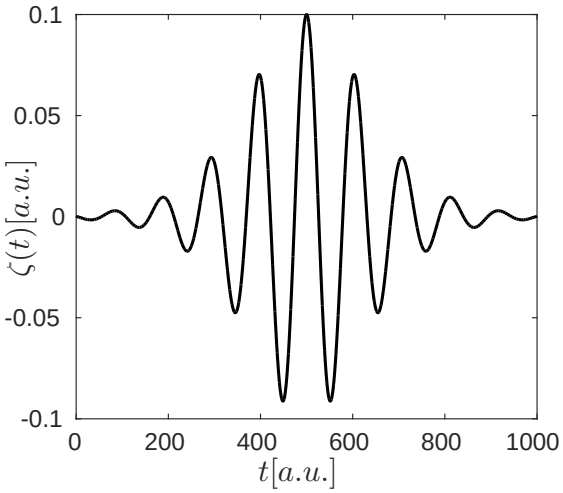

Figure 3: The electric field

The dynamics is governed by the time-dependent Schrödinger equation, Eq. (4).

4.2 Numerical implementation of the problem

The Fourier grid method [23] is employed for the Hamiltonian operation.

The x domain is $[-240, 240)$. We use an equidistant grid, with 768 points. The distance between adjacent grid points becomes $0.625\, a.u.$.

The present physical situation, which involves a partial ionization of the electron, requires a special numerical treatment, in order to prevent the appearance of spurious effects. The reason is that the ionized part of the electron behaves as a free particle, and is spread to very large spatial distances from the parent atom. Hence, the problem cannot be described as is in a finite spatial grid of a reasonable length. The description of the problem by a finite grid involves spurious effects of wraparound or reflection (depending on the computational method) of the wave function at the boundaries of the grid.

Usually, this problem is overcome by the employment of absorbing boundary conditions. The absorbing boundaries are implemented here by the addition of a complex absorbing potential at both boundaries of the grid (for a thorough review see [31]). The part of the wave-function which incomes into the complex absorbing potential decays gradually under the influence of the potential, until it becomes practically zero at the edge of the grid. Thus, the spurious effects are prevented. With the addition of the complex potential, the Hamiltonian becomes non-Hermitian, and consequently, the eigenvalue spectrum becomes complex.

Different absorbing potentials vary in their absorption capabilities. The part of the amplitude which is not absorbed by the potential is either reflected by the potential or transmitted. Thus, the efficiency of the absorbing boundaries in the prevention of spurious effects depends on the choice of the absorbing potential. The question of the choice of the absorbing potential becomes important when there is an interest in small amplitude effects.

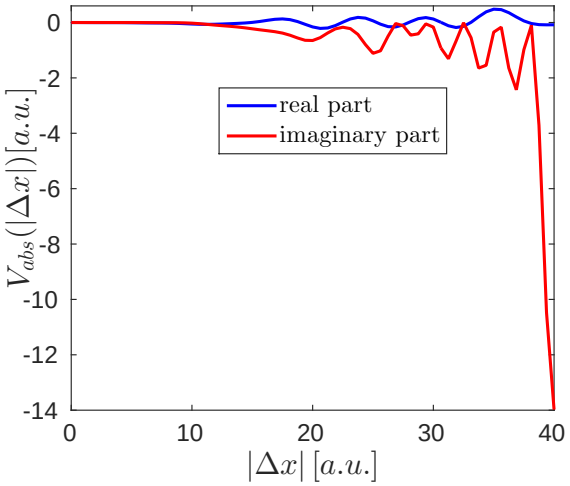

Figure 4: The real and imaginary parts of the absorbing potential, as a function of the absolute distance $|\Delta x|$ from the beginning of the absorbing boundary at $x = \pm 200\, a.u.$.

One of the major applications of the present physical situation is in the generation of high-harmonic spectrum, which is a small amplitude effect. It has already been recognized in the former high-harmonic generation simulations (see [25]) that reflection from the absorbing boundaries is responsible to large spurious effects in the calculation of the high-harmonic spectrum. An appropriate absorbing potential for this calculation could not be found by inspection.

In our simulation, we use an absorbing potential which is optimized numerically to maximize the absorption. The procedure basically relies on the principles presented in [36], but with several necessary modifications. The real part of the absorbing potential is constructed from a finite cosine series. The imaginary part is constructed from another finite cosine series, where the imaginary potential is given by squaring the cosine series and adding a minus sign. The optimization parameters are the cosine coefficients. This topic will be hopefully presented elsewhere. We choose the length of the absorbing boundary to be $40\, a.u.$. The obtained potential has a large imaginary part, which induces a significant shift of the Hamiltonian eigenvalues from the real axis into the fourth quarter of complex plane. The real and imaginary parts of the potential are plotted in Fig. 4 as a function of the absolute distance from the beginning of the absorbing boundary at $x = \pm 200\, a.u.$. The absorbing potential added at the left boundary is the mirror image of that added at the right boundary. The values of the absorbing potential are available in the complementary material.

It was verified that the results do not change significantly if the grid length is doubled, and the form of the high-harmonic spectrum is very well preserved (a test which failed in Ref. [25] for high intensity field, even for very large grid). The peak error of $|u_i|^2$ from the doubled grid does not exceed the order of 10^{-5}, where u_i is the i'th component of $\vec{\mathbf{u}}$ (of course, in the physical region, $|x| \leq 200$). Thus, the boundary effects are reduced to a

reasonable magnitude.

Since the absorbing potential is optimized for an ideal absorption, the influence of the physical potential at the boundaries should be "turned off", in order that the absorption will not be damaged. This is achieved by a modification of the physical potential to a potential which is constant at the absorbing boundaries. In order to avoid discontinuities in the potential derivatives, it is desirable to "turn off" the physical potential in a continuous manner. For that purpose, we use the following practice. Let us define a *soft rectangular function*:

$$\Omega(x;a,b,\alpha) = \frac{1}{2}\left\{\tanh\left[\alpha(x-a)\right] - \tanh\left[\alpha(x-b)\right]\right\} \tag{100}$$

The function is plotted in Fig. 5 for arbitrary parameters. The modified potential, which is constant at the absorbing boundaries, will be denoted as $V_{mod}(x)$. It is defined to satisfy the following conditions:

$$V_{mod}(0) = V_{phys}(0) \tag{101}$$
$$V'_{mod}(x) = V'_{phys}(x)\Omega(x;a,b,\alpha) \tag{102}$$

$V_{mod}(x)$ is obtained by integration in the following way:

$$\begin{aligned} V_{mod}(x) &= V_{phys}(0) + \int_0^x V'_{phys}(\xi)\Omega(\xi;a,b,\alpha)\,d\xi \\ &= V_{phys}(0) + V_{phys}(x)\Omega(x;a,b,\alpha) - V_{phys}(0)\Omega(0;a,b,\alpha) - \int_0^x V_{phys}(\xi)\Omega'(\xi;a,b,\alpha)\,d\xi \\ &\approx V_{phys}(x)\Omega(x;a,b,\alpha) - \int_0^x V_{phys}(\xi)\Omega'(\xi;a,b,\alpha)\,d\xi \end{aligned} \tag{103}$$

where we utilized the fact that $V_{phys}(0)\Omega(0;a,b,\alpha) \approx V_{phys}(0)$. We have:

$$\Omega'(x;a,b,\alpha) = \frac{1}{2}\alpha\left\{\mathrm{sech}^2\left[\alpha(x-a)\right] - \mathrm{sech}^2\left[\alpha(x-b)\right]\right\} \tag{104}$$

The integration in Eq. (103) is performed numerically.

In our problem, we choose the following parameters: $a = -197.5\,a.u.$, $b = 197.5\,a.u.$, $\alpha = 1$.

The numerical potential is given by

$$V_{num}(x) \equiv V_{mod}(x) + \begin{cases} 0 & |x| < 200 \\ V_{abs}(|x| - 200) & |x| \geq 200 \end{cases} \tag{105}$$

In Fig. 6, we plot both $V_{mod}(x)$ and the real part of $V_{num}(x)$ for a relatively small field, $\zeta = 0.005\,a.u.$.

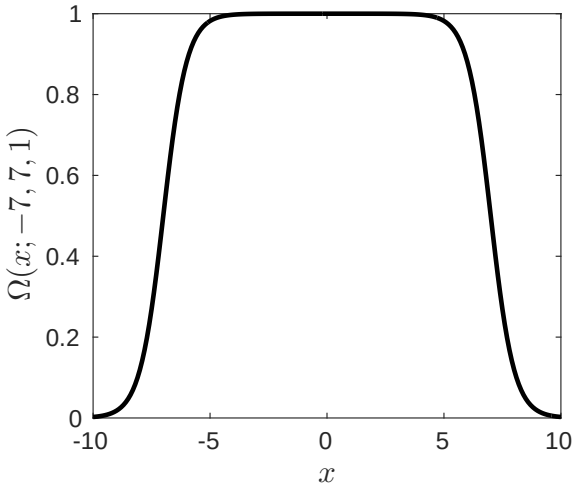

Figure 5: The soft rectangular function, with parameters $a = -7$, $b = 7$, $\alpha = 1$.

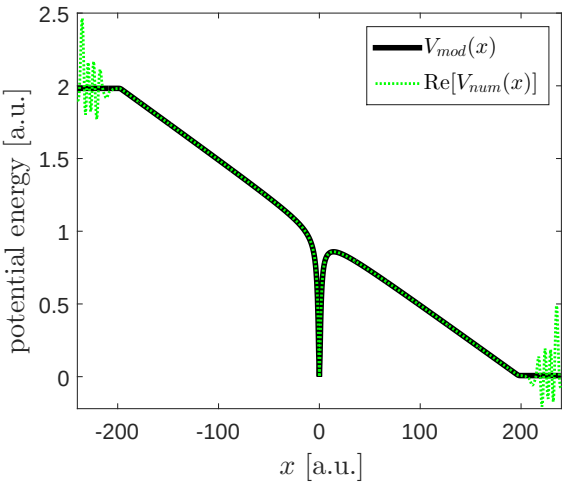

Figure 6: $V_{mod}(x)$ and the real part of $V_{num}(x)$ for $V_{phys}(x) = V_{atom}(x) - 0.005x$.

4.3 Results

The problem was solved by the semi-global propagator for different choices of M and K values. For each M and K choice, the problem was solved several times with different values of Δt (the time-step is constant throughout the propagation, as well as M and K). We compute the magnitude of the relative error of the final solution for each parameter choice. The efficiency of the propagator is demonstrated by a comparison of the resulting errors with those obtained by Runge-Kutta of the 4'th order (RK4, see Sec. 2.2). We compare also between the results of the semi-global propagator for the different M and K values.

In order to compare between different methods and parameter choices, we should compare the computational effort required for a similar accuracy. This may be done by choosing several specific examples. However, a fuller and a more reliable comparison is obtained by investigating the *behaviour* of the error decay with the computational effort. This is done by plotting the *error decay curve* for each method and parameter choice. In the error decay curve, the error is plotted Vs. the computational effort for several choices of Δt. A log-log plot is used. The computational effort is measured here by the number of Hamiltonian operations, which constitute the majority of the computational effort.

In order to obtain a consistent behaviour of the error decay for the semi-global propagator, we use a slightly different version of the algorithm from that presented in Sec. 3.2; we restrict the number of iterations to a single iteration, i. e. the steps of the loop in stage 2c of the algorithm are performed just once. This is with the exception of the first time-step, in which the solution is computed without a limitation on the number of iterations, where the parameter ϵ (see Sec. 3.2) represents the machine accuracy of the double-precision. This version of the algorithm restricts the inner freedom in the algorithm, thus ensures the consistency of the error decay curve.

The relative error should be computed from a highly accurate solution of the problem. A highly accurate solution cannot be obtained from RK4 with double-precision, even with an extremely small time-step. This is due to accumulation of the machine errors. Hence, we use a reference solution obtained by the semi-global propagator, with the following parameters: $M = 9,\ K = 13,\ \Delta t = 1/30$. No limitation is imposed on the number of iterations, and ϵ represents the machine accuracy of the double precision. It was verified that the estimated errors, computed by the tests for the different sources of the error in Appendix D, do not exceed the order of the machine accuracy. The high accuracy of the obtained solution is evident from the shapes of the error decay curves.

First, we shall compare the results of the semi-global propagator with the parameters $M = K = 7$, with those of RK4. The error decay curves are plotted in Fig. 7. The sampling points represent gradually decreasing values of Δt, for which the computational effort gradually increases. In each curve, Δt is decreased until the error stops to decay, due to the effects of roundoff errors.

The linear behaviour of the RK4 curve is apparent. This is in consistence with theory—the error of the RK4 method is of $O(\Delta t^4)$ (see Sec. 2.2). Since $\Delta t = T/N_t$, the error decays as N_t^{-4}. The number of Hamiltonian operations is linear with N_t. Thus, the log-log plot

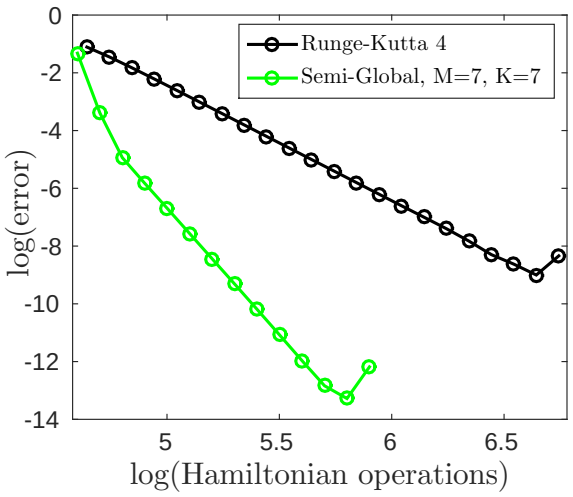

Figure 7: The error decay curves of the RK4 method, and the semi-global propagator with parameters $M = K = 7$. The $\log_{10}$ of the relative error is plotted Vs. the $\log_{10}$ of the number of Hamiltonian operations. The last sampling point in each curve represents the limit in which the effects of roundoff errors become important, and the error ceases to decay. The RK4 curve shows a linear behaviour with a slope very close to -4. There is also a seemingly linear region in the semi-global curve, with a slope close to -9.

yields a -4 slope. The slope obtained by a linear fit of the linear region of the curve agrees very well with theory (the obtained slope is -3.99). The semi-global curve has also a seemingly linear region. The slope obtained by a linear fit of the linear region is close to -9 (the precise value obtained is -8.77). The advantage of the semi-global propagator can be clearly seen.

Another advantage of the semi-global propagator is the maximal accuracy which can be obtained with the same machine accuracy. The minimal relative error which was obtained by RK4 is 9.96×10^{-10}. Thus, in this problem, the RK4 method with double-precision is limited to an accuracy of about 10^{-9}. The minimal error obtained for the semi-global propagator with this choice of parameters is 5.25×10^{-14}.

Relying on the linear fit of both curves, one can estimate the number of Hamiltonian operations needed for a requested accuracy for each method. Regular accuracy requirements for most physical applications are of the order of 10^{-5}. One finds that for an accuracy of 10^{-5}, the RK4 method requires 6.8 times the number of Hamiltonian operations required for the semi-global propagator. The advantage of the semi-global propagator becomes more apparent for applications which require a high accuracy solution. For an accuracy of 10^{-9}, the RK4 method requires 24 times the number of Hamiltonian operations required for the semi-global propagator.

