OCTBEC – A Matlab toolbox for optimal quantum control of Bose-Einstein condensates

Ulrich Hohenester

Abstract

OCTBEC is a Matlab toolbox designed for optimal quantum control, within the framework of optimal control theory (OCT), of Bose-Einstein condensates (BEC). The systems we have in mind are ultracold atoms in confined geometries, where the dynamics takes place in one or two spatial dimensions, and the confinement potential can be controlled by some external parameters. Typical experimental realizations are atom chips, where the currents running through the wires produce magnetic fields that allow to trap and manipulate nearby atoms. The toolbox provides a variety of Matlab classes for simulations based on the Gross-Pitaevskii equation, the multi-configurational Hartree method for bosons, and on generic few-mode models, as well as optimization problems. These classes can be easily combined, which has the advantage that one can adapt the simulation programs flexibly for various applications.

Keywords: Bose-Einstein condensates, optimal quantum control, atom chips, Gross-Pitaevskii equation, Multi-configurational time dependent Hartree method for bosons

Program summary

Program title: OCTBEC
Programming language: Matlab 7.11.0 (R2010b)
Computer: Any which supports Matlab 7.11.0 (R2010b)
Operating system: Any which supports Matlab 7.11.0 (R2010b)
RAM required to execute with typical data: ≥ 1 GByte
Has the code been vectorised or parallelized?: no
Keywords: Bose-Einstein condensates, optimal quantum control, atom chips
External routines/libraries used: none
Nature of problem: Simulation of Bose-Einstein condensates and optimal quantum control
Solution method: Gross-Pitaevskii equation, multi-configurational Hartree method for bosons, generic few-mode models
Running time: between seconds and hours

1. Introduction

Bose-Einstein condensates and ultracold atoms in atom chips provide an ideal laboratory for the study of quantum physics under well-controlled conditions. The possibility to store, manipulate, and measure single quantum systems with extremely high precision has initiated great stimulus in various fields of research,
ranging from atom interferometry [4, 6–8], over quantum gates [9–11] and resonant condensate transport [12], to nonlinear atom optics [13–16]. In the vast majority of these schemes the wavefunction of the Bose-Einstein condensate, trapped in the vicinity of an atom chip [17], is manipulated through variation of the magnetic confinement potential. This is achieved by changing the currents through the gate wires mounted on the chip or modifying the strength of additional radio-frequency fields [5, 18–21]. These external, time-dependent parameters thus provide a versatile control for wavefunction manipulations, and make atom chips attractive candidates for quantum control applications.

In the past, we have successfully developed and implemented an optimal quantum control approach for Bose-Einstein condensates in magnetic microtraps within the framework of the Gross-Pitaevskii equation [22, 23]. More recently, this approach has been generalized to more sophisticated description schemes, such as the two-mode model [24, 25] or the multi-configurational Hartree method for bosons (MCTDHB) [26], which allow to additionally describe condensate fragmentations and excitations. Atom number-squeezing and atom interferometry at the Heisenberg limit has been studied in a number of papers [27–32]. Most recently, we have presented first results that show that optimal quantum control protocols can be successfully implemented in experiment [33–35].

Our computational approach is based on a Runge-Kutta or Crank-Nicolson solution of the dynamic equations, together with an optimal control framework using conjugate gradient or quasi-Newton minimization techniques [22, 23, 28, 36, 37]. Despite the large diversity of problems studied in the past, the simulations are similar in many aspects, and at some point we realized that we were wasting a significant amount of time copying code from one program to another one.

The purpose of the OCTBEC toolbox is to provide a unified platform for the simulation of Bose-Einstein condensates and quasi-condensates in restricted geometries, where the confinement potential can be modified by some external control parameters, and to allow for the optimization of the time variation of these control parameters within the framework of optimal control theory. Typical simulation scenarios consist of a few hundreds to thousands of atoms, with confinement lengths and manipulation times of the order of micrometers and milliseconds, respectively. A modular structure of the program has been achieved by using classdef objects available with Matlab 7.6 or higher versions. These classes can be easily combined such that one can adapt the simulation programs flexibly to the user’s needs. A comprehensive help is available for all classes and functions of the toolbox through the doc command. In addition, we have created detailed help pages, accessible in the Matlab help browser, together with a complete list of the classes and functions of the toolbox, and a number of demo programs. In this paper we provide an ample overview of the OCTBEC toolbox, but leave several details to the help pages.

The model systems covered by the toolbox include:

**Gross-Pitaevskii.** The Gross-Pitaevskii equation describes the condensate dynamics in terms of a single wavefunction, and the non-linear atom-atom interactions are accounted for through a mean-field approach [38, 39]. The Gross-Pitaevskii framework proves to be extremely successful for problems where condensate fragmentation or excitation are of minor importance.

**Few-mode model.** In the few-mode model, atoms become distributed between static or time-dependent orbitals and the time evolution is governed by a Hamiltonian matrix [24, 25]. The whole condensate dynamics is then associated with the atom number distribution, whereas the orbital degrees of freedom are lumped into a few effective parameters.

**MCTDHB.** In the multiconfigurational time-dependent Hartree method for bosons (MCTDHB) one accounts for both the spatial and atom-number dynamics. The approach has been developed by Cederbaum, Alon, and coworkers [26], and allows for an ab-initio description of the condensate dynamics, at least in principle. We refer the interested reader to www.pci.uni-heidelberg.de/tc/usr/mctdhb/ where a collection of papers and software programs can be found.

In the toolbox we provide implementations for the simulation of the above models. In comparison to the OpenMCTDHB software provided by the Cederbaum group, our implementation is probably less refined, but
can be used in combination with optimal control applications. As regarding the structure of the toolbox and the philosophy behind its implementation, a number of comments are at place.

Whenever possible, we have tried to favor readability and transparency over runtime and memory requirements. It is likely that the performance of most programs could be significantly improved, yet we have tried to stick to a modular structure and to strict programming rules throughout. We hope that with this approach the programs are easier to read, and that other model systems can be implemented without too much knowledge about the working principles of the remaining toolbox. Quite generally, our primary interest is to apply the software to physically interesting problems, rather than to develop software and methodology. In this respect, a flexible software platform is extremely helpful because it allows to build on already established expertise, and to devote work and time to the novel aspects of a given problem only.

This paper is organized as follows. In Sec. 2 we briefly review the different model systems and provide a short overview of the toolbox. The toolbox installation and a few selected examples are discussed in Sec. 3. In Sec. 4 we describe the unit system, the computational grid, the control parameters, and the ODE solvers provided by the toolbox. The implementation of the different model systems is explained in Sec. 5. Finally, optimal control theory and its implementation are explained in Sec. 6 and applied to the different model systems in Sec. 7. Details about some of our Crank-Nicolson and OCT implementations can be found in the Appendices.

2. Theory and brief overview

2.1. Model systems

Our starting point for the description of condensate dynamics in restricted geometries is the many-body Hamiltonian in second-quantized form \[ H = \int \hat{\Psi}^\dagger(x) \left[ -\frac{\hbar^2}{2M} \frac{\partial^2}{\partial x^2} + V(x, \lambda(t)) \right] \hat{\Psi}(x) \, dx + \kappa \int \hat{\Psi}^\dagger(x) \hat{\Psi} \, dx, \] (1)
The first term on the right-hand side accounts for the kinetic energy, where \( \hbar \) is the reduced Planck constant and \( M \) the atom mass, as well as for the magnetic confinement potential \( V(x, \lambda(t)) \). The control parameter \( \lambda(t) \) determines the variation of the confining potential when changing the external parameters \[ \] (for details see below). Through \( \lambda(t) \) it is possible to manipulate the trapped Bose-Einstein condensate, e.g. to split it by varying the potential from a single to a double well, or to excite it by displacing the potential minimum. The second term on the right-hand side accounts for the atom-atom interactions, where we have chosen a contact potential approximation for the interatomic potential \[ \] . The bosonic field operators \( \hat{\Psi}(x) \) and \( \hat{\Psi}^\dagger(x) \) obey the usual equal-time commutation relations.

Different physical regimes emerge from Eq. (1) by restricting the field operators to certain types of basis functions. First, the Gross-Pitaevskii equation \[ \] is obtained by assuming that all atoms reside in a single “orbital” \( \psi(x, t) \). Correspondingly, the field operator is
\[ \hat{\Psi}(x) = \hat{a}_0 \psi(x, t), \] (2)
where \( \hat{a}_0 \) is the operator associated with the condensate. The Gross-Pitaevskii equation properly accounts for the mean-field dynamics of the condensate, described by the orbital \( \psi(x, t) \), but cannot cope with correlation effects and fragmentation, where more than a single orbital becomes populated.

A prominent example is splitting of a condensate, which can be achieved by transforming the confinement potential from a single to a double well, where at some point the condensate breaks up into two parts, \( \psi_L(x) \) and \( \psi_R(x) \), which are localized in either the left or right well. Close to the splitting point, we can approximately ignore the dynamics of the orbitals \( \psi_{L,R}(x) \), and recover the \textit{two-mode model} \[ \] \[ \hat{\Psi}(x) = \hat{a}_L \psi_L(x) + \hat{a}_R \psi_R(x). \] (3)
Here, the whole condensate dynamics is associated with the atom number distribution, through the field operators \( \hat{a}_L \) and \( \hat{a}_R \), whereas the orbital degrees of freedom are lumped into a few effective parameters.
Figure 1: Overview of the OCTBEC Toolbox for the solution of the Gross-Pitaevskii, MCTDHB, and fewmode models. An external control parameter allows to modify the confinement potential or other model parameters through which the wavefunction can be steered. Within the toolbox one can either solve the dynamic equations, or submit the problem to optimal control theory. Here an optimal control is determined, which minimizes a cost function that parameterizes the control objective. In the figure we add class and function names that can be assigned to the specific tasks.

We will refer to such kind of systems as few-mode models. Quite generally, such models can also describe the atom dynamics in optical lattices [40] within the Bose-Hubbard framework [41].

In the most general case, we can neither neglect the orbital nor the atom number dynamics. This can be done by choosing a field operator of the form

\[ \hat{\Psi}(x) = \hat{a}_L(t) \psi_L(x,t) + \hat{a}_R(t) \psi_R(x,t), \] (4)

where \( \psi_{L,R}(x,t) \) are time-dependent orbitals that have to be determined self-consistently. A convenient approach is provided by the multi-configurational time dependent Hartree method for bosons (MCTDHB) [26] which determines the orbitals from a variational principle.

2.2. Brief overview

The main purpose of the OCTBEC toolbox is to provide a flexible toolkit for the simulation of condensate dynamics in confined geometries, where the confinement can be modified by some external control parameters. The theoretical frameworks covered by the toolbox go from the Gross-Pitaevskii model, over fewmode models, to the multi-configurational Hartree method for bosons. Figure 1 provides a detailed overview of the different tasks which can be separated into (i) simulation of the dynamics, and (ii) quantum control within the framework of optimal control theory. In the following we discuss prototypical examples and provide details about the different classes and functions of the toolbox.

3. Getting started

3.1. Installation of the toolbox

To install the toolbox, one must simply add the path of the main directory octbecdir of the OCTBEC toolbox as well as the paths of all subdirectories to the Matlab search path. This can be done, for instance, through
addpath(genpath(octbecdir));

To set up the help pages, one must once change to the main directory of the OCTBEC toolbox and run the program makeoctbechelp

```matlab
>> cd octbecdir;
>> makeoctbechelp;
```

Once this is done, the help pages, which provide detailed information about the toolbox, are available in the Matlab help browser. Note that one may have to call Start > Desktop Tools > View Start Button Configuration > Refresh to make the help accessible. Under Matlab 2012 the help pages can be found on the start page of the help browser under Supplemental Software. Figure 2 shows a screenshot of the OCTBEC help pages. This manuscript closely follows the help pages, but additionally provides further details about the theory and methodology underlying our computational approach.

3.2. A few selected examples

To get a first idea what the toolbox can do, it is probably best to work through some of the demo programs listed in the OCTBEC Examples section of the help pages. By typing

```matlab
>> demogp1
```

at the Matlab prompt, a simulation is performed where a condensate wavefunction is split by transforming the potential from a single to a double well (the demo file can be opened in the Matlab editor with `edit demogp1.m`). Results are shown in Fig. 3(a). Such splitting has been described in some length in Ref. [22], and details of the demo program will be presented in Sec. 5.1.

The demo program `demomctdhb1` investigates a similar splitting scenario, however within the framework of the MCTDHB(2) equations [26]. The program performs somewhat slower, which is due to the complexity
of the underlying equations that account for both the orbital dynamics and the distribution of atoms between these orbitals. For the investigated splitting scenario the results are very similar to those of the Gross-Pitaevskii equation.

