GPELab

A Matlab toolbox for computing stationary solutions and dynamics of Gross-Pitaevskii Equations (GPE)

Université de Lorraine
Institut Elie Cartan de Lorraine, UMR CNRS 7502
F-54506 Vandoeuvre-lès-Nancy Cedex
FRANCE

Contents

List of Figures

List of Tables

Chapter 1

Introduction & informations

1.1 Overview

GPELab (Gross-Pitaevskii Equation LABoratory) is a Matlab toolbox devoted to the *numerical computation of stationary and dynamical solutions of the 1d-2d-3d GPE*, for general nonlinearities, including rotation terms and possibly multi-components problems. This user guide explains which problems GPELab can solve and which algorithms are used to effectively solve them. In addition, we discuss and detail all the functions available in the code. Finally, complete examples are given in such a way that any user can see how easy it is to manipulate it. GPELab is developed to be installed on any computer that has a basic version of Matlab. The methods are based on pseudospectral approximation techniques and therefore provide highly accurate solutions. The structure of the document is the following.

Section 1.2 of Chapter 1 shows which kinds of equations are solved and which dimensionless form of the GPE is used. We then discuss rapidly the classes of problems that GPELab can solve. A few remarks end Chapter 1.

Chapter 2 explains the numerical techniques that are coded in GPELab for stationary solutions. Mainly, the method is based on an imaginary time formulation of the problem. The discretization of the resulting equation uses a semi-implicit Backward Euler (BE) time approximation and a SPectral scheme in space based on Fast Fourier Transforms (FFTs). The method, called BESP, is extended to the case of multi-components systems and high rotations values can be considered. The nonlinearities can be classical like for the cubic case but can also be more complicate when considering dipole-dipole problems that handle a convolution kernel. Other methods included in GPELab are based on the Crank-Nicolson approximation scheme in time and the standard (3-points and 5-points) second-order finite differences scheme in space (for the 1d and 2d cases but not for the 3d case). We also discuss the possible classical choices of potentials that are met in the physics of Bose-Einstein condensates (but any other potential can be defined by the user itself) and how to choose the initial data for evolving the imaginary time method.

In Chapter 3, a simple example of code is given for one complete problem. It explains the general philosophy of GPELab and shows step-by-step the model code that a user has to define. Next, we give the full definition of all existing functions inside GPELab, and describe them in details. In particular, the way the arguments must be defined is carefully explained.

In Chapter 4, we describe how different 1d, 2d, 3d stationary problems can be solved by GPELab. This provides some generic codes for considering other problems that any user can meet. In particular, we give a few 1d, 2d and 3d examples including different potentials, nonlinearities, coupled systems. This offers the possibility of slightly modifying the code if your problem is not far from one of these examples. In the case of a different problem, we also explain what must be done, in conjunction with Chapter 3 that gives more insight into the function use.

1.2 The Gross-Pitaevskii Equation (GPE)

1.2.1 The GPE equation coming from physics

The aim of GPELab is to compute both stationary solutions and the dynamics of Bose-Einstein Condensates (BECs) based on the Gross-Pitaevskii Equation (GPE). We do not want here to describe the complex physics behind the BECs and GPE but only to state a few well-known facts about GPE and explain how to rewrite the physical GPE into a dimensionless GPE which is the model used in GPELab. It is also developed in such a way that the user can define its own equations and compute its proper physical outputs of interest.

Let us assume that the temperature T is much smaller than the critical temperature T^{critical}. Then, we can describe a BEC under a rotation effect through a macroscopic wave function ψ which depends on the spatial variable $\mathbf{x} := (x, y, z) \in \mathbb{R}^3$, and the time $t > 0$. This function has a dynamic which is governed by a specific nonlinear Schrödinger equation, the so-called Gross-Pitaevskii Equation, given by

$$i\hbar\frac{\partial\psi}{\partial t} = \frac{\delta E}{\delta\psi^*}(\psi) = H\psi = (-\frac{\hbar^2}{2m}\Delta + V(t,\mathbf{x}) + (N-1)U_0|\psi|^2 - \Omega L_z)\psi. \tag{1.1}$$