We proceed with a comparison between different choices of K and M. We shall compare between the following choices: $M = K = 5$, $M = K = 7$, $M = K = 9$ (the results of $M = K = 7$ have already been presented in Fig. 7). The error decay curves are shown in

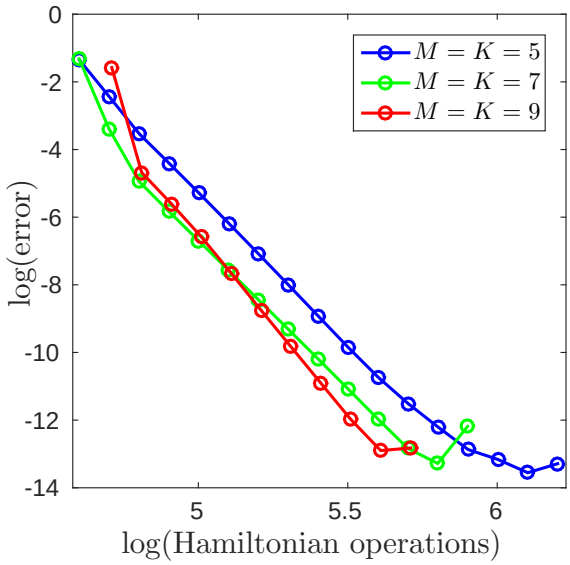

Figure 8: The error decay curves of the semi-global propagator with different choices of M and K. The $\log_{10}$ of the relative error is plotted Vs. the $\log_{10}$ of the number of Hamiltonian operations. All curves include a region with a seemingly linear behaviour.

Fig. 8. The choice of $M = K = 5$ is shown to give inferior results in comparison to the higher orders. The $M = K = 7$ choice is slightly advantageous over the $M = K = 9$ for regular accuracy requirements. The $M = K = 9$ choice becomes superior for high accuracy requirements. These findings are by no means general; the ideal parameter choice for a required accuracy is problem dependent.

All curves include a region with a seemingly linear behaviour. The slope of the linear region for $M = K = 5$ is very close to -9 (the precise value is -8.98). The slope for $M = K = 9$ is -10.6. As has already been mentioned, the slope of $M = K = 7$ is -8.77.

The explanation of these results requires a detailed error analysis. One can show that the error resulting from each of the three error sources, mentioned in Appendix D, behaves polynomially with Δt, under certain approximation assumptions. This is the origin of the seemingly linear behaviour in the error decay curves. However, the order of each error source with Δt is different. Since the overall error depends on several error sources, its behaviour with Δt is considerably more complicated than that of RK4, in which there is only one error source. A detailed error analysis is beyond the scope of the present paper, and is left for a future publication.

Nevertheless, the error decay rate in each curve is shown to be much advantageous over the common Taylor methods. The error decay rate is higher for each parameter choice than a Taylor method of the same order (the order of approximation for each of the three parameter choices is $M - 1$, and the slope predicted for a corresponding Taylor method is $-(M - 1)$). Particularly, the $M = K = 5$ choice has the same approximation order as RK4, and the decay rate is much higher.

We can summarize that the semi-global propagator has two advantages over the Taylor

approach, which lead to a higher error decay rate:

1. In general, approximation by higher orders leads to higher error decay rate. The use of high order expansion is not recommended in Taylor methods, because of the inefficiency of a Taylor series as an approximation tool in higher orders (see Sec. 2.2). The semi-global approach, being free of Taylor considerations, allows to use higher order expansions than the Taylor approach;

2. The error decay rate of the semi-global propagator is higher even for the same expansion order as the Taylor method.

In this context, it is interesting to compare between the current approach and a class of propagators, known as *exponential integrators* (see, e. g. , [8, 17–19, 30, 43]). The propagation technique of the exponential integrators is also based on the $f_m(z,t)$ functions (defined in Eq. (35)). Certain elements of the current approach can be found to have been employed in exponential integrators (see, in particular, [13]). However, there is a fundamental difference between this class of propagators and the current approach: The exponential integrator studies always employ local Taylorian considerations for constructing the propagation, while the propagation technique of the present approach is Taylor free. Hence, the error decay rate of the exponential integrators is limited by the order of the Taylor approximation employed, with the slope predicted for a simple Taylor method of the same order (see, for example, [43]). In contrast, the error decay rates in the present algorithm significantly exceed those of a Taylor method with the same expansion order. This fundamental difference between the approaches also allows to use higher expansion orders and larger time steps in the present approach in comparison to the exponential integrators.

5 Conclusion

The solution of the time-dependent Schrödinger equation is one of the most important tasks in quantum physics. It can be solved by global means when the Hamiltonian is stationary. This approach leads to vast improvement in accuracy and efficiency. In the present paper we presented a generalization of the global approach to the general case of a time-dependent, nonlinear Hamiltonian, with the additional inclusion of an inhomogeneous source term. The global approach can be implemented for the inhomogeneous Schrödinger equation with a stationary Hamiltonian. The solution method in this case constitutes the basis for the present approach for the general case of time-dependence or nonlinearity of the Hamiltonian. The general case can be treated by a semi-global approach, which combines global and local elements. The semi-global approach is characterized by propagation in relatively large time-steps, each of which is treated by global means.

The semi-global approach was shown to be significantly more efficient than the common local approach, which is based on Taylor considerations. Since the propagation method is Taylor free, it has the advantage of being able to use higher order approximations. The

error decay rates were shown to be much higher in the new approach, thus enabling to achieve highly accurate solutions with a vast decrease in computational effort.

The semi-global algorithm applies also to the solution of a general set of ODE's, a fundamental problem in numerical analysis. The solution of the Schrödinger equation is an application in a special case of the general problem.

It was demonstrated that the semi-global algorithm is applicable also to non-Hermitian operators by the use of the Arnoldi approach. Thus, it applies also to problems in non-Hermitian quantum mechanics or to the solution of the Liouville von-Neumann equation.

We asserted that the success of the present approach is mainly attributed to the global element of the method, in which large intervals are treated as a whole in a unified process. This significantly reduces the main problem in the regular local schemes, in which the step-by-step propagation leads to a large numerical effort and error accumulation.

This advantage of the global approach over the local one reveals a more fundamental difference. The local approach relies (explicitly or implicitly) on a Taylor approximation. The Taylor considerations are based on the derivative concept, which is local in nature. The ability to deduce global information from local information is limited. Hence, it is not surprising that the Taylor expansion has poor convergence properties, thus becomes an inefficient tool for approximation purposes. In our opinion, the extreme importance of the Taylor expansion for analysis led, unjustly, to its wide spread as an approximation tool. In contrary, the present approach relies on orthogonal polynomial expansions, which have fast convergence properties. The approximation by an orthogonal set is intimately related to the fundamental concept of *interpolation* (see Appendix A.2). The interpolation concept is global in nature, where the information is deduced from samplings which are distributed all over the approximation interval. Thus, it becomes significantly advantageous over the Taylor expansion as an approximation tool.

It should be noted that even the local element in the semi-global approach is advantageous over the local Taylor considerations. The initial information for the propagation into the next time-step is obtained by *extrapolation*, rather than local derivative information. The extrapolation concept is just an extension of the interpolation concept. A global interpolation approximation inside the interpolation interval can supply relatively accurate information for extrapolation outside the interval.

The main disadvantage of the present version of the algorithm is the necessity of specifying three parameters by the user. A successful choice of the parameters requires a trial and error process and experience. In order to accommodate with this problem, a parameter free version of the algorithm should be developed. The parameters will be determined adaptively by the procedure during the propagation process, to achieve maximal efficiency for the required accuracy. Such a version of the algorithm is expected also to significantly enhance the efficiency. The Chebyshev approximation should be replaced by a Leja approximation [39] for the flexibility of the parameter determination process. The efficiency of the procedure requires an accurate error estimation. One of the advantages of interpolation approximations is the relative ease of the error analysis and estimation. Thus, the adaptive semi-global scheme is expected to be more successful than the available adaptive schemes for Taylor propagators.

We believe that a proliferation of the semi-global algorithm will lead to a significant improvement of accuracy and efficiency in quantum applications, as well as in the vast variety of problems which require the solution of a large set of ODE's.

Acknowledgements

We want to thank Christiane Koch, Lutz Marder and Erik Torrontegui for their interest and help in this project. Calculations were carried out on high performance computers purchased with the help of the Wolfson Foundation. Work supported by Army Research Office (ARO) contract number W911NF-15-10250.

A Polynomial approximations

A.1 Approximation by a Newton interpolation

A.1.1 The Newton interpolation

The Newton interpolation is a polynomial interpolation of a function $f(x)$. The function is sampled at specific points distributed in the domain of approximation. The Newton interpolation polynomial approximates $f(x)$ within the domain from the sampling points. The approximation becomes exact at the sampling points.

Let us denote the sampling points by

$$x_j, \qquad j = 0, 1, \ldots, N \tag{106}$$

The value of $f(x_j)$ is given for all sampling points. The Newton interpolation approximates $f(x)$ using the following form:

$$\begin{aligned} f(x) &\approx a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_N(x - x_0)(x - x_1)\cdots(x - x_{N-1}) \\ &= \sum_{n=0}^{N} a_n R_n(x) \end{aligned} \tag{107}$$

where the a_n's are coefficients, and the $R_n(x)$'s are defined by

$$\begin{aligned} R_0(x) &= 1 \\ R_n(x) &= \prod_{j=0}^{n-1}(x - x_j) \qquad\qquad n > 0 \end{aligned} \tag{108}$$

The $R_n(x)$'s are called "Newton basis polynomials". In order to find the a_n's, we first have to become familiar with the concept of *divided difference*, which will be presented below.

Let us consider a function $f(x)$, and a set of points, as in Eq. (106). The divided differences have a recursive definition. We will give the definition, and examples will follow immediately. The divided difference for a single point x_0 is defined as

$$f[x_0] \equiv f(x_0) \tag{109}$$

The divided difference of more than a single point is defined as

$$f[x_0, x_1, \ldots, x_n] = \frac{f[x_1, x_2, \ldots, x_n] - f[x_0, x_1, \ldots, x_{n-1}]}{x_n - x_0} \tag{110}$$

For instance, the divided difference of two points is

$$f[x_0, x_1] \equiv \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \tag{111}$$

The divided difference of three points is

$$f[x_0, x_1, x_2] \equiv \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2 - x_0} \tag{112}$$

The a_n coefficients are given by the divided differences as follows:

$$a_n = f[x_0, x_1, \ldots, x_n] \tag{113}$$

A.1.2 Interpolation at Chebyshev points

When using a high order polynomial interpolation at equally spaced points, we may encounter a phenomenon which prevents it from being useful. The interpolation polynomial does not converge to the function $f(x)$ at the edges of the domain of approximation (even though it may converge very well at the middle of the domain). Instead, we might observe very large oscillations at the edges. The problem becomes more severe as N grows. This phenomenon is known as *Runge phenomenon*.

The Runge phenomenon disappears when we choose the sampling points of $f(x)$ appropriately. There is more than one appropriate choice of a set of sampling points. The most commonly used set is given by the so called *Chebyshev points* of the domain of approximation. The Chebyshev points become denser at the edges of the domain (see Fig. 9; this property is common to all sets of points which solve the Runge phenomenon). The Chebyshev points are originally defined for the domain $[-1, 1]$, but they can be easily transformed to another domain by a simple linear transformation, as will be described later.

The Chebyshev points for the domain $[-1, 1]$ are:

$$y_j \equiv \cos\left[\frac{(2j+1)\pi}{2(N+1)}\right], \qquad j = 0, 1, \ldots, N \tag{114}$$

Note that $y_0 > y_1 > \ldots > y_N$. Frequently, it is more convenient to index the points in an increasing order. We can reverse the order of the points, by defining them in the following way:

$$y_j \equiv -\cos\left[\frac{(2j+1)\pi}{2(N+1)}\right], \qquad j = 0, 1, \ldots, N \tag{115}$$

There is an alternative set of Chebyshev points, with similar characteristics:

$$y_j \equiv \cos\left(\frac{j\pi}{N}\right), \qquad j = 0, 1, \ldots, N \tag{116}$$

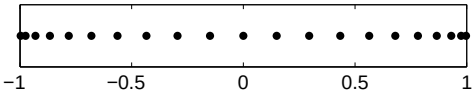

Figure 9: The Chebyshev points (Eq. (114)) for $N = 20$.

or, with a reversed order:

$$y_j \equiv -\cos\left(\frac{j\pi}{N}\right), \qquad j = 0, 1, \ldots, N \tag{117}$$

In this set of points, the function is sampled at the boundaries of the domain, unlike in the set (114). This can be advantageous in certain circumstances (for example, in the context of the semi-global propagation scheme, in which adjacent time-steps share a common point; see Sec. 3.1).

The Chebyshev points can be transformed to an arbitrary domain on the real axis, $[x_{min}, x_{max}]$. First, they are stretched or compressed to match the size $\Delta x = x_{max} - x_{min}$ of the domain:

$$y_j \longrightarrow \frac{\Delta x}{2} y_j$$

Then, they are shifted to the middle of the domain by adding the middle point,

$$\frac{x_{min} + x_{max}}{2}$$

The sampling points are finally obtained by the following linear transformation:

$$x_j \equiv \frac{1}{2}(y_j\,\Delta x + x_{min} + x_{max}) \tag{118}$$

There are other sets of points which solve the Runge phenomenon. The set of *Leja points* [39] can be advantageous when the required degree of approximation N is difficult to be estimated in advance. This topic is beyond the scope of this paper.

A.1.3 Numerical stability of the Newton interpolation

The Newton interpolation usually becomes numerically unstable when N grows. The main problem is that the a_n's of high n tend to become very large, and the corresponding adjacent polynomials become very small, or vice versa. This problem does not exist when the domain of approximation, $[x_{min}, x_{max}]$, is of length 4 [46]. Hence, for a domain defined on the real axis, the problem can be overcome by transforming the problem to a length 4 domain.

First, we transform the sampling points to a length 4 domain. The points in the new domain are defined as

$$\bar{x}_j \equiv \frac{4}{\Delta x} x_j \tag{119}$$

In general, we can define a transformed *variable*:

$$\bar{x} = \frac{4}{\Delta x} x \tag{120}$$

We also define a new function $\bar{f}(x)$, such that:

$$\bar{f}(\bar{x}) = f(x) \tag{121}$$

Then, we can approximate $f(x)$ at an arbitrary x value in the original domain, by using the Newton interpolation for the function $\bar{f}(x)$ at the sampling points $\bar{x}_j$. The approximation to $f(x)$ at an arbitrary x is given by the interpolation polynomial value at $\bar{x}$.

