Commun. Comput. Phys.
doi: 10.4208/cicp.131210.180411a

Vol. **11**, No. 3, pp. 739-755
March 2012

SPEEDUP Code for Calculation of Transition Amplitudes via the Effective Action Approach

Antun Balaž*, Ivana Vidanović, Danica Stojiljković,
Dušan Vudragović, Aleksandar Belić and Aleksandar Bogojević

*Scientific Computing Laboratory, Institute of Physics Belgrade,
University of Belgrade, Pregrevica 118, 11080 Belgrade, Serbia.*

Received 13 December 2010; Accepted 18 April 2011

Communicated by Leonardo Golubovic

Available online 28 October 2011

Abstract. We present Path Integral Monte Carlo C code for calculation of quantum mechanical transition amplitudes for 1D models. The SPEEDUP C code is based on the use of higher-order short-time effective actions and implemented to the maximal order p=18 in the time of propagation (Monte Carlo time step), which substantially improves the convergence of discretized amplitudes to their exact continuum values. Symbolic derivation of higher-order effective actions is implemented in SPEEDUP Mathematica codes, using the recursive Schrödinger equation approach. In addition to the general 1D quantum theory, developed Mathematica codes are capable of calculating effective actions for specific models, for general 2D and 3D potentials, as well as for a general many-body theory in arbitrary number of spatial dimensions.

AMS subject classifications: 81Q05, 81Q15, 65Y04, 65B99

Key words: Transition amplitude, effective action, path integrals, Monte Carlo.

1 Introduction

Exact solution of a given many-body model in quantum mechanics is usually expressed in terms of eigenvalues and eigenfunction of its Hamiltonian

$$\hat{H}=\sum_{i=1}^{M}\frac{\hat{\mathbf{p}}_i^2}{2m_i}+\hat{V}(\hat{\mathbf{q}}_1,\cdots,\hat{\mathbf{q}}_M), \tag{1.1}$$

but it can be also expressed through analytic solution for general transition amplitude $A(\mathbf{a},\mathbf{b};T)=\langle\mathbf{b}|e^{-iT\hat{H}/\hbar}|\mathbf{a}\rangle$ from the initial state $|\mathbf{a}\rangle$ to the final state $|\mathbf{b}\rangle$ during the time of

*Corresponding author. *Email addresses:* antun@ipb.ac.rs (A. Balaž), ivanavi@ipb.ac.rs (I. Vidanović), danica@ipb.ac.rs (D. Stojiljković), dusan@ipb.ac.rs (D. Vudragović), abelic@ipb.ac.rs (A. Belić), alex@ipb.ac.rs (A. Bogojević)

propagation T. Calculation of transition amplitudes is more suitable if one uses path integral formalism [1–3], but in principle, if eigenproblem of the Hamiltonian can be solved, one should be able to calculate general transition amplitudes, and vice versa. However, mathematical difficulties may prevent this, and even more importantly, exact solutions can be found only in a very limited number of cases. Therefore, use of various analytic approximation techniques or numerical treatment is necessary for detailed understanding of the behavior of almost all models of interest.

In numerical approaches it could be demanding and involved to translate numerical knowledge of transition amplitudes to (or from) eigenstates, but practically can be always achieved. It has been implemented in various setups, e.g. through extraction of the energy spectra from the partition function [2–5], and using the diagonalization of space-discretized matrix of the evolution operator, i.e. matrix of transition amplitudes [6–10]. All these applications use the imaginary-time formalism [11, 12], typical for numerical simulations of such systems.

Recently introduced effective action approach [13–17] provides an ideal framework for exact numerical calculation of quantum mechanical amplitudes. It gives systematic short-time expansion of amplitudes for a general potential, thus allowing accurate calculation of short-time properties of quantum systems directly, as has been demonstrated in [8–10]. For numerical calculations that require long times of propagation to be considered using e.g. Monte Carlo method, effective action approach provides improved discretized actions leading to the speedup in the convergence of numerically calculated discretized quantities to their exact continuum values. This has been also demonstrated in Monte Carlo calculations of energy expectation values using the improved energy estimators [5, 18].

In this paper we present SPEEDUP codes [19] which implement the effective action approach, and which were used for numerical simulations in [4, 5, 8–10, 13–17]. The paper is organized as follows. In Section 2 we briefly review the recursive approach for analytic derivation of higher-order effective actions. SPEEDUP Mathematica codes capable of symbolic derivation of effective actions for a general one- and many-body theory as well as for specific models is described in detail in Section 3, while in Section 4 we describe SPEEDUP Path Integral Monte Carlo C code, developed for numerical calculation of transition amplitudes for 1D models. Section 5 summarizes presented results and gives outlook for further development of the code.

2 Theoretical background

From inception of the path integral formalism, expansion of short-time amplitudes in the time of propagation was used for the definition of path integrals through the time-discretization procedure [2, 3]. This is also straightforwardly implemented in the Path Integral Monte Carlo approaches [20], where one usually relies on the naive discretization of the action. Several improved discretized actions, mainly based on the Trotter formula

and its generalizations, were developed and used in the past [21–23]. A recent analysis of this method can be found in Jang et al [24]. Several related investigations dealing with the speed of convergence have focused on improvements in short-time propagation [25, 26] or the action [27]. More recently, split-operator method has also been developed [28–32], later extended to include higher-order terms [33–36], and systematically improved using the multi-product expansion [37–39].

The effective action approach is based on the ideal discretization concept [16]. It was introduced first for single-particle 1D models [13–15] and later extended to general many-body systems in arbitrary number of spatial dimensions [5, 17]. This approach allows systematic derivation of higher-order terms to a chosen order p in the short time of propagation.

Recursive method for deriving discretized effective actions, established in [17], is based on solving the underlying Schrödinger equation for the amplitude. It has proven to be the most efficient tool for treatment of higher-order expansion. In this section we give brief overview of the recursive method, which will be implemented in Mathematica in the next section. We start with the case of single particle in 1D, used in the SPEEDUP C code. Throughout the paper we will use natural system of units, in which $\hbar$ and all masses are set to unity.