The atomic mass is m, $\hbar$ is the Planck constant and H is the hamiltonian operator of the system. The notation $\frac{\delta E}{\delta\psi^*}$ designates the functional derivatives of the energy E of the system, ψ^* being the complex conjugate of ψ. The number of atoms in the condensate is N and Ω is the angular velocity. The potential function V is an external trap which depends on $\mathbf{x}$ but may also depend on t according to the physical situation. The typical example of potential V is the confining harmonic (or quadratic potential) trap

$$V(\mathbf{x}) = \frac{m}{2}(\omega_x^2 x^2 + \omega_y^2 y^2 + \omega_z^2 z^2), \tag{1.2}$$

where ω_x, ω_y and ω_z are the trap frequencies in the directions x, y and z, respectively. The quantity U_0, defined by

$$U_0 = \frac{4\pi\hbar^2 a_s}{m}, \tag{1.3}$$

describes the interaction between atoms of the condensate, a_s being the scattering length which is positive for a repulsive interaction and negative for an attractive interaction. The operator L_z is such that

$$L_z = xp_y - yp_x = -i\hbar(x\partial_y - y\partial_x). \tag{1.4}$$

This is the z-component of the angular momentum $L = \mathbf{x} \times \mathbf{P}$, where the momentum operator $\mathbf{P} = -i\hbar\nabla = (p_x, p_y, p_z)^T$. The energy of the functional is defined by

$$E(\psi) = \int_{\mathbb{R}^3}[\frac{\hbar^2}{2m}|\nabla\psi|^2 + V|\psi|^2 + \frac{NU_0}{2}|\psi|^4 - \Omega\psi^* L_z\psi]d\mathbf{x}. \tag{1.5}$$

The wave function is generally normalized

$$\int_{\mathbb{R}^3}|\psi(t,\mathbf{x})|d\mathbf{x} = 1. \tag{1.6}$$

1.2.2 The dimensionless GPE

Let us introduce the following changes of variables

$$\begin{aligned} & t \to \frac{t}{\omega_m}, \quad \omega_m = \min(\omega_x, \omega_y, \omega_z), \\ & \mathbf{x} \to \mathbf{x} a_0, \quad a_0 = \sqrt{\frac{\hbar}{m\omega_m}}, \\ & \psi \to \frac{\psi}{a_0^{3/2}}, \\ & \Omega \to \Omega \omega_m, \\ & E(\cdot) \to \hbar \omega_m E_{\beta,\Omega}(\cdot). \end{aligned} \tag{1.7}$$

One obtains the dimensionless GPE

$$i\frac{\partial \psi}{\partial t} = \frac{\delta E_{\beta,\Omega}}{\delta \psi^*}(\psi) = H\psi = (-\frac{1}{2}\Delta + V + \beta|\psi|^2 - \Omega L_z)\psi, \tag{1.8}$$

where

$$\beta = \frac{U_0 N}{a_0^3 \hbar \omega_m} = \frac{4\pi a_s N}{a_0}, \tag{1.9}$$

and $L_z = -i(x\partial_y - y\partial_x)$. The potential is now

$$V(\mathbf{x}) = \frac{1}{2}(\gamma_x^2 x^2 + \gamma_y^2 y^2 + \gamma_z^2 z^2), \tag{1.10}$$

setting $\gamma_{x,y,z} = \omega_{x,y,z}/\omega_m$. The dimensionless energy functional $E_{\beta,\Omega}$ is defined by

$$E_{\beta,\Omega}(\psi) = \int_{\mathbb{R}^3} \left[\frac{1}{2}|\nabla\psi|^2 + V|\psi|^2 + \frac{\beta}{2}|\psi|^4 - \Omega\psi^* L_z \psi\right] d\mathbf{x}. \tag{1.11}$$

In the special case of a disk-shaped condensation, $\omega_x \approx \omega_y$ and $\omega_z \gg \omega_x$. This means that we have: $\gamma_x = 1$, $\gamma_y \approx 1$, and $\gamma_z \gg 1$, with $\omega = \omega_x$.

Then, the 3d GPE reduces to a 2d GPE given by

$$i\frac{\partial \psi}{\partial t} = (-\frac{1}{2}\Delta + V_2 + \beta_2(x,y)|\psi|^2 - \Omega L_z)\psi, \tag{1.12}$$

where $\beta_2 \approx \beta_2^a = \sqrt{\frac{\gamma_z}{2\pi}}$ and

$$V_2(x,y) = \frac{1}{2}(\gamma_x^2 x^2 + \gamma_y^2 y^2). \tag{1.13}$$

As a conclusion, one may write the GPE in d-dimensions through

$$i\frac{\partial \psi}{\partial t} = (-\frac{1}{2}\Delta + V_d + \beta_d(x,y)|\psi|^2 - \Omega L_z)\psi, \tag{1.14}$$

for $\mathbf{x} \in \mathbb{R}^d$, $t > 0$, $\beta_3 = \beta$ and $V_3(x,y,z) = V(x,y,z)$.

1.3 Which problems can GPELab solve?

GPELab (Gross-Pitaevskii Equation Laboratory) is a *versatile and robust Matlab code* that provides highly accurate (mainly spectral) numerical methods for computing the solutions of Gross-Pitevskii equations often used to simulate the physics of Bose-Enstein Condensates for superfluids. The main features contained in GPELab are

- computation of stationary states and dynamics of solutions for GPE,
- one-, two- and three-dimensional problems,
- general potentials, general local and nonlocal (dipolar) nonlinearities,
- fast rotating gazes,
- multi-components GPE,
- inclusion of stochastic terms at different places.