Finally, we briefly discuss an optimal control simulation where the protocol for the splitting process is optimized such that the condensate ends up in the groundstate of the split trap at the terminal time. Such splitting has been investigated in Ref. [22], and details will be presented in Sec. 7.1. We start the program with

```matlab
>> demogpoct1  % runtime about 90 sec
```

The program opens a graphics window that shows the iterative improvement of the control parameter. In addition, in the Matlab window the progress of the optimal control loop is reported

```plaintext
it= 1  f=1.518970e-001  ||g||=9.910656e-001  sig=0.125
it= 2  f=5.713967e-002  ||g||=9.593845e-003  sig=0.126
it= 3  f=2.144892e-002  ||g||=3.831657e-002  sig=5.144
it= 4  f=1.229915e-002  ||g||=3.22441e-003  sig=0.951
it= 5  f=1.78243e-002  ||g||=5.26612e-004  sig=1.363
it= 6  f=1.699185e-002  ||g||=4.061750e-003  sig=5.821
it= 7  f=1.456352e-002  ||g||=5.018247e-004  sig=3.480
it= 8  f=1.451606e-002  ||g||=3.001156e-004  sig=0.522
```

Here, `it` gives the iteration number of the optimization loop, `f` is the cost function value [22], and `||g||` is the norm of the gradient which should become zero for the optimal control. Figs. 3(b,c) show the time evolution of the particle density and the initial and optimal control, respectively. As can be seen, for the optimal control the condensate ends up close to the groundstate of the split trap, and only minor oscillations occur at later times when the trap is kept constant.

Table 1 lists a number of additional demo programs, and provides typical runtimes as well as a brief explanation. Details about the classes and functions can be found in the help pages of the toolbox or by typing `doc @classname` at the Matlab prompt.

4. Preliminaries

In this section we introduce several basic classes and concepts needed for the simulation of BECs in confined geometries. We first briefly describe the unit system adopted in the demo files (the toolbox itself does not rely on a specific unit system), and then introduce the computational grid used for the discretization of the spatial domain and our implementation of control parameters. Finally, we present the solvers for ordinary differential equations provided by the toolbox.
Table 1: Selected examples provided by the OCTBEC toolbox. We list the names of the programs, typical runtimes, and give brief explanations. The programs were tested on a standard PC (Intel i7–2600 CPU, 3.40 GHz, 8 GB RAM). The different runtimes for Gross-Pitaevskii and MCTDH simulations can be inferred from the comparison of `demogp1` and `demomctdhb1`, as well as `demogpoc1t` and `demomctdhoct1` for OCT simulations.

| Demo program   | Runtime | Description                                                                 |
|----------------|---------|-----------------------------------------------------------------------------|
| demogp1        | 0.75 sec| Splitting of condensate wavefunction within Gross-Pitaevskii framework       |
| demogp1split   | 0.57 sec| Same as `demogp1` but using split-operator integration                       |
| demogp2        | 0.80 sec| Solution of 2d Gross-Pitaevskii equation with split operator                 |
| demogp3        | 4.47 sec| Shake-up process within Gross-Pitaevskii framework                          |
| demogpoc1t     | 89.02 sec| OCT simulation of condensate splitting within Gross-Pitaevskii framework     |
| demogpoc2t     | 93.21 sec| Same as `demogpoc1t` but energy minimization for cost function              |
| demogpoc3t     | 350.06 sec| OCT of shake-up process within Gross-Pitaevskii framework                    |
| demonctdhb1    | 4.91 sec| Splitting of condensate wavefunction within MCTDHB framework                |
| demonctdhoct1  | 942.27 sec| Same as `demogpoc1t` but for MCTDHB model                                    |
| demonctdhoct2  | 853.09 sec| Same as `demogpoc2t` but for MCTDHB model                                   |
| demofewmodepair| 1.42 sec| Number squeezing through splitting                                           |
| demofewmodepairbloch | 11.31 sec| Visualization of number squeezing on Bloch sphere                           |

Table 2: A few selected properties and methods of the `grid1d` class. Use `doc @grid1d` to get a complete listing of all class properties and methods.

| Property | Type       | Description                          |
|----------|------------|--------------------------------------|
| n        | Integer    | Number of positions                  |
| x        | Double array| Positions of grid                    |
| grad     | Sparse matrix| Derivative operator                  |
| grad4    | Sparse matrix| Derivative operator (4th order accuracy) |
| lap      | Sparse matrix| Laplace operator                     |
| lap4     | Sparse matrix| Laplace operator (4th order accuracy) |
| norm     | Function    | Norm of wavefunction                 |
| normalize| Function    | Normalize wavefunction               |
| inner    | Function    | Inner product of two wavefunctions   |
| integrate| Function    | Integrate function on grid           |

4.1. Units

The toolbox does not use a specific unit system, and the choice of units is in principle left to the user. In the demo files we use units where length is measured in micrometers, time is measured in milliseconds, and the reduced Planck constant is set to \( \hbar = 1 \). With this choice, atom masses have to be given in units of

\[
M_{\text{nucl}} / (h \times L),
\]

with \( M_{\text{nucl}} \) being the nucleon mass and \( L = 1 \mu m \) the length unit. For instance, to properly set the mass of Rubidium atoms one then proceeds as follows:

```matlab
units; % load units (defines mass)
massRb = 87 * mass; % mass of Rubidium atoms (mass number 87)
```

We believe that this unit system is well suited for the problems under study, and recommend to use it whenever possible.

4.2. Spatial grid

4.2.1. One-dimensional grid

In our computational approach, we represent the spatial domain by a grid of discrete points and approximate function derivatives by finite differences. Consider the situation where the one-dimensional domain \( x \in [x_{\text{min}}, x_{\text{max}}] \) is represented by \( n \) discrete points. Within the OCTBEC toolbox, one calls
Figure 4: Confinement potential (blue dashed line) and eigenfunctions (solid lines) for a harmonic confinement potential.

grid = grid1d( xmin, xmax, n ); % initialization of computational grid in 1d

Upon initialization of grid, a Matlab object is created whose properties are listed in table 2 (use doc @grid1d to get a complete listing of all class properties and methods). The derivative operators, such as grad or lap, can be applied to functions through simple multiplication. Let us consider the example of Rubidium atoms inside a harmonic trap. To compute the condensate groundstate in absence of nonlinear atom-atom interactions, we proceed as follows.

units; % length in micrometers, time in ms, and hbar = 1
grid = grid1d( -3, 3, 201 ); % initialize grid
% kinetic energy
t = -0.5 * grid.lap4 / ( 87 * mass );
% potential energy for 1 kHz confinement potential
v = 0.5 * ( 87 * mass ) * ( 2 * pi ) ^ 2 * spdiag( grid.x ) .^ 2;
% compute the four eigenfunctions and eigenvalues of lowest energy
[ psi, ene ] = eigs( t + v, 4, 'sa', struct( 'disp', 0 ) );
% plot eigenfunctions and confinement potential
plot( grid.x, 1e-3 * diag( v ), 'b--', grid.x, psi );

We have used the toolbox function spdiag(a) to place the values of a vector a on the diagonal of a sparse matrix. Figure 4 shows the confinement potential and eigenfunctions for this harmonic confinement potential. The grid1d class provides a number of methods that can be used to manipulate wavefunctions. Below we show several examples.

% norm of wavefunction
grid.norm( psi ) % 0.1732 0.1732 0.1732 0.1732
% normalize wavefunctions
psi = grid.normalize( psi );
% norm of wavefunctions
grid.integrate( abs( psi ) .^ 2 ) % 1.0000 1.0000 1.0000 1.0000
% inner product of two wavefunctions
grid.inner( psi(:,1), psi(:,2) ) % -4.2279e-017

4.2.2. Two-dimensional grid

The toolbox provides with grid2d also a two-dimensional grid. Similarly to the one-dimensional case, initialization is done via
Most methods and functions are similar to \texttt{grid1d}, in addition the structure \texttt{grid.mesh} provides the \(x\) and \(y\) coordinates obtained from \texttt{meshgrid} for plotting within Matlab (e.g. through \texttt{imagesc}). A more detailed explanation can be found in the help pages, or can be obtained through \texttt{doc @grid2d}. At present we do not provide a class for grids in all three spatial dimensions.

4.3. Control parameter

The OCTBEC toolbox considers situations where the confinement potential for atoms can be controlled by some external control parameters \(\lambda(t)\) \cite{22}. A typical example are the magnetic fields of an atom chip, where the potential can be modified by changing the currents running through the wires of the chip. To properly access the classes and functions of the toolbox, one usually has to specify the time variation of the control field (or the control fields, if needed), which we assume to be real-valued throughout. To this end, one calls:

\begin{verbatim}
  ttab = linspace( 0, 3, 100 );       % control field tabulated at given times
  lamtab = 0.1 * ttab;               % time variation of control field
  lambda = control( ttab, lamtab );  % set up control field

  lambda( 0.5 )                       % 0.05, interpolate control field at intermediate time
\end{verbatim}

Here we use tabulated values to set up the control object. If several control fields are needed, \texttt{lamtab} should be a matrix rather than an array. The control object can perform interpolations between the tabulated values. Note that \texttt{lam=lambda(t)} returns a \texttt{controltype} object which behaves very much like a double variable or double array, depending on the dimension of the control parameter. However, through \texttt{lam.t} one can additionally access the time at which the control parameter is evaluated, which might be useful for problems with an explicit time dependence. The full strength of the \texttt{control} objects will become clear in connection with optimal control theory (OCT), which seeks for optimal variations of the control fields in order to fulfill certain control objectives.

Usually the control parameter works together with a function or function handle for the confinement potential. For instance,

\begin{verbatim}
  v = @( lambda ) ( 0.5 * k * ( grid.x - lambda ).^ 2 ); % harmonic confinement potential
\end{verbatim}

defines a harmonic confinement potential whose origin is shifted by the control parameter. As another example, we consider the potential of Lesanovsky et al. \cite{19} for the transition from a single well to a double well, as shown in Fig.\texttt{5}.

\begin{verbatim}
  grid = grid2d( xmin, xmax, nx, ymin, ymax, ny );   % initialization of 2d mesh

  grid = grid2d( -3, 3, 301 );                     % set up computational grid
  v = @( grid, lambda ) ( lesanovsky1d( grid.x, lambda ) ); % confinement potential

  tout = linspace( 0, 4, 201 );                    % simulation times
  lambda = control( tout, 0.2 + tout / max( tout ) * 0.6 ); % temporal variation control parameter

  % loop over times
  for t = linspace( 0, max( tout ), 7 )
    plot( grid.x, v( lambda( t ) ), 'LineWidth', 0.5 + t ); end
\end{verbatim}

4.4. ODE solver

The OCTBEC toolbox provides a solver for ordinary differential equations (ODEs) that works together with generic Matlab classes. To work properly, the following operations and functions must be implemented for the class:

\begin{verbatim}
  grid = grid2d( xmin, xmax, nx, ymin, ymax, ny );   % initialization of 2d mesh

  grid = grid2d( -3, 3, 301 );                     % set up computational grid
  v = @( grid, lambda ) ( lesanovsky1d( grid.x, lambda ) ); % confinement potential

  tout = linspace( 0, 4, 201 );                    % simulation times
  lambda = control( tout, 0.2 + tout / max( tout ) * 0.6 ); % temporal variation control parameter

  % loop over times
  for t = linspace( 0, max( tout ), 7 )
    plot( grid.x, v( lambda( t ) ), 'LineWidth', 0.5 + t ); end
\end{verbatim}
Figure 5: Confinement potentials for different values of the control parameter $\lambda(t)$ and for the potential considered by Lesanovsky et al. [19] which describes the transition from a single to a double well.

The function \texttt{deriv} is used by the Runge-Kutta solver and the function \texttt{crank} by the Crank-Nicolson solver. The calls to these functions are of the form

```
dy = deriv( y, lambda, ~ ); % derivative function for Runge-Kutta solver
ynew = crank( yold, lambda, dt, ~ ); % Crank-Nicolson step
```

Here \texttt{y} and \texttt{yold} are user-defined objects, \texttt{lambda} is a control parameter, and \texttt{dt} is the step size of the Crank-Nicolson scheme. Note that the time argument can be accessed via \texttt{lambda.t}. The last argument in the above functions can be used to pass options to the derivative functions. The OCTBEC toolbox provides a number of classes that have already defined all of the above operations and functions.

Once the object is defined, one can solve the ODE through

```
[ yout, tout ] = solve( y0, tout, lambda, op ); % solve differential equation
```

Here \texttt{y0} is the initial value, \texttt{tout} are the times where the output is requested, \texttt{lambda} is a control object, and \texttt{op} is an option structure that controls the ODE integration. The ODE solver returns a cell array \texttt{yout} of output values. The option structure can have the following fields

- \texttt{nout} intermediate output (waitbar) after \texttt{nout} timesteps,
- \texttt{nsub} subdivide each timestep into \texttt{nsub} sub-timesteps.
- \texttt{stepfun} 'crank' for Crank-Nicolson, 'runge4', or any Matlab ODE solver such as 'ode23',
- \texttt{funiter} function to be called after each time step,
- \texttt{funout} function to be called at output steps.