A.2 Chebyshev approximation

In the Chebyshev approximation, a function $f(x)$ is approximated by a truncated series of orthogonal polynomials, in the form of Eq. (15). The basis of expansion consists of the *Chebyshev polynomials*. They are defined as follows:

$$T_n(x) = \cos(n \cos^{-1} x), \qquad x \in [-1, 1], \qquad n = 0, 1, \ldots \tag{122}$$

In order to clarify the meaning of this weird definition, let us define a variable θ such that

$$x = \cos\theta \tag{123}$$

Then we obtain the following equivalent definition of the Chebyshev polynomials:

$$T_n(\cos\theta) = \cos(n\theta), \qquad \theta \in [0, \pi], \qquad n = 0, 1, \ldots \tag{124}$$

For instance,

$$T_0(x) = \cos(0) = 1 \tag{125}$$

$$T_1(x) = \cos\theta = x \tag{126}$$

$$T_2(x) = \cos(2\theta) = 2\cos^2\theta - 1 = 2x^2 - 1 \tag{127}$$

Note that the functions $\cos(n\theta)$ from the RHS of Eq. (124) span the function space in the domain $\theta \in [0, \pi]$. In addition, they are *orthogonal* in this domain:

$$\int_0^\pi \cos(m\theta)\cos(n\theta)\, d\theta = \frac{\pi\alpha_n}{2}\delta_{mn}, \qquad \alpha_n \equiv \begin{cases} 2 & n = 0 \\ 1 & n > 0 \end{cases} \tag{128}$$

The Chebyshev polynomials are obtained from the cosine basis by mapping θ into x with the nonlinear transformation (123). The resulting functions are also orthogonal in the x space, but with respect to a *weight function* in the new space. This can be seen by changing the integration variable of Eq. (128) from θ to x. We obtain:

$$\int_0^\pi \cos(m\theta)\cos(n\theta)\, d\theta = -\int_1^{-1} \frac{T_m(x)T_n(x)}{\sin\theta}\, dx = \int_{-1}^1 \frac{T_m(x)T_n(x)}{\sqrt{1-x^2}}\, dx = \frac{\pi\alpha_n}{2}\delta_{mn} \tag{129}$$

We have obtained the orthogonality relation of the Chebyshev polynomials under the weight function

$$w(x) = \frac{1}{\sqrt{1-x^2}} \tag{130}$$

The Chebyshev polynomials satisfy the following recurrence relation:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \qquad n \geq 1 \tag{131}$$

All Chebyshev polynomials can be obtained recursively from $T_0(x)$ and $T_1(x)$ (see Eqs. (125), (126)), using Eq. (131).

Now suppose we want to approximate a function $f(x)$ in a given domain $[x_{min}, x_{max}]$, from several samplings of the function in the domain. Suppose we can choose the sampling points as we wish. The function can be approximated from the sampling points by a Chebyshev series, as will be described below. For simplicity, let us first assume that the domain of approximation is $[-1, 1]$.

$f(x)$ can be spanned in the following form:

$$f(x) \approx \sum_{n=0}^{N} c_n T_n(x) \tag{132}$$

The c_n's are called the *Chebyshev coefficients* of $f(x)$. They can be obtained by projecting $f(x)$ onto each of the $T_n(x)$ basis functions. Using the orthogonality relation (129), the Chebyshev coefficients are given by the following scalar product expression:

$$c_n = \frac{2}{\pi\alpha_n} \int_{-1}^{1} f(x)T_n(x)w(x)\,dx \tag{133}$$

Equivalently, we can perform the scalar product in the θ space:

$$c_n = \frac{2}{\pi\alpha_n} \int_{0}^{\pi} f(\cos\theta)\cos(n\theta)\,d\theta \tag{134}$$

Suppose that the form of $f(x)$ is unknown, or that the integral cannot be performed analytically. We have to compute the Chebyshev coefficients *numerically* from a finite number of samplings of $f(x)$ within the domain. Fortunately, the problem of computing the Chebyshev coefficients can be reformulated by discrete means, without reduction of the quality of the approximation.

We utilize the fact that there exist discrete versions of the orthogonality relations between the cosine basis functions. The orthogonality relations are given by a finite sum over samplings of the cosine functions in equally spaced points in θ. For instance, we have the following orthogonality relation:

$$\sum_{j=0}^{K} \cos(k\theta_j)\cos(l\theta_j) = \frac{(K+1)\alpha_k}{2}\delta_{kl}, \qquad \theta_j \equiv \frac{(2j+1)\pi}{2(K+1)} \tag{135}$$

where α_k is defined in Eq. (128), and $K \geq 0$. Note that in the x space, the points $x_j = \cos\theta_j$ are just the Chebyshev points defined in (114). The orthogonality relation (135) can be utilized in order to find the Chebyshev coefficients from the sampling of $f(x)$ at the Chebyshev points, as we shall see. Let us define:

$$g(\theta) \equiv f(\cos\theta) \tag{136}$$

Eq. (132) can be rewritten as

$$g(\theta) \approx \sum_{m=0}^{N} c_m \cos(m\theta) \tag{137}$$

Let us multiply $g(\theta)$ by the basis function $\cos(n\theta)$, and sum over the set of $N+1$ Chebyshev points:

$$\sum_{j=0}^{N} g(\theta_j)\cos(n\theta_j), \qquad \theta_j = \frac{(2j+1)\pi}{2(N+1)}$$

We substitute $g(\theta)$ with the approximation (137), and obtain:

$$\sum_{j=0}^{N} g(\theta_j)\cos(n\theta_j) \approx \sum_{j=0}^{N}\sum_{m=0}^{N} c_m \cos(m\theta_j)\cos(n\theta_j) = \frac{(N+1)\alpha_n}{2} c_n \tag{138}$$

where we applied the orthogonality relation (135). The Chebyshev coefficients are finally given by

$$c_n = \frac{2}{(N+1)\alpha_n}\sum_{j=0}^{N} g(\theta_j)\cos(n\theta_j) = \frac{2}{(N+1)\alpha_n}\sum_{j=0}^{N} f\left[\cos\left(\frac{(2j+1)\pi}{2(N+1)}\right)\right]\cos\left[\frac{n(2j+1)\pi}{2(N+1)}\right] \tag{139}$$

Note that the Chebyshev coefficients defined by Eq. (139) are not identical with those defined by the integral version of Eq. (134). However, the inaccuracy in (139) is originated in the truncation error of Eq. (132) itself (see Eq. (138)). Thus, the total error will be of the same order of magnitude as the truncation error, and the quality of the approximation will be similar.

Actually, the set of $N+1$ equations defined by (139) is a *linear transformation* of the function value vector, $[g(\theta_0), g(\theta_1), \ldots, g(\theta_N)]^T$, into the coefficient vector, $[c_0, c_1, \ldots, c_N]^T$. This transformation is called a *discrete cosine transform* (DCT). It has an apparent similarity to the discrete Fourier transform (DFT). There are several kinds of DCT's. The transformation defined in Eq. (139) is sometimes referred as a *DCT of the second kind*.

The DCT's are reversible transformations. The inverse transformation of Eq. (139) is actually defined by Eq. (137) for the set of θ_j's. Hence, the approximation is *exact* for the Chebyshev sampling points. In that sense, Eq. (132) with the c_n's computed by Eq. (139) defines an *interpolation* of $f(x)$ in the set of $N+1$ Chebyshev points. A fundamental theorem of interpolation theory states that the interpolation polynomial for a

given set of sampling points is *unique*. Thus, the approximation presented here is equivalent to the approximation by a Newton interpolation at the Chebyshev points, described in Sec. (A.1.2).

The DCT transformations can be computed very efficiently by algorithms derived from the fast Fourier transform (FFT) algorithm. Hence, the scaling of the computational effort is of $O(N \ln N)$. There are available programs for computation of the DCT. When using them, a care should be taken on the exact definition of the transformation—usually, there are slight differences in the definition of the transformation coefficients.

It is also possible to approximate the Chebyshev coefficients by sampling $f(x)$ at the set of boundary including Chebyshev points (see Eq. (116)). We use the following discrete orthogonality relation:

$$\sum_{j=0}^{K} \frac{1}{\beta_j} \cos(k\theta_j) \cos(l\theta_j) = \frac{K\beta_k}{2} \delta_{kl}, \qquad \theta_j \equiv \frac{j\pi}{K}, \qquad \beta_j \equiv \begin{cases} 2 & j = 0, K \\ 1 & 1 \leq j \leq K-1 \end{cases} \tag{140}$$

where $K \geq 1$. Following similar steps as above, we obtain:

$$c_n = \frac{2}{N\beta_n} \sum_{j=0}^{N} \frac{1}{\beta_j} g(\theta_j) \cos(n\theta_j) = \frac{2}{N\beta_n} \sum_{j=0}^{N} \frac{1}{\beta_j} f\left[\cos\left(\frac{j\pi}{N}\right)\right] \cos\left(\frac{nj\pi}{N}\right) \tag{141}$$

Here again, the set of $N+1$ equations defined by Eq. (141) is a linear transformation of the function value vector into the coefficient vector. It is another kind of a DCT, sometimes referred as a *DCT of the first kind*. Its inverse is defined by Eq. (137) for this set of points. The boundary including points are preferable when the exact value of $f(x)$ at the boundaries is important.

In the case that the domain of approximation of $f(x)$ is different from the Chebyshev domain, $[-1, 1]$, it is required to shift the problem to the Chebyshev domain. The treatment of the problem is similar to that of Sec. A.1.2. For instance, let us discuss the approximation by the boundary including Chebyshev points. We denote the domain of approximation by $x \in [x_{min}, x_{max}]$. Let us denote the set of Chebyshev points by y_j:

$$y_j \equiv \cos\left(\frac{j\pi}{N}\right), \qquad j = 0, 1, \ldots, N \tag{142}$$

The Chebyshev points in the domain $[x_{min}, x_{max}]$ are defined by the linear transformation (118). We can also define a *variable* $y \in [-1, 1]$, which is given by the inverse linear transformation of x:

$$y \equiv \frac{2x - x_{min} - x_{max}}{\Delta x} \tag{143}$$

We define a function $\bar{f}(x)$ such that

$$\bar{f}(y) = f(x) \tag{144}$$

The approximation to $f(x)$ is given by

$$f(x) = \bar{f}(y) \approx \sum_{n=0}^{N} c_n T_n(y) \tag{145}$$

where

$$c_n = \frac{2}{N\beta_n} \sum_{j=0}^{N} \frac{1}{\beta_j} \bar{f}(y_j) \cos(n\theta_j) = \frac{2}{N\beta_n} \sum_{j=0}^{N} \frac{1}{\beta_j} f(x_j) \cos\left(\frac{nj\pi}{N}\right) \tag{146}$$

We see that the Chebyshev coefficients are simply given by a discrete cosine transform of $f(x)$, sampled at the Chebyshev points of the domain $[x_{min}, x_{max}]$.

The treatment in the case of the points of (114) is completely identical. We obtain:

$$c_n = \frac{2}{(N+1)\alpha_n} \sum_{j=0}^{N} f(x_j) \cos\left[\frac{n(2j+1)\pi}{2(N+1)}\right] \tag{147}$$

where the x_j's are given by Eq. (118), with the y_j's of Eq. (114).

As was mentioned in Sec. A.1.2, it is often more convenient to reverse the order of the Chebyshev points, in order to obtain increasing values of x with the point index. This can be done by defining them as in Eqs. (115), (117). However, care should be taken to preserve the original form of Eqs. (146), (147), in which each of the $f(x_j)$'s is multiplied by the cosine of the corresponding angle. Hence, the order of the angles should also be reversed. This is equivalent to the addition of a minus sign to the RHS of the two equations. Eq. (146) is replaced by

$$c_n = -\frac{2}{N\beta_n} \sum_{j=0}^{N} \frac{1}{\beta_j} f(x_j) \cos\left(\frac{nj\pi}{N}\right) \tag{148}$$

and Eq. (147) is replaced by

$$c_n = -\frac{2}{(N+1)\alpha_n} \sum_{j=0}^{N} f(x_j) \cos\left[\frac{n(2j+1)\pi}{2(N+1)}\right] \tag{149}$$

B Approximation methods for the multiplication of a vector by a function of matrix

Here we discuss several methods for the computation of the following vector:

$$\vec{\mathbf{u}} = f(A)\vec{\mathbf{v}} \tag{150}$$

where $\vec{\mathbf{v}}$ is an arbitrary vector, A is a matrix, and $f(x)$ is a function. All approximation methods are based on the following realization: When the multiplication of $f(A)$ with

$\vec{\mathbf{v}}$ is all what required, we can avoid the direct computation of $f(A)$, which is highly demanding numerically. Instead, we use successive multiplications of vectors by the matrix A. As has already been mentioned in Sec. 2.3.1, in certain cases we can replace the direct multiplication of the vector by the matrix A, by a computational procedure which is less demanding numerically.

B.1 Polynomial series approximations

The first two methods presented here are based on approximation of $f(x)$ by a polynomial $Q_L(x)$ of degree L. $Q_L(x)$ is a truncated polynomial series of $f(x)$ (Cf. Eq. (15)):

$$f(x) \approx Q_L(x) \equiv \sum_{n=0}^{L} b_n P_n(x) \tag{151}$$

where the $P_n(x)$'s are polynomials of degree n, and the b_n's are the corresponding expansion coefficients. We can approximate $\vec{\mathbf{u}}$ by the following expression (Cf. Eq.(16)):

$$\vec{\mathbf{u}} \approx Q_L(A)\vec{\mathbf{v}} = \sum_{n=0}^{L} b_n P_n(A)\vec{\mathbf{v}} \tag{152}$$

The idea is to compute the expressions $P_n(A)\vec{\mathbf{v}}$ by successive multiplications of vectors by A, instead of computing $P_n(A)$ and multiplying $\vec{\mathbf{v}}$ by the resulting matrix.

As we have seen in Appendix A, when using a polynomial expansion for the approximation of a function, we first have to define the approximation domain, $[x_{min}, x_{max}]$. The approximation is expected to be accurate only inside the approximation domain. In our problem, the approximation should be accurate in the *eigenvalue domain* of A. This can be readily seen by considering the decomposition of $\vec{\mathbf{v}}$ into the eigenvectors of A:

$$\vec{\mathbf{v}} = \sum_{j=0}^{N-1} v_j \vec{\boldsymbol{\varphi}}_j \tag{153}$$

where the $\vec{\boldsymbol{\varphi}}_j$'s are the eigenvectors of A, the v_j's are the components of $\vec{\mathbf{v}}$ in the eigenvector basis, and N is the dimension of A. Plugging (153) into (150), we obtain:

$$\vec{\mathbf{u}} = \sum_{j=0}^{N-1} v_j f(\lambda_j) \vec{\boldsymbol{\varphi}}_j \tag{154}$$

where λ_j is the eigenvalue of $\vec{\boldsymbol{\varphi}}_j$. When using the approximation of Eq. (151), we actually replace the accurate expression of Eq. (154) by the following approximated expression:

$$\vec{\mathbf{u}} \approx \sum_{j=0}^{N-1} v_j Q_L(\lambda_j) \vec{\boldsymbol{\varphi}}_j \tag{155}$$

It is clear that $Q_L(\lambda_j)$ should be accurate for each of the λ_j's. Hence, the approximation domain has to cover the whole eigenvalue domain of A.

Frequently, the eigenvalue domain of A is unknown, and we have to estimate it. Note that an overestimation of the eigenvalue domain size costs additional numerical effort, because more terms are required to approximate $f(x)$. In cases that the eigenvalue domain cannot be estimated, or when the eigenvalue domain is complex, the Arnoldi approach (see Sec. B.2) should be used instead of the polynomial expansion methods.

B.1.1 Newton interpolation

One approach for approximation of $\vec{\mathbf{u}}$ by a polynomial series is by using a Newton interpolation of $f(x)$ at the Chebyshev points of the eigenvalue domain, defined by Eq. (118) (see Appendix A.1.2) [46]. The Newton interpolation polynomial is (Cf. Eq. (107)):

$$Q_L(x) = \sum_{n=0}^{L} a_n R_n(x) \tag{156}$$

The x_j sampling points which define the a_n's and the $R_n(x)$'s are given by Eq. (118). The $R_n(x)$'s satisfy the following recurrence relation:

$$R_{n+1}(x) = (x - x_n)R_n(x) \tag{157}$$

$\vec{\mathbf{u}}$ is approximated by

$$\vec{\mathbf{u}} \approx \sum_{n=0}^{L} a_n R_n(A)\vec{\mathbf{v}} \tag{158}$$

The recurrence relation (157) can be utilized in order to compute the expressions $R_n(A)\vec{\mathbf{v}}$ successively.

