Computer Physics Communications 180 (2009) 1700–1708

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

An arbitrary order diffusion algorithm for solving Schrödinger equations ☆

S.A. Chin [a], S. Janecek [b], E. Krotscheck [b,*]

[a] Department of Physics, Texas A&M University, College Station, TX 77843, USA
[b] Institut für Theoretische Physik, Johannes Kepler Universität Linz, A-4040 Linz, Austria

ARTICLE INFO

Article history:
Received 12 February 2009
Accepted 3 April 2009
Available online 7 April 2009

PACS:
02.70.-c
31.15.-p

Keywords:
Schrödinger equation
Diffusion algorithm
Operator factorizations

ABSTRACT

We describe a simple and rapidly converging code for solving the local Schrödinger equation in one, two, and three dimensions that is particularly suited for parallel computing environments. Our algorithm uses high-order imaginary time propagators to project out the eigenfunctions. A recently developed multi-product, operator splitting method permits, in principle, convergence to any even order of the time step. We review briefly the theory behind the method and discuss strategies for assessing convergence and accuracy. A forward time step, single product fourth-order factorization of the imaginary time evolution operator can also be used.

Our code requires one user defined function which specifies the local external potential. We describe the definition of this function as well as input and output functionalities and convergence criteria. Compared to our previously published code [Computer Physics Communications 178 (2008) 835], the new algorithms can converge at a rate that is only limited by machine precision.

Program summary

Program title: ndsch
Catalogue identifier: AEDR_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEDR_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: Standard CPC licence, http://cpc.cs.qub.ac.uk/licence/licence.html
No. of lines in distributed program, including test data, etc.: 9282
No. of bytes in distributed program, including test data, etc.: 77 824
Distribution format: tar.gz
Programming language: Fortran 90
Computer: Tested on x86, amd64, and Itanium2 architectures. Should run on any architecture providing a Fortran 90 compiler
Operating system: So far tested under UNIX/Linux, Mac OSX and Windows. Any OS with a Fortran 90 compiler available should suffice
RAM: 2 MB to 16 GB, depending on system size
Classification: 6.10
External routines: FFTW3 (http://www.fftw.org/), Lapack (http://www.netlib.org/lapack/)
Nature of problem: Numerical calculation of the lowest few hundred states of 1D, 2D, and 3D local Schrödinger equations in configuration space.
Solution method: Arbitrary even-order multi-product operator splitting, as well as a single product fourth-order factorization, of the imaginary time evolution operator.
Additional comments: Sample input files for the 1D, 2D, and the 3D version as well as a gnuplot script for assessing convergence are included in the distribution file.
Running time: Seconds to hours, depending on system size.

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: eckhard.krotscheck@jku.at (E. Krotscheck).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2009.04.003

1. Introduction

This paper presents an update and significant extension of a recent program package [1] for solving Schrödinger equations for a particle in a local, external potential in one, two, and three dimensions. Even the smaller personal computers of the past decade have enough memory that the storage of several real-space wave functions is no longer a problem. With the advent of multi-core and multi-processor machines on the desktop, users can now utilize advanced hardware for implementing parallel executions of the most computationally intensive parts of a code on a desktop. The method will also be useful for real-space implementations of density functional theory.

Our method is to evolve the system in imaginary time. We have shown in previous work that the use of high-order single product [2–4] and multi-product approximations [5,6] of the evolution operator produces rapidly converging algorithms for solving the Schrödinger equation. This communication presents an easy-to-use and user-friendly implementation of these methods that is also suitable for parallel computing environments. Its purpose is the calculation of low-lying (but can be a few hundreds) bound-states or periodic solutions of the one-, two-, and three-dimensional local Schrödinger equation.

2. Algorithm

2.1. Diffusion algorithm

We consider the problem of solving the one-body Schrödinger equation

$$H(\mathbf{r})\psi_j(\mathbf{r}) = E_j\psi_j(\mathbf{r}) \tag{1}$$

for a *local* potential, *i.e.* for a Hamiltonian of the form

$$H(\mathbf{r}) = T + V(\mathbf{r}) = -\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r}). \tag{2}$$

The eigenfunctions corresponding to the lowest n energy eigenvalues of Eq. (1) can be obtained follows:

(1) Start with a set of trial vectors $\{\psi_j^{(k=0)}(\mathbf{r}), j = 1,\ldots,n\}$.
(2) Apply the *diffusion*, *i.e.*, the imaginary time evolution operator

$$\mathcal{T}(\epsilon) \equiv e^{-\epsilon H} \tag{3}$$

on the set of trial states,

$$\phi_j^{(k+1)}(\mathbf{r}) \equiv \mathcal{T}(\epsilon)\psi_j^{(k)}(\mathbf{r}). \tag{4}$$

(3) Orthonormalize the states $\{\phi_j^{(k+1)}(\mathbf{r})\}$, which produces a new set of trial states $\{\psi_j^{(k+1)}(\mathbf{r})\}$.
(4) Repeat steps (2) and (3) until the set of eigenvalues and corresponding eigenstates have converged.