Through \texttt{nsub} it is possible to refine the numerical integration without modifying the output results. Note that with the 'runge4' option a 4th-order Runge-Kutta integration with fixed time steps will be performed. The setting of \texttt{nsub} is decisive only for the ODE solvers \texttt{crank} and \texttt{runge4} with fixed stepsize, for the built-in Matlab ODE solvers \texttt{nsub} simply determines the initial stepsize. In general, we recommend to either use the 'crank' option for fast and efficient integration with low accuracy, or 'ode23' which performs a sufficiently fast Runge-Kutta integration with adaptive step size. The user-defined functions must be of the form

```
y = funiter( t, y ); % function to be called after each iteration
yout = funout( t, y ); % function to be called for output
```
The function `funiter` allows, for instance, to normalize a wavefunction after each time step, to avoid numerical rounding errors or to be used for imaginary time propagation, whereas the function `funout` allows to save only part of the dynamic variables in the output.

5. Model systems

Within the OCTBEC toolbox three model systems have been already implemented, namely the Gross-Pitaevskii, MCTDHB, and fewmode model. All models have a similar structure but their own specialities, and can be submitted to the ODE solvers for solutions with either the Runge-Kutta or Crank-Nicolson stepper scheme. In the following we provide details and examples for the different models. Table 3 lists their methods and properties.

5.1. Gross-Pitaevskii equation

The `grosspitaevskii` object allows for the solution of the Gross-Pitaevskii equation [38, 39]

\[
\frac{i\hbar}{\partial t} \frac{\partial \psi(r,t)}{\partial t} = \left( -\frac{\hbar^2 \nabla^2}{2M} + V(r, \lambda(t)) + \kappa |\psi(r,t)|^2 \right) \psi(r,t).
\]  

The first term on the right-hand side is the operator for the kinetic energy, the second one is the confinement potential that can be controlled by some external parameter \(\lambda(t)\), and the last term is the nonlinear atom-atom interaction in the mean field approximation of the Gross-Pitaevskii framework. \(M\) is the atom mass and \(\kappa\) is the strength of the atom-atom interactions. To set up the `grosspitaevskii` object one calls

```matlab
psi = grosspitaevskii( grid, ham, kappa ); % initialzation of the Gross-Pitaevskii object
```

Here `grid` is the computational grid, `ham` is the Hamiltonian consisting of the kinetic energy and the confinement potential, and `kappa` is the nonlinearity. Note that `ham` must be a function or function handle that depends on the control parameter. The main purpose of the `grosspitaevskii` class is to allow for the simulation of the condensate time evolution within the framework of the Gross-Pitaevskii equation, using confinement potentials that can be controlled by some external control parameter (see Sec. 4.3). Figure 6 schematically depicts the solution scheme we are aiming at.

Let us consider the situation where a condensate initially resides in a one-dimensional harmonic-type well, and the confinement potential is subsequently transformed to a double well. Such a setup has been analyzed in Ref. [22]. In the demo program `demogp1.m` we first set up the computational grid, the confinement potential, and the nonlinearity.
Table 3: Methods and properties for the different model classes. Detailed help can be obtained by typing `doc @grosspitaevskii`, `doc @mctdhb`, or `doc @fockstate`. For writing a new class, one has to implement the methods and properties listed under “All”.

| Model        | Methods                  | Description                                                                 |
|--------------|--------------------------|-----------------------------------------------------------------------------|
| All          | `plus`, `mtimes`, `mrdivide` operations `+`, `*`, `/` | operations `+`, `*`, `/`                                                   |
|              | `display`                | Command window display of class properties                                  |
|              | `subsref`                | Access class properties and functions                                        |
|              | `deriv`                  | Derivative function for Runge-Kutta solvers                                 |
|              | `crank`                  | Crank-Nicolson step                                                         |
|              | `optderiv`               | Derivative for OCT optimality system and Runge-Kutta integration             |
|              | `optcrank`               | Crank-Nicolson step for OCT optimality system                               |
|              | `derivpotential`         | Derivative of Lagrange function with respect to control parameter           |
|              | `pack`                   | Pack wavefunction to column vector for use with Matlab ODE solvers          |
|              | `unpack`                 | Unpack wavefunction from column vector for use with Matlab ODE solvers      |
| Gross-Pitaevskii | `grid`                  | Real-space grid                                                             |
|              | `ham`                    | Single-particle Hamiltonian                                                  |
|              | `kappa`                  | Nonlinearity parameter                                                      |
|              | `orbital`                | Wavefunction of Gross-Pitaevskii object                                      |
|              | `density`                | Particle density in real space                                              |
|              | `groundstate`            | Gross-Pitaevskii groundstate                                                |
|              | `split`                  | Split operator step                                                         |
|              | `optsplit`               | Split operator step for optimality system                                   |
| MCTDHB       |                          | All methods and properties of the Gross-Pitaevskii class are also implemented for the MCTDHB class, with the exception of `split` and `optsplit`. |
|              | `m`                      | Number of orbitals                                                          |
|              | `n`                      | Number of atoms                                                             |
|              | `num`                    | Atom number part of wavefunction                                            |
|              | `spin`                   | Pseudospin object for atom number part of wavefunction                      |
|              | `densitymatrix`          | One- and two-particle density matrices                                       |
| Fockstate    | `num`                    | Atom number part of wavefunction                                            |
|              | `spin`                   | Pseudospin object for atom number part of wavefunction                      |
|              | `ham`                    | Many-particle Hamiltonian                                                    |
|              | `groundstate`            | Groundstate of atom number wavefunction                                      |
units; % length in micrometers, time in ms, hbar = 1
grid = gridid(-3, 3, 101); % simulation grid
v = @( lambda ) ( lesanovsky1d( grid.x, lambda )); % confinement potential
ham = @( lambda ) ( -0.5 * grid.lap4 / ( 87 * mass ) + spdiag( v( lambda ) )); % single particle Hamiltonian
kappa = pi; % nonlinearity

tout = linspace( 0, 2, 100 ); % output time
nt = length( tout ); % number of time steps
lambda = control( tout, sqrt( min( 2 * tout / max( tout ), 1 ) )); % control parameter

Note that the Hamiltonian must be a sparse matrix $H_{ij}$ in real space. For that reason we put the confinement potential via spdiag on the diagonal of the matrix. In the last lines of the above demo program we have also specified the time interval $[0, 2]$ of the simulation and have defined the time variation of the control parameter, that describes how the confinement potential is transformed from a single well to a double well. With the chosen form the transformation occurs in the first half of the time interval, and the potential remains fixed in the second half of the interval.

We next compute the Gross-Pitaevskii ground state using the optimal damping algorithm of Dion et al. [42].

% groundstate wavefunction
psi0 = groundstate( grosspitaevskii( grid, ham, kappa ), lambda( 0 ) ); % plot ground state
plot( grid.x, psi0, 'b-', grid.x, 1e-3 * v( 0 ), 'r--' );

For double-well potentials the algorithm sometimes fails. In these cases we recommend to use a more robust but less efficient algorithm based on state mixing by calling groundstate(...,'mix',1e-2) instead. Finally, we set up an ODE solver and solve the Gross-Pitaevskii equation for the control field $\lambda(t)$ in time

op = struct( 'nsub', 20, 'nout', 10, 'stepfun', 'ode23' ); % options for ODE solver
[ psiout, tout ] = solve( psi0, tout, lambda, op ); % solve differential equation

% final output
imagesc( tout, grid.x, density( psiout ) );
xlabel( 'Time (ms)' );
ylabel( 'Position (\mu m)' );
title( 'Gross Pitaevskii' );

Figure 7: Density plot for time evolution of particle density for different splitting times. The left (right) panel reports results for a splitting time of 1 ms (2 ms). In the second half of the simulation the confinement potential is kept constant.
The density plot in the left panel of Fig. 7 shows the time evolution of the particle density. One sees that the condensate wavefunction splits and the density oscillates in the separated wells of the double-well potential. If we increase the time of the splitting process by a factor of two, the splitting becomes more adiabatic and the wavefunction oscillates less, as shown in the right panel of the figure.

The OCTBEC toolbox additionally allows for solutions of the Gross-Pitaevskii equation with the Crank-Nicolson or split operator techniques. Details of this approach can be found in Ref. [22] and in Appendix A. For the Crank-Nicolson technique we simply have to change the options for the ODE solver

% options for ODE solver using the Crank-Nicolson scheme
op = struct( 'nsub', 2, 'nout', 10, 'stepfun', 'crank' );

The Crank-Nicolson scheme has the advantage that the norm of the wavefunction is always preserved and that one can typically use significantly larger time steps than with the Runge-Kutta technique. This is particularly advantageous for OCT calculations where the Gross-Pitaevskii equation has to be solved many times. Note that we properly include the nonlinear term in our Crank-Nicolson approach, using a Newton iteration at each time step [28], as described in more detail in Appendix A.

To implement the split operator approach, we must equip the initial wavefunction with a kinetic energy operator evaluated in wavenumber space, and a confinement potential operator evaluated in real space.

% initialize split operator
psi0.t = @( lambda ) ( - 0.5 * grid.ilap / ( 87 * mass ) );
psi0.v = @( lambda ) ( v( lambda ) );
% options for ODE solver using the Crank-Nicolson scheme
op = struct( 'nsub', 2, 'nout', 10, 'stepfun', 'split' );

Again the norm of the wavefunction is always preserved and one can use significantly larger time steps in comparison to the Runge-Kutta scheme. To access the wavefunction and the particle density of a grosspitaevskii object psi, we can use the following commands.

orb = orbital( psi ); % or double( psi ), Gross-Pitaevskii wavefunction
den = density( psi ); % particle density for Gross-Pitaevskii wavefunction

Note that the above commands also work for cell arrays as returned from the ODE solvers.

5.2. Multi-configurational Hartree method for bosons (MCTDHB)

The multi-configurational Hartree method for bosons (MCTDHB) has been developed in recent years by Cederbaum, Alon, and coworkers [26]. The main idea is to provide several orbitals $\phi_i(r)$, which are determined in a self-consistent fashion, and to distribute the atoms among these orbitals. The total wavefunction for $m$ orbitals is then of the form

$$
\Psi = \sum_{i_1} \cdots \sum_{i_m} \sum_{\cdots} C_{i_1, i_2, \ldots, i_m} (a_{i_1}^\dagger)^{i_1} (a_{i_2}^\dagger)^{i_2} \cdots (a_{i_m}^\dagger)^{i_m} |\text{vac}\rangle,
$$

where $i_1 + i_2 + \cdots = N$ gives the total number of bosons and $C_{i_1, i_2, \ldots}$ characterizes the distribution of atoms between the orbitals. The dynamics of the orbitals $\phi_i(r, t)$ and the atom-number part $C$ of the wavefunction are obtained from a variational principle that chooses the orbitals in an “optimal way”, as discussed in some length in Ref. [26]. The working equation for the orbitals then becomes

$$
i\dot{\phi}_i = \mathcal{P} \left[ -\frac{\hbar^2 \nabla^2}{2M} + V(r, \lambda(t)) \right] \phi_i + \kappa \sum_{jklm} \rho^{-1}_{jmkl} \rho^{(2)}_{ijkl} \phi_j^* \phi_k \phi_l \phi_i,
$$

where $\rho$ and $\rho^{(2)}$ denote the one- and two-particle density matrix, respectively, and the projector $\mathcal{P} = 1 - \sum_i |\phi_i\rangle \langle \phi_i|$ assures that the ensuing term is orthogonal to the orbitals [26]. The atom number part
Figure 8: MCTDHB groundstate orbitals for (a) single and (b) double well potential. The dashed lines report the confinement potential. (c) Atom number part $C_{i,N-i}$ of wavefunction. For the single well only the orbital with gerade symmetry is significantly populated, whereas for the split trap both orbitals of gerade and ungerade symmetry become populated, thus indicating condensate fragmentation.

is governed by the Schrödinger equation $i\dot{C} = HC$, with the many-body Hamiltonian $H$, which has to be solved in parallel with the orbital part of Eq. (8).