Let us index the sampling points by $j = 0, 1, \ldots, L$. The algorithm for the computation of $\vec{\mathbf{u}}$ is described below:

1. Compute the Chebyshev points y_j by Eq. (115) or by Eq. (117).
2. Compute the Chebyshev points x_j in the eigenvalue domain $[x_{min}, x_{max}]$ from the y_j's by Eq. (118).
3. Compute the function values $f_j = f(x_j)$.
4. Compute the divided differences a_n, $n = 0, 1, \ldots, L$, recursively from f_j and x_j, using Eqs. (113), (109), (110).
5. $\vec{\mathbf{w}} = \vec{\mathbf{v}}$
6. $\vec{\mathbf{u}} = a_0\vec{\mathbf{w}}$
7. for $i = 1$ to L

(a) $\vec{\mathbf{w}} = A\vec{\mathbf{w}} - x_{i-1}\vec{\mathbf{w}}$

(b) $\vec{\mathbf{u}} = \vec{\mathbf{u}} + a_i\vec{\mathbf{w}}$

8. end for

In practice, the Newton interpolation problem should be transferred to a domain of length 4, for numerical stability (see Appendix A.1.3). This amounts of two slight changes. We have to add to the recursion relation of Eq. (157) an additional conversion factor, in order to transform x to the domain of length 4:

$$R_{n+1}(x) = \frac{4}{\Delta x}(x - x_n)R_n(x) \tag{159}$$

where $\Delta x = x_{max} - x_{min}$. In addition, the sampling points x_j that appear in the denominator of the divided difference formula (110) are replaced by $\bar{x}_j = 4x_j/\Delta x$. Accordingly, we insert the following changes into the algorithm:

1. The x_j's in stage 4 are replaced by $4x_j/\Delta x$ (note that the f_j's from stage 3 remain the same).

2. Stage 7a is replaced by $\vec{\mathbf{w}} = (A\vec{\mathbf{w}} - x_{i-1}\vec{\mathbf{w}})4/\Delta x$

$\vec{\mathbf{u}}$ can be computed with other sets of sampling points which solve the Runge phenomenon (for instance, the Leja points; see Appendix A.1.2). We use the same recursion formula and algorithm, where the x_j's are the desired set of points.

B.1.2 Chebyshev expansion

Another approach for approximating $\vec{\mathbf{u}}$ is by the expansion of $f(A)$ in a Chebyshev series [45]. The approximation polynomial $Q_L(x)$ is given by

$$Q_L(x) = \sum_{n=0}^{L} c_n T_n(y) \tag{160}$$

where

$$y \equiv \frac{2x - x_{min} - x_{max}}{\Delta x} \tag{161}$$

The c_n's are given by Eq. (147) or Eq. (146), where the x_j's are given by Eq. (118), together with Eq. (114) or Eq. (116), respectively (see Appendix A.2).

$\vec{\mathbf{u}}$ is approximated by

$$\vec{\mathbf{u}} \approx Q_L(A)\vec{\mathbf{v}} = \sum_{n=0}^{L} c_n T_n(\bar{A})\vec{\mathbf{v}} \tag{162}$$

where

$$\bar{A} = \frac{2A - x_{min} - x_{max}}{\Delta x} \tag{163}$$

The Chebyshev polynomials satisfy the recurrence relation

$$T_{n+1}(y) = 2yT_n(y) - T_{n-1}(y), \qquad n \geq 1 \tag{164}$$

where

$$T_0(y) = 1 \tag{165}$$
$$T_1(y) = y \tag{166}$$

We can compute the expressions $T_n(\bar{A})\vec{\mathbf{v}}$ successively by utilizing the recurrence relation, where y in Eqs. (164), (166), is substituted by $\bar{A}$.

The algorithm for the computation of $\vec{\mathbf{u}}$ is described below:

1. Compute the Chebyshev points y_j by Eq. (114) or by Eq. (116).
2. Compute the Chebyshev points x_j in the eigenvalue domain $[x_{min}, x_{max}]$ from the y_j's by Eq. (118).
3. Compute the function values $f_j = f(x_j)$.
4. Compute the Chebyshev coefficients c_n from the f_j's by Eq. (147) or by Eq. (146).
5. $\vec{\mathbf{w}}_1 = \vec{\mathbf{v}}$
6. $\vec{\mathbf{w}}_2 = [2A\vec{\mathbf{v}} - (x_{min} + x_{max})\vec{\mathbf{v}}]/\Delta x$
7. $\vec{\mathbf{u}} = c_0\vec{\mathbf{w}}_1 + c_1\vec{\mathbf{w}}_2$
8. for $i = 2$ to L
   (a) $\vec{\mathbf{w}}_3 = 2[2A\vec{\mathbf{w}}_2 - (x_{min} + x_{max})\vec{\mathbf{w}}_2]/\Delta x - \vec{\mathbf{w}}_1$
   (b) $\vec{\mathbf{u}} = \vec{\mathbf{u}} + c_i\vec{\mathbf{w}}_3$
   (c) $\vec{\mathbf{w}}_1 = \vec{\mathbf{w}}_2$
   (d) $\vec{\mathbf{w}}_2 = \vec{\mathbf{w}}_3$
9. end for

B.2 Arnoldi approach

The approximations of Sec. B.1 are based on the assumption that the eigenvalue domain is known, or can be estimated. They cannot be applied when it is impossible to estimate the eigenvalue domain. The difficulty in the estimation of the eigenvalue domain becomes severe when the eigenvalues of A are distributed on the complex plane, and not only on the real or on the imaginary axis.

Moreover, the concept of Chebyshev sampling is defined for a one dimensional *axis*, which may be the real or imaginary axis. Hence, a Chebyshev approximation can be applied

for functions of a real variable, or a purely imaginary variable. However, a Chebyshev approximation is not suitable for functions of *complex* variables, which are distributed on the two dimensional *complex plane*. Thus, the methods of Sec. B.1 are not applicable when the eigenvalue spectrum of A is complex.

In the present section, we introduce the Arnoldi approach for approximation of (150) [47]. In the Arnoldi approach, the eigenvalue domain needn't be known, and the sampling is chosen by different considerations. Thus, it becomes suitable also for the treatment of matrices with a complex spectrum.

The Arnoldi approximation is intimately related to the polynomial approximations, but the approach to the problem is different. We can view our basic problem as the problem of reduction of large-scale matrix calculations into simplified approximations. The polynomial approximations reduce the *calculation* of the matrix into a *simplified calculation* of the *same matrix*, in which a function is reduced into a low degree polynomial. In the Arnoldi approach, the *matrix itself* is reduced into a small-scale matrix. This is done by the choice of a "good" set of a small number of vectors, which can be representative in the framework of the specific problem. The matrix A is represented in the reduced subspace spanned by the vectors.

The approximation is based on a reduction of the problem to the subspace spanned by the following vectors:

$$\vec{\mathbf{v}}, A\vec{\mathbf{v}}, A^2\vec{\mathbf{v}}, \ldots, A^L\vec{\mathbf{v}} \tag{167}$$

The subspace is spanned by multiplications of the vector $\vec{\mathbf{v}}$ by powers of A, from degree 0 to L. A vector space of this kind is called a *Krylov space*. The space spanned by (167) is a *Krylov space of dimension* $L+1$. The Krylov space is a "good" subspace in our problem for two reasons, which are interrelated:

1. Any polynomial approximation of $\vec{\mathbf{u}}$ of degree L in the form of (152) can be spanned by the Krylov space. This is a direct consequence of the fact that any polynomial can be expressed in the terms of the Taylor polynomials. Thus, the Krylov space is actually the characteristic subspace of polynomial approximations of degree L.

2. Suppose $\vec{\mathbf{v}}$ can be spanned by a subspace in the eigenvector spectrum of A. The subspace is invariant to multiplication by A or $f(A)$, which are diagonal in the eigenvector basis (this can be readily seen by expanding $\vec{\mathbf{v}}$ in the eigenvector space, as in Eq. (154)). Thus, $\vec{\mathbf{u}}$ remains in the same eigenvector subspace as $\vec{\mathbf{v}}$. The Krylov space also remains in the eigenvector subspace, for the same reason. Hence, it is expected to be effective for the representation of $\vec{\mathbf{u}}$. Note that if $\vec{\mathbf{v}}$ is spanned by an eigenvector space of dimension up to $L+1$, $\vec{\mathbf{u}}$ can be fully represented by the Krylov space. Even when $\vec{\mathbf{v}}$ is spanned by the whole eigenvector space, frequently a small number of eigenvectors dominate its composition. Thus, the Krylov space may still be effective in approximating it.

The vectors of Eq. (167) are in general non-orthogonal. Moreover, the vectors are getting closer to be parallel with the degree of A. When using them as a basis set, this

might be a source of numerical instability. We should work with an orthonormal basis set for spanning the Krylov space, for the sake of numerical stability and the simplicity of the calculations.

In order to obtain an orthonormal basis, we use the *Gram-Schmidt process*. The idea underlying the Gram-Schmidt orthonormalization goes as follows: The orthonormal basis vectors are computed successively from the original non-orthogonal set of vectors. We subtract from each vector from the original set its projection on the subspace spanned by the already computed orthonormal vectors. We are left with a vector which is orthogonal to the subspace spanned by the previous vectors. Then, we simply normalize it, and obtain an orthonormal set which is enlarged by one dimension. Then, we continue to the next vector from the original set, and so on.

In practice we use the *Modified-Gram-Schmidt* (MGS) algorithm which is equivalent, mathematically, to Gram-Schmidt but less sensitive to roundoff errors. The orthonormalization in the context of the Krylov space can be implemented by an iterative process, as will be seen. The iterative algorithm is referred as *Arnoldi iteration*.

Let us denote the orthonormal set by

$$\vec{\boldsymbol{v}}_0, \vec{\boldsymbol{v}}_1, \ldots, \vec{\boldsymbol{v}}_L$$

As will be seen, the vector $\vec{\boldsymbol{v}}_n$ belongs to the Krylov space of dimension $n+1$. In the algorithm, we compute an additional orthonormal vector, $\vec{\boldsymbol{v}}_{L+1}$, which belongs to the Krylov space of dimension $L+2$. The necessity of its computation will be clarified in what follows. We denote the scalar product of two vectors, $\vec{\mathbf{r}}$ and $\vec{\mathbf{s}}$, by $\langle \vec{\mathbf{r}}, \vec{\mathbf{s}} \rangle$.

The algorithm is described below. During the algorithm, we also store in the memory a set of constants, which participate in the computation. The necessity of this will be clarified in what follows.

1. Compute the first vector in the orthonormal set as the normalized $\vec{\mathbf{v}}$:

$$\vec{\boldsymbol{v}}_0 = \frac{\vec{\mathbf{v}}}{\|\vec{\mathbf{v}}\|}$$

2. for $j = 0$ to L
   (a) Compute a non-orthonormalized new vector by setting: $\vec{\boldsymbol{v}}_{j+1} = A\vec{\boldsymbol{v}}_j$.
   (b) for $i = 0$ to j
      i. Set: $\Gamma_{i,j} = \langle \vec{\boldsymbol{v}}_i, \vec{\boldsymbol{v}}_{j+1} \rangle$.
      ii. Subtract from $\vec{\boldsymbol{v}}_{j+1}$ its projection on $\vec{\boldsymbol{v}}_i$: $\vec{\boldsymbol{v}}_{j+1} = \vec{\boldsymbol{v}}_{j+1} - \Gamma_{i,j}\vec{\boldsymbol{v}}_i$.
   (c) end for
   (d) Set: $\Gamma_{j+1,j} = \|\vec{\boldsymbol{v}}_{j+1}\|$.
   (e) Normalize $\vec{\boldsymbol{v}}_{j+1}$ by setting:

$$\vec{\boldsymbol{v}}_{j+1} = \frac{\vec{\boldsymbol{v}}_{j+1}}{\Gamma_{j+1,j}}$$

3. end for

Clearly, the $\vec{\boldsymbol{v}}_{j+1}$ computed in stage 2a belongs to the Krylov space of dimension $j+2$, since it is composed from some linear combination of the first $j+2$ Krylov vectors. Thus, in each iteration, the Krylov space is enlarged by one dimension.

At the end of the algorithm, we are left with an orthonormal set of dimension $L+2$. We also computed, seemingly by the way, the $\Gamma_{i,j}$'s. These have a special significance. Actually, $\Gamma_{i,j}$ is the *matrix element of A in the orthonormal basis*, i. e. , it is equivalent to $A_{i,j} = \langle \vec{\boldsymbol{v}}_i, A\vec{\boldsymbol{v}}_j \rangle$. This can be readily seen from the algorithm. In stage 2(b)i, the vector $\vec{\boldsymbol{v}}_{j+1}$ is given by

$$A\vec{\boldsymbol{v}}_j - \sum_{k=0}^{i-1} \Gamma_{k,j} \vec{\boldsymbol{v}}_k$$

with the summation convention of Eq. (38). Using the orthonormality relations between the vectors, we obtain immediately:

$$\Gamma_{i,j} = \left\langle \vec{\boldsymbol{v}}_i, A\vec{\boldsymbol{v}}_j - \sum_{k=0}^{i-1} \Gamma_{k,j} \vec{\boldsymbol{v}}_k \right\rangle = \langle \vec{\boldsymbol{v}}_i, A\vec{\boldsymbol{v}}_j \rangle = A_{i,j}, \qquad i \leq j \tag{168}$$

It can also be shown that $A_{j+1,j}$ is equivalent to $\Gamma_{j+1,j}$, computed in stage 2d. In analogy to stage 2(b)i, the matrix element $A_{j+1,j}$ is given by the projection of $\vec{\boldsymbol{v}}_{j+1}$ in stage 2d, on the final $\vec{\boldsymbol{v}}_{j+1}$, obtained in stage 2e. In stage 2d, $\vec{\boldsymbol{v}}_{j+1}$ is in its final direction, but is still unnormalized. The projection of a vector on a unit vector in the same direction gives its norm. This clarifies the definition of $\Gamma_{j+1,j}$ in stage 2(b)i.

The matrix elements $\Gamma_{i,j}$ with $i > j+1$ are not computed in the algorithm. It can be easily seen that they vanish. The expression $A\vec{\boldsymbol{v}}_j$ belongs to the Krylov space of dimension $j+2$. The vectors $\vec{\boldsymbol{v}}_i$ with $i > j+1$ are orthogonal to this space, and hence, the scalar product vanishes.

Now we are able to construct the matrix representation of A in the reduced orthonormalized Krylov basis. It will be denoted by Γ. The matrix is a square matrix of dimension $L+1$, with the following structure:

$$\Gamma \equiv \begin{bmatrix} \Gamma_{0,0} & \Gamma_{0,1} & \Gamma_{0,2} & \cdots & \Gamma_{0,L-1} & \Gamma_{0,L} \\ \Gamma_{1,0} & \Gamma_{1,1} & \Gamma_{1,2} & \cdots & \Gamma_{1,L-1} & \Gamma_{1,L} \\ & \Gamma_{2,1} & \Gamma_{2,2} & \cdots & \Gamma_{2,L-1} & \Gamma_{2,L} \\ & & \Gamma_{3,2} & \cdots & \Gamma_{3,L-1} & \Gamma_{3,L} \\ & \mathbf{0} & & \ddots & \vdots & \vdots \\ & & & & \Gamma_{L,L-1} & \Gamma_{L,L} \end{bmatrix} \tag{169}$$

The matrix structure is similar to an upper triangular matrix, with zero elements below the first subdiagonal. A matrix of this structure is called an *upper Hessenberg matrix*. In the context of the Arnoldi process, Γ is referred as the *Hessenberg matrix of A*. Note that in the algorithm, we compute also $\Gamma_{L+1,L}$, which is not included in the definition of Γ. It

will be useful to define also an extended matrix $\bar{\Gamma}$, of dimension $(L+2)\times(L+1)$. $\bar{\Gamma}$ is given by the extension of Γ by one row, in the following way:

$$\bar{\Gamma} \equiv \begin{bmatrix} & & \Gamma & & \\ 0 & 0 & \cdots & 0 & \Gamma_{L+1,L} \end{bmatrix} \tag{170}$$

Note that the last column of $\bar{\Gamma}$ is computed in the algorithm during the process of obtaining $\vec{\boldsymbol{v}}_{L+1}$. We see that this process is necessary for the computation of the Hessenberg matrix, even though $\vec{\boldsymbol{v}}_{L+1}$ does not participate in the approximation itself, which takes place in a Krylov space of dimension $L+1$ only. The computation of an additional vector costs an additional matrix-vector multiplication. Thus, in the Arnoldi process, we need $L+1$ matrix-vector multiplications, in comparison to L for a polynomial approximation of the same order. Nevertheless, the extension of the Krylov space by one dimension is useful for the estimation of the error of the approximation, as will be seen.