The propagation step (2) above represents the core of the algorithm and is described in more detail in the following sections. It should be noted that the orthogonalization step (3) can greatly affect the convergence properties of the algorithm. It has been shown [7] that the subspace orthogonalization procedure, which is an application of the Petrov–Galerkin method [8], yields superior convergence compared to simple Gram–Schmidt orthogonalization. Subspace orthogonalization is carried out as described in Ref. [1]: Calculate the overlap matrix

$$M_{ij} = \langle\psi_i^{(k)}\mathcal{T}(\epsilon)|\mathcal{T}(\epsilon)\psi_j^{(k)}\rangle = \langle\phi_i^{(k+1)}|\phi_j^{(k+1)}\rangle \tag{5}$$

and solve the eigenvalue problem

$$\sum_j M_{ij}c_j^{(n)} = m_n c_i^{(n)}. \tag{6}$$

The linear combinations

$$|\psi_j^{(k+1)}\rangle \equiv \frac{1}{\sqrt{m_j}}\sum_i c_i^{(j)}|\phi_i^{(k+1)}\rangle \tag{7}$$

then form an orthonormal set of linear combinations of the propagated wave functions. Moreover, the eigenvalues m_j converge towards

$$m_j^{(k)} \equiv \exp\bigl(-2\epsilon E_j^{(k)}(\epsilon)\bigr) \to m_j = \exp\bigl(-2\epsilon E_j(\epsilon)\bigr). \tag{8}$$

We will refer to the $E_j^{(k)}(\epsilon)$ as the *normalization* energies.

2.2. Operator factorizations

For the general Hamiltonian (2), the evolution operator (3) can only be calculated approximately. A popular approximation is the second-order Störmer/Verlet factorization:

$$\mathcal{T}_2(\epsilon) \equiv e^{-\frac{1}{2}\epsilon V}e^{-\epsilon T}e^{-\frac{1}{2}\epsilon V} = \mathcal{T}(\epsilon) + \mathcal{O}\bigl(\epsilon^3\bigr). \tag{9}$$

With this factorization one has replaced the exact Hamiltonian by an approximate Hamiltonian; the induced error depends on the size of the time step ϵ. When ϵ is small, the induced error is small, but many iterations of the above algorithm are needed for convergence.

When ϵ is large, the iterations converge more quickly, but towards a poorer approximation of the correct eigenvalue/eigenfunction pairs. Thus, the method is, while robust and stable, rather time consuming.

An obvious improvement would be to iterate higher-order algorithms at larger time steps. Unfortunately, Sheng [11] and Suzuki [12] have proved that, beyond second order, no factorization of the product form

$$\mathcal{T}(\epsilon) = \prod_{i=1}^{M} e^{-a_i \epsilon T} e^{-b_i \epsilon V} \tag{10}$$

can have all positive coefficients $\{a_i, b_i\}$. This *forward-time step* requirement is essential for imaginary time propagation, because otherwise, the diffusion operators would be unbounded, resulting in unstable algorithms corresponding to unphysical backwards diffusion in time.

Several fourth-order, *forward-time step* factorizations have been derived by Suzuki [2] and Chin [3] by introducing an additional correction to the potential of the form $[V,[T,V]] = (\hbar^2/m)|\nabla V|^2$. We have studied these methods extensively in previous work [7,9] and implemented it in a generic Schrödinger solver package [1]. In that work, we have opted for the version

$$\mathcal{T}_4(\epsilon) \equiv \mathrm{e}^{-\frac{1}{6}\epsilon V} \mathrm{e}^{-\frac{1}{2}\epsilon T} \mathrm{e}^{-\frac{2}{3}\epsilon \widetilde{V}} \mathrm{e}^{-\frac{1}{2}\epsilon T} \mathrm{e}^{-\frac{1}{6}\epsilon V} = \mathcal{T}(\epsilon) + \mathcal{O}\big(\epsilon^5\big), \tag{11}$$

where

$$\widetilde{V}(\mathbf{r}) = V(\mathbf{r}) + \frac{1}{48}\epsilon^2 \big[V,[T,V]\big](\mathbf{r}) = V(\mathbf{r}) + \frac{\hbar^2 \epsilon^2}{48m} \big|\nabla V(\mathbf{r})\big|^2. \tag{12}$$

The individual factors in Eq. (11) are diagonal in coordinate or momentum space, respectively. The computations are performed by Fourier transforming the wave functions back and forth in successive application of each component of the factorized operator (11).