Within the OCTBEC toolbox, we provide an implementation with $m$ orbitals, although we have primarily used and tested two orbitals. The MCTDHB wavefunction is initialized with

\[
\psi = \text{mctdhb}( \text{grid}, \text{ham}, \text{kappa}, m, n ); \quad \% \text{initialization of MCTDHB wavefunction}
\]

Here \text{grid} is the computational grid, \text{ham} is the single-particle Hamiltonian, \text{kappa} is the nonlinear atom-atom interaction, \text{m} is the number of orbitals, and \text{n} is the number of atoms. Upon initialization, a \text{mctdhb} object with the following properties is created:

\[
\begin{align*}
\text{orb} & \quad \text{orbital part of wavefunction}, \\
\text{num} & \quad \text{atom number part of wavefunction}.
\end{align*}
\]

Let us consider a simple example of condensate splitting by transforming the confinement potential from a single to a double well, as shown in the demo program \text{demomctdhb1.m}. We first set up a computational grid and define a single-particle Hamiltonian with a confinement potential that can be modified through a control parameter. These steps are very similar to the Gross-Pitaevskii simulations previously discussed.

\[
\begin{align*}
\text{units;} & \quad \% \text{length in micrometers, time in ms, hbar = 1} \\
\text{grid} = \text{gridid}( -3, 3, 101 ); & \quad \% \text{set up computational grid} \\
\text{v} = @( \text{lambda} ) ( \text{lesanovsky1d}( \text{grid.x}, \text{lambda} ) ); & \quad \% \text{confinement potential} \\
\text{ham} = @( \text{lambda} ) ( -0.5 * \text{grid.lap4} / ( 87 * \text{mass} ) + \text{spdiag}( \text{v}( \text{lambda} ) ) ); & \quad \% \text{single particle Hamiltonian} \\
\text{n} = 100; & \quad \% \text{number of particles} \\
\text{kappa} = 0.5 * 2 * \pi / ( \text{n} - 1 ); & \quad \% \text{nonlinearity (500 Hz)} \\
\text{m} = 2; & \quad \% \text{number of orbitals} \\
\text{psi0} = \text{groundstate}( \text{mctdhb}( \text{grid}, \text{ham}, \text{kappa}, \text{m}, \text{n} ), 0 ); & \quad \% \text{MCTDHB groundstate}
\end{align*}
\]

\[
\begin{align*}
\% \text{plot orbitals and C vector} \\
\text{subplot}( 1, 2, 1 ); & \quad \text{plot}( \text{grid.x}, \text{psi0.orb}, \text{grid.x}, 1e-2 * \text{v}( 0 ), 'r--' ); \\
\text{subplot}( 1, 2, 2 ); & \quad \text{plot}( \text{abs}( \text{psi0.num } ), 'o' );
\end{align*}
\]

In the last lines we plot the orbitals for the MCTDHB groundstate, which exhibit gerade and ungerade symmetry, as well as the distribution of atoms among these orbitals (see Fig. 8). For the single well practically all atoms reside in the orbital with gerade symmetry, and the occupation of the ungerade orbital is very small (corresponding to the situation of an undepleted condensate). The situation changes dramatically for a double well potential, as shown in Fig. 8.
Figure 9: Splitting process of MCTDHB wavefunction. Time evolution of (a) particle density, as well as of orbitals with (b) gerade and (c) ungerade symmetry. The atom number part \( C \) of the wavefunction (not shown) determines how the atoms become distributed between these orbitals.

\[
\begin{align*}
\text{% MCTDHB groundstate for initial double well potential} \\
\text{psi0 = groundstate( mctdhb( grid, ham, kappa, m, n ), 0.85 );}
\end{align*}
\]

Here a much larger fraction of the atoms populates the first excited orbital of ungerade symmetry, corresponding to condensate fragmentation. As previously discussed for the Gross-Pitaevskii groundstate, for further splitting one should use \texttt{psi0=groundstate(...'mix',1e-2)}.

We next compute the time evolution of a system that initially starts in the groundstate of a single-well potential, and the potential is subsequently transformed to a double well.

\[
\begin{align*}
\text{tout = linspace( 0, 3, 100 );} & \quad \% \text{ output times} \\
\text{nt = length( tout );} & \quad \% \text{ number of time steps} \\
\text{lambda = control( tout, sqrt( tout / max( tout ) ) );} & \quad \% \text{ control parameter} \\
\text{op = struct( 'nsub', 20, 'nout', 5, 'stepfun', 'ode23' );} & \quad \% \text{ options for ODE solver} \\
\text{[ psiout, tout ] = solve( psi0, tout, lambda, op );} & \quad \% \text{ solve MCTDHB equations} \\
\end{align*}
\]

\[
\begin{align*}
\text{% final output} \\
\text{imagesc( tout, grid.x, density( psiout ) );} \\
\text{xlabel( 'Time (ms)' );} \\
\text{ylabel( 'Position (\mu m)' );} \\
\text{title( 'MCTDHB(2)' );}
\end{align*}
\]

We can also plot the time evolution of the two orbitals, as shown in Fig. 9.

\[
\begin{align*}
\text{% plot time evolution of orbitals} \\
\text{subplot( 1, 2, 1 ); imagesc( tout, grid.x, abs( orbital( psiout, 1 ) ) );} \\
\text{subplot( 1, 2, 2 ); imagesc( tout, grid.x, abs( orbital( psiout, 2 ) ) );}
\end{align*}
\]

The number part of the MCTDHB wavefunction is often hard to interpret because atoms are distributed among time-varying orbitals. Plotting of the time dependence of \( C \) can be done through

\[
\begin{align*}
\text{plot( tout, abs( num( psiout{ 1 } ) ) );} & \quad \% \text{ plot modulus of C vector}
\end{align*}
\]

The \texttt{OCTBEC} toolbox additionally allows for solutions of the MCTDHB equation with the Crank-Nicolson technique. Details of this approach can be found in Ref. \[28\] as well as in \texttt{Appendix A}. For the Crank-Nicolson technique we simply have to change the options for the ODE solver

\[
\begin{align*}
\text{op = struct( 'nsub', 2, 'nout', 10, 'stepfun', 'crank' );}
\end{align*}
\]
Figure 10: (a) Schematics of two-mode model. The atoms reside in two orbitals (left and right), and are coupled by a tunneling element $\Omega$. By turning off the tunnel coupling, the atom number distribution becomes squeezed because of the nonlinear atom-atom interaction $\kappa$. (b) Time evolution of atom number distribution for an exponential decrease of $\Omega$.

The Crank-Nicolson scheme has the advantage that the norm of the wavefunction is always preserved and that one can typically use significantly larger time steps than with the Runge-Kutta technique. This is particularly advantageous for OCT calculations where the MCTDHB equations have to be solved many times. Finally, for the calculation of the true MCTDHB groundstate we refer to the help pages and to the demo program `demogroundstatemctdhb.m`.

5.3. Generic few-mode models

The OCTBEC toolbox provides several classes for the description of the atom number part, where atoms become distributed between static or time-dependent orbitals, and the time evolution is governed by a Hamiltonian matrix. The most general classes are `fewmodepair` and `fockstate`, which we will describe first. The classes `twomodepair` and `twomodespin` are more specialized and can be employed for the solution of a two-mode model, that has a longstanding history in the description of BECs [24, 25].

Consider a system of $n$ atoms that reside in $m$ orbitals. The wavefunction can be expanded in the atom number Hilbert space, the Fock space, according to Eq. (7). To set up the Fock space, we call

```matlab
spin = fewmodepair( n, m, cutoff ); % set up Fock space for n atoms and m orbitals
```

Here `cutoff` is an optional vector that determines the maximal number of atoms within a given orbital. For instance, we obtain

```matlab
spin = fewmodepair( 100, 2 ) % set up Fock space for 100 atoms in 2 orbitals
```

`fewmodepair`:

```matlab
n: 100
j: {2x2 cell}
state: [101x2 double]
```

where $j\{k,l\}$ denotes the pseudospin operator $J_{kl} = \hat{a}_k^\dagger \hat{a}_l$ and `state` is a matrix for all possible atom number configurations. The size of the Fock space is given by `size(spin.state,1)`.

We next consider the two-mode model discussed by Javanainen and Ivanov [25], where atoms reside in the left or right well of a double-well potential, subject to tunneling and nonlinear interactions, as shown in Fig. 10(a). In the figure the blue line represents the double-well potential and the red line the orbital part of the wavefunction. The Hamilton describing the dynamics of this two-mode model is of the form

$$H = -\frac{\Omega}{2} \left( \hat{a}_1^\dagger \hat{a}_2 + \hat{a}_2^\dagger \hat{a}_1 \right) + \kappa \left( \hat{a}_1^\dagger \hat{a}_1^\dagger \hat{a}_1 \hat{a}_1 + \hat{a}_2^\dagger \hat{a}_2^\dagger \hat{a}_2 \hat{a}_2 \right).$$

The first term describes tunneling between two wells, with the tunnel coupling $\Omega$, and $\kappa$ is the nonlinear atom-atom interaction. The demo program `demofewmodepair.m` shows the implementation of such a two-mode model.
Figure 11: Time evolution of number squeezing, through exponential turning-off of the tunnel coupling, as visualized on Bloch sphere. See demofewmodepairbloch.m for the corresponding simulation.

\begin{verbatim}
n = 100; % number of particles
kappa = 1 / n; % nonlinearity parameter
spin = fewmodepair( n, 2 ); % pseudospin object

% tunneling and nonlinear coupling
tun = - 0.5 * [ 0, 1; 1, 0 ];
non = kappa * accumarray( [ 1, 1, 1, 1; 2, 2, 2, 2 ], 1 );
% transform to operators
optun = spin.op( tun );
opnon = spin.op( non );

ham = @( lambda ) ( lambda * optun + opnon ); % Hamiltonian function
The operators, defined through the fewmodepair class, work together with fockstate objects that define a wavefunction within the Fock space. A fockstate object is initialized with

psi = fockstate( spin, ham ); % initialize FOCKSTATE object
Here spin is for instance the fewmodepair object defined above, which defines the Hilbert space in the atom number basis, and ham is a Hamiltonian acting in this Fock space. We assume that the Hamiltonian can be modified by some external control parameter. Upon initialization, the fockstate object has the following properties

  ham  Hamiltonian in atom-number basis,
  num  wavefunction in atom number space,
  spin pseudospin object.

The combination of the fewmodepair and fockstate classes allows to solve time-dependent problems in a quite elegant fashion. Consider the situation where the system is initially in the ground state governed by tunneling, and the tunneling is turned off exponentially at later times.

tout = linspace( 0, 80, 100 ); % output times
lambda = control( tout, 3 * exp( - tout / 10 ) ); % tunnel coupling
[ psi0, ene ] = groundstate( fockstate( spin, ham ), lambda( 0 ) ); % initial wavefunction

op = struct( 'nsub', 100, 'nout', 10, 'stepfun', 'runge4' ); % Runge-Kutta integration
[ psiout, tout ] = solve( psi0, tout, lambda, op ); % solve differential equation

% density map of atom number part
imagesc( tout, 0.5 * n * [ - 1, 1 ], abs( num( psiout ) ) );
xlabel( 'Time' );
ylabel( 'Atom number difference' );
\end{verbatim}

The simulation results is shown in Fig. 11(b). One sees that the atom number fluctuations around the mean value of \( n/2 \) are initially relatively large, corresponding to a binomial state, and become significantly reduced at later times owing to the nonlinear atom-atom interaction.

To use a Crank-Nicolson solution scheme, we have to replace the options for the ODE solver with
% options for Crank-Nicolson integration
op = struct( 'nsub', 10, 'nout', 10, 'stepfun', 'crank' );

When psiout is a cell array of fockstate objects, as usually returned from the ODE solver, the map of atom number wavefunctions can be obtained through

psi = num( psiout ); % wavefunction map from cell array PSIOUT

For a two-mode model the wavefunction in the atom number space can be conveniently plotted on the Bloch sphere, as discussed by Arecchi et al. [43]. Within the OCTBEC toolbox we set up the Bloch sphere with

% default initialization of Bloch sphere
sph = blochsphere( n );
% initialization of Bloch sphere with user-defined discretization
sph = blochsphere( n, nsph );

Here \( n \) is the number of atoms and \( nsph \) is an optional parameter (default value 200) that controls the discretization of the Bloch sphere. Through \( \text{plot}(\text{sph}, \text{num}(\text{psiout}{1})) \) we next plot the wavefunction on the Bloch sphere. Fig. [11] show the time evolution of the splitting process on the Bloch sphere, as computed with the demo program demoefuwmodepairbloch.m. Initially the system is in a binomial state (left) and number squeezing occurs at the end of the splitting process (right). The OCTBEC toolbox additionally provides a class twomodepair, that is very similar to the fewmodepair class for two orbitals, as well as a twomodespin class that uses pseudospin operators introduced e.g. by Milburn et al. [24]. Details can be found in the help pages and the demo files.

6. Optimal quantum control

6.1. Optimal control theory

The main purpose of the OCTBEC toolbox is to submit the different model systems to optimal control theory (OCT). In this section we provide the theoretical background for the OCT framework. A more detailed discussion can be found in Refs. [22] [29] [35]. In the following we first discuss optimization for a Gross-Pitaevskii wavefunction. The general OCT framework will be discussed at the end. For the working equations of the MCTDHB model, we refer the interested reader to Ref. [28] and Appendix B.