The Arnoldi process can be summarized by $L+1$ vector equations in the following way:

$$\vec{\boldsymbol{v}}_{n+1} = \frac{A\vec{\boldsymbol{v}}_n - \sum_{k=0}^{n} \Gamma_{k,n}\vec{\boldsymbol{v}}_k}{\Gamma_{n+1,n}}, \qquad 0 \le n \le L \tag{171}$$

or,

$$A\vec{\boldsymbol{v}}_n = \sum_{k=0}^{n+1} \Gamma_{k,n}\vec{\boldsymbol{v}}_k, \qquad 0 \le n \le L \tag{172}$$

These equations can be written compactly in a matrix form:

$$A\Upsilon = \bar{\Upsilon}\bar{\Gamma} \tag{173}$$

where Υ is an $N\times(L+1)$ matrix (N denotes the dimension of A) which its columns are the $\vec{\boldsymbol{v}}_n$'s:

$$\Upsilon \equiv \begin{bmatrix} \vec{\boldsymbol{v}}_0, & \vec{\boldsymbol{v}}_1, & \cdots & \vec{\boldsymbol{v}}_L \end{bmatrix} \tag{174}$$

and $\bar{\Upsilon}$ is an extension of Υ by one column, which contains $\vec{\boldsymbol{v}}_{L+1}$:

$$\bar{\Upsilon} \equiv \begin{bmatrix} \vec{\boldsymbol{v}}_0, & \vec{\boldsymbol{v}}_1, & \cdots & \vec{\boldsymbol{v}}_L, & \vec{\boldsymbol{v}}_{L+1} \end{bmatrix} \tag{175}$$

We can write Eq. (173) in an alternative form, which does not involve the extended matrices. We utilize the fact that the last row of $\bar{\Gamma}$ contains only one non-zero element, which affects only the last column of the resulting $N\times(L+1)$ matrix. It can be easily seen that we have:

$$A\Upsilon = \Upsilon\Gamma + \Gamma_{L+1,L}\,\vec{\boldsymbol{v}}_{L+1}\,\vec{\mathbf{e}}_{L+1}^{T} \tag{176}$$

where $\vec{\mathbf{e}}_n$ denotes a unit vector of dimension $L+1$, which its j'th element is given by $\delta_{j,n}$.

Now we are about to use the reduced basis, obtained by the Arnoldi iteration, for writing an approximation of $\vec{\mathbf{u}}$. The $\vec{\boldsymbol{v}}_n$'s in the orthonormalized Krylov basis representation are

given by $\vec{\mathbf{e}}_{n+1}$. Thus, we can define the vector which represents $\vec{\mathbf{v}}$ in the reduced basis in the following way:

$$\vec{\boldsymbol{\omega}} \equiv \|\vec{\mathbf{v}}\| \, \vec{\mathbf{e}}_1 \tag{177}$$

Υ has an important significance—it is the transformation matrix from the reduced Krylov basis representation to the original N dimensional representation of A and $\vec{\mathbf{v}}$:

$$\vec{\boldsymbol{\upsilon}}_n = \Upsilon \vec{\mathbf{e}}_{n+1} \tag{178}$$

Following the reasoning that was introduced in the beginning of this section, we can approximate $\vec{\mathbf{u}}$ by performing the calculation in the reduced basis representation, and transforming the result to the original basis. Let us define the corresponding vector of $f(A)\vec{\mathbf{v}}$ in the reduced representation:

$$\vec{\boldsymbol{\eta}} \equiv f(\Gamma)\vec{\boldsymbol{\omega}} \tag{179}$$

$\vec{\boldsymbol{\eta}}$ is transformed to the original representation, to yield the following approximation of $\vec{\mathbf{u}}$:

$$\vec{\mathbf{u}} \approx \Upsilon \vec{\boldsymbol{\eta}} \tag{180}$$

As was mentioned in Sec. 2.3.1, the direct computation of a function of matrix via diagonalization, in the form of Eq. (14), is very expensive numerically for a large scale matrix like A. However, it is not an expensive operation to diagonalize Γ, which is a small scale matrix, in order to calculate $\vec{\boldsymbol{\eta}}$. Thus, $\vec{\boldsymbol{\eta}}$ can be computed as

$$\vec{\boldsymbol{\eta}} = S f(D) S^{-1} \vec{\boldsymbol{\omega}} \tag{181}$$

where D is the diagonal matrix which represents Γ in the basis of the Γ's eigenvectors, and S is the transformation matrix from the diagonalized basis representation to the $\vec{\boldsymbol{\upsilon}}_n$ basis representation.

In the approximation obtained, we do not need any previous knowledge about the eigenvalue domain of A. Moreover, we do not rely on any assumption on the form of the eigenvalue domain. Hence, it can be applied also for A with a complex spectrum.

The justification that was given to the approximation in the form of Eq. (180) is intuitive rather than rigorous. Hence, it becomes unclear what is the quality of the approximation, and how to estimate the resulting error. In what follows, we shall give a fuller justification to the Arnoldi approach approximation. It will be shown that Eq. (180) is actually equivalent to a *polynomial approximation* of $\vec{\mathbf{u}}$. This will enable us to estimate the error of the approximation.

Let us return, for the moment, to the polynomial approximation methods. We shall present a new approach for the choice of the approximation polynomial $Q_L(z)$ (see Eqs. (151), (152)), which is led by different considerations from the polynomial approximations of Sec. B.1.

Until now, we used the concept of interpolation as an approximation tool for a function. Now we shall see that there exists a more intimate relation between a function of matrix and

the interpolation concept. Let $g(z)$ be a function which interpolates $f(z)$ in the eigenvalues of A, i. e. :

$$g(\lambda_j) = f(\lambda_j), \qquad j = 0, 1, \ldots, N-1 \tag{182}$$

It can be easily shown that when the set of eigenvectors spans the N dimensional space, then $g(A)$ is completely equivalent to $f(A)$:

$$g(A) = f(A). \tag{183}$$

This can be proved by showing that

$$[f(A) - g(A)]\vec{\mathbf{w}} = \vec{\mathbf{0}}$$

for an arbitrary vector $\vec{\mathbf{w}}$. Let us expand $\vec{\mathbf{w}}$ in the eigenvectors of A:

$$\vec{\mathbf{w}} = \sum_{j=0}^{N-1} w_j \vec{\boldsymbol{\varphi}}_j \tag{184}$$

This yields:

$$[f(A) - g(A)]\vec{\mathbf{w}} = \sum_{j=0}^{N-1} w_j [f(\lambda_j) - g(\lambda_j)]\vec{\boldsymbol{\varphi}}_j = \vec{\mathbf{0}} \tag{185}$$

The condition (182) on $g(z)$ is necessary for the equivalence of the *matrices* (Eq. (183)). Note that the condition for the *vector* equivalence $g(A)\vec{\mathbf{v}} = f(A)\vec{\mathbf{v}}$ may be even weaker. This happens when $\vec{\mathbf{v}}$ is spanned by a subspace of several eigenvectors. Then, it is sufficient to choose $g(z)$ which interpolates $f(z)$ in the eigenvalues of these eigenvectors only, as is apparent from (185).

We see that in order to obtain $f(A)$, we needn't know the behaviour of $f(z)$ everywhere, but only its values at the λ_j's. The origin of this somewhat surprising finding lies in the fact that the spectrum of A is *discrete*. The discrete spectrum originates in the fact that A itself is a discrete entity. Thus, $f(A)$ itself is also a discrete entity, and its approximation requires discrete knowledge on $f(z)$.

This leads to a different approach for the choice of $Q_L(z)$: Instead of trying to approximate the full behaviour of $f(z)$ by the ideal approximation polynomial of $f(z)$ in the domain, we can choose a polynomial which well represents the behaviour of $f(z)$ in several representative eigenvalues of A. This feature is satisfied by an *interpolation polynomial of* $f(z)$ in these eigenvalues. The question that remains is: How can we know the eigenvalues of A, without diagonalizing it?

Here the Arnoldi approach enters: We state that the eigenvalues of the reduced matrix Γ provide estimation of several representative eigenvalues of A itself. These can be used as interpolation points of $f(z)$. The eigenvalues of Γ tend to be distributed in the eigenvalue domain of A in the same way as the whole eigenvalue spectrum of A. In the case that several eigenvectors dominate the composition of $\vec{\mathbf{v}}$, the spectrum of Γ usually contains estimation of most of them. This is because of the second feature of the Krylov space mentioned in

the beginning of this section—the Krylov space remains in the same eigenvector subspace as $\vec{\mathbf{v}}$, and reflects its eigenvector composition, at least to some extent. This characteristic of the Γ spectrum is particularly useful for the specific approximation of $f(A)\vec{\mathbf{v}}$, since the interpolation polynomial becomes adjusted to represent the behaviour of $f(z)$ in the dominant eigenvectors in $\vec{\mathbf{v}}$.

Let us denote the eigenvalues of Γ by $\tilde{\lambda}_j$. Our approximation polynomial is a Newton interpolation polynomial,

$$Q_L(z) = \sum_{n=0}^{L} a_n R_n(z) \tag{186}$$

with

$$\begin{aligned} R_0(z) &\equiv 1 \\ R_n(z) &\equiv \prod_{j=0}^{n-1}(z - \tilde{\lambda}_j), \qquad n > 0 \end{aligned} \tag{187}$$

$\vec{\mathbf{u}}$ is approximated in the form of Eq. (158). If we follow the scheme of Sec. (B.1.1), we need L matrix-vector multiplications for this operation. This is in addition to the $L+1$ matrix-vector multiplications required for the Arnoldi iteration process. It seems that the overall cost of this approximation is more than twice the cost of the approximations of Sec. (B.1). Actually, we can avoid any additional large-scale matrix-vector multiplication, as will be seen. Thus, the overall number of the required large-scale matrix-vector multiplications will remain $L+1$.

In order to show that, we shall use the first feature of the Krylov space, mentioned in the beginning of this section. Let us consider again Eq. (152), with an arbitrary $Q_L(z)$. Both $\vec{\mathbf{v}}$ and the approximated $\vec{\mathbf{u}}$ lie in the Krylov space of dimension $L+1$, since $Q_L(A)\vec{\mathbf{v}}$ can be decomposed into the Taylor polynomial vectors (167). Thus, the operation of the matrix $Q_L(A)$ on $\vec{\mathbf{v}}$ takes place in the reduced Krylov space, as well as the operation of any of the $Q_L(A)$'s polynomial components. Hence, the whole calculation can take place in the reduced space, where A is replaced by its reduced representation, Γ, and $\vec{\mathbf{v}}$ is replaced by $\vec{\boldsymbol{\omega}}$. The result is transferred back to the full N dimensional space, to yield the same $\vec{\mathbf{u}}$ as in Eq. (152), without any further approximation. Thus, the following equality is *exact* for *any approximation polynomial* $Q_L(z)$:

$$Q_L(A)\vec{\mathbf{v}} = \Upsilon Q_L(\Gamma)\vec{\boldsymbol{\omega}} \tag{188}$$

In our particular case, $\vec{\mathbf{u}}$ is approximated as

$$\vec{\mathbf{u}} \approx \Upsilon \sum_{n=0}^{L} a_n R_n(\Gamma)\vec{\boldsymbol{\omega}} \tag{189}$$

Now we note that our approximation polynomial, which interpolates $f(z)$ in the entire spectrum of Γ, is actually equivalent to $f(\Gamma)$, by Eq. (183). Thus, we have (Cf. (179)):

$$\vec{\boldsymbol{\eta}} = \sum_{n=0}^{L} a_n R_n(\Gamma)\vec{\boldsymbol{\omega}} \tag{190}$$

Then, Eq. (189) becomes equivalent to Eq. (180). This concludes the justification to the approximation of Eq. (180), which is shown to be equivalent to a specific polynomial approximation.

The new interpretation of the Arnoldi approximation enables us to estimate the resulting error. Assuming fast convergence of the polynomial expansion with n, we can estimate the truncation error of Eq. (189) by the magnitude of the next term in the sum,

$$E = |a_{L+1}| \, \|R_{L+1}(A)\vec{\mathbf{v}}\| \tag{191}$$

In order to specify the next term, we first have to choose an additional sampling point. The additional point should be in the eigenvalue domain of A, which is unknown. It is reasonable to choose the average point of the estimated eigenvalues:

$$z_{L+1} = \frac{\sum_{j=0}^{L} \tilde{\lambda}_j}{L+1} \tag{192}$$

It should be verified that z_{L+1} is not equal to any of the $\tilde{\lambda}_j$'s. $R_{L+1}(z)$ is independent of z_{L+1}, and is given by Eq. (187) with $n = L+1$. z_{L+1} determines a_{L+1} only.