2.3. Multi-product expansion

If the decomposition of $\mathcal{T}(\epsilon)$ is restricted to a single product as in Eqs. (9) or (11), then there is no practical means of implementing a sixth- or higher-order forward algorithm [10]. However, if this restriction is relaxed to a sum of products,

$$\mathrm{e}^{-\epsilon(T+V)} = \sum_k c_k \prod_i \mathrm{e}^{-a_{k,i}\epsilon T} \mathrm{e}^{-b_{k,i}\epsilon V}, \tag{13}$$

then the requirement that $\{a_{k,i}, b_{k,i}\}$ be positive means that each product can only be second order [11–14]. Since $\mathcal{T}_2(\epsilon)$ is second order with positive coefficients, its powers $\mathcal{T}_2^k(\epsilon/k)$ can form a basis for such a multi-product expansion. Recent work [5] shows that such an expansion is indeed possible and takes the form

$$\mathrm{e}^{-\epsilon(T+V)} = \sum_{k=1}^{n} c_k \mathcal{T}_2^k\left(\frac{\epsilon}{k}\right) + \mathcal{O}\big(\epsilon^{2n+1}\big), \tag{14}$$

where the coefficients c_k are given in closed form for any n:

$$c_i = \prod_{j=1(\neq i)}^{n} \frac{k_i^2}{k_i^2 - k_j^2} \tag{15}$$

with $\{k_1, k_2, \ldots k_n\} = \{1, 2, \ldots n\}$. Since the symmetric factorization $\mathcal{T}_2(\epsilon)$ has only odd powers in ϵ,

$$\mathcal{T}_2(\epsilon) = \exp\big[-\epsilon(T+V) - \epsilon^3 E_3 - \epsilon^5 E_5 - \cdots\big], \tag{16}$$

where E_i are higher-order commutators of T and V, the expansion (14) is just a systematic extrapolation which successively removes each odd-order error. For example,

$$\mathcal{T}_4(\epsilon) = -\frac{1}{3}\mathcal{T}_2(\epsilon) + \frac{4}{3}\mathcal{T}_2^2\left(\frac{\epsilon}{2}\right), \tag{17}$$

$$\mathcal{T}_6(\epsilon) = \frac{1}{24}\mathcal{T}_2(\epsilon) - \frac{16}{15}\mathcal{T}_2^2\left(\frac{\epsilon}{2}\right) + \frac{81}{40}\mathcal{T}_2^3\left(\frac{\epsilon}{3}\right), \tag{18}$$

$$\mathcal{T}_8(\epsilon) = -\frac{1}{360}\mathcal{T}_2(\epsilon) + \frac{16}{45}\mathcal{T}_2^2\left(\frac{\epsilon}{2}\right) - \frac{729}{280}\mathcal{T}_2^3\left(\frac{\epsilon}{3}\right) + \frac{1024}{315}\mathcal{T}_2^4\left(\frac{\epsilon}{4}\right). \tag{19}$$

Since the extrapolation requires subtractions, how high an order one can use depends on ϵ and one's machine precision. For double precision, we have found that it is practical to go up to about the 12th order.

Since each $\mathcal{T}_2(\epsilon)$ requires one complete (forward and backward) FFT, the above series of $2n$-order algorithms only requires $n(n+1)/2$ complete FFTs. Thus algorithms of order 4, 6, 8 and 10 require 3, 6, 10 and 15 complete FFTs.

2.4. Convergence criteria

The iterations normally start with a relatively large time step with a rough guess of an orthonormal set of initial wave functions. In an "inner loop", these states are propagated, for fixed time step, a number of times until convergence. In an "outer loop" the time step is then reduced, and the process is repeated until the time step is small enough such that one is practically in the asymptotic regime. The advantage of high-order approximations is that the time step necessary for good accuracies can be orders of magnitude larger than in the conventional second-order methods.

There are several useful criteria for the termination of both the inner and the outer iteration loop: In the inner loop, where the time step is kept fixed, the iterations (4) converge towards the eigenvalue/eigenvector pairs of the transition operator $\mathcal{T}(\epsilon)$. The simplest estimate is to monitor the convergence of the eigenvalues as a function of iteration number k

$$\Delta E^{(k)}(\epsilon) = \left[\frac{\sum_j (E_j^{(k)}(\epsilon) - E_j^{(k-1)}(\epsilon))^2}{\sum_j (E_j^{(k)}(\epsilon))^2}\right]^{1/2}. \tag{20}$$