Consider the example demogp1.m for condensate splitting within the Gross-Pitaevskii framework, discussed in Sec. [51] where after splitting the wavefunction oscillates around the minima of the split trap (Fig. [7]). In what follows we are seeking for a time variation of the control parameter \( \lambda(t) \) such that the condensate wavefunction remains at rest after the splitting process. This task can be accomplished with optimal control theory. Let \( \psi_d(x) \) denote the desired groundstate wavefunction of the splitting process, which we can compute through

% desired wavefunction at terminal time
psid = groundstate( grosspitaevskii( grid, ham, kappa ), lambda.last, 'mix', 1e-2 );

OCT determines the optimal control \( \lambda_{opt}(t) \) such that the system is brought from the initial state \( \psi_0(x) \) to the desired state \( \psi_d(x) \) in an optimized fashion. First, we introduce a cost function \( J(\psi, \lambda) \) that measures the success of a given control \( \lambda(t) \), e.g., through

\[
J(\psi, \lambda) = \frac{1}{2} \| \psi(T) - \psi_d \|^2 + \frac{\gamma}{2} \int_0^T \left( \lambda(t) \right)^2 \, dt .
\] (10)

The first term on the right-hand side becomes minimal when the terminal wavefunction \( \psi(T) \) matches the desired state. The second term penalizes strong variations of the control parameter and is needed to make the OCT problem well posed. Through \( \gamma \) it is possible to weight the importance of wavefunction matching
Figure 12: Schematics for the OCT loop using the Gross-Pitaevskii equation. For some initial guess for the control parameter $\lambda(t)$, we (1) solve the Gross-Pitaevskii equation forwards in time. The terminal wavefunction $\psi(T)$ determines the value of the cost function, which becomes minimized within OCT, and allows to (2) compute the terminal condition for the adjoint variable $p(T)$, which is integrated backwards in time. From the knowledge of $\psi(t)$ and $p(t)$ we can compute the search direction $\delta L/\delta \lambda$ for the control parameter, and (3) come up with an improved guess for the control parameter. The loop is iterated until a given maximal number of iterations is achieved or the minimum of the cost function is reached.

and control smoothness, and in most cases we set $\gamma \ll 1$. The optimal control problem under consideration can now be written as

$$\min J(\psi, \lambda) \quad \text{subject to} \quad i\dot{\psi} = (H_\lambda + \kappa |\psi|^2) \psi, \quad \psi(0) = \psi_0. \quad (11)$$

Note that we have explicity indicated the $\lambda$-dependence of the single-particle Hamiltonian. The above equation states that we are looking for an optimal control that minimizes the cost function. But in order to bring the system from the initial state $\psi_0$ to the terminal state $\psi(T)$ we have to fulfill the Gross-Pitaevskii equation, which enters as a constraint in our optimization problem.

The constrained optimization problem can be turned into an unconstrained one by means of Lagrange multipliers $p(t)$. To this end, we introduce a Lagrange function

$$L(\psi, p, \lambda) = J(\psi, \lambda) + \text{Re} \left[ \int_0^T \left< p(t) \left| i\dot{\psi}(t) - (H_\lambda + \kappa |\psi(t)|^2) \psi(t) \right| \right> dt \right]. \quad (12)$$

The Lagrange function has a saddle point at the minimum of $J(\psi, \lambda)$ where all derivatives $\delta L/\delta \psi^$, $\delta L/\delta p^*$, and $\delta L/\delta \lambda$ become zero. Performing functional derivatives in the Lagrange function, we then arrive at the following set of equations

$$i\dot{\psi} = (H_\lambda + \kappa |\psi|^2) \psi, \quad \psi(0) = \psi_0 \quad (13)$$
$$i\dot{p} = (H_\lambda + 2\kappa |\psi|^2) p + \kappa \psi^2 p^*, \quad ip(T) = \psi(T) - \psi_d \quad (14)$$
$$\gamma \ddot{\lambda} = -\text{Re} \left< p \left| \frac{\partial H_\lambda}{\partial \lambda} \right| \psi \right>, \quad \lambda(0) = \lambda_0, \quad \lambda(T) = \lambda_1. \quad (15)$$

Equation (13) is the initial value problem of the Gross-Pitaevskii equation, whereas Eq. (14) is a terminal value problem for the adjoint variable $p$. Finally, Eq. (15) determines the optimal control and is a boundary value problem where both the initial and terminal value are fixed.

In most cases of interest it is impossible to guess an optimal control such that all equations (13-15) are fulfilled simultaneously. However, the above set of equations can be also used for an iterative procedure. Suppose that we start with some reasonable guess for $\lambda(t)$. We can then solve Eq. (13) forwards in time,
determine the terminal condition $p(T)$, and solve Eq. (14) backwards in time. For a non-optimal control Eq. (15) is no longer fulfilled. However,

$$
\frac{\delta L}{\delta \lambda} = -\gamma \ddot{\lambda} - \text{Re}\langle p| \frac{\partial H}{\partial \lambda} |\psi\rangle.
$$

(16)

then provides us with a search direction for an optimized control. Upon searching in the direction $\delta L/\delta \lambda$ we obtain a better control, that brings the terminal $\Psi(T)$ closer to the desired $\psi_d$, and the optimization can be repeated until we reach the minimum of the optimal control. Figure [12] schematically depicts the OCT loop, which continues until a given maximal number of iterations is achieved or the minimum of the cost function is reached. Quite generally, for the optimization we either use a nonlinear conjugate gradient method or a quasi-Newton BFGS scheme.

The cost function of Eq. (10) for wavefunction trapping is overly restrictive as it requires that the final wavefunction matches the desired one up to the global phase, which is of no relevance. A better choice $J(\psi) = \frac{1}{2} \left[ 1 - |\langle \psi_d |\psi(T)\rangle|^2 \right]$ for the cost function has been formulated in Ref. [22] and is also discussed in the help pages. In general, the choice of the cost function is determined by the problem under study. Cost function and terminal condition for the adjoint variable are connected through

$$
\frac{p(T)}{\delta \psi(0)} = -2\frac{\delta J(\psi, \lambda)}{\delta \psi(0)}.
$$

(17)

Within the OCTBEC toolbox the terminal condition must be computed by the user through functional derivatives. However, even if one feels uneasy with functional derivatives this step is usually not overly difficult. In Sec. 7.1 we will present a test scheme for figuring out whether the functional derivative and the implementation have been performed properly.

6.1.1. $L^2$ versus $H^1$ norm

In some cases, there is a problem related to the numerical implementation of Eq. (15). It is a boundary value problem, where the conditions $\lambda(0) = \lambda_0$ and $\lambda(T) = \lambda_1$ hold. For the initial guess one often ends up in a state that is quite far away from the desired solution. In turn, the terminal value for $p(T)$ is large and so is the gradient $\delta L/\delta \lambda$ of Eq. (16). This has the consequence that the largest variation of the control field is initial at the terminal time $T$, while from a control perspective it would often be better to change the control at early times first. Only with increasing number of iterations in the minimization procedure the control becomes modified within the whole time interval. This can lead to a tedious and time-consuming optimization process. A more convenient approach was formulated by von Winckel and Borzì [37]. The main idea is to use a different norm in the integrals for the cost as well as in the Lagrange function. In the cost functions the penalization for the control field $(\gamma/2) \left( \dot{\lambda}, \dot{\lambda} \right)_{L^2}$ can be reformulated as $(\gamma/2) \left( \lambda, \dot{\lambda} \right)_{H^1}$, where the definition of the $H^1$ inner product is $(u,v)_{H^1} = (\dot{u}, \dot{v})_{L^2}$. It is important to realize that this different norm does neither affect the value of the cost function nor the principal or sensitivity Eqs. (13,14). However, it does affect the equation for the control field, which now satisfies a Poisson equation

$$
-\frac{d^2}{dt^2} \left[ \frac{\delta L}{\delta \lambda} \right] = -\gamma \ddot{\lambda} - \text{Re}\langle p| \frac{\partial H}{\partial \lambda} |\psi\rangle.
$$

(18)

The advantage of this equation is that changes due to large values of the second term on the right-hand side are distributed, through the solution of the Poisson equation, over the whole time interval. Most of our optimizations show that the $H^1$ optimization is faster and more robust than the $L^2$ optimization, the resulting control fields are significantly smoother, and the global structure of $\lambda(t)$ is optimized.

6.1.2. General OCT problem

A general OCT problem is governed by a set of differential equations, which we might write as

$$
\dot{\psi}(t) = f(\psi(t), \lambda(t)), \quad \psi(0) = \psi_0,
$$

(19)
where $\psi(t)$ are the dynamic variables, such as wavefunctions or orbitals, and $f(\psi(t), \lambda(t))$ is a general functional that determines the time evolution of $\psi(t)$. The cost function $J(\psi, \lambda)$ can then depend on the terminal value $\psi(T)$ as well as on the intermediate values $\psi(t)$. We assume that one can separate the cost function into a terminal and intermediate part

$$J(\psi, \lambda) = J_{\text{final}}(\psi(T), \lambda_1) + J_{\text{inter}}(\psi(t), \lambda(t)) + \gamma \int_0^T \left( \dot{\lambda}(t) \right)^2 \, dt.$$  \hspace{5em} (20)

In practically all our OCT calculations used in the past and given below we set the intermediate cost to zero. However, for completeness and for future use we have implemented the general scheme, consisting of:

- the final cost function $J_{\text{final}}(\psi(T), \lambda_1)$,
- the terminal conditions $p(T) = -2i[\delta J_{\text{final}}/\delta \psi^*(T)]$ for the adjoint variables,
- the intermediate cost function $J_{\text{inter}}(\psi, \lambda)$,
- the functional derivative $f(t) = 2[\delta J_{\text{inter}}/\delta \psi^*(t)]$,
- the functional derivative $[\delta J_{\text{inter}}/\delta \lambda(t)]$.

Table 4 summarizes the basic ingredients of our OCT implementation.

6.2. Cost function

We next discuss how to implement the cost function within the OCTBEC toolbox. To this end, we have to extend the initialization of the control function (see Sec. 4.3) according to

```matlab
gamma = 1e-2; % control penalization
lambda = control( ttab, lamtab, gamma, 'H1' ); % initialize control parameter
```

Here `ttab` and `lamtab` are the tabulated control parameters for the initial guess, `gamma` is the control penalization, and the last parameter is either `’H2’` or `’L2’` and determines the norm for the inner products. Once this initialization is performed, the `control` object is ready for use within OCT simulations. In addition to the cost for the control parameter, we also have to specify the cost e.g. for wavefunction trapping. Suppose that `psid` is Gross-Pitaevskii object for the desired state. We then define a `costfunction` object

```matlab
% cost function for state trapping
valfin = @( psi, lambda ) ( 0.5 * ( 1 - abs( grid.inner( psid.val, psi.val ) ) ^ 2 ) );
% terminal condition for state trapping
final = @( psi, lambda ) ( 1i * grid.inner( psid.val, psi.val ) * psid );
% cost function object
cost = costfunction( valfin, final );
```

`valfin` gives the cost function for the state trapping, and `final` returns a `grosspitaevskii` object that holds the terminal conditions for the adjoint variable. In case of an additional intermediate cost, one additionally has to provide the functions `valint`, `inter`, and `deriv` for the intermediate cost, as well as the derivatives of the intermediate cost with respect to the dynamic variables and the control parameters. The complete cost function definition is of the form

```matlab
% initialize cost function with terminal and intermediate cost
cost = costfunction( valfin, final, valint, inter, deriv );
```

`costfunction` objects can be multiplied with constant factors, and can be added together. In the latter case the OCT optimization will come up with control parameters that bridge between the different control objectives.