The error should be computed without additional large-scale matrix-vector multiplications. Hence, it is desirable to express $R_{L+1}(A)\vec{\mathbf{v}}$ in the terms of the reduced Krylov space, which involves only small-scale calculations. Here we encounter the problem that the expression $R_{L+1}(A)\vec{\mathbf{v}}$ belongs to a Krylov space of dimension $L+2$. Hence, A cannot be simply replaced by Γ, which contains information on the Krylov space of dimension $L+1$ only. However, the computation of the reduced basis vectors involved also the additional vector, $\vec{\boldsymbol{v}}_{L+1}$. Thus, it is still possible to obtain a reduced calculation of (191) by the information we already have. First, we write:

$$R_{L+1}(A)\vec{\mathbf{v}} = (A - \tilde{\lambda}_L) R_L(A)\vec{\mathbf{v}} \tag{193}$$

$R_L(A)\vec{\mathbf{v}}$ lies in the Krylov space of dimension $L+1$. Hence, we can express it in the terms of the $L+1$ dimensional space:

$$R_L(A)\vec{\mathbf{v}} = \Upsilon R_L(\Gamma)\vec{\boldsymbol{\omega}} = \Upsilon \vec{\boldsymbol{\mu}} \tag{194}$$

where we defined:

$$\vec{\boldsymbol{\mu}} \equiv R_L(\Gamma)\vec{\boldsymbol{\omega}} \tag{195}$$

Eq. (193) becomes:

$$R_{L+1}(A)\vec{\mathbf{v}} = A\Upsilon\vec{\boldsymbol{\mu}} - \tilde{\lambda}_k \Upsilon \vec{\boldsymbol{\mu}} \tag{196}$$

Then, we apply Eq. (176) to yield:

$$\begin{aligned} R_{L+1}(A)\vec{\mathbf{v}} &= \left(\Upsilon\Gamma + \Gamma_{L+1,L}\, \vec{\boldsymbol{v}}_{L+1}\, \vec{\mathbf{e}}_{L+1}^{T}\right) \vec{\boldsymbol{\mu}} - \tilde{\lambda}_k \Upsilon \vec{\boldsymbol{\mu}} \\ &= \Upsilon \left(\Gamma - \tilde{\lambda}_k\right) \vec{\boldsymbol{\mu}} + \Gamma_{L+1,L}\mu_{L+1}\, \vec{\boldsymbol{v}}_{L+1} \\ &= \Upsilon R_{L+1}(\Gamma)\vec{\boldsymbol{\omega}} + \Gamma_{L+1,L}\mu_{L+1}\, \vec{\boldsymbol{v}}_{L+1} \end{aligned} \tag{197}$$

where μ_{L+1} is the $(L+1)$'th element of $\vec{\boldsymbol{\mu}}$. We see that the resulting expression is spanned by the extended orthonormalized Krylov set, which includes $\vec{\boldsymbol{\upsilon}}_{L+1}$. We can use an extended representation of the extended set, of dimension $L+2$, in order to represent $R_{L+1}(A)\vec{\mathbf{v}}$. In the extended representation, $R_{L+1}(A)\vec{\mathbf{v}}$ is defined by the following vector:

$$\vec{\bar{\boldsymbol{\mu}}} = \begin{bmatrix} R_{L+1}(\Gamma)\vec{\boldsymbol{\omega}} \\ \Gamma_{L+1,L}\mu_{L+1} \end{bmatrix} \tag{198}$$

In principle, $\vec{\bar{\boldsymbol{\mu}}}$ can be transferred back to the original representation by the extended transformation matrix, $\bar{\Upsilon}$, in the following way:

$$R_{L+1}(A)\vec{\mathbf{v}} = \bar{\Upsilon}\vec{\bar{\boldsymbol{\mu}}} \tag{199}$$

However, this operation is unnecessary, as long as we are interested only in the *norm* of this expression (see (191)). The vectors in $\bar{\Upsilon}$ are *unit vectors*, and the norm remains unaltered by the change of representation:

$$\|\bar{\Upsilon}\vec{\bar{\boldsymbol{\mu}}}\| = \|\vec{\bar{\boldsymbol{\mu}}}\| \tag{200}$$

The error is finally given by

$$E = |a_{L+1}|\,\|\vec{\bar{\boldsymbol{\mu}}}\| \tag{201}$$

The relative error is given by

$$E_{rel} = \frac{E}{\|\vec{\mathbf{u}}\|} \approx \frac{E}{\|\Upsilon\vec{\boldsymbol{\eta}}\|} = \frac{E}{\|\vec{\boldsymbol{\eta}}\|} \tag{202}$$

If we compute the error, Eq. (181) becomes unnecessary; $\vec{\boldsymbol{\eta}}$ should be computed by the Newton interpolation form, Eq. (190), utilizing the operations needed for the computation of the error. The $R_n(\Gamma)\vec{\boldsymbol{\omega}}$ terms and the a_n's are computed iteratively, as explained in Appendix A.1 and Sec. B.1.1. $R_{L+1}(\Gamma)\vec{\boldsymbol{\omega}}$ and a_{L+1} for the error estimation are obtained by continuing the iterative process to the next order.

In order to increase the numerical stability of the Newton interpolation, the size of the approximation domain should be changed, as in the case of interpolation on a one-dimensional axis (see Appendix A.1.3). In the case of a two-dimensional domain, defined on the complex plain, the problem should be transformed to a domain which its size is divided by the *capacity of the domain* [45]. We give only an expression for the estimation of the capacity, and not an exact definition. The capacity is estimated by choosing a point in the domain, z_p, and computing the following expression:

$$\rho = \left(|z_p - z_0|\,|z_p - z_1|\cdots|z_p - z_N|\right)^{\frac{1}{N+1}} \tag{203}$$

where the z_n's $(n = 0, 1, \ldots, N)$ are the sampling points, and $N+1$ is the number of sampling points. z_p can be chosen as the average point of the sampling points, as in Eq. (192). In the case of interpolation on an axis, in a domain of length 4 (see Appendix A.1.3), the capacity is 1. This can be observed intuitively from (203), by choosing z_p in the middle of the domain. If the capacity of the domain is different from 1, we should perform a

transformation similar to that described in Appendix A.1.3, where the conversion factor of $4/\Delta x$ is replaced by $1/\rho$. In practice, the instructions at the end of Sec. (B.1.1) should be followed, with the appropriate conversion factor.

The interpolation polynomial interpretation of the Arnoldi approach reveals an important advantage of the Arnoldi algorithm over the polynomial approximations. This advantage exists in certain circumstances, even for A with eigenvalues distributed on a one-dimensional axis. If most of the eigenvalues are concentrated in a portion of the eigenvalue domain, with an additional small number of spread eigenvalues in the whole domain, the Arnoldi approach is expected to be more efficient. The reason is that the $\tilde{\lambda}$'s are distributed in the eigenvalue domain in a similar way to the λ's. Thus, the important region in which most of the eigenvalues are concentrated is better represented than the less important regions. In contrary, the Chebyshev sampling is uniformly distributed in the domain.

In the application of the time-dependent Hamiltonian propagator, we apply a time-step scheme. The short time intervals can be treated by a relatively small Krylov space, typically—up to $L = 15$. When the Hamiltonian is time-independent, the whole time-interval is treated in a single step, and the required approximation space is large. The Arnoldi approach usually becomes problematic in a large space. The main reasons are:

1. In the Arnoldi process, we store $L + 2$ vectors in the memory. For very large N and large L, this might be impermissible.

2. During the Arnoldi process, we perform $(L + 1)^2/2$ scalar products, each of which scales as N. This becomes quite demanding for large L.

When a large dimension approximation is required, a *restarted Arnoldi algorithm* should be used (see, for example, [47]). This topic is beyond the scope of this paper.

C Conversion schemes of polynomial expansions to a Taylor form

C.1 Conversion scheme for a Newton expansion

C.1.1 Conversion scheme for the $q_{n,m}$ coefficients

Let us write the Newton expansion form for $\vec{\mathbf{s}}(t)$ (Cf. Eq. (107)):

$$\vec{\mathbf{s}}(t) \approx \sum_{n=0}^{M-1} \vec{\mathbf{a}}_n R_n(t) \tag{204}$$

The $R_n(t)$'s satisfy the following recursion formula (see Eq. (108)):

$$R_{n+1}(t) = (t - t_n) R_n(t) \tag{205}$$

where

$$R_0(t) = 1 \tag{206}$$

and the t_n's are the sampling points.

We need the conversion coefficients of the $R_n(t)$'s to a Taylor form (Cf. Eq. (51)):

$$R_n(t) = \sum_{m=0}^{n} q_{n,m} \frac{t^m}{m!} \tag{207}$$

It can be immediately observed from Eq. (206) that:

$$q_{0,0} = 1 \tag{208}$$

The rest of the $q_{n,m}$'s can be computed from Eq. (208) by the derivation of recurrence relations. Plugging Eq. (207) into (205) we obtain:

$$R_{n+1}(t) = \sum_{m=0}^{n} q_{n,m} \frac{t^{m+1}}{m!} - t_n \sum_{m=0}^{n} q_{n,m} \frac{t^m}{m!} = \sum_{m=1}^{n+1} q_{n,m-1} \frac{t^m}{(m-1)!} - t_n \sum_{m=0}^{n} q_{n,m} \frac{t^m}{m!} \tag{209}$$

On the other hand, we have from (207):

$$R_{n+1}(t) = \sum_{m=0}^{n+1} q_{n+1,m} \frac{t^m}{m!} \tag{210}$$

Equating the RHS of Eqs. (209) and (210) we obtain:

$$\sum_{m=0}^{n+1} q_{n+1,m} \frac{t^m}{m!} = \sum_{m=1}^{n+1} q_{n,m-1} \frac{t^m}{(m-1)!} - t_n \sum_{m=0}^{n} q_{n,m} \frac{t^m}{m!} \tag{211}$$

Equating the coefficients of similar powers of t, we obtain the following recurrence relations for the $q_{n,m}$'s:

$$q_{n+1,0} = -t_n q_{n,0} \tag{212}$$

$$q_{n+1,m} = m q_{n,m-1} - t_n q_{n,m} \qquad 1 \le m \le n \tag{213}$$

$$q_{n+1,n+1} = (n+1) q_{n,n} \tag{214}$$

Starting from Eq. (208), all $q_{m,n}$'s can be computed in a recursive manner, using Eqs. (212)-(214).

Once the $q_{n,m}$'s are obtained, the $\vec{\mathbf{s}}_m$ Taylor coefficients can be computed by (Cf. Eq. (54)):

$$\vec{\mathbf{s}}_m = \sum_{n=m}^{M-1} q_{n,m} \vec{\mathbf{a}}_n \tag{215}$$

C.1.2 Conversion scheme for the $\tilde{q}_{n,m}$ coefficients

As was mentioned in Sec. 3.3, for numerical stability, it is recommended to absorb the $1/m!$ factor in the $\vec{\mathbf{s}}_m$ Taylor coefficients from Eq. (22). Accordingly, Eq. (207) is replaced by:

$$R_n(t) = \sum_{m=0}^{n} \tilde{q}_{n,m} t^m \tag{216}$$

The recurrence relations for the $\tilde{q}_{n,m}$'s are slightly different from those of the $q_{n,m}$'s. Following the same steps as above, we obtain the recursion formulas:

$$\tilde{q}_{n+1,0} = -t_n \tilde{q}_{n,0} \tag{217}$$

$$\tilde{q}_{n+1,m} = \tilde{q}_{n,m-1} - t_n \tilde{q}_{n,m} \qquad 1 \le m \le n \tag{218}$$

$$\tilde{q}_{n+1,n+1} = \tilde{q}_{n,n} \tag{219}$$

In addition, we have from (206):

$$\tilde{q}_{0,0} = 1 \tag{220}$$

which completes the required information for computing the $\tilde{q}_{n,m}$'s recursively.

The $\vec{\tilde{\mathbf{s}}}_m$'s are computed by:

$$\vec{\tilde{\mathbf{s}}}_m = \sum_{n=m}^{M-1} \tilde{q}_{n,m} \vec{\mathbf{a}}_n \tag{221}$$

C.1.3 Conversion scheme for a length 4 domain

We mentioned in Sec. A.1 that it is recommended to use a domain of length 4 in a Newton expansion, for numerical stability. When transferring the original t domain to a length 4 domain, the recursion formula for the $R_n(t)$'s gains an additional conversion factor (Cf. Eq. (205)):

$$R_{n+1}(t) = \frac{4}{\Delta t}(t - t_n) R_n(t) \tag{222}$$

where Δt is the length of the original domain. Accordingly, the RHS of the recurrence relations (212)-(214) and (217)-(219) is multiplied by the same factor, to yield:

$$q_{n+1,0} = -\frac{4}{\Delta t} t_n q_{n,0} \tag{223}$$

$$q_{n+1,m} = \frac{4}{\Delta t}(m q_{n,m-1} - t_n q_{n,m}) \qquad 1 \le m \le n \tag{224}$$

$$q_{n+1,n+1} = \frac{4}{\Delta t}(n+1) q_{n,n} \tag{225}$$

and

$$\tilde{q}_{n+1,0} = -\frac{4}{\Delta t} t_n \tilde{q}_{n,0} \tag{226}$$

$$\tilde{q}_{n+1,m} = \frac{4}{\Delta t}(\tilde{q}_{n,m-1} - t_n \tilde{q}_{n,m}) \qquad 1 \le m \le n \tag{227}$$

$$\tilde{q}_{n+1,n+1} = \frac{4}{\Delta t} \tilde{q}_{n,n} \tag{228}$$

The $\vec{\mathbf{s}}_m$'s and the $\vec{\tilde{\mathbf{s}}}_m$'s are computed as in Eqs. (215), (221).

C.2 Conversion scheme for a Chebyshev expansion

The Chebyshev expansion is defined for the domain $[-1, 1]$. Suppose we want approximate $\vec{\mathbf{s}}(t)$ in an arbitrary domain $t \in [t_{min}, t_{max}]$ by a Chebyshev expansion. We define a variable $y \in [-1, 1]$ such that

$$y \equiv \frac{2t - t_{min} - t_{max}}{\Delta t} \tag{229}$$

where $\Delta t = t_{max} - t_{min}$ (see Sec. A.2). Then we expand $\vec{\mathbf{s}}(t)$ in a Chebyshev series:

$$\vec{\mathbf{s}}(t) \approx \sum_{n=0}^{M-1} \vec{\mathbf{c}}_n T_n(y) = \sum_{n=0}^{M-1} \vec{\mathbf{c}}_n T_n\left(\frac{2t - t_{min} - t_{max}}{\Delta t}\right) \tag{230}$$

Let us define the following set of polynomials:

$$\phi_n(t) \equiv T_n\left(\frac{2t - t_{min} - t_{max}}{\Delta t}\right) \tag{231}$$

Note that (229) is a linear transformation. Hence, $\phi_n(t)$ remains a polynomial of degree n, like $T_n(y)$. The Chebyshev expansion can be rewritten as a polynomial series in the terms of the $\phi_n(t)$'s:

$$\vec{\mathbf{s}}(t) \approx \sum_{n=0}^{M-1} \vec{\mathbf{c}}_n \phi_n(t) \tag{232}$$

Using this form, the coefficients of the Taylor like form, $\vec{\mathbf{s}}_m$, can be computed from the Chebyshev coefficients, $\vec{\mathbf{c}}_m$, via Eq. (54).

First, we expand the $\phi_n(t)$'s in a Taylor form (Cf. Eq. (51)):

$$\phi_n(t) = \sum_{m=0}^{n} q_{n,m} \frac{t^m}{m!} \tag{233}$$

We can utilize the recurrence relation between the Chebyshev polynomials in order to find the $q_{n,m}$'s. The Chebyshev polynomials satisfy the following recursion formula:

$$T_{n+1}(y) = 2yT_n(y) - T_{n-1}(y) \tag{234}$$

where

$$T_0(y) = 1 \tag{235}$$
$$T_1(y) = y \tag{236}$$

The recursion formula can be rewritten in the terms of t and the $\phi_n(t)$'s:

$$\phi_{n+1}(t) = \frac{4t - 2(t_{min} + t_{max})}{\Delta t}\phi_n(t) - \phi_{n-1}(t) \tag{237}$$

where

$$\phi_0(t) = 1 \tag{238}$$
$$\phi_1(t) = \frac{2t - t_{min} - t_{max}}{\Delta t} \tag{239}$$

It can be observed from Eqs. (238), (239), that:

$$q_{0,0} = 1 \tag{240}$$
$$q_{1,0} = -\frac{t_{min} + t_{max}}{\Delta t} \tag{241}$$
$$q_{1,1} = \frac{2}{\Delta t} \tag{242}$$

Here, again, the rest of the $q_{n,m}$'s can be obtained from Eqs. (240)-(242) by the derivation of recurrence relations.