The above estimates are, however, sometimes insensitive when eigenvalues cross during the iterations or are almost degenerate. A more sensitive criterion is derived from the *variance* of all states with respect to the evolution operator,

$$R_j^T(\epsilon) = \frac{1}{|\epsilon E_j|}\langle \psi_j^{(k)} | \left[\mathcal{T}_n(\epsilon) - e^{\epsilon E_j}\right]^2 | \psi_j^{(k)}\rangle^{1/2}, \tag{21}$$

as well as the total variance

$$R^T(\epsilon) = \sqrt{\sum_j \left(R_j^T(\epsilon)\right)^2}. \tag{22}$$

The inner loop iterations can be terminated when the variance becomes smaller than an empirically determined value,

$$R_j^T < \epsilon_I. \tag{23}$$

A good value for ϵ_I depends on the system and, of course, on the desired accuracy. We found generally that this value should be of the order of the desired accuracy of the final result; otherwise the iterations with smaller time steps tend to take longer time. Note that the termination condition applies individually to every eigenvalue/eigenfunction pair. Those that satisfy the condition (23) can be "frozen" and only those that do not satisfy the condition are propagated and orthogonalized further.

The above estimates only give information on how well the iterations (4), (7) have converged at a fixed time step. The convergence of the "outer loop", *i.e.* the convergence with respect to the time step, can also be assessed in different ways. A direct measure can be obtained by calculating the *expectation* energy

$$H_j^{(k)}(\epsilon) = \langle \psi_j^{(k)} | H | \psi_j^{(k)}\rangle. \tag{24}$$

The accuracy of the expectation energies is of order ϵ^{4n} for the $2n$th order algorithm. One expects therefore that the expectation energies are, as a function of the time step, more accurate than the normalization energies defined in Eq. (8). The quantity

$$\Delta H^{(k)}(\epsilon) = \left[\frac{\sum_j (E_j^{(k)}(\epsilon) - H_j^{(k)}(\epsilon))^2}{\sum_j (E_j^{(k)}(\epsilon))^2}\right]^{1/2} \tag{25}$$

is therefore a simple and useful estimate how close one is to the solution as a function of the time step. A more sensitive measure is, again, the variance

$$R_j^H(\epsilon) = \frac{1}{|E_j|}\langle \psi_j^{(k)} | [H - E_j]^2 | \psi_j^{(k)}\rangle^{1/2} \tag{26}$$

as well as the total H-variance

$$R^H(\epsilon) = \sqrt{\sum_j \left(R_j^H(\epsilon)\right)^2}. \tag{27}$$

As a termination condition one may use that the *rms*-error of the normalization energies is less than a desired value ϵ_R:

$$R_j^H(\epsilon) < \epsilon_R. \tag{28}$$

3. Parallelization

High-order algorithms have the advantage of converging faster at larger time steps. This advantage is partly compensated by the fact that individual time steps are more costly — recall that one time step of the $2n$-order algorithm requires $n(n+1)/2$ FFTs. We have determined that, without parallelization, the forward 4th order algorithm (11) and the 4th and 6th order multi-product algorithms provide the best cost/performance ratio.

Parallelization of all versions of the algorithm is quite simple and has been implemented with standard OpenMP instructions. The propagation step (4) is parallelized by distributing the individual states over different processors. For the propagation of each state, practically no communication between individual threads is necessary. With increasing size of the individual states and increasing order of the expansion, the parallel speedup of the propagation step quickly approaches the ideal linear limit [6]. At this point, the advantage of high-order multi-product expansions comes to bear: Fewer, but more costly propagation steps are needed with increasing order of the expansion. However, for higher-order algorithms, orthogonalization only takes place after all states have been propagated at a larger time step. Thus, fewer orthogonalization steps are needed for higher-order algorithms and the computational weight is shifted to the propagation step, which can be parallelized with almost linear speedup.

Therefore, applications of the method that require the calculation of many states (as it is the case in density functional theory) will profit most from parallelization. At this time, no efforts have been made to parallelize the orthogonalization step (7) other than the standard parallel implementations provided by Lapack.

- Even if one does not take parallelization into account, it is a big advantage that fewer orthogonalization steps are needed: Orthogonalization is also a bottleneck if one wants to calculate many states (m), because the effort goes with m^2. Thus, higher-order expansions are not only expected to scale better with the number of threads, but also should scale better when increasing the number of states.
- The algorithm is almost ideal for stream computing because the overwhelming parts of the computation are carried out in linear algebra (specifically matrix–matrix products) and FFT routines which are expected to be supplied by the manufacturers of these devices and, thus, ideally suited to the hardware.