2.1 One particle in one dimension

In the effective action approach, transition amplitudes are expressed in terms of the ideal discretized action S^* in the form

$$A(a,b;T) = \frac{1}{\sqrt{2\pi T}} e^{-S^*(a,b;T)}, \tag{2.1}$$

which can be also seen as a definition of the ideal action [16]. Therefore, by definition, the above expression is correct not only for short times of propagation, but for arbitrary large times T. We also introduce the ideal effective potential W,

$$S^*(a,b;T) = T\left[\frac{1}{2}\left(\frac{b-a}{T}\right)^2 + W\right], \tag{2.2}$$

reminiscent of the naive discretized action, with the arguments of the effective potential $(a,\ b,\ T)$ usually written as $W\left(\frac{a+b}{2}, \frac{b-a}{2}; T\right)$, to emphasize that we will be using mid-point prescription.

However, ideal effective action and effective potential can be calculated analytically only for exactly solvable models, while in all other cases we have to use some approximative method. We use expansion in the time of propagation, assuming that the time T is small. If this is not the case, we can always divide the propagation into N time steps, so that $\varepsilon = T/N$ is small. Long-time amplitude is than obtained by integrating over all short-time ones,

$$A(a,b;T) = \int dq_1 \cdots dq_{N-1}\ A(a,q_1;\varepsilon)\, A(q_1,q_2;\varepsilon) \cdots A(q_{N-1},b;\varepsilon), \tag{2.3}$$

paving the way towards Path Integral Monte Carlo calculation, which is actually implemented in the SPEEDUP C code.

If we consider general amplitude $A(q,q';\varepsilon)$, introduce the mid-point coordinate $x=(q+q')/2$ and deviation $\bar{x}=(q'-q)/2$, and express A using the effective potential,

$$A(q,q';\varepsilon)=\frac{1}{\sqrt{2\pi\varepsilon}}e^{-\frac{2}{\varepsilon}\bar{x}^2-\varepsilon W(x,\bar{x};\varepsilon)}, \tag{2.4}$$

the time-dependent Schrödinger equation for the amplitude leads to the following equation for W

$$W+\bar{x}\bar{\partial}W+\varepsilon\partial_\varepsilon W-\frac{1}{8}\varepsilon\partial^2 W-\frac{1}{8}\varepsilon\bar{\partial}^2 W+\frac{1}{8}\varepsilon^2(\partial W)^2+\frac{1}{8}\varepsilon^2(\bar{\partial}W)^2=\frac{1}{2}(V_++V_-), \tag{2.5}$$

where $V_\pm=V(x\pm\bar{x})$, i.e. $V_-=V(q)$, $V_+=V(q')$. The short-time expansion assumes that we expand W to power series in ε to a given order, and calculate the appropriate coefficients using Eq. (2.5). We could further expect that this results in coefficients depending on the potential $V(x)$ and its higher derivatives. However, this scheme is not complete, since the effective potential depends not only on the mid-point x, but also on the deviation $\bar{x}$, and the obtained equations for the coefficients cannot be solved in a closed form. In order to resolve this in a systematic way, we make use of the fact that, for short time of propagation, deviation $\bar{x}$ is on the average given by the diffusion relation $\bar{x}^2\propto\varepsilon$, allowing double expansion of W in the form

$$W(x,\bar{x};\varepsilon)=\sum_{m=0}^{\infty}\sum_{k=0}^{m}c_{m,k}(x)\varepsilon^{m-k}\bar{x}^{2k}. \tag{2.6}$$

Restricting the above sum over m to $p-1$ leads to level p effective potential $W_p(x,\bar{x};\varepsilon)$ which gives expansion of the effective action S_p^* to order ε^p, and hence the level designation p for both the effective action and the corresponding potential W_p. Thus, if the diffusion relation is applicable (which is always the case in Monte Carlo calculations), instead of the general double expansion in $\bar{x}$ and ε, we are able to obtain simpler, systematic expansion in ε only.

As shown previously [13–15], when used in Path Integral Monte Carlo simulations for calculation of long time amplitudes according to Eq. (2.3), use of level p effective action leads to the convergence of discretized amplitudes proportional to ε^p, i.e. as $1/N^p$, where N is the number of time steps used in the discretization.

If we insert the above level p expansion of the effective potential to Eq. (2.5), we obtain the recursion relation derived in [17],

$$\begin{aligned}8(m+k+1)c_{m,k}=&(2k+2)(2k+1)c_{m,k+1}+c''_{m-1,k}-\sum_{l=0}^{m-2}\sum_r c'_{l,r}c'_{m-l-2,k-r}\\&-\sum_{l=1}^{m-2}\sum_r 2r(2k-2r+2)c_{l,r}c_{m-l-1,k-r+1},\end{aligned} \tag{2.7}$$

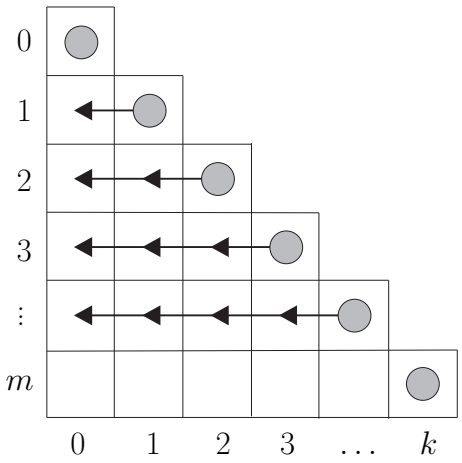

Figure 1: Order in which the coefficients $c_{m,k}$ are calculated: diagonal ones from Eq. (2.8), off-diagonal from recursion (2.7).

where the sum over r goes from $\max\{0,\ k-m+l+2\}$ to $\min\{k,\ l\}$. This recursion can be used to calculate all coefficients $c_{m,k}$ to a given level p, starting from the known initial condition, $c_{0,0}=V$. The diagonal coefficients can be calculated immediately,

$$c_{m,m}=\frac{V^{(2m)}}{(2m+1)!}, \tag{2.8}$$

and for a given value of $m=0,\cdots p-1$, the coefficients $c_{m,k}$ follow recursively from evaluating (2.7) for $k=m-1,\cdots,1,0$, as illustrated in Fig. 1.