Plugging Eq. (233) into (237) we obtain:

$$\begin{aligned}\phi_{n+1}(t) &= \frac{4}{\Delta t}\sum_{m=0}^{n} q_{n,m}\frac{t^{m+1}}{m!} - \frac{2(t_{min} + t_{max})}{\Delta t}\sum_{m=0}^{n} q_{n,m}\frac{t^m}{m!} - \sum_{m=0}^{n-1} q_{n-1,m}\frac{t^m}{m!} \\ &= \frac{4}{\Delta t}\sum_{m=1}^{n+1} q_{n,m-1}\frac{t^m}{(m-1)!} - \frac{2(t_{min} + t_{max})}{\Delta t}\sum_{m=0}^{n} q_{n,m}\frac{t^m}{m!} - \sum_{m=0}^{n-1} q_{n-1,m}\frac{t^m}{m!}\end{aligned} \tag{243}$$

On the other hand, from (233) we have:

$$\phi_{n+1}(t) = \sum_{m=0}^{n+1} q_{n+1,m}\frac{t^m}{m!} \tag{244}$$

From Eqs. (243), (244), we obtain:

$$\sum_{m=0}^{n+1} q_{n+1,m}\frac{t^m}{m!} = \frac{4}{\Delta t}\sum_{m=1}^{n+1} q_{n,m-1}\frac{t^m}{(m-1)!} - \frac{2(t_{min} + t_{max})}{\Delta t}\sum_{m=0}^{n} q_{n,m}\frac{t^m}{m!} - \sum_{m=0}^{n-1} q_{n-1,m}\frac{t^m}{m!} \tag{245}$$

The recurrence relations are obtained by equating the coefficients of similar powers of t:

$$q_{n+1,0} = -\frac{2(t_{min}+t_{max})}{\Delta t} q_{n,0} - q_{n-1,0} \tag{246}$$

$$q_{n+1,m} = \frac{4}{\Delta t} m q_{n,m-1} - \frac{2(t_{min}+t_{max})}{\Delta t} q_{n,m} - q_{n-1,m} \qquad 1 \le m \le n-1 \tag{247}$$

$$q_{n+1,n} = \frac{4}{\Delta t} n q_{n,n-1} - \frac{2(t_{min}+t_{max})}{\Delta t} q_{n,n} \tag{248}$$

$$q_{n+1,n+1} = \frac{4}{\Delta t}(n+1) q_{n,n} \tag{249}$$

The Taylor like coefficients are given by (Cf. Eq. (54)):

$$\vec{\mathbf{s}}_m = \sum_{n=m}^{M-1} q_{n,m} \vec{\mathbf{c}}_n \tag{250}$$

The $\tilde{q}_{n,m}$'s can be computed in an analogous manner to the $q_{n,m}$'s. From Eqs. (238), (239), we have:

$$\tilde{q}_{0,0} = 1 \tag{251}$$

$$\tilde{q}_{1,0} = -\frac{t_{min}+t_{max}}{\Delta t} \tag{252}$$

$$\tilde{q}_{1,1} = \frac{2}{\Delta t} \tag{253}$$

Using the same technique as for the $q_{n,m}$'s, we obtain the following recurrence relations:

$$\tilde{q}_{n+1,0} = -\frac{2(t_{min}+t_{max})}{\Delta t} \tilde{q}_{n,0} - \tilde{q}_{n-1,0} \tag{254}$$

$$\tilde{q}_{n+1,m} = \frac{4}{\Delta t} \tilde{q}_{n,m-1} - \frac{2(t_{min}+t_{max})}{\Delta t} \tilde{q}_{n,m} - \tilde{q}_{n-1,m} \qquad 1 \le m \le n-1 \tag{255}$$

$$\tilde{q}_{n+1,n} = \frac{4}{\Delta t} \tilde{q}_{n,n-1} - \frac{2(t_{min}+t_{max})}{\Delta t} \tilde{q}_{n,n} \tag{256}$$

$$\tilde{q}_{n+1,n+1} = \frac{4}{\Delta t} \tilde{q}_{n,n} \tag{257}$$

The $\vec{\tilde{\mathbf{s}}}_m$'s are given by

$$\vec{\tilde{\mathbf{s}}}_m = \sum_{n=m}^{M-1} \tilde{q}_{n,m} \vec{\mathbf{c}}_n \tag{258}$$

D Error estimation and control

One of the most important issues in any numerical method is the ability to estimate the magnitude of the error of the method. When the error can be estimated, it can usually

be controlled by altering the parameters of the approximation. Thus, we should point out the possible sources of inaccuracy in the propagation procedure, and provide estimations of the magnitude of the error.

First, we focus on the error of the local solution *in a given time-step and a given iteration*. Then, we discuss the relation between the local errors and the global error of the algorithm, i. e. the error of the whole propagation process.

D.1 Local error

The local solution is computed in step 2(c)vi of the algorithm in Sec. 3.2. There are three sources of inaccuracy in this computation:

1. *Convergence error*: The computation is based on the previous $\vec{\mathbf{u}}(t_{k,l})$, i. e. the guess solution or the solution from the previous iteration (step 2(c)i);

2. *Time-discretization error*: The time behaviour of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t)$ is approximated from sampling at the discrete Chebyshev points (steps 2(c)i, 2(c)ii);

3. *Function of matrix computation error*: $f_{M_k}(\widetilde{G}, t_{k,l} - t_{k,0})\vec{\mathbf{v}}_{M_k}$ is approximated by a polynomial expansion, or by the Arnoldi approach.

The different sources of inaccuracy result in an inadequate representation of Eq. (65) by the algorithm. The effects of the different inaccuracy sources on this representation vary. In any case, the algorithm does not represent Eq. (65) accurately, but something else. It is important to understand what the algorithm does represent, for the understanding of the behaviour of the algorithm in each situation in which the algorithm fails to yield the required accuracy. The different situations can be classified as follows:

1. The algorithm represents an equation which differs from Eq. (65); we can distinguish between two situations, which correspond to different sources of inaccuracy:

   (a) *Time-discretization error*: The time-sampling is insufficient to represent $G(t)$ or $\vec{\mathbf{s}}(t)$ properly. We can view this situation in the following way: We actually solve an equation of the general form of Eq. (56), but for another problem, in which $G(t)$ or $\vec{\mathbf{s}}(t)$ are replaced by their truncated time-expansions;

   (b) *Function of matrix computation error*: The expansion of $f_{M_k}(\widetilde{G}, t_{k,l} - t_{k,0})\vec{\mathbf{v}}_{M_k}$ does not approximate the expression properly. In this case, the equation represented by the algorithm does not correspond to an equation of the form of Eq. (56).

2. The algorithm does not represent a continuous *equation* of time, but a discretized *vector problem* in time. The algorithm is based on samplings of $\vec{\mathbf{u}}(t)$ at several discrete time-points, which constitute a time-vector. When the time-sampling is insufficient to represent the time-behaviour of $\vec{\mathbf{u}}(t)$ properly, a time-discretization error results. In such a case, the algorithm does represent the requirement that

Eq. (65) will be satisfied at the sampling Chebyshev time-points, but fails to represent the requirement that it will be satisfied at the intermediate points. One outcome is that the resulting $\vec{\mathbf{v}}_j$'s become inaccurate, and so is $\vec{\mathbf{u}}(t)$. Another outcome is that Eq. (65) is not satisfied at the intermediate points, even with the resulting $\vec{\mathbf{v}}_j$'s.

3. The algorithm does not represent an equality at all. Eq. (65) is an implicit equation, because of the dependence of the $\vec{\mathbf{v}}_j$'s on $\vec{\mathbf{u}}(t)$. Step 2(c)vi would have represented it accurately only if the $\vec{\mathbf{u}}(t_{k,l})$'s in step 2(c)i were exact. Because of the convergence error, step 2(c)vi may represent the equality only within the required extent of accuracy. If the convergence error is too large, the algorithm fails to represent the equality to the required accuracy.

In what follows, we give estimations to the magnitude of the error for the three inaccuracy sources, and discuss the ways to control the error for each source.

D.1.1 Convergence error

An estimation of the convergence error is already included in the algorithm in Sec. 3.2, as the convergence criterion in step 2(c)vii. The convergence rate of the iterative process can be assumed to be fast. Consequently, the error of $\vec{\mathbf{u}}_{old}$ is larger by orders of magnitude than the error of the new solution. Hence, the $\vec{\mathbf{u}}(t_{k,M_k-1})$ obtained in step 2(c)vi can practically represent the accurate solution in this context. Thus, the convergence criterion yields an excellent approximation to the relative error of the old solution at the edge of the time-step,

$$\frac{\|\vec{\mathbf{u}}_{old} - \vec{\mathbf{u}}(t_{k,M_k-1})\|}{\|\vec{\mathbf{u}}(t_{k,M_k-1})\|}$$

where $\vec{\mathbf{u}}(t)$ here is the *exact solution*. We assumed that $\|\vec{\mathbf{u}}_{old}\|$ in the denominator of step 2(c)vii is close enough to the converged solution, and can safely replace $\|\vec{\mathbf{u}}(t_{k,M_k-1})\|$. Assuming that the iterative process converges, the error of the old solution yields an upper limit to the error of the new one.

The problem with this estimation is that it greatly overestimates the convergence error, since the error of the old solution is assumed to be larger than the error of the new one by several orders of magnitude. Much better estimations to the convergence error of the new solution can be obtained with some extra numerical effort. This topic is left for a future publication.

In the algorithm in Sec. 3.2, the magnitude of the convergence error is controlled by the number of iterations. The convergence error can be controlled also by altering the other parameters of the algorithm. A decrement of the length of the time-step, Δt_k, is effective in the reduction of the convergence error. An increment of the number of expansion terms in the time-expansion *in the previous time-step*, M_{k-1}, may be also helpful, since the guess solution becomes more accurate (unless M_{k-1} becomes too large; see Sec. 3.4). The same is true for K_{k-1}, the number of expansion terms for the function of matrix in the previous time-step.

D.1.2 Time-discretization error

The estimation of the time-discretization error requires some additional insight into the origin of the error. The time-discretization error actually results from the replacement of the exact $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t),t)$, by another time-dependent inhomogeneous term, which approximates it by interpolation at the Chebyshev points. Let us denote the approximated $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t),t)$ by $\vec{\mathbf{s}}^{int}_{ext}(t)$. The absolute time-discretization error is obtained by the difference between the solutions of two different problems—the approximated problem and the actual one:

$$E^{int}(t) = \left\| \vec{\mathbf{u}}^{int}(t) - \vec{\mathbf{u}}(t) \right\| \tag{259}$$

where $\vec{\mathbf{u}}^{int}(t)$ denotes the solution of the problem with $\vec{\mathbf{s}}^{int}_{ext}(t)$ as the inhomogeneous term. By the Duhamel principle (Eqs. (20), (61)), we have:

$$\vec{\mathbf{u}}(t) = \exp\left[\widetilde{G}(t-t_{k,0})\right]\vec{\mathbf{u}}(t_{k,0}) + \int_{t_{k,0}}^{t} \exp\left[\widetilde{G}(t-\tau)\right]\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(\tau),\tau)\,d\tau \tag{260}$$

$$\vec{\mathbf{u}}^{int}(t) = \exp\left[\widetilde{G}(t-t_{k,0})\right]\vec{\mathbf{u}}(t_{k,0}) + \int_{t_{k,0}}^{t} \exp\left[\widetilde{G}(t-\tau)\right]\vec{\mathbf{s}}^{int}_{ext}(\tau)\,d\tau \tag{261}$$

and thus:

$$\begin{aligned} E^{int}(t) &= \left\| \int_{t_{k,0}}^{t} \exp\left[\widetilde{G}(t-\tau)\right]\left[\vec{\mathbf{s}}^{int}_{ext}(\tau) - \vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(\tau),\tau)\right]\,d\tau \right\| \\ &= \left\| \int_{t_{k,0}}^{t} \exp\left[\widetilde{G}(t-\tau)\right]\Delta\vec{\mathbf{s}}^{int}_{ext}(\tau)\,d\tau \right\| \end{aligned} \tag{262}$$

where we defined:

$$\Delta\vec{\mathbf{s}}^{int}_{ext}(t) \equiv \vec{\mathbf{s}}^{int}_{ext}(t) - \vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t),t). \tag{263}$$

The integral expression can be readily used to yield an upper limit for the error:

$$\begin{aligned} E^{int}(t) &\leq \max_{\tau\in[t_{k,0},t]} \left\| \exp\left[\widetilde{G}(t-\tau)\right]\Delta\vec{\mathbf{s}}^{int}_{ext}(\tau) \right\| (t-t_{k,0}) \\ &\equiv \left\| \exp\left[\widetilde{G}(t-t_{max})\right]\Delta\vec{\mathbf{s}}^{int}_{ext}(t_{max}) \right\| (t-t_{k,0}) \end{aligned} \tag{264}$$

where t_{max} denotes the time-point in the interval $[t_{k,0},t]$, which maximizes the magnitude of the expression.

Next, we utilize the fact that the time-step is assumed to be small, due to the stability requirements of the algorithm. Hence, we can assume that

$$\left\| \exp\left[\widetilde{G}(t-t_{max})\right]\Delta\vec{\mathbf{s}}^{int}_{ext}(t_{max}) \right\| \approx \left\| \Delta\vec{\mathbf{s}}^{int}_{ext}(t_{max}) \right\| \tag{265}$$

Note that for a Hermitian Hamiltonian, $\widetilde{G}$ becomes anti-Hermitian, and Eq. (265) becomes exact.

At the edge of the time-step, the expression for the absolute time-discretization error yields the following estimation:

$$E^{int}(t_{k,M_k-1}) \approx \left\| \Delta\vec{\mathbf{s}}^{int}_{ext}(t_{max}) \right\| \Delta t_k \tag{266}$$

The relative error can be estimated by

$$E^{int}_{rel}(t_{k,M_k-1}) \approx \frac{\left\| \Delta\vec{\mathbf{s}}^{int}_{ext}(t_{max}) \right\| \Delta t_k}{\left\| \vec{\mathbf{u}}^{int}(t_{k,M_k-1}) \right\|} \tag{267}$$

Observing Eqs. (266), (267), one encounters the problem that t_{max} is unknown. However, the precise knowledge of t_{max} is unnecessary, since we need only an estimation for the order of magnitude of the error. It is reasonable to use the point which is furthest from neighbouring sampling points instead of the precise t_{max}. Hence, we can choose the middle point between t_{mid} and the next Chebyshev point.

$\Delta\vec{\mathbf{s}}^{int}_{ext}(t)$ at the estimated t_{max} can be computed directly, via Eq. (263). $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t)$ at the estimated t_{max} is computed in the same way as the samplings of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t)$ at the Chebyshev points (step 2(c)i of the algorithm in Sec. 3.2). $\vec{\mathbf{s}}^{int}_{ext}(t)$ at the estimated t_{max} is computed by the evaluation of the approximation polynomial of $\vec{\mathbf{s}}_{ext}(\vec{\mathbf{u}}(t), t)$ at the point, using the coefficients computed in step 2(c)ii of the algorithm.

This error estimation tends to overestimate the time-discretization error, typically by one or two orders of magnitude at the time-step edge. The reason lies in the oscillatory nature of $\Delta\vec{\mathbf{s}}^{int}_{ext}(t)$, which is responsible for cancellation of errors during the integration in Eq. (262). An accurate estimation of the time-discretization error requires a more detailed analysis. This topic is left for a future publication.

The magnitude of the time-discretization error can be primarily controlled by the number of Chebyshev time-points, M_k. However, M_k cannot be increased indefinitely in order to reduce the error, because of reduction in the efficiency of the algorithm in higher orders (see Sec. 3.4). An alternative option for error reduction is the decrement of Δt_k.