4. Program description

4.1. Physical model

The Schrödinger equation is solved in a rectangular box centered around the origin, $-L_i \leqslant x_i < L_i$, $x_i \in \{x, y, z\}$. The physical model is defined by the value of $\hbar^2/2m$ and a user-supplied function that returns the local potential $V(\mathbf{r})$ in units of $\hbar^2/2m$. Since we make heavy use of fast Fourier transforms, we solve in principle the problem subject to periodic boundary conditions. Bound states are calculated by having a sufficiently repulsive potential at the boundaries of the discretization box; it is the user's responsibility to check for unwanted superlattice effects. We have in previous work [6,7] supplied documentation for the convergence and speed of both the fourth-order imaginary time step method as well as the multi-product expansions for local potentials, there is no need to repeat these discussions here.

4.2. Program structure

The program consists of a few Fortran files, which are summarized in Table 1. Practically all of the time consuming operations are performed in the FFTW library routines [15] for fast Fourier transforms and a set of Lapack [16] routines. We have tested the code using the Intel Fortran compiler [17] on SGI Altix systems [18], the free GNU g95 [19], gfortran [23], Sun's f95 Fortran compiler [20] and the NAg Fortran compiler [24] on several Linux PCs and under Mac OSX. Sample makefiles are provided for these compilers. We have successfully tested the code using the Lapack reference implementation on Netlib [16], the "automatically optimized" Atlas library [21], Intel's Math Kernel library [17], and FFTW versions 3.0 and 3.1. We have also installed the code under Windows XP running Cygwin; installation instructions are given in the `README` file.

As far as possible, Fortran program files and include files have been written in a dimension-independent way, *i.e.* they are identical in the 1D, 2D, and 3D version of the code.

4.3. Program usage

All three versions of the program are called by `ndsch prefix` and look for two input files in the current directory. These are named `prefix.mesh` and `prefix.model`; they are described in Tables 2 and 4, respectively. All further discussion refers the 3D code, in the 1D and 2D version all references to the y- and/or z-coordinate are simply omitted as appropriate.

During the execution, the program writes to standard output and to a number of output files. The file `prefix.eval` file contains the converged eigenvalues obtained for each timestep, such that the last line shows the eigenvalues in the desired accuracy. The second file, `prefix.hvar`, contains the variances (25). The contents of both output files are described in Table 5.

Depending on the value of `IMSG` (defined in `prefix.mesh`, see also Table 3), the converged wave functions are written to a binary file on exit. The name of this file can be configured in the input file `prefix.model`. At startup, the program looks for a file of the same structure, also defined in `prefix.model`. If the binary file exists, the contained wave functions are read in as starting values, providing a

Table 1
Fortran source code files of the program.

`ndsch.f90`	Main program. Independent of dimension.
`ndsch.inc`	Definitions of a few scalar global variables. Independent of dimension.
`globals.f90`	Definitions of the vector global variables.
`preset.f90`	Routine to initialize the code at the beginning. Allocates memory.
`readd.f90`	Reads the input files and writes wave functions to a large file for further processing. Independent of dimension.
`inivect.f90`	Before the first iteration, the wave functions have to be initialized with some starting values. Here we are using "particle-in-a-box" wave functions.
`imstep.f90`	The core of the program, implements the imaginary timestep algorithm.
`prpvec.f90`	Various versions of propagation, contains the $\mathcal{T}_2(\epsilon)$ propagator (9) as well as the fourth-order propagator (11).
`ortho.f90`	Orthogonalization of the wave functions using the subspace orthogonalization procedure. Independent of dimension.
`vare.f90`	Routine to calculate the expectation energy and its variance (see Section 2.4).
`vext.f90`	Initializes the external potential and the auxiliary potential $\widetilde{V}(\mathbf{r})$ defined in Eq. (12) from the user-supplied functions `vusr.f90` or `vusr.c`.
`mytimer.f90`	This is a simple timer that should be adapted by the user according to his/her needs. The idea is here that the function `CPU_TIME` is more accurate in linear execution mode, but it will normally return invalid results in parallel mode. For the latter case, `CPU_TIME` is replaced by `SYSTEM_CLOCK`.
`vusr.f90`	Contains the user-defined function for the local potential (Fortran version).
`vusr.c`	Contains the user-defined function for the local potential (C version).

Table 2
Contents of the input file `prefix.mesh`.