6.3. Optimality system

The OCTBEC toolbox uses an `optimality` object to compute the forward and backward equations, and an `optimize` object to perform the OCT loop. For the model systems considered in the toolbox we set up the optimality system, e.g., with
Table 4: Classes and class methods for the implementation of optimal control within the framework of the OCTBEC toolbox. The *optimality* object solves the optimality Eqs. (13–18), and returns either the value of the cost function or a new search direction. The *optimize* object is a generic optimization class using nonlinear conjugate gradient or quasi-Newton BFGS techniques, assuming $L^2$ or $H^1$ norm for the inner products. Detailed information about the classes can be obtained through `doc @classname`.

| Object          | Matlab implementation | Description                                                                 |
|-----------------|-----------------------|------------------------------------------------------------------------------|
| Model           | deriv(psi,lambda,op)  | Derivative function for forward Eq. (13)                                     |
|                 | crank(psi,lambda,dt,op)| Crank-Nicolson step for forward Eq. (13)                                    |
|                 | optderiv(psi,p,lambda,finiter,op) | Derivative function for adjoint Eq. (14)                                  |
|                 | optcrank(psi,p,lambda,finiter,dt,op) | Crank-Nicolson step for adjoint Eq. (14)                                  |
| control         | control(t,lambda,penalty,flag) | Control function initialization                                               |
|                 | cost(lambda)          | Cost ($\gamma/2)(\lambda,\lambda)_H^1$ for control parameter               |
|                 | dlambda=deriv(lambda,f) | Solve Eq. (16) or Eq. (18)                                                  |
|                 | ip=inner(u,v)         | Inner product with $L^2$ or $H^1$ norm                                       |
|                 | norm(lambda)          | Norm of control parameter                                                   |
|                 | +,-,*,/               | Basic arithmetic operations for one or two control parameters                |
| controltype     | lam=lambda(t)         | Return value of control parameter object                                    |
|                 | lam or lam(;)         | Control values at time t                                                     |
|                 | lam.t                 | Time argument                                                               |
|                 | lam.lambda            | Reference to control parameter object                                        |
| costfunction    | costfunction(valfin,final,valint,inter,deriv) | Initialization of costfunction object                                      |
|                 | deriv(cost,psi,p,lambda) | Derivative of cost function wrt \( \lambda \)                              |
|                 | p=final(cost,psi,lambda) | Terminal condition for adjoint variable                                     |
|                 | inter(cost,psi,lambda) | Inhomogeneity for adjoint equation due to intermediate cost                  |
|                 | valfin(cost,psi,lambda) | Cost value at final time                                                     |
|                 | valint(cost,tout,psiout,lambda) | Cost value at intermediate times                                           |
|                 | *,+,-,*,/             | Basic arithmetic operations for cost functions                              |
| optimality       | optimality(psi0,tout,lambda,costfun,op) | Initialization of optimality system                                       |
|                 | [val,fin]=cost(opt)   | Cost value and terminal condition for adjoint variable                       |
|                 | opt=solve(opt)        | Solve forward Eq. (13)                                                      |
|                 | dlambda=isolve(opt)   | Compute search direction for control parameter                              |
| optimize         | optimize(opt,op)      | Initialize OCT object                                                        |
|                 | [oct,lambda,psiout]=improve(obj,niter) | Perform OCT loop                                                            |
|                 | info(oct)             | Print statistics for OCT loop                                                |

23
op1 = struct( 'nsub', 2, 'nout', 0, 'stepfun', 'crank' ); % options for optimality system
opt = optimality( psi0, tout, lambda, cost, op1, 1 ); % set up optimality system

The option arguments are for the solution of the dynamic equations and are given in Sec. 4.4. Additional op arguments will be passed to the derivative functions, e.g. to control the tolerance for Newton iterations. In the initialization of the optimality object we provide the initial value for the dynamic variable, the time arguments, the control parameter, the cost function, and the options for the ODE solution. The last argument will give an intermediate plotting of the control parameters during the OCT loop, and can be omitted or replaced by another function or function handle. In most cases the optimality object is only used internally (see table 4 for details).

Finally, the OCT loop is performed with an optimize object. It is initialized through

op2 = struct( 'mode', 'BFGS', 'tol', 1e-4 ); % options for minimization
oct = optimize( opt, op2 ); % set up optimal control object

The following options can be passed to the optimize object:
- mode 'grad' for nonlinear conjugate gradient and 'BFGS' for quasi-Newton optimization,
- tol tolerance for the termination of the OCT loop,
- bounds [lmin,lmax] for lower and upper bounds for the control parameter.

In general, grad performs significantly faster than BFGS but usually gives slightly worse control fields. We recommend to use BFGS whenever possible. Bounds can be provided for BFGS simulations in order to avoid too large control fields, for which the ODE integration may fail. In general we recommend to use not too tight bounds. If a third value [lmin,lmax,dlmax] is provided, the control parameters are also restricted to |λ| < dlmax. Without bounds the linesearch algorithm fminunc is used, and fmincon otherwise. Additional options to the optimize object are passed to these functions.

The OCT loop is then performed through

[ oct, lambda, psiout ] = oct.improve( 10 ); % perform OCT iterations

In this case either 10 OCT iterations are performed, or the OCT loop terminates when \( \|\delta L/\delta \lambda\| \) becomes smaller than tol times its initial value. Upon termination of the OCT loop, we obtain
- oct updated OCT object,
- lambda optimized control parameter,
- psiout history for dynamic variables as computed by the ODE solver.

If one wants to continue with the optimization, one can call again oct.improve(niter) to further improve the optimal control. Finally, some information about the optimization is available through

>> oct.info; % give information about OCT loop

Optimal control loop:

Number of iterations: 10
Search algorithm: BFGS

Total elapsed time: 43.426825
Mean time for forward solution: 0.240128
Mean time for backward solution: 1.179120

Time percentage for forward solution: 68.566
Time percentage for backward solution: 27.152
Time percentage for rest: 4.282

Number of forward solutions: 124
Number of backward solutions: 10
7. Optimization of model systems

7.1. Optimization of the Gross-Pitaevskii equation

The demo program *demogpoct1.m* provides an example for the optimization of a splitting process within the framework of optimal control theory. The basic steps are:

1. set up computational grid and Hamiltonian, and compute initial wavefunction,
2. define time interval and initial guess for control parameter,
3. define cost properties for control (penalization and \( L^2 \) or \( H^1 \) norm) and cost function,
4. set up options for solution of Gross-Pitaevskii equation,
5. submit problem to OCT minimization.

Steps 1.2 and 4 are similar to the solution of the Gross-Pitaevskii equation, and step 5 in general involves just a few program lines. The definition of the cost function and its derivative of step 3 provide the main challenge of OCT simulations. Typically the implementation of step 3 is relatively simple, but has to be done with care. In our demo program we first set up the computational grid and Hamiltonian, and define the nonlinearity.

```matlab
units; % internal units
grid = grid1d( - 3, 3, 101 ); % simulation grid
v = @( lambda ) ( lesanovsky1d( grid.x, lambda ) ); % confinement potential
ham = @( lambda ) ( - 0.5 * grid.lap4 / ( 87 * mass ) + spdiag( v( lambda ) ) ); % single particle Hamiltonian
kappa = pi; % nonlinearity
```

The above steps have been discussed in length in the Sec. 4 and 5.1. We next define the time interval of the optimization and define an initial guess for the control parameter.

```matlab
tmax = 1.2; % maximal time of simulation
tout = linspace( 0, tmax, 100 ); % output times

gamma = 1e-2; % penalization of control field
lambda = sqrt( tout / max( tout ) ); % control parameter
lambda = control( tout, lambda, gamma, 'H1' );
```

In the last line we set up a control object using the initial guess for the control parameter. The last two parameters are gamma for the penalization of the control parameter, and 'L2' or 'H1' for the norm used in the OCT simulations (we recommend to always use 'H1'). Next, we compute the Gross-Pitaevskii groundstate and the desired OCT state, within which the system should end up after the splitting process. Here we define the desired state as the groundstate of the split trap. With the ground and desired state we can now set up a costfunction object.

```matlab
% groundstate wavefunction
psi0 = groundstate( grosspitaevskii( grid, ham, kappa ), lambda.first );
% desired wavefunction at terminal time
psid = groundstate( grosspitaevskii( grid, ham, kappa ), lambda.last, 'mix', 1e-2 );

% cost function for state trapping
value = @( psi, lambda ) ( 0.5 * ( 1 - abs( grid.inner( psid.val, psi.val ) )^2 ) );
% terminal condition for adjoint variable
final = @( psi, lambda ) ( 1i * grid.inner( psid.val, psi.val ) * psid );
% initialize cost function object
cost = costfunction( value, final );
```

We next define the options for the ODE solver (see Sec. 4.4) and define the optimality system.
In the help pages we show how the optimality object can be used to compute the search direction for the optimal control. Finally, the optimal control loop is performed with the following commands.

\[
\text{op2} = \text{struct( 'mode', 'BFGS', 'tol', 1e-4 );} \quad \% \text{options for OCT minimization}
\]
\[
\text{oct} = \text{optimize( opt, op2 );} \quad \% \text{set up optimal control object}
\]
\[
[\text{oct}, \lambda, \psiout] = \text{oct.improve(10);} \quad \% \text{perform OCT loop}
\]

The first command defines the options for the OCT minimization. We use a quasi-Newton scheme and use a tolerance of \( \text{tol}=1e-4 \) for the termination of the OCT loop. The call \( \text{oct.improve(10)} \) performs the OCT optimization, which ends after 10 iterations or when the tolerance is reached. Further iterations can be performed by either increasing the number of iterations (e.g. from 10 to 100) or by calling again the \( \text{oct.improve} \) function. As the BFGS algorithm uses the previous function calls in order to estimate the Hessian of the control space, the two approaches will give slightly different results. If possible, we recommend to increase the number of iterations. Results of the OCT simulation have already been shown in the introductory Sec. 3.2 and in Fig. 3.

The functional derivative of the cost function must be implemented properly. Otherwise the OCT algorithm will not terminate properly, usually because the cost function does not decrease along the search direction, or run forever. It is usually a good idea to make consistency checks, as shown in the demo program \( \text{demogpocttest.m} \). The idea is as follows. Suppose that \( \lambda(t) \) is a control field and \( u(t) \) some smooth function with the boundary conditions \( u(0)=u(T)=0 \). Then, for a small parameter \( \eta \) the following relation holds

\[
\frac{1}{\eta} \left[ J(\psi, \lambda + \eta u) - J(\psi, \lambda) \right] = \left\langle u(t), \frac{\delta L(\psi, p, \lambda)}{\delta \lambda(t)} \right\rangle_{L^2 \text{ or } H^1} + O(\eta), \tag{21}
\]

which can be used to test the OCT implementation.

In general the two results will somewhat differ because of numerical rounding errors. However, significant differences between these two computation approaches usually indicate improper OCT implementations.

% small quantity
eta = 1e-6;
% variation of control parameter
dlambda = control( tout, sin( tout / max( tout ) * 6 * pi ), 1e-10, 'L2' );
% options for ODE solver
op = struct( 'nsub', 2, 'nout', 0, 'stepfun', 'crank' );
% set up optimality systems
opt1 = optimality( psi0, tout, lambda, cost, op );
opt2 = optimality( psi0, tout, lambda + eta * dlambda, cost, op );
% solve Gross-Pitaevskii equation and compute cost function
opt1 = solve( opt1 );  \( j1 = \text{opt1.cost();} \)
opt2 = solve( opt2 );  \( j2 = \text{opt2.cost();} \)

deriv = isolve( opt1 ); \% OCT derivative
dj = inner( dlambda, deriv ); \% cost difference from OCT derivative

\[
\text{Direct: } -0.35799 \\
\text{OCT: } -0.35832
\]
Figure 13: Optimization of number squeezing for two-mode model and Hamiltonian of Eq. (9). (a) Density fluctuation in units of $\sqrt{n}/2$ for initial guess (dashed line) and optimized control parameter (solid line), as computed with demo file demofewmodepairoct.m. Panel (b) reports the corresponding control parameters.

which must be corrected. We recommend to always employ this testing approach before running the OCT loop.

7.2. Optimization of the multi-configurational Hartree method for bosons (MCTDHB)

Optimization of control fields within the MCTDHB framework is very similar to the optimization within the Gross-Pitaevskii framework, although the underlying equations are more difficult to solve. In Appendix B we provide some details about the working equations. The choice of the cost function within MCTDHB is less obvious, in demomctdhb1.m we provide an example for orbital trapping and in demomctdhb2.m for energy minimization of the terminal states. See also Refs. [27–32] for further examples. The programs perform significantly slower than for Gross-Pitaevskii OCT simulations, with typical runtimes listed in table 1.