2.2 Extension to many-body systems

The above outlined approach can be straightforwardly applied to many-body systems. Again the amplitude is expressed through the effective action and the corresponding effective potential, which now depends on mid-point positions and deviations of all particles. For simplicity, these vectors are usually combined into $D\times M$ dimensional vectors $\mathbf{x}$ and $\bar{\mathbf{x}}$, where D is spatial dimensionality, and M is the number of particles. In this notation,

$$A(\mathbf{q},\mathbf{q}';\varepsilon)=\frac{1}{(2\pi\varepsilon)^{DM/2}}e^{-\frac{2}{\varepsilon}\bar{\mathbf{x}}^2-\varepsilon W(\mathbf{x},\bar{\mathbf{x}};\varepsilon)}, \tag{2.9}$$

where initial and final position $\mathbf{q}=(\mathbf{q}_1,\cdots,\mathbf{q}_M)$ and $\mathbf{q}'=(\mathbf{q}'_1,\cdots,\mathbf{q}'_M)$ are analogously defined $D\times M$ dimensional vectors. Here we will not consider quantum statistics of particles. The required symmetrization or antisymmetrization must be applied after transition amplitudes are calculated using the effective potential.

Many-body transition amplitudes satisfy $D\times M$-dimensional generalization of the time-dependent Schrödinger equation, which leads to the equation for the effective po-

tential similar to Eq. (2.5), with vectors replacing previously scalar quantities,

$$W+\bar{\mathbf{x}}\cdot\bar{\boldsymbol{\partial}}W+\varepsilon\partial_\varepsilon W-\frac{1}{8}\varepsilon\boldsymbol{\partial}^2W-\frac{1}{8}\varepsilon\bar{\boldsymbol{\partial}}^2W+\frac{1}{8}\varepsilon^2(\boldsymbol{\partial}W)^2+\frac{1}{8}\varepsilon^2(\bar{\boldsymbol{\partial}}W)^2=\frac{1}{2}(V_++V_-). \tag{2.10}$$

The effective potential for short-time amplitudes again can be written in the form of the double expansion in ε and $\bar{\mathbf{x}}$. However, it turns out to be advantageous to use the expansion

$$W(\mathbf{x},\bar{\mathbf{x}};\varepsilon)=\sum_{m=0}^{\infty}\sum_{k=0}^{m}\varepsilon^{m-k}W_{m,k}(\mathbf{x},\bar{\mathbf{x}})\ , \tag{2.11}$$

and work with fully contracted quantities $W_{m,k}$

$$W_{m,k}(\mathbf{x},\bar{\mathbf{x}})=\bar{x}_{i_1}\bar{x}_{i_2}\cdots\bar{x}_{i_{2k}}c_{m,k}^{i_1,\cdots,i_{2k}}(\mathbf{x}), \tag{2.12}$$

rather than with the respective coefficients $c_{m,k}^{i_1,\cdots,i_{2k}}$. In this way we avoid the computationally expensive symmetrization over all indices $i_1,\cdots,i_{2k}$. After inserting the above expansion into the equation for the effective potential, we obtain the recursion relation which represents a generalization of previously derived Eq. (2.7) for 1D case, and has the form

$$\begin{aligned}8(m+k+1)W_{m,k}=&\boldsymbol{\partial}^2W_{m-1,k}+\bar{\boldsymbol{\partial}}^2W_{m,k+1}-\sum_{l=0}^{m-2}\sum_r(\boldsymbol{\partial}W_{l,r})\cdot(\boldsymbol{\partial}W_{m-l-2,k-r})\\&-\sum_{l=1}^{m-2}\sum_r(\bar{\boldsymbol{\partial}}W_{l,r})\cdot(\bar{\boldsymbol{\partial}}W_{m-l-1,k-r+1}).\end{aligned} \tag{2.13}$$

The sum over r runs from $\max\{0,\ k-m+l+2\}$ to $\min\{k,\ l\}$, while diagonal quantities $W_{m,m}$ can be calculated directly,

$$W_{m,m}=\frac{1}{(2m+1)!}(\bar{\mathbf{x}}\cdot\boldsymbol{\partial})^{2m}V. \tag{2.14}$$

The above recursion disentangles, in complete analogy with the previously outlined case of one particle in 1D, and is solved in the order shown in Fig. 1.

3 SPEEDUP Mathematica codes for deriving the higher-order effective actions

The effective action approach can be used for numerically exact calculation of short-time amplitudes if the effective potential W_p can be analytically derived to sufficiently high values of p such that the associated error is smaller than the required numerical precision. The error ε^p for the effective action, obtained when level p effective potential is used, translates into $\varepsilon^{p-DM/2}$ for a general many-body short-time amplitude. However, when amplitudes are calculated using the Path Integral Monte Carlo SPEEDUP C

code [19], which will be presented in the next section, the errors of numerically calculated amplitudes are always proportional to $\varepsilon^p \sim 1/N^p$, where N is number of time-steps in the discretization of the propagation time T.

Therefore, accessibility of higher-order effective actions is central to the application of this approach if it is used for direct calculation of short-time amplitudes [8–10], as well as in the case when PIMC code is used [4, 5, 18]. However, increase in the level p leads to the increase in complexity of analytic expressions for the effective potential. On one hand, this limits the maximal accessible level p by the amount of memory required for symbolic derivation of the effective potential. On the other hand, practical use of large expressions for W_p may slow down numerical calculations, and one can opt to use lower than the maximal available level p when optimizing total CPU time required for numerical simulation. The suggested approach is to study time-complexity of the algorithm in practical applications, and to choose optimal level p by minimizing the execution time required to achieve fixed numerical precision.

We have implemented efficient symbolic derivation of higher-order effective actions in Mathematica using the recursive approach. All source files described in this section are located in the `Mathematica` directory of the SPEEDUP code distribution.

3.1 General 1D Mathematica code

SPEEDUP code [19] for symbolic derivation of the effective potential to specified level p is implemented in Mathematica [40], and is available in the `EffectiveAction-1D.nb` notebook. It implements the algorithm depicted in Fig. 1 and calculates the coefficients $c_{m,k}$ for $m=0,\cdots,p-1$ and $k=m,\cdots,0$, starting from the initial condition $c_{0,0}=V$. For a given value of m, the diagonal coefficient $c_{m,m}$ is first calculated from Eq. (2.8), and then all off-diagonal coefficients are calculated from the recursion (2.7).