D.1.3 Function of matrix computation error

The estimation of the function of matrix computation error is more direct. It can be readily observed from the computation in step 2(c)vi that the absolute error of the computation of $f_{M_k}(\widetilde{G}, t_{k,l} - t_{k,0})\vec{\mathbf{v}}_{M_k}$ is just the same as the resulting absolute error of $\vec{\mathbf{u}}(t_{k,l})$ itself. Let us denote the absolute error as $E^{fm}(t)$; the relative error at the edge of the time-step can be estimated by

$$E^{fm}_{rel}(t_{k,M_k-1}) = \frac{E^{fm}(t_{k,M_k-1})}{\left\| \vec{\mathbf{u}}^{fm}(t_{k,M_k-1}) \right\|} \tag{268}$$

where $\vec{\mathbf{u}}^{fm}(t)$ denotes the solution resulting from the approximation of $f_{M_k}(\widetilde{G}, t - t_{k,0})\vec{\mathbf{v}}_{M_k}$. An estimation to $E^{fm}(t)$ is given by an estimation of the truncation error of the expansion of $f_{M_k}(\widetilde{G}, t - t_{k,0})\vec{\mathbf{v}}_{M_k}$. In the case that a polynomial expansion approximation is used, one simple way to estimate the truncation error is by computation of the error of the

same approximation at several test points; the value of $f_{M_k}(z, \Delta t_k)$ is interpolated in z at several representative test points in the eigenvalue domain, and the relative error from the exact value is computed; the estimated $E^{fm}(t_{k,M_k-1})$ is given by the multiplication of the obtained relative error by $\|f_{M_k}(\widetilde{G}, \Delta t_k)\vec{\mathbf{v}}_{M_k}\|$. An estimation of the error for the Arnoldi approach is given in Appendix B.2.

The magnitude of the function of matrix computation error can be primarily controlled by the number of expansion terms for the approximation, K_k. A decrement of Δt_k also reduces the error.

D.2 Global error

Our main interest is in the estimation of the global error of the final solution obtained by the algorithm. One would have suggest that the global error of the algorithm is just an additive sum of the local errors in each time-step. This is indeed the case when the propagation process is numerically stable. However, it is important to be aware of the fact that the global behaviour of the algorithm might be different; we may observe three distinguished behaviours of error accumulation in the algorithm:

1. *Additive accumulation*: The final error is the sum of the errors of the solutions in the last iteration of each time-step;

2. *Divergence with propagation*: The error accumulates in an explosive manner during the propagation. The magnitude of the solution tends to infinity *with the propagation*;

3. *Divergence of the iterative process*: The iterative process in a specific time-step fails to converge. The magnitude of the solution tends to infinity *with the number of iterations*.

In the last two behaviours, the estimation of the local error is useless for the estimation of the global error. The divergence of the iterative process can be always detected, since the algorithm fails to continue the propagation process. Most frequently, the divergence with propagation is also easily detected, by the occurrence of an overflow, or an unreasonably large magnitude of the solution. Seldom, it may occur that the divergent process has stopped at an early stage at the end of the propagation, and the magnitude of the solution is not unreasonable. In this case, it might not be easy to detect the divergent behaviour of the error.

Of course, it is highly desirable to prevent an unstable behaviour of the algorithm. First, we should attribute the different unstable behaviours to the responsible causes.

The divergence of the iterative process clearly originates in a too large time-step, which is outside the convergence radius of the algorithm. If a divergence of the iterative process occurs, the length of the time-step should be decreased.

The origin of the divergence with propagation is less obvious. Let us recall the classification into the different situations in which the algorithm fails to represent Eq. (65) adequately. When the algorithm represents an equation of the general form of Eq. (56)

(situation 1a above), the behaviour of the solution is expected to preserve the characteristic features of Eq. (56). For instance, in the homogeneous Schrödinger equation (with a Hermitian Hamiltonian) the norm of the solution is expected to be conserved. Hence, a divergent behaviour with the propagation is not expected. In contrary, the behaviour of the algorithm in the other situations is unexpected.

In practice, a divergent behaviour was observed only in situation 1b above, i. e. when there is a function of matrix computation error. Experience shows that only a low accuracy expansion of $f_{M_k}(\widetilde{G}, t_{k,l} - t_{k,0})\vec{\mathbf{v}}_{M_k}$ leads to a divergent behaviour. The instability disappears when more expansion terms are used. We can conclude that it is not recommended to expand $f_{M_k}(z, t_{k,l} - t_{k,0})$ by a low accuracy expansion, even when a low accuracy solution is sufficient. Further research is required to estimate the maximal allowed inaccuracy in the expansion for stability of the propagation. In the problems that were tested so far, the following criterion was found to be sufficient for stability:

$$\frac{E^{fm}(t_{k,M_k-1})}{\left\| f_{M_k}(\widetilde{G}, \Delta t_k)\vec{\mathbf{v}}_{M_k} \right\|} < 10^{-5}$$

It is noteworthy that a divergent behaviour with the propagation might be observed also for the convergence error, in a different version of the algorithm than that presented in Sec. 3.2; if one restricts the allowed number of iterations in each time-step (like in the numerical example of Sec. 4), a divergence with propagation might occur. This phenomenon is typical for high order M values. The use of high order M is not recommended anyway, by efficiency considerations (see Sec. 3.4).

It should be noted that the phenomenon of divergence with propagation is common to other propagators, when the time-step is too large.

The inclusion of tests for the magnitude of the error in the algorithm increases the robustness of the algorithm. The tests may be also used for an adaptive choice of the parameters during the propagation. In the future, we plan to develop an improved version of the algorithm, based on this principle.

References

[1] Guy Ashkenazi, Ronnie Kosloff, Sanford Ruhman, and Hillel Tal-Ezer, *Newtonian propagation methods applied to the photodissociation dynamics of I_3^-*, The Journal of Chemical Physics **103** (1995), no. 23, 10005–10014.

[2] Philipp Bader, Arieh Iserles, Karolina Kropielnicka, and Pranav Singh, *Efficient methods for linear Schrödinger equation in the semiclassical regime with time-dependent potential*, Proc. R. Soc. A, vol. 472, The Royal Society, 2016, p. 20150733.

[3] Roi Baer, *Accurate and efficient evolution of nonlinear Schrödinger equations*, Phys. Rev. A **62** (2000), 063810.

[4] Weizhu Bao, Dieter Jaksch, and Peter A Markowich, *Numerical solution of the Gross–Pitaevskii equation for Bose–Einstein condensation*, Journal of Computational Physics **187** (2003), no. 1, 318–342.

[5] Michael Berman, Ronnie Kosloff, and Hillel Tal-Ezer, *Solution of the time-dependent Liouville-von Neumann equation: Dissipative evolution*, Journal of Physics A: Mathematical and General **25** (1992), no. 5, 1283.

[6] Sergio Blanes, Fernando Casas, JA Oteo, and José Ros, *The Magnus expansion and some of its applications*, Physics Reports **470** (2009), no. 5, 151–238.

[7] JC Butcher, *Runge–Kutta methods*, Numerical Methods for Ordinary Differential Equations, 143–331.

[8] Marco Caliari and Alexander Ostermann, *Implementation of exponential Rosenbrock-type integrators*, Applied Numerical Mathematics **59** (2009), no. 34, 568 – 581, Selected Papers from NUMDIFF-11.

[9] Rongqing Chen and Hua Guo, *Discrete energy representation and generalized propagation of physical systems*, The Journal of chemical physics **108** (1998), no. 15, 6068–6077.

[10] O Chuluunbaatar, VL Derbov, A Galtbayar, AA Gusev, MS Kaschiev, SI Vinitsky, and T Zhanlav, *Explicit Magnus expansions for solving the time-dependent Schrödinger equation*, Journal of Physics A: Mathematical and Theoretical **41** (2008), no. 29, 295203.

[11] C Cohen-Tannoudji, B Diu, and F Laloë, *Quantum Mechanics*, Wiley-Interscience, 2005.

[12] MD Feit, JA Fleck, and A Steiger, *Solution of the Schrödinger equation by a spectral method*, Journal of Computational Physics **47** (1982), no. 3, 412–433.

[13] Richard A. Friesner, Laurette S. Tuckerman, Bright C. Dornblaser, and Thomas V. Russo, *A method for exponential propagation of large systems of stiff nonlinear differential equations*, Journal of Scientific Computing **4** (1989), no. 4, 327–354.

[14] EKU Gross and W Kohn, *Time-dependent density functional theory*, Adv. Quant. Chem **21** (1990), 255–291.

[15] Hua Guo, *Recursive solutions to large eigenproblems in molecular spectroscopy and reaction dynamics*, Rev. Comput. Chem **25** (2007), 285–347.

[16] Hua Guo and Rongqing Chen, *Short-time Chebyshev propagator for the Liouville–von Neumann equation*, The Journal of chemical physics **110** (1999), no. 14, 6626–6634.

[17] M Hochbruck, A Ostermann, and J Schweitzer, *Exponential integrators of Rosenbrock-type*, Oberwolfach Reports **3** (2006), 1107–1110.

[18] Marlis Hochbruck and Christian Lubich, *Exponential integrators for quantum-classical molecular dynamics*, BIT Numerical Mathematics **39** (1999), no. 4, 620–645.

[19] Marlis Hochbruck and Alexander Ostermann, *Explicit exponential Runge–Kutta methods for semilinear parabolic problems*, SIAM Journal on Numerical Analysis **43** (2005), no. 3, 1069–1090.

[20] Youhong Huang, Donald J Kouri, and David K Hoffman, *General, energy-separable Faber polynomial representation of operator functions: Theory and application in quantum scattering*, The Journal of chemical physics **101** (1994), no. 12, 10493–10506.

[21] Wilhelm Huisinga, Lorenzo Pesce, Ronnie Kosloff, and Peter Saalfrank, *Faber and Newton polynomial integrators for open-system density matrix propagation*, The Journal of Chemical Physics **110** (1999), no. 12, 5538–5547.

[22] Christiane P Koch, Mamadou Ndong, and Ronnie Kosloff, *Two-photon coherent control of femtosecond photoassociation*, Faraday Discussions **142** (2009), 389–402.

[23] D Kosloff and R Kosloff, *A Fourier method solution for the time dependent Schrödinger equation as a tool in molecular dynamics*, Journal of Computational Physics **52** (1983), no. 1, 35–53.

[24] Ronnie Kosloff, *Propagation methods for quantum molecular dynamics*, Annual Review of Physical Chemistry **45** (1994), no. 1, 145–178.

[25] Jeffrey L. Krause, Kenneth J. Schafer, and Kenneth C. Kulander, *Calculation of photoemission from atoms subject to intense laser fields*, Phys. Rev. A **45** (1992), 4998–5010.

[26] Kenneth C Kulander, *Time-dependent Hartree-Fock theory of multiphoton ionization: Helium*, Physical Review A **36** (1987), no. 6, 2726.

[27] Claude Leforestier, RH Bisseling, Charly Cerjan, MD Feit, Rich Friesner, A Guldberg, A Hammerich, G Jolicard, W Karrlein, H-D Meyer, et al., *A comparison of different propagation schemes for the time dependent Schrödinger equation*, Journal of Computational Physics **94** (1991), no. 1, 59–80.

[28] Wilhelm Magnus, *On the exponential solution of differential equations for a linear operator*, Communications on pure and applied mathematics **7** (1954), no. 4, 649–673.

[29] H-D Meyer, Uwe Manthe, and Lorenz S Cederbaum, *The multi-configurational time-dependent Hartree approach*, Chemical Physics Letters **165** (1990), no. 1, 73–78.

[30] Borislav V Minchev and Will M Wright, *A review of exponential integrators for first order semi-linear problems*, (2005).

[31] J.G. Muga, J.P. Palao, B. Navarro, and I.L. Egusquiza, *Complex absorbing potentials*, Physics Reports **395** (2004), no. 6, 357 – 426.

[32] Mamadou Ndong, Hillel Tal-Ezer, Ronnie Kosloff, and Christiane P Koch, *A Chebychev propagator for inhomogeneous Schrödinger equations*, The Journal of chemical physics **130** (2009), no. 12, 124108.

[33] ———, *A Chebychev propagator with iterative time ordering for explicitly time-dependent Hamiltonians*, The Journal of Chemical Physics **132** (2010), no. 6, 064105.

[34] Daniel Neuhauser and Michael Baer, *The application of wave packets to reactive atom–diatom systems: a new approach*, The Journal of chemical physics **91** (1989), no. 8, 4651–4657.

[35] José P Palao, Ronnie Kosloff, and Christiane P Koch, *Protecting coherence in optimal control theory: State-dependent constraint approach*, Physical Review A **77** (2008), no. 6, 063412.

[36] J.P. Palao and J.G. Muga, *A simple construction procedure of absorbing potentials*, Chemical Physics Letters **292** (1998), no. 12, 1 – 6.

[37] Tae Jun Park and JC Light, *Unitary quantum time evolution by iterative Lanczos reduction*, The Journal of chemical physics **85** (1986), no. 10, 5870–5876.

[38] Uri Peskin, Ronnie Kosloff, and Nimrod Moiseyev, *The solution of the time dependent Schrödinger equation by the (t, t') method: The use of global polynomial propagators for time dependent hamiltonians*, The Journal of Chemical Physics **100** (1994), no. 12, 8849–8855.

[39] Lothar Reichel, *Newton interpolation at Leja points*, BIT **30** (1990), no. 2, 332–346.

[40] JM Sanz-Serna, *Methods for the numerical solution of the nonlinear Schrödinger equation*, mathematics of computation **43** (1984), no. 167, 21–27.

[41] Ido Schaefer and Ronnie Kosloff, *Optimal-control theory of harmonic generation*, Physical Review A **86** (2012), 063417.

[42] I. Serban, J. Werschnik, and E. K. U. Gross, *Optimal control of time-dependent targets*, Physical Review A **71** (2005), no. 5, 053810.

[43] A. Y. Suhov, *An accurate polynomial approximation of exponential integrators*, Journal of Scientific Computing **60** (2014), no. 3, 684–698.

[44] Zhigang Sun, Weitao Yang, and Dong H Zhang, *Higher-order split operator schemes for solving the Schrödinger equation in the time-dependent wave packet method: applications to triatomic reactive scattering calculations*, Physical Chemistry Chemical Physics **14** (2012), no. 6, 1827–1845.

[45] Hillel Tal-Ezer, *Polynomial approximation of functions of matrices and applications*, Journal of Scientific Computing **4** (1989), no. 1, 25–60.

[46] ———, *High degree polynomial interpolation in Newton form*, SIAM journal on scientific and statistical computing **12** (1991), no. 3, 648–667.

[47] ———, *On restart and error estimation for Krylov approximation of* $w = f(A)v$, SIAM Journal on Scientific Computing **29** (2007), no. 6, 2426–2441.

[48] Hillel Tal-Ezer and R Kosloff, *An accurate and efficient scheme for propagating the time dependent Schrödinger equation*, The Journal of Chemical Physics **81** (1984), no. 9, 3967–3971.

[49] Hillel Tal-Ezer, Ronnie Kosloff, and Charles Cerjan, *Low-order polynomial approximation of propagators for the time-dependent Schrödinger equation*, Journal of Computational Physics **100** (1992), no. 1, 179–187.

[50] Hillel Tal-Ezer, Ronnie Kosloff, and Ido Schaefer, *New, highly accurate propagator for the linear and nonlinear Schrödinger equation*, Journal of Scientific Computing **53** (2012), no. 1, 211–221.

[51] Amrendra Vijay and Horia Metiu, *A polynomial expansion of the quantum propagator, the Green's function, and the spectral density operator*, The Journal of chemical physics **116** (2002), no. 1, 60–68.

[52] J Werschnik and E K U Gross, *Quantum optimal control theory*, Journal of Physics B: Atomic, Molecular and Optical Physics **40** (2007), no. 18, R175.

[53] P. Zhao, H. De Raedt, S. Miyashita, F. Jin, and K. Michielsen, *Dynamics of open quantum spin systems: An assessment of the quantum master equation approach*, Phys. Rev. E **94** (2016), 022126.

[54] Wei Zhu, Youhong Huang, DJ Kouri, Colston Chandler, and David K Hoffman, *Orthogonal polynomial expansion of the spectral density operator and the calculation of bound state energies and eigenfunctions*, Chemical physics letters **217** (1994), no. 1-2, 73–79.