`MX`	Mesh points in x-direction. The mesh points are counted from `-MX` to `MX-1`, so there are a total of `2*MX` mesh points in x-direction.
`MY`	Mesh in y-direction is -MY:MY-1.
`MZ`	Mesh in z-direction is -MZ:MZ-1.
	The mesh sizes must be multiples of 2, but for optimum performance they should be a power of 2.
`HR`	Discretization in coordinate space (distance of mesh points). The same value is used in all three directions.
`MAXIM`	Maximum number of imaginary timestep iterations.
`MORB`	Number of states that are propagated. This number may be larger than the number of states calculated to improve convergence.
`ORDER`	Order of the finite-difference approximation of the Laplacian.
`RMUL`	Multiplicator of timestep ramp. 0.5 is a good value.
`ESTP`	Convergence factor of the imaginary timestep iterations. This is the value at which the iterations start.
`ESTE`	Convergence factor of the imaginary timestep iterations. This is the smallest value that is used during the iterations. Setting `ESTE` $= 0$ will decrease the time step until the accuracy limit determined by `EPSR` has been reached.
`IMSG`	A bit field, defining the level at which output messages on the iterations are generated, and controlling some technical aspects of the calculation. The possible values are described in Table 3.
`MANY`	If the second-order algorithm is specified in the bit-field IMSG, then the "any-order" algorithm with order $= 2 \times$ MANY is selected.
`EPSI`	Desired limit for the variance of the overlap energies during the iterations at fixed time step, see Eq. (23).
`EPSR`	Desired accuracy for the variance of the energy as a function of time step, see Eq. (28).

Table 3
Definition of the bit-field `IMSG` contained in `prefix.mesh`.

IMSG	BIT	Function
0	–	No information on iterations.
1	0	Show errors and timings after each converged time step.
2	1	Show errors and timing after each imaginary timestep iteration.
4	2	Show timings.
8	3	Dump eigenvalues and wave functions into a binary data file. Depending on the grid, this may generate a huge file. Meant for testing, documentation, and further processing. The file name is configured in `prefix.model`.
16	4	Use the any order algorithm. The value of `MANY` then determines the order of the algorithm.

Table 4
Contents of the input file `prefix.model`.

`H2M`	Value of $\frac{\hbar}{2m}$. Can be used to set the unit system to atomic Rydberg or Hartree units, but also to use an effective mass.
`NORB`	Number of states that are converged to the desired accuracy.
`RPAR(1:10)`	Array for user-defined double precision parameters that are passed to the user-defined potential function in `vusr.f90`.
`IPAR(1:10)`	Array for user-defined integer parameters that are passed to the user-defined potential function in `vusr.f90`.
`INFILE`	String containing the name of a binary input file from which the program tries to read initial wave functions. If the file is non-existent, or has the wrong structure, the wave functions are initialized by "particle in a box" states.
`OUTFIL`	String containing the name of the binary output file the wave functions are written to.

mechanism to use starting values different than the pre-configured "particle-in-a-box" wave functions in `inivect.f90`. This may speed up the calculation if, for example, several runs with slightly different potentials are carried out. The code that reads the binary file can be found in `readd.f90`. The routine does not rely on global definitions and can also be used in a user-written post-processing code.

4.4. User supplied function

The local potential $V(\mathbf{r})$ must be supplied by the user in the Fortran subroutine `vusr.f90`, which is called by

```
CALL VUSR(V, X, Y, Z, RPAR, IPAR, IER)
```

It is also possible to use a C function instead of Fortran, a C wrapper file is provided in `vusr.c`. The parameters `RPAR(1:10)` and `IPAR(1:10)` defined in the input file `prefix.model` (see Table 4) are passed to the user defined function to allow for easy customization of the potential.

The value of the potential is expected in units of $\hbar^2/2m$.

Table 5
Structure of the output files `prefix.eval` and `prefix.hvar`. For each timestep ϵ, one line is written to `prefix.eval` containing the number of iterations needed for this timestep, the *rms*-errors Eqs. (20), (25) and the converged normalization and expectation energies from Eq. (8) and (24). The file `prefix.hvar` contains, in the first four entries, the same information. Entries 5 to `MORB` + 4 contain the variances residues R_j^H, Eq. (26).

Column	Format	Description
1	`I5`	Number of iterations needed.
2	`F12.4`	Imaginary timestep ϵ.
3	`ES12.3`	*rms*-error ΔE of the normalization energies, Eq. (20).
4	`ES12.3`	*rms*-error ΔH of the normalization energies with respect to the expectation energies, Eq. (25).
5...2*`MORB` + 4	`ES24.15`	Converged normalization energies ($E_i(\epsilon)$, Eq. (8)) and expectation energies ($H_i(\epsilon)$, Eq. (24)) for all states in the format $E_1(\epsilon), H_1(\epsilon), E_2(\epsilon), H_2(\epsilon), \ldots, E_{\mathtt{MORB}}(\epsilon), H_{\mathtt{MORB}}(\epsilon)$. `MORB` is the number of propagated states defined in the file `prefix.mesh`, see Table 2.

Table 6
Calling parameters for the user-defined subroutine `vusr`. The length of the model-specification vectors `RPAR(1:10)` and `IPAR(1:10)` may be changed at compile time.