In this code the potential $V(x)$ is not specified, and the effective potential is derived for a general one-particle 1D theory. The resulting coefficients $c_{m,k}$ and the effective potential are expressed in terms of the potential V and its higher derivatives. Level p effective potential, constructed as

$$W_p(x,\bar{x};\varepsilon)=\sum_{m=0}^{p-1}\sum_{k=0}^{m} c_{m,k}(x)\,\varepsilon^{m-k}\bar{x}^{2k}, \tag{3.1}$$

contains derivatives of V to order $2p-2$.

The only input parameter of this Mathematica code is the level p to which the effective potential should be calculated. As the code runs, it prints used amount of memory (in MB) and CPU time. This information can be used to estimate the required computing resources for higher values of p. The calculated coefficients can be exported to a file, and later imported for further numerical calculations. As an illustration, the file `EffectiveAction-1D-export-p5.m` contains exported definition of all the coefficients $c_{m,k}$ calculated at level $p=5$, while the notebook `EffectiveAction-1D-matching-p5.nb`

contains matching output from the interactive session used to produce the above $p=5$ result.

The execution of this code on a typical 2 GHz CPU for level $p=10$ requires 10-15 MB of RAM and several seconds of CPU time. We have successfully run this code for levels as high as $p=35$ [19]. SPEEDUP C code implements effective actions to the maximal level $p=18$, with the size of the corresponding C function around 2 MB. If needed, higher levels p can be easily implemented in C and added to the existing SPEEDUP code.

3.2 General 2D and 3D Mathematica code

Although we have developed Mathematica code capable of deriving effective actions for a general many-body theory in arbitrary number of spatial dimensions, in practical applications in 2D and 3D it can be very advantageous to use simpler codes, able to produce results to higher levels p than the general code [9,10].

This is done in files `EffectiveAction-2D.nb` and `EffectiveAction-3D.nb`, where the recursive approach is implemented directly in 2D and 3D. Execution of these codes requires more memory: for $p=10$ effective action one needs 60 MB in 2D case, while in 3D case the needed amount of memory increases to 860 MB. On the other hand, the execution time is several minutes for 2D code and around 30 minutes for 3D code.

The distribution of the SPEEDUP code contains exported $p=5$ definitions of contractions $W_{m,k}$ for both 2D and 3D general potential, as well as matching outputs from interactive sessions used to generate these results.

3.3 Model-specific Mathematica codes

When general expressions for the effective actions, obtained using the above described SPEEDUP Mathematica codes, are used in numerical simulations, one has to specify the potential V and its higher derivatives to order $2p-2$ in order to be able to calculate transition amplitudes. Such approach is justified for systems where the complexity of higher derivatives increases. However, for systems where this is not the case, or where only a limited number of derivatives is non-trivial (e.g. polynomial interactions), it might be substantially beneficial to specify the potential at the beginning of the Mathematica code and calculate the derivatives explicitly when iterating the recursion.

Using this approach, one is able to obtain coefficients $c_{m,k}$ and the effective potential W directly as functions of the mid-point x. This is implemented in the notebooks `EffectiveAction-1D-AHO.nb` and `EffectiveAction-2D-AHO.nb` for the case of anharmonic oscillators in 1D and 2D,

$$V_{1D-AHO}(x)=\frac{A}{2}x^2+\frac{g}{24}x^4, \tag{3.2}$$

$$V_{2D-AHO}(x)=\frac{A}{2}(x^2+y^2)+\frac{g}{24}(x^2+y^2)^2. \tag{3.3}$$

These codes can be easily executed within few seconds and with the minimal amounts of memory even for $p=20$. For 1D anharmonic oscillator we have successfully calculated effective actions to excessively large value $p=144$, and in 2D to $p=67$ [19], to illustrate the advantage of this model-specific method.

Similar approach can be also used in another extreme case, when the complexity of higher derivatives of the potential V increases very fast, so that entering the corresponding expressions to the code becomes impractical. Even in this situation expressions for effective actions can be usually simplified using some appropriate model-specific ansatz. The form of such ansatz can be deduced from the form of model-specific effective potentials, and then used to simplify their derivation. Such use-case is illustrated in the SPEEDUP Mathematica code for the modified Pöscl-Teller potential,

$$V_{1D-MPT}(x)=-\frac{\lambda}{(\cosh\alpha x)^2}. \tag{3.4}$$

For this potential, the coefficients $c_{m,k}$ of the effective potential can be expressed in the form

$$c_{m,k}(x)=\sum_{l=0}^{m} d_{m,k,l}\,\frac{(\tanh\alpha x)^{2l}}{(\cosh\alpha x)^{2m-2l+2}}, \tag{3.5}$$

and newly introduced constant coefficients $d_{m,k,l}$ can be calculated using the model-specific recursion in `EffectiveAction-1D-MPT.nb`. The form of the ansatz (3.5) is deduced from the results of executing general 1D Mathematica code, with the model-specific potential (3.4) defined before the recursion calculation of the coefficients is performed. Using this approach, we were able to obtain maximal level $p=41$ effective action [19].

3.4 General many-body Mathematica code

SPEEDUP Mathematica code for calculation of effective action for a general many-body theory is implemented using the MathTensor [41] package for tensorial calculations in Mathematica. This general implementation required some new functions related to the tensor calculus to be defined in the source notebook `EffectiveAction-ManyBody.nb` provided with the SPEEDUP code.

The function `GenNewInd[n]` generates the required number `n` of upper and lower indices using the MathTensor function `UpLo`, with the assigned names `up1`, `lo1`,$\cdots$, as well as lists `upi` and `loi`, each containing `n` strings corresponding to the names of generated indices. These new indices are used in the implementation of the recursion for calculation of derivatives of $W_{m,k}$, contractions of the effective potential, and for this reason had to be explicitly named and properly introduced.