7.3. Optimization of generic few-mode models

We finally briefly comment on OCT optimization for the atom number part of the wavefunction. In Sec. 5.3 we have discussed the demo file demofewmodepair.m where within a two-mode model atom number squeezing is achieved by adiabatically turning off the tunneling coupling. In demofewmodepairoct.m we present an OCT optimization for such number squeezing, as also discussed in Ref. [27, 28]. Our goal is to minimize in the terminal state atom-number fluctuations between the left and right well. The atom number imbalance between the two wells is measured by the pseudospin operator

$$ \hat{J}_z = \frac{1}{2} (\hat{a}_1^\dagger \hat{a}_1 - \hat{a}_2^\dagger \hat{a}_2) .$$

The control objective is to minimize atom number fluctuations in the terminal state, once tunneling has been switched off. To this end, we introduce the cost function

$$ J(C) = \langle \hat{J}_z^2 \rangle - \left( \langle \hat{J}_z \rangle \right)^2 = \langle C | \hat{J}_z^2 | C \rangle .$$

Here $C$ is the atom-number wavefunction, and to arrive at the last expression we have used $\langle \hat{J}_z \rangle = 0$. Taking the functional derivative of the cost function with respect to $C^\dagger$, we arrive for the adjoint variable $D$ at the terminal condition $D(T) = -2i\hat{J}_z^2 C(T)$. Within our program, we implement the cost function object in the general way

```matlab
jz = spin.op( 0.5 * [ 1, 0; 0, -1 ] ); % pseudospin operator
% cost function for squeezing (use REAL to avoid rounding errors)
value = @( psi, lambda ) real( psi.num' * ( jz * 2 ) * psi.num - ( psi.num' * jz * psi.num ) * 2 );
```

27
\[
\text{final} = @(\ psi, \ lambda) \ ( -2i \ast ( jz \ast 2 - 2 \ast (\ psi\cdot\num' \ast jz \ast psi\cdot\num) ) \ast psi);
\]

Division with \(n/4\) ensures that we measure the (square of the) fluctuations with respect to the fluctuations of a binomial state. We next set up the optimality system and the optimal control object.

\[
op = \text{struct}( '\text{nsub}', 2, '\text{nout}', 0, '\text{stepfun}', '\text{crank}' ); \quad \% \text{options for ODE solver}
\]

\[
\text{opt} = \text{optimality}( psi0, \text{tout}, \lambda, \text{cost}, \text{op}, 1 ); \quad \% \text{set up optimality system}
\]

\[
op = \text{struct}( '\text{mode}', '\text{BFGS}', '\text{tol}', 1e-6 ); \quad \% \text{options for OCT minimization}
\]

\[
\text{oct} = \text{optimize}( \text{opt}, \text{op} ); \quad \% \text{set up optimal control object}
\]

\[
[ \text{oct}, \lambda, \psi_{\text{out}} ] = \text{oct}.\text{improve}(10); \quad \% \text{perform OCT iterations}
\]

Figure 13 shows the time evolution of the atom-number fluctuations measured in units of the fluctuations of a binomial state. As regarding the interpretation of the control strategy we refer to Ref. [28]. It is also instructive to visualize the state evolution on the Bloch sphere, as can be done by uncommenting the last lines in \text{demogoctfewmodepair.m}.

Acknowledgment

I am grateful to Jörg Schmiedmayer, Julian Grond, Georg Jäger, and Robert Schützky for most helpful discussions. Alfio Borzì and Greg von Winckel are cordially acknowledged for their contributions to the OCT implementation. This work has been supported in part by the Austrian science fund FWF under project P24248.

Appendix A. Crank-Nicolson scheme

In this appendix we provide some details for the solution of Schrödinger-type equations with the Crank-Nicolson scheme. We will derive the pertinent expressions needed for the solution of the Gross-Pitaevskii equation, the MCTDHB equations, and the fewmode model.

Appendix A.1. Schrödinger equation

For the sake of completeness, we briefly review the solution of the Schrödinger equation with the Crank-Nicolson scheme. Let \(\psi^n\) denote the wavefunction at time \(t_n\). The time derivative is approximated by the time-discretized version (\(\hbar = 1\))

\[
\frac{i}{\Delta t} (\psi^{n+1} - \psi^n) = H\frac{\psi^n + \psi^{n+1}}{2},
\]

where \(H\) is a sparse matrix consisting of the kinetic and potential term evaluated at the mid-time \(t_n + t_{n+1}/2\).

Note that for simplicity we have not indicated the space dependence of the wavefunction and the Hamiltonian matrix. In the solution of Eq. (A.1) we know the wavefunction \(\psi^n\) at the initial time and are seeking for the wavefunction \(\psi^{n+1}\) at the end of the time step. Solving Eq. (A.1) for the unknown then gives the final expression

\[
\psi^{n+1} = \left( I + \frac{i}{2\Delta t} H \right)^{-1} \left( I - \frac{i}{2\Delta t} H \right) \psi^n.
\]

The solution of Eq. (A.2) is simplified by the sparseness of the Hamiltonian matrix which allows for a fast inversion through a LU decomposition [44].
Appendix A.2. Gross-Pitaevskii

Submitting the Gross-Pitaevskii equation \( \psi \) to the Crank-Nicolson discretization, we obtain

\[
\frac{i}{\Delta t} (\psi^{n+1} - \psi^n) = \frac{H \psi^n + \psi^{n+1}}{2} + \kappa \left| \psi^n + \psi^{n+1} \right| \psi^n + \psi^{n+1}. \tag{A.3}
\]

Due to the atom-atom interaction the equation becomes nonlinear and cannot be solved straightforwardly. We here follow the approach presented in Ref. [28] which solves Eq. (A.3) iteratively through a Newton iteration scheme. In absence of the nonlinearity we first use Eq. (A.2) to obtain an approximate solution \( \psi_0^{n+1} \). To obtain the true solution \( \psi^{n+1} \equiv \psi_0^{n+1} + \delta \psi \), we next linearize Eq. (A.3) with respect to \( \delta \psi \),

\[
\frac{i}{\Delta t} (\psi_+ + \delta \psi) = H \left( \psi_+ + \frac{\delta \psi}{2} \right) + \kappa |\psi_+|^2 \left( \psi_+ + \delta \psi \right) + \kappa \psi_+^\ast \frac{\delta \psi^\ast}{2}, \tag{A.4}
\]

where we have introduced the abbreviations \( \psi_\pm = \psi_0^{n+1} \pm \psi^n \). Separating the expressions with \( \delta \psi \) from the rest, we get

\[
\left( \mathbb{I} + \frac{\Delta t}{2} \left[ H + 2\kappa |\psi_+|^2 \right] \right) \delta \psi + i \frac{\Delta t}{2} \kappa \psi_+^2 \delta \psi^\ast = -\psi_\mp - i \Delta t \left( H + \kappa |\psi_+|^2 \right) \psi_+. \tag{A.5}
\]

This equation is of the form \( Ax + Bx^\ast = b \), where \( A \) and \( B \) are matrices, \( x \) is the solution vector, and \( b \) an inhomogeneity. Together with the complex conjugate of the equation we obtain the matrix equation

\[
\begin{pmatrix} A & B \\ A^\ast & B^\ast \end{pmatrix} \begin{pmatrix} x \\ x^\ast \end{pmatrix} = \begin{pmatrix} b \\ b^\ast \end{pmatrix}, \tag{A.6}
\]

which can be solved by inversion and keeping only the \( x \) part of the solution vector. Due to the sparseness of the matrices \( A \) and \( B \) the inversion is again fast and efficient.

In our computational approach, we solve Eq. (A.5) iteratively, by replacing in a second step the initial guess with \( \psi_0^{n+1} + \delta \psi \) and subsequent solution of Eq. (A.5), and repeat this procedure until convergence. With a tolerance \( \text{tol}=1e-6 \) for the termination of the Newton iteration loop, we typically need two to three iterations.

Appendix A.3. Schrödinger-type equation with inner products

Let us consider the Schrödinger equation

\[
\frac{i}{\Delta t} \psi(t) = (H - \langle \psi|H|\psi \rangle) \psi, \tag{A.7}
\]

which e.g. arises for the atom-number part of the wavefunction. Submitting the equation to the Crank-Nicolson scheme, we get

\[
\frac{i}{\Delta t} \left( \psi_\mp + \delta \psi \right) = \left( H - \langle \psi_\pm|H|\psi_\pm \rangle \right) \left( \psi_\mp + \frac{\delta \psi}{2} \right) - \frac{1}{2} \left( \langle \psi_\mp|H|\delta \psi \rangle + \langle \delta \psi|H|\psi_\mp \rangle \right) \psi_\mp. \tag{A.8}
\]

\( \psi_0 \) is an approximate solution at the end of the time step, which is e.g. obtained by solving Eq. (A.7) with the subtraction term \( \langle \psi^n|H|\psi^n \rangle \), and we have again introduced the abbreviations \( \psi_\pm = \psi_0^{n+1} \pm \psi^n \) as well as \( \delta \psi \) for the difference between \( \psi_0 \) and the true solution.

Eq. (A.8) is of the form

\[
Ax + Bx^\ast + (V_1^T x)U_1 + (V_2^T x^\ast)U_2 = b, \tag{A.9}
\]

where \( A \) and \( B \) are matrices (matrix \( B \) has been added for completeness), and \( U_k \) and \( V_k \) are vectors. Together with the complex conjugate of the above equation we get

\[
\begin{pmatrix} A & B \\ A^\ast & B^\ast \end{pmatrix} \begin{pmatrix} x \\ x^\ast \end{pmatrix} + (V_1^T x) \begin{pmatrix} U_1 \\ 0 \end{pmatrix} + (V_1^T x)^\ast \begin{pmatrix} 0 \\ U_1^\ast \end{pmatrix} + (V_2^T x^\ast) \begin{pmatrix} U_2 \\ 0 \end{pmatrix} + (V_2^T x)^\ast \begin{pmatrix} 0 \\ U_2^\ast \end{pmatrix} = \begin{pmatrix} b \\ b^\ast \end{pmatrix}. \tag{A.10}
\]
which can be written in the compact form
\[
\tilde{A}\tilde{x} + \sum_k \left(\tilde{V}_k^T \tilde{x}\right) \tilde{U}_k = \left(\tilde{A} + \tilde{U}\tilde{V}^T\right) \tilde{x} = \tilde{b}.
\] (A.11)

Here \(\tilde{U} = (\tilde{U}_1, \tilde{U}_2, \ldots)\) and \(\tilde{V} = (\tilde{V}_1, \tilde{V}_2, \ldots)\) are matrices consisting of the vectors \(\tilde{U}_k\) and \(\tilde{V}_k\). The matrix \(\tilde{U}\tilde{V}^T\) is of low rank as the two matrices \(\tilde{U}\) and \(\tilde{V}\) only consists of a few column vectors. One can thus employ the Sherman–Morrison–Woodbury formula for the inversion
\[
\left(\tilde{A} + \tilde{U}\tilde{V}^T\right)^{-1} = \tilde{A}^{-1} - \left[\tilde{A}^{-1}\tilde{U}\right] \left[1 + \tilde{V}^T \left[\tilde{A}^{-1}\tilde{U}\right]\right]^{-1} \tilde{V}^T \tilde{A}^{-1}.
\] (A.12)

When applying this matrix to \(\tilde{x}\), we have to solve \(\tilde{A}^{-1}\tilde{x}\) and \(\tilde{A}^{-1}\tilde{U}\) which is typically efficient because of the sparseness of the matrix \(\tilde{A}\). The inversion of the matrix on the right-hand side of Eq. (A.12) is also fast because of the small number of \(\tilde{U}_k\) and \(\tilde{V}_k\) vectors involved. For these reasons, the solution of \(\left(\tilde{A} + \tilde{U}\tilde{V}^T\right)^{-1}\tilde{x}\) can be very fast and efficient. As for the Schrödinger equation (A.7), we then solve Eq. (A.8) iteratively with a Newton iteration scheme.

**Appendix A.4. Multi-configurational Hartree method for bosons**

The Sherman–Morrison–Woodbury formula can be also employed in the Crank-Nicolson solution of the MCTDHB equations, which consist of one equation for the atom number part \(C\) and another set of equations for the orbitals \(\phi\). As for the atom number part, we obtain in correspondence to Eq. (A.8) the expression
\[
\frac{i}{\Delta t} (C_+ + \delta C) = (H - \langle C_+ |H|C_+ \rangle) \left(C_+ + \frac{\delta C}{2}\right) - \frac{1}{2} \left(\langle C_+ |H|\delta C \rangle + \langle \delta C |H|C_+ \rangle\right)C_+,
\] (A.13)

with the shorthand notations \(C_\pm = C_0^{n+1} \pm C^n\) and \(\delta C = C_0^{n+1} - C_0^n\), where \(C_0\) is some initial guess. Quite generally, the Hamiltonian \(H\) depends on the orbitals \(\phi_k^{n+1}\) which are not known at the first iteration. In our computational approach we do not consider this dependence explicitly, but solve Eq. (A.13) in parallel with Eq. (A.15), to be derived below. As the dependence of \(H\) on the orbitals is generally rather weak (as well as the dependence of the density matrices on \(C\)), one can expect that the iterative solution of the coupled atom and orbital equations leads to convergence even without explicit consideration of these dependencies. Indeed, in all situations considered so far we found convergence after a few iterations.