Variable	Intent	Description
`V`	out	Value of the potential in units of $\hbar^2/2m$.
`X`	in	Value of the x-coordinate.
`Y`	in	Value of the y-coordinate.
`Z`	in	Value of the z-coordinate.
`RPAR(1:10)`	in	User supplied vector of up to 10 real parameters.
`IPAR(1:10)`	in	User supplied vector of up to 10 integer parameters.
`IERR`	out	Error flag.

The subroutines `vusr.f90` and `vusr.c` (see Table 1) contain examples for the simple potentials

$$V(\mathbf{r}) = \frac{\hbar^2}{2m}\left[r_1 x^{n_1} + r_2 y^{n_2} + r_3 z^{n_3}\right], \tag{29}$$

where the r_i and n_i are input, through the file `prefix.model`, in the vectors `RPAR` and `IPAR`.

In the 3D case, we also supply, in file `c60.f90` a simple model of a C_{60} molecule, where the carbon atoms are modeled by troughs of the form

$$V(\mathbf{r}) = -\sum_i \frac{V_0}{\cosh^2(|\mathbf{r} - \alpha\mathbf{R}_i|)}, \tag{30}$$

where the $\mathbf{R}_i$ are the positions of the carbon atoms which can be scaled by the parameter $\alpha =$ `RPAR(1)`. The strength is $V_0 =$ `RPAR(2)`.

4.5. Typical use

The program starts with an initial time step ϵ, defined by the input parameter `ESTP`. For this fixed time step, the transport equation is iterated until all states have converged according to Eq. (23), where ϵ_I is given by the parameter `EPSI` in the `prefix.mesh` file. A maximum of `MAXIM` iterations are performed if convergence cannot be reached. The time step is then reduced by a factor `RMUL` and the process is repeated until all states have converged according to Eq. (28), where ϵ_R is given by the parameter `EPSR`, or until the time step is less than or equal to `ESTE`. A value `RMUL` $= 0.5$ is a good choice for production runs. If it takes many iterations per time step, it might be that the highest calculated state is almost degenerate with a state that has been omitted, in that case it is advisable to increase the number `MORB`.

The user should, in the first trial runs, check the expected ϵ^{2n} convergence for the normalization energies, and the ϵ^{4n} convergence of the expectation energies. For the forward fourth-order algorithm (11), deviation from this behavior for small ϵ may indicate a too coarse a discretization for computing the potential gradient. There is no such problem for the "any-order" implementations. In these cases, if the second-order algorithm shows the correct power behavior, then higher-order algorithms can only deviate from the expected power laws at small time steps due to rounding errors. To determine the correct power law, a test run with a ramp-multiplicator `RMUL` ≈ 0.9 may be useful. There is no rigorous statement at what value of the time step the asymptotic power law should dominate, this depends on the physical model. A good estimate for the absolute accuracy of the eigenvalues is the *rms*-error $\Delta H(\epsilon)$.

5. Summary

We have presented in this paper a real-space program for solving the local, two- and three-dimensional Schrödinger equation using approximations of the evolution operator (3) of arbitrary order in the time step. We have demonstrated in previous work [6] that high-order factorizations of local Hamiltonians improve the convergence and are superior to second-order factorizations by one to two orders of magnitude, depending on the accuracy wanted. They also outperform standard eigenvalue solvers [22] by almost an order of magnitude and are easily parallelized.

Acknowledgements

This work was supported by the Austrian Science Fund FWF under grant No. P18134. We would like to thank M. Aichinger and E. Hernández for useful discussions.

Appendix A. Test run output

The following table shows a typical test output for the included example of a harmonic oscillator in three dimensions. This test run has been carried out on an AMD Opteron system, using the f95 compiler. After a few lines describing essential parameters of the run, each iteration generates one line of output, describing

- The number of the iteration.
- The lowest state that has been included in this iteration. When low-lying states have converged to the desired accuracy, the code will "freeze" these states to save time.
- The time step for this iteration.
- Propagation time.
- Orthogonalization time.
- Error $\Delta E(\epsilon)$.
- Variance of the transition operator, $R^T(\epsilon)$.

This output can be turned off by unsetting bit 2 in the variable `IMSG`.

After the "inner loop" iterations have converged to the desired accuracy, the following output is generated:

- The number of iterations needed for convergence at this time step.
- The lowest state that has been included in all iterations. When low-lying states have converged to the desired accuracy, the code will "freeze" these states to save time.
- The time step for this iteration.
- Propagation time.
- Orthogonalization time.
- Error $\Delta E(\epsilon)$.
- Variance of the transition operator, $R^T(\epsilon)$.
- Error $\Delta H(\epsilon)$.
- Variance of the Hamiltonian, $R^H(\epsilon)$.

This output can be turned off by unsetting bit 1 in the variable `IMSG`.