The expressions obtained by iterating the recursion contain large numbers of contractions, and function `NewDefUnique[contr]` replaces all contracted indices with the newly-introduced dummy ones in the contraction `contr`, so that they do not interfere with the calculation of derivatives in the recursion. This is necessary since the derivatives in recursion do not distinguish contracted indices from non-contracted ones if their names

happen to be generated by the function `GenNewInd`. Note that the expression `contr` does not have to be full contraction, i.e. function `NewDefUnique` will successfully act on tensors of any kind if they have contracted indices, while it will leave them unchanged if no contractions are present.

The function `NewDerivativeVec[contr, vec, ind]` implements calculation of the first derivative of the tensor `contr` (which may or may not contain contracted indices, but if it does, they are supposed to be uniquely defined dummy ones, which is achieved using the function `NewDefUnique`). The derivative is calculated with respect to vector `vec` with the vectorial index `ind`. The index `ind` can be either lower or upper one, and has to be generated previously by the function `GenNewInd`.

Finally, the function `NewLaplacianVec[contr, vec]` implements the Laplacian of the tensor `contr` with respect to the vector `vec`, i.e. it performs the calculation of contractions of the type

$$\frac{\partial}{\partial \mathtt{vec}_i}\frac{\partial}{\partial \mathtt{vec}^i}\mathtt{contr}. \tag{3.6}$$

After all described functions are defined, the execution of the code proceeds by setting the desired level of the effective action `p`, generating the needed number of named indices using the function call `GenNewInd[2 p + 2]`, and then by performing the recursion according to the scheme illustrated in Fig. 1. The use of MathTensor function `CanAll` in the recursion ensures that the obtained expressions for `W[m, k]` will be simplified if possible. This is achieved in MathTensor by sorting and renaming all dummy indices using the same algorithm and trying to simplify the expression obtained in such way. By default, Mathematica will distinguish contracted indices in two expressions if they are named differently, and MathTensor works around it using the renaming scheme implemented in `CanAll`.

The computing resources required for the execution of the many-body SPEEDUP Mathematica code depend strongly on the level of the effective action. For example, for level $p=5$ the code can be run within few seconds with the minimal memory requirements. The notebook with the matching output of this calculation is available as `EffectiveAction-ManyBody-matching-p5.nb`, and the exported results for `W[m, k]` are available in `EffectiveAction-ManyBody-export-p5.m`. We were able to achieve maximal level $p=10$ [19], with the CPU time of around 2 days on a recent 2 GHz processor. The memory used by Mathematica was approximately 1.6 GB.

Note that exporting the definition of the effective potential from Mathematica to a file will yield lower and upper indices named `ll1`, `uu1`, etc. In order to import previous results and use them for further calculations with the provided Mathematica code, it is necessary to replace indices in the exported file to the proper index names used by the function `GenNewInd`. This is easily done using find/replace feature of any text editor. Prior to importing definition of the effective potential, it is necessary to initialize MathTensor and all additional functions defined in the notebook `EffectiveAction-ManyBody.nb`, and to generate the needed number of named indices using the function call `GenNewInd[2p+2]`.

4 SPEEDUP C codes for Monte Carlo calculation of 1D transition amplitudes

For short times of propagation, the effective actions derived using the above described Mathematica codes can be directly used. This has been extensively used in [8, 9], where SPEEDUP codes were applied for numerical studies of several lower-dimensional models and calculation of large number of energy eigenvalues and eigenfunctions. The similar approach is used in [10], where SPEEDUP code was used to study properties of fast-rotating Bose-Einstein condensates in anharmonic trapping potentials. The availability of a large number of eigenstates allowed not only precise calculation of global properties of the condensate (such as condensation temperature and ground state occupancy), but also study of density profiles and construction of time-of-flight absorption graphs, with the exact quantum treatment of all available eigenfunctions.

However, in majority of applications the time of propagation cannot be assumed to be small. The effective actions are found to have finite radius of convergence [8], and if the typical propagation times in the considered case exceed this critical value, Path Integral Monte Carlo approach must be used in order to accurately calculate the transition amplitudes and the corresponding expectation values [4, 18]. As outlined earlier, in this case the time of propagation T is divided into N time steps, such that $\varepsilon = T/N$ is sufficiently small and that the effective action approach can be used. The discretization of the propagation time leads to the following expression for the discretized amplitude

$$A_N^{(p)}(a,b;T) = \int \frac{dq_1 \cdots dq_{N-1}}{(2\pi\varepsilon)^{N/2}} e^{-S_N^{(p)}}, \tag{4.1}$$

where $S_N^{(p)}$ stands for the discretized level p effective action,

$$S_N^{(p)} = \sum_{k=0}^{N-1} \left[\frac{(q_{k+1}-q_k)^2}{2\varepsilon} + \varepsilon W_p(x_k, \bar{x}_k; \varepsilon) \right], \tag{4.2}$$

and $q_0 = a$, $q_N = b$, $x_k = (q_{k+1}+q_k)/2$, $\bar{x}_k = (q_{k+1}-q_k)/2$.

Level p discretized effective action is constructed from the corresponding effective potential W_p, calculated as power series expansion to order ε^{p-1}. Since it enters the action multiplied by ε, this leads to discretized actions correct to order ε^p, i.e. with the errors of the order ε^{p+1}. The long-time transition amplitude $A_N^{(p)}(a,b;T)$ is a product of N short-time amplitudes, and its errors are expected to scale as $N \cdot \varepsilon^{p+1} \sim 1/N^p$, as has been shown in [5, 13–15] for transition amplitudes, and in [5, 18] for expectation values, calculated using the corresponding consistently improved estimators.

4.1 Algorithm and structure of the code

SPEEDUP C source is located in the `src` directory of the code distribution [19]. It uses the standard Path Integral Monte Carlo algorithm for calculation of transition amplitudes.

The trajectories are generates by the bisection algorithm [20], hence the number of time-steps N is always given as a power of two, $N=2^s$. When the amplitude is calculated with 2^s time steps, we can also easily calculate all discretized amplitudes in the hierarchy $2^{s-1},\cdots,2^0$ at no extra cost. This requires only minor additional CPU time and memory, since the needed trajectories are already generated as subsets of maximal trajectories with 2^s time-steps.