As for the orbital part, we have to solve
\[
i \frac{d\phi_i}{dt} = \mathcal{P} \left[\hbar \phi_i + \kappa \sum_{jkl} r_{ijkl} \phi_j^* \phi_k \phi_l\right] \equiv \mathcal{P} f_i, \quad r_{ijkl} = \sum_a \rho_{aa}^{-1} \rho_{ijkl}^{(2)}.
\] (A.14)

Here \(\mathcal{P}\) is the projection operator [26]. Submitting this equation to the Crank-Nicolson scheme, and ignoring the dependence of \(\rho\) on the atom number part \(C\), we obtain
\[
i \frac{\phi_i^- + \delta \phi_i}{\Delta t} = f_i^+ + \delta f_i - \sum_j \langle \phi_j^+ | f_i^+ \rangle + \langle \delta f_i | \phi_j^+ \rangle - \frac{1}{2} \sum_j \left[\langle \delta \phi_j | f_i^+ \rangle \phi_j^+ + \langle \phi_j^+ | f_i^+ \rangle \delta \phi_j\right],
\] (A.15)

with
\[
\delta f_i = \hbar \frac{\delta \phi_i}{2} + \kappa \sum_{jkl} r_{ijkl} \left(\phi_j^* \phi_k \delta \phi_l + \frac{1}{2} \delta \phi_j^* \phi_k \phi_l\right).
\]

Eq. (A.15) can be solved with the Sherman–Morrison–Woodbury formula of Eq. (A.12). In many cases, we neglect in Eq. (A.15) the last term on the right-hand side, which is due to the projector in the orbital equations. This can be controlled through the \texttt{proj} argument in the options which can be set to 'on' or 'off'. For two orbitals the Newton iteration converges even without the projector term, and the computation becomes significantly faster.
Appendix B. Optimal control equations for MCTDHB model

Here we derive the somewhat lengthy optimal control equations for the MCTDHB equations. Let us first consider the atom number Hamiltonian

\[ H = \sum_{ij} (i|H|j) \hat{a}_i^\dagger \hat{a}_j + \frac{\hbar}{2} \sum_{ijkl} \int \phi_i^* \phi_k \phi_l \phi_j dx \hat{a}_i^\dagger \hat{a}_j \hat{a}_k \hat{a}_l. \]  

(B.1)

Differentiation with respect to the atomic orbitals gives

\[ \frac{\delta H}{\delta \phi_i} = \sum_j h \phi_j \hat{a}_j^\dagger \hat{a}_j + \frac{\hbar}{2} \sum_{jkl} \phi_i^* \phi_k \phi_l \hat{a}_i^\dagger \hat{a}_j \hat{a}_k \hat{a}_l \equiv \sum_\mu \varphi_{i\mu} h_{i\mu}, \]  

(B.2)

where the last expression defines the orbital functions \( \varphi_{i\mu} \) and the corresponding atom number operators \( h_{i\mu} \). For a given cost function \( J \), the Lagrange function of reads

\[ L = \frac{1}{2} \int \left[ \dot{\bar{C}} + (H - \bar{H})C \right] dt + \frac{1}{2} \sum_i \left[ \bar{\phi}_i^* \left(i\dot{\phi}_i - \mathcal{P} \dot{f}_i\right) + \left(i\dot{\phi}_i^* - \mathcal{P} f_i^*\right) \phi_i \right] dx dt + J, \]  

(B.3)

where we have introduced the shorthand notation \( \bar{H} = \langle C|H|C \rangle \). Here \( \bar{C} \) and \( \bar{\phi}_i \) are the adjoint variables introduced as constraints in order to fulfill the dynamic MCTDHB equations. To derive the OCT equations, we have to perform variational derivatives with respect to \( C \) and \( \phi_i \). As for the orbital part, we obtain

\[ i\dot{\phi}_i = \hbar \left( \mathcal{P} \bar{\phi}_i \right) + \kappa \sum_{jkl} r_{ijkl} \left[ \left( \mathcal{P} \bar{\phi}_j \right)^* \phi_k \phi_l + 2 \phi_j^* \phi_k \left( \mathcal{P} \bar{\phi}_i \right) \right] \]

\[ - \sum_j \left( \langle \bar{\phi}_j | \phi_i \rangle f_j - \langle f_j | \phi_i \rangle \bar{\phi}_j \right) + \sum_\mu \left[ \langle \bar{C} | h_{i\mu} - \bar{h}_{i\mu} \rangle \phi_i \right] \]  

(B.4)

where \( \bar{h}_{i\mu} = \langle C|h_{i\mu}|C \rangle h_{i\mu} \). As for the atom-number part of the adjoint equations, we next investigate the derivative of the density matrix term \( r = \rho^{-1} \rho^{(2)} \). From \( \rho^{-1} \rho = 1 \) we find \( \delta \rho^{-1} = \rho^{-1} \delta \rho + \rho^{-2} \delta \rho \), which finally yields \( \delta \rho^{-1} = -\rho^{-2} \rho \delta \rho \). Thus, \( \delta \rho = \rho^{-3} [\rho \delta \rho + \delta \rho^{(2)}] \). We then find

\[ \delta \frac{\delta}{\delta \bar{C}} \sum_{jkl} \bar{r}_{ijkl} \phi_j^* \phi_k \phi_l = \sum_{jklm} \left[ -(\rho^{-1})_{lm} \bar{r}_{ijkl} + \rho_{lm}^{(2)} \rho_{ijkl} \right] C \equiv \sum g_{ij} r_{i\mu} C, \]  

(B.5)

where we have introduced the operators \( \bar{\rho}_{ijkl} = \hat{a}_i^\dagger \hat{a}_j \hat{a}_k \hat{a}_l \) and \( \rho_{ijkl}^{(2)} = \hat{a}_i^\dagger \hat{a}_j \hat{a}_k \hat{a}_l \). The adjoint equation for the atom-number part of the wavefunction finally reads

\[ i\dot{\bar{C}} = (H - \bar{H}) C + 2Re \left[ \bar{C}^\dagger \cdot C \right] HC + \sum_{ij} \left[ \langle \bar{\phi}_i | \mathcal{P} g_{ij} \rangle r_{i\mu} + \langle P g_{ij} | \bar{\phi}_i \rangle r_{i\mu}^\dagger \right] C. \]  

(B.6)

References

[1] A. G"unther, S. Kraft, M. Kemmler, D. Koelle, R. Kleiner, C. Zimmermann, J. Fortagh, Phys. Rev. Lett. 95 (2005) 170405.
[2] X. Luo, P. Kr"uger, K. Brugger, S. Wildermuth, H. Gimpel, M. W. Klein, S. Groth, R. Folman, I. Bar-Joseph, J. Schmiedmayer, Optics Letters 29 (2004) 2145.
[3] P. Kr"uger, X. Luo, M. W. Klein, K. Brugger, A. Haase, S. Wildermuth, S. Groth, I. Bar-Joseph, R. Folman, J. Schmiedmayer, Phys. Rev. Lett. 91 (2003) 233201.
[4] T. Schumm, S. Hofferberth, L. M. Andersson, S. Wildermuth, S. Groth, I. Bar-Joseph, J. Schmiedmayer, P. Kr"uger, Nature Phys. 1 (2005) 57.
[5] S. Hofferberth, I. Lesanovsky, B. Fischer, J. Verdu, J. Schmiedmayer, Nature Physics 2 (2006) 710.

31
[6] Y.-J. Wang, D. Z. Anderson, V. M. Bright, E. A. Cornell, Q. Diot, T. Kishimoto, M. Prentiss, R. A. Saravanan, S. R. Segal, S. Wu, Phys. Rev. Lett. 94 (2005) 090405.
[7] W. Hänsel, J. Reichel, F. Hommelhoff, T. W. Hänsch, Phys. Rev. A 64 (2001) 063607.
[8] E. Andersson, T. Calarco, R. Folman, M. Andersson, B. Hessmo, J. Schmiedmayer, Phys. Rev. Lett. 88 (2002) 100401.
[9] T. Calarco, E. A. Hinds, D. Jaksh, J. Schmiedmayer, J. I. Cirac, P. Zoller, Phys. Rev. A 61 (2000) 022304.
[10] E. Charron, M. Cirone, A. Negretti, J. Schmiedmayer, T. Calarco, Phys. Rev. A 74 (2006) 012308.
[11] P. Treutlein, T. W. Hänsch, J. Reichel, A. Negretti, M. A. Cirone, T. Calarco, Phys. Rev. A 74 (2006) 022312.
[12] T. Paul, K. Richter, P. Schlagheck, Phys. Rev. Lett. 94 (2005) 020404.
[13] L. Deng, E. W. Hagley, J. Wen, M. Trippenbach, Y. Band, P. S. Julienne, J. E. Simsarian, K. Helmerson, S. L. Rolston, W. D. Phillips, Nature 298 (1999) 218.
[14] C. Orzel, A. K. Tuchman, M. L. Fenselau, M. Yasuda, M. A. Kasevich, Science 291 (2001) 2386.
[15] G. K. Campbell, J. Mun, M. Boyd, E. W. Streed, W. Ketterle, D. E. Pritchard, Phys. Rev. Lett. 96 (2006) 020406.
[16] A. Perrin, H. Chang, V. Krachmalnikoff, M. Schellekens, D. Boiron, A. Aspect, C. I. Westbrook, Phys. Rev. Lett. 99 (2007) 150405.
[17] J. Reichel, V. Valetic (Eds.), Atom Chips, Wiley, Weinheim, 2011.
[18] R. Folman, P. Krüger, J. Schmiedmayer, J. Denschlag, C. Henkel, Adv. in Atom. Mol. and Opt. Phys. 48 (2002) 263.
[19] I. Lesanovsky, T. Schumm, S. Hofferberth, L. M. Andersson, P. Krüger, J. Schmiedmayer, Phys. Rev. A 73 (3) (2006) 033619.
[20] I. Lesanovsky, S. Hofferberth, J. Schmiedmayer, P. Schmelcher, Phys. Rev. A 74 (2006) 033619.
[21] S. Wildermuth, S. Hofferberth, I. Lesanovsky, S. Groth, P. Krüger, J. Schmiedmayer, I. Bar-Joseph, Appl. Phys. Lett. 88 (2006) 264103.
[22] U. Hohenester, P. K. Rekdal, A. Borzi, J. Schmiedmayer, Phys. Rev. A 75 (2007) 023602.
[23] A. Borzi, U. Hohenester, SIAM Journal of Scientific Computing 30 (2008) 441.
[24] G. J. Milburn, J. Corney, E. M. Wright, D. F. Walls, Phys. Rev. A 55 (1997) 4318.
[25] J. Javanainen, M. Y. Ivanov, Phys. Rev. A 60 (1999) 2351.
[26] O. E. Alon, A. I. Streltsov, L. S. Cederbaum, Phys. Rev. A 77 (2008) 033613.
[27] J. Grond, J. Schmiedmayer, U. Hohenester, Phys. Rev. A 79 (2009) 021603(R).
[28] J. Grond, G. von Winckel, J. Schmiedmayer, U. Hohenester, Phys. Rev. A 80 (2009) 033625.
[29] U. Hohenester, J. Grond, J. Schmiedmayer, Fortschr. Phys. 57 (2009) 1121.
[30] J. Grond, U. Hohenester, I. Mazets, J. Schmiedmayer, New J. Phys. 12 (2010) 065036.
[31] J. Grond, T. Betz, U. Hohenester, N. J. Mauser, J. Schmiedmayer, T. Schumm, New J. Phys. 13 (2011) 065026.
[32] J. Grond, U. Hohenester, J. Schmiedmayer, A. Smerzi, Phys. Rev. A 84 (2011) 023619.
[33] R. Bücker, J. Grond, S. Manz, T. Berrada, T. Betz, C. Koller, U. Hohenester, T. Schumm, A. Perrin, J. Schmiedmayer, Nature Phys. 7 (2011) 508.
[34] R. Bücker, U. Hohenester, T. Berrada, S. van Frank, A. Perrin, S. Manz, T. Betz, J. Grond, T. Schumm, J. Schmiedmayer, Phys. Rev. A 86 (2012) 013638.
[35] R. Büker, T. Berrada, S. van Frank, T. Schumm, J. F. Schaff, J. Schmiedmayer, G. Jäger, J. Grond, U. Hohenester, J. Phys. B 46 (2013) 104012.
[36] G. Jäger, U. Hohenester, Phys. Rev. A 88 (2013) 035601.
[37] G. von Winckel, A. Borzi, Inverse Problems 24 (2008) 034007.
[38] F. Dalfovo, S. Giorgini, L. P. Pitaevskii, S. Stringari, Rev. Mod. Phys. 71 (1999) 463.
[39] A. Leggett, Rev. Mod. Phys. 73 (2001) 307.
[40] O. Morsch, M. Oberthaler, Rev. Mod. Phys. 78 (2006) 179.
[41] D. Jaksh, C. Bruder, J. I. Cirac, C. W. Gardiner, P. Zoller, Phys. Rev. Lett. 81 (1998) 3108.
[42] C. M. Dion, E. Canes, Comp. Phys. Commun. 177 (2007) 787.
[43] F. T. Arecchi, E. Courtens, R. Gilmore, H. Thomas, Phys. Rev. A 6 (1972) 2211.
[44] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical Recipes in C++: The Art of Scientific Computing, 2nd Edition, Cambridge Univ. Press, Cambridge, 2002.