After the time step has been reduced to the desired value, or the variances of all eigenvalues are less then the limit ϵ_R, the code prints the resulting eigenvalues. Convergence is tested only for the lowest `NORB` states but, to potentially improve convergence of the subspace orthogonalization, a larger number `MORB` can be propagated. The sample run is for `NORB=4` and `MORB=8`, these are separated in the output by a vertical bar.

```
CMDLINE> ./ndsch hosc
 Mesh   file = hosc.mesh
 \xch{Running}{Runnig} "any" order algorithm with N =  8
 Output file = osc.dat
 MX =  32 MY =  32 MZ =  32 HR =   0.25
 ITIM =  99    4 STATES
 Cannot open binary input data file osc.dax
 Re-initializing...

 ==== 3D Schroedinger equation solver  ==== Date: 27-01-2009 ==== Time: 13:12:38 ====

   7   5    2.000    1.264    0.124   8.294E-16   2.247E-13   8.033E-05   4.821E-06
   9   5    1.000    1.216    0.112   2.374E-16   2.211E-13   5.951E-07   1.821E-07
  12   5    0.500    1.072    0.112   3.996E-16   7.144E-13   2.305E-09   1.887E-09
   8   5    0.250    1.196    0.120   3.996E-16   9.553E-13   1.024E-11   1.167E-11
   7   5    0.125    1.112    0.112   8.180E-16   9.015E-13   1.072E-12   1.601E-14
   1   5    0.063    1.240    0.104   5.616E-14   8.593E-13   9.275E-13   6.729E-14
   1   5    0.031    1.204    0.108   3.945E-15   8.294E-13   8.640E-13   5.845E-14
   1   5    0.016    1.244    0.104   2.064E-14   8.207E-13   8.352E-13   3.932E-14

 Timing:      T_tot =    62.81  T_pro =    55.23  T_ort =     5.10

 Converged | propagated states: ESTP =  0.0156
 N =   4 E_n(1: 10):  1.5000    2.5000    2.5000    2.5000 |  3.5000    3.5000
                      3.5000    3.5000    3.5000    3.5000


CMDLINE>
```

References

[1] S. Janecek, E. Krotscheck, A fast and simple program for solving local Schrödinger equations in two and three dimensions, Comput. Phys. Comm. 178 (11) (2008) 835–842.
[2] M. Suzuki, New scheme of hybrid exponential product formulas with applications to quantum Monte Carlo simulations, in: D.P. Landau, K.K. Mon, H.-B. Schüttler (Eds.), Computer Simulation Studies in Condensed Matter Physics, vol. 8, Springer, Berlin, 1996, pp. 1–6.

[3] S.A. Chin, Symplectic integrators from composite operator factorizations, Phys. Lett. A 226 (1997) 344–348.
[4] S.A. Chin, C.R. Chen, Fourth order gradient symplectic integrator methods for solving the time-dependent Schrödinger equation, J. Chem. Phys. 114 (2001) 7338.
[5] S.A. Chin, Multi-product splitting and Runge–Kutta–Nyström integrators, arXiv:0809.0914, 2008.
[6] S.A. Chin, S. Janecek, E. Krotscheck, Any order imaginary time propagation method for solving the Schrödinger equation, arXiv:0809.3739, 2008.
[7] M. Aichinger, E. Krotscheck, A fast configuration space method for solving local Kohn–Sham equations, Comput. Materials Sci. 34 (2) (2005) 188–212.
[8] T. Torsti, et al., Physica Status Solidi B 243 (2006) 1016.
[9] E.R. Hernández, S. Janecek, M.S. Kaczmarski, E. Krotscheck, An evolution-operator method for density functional theory, Phys. Rev. B 75 (2007) 075108.
[10] S.A. Chin, Structure of positive decompositions of exponential operators, Phys. Rev. E 71 (2005) 016703.
[11] Q. Sheng, Solving linear partial differential equations by exponential splitting, IMA J. Numer. Anal. 9 (1989) 199–212.
[12] M. Suzuki, General theory of fractal path-integrals with applications to many-body theories and statistical physics, J. Math. Phys. 32 (1991) 400.
[13] D. Goldman, T.J. Kaper, *N*th-order operator splitting schemes and nonreversible systems, SIAM J. Numer. Anal. 33 (1996) 349.
[14] S.A. Chin, A fundamental theorem on the structure of symplectic integrators, Phys. Lett. A 354 (2006) 373.
[15] www.fftw.org.
[16] www.netlib.org/lapack.
[17] www.intel.com.
[18] Altix is a registered trademark of Silicon Graphcs, Inc.
[19] www.g95.org.
[20] www.sun.com.
[21] www.atlas.org.
[22] http://www.caam.rice.edu/software/ARPACK/.
[23] gcc.gnu.org/fortran/.
[24] www.nag.com.