The trajectory is constructed starting from bisection level $n=0$, where we only have initial and final position of the particle. At bisection level $n=1$ the propagation is divided into two time-steps, and we have to generate coordinate q of the particle at the moment $T/2$, thus constructing the piecewise trajectory connecting points a at the time $t=0$, q at $t=T/2$, and b at $t=T$. The coordinate q is generated from the Gaussian probability density function centered at $(a+b)/2$ and with the width $\sigma_1=\sqrt{T/2}$. The procedure continues iteratively, and each time a set of points is added to the piecewise trajectory. At each bisection level n the coordinates are generated from the Gaussian centered at mid-point of coordinates generated at level $n-1$, with the width $\sigma_n=\sqrt{T/2^n}$. To generate numbers η from the Gaussian centered at zero we use Box-Müller method,

$$\eta=\sqrt{-2\sigma_n^2\ln\xi_1}\cos 2\pi\xi_2, \tag{4.3}$$

where numbers ξ_1 and ξ_2 are generated from the uniform distribution on the interval $[0,1]$, using the SPRNG library [42]. If the target bisection level is s, then at bisection level $n\leq s$ we generate 2^{n-1} numbers using the above formula, and construct the new trajectory by adding to already existing points the new ones, according to

$$q[(1+2i)\cdot 2^{s-n}]=\eta_i+\frac{q[i\cdot 2^{s-n+1}]+q[(i+1)\cdot 2^{s-n+1}]}{2}, \tag{4.4}$$

where i runs from 0 to $2^{n-1}-1$. This ensures that at bisection level s we get trajectory with $N=2^s$ time-steps, consisting of $N+1$ points, with boundary conditions $q[0]=a$ and $q[N]=b$. At each lower bisection level n, the trajectory consists of 2^n+1 points obtained from the maximal one (level s trajectory) as a subset of points $q[i\cdot 2^{s-n}]$ for $i=0,1,\cdots,2^n$.

The use of trajectories generated by the bisection algorithm requires normalization factors from all Gaussian probability density functions with different widths to be taken into account. This normalization is different for each bisection level, but can be calculated easily during the initialization phase.

The basic C code is organized in three source files, `main.c`, `p.c` and `potential.c`, with the accompanying header files. The file `potential.c` (its name can be changed, and specified at compile time) must contain a user-supplied function `V0()`, defining the potential V. For a given input value of the coordinate, `V0()` should initialize appropriate variables to the value of the potential V and its higher derivatives to the required order $2p-2$. When this file is prepared, SPEEDUP code can be compiled and used. The distributed source contains definition of 1D-AHO potential in the file `potential.c`, the same as in the file `1D-AHO.c`.

The execution of the SPEEDUP code starts with the initialization and allocation of memory in the `main()` function, and then the array of amplitudes and associated MC error estimates for each bisection level $n = 0, \cdots, s$ is calculated by calling the function `mc()`. After printing the output, `main()` deallocates used memory and exits. Function `mc()` which implements the described MC algorithm is also located in the file `main.c`, as well as the function `distr()`, which generates maximal (level s) trajectories.

The function `mc()` contains main MC sampling loop. In each step new level s trajectory is generated by calling the function `distr()`. Afterwards, for each bisection level n, function `func()` is invoked. This function is located in the file `p.c`, and returns the value of the function e^{-S}, properly normalized, as described earlier. This value (and its square) is accumulated in the MC loop for each bisection level n and later averaged to obtain the estimate of the corresponding discretized amplitude and the associated MC error.

The function `func()` makes use of C implementation of earlier derived effective actions for a general 1D potential. For a given trajectory at the bisection level n, `func()` will first initialize appropriate variables with the values of the potential and its higher derivatives (to the required level $2p-2$) by calling the user-supplied function `V0()`, located in the file `potential.c`. Afterwards the effective action is calculated according to Eq. (4.2), where the effective potential is calculated by the function `Wp()`, located in the file `p.c`. The desired level p of the effective action is selected by defining the appropriate pre-processor variable when the code is compiled.

In addition to this basic mode, when SPEEDUP code uses general expression for level p effective action, we have also implemented model-specific mode, described earlier. If effective actions are derived for a specific model, then user can specify an alternative `p.c` file to be used within the directory `src/models/<model>`, where `<model>` corresponds to the name of the model. If this mode is selected at compile time, the compiler will ignore `p.c` from the top `src` directory, and use the model-specific one, defined by the user. The distributed source contains model definitions for 1D-AHO and 1D-MPT potentials in directories `src/models/1D-AHO` and `src/models/1D-MPT`. Note that in this mode the potential is specified directly in the definition of the effective potential, and therefore the function `V0()` is not used (nor the `potential.c` file).

4.2 Compiling and using SPEEDUP C code

SPEEDUP C source can be easily compiled using the `Makefile` provided in the top directory of the distribution. The compilation has been thoroughly tested with GNU, Intel and IBM XLC compilers. In order to compile the code one has to specify the compiler which will be used in the `Makefile` by setting appropriately the variable `COMPILER`, and then to proceed with the standard command of the type `make <target>`, where `<target>` could be one of `all`, `speedup`, `sprng`, `clean-all`, `clean-speedup`, `clean-sprng`.

The SPRNG library [42] is an external dependency, and for this reason it is located in the directory `src/deps/sprng4.0`. In principle, it has to be compiled only once, after the compiler has been set. This is achieved by executing the command `make sprng`. After-

wards the SPEEDUP code can be compiled and easily linked with the already compiled SPRNG library. Note that if the compiler is changed, SPRNG library has to be recompiled with the same compiler in order to be successfully linked with the SPEEDUP code.

To compile the code with level $p=10$ effective action and user-supplied function `V0()` located in the file `src/1D-AHO.c`, the following command can be used:

```
make speedup P=10 POTENTIAL=1D-AHO.c
```

If not specified, `POTENTIAL=potential.c` is used, while the default level of the effective action is `P=1`. To compile the code using a model-specific definition of the effective potential, instead of the `POTENTIAL` variable, we have to appropriately set the `MODEL` variable on the command line. For example, to compile the supplied `p.c` file for 1D-MPT model located in the directory `src/models/1D-MPT` using the level $p=5$ effective action, the following command can be used:

```
make speedup P=5 MODEL=1D-MPT
```

All binaries compiled using the `POTENTIAL` mode are stored in the `bin` directory, while the binaries for the `MODEL` mode are stored in the appropriate `bin/models/<model>` directory. This information is provided by the `make` command after each successful compilation is done.

The compilation is documented in more details in the supplied `README.txt` files. The distribution of the SPEEDUP code also contains examples of compilation with the GNU, Intel and IBM XLC compilers, as well as matching outputs and results of the execution for each tested compiler, each model, and for a range of levels of the effective action p.

Once compiled, the SPEEDUP code can be used to calculate long-time amplitudes of a system in the specified potential V. If executed without any command-line arguments, the binary will print help message, with details of the usage. The obligatory arguments are time of propagation `T`, initial and final position `a` and `b`, maximal bisection level `s`, number of MC samples `Nmc` and `seed` for initialization of the SPRNG random number generator. All further arguments are converted to numbers of the `double` type and made available in the array `par` to the function `V0()`, or to the model-specific functions in the file `src/models/<model>/p.c`. The output of the execution contains calculated value of the amplitude for each bisection level $n=0,\cdots,s$ and the corresponding MC estimate of its error (standard deviation). At bisection level $n=0$, where no integrals are actually calculated and the discretized $N=1$ amplitude is simply given by an analytic expression, zero is printed as the error estimate.

Fig. 2 illustrates the typical results obtained from the SPEEDUP code on the example of 1D-MPT theory. In this figure we can see the convergence of numerically calculated amplitudes with the number of time-steps N to the exact continuum value, obtained in the limit $N\to\infty$. Such convergence is obtained for each level p of the effective action used. However, the convergence is much faster when higher-order effective action is used. Note that all results corresponding to the one value of level p on the graph are obtained from a single run of the SPEEDUP code with the maximal bisection level $s=10$. The simplest way to estimate the continuum value of the amplitude is to fit numerical

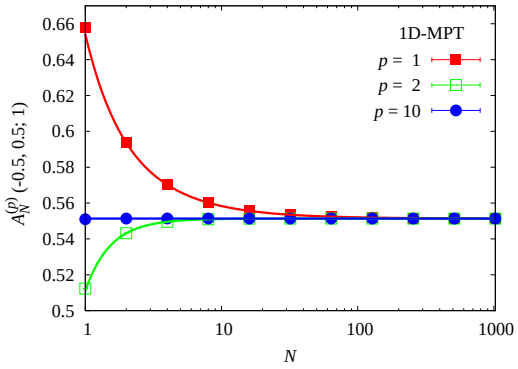

Figure 2: Convergence of SPEEDUP Monte-Carlo results for the transition amplitude $A_N^{(p)}(-0.5,0.5;1)$ of 1D-MPT potential as a function of the number of time steps N, calculated with level $p=1,2,10$ effective actions, with the parameters of the potential $\lambda=\alpha=1$. The full lines give the fitted functions (4.5), where the constant term A_p corresponds to the continuum-theory amplitude $A(-0.5,0.5;1)$. The number of Monte-Carlo samples was $N_{\rm MC}=10^6$.

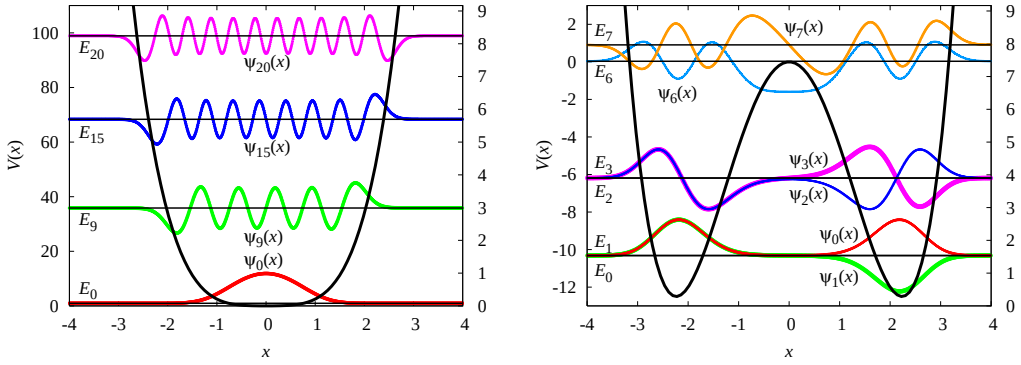

Figure 3: (left) The anharmonic potential 1D-AHO, its energy eigenvalues (horizontal lines) and eigenfunctions, obtained by direct diagonalization of the space-discretized matrix of the evolution operator with level $p=21$ effective action and parameters $A=1$, $g=48$. The discretization cutoff was $L=8$, spacing $\Delta=9.76\cdot10^{-4}$, and time of propagation $t=0.02$. (right) Results for the double-well potential, $A=-10$, $g=12$, $L=10$, $\Delta=1.22\cdot10^{-3}$, $t=0.1$. On both graphs, left y-axis corresponds to $V(x)$ and energy eigenvalues, while scale on the right y-axis corresponds to values of eigenfunctions, each vertically shifted to level with the appropriate eigenvalue.

results from single run of the code to the appropriate level p fitting function [13–15],

$$A_N^{(p)} = A^{(p)} + \frac{B^{(p)}}{N^p} + \frac{C^{(p+1)}}{N^{p+1}} + \cdots \tag{4.5}$$

The constant term obtained by fitting corresponds to the best estimate of the exact amplitude which can be found from the available numerical results.

As mentioned earlier, the effective action approach can be used for accurate calculation of a large number of energy eigenstates and eigenvalues by diagonalization of the space-discretized matrix of transition amplitudes [6–10]. Fig. 3 illustrates this for the case

of an anharmonic and double-well potential. The graph on the left gives several eigenvalues and eigenstates for 1D-AHO potential with $A=1$ and quartic anharmonicity $g=48$, while the graph on the right gives low-lying spectrum and eigenfunctions of the double-well potential, obtained for $A=-10$, with the moderate anharmonicity $g=12$. More details on this approach, including study of all errors associated with the discretization process, can be found in [8,9].

5 Conclusions

In this paper we have presented SPEEDUP Mathematica and C codes, which implement the effective action approach for calculation of quantum mechanical transition amplitudes. The developed Mathematica codes provide an efficient tool for symbolic derivation of effective actions to high orders for specific models, for a general 1D, 2D and 3D single-particle theory, as well as for a general many-body systems in arbitrary number of spatial dimensions. The recursive implementation of the code allows symbolic calculation of extremely high levels of effective actions, required for high-precision calculation of transition amplitudes.

For calculation of long-time amplitudes we have developed SPEEDUP C Path Integral Monte Carlo code. The C implementation of a general 1D effective action to maximal level $p=18$ and model-specific effective actions provide fast $1/N^p$ convergence to the exact continuum amplitudes.

Further development of the SPEEDUP C codes will include parallelization using MPI, OPENMP and hybrid programming model, C implementation of the effective potential to higher levels p, as well as providing model-specific effective actions for relevant potentials, including many-body systems.

Acknowledgments

The authors gratefully acknowledge useful discussions with Axel Pelster and Vladimir Slavnić. This work was supported in part by the Ministry of Education and Science of the Republic of Serbia, under project No. ON171017, and bilateral project NAD-BEC funded jointly with the German Academic Exchange Service (DAAD), and by the European Commission under EU FP7 projects PRACE-1IP, HP-SEE and EGI-InSPIRE.

References

[1] R. P. Feynman, Rev. Mod. Phys. 20, 367 (1948).
[2] R. P. Feynman and A. R. Hibbs, Quantum Mechanics and Path Integrals (McGraw-Hill, New York, 1965).
[3] H. Kleinert, Path Integrals in Quantum Mechanics, Statistics, Polymer Physics, and Financial Markets, 5th ed. (World Scientific, Singapore, 2009).

[4] D. Stojiljković, A. Bogojević, and A. Balaž, Phys. Lett. A 360, 205 (2006).
[5] A. Bogojević, I. Vidanović, A. Balaž, and A. Belić, Phys. Lett. A 372, 3341 (2008).
[6] A. Sethia, S. Sanyal, and Y. Singh, J. Chem. Phys. 93, 7268 (1990).
[7] A. Sethia, S. Sanyal, and F. Hirata, Chem. Phys. Lett. 315, 299 (1999).
[8] I. Vidanović, A. Bogojević, and A. Belić, Phys. Rev. E 80, 066705 (2009).
[9] I. Vidanović, A. Bogojević, A. Balaž, and A. Belić, Phys. Rev. E 80, 066706 (2009).
[10] A. Balaž, I. Vidanović, A. Bogojević, and A. Pelster, Phys. Lett. A 374, 1539 (2010).
[11] R. P. Feynman, Statistical Mechanics (W. A. Benjamin, New York, 1972).
[12] G. Parisi, Statistical Field Theory (Addison Wesley, New York, 1988).
[13] A. Bogojević, A. Balaž, and A. Belić, Phys. Rev. Lett. 94, 180403 (2005).
[14] A. Bogojević, A. Balaž, and A. Belić, Phys. Rev. B 72, 064302 (2005).
[15] A. Bogojević, A. Balaž, and A. Belić, Phys. Lett. A 344, 84 (2005).
[16] A. Bogojević, A. Balaž, and A. Belić, Phys. Rev. E 72, 036128 (2005).
[17] A. Balaž, A. Bogojević, I. Vidanović, and A. Pelster, Phys. Rev. E 79, 036701 (2009).
[18] J. Grujić, A. Bogojević, and A. Balaž, Phys. Lett. A 360, 217 (2006).
[19] SPEEDUP code distribution, `http://www.scl.rs/speedup/`
[20] D. M. Ceperley, Rev. Mod. Phys. 67, 279 (1995).
[21] M. Takahashi and M. Imada, J. Phys. Soc. Jpn. 53, 3765 (1984).
[22] X. P. Li and J. Q. Broughton, J. Chem. Phys. 86, 5094 (1987).
[23] H. De Raedt and B. De Raedt, Phys. Rev. A 28, 3575 (1983).
[24] S. Jang, S. Jang, and G. Voth, J. Chem. Phys. 115, 7832 (2001).
[25] N. Makri and W. H. Miller, Chem. Phys. Lett. 151, 1 (1988); N. Makri and W. H. Miller, J. Chem. Phys. 90, 904 (1989).
[26] N. Makri, Chem. Phys. Lett. 193, 435 (1992).
[27] M. Alford, T. R. Klassen, and G. P. Lepage, Phys. Rev. D 58, 034503 (1998).
[28] S. A. Chin and E. Krotscheck, Phys. Rev. E 72, 036705 (2005).
[29] E. R. Hernández, S. Janecek, M. Kaczmarski, E. Krotscheck, Phys. Rev. B 75, 075108 (2007).
[30] O. Ciftja and S. A. Chin, Phys. Rev. B 68, 134510 (2003).
[31] K. Sakkos, J. Casulleras, and J. Boronat, J. Chem. Phys. 130, 204109 (2009).
[32] S. Janecek and E. Krotscheck, Comput. Phys. Comm. 178, 835 (2008).
[33] A. D. Bandrauk and H. Shen, J. Chem. Phys. 99, 1185 (1993).
[34] S. A. Chin and C. R. Chen, J. Chem. Phys. 117, 1409 (2002).
[35] I. P. Omelyan, I. M. Mryglod, and R. Folk, Comput. Phys. Commun. 151, 272 (2003).
[36] G. Goldstein and D. Baye, Phys. Rev. E 70, 056703 (2004).
[37] S. A. Chin, arXiv:0809.0914.
[38] S. A. Chin, S. Janecek, and E. Krotscheck, Comput. Phys. Comm. 180, 1700 (2009).
[39] S. A. Chin, S. Janecek, and E. Krotscheck, Chem. Phys. Lett. 470, 342 (2009).
[40] Mathematica software package, `http://www.wolfram.com/mathematica/`
[41] MathTensor package, `http://smc.vnet.net/mathtensor.html`
[42] Scalable Parallel Random Number Generator library, `http://sprng.fsu.edu/`