Furthermore, Matlab offers different visualization tools to observe and to represent the solutions of your calculations. GPELab also provides physical quantities of interest for the user. For beginners, GPELab is easy to use in many standard situations. For more advanced users, the flexibility of the code allows you to have your own defined output quantities that are not already contained in the code.

In terms of performance, GPELab is developed by using an optimized implementation in Matlab. Furthermore, the algorithms use recent developments in numerical methods. GPELab can also be used on parallel platforms so that it uses the power of parallel architecture. The time of computation of one given problem depends strongly of your physical configuration (for example strong nonlinear interactions or fast rotating gazes) and the computer resources that you can access.

1.4 How to read this manual

One way to read the manual is the following. First, Chapter 1 provides the general notations, the equations and the physical quantities that are involved into the equations. Then, you can read Sections 2.1 and 2.2 to understand what we wish to compute in the case of stationary solutions. Next you can go to Chapter 4 to see examples of computations for different physical situations and the associated GPELab codes. If you are interested in the numerical methods that are coded for the stationary states computation, you can read Sections 2.3 and 2.4 for the two-dimensional case, and next Section 2.5 for the 1d and 3d cases. Furthermore, if you want to understand the numerical schemes that are used as well as the notations for a multi-component case, then you have to take a look at Section 2.6. Finally, Section 3 explains in details the different functions that are included in GPELab. Reading again the examples of Section 4 will give you more understanding into the codes.

Another way of reading the user manual is the standard linear one, chapter after chapter.

1.5 Bug reports

In case of problems or bug, please send an email to the following address: `gpelab@univ-lorraine.fr`. We will do our best to provide an answer rapidly.

1.6 Copying conditions

GPELab is a ”free software”; this means that everyone is free to use it and to redistribute it on a free basis. GPELab is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of GPELab that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of GPELab, that you receive source code or else can get it if you want it, that you can change GPELab or use pieces of GPELab in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of GPELab, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for GPELab. If GPELab is modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the license for GPELab are found in the General Public License that accompanies the source code (see Appendix B [License], page 109). Further information about this license is available from the GNU Project webpage [1]. Detailed copyright information can be found in Appendix A [Copyright & credits], page 107.

If you want to integrate parts of GPELab into a closed-source software, or want to sell a modified closed-source version of GPELab, you will need to obtain a different license. Please contact us directly for more information.

[1] GNU License

Chapter 2

Computation of stationary states for the GPE

2.1 The Gross-Pitaevskii Equation and its properties

The aim of this section is to present the physical model, the mathematical notations and numerical methods that are developed in GPELab for computing stationary states solution like ground states and excited states. Furthermore, we explain which Matlab functions correspond to each method, and other physical quantities already existing in GPELab. Let us note that this part is mathematically a little bit technical. If you directly want to make computations, Chapter 3 provides the full details on how to use the code. Many examples are provided to help the beginner to directly launch some computations in Chapter 4.

2.1.1 The time-dependent GPE

Our aim is to solve the Gross-Pitaevskii Equation (GPE) with rotating terms. We essentially consider the dimensionless equation even if the equation with the usual physical parameters could be used. In the two-dimensional framework, the equation writes down [12, 15, 16]

$$i\partial_t\psi(t,\mathbf{x}) = -\frac{1}{2}\Delta\psi(t,\mathbf{x}) + V(t,\mathbf{x})\psi(t,\mathbf{x}) + \beta f(|\psi(t,\mathbf{x})|)\psi(t,\mathbf{x}) - \Omega L_z\psi(t,\mathbf{x}), (t,\mathbf{x}) \in \mathbb{R}^{*+}\times\mathbb{R}^2, \quad (2.1)$$

where ψ is the condensate wave function and $\mathbb{R}^{*+} :=]0;+\infty[$. The Laplace operator is defined as: $\Delta = \nabla^2$. Function V is the external (usually trapping) potential (for example harmonic). Parameter β is the nonlinearity strenght which describes the interaction between atoms in the condensate. It is related to the s-scattering length (a_s). It is positive for a repulsive interaction and negative for attractive interactions. Function f describes the nonlinearity arising in the problem, which is usually cubic: $f(|\psi|) = |\psi|^2$ but other cases will be considered later. For vortices creation, a rotating term is added. The operator L_z is defined as the z-component of the angular momentum $\mathbf{L} = (p_x, p_y, p_z)^t = \mathbf{x}\wedge\mathbf{P}$, with the momentum operator $\mathbf{P} = -i\nabla$. In the two-dimensional case and after some manipulations, its expression is

$$L_z = -i(x\partial_y - y\partial_x). \quad (2.2)$$

Two invariants must be satisfied by ψ, after normalization. The mass must be conserved

$$N(\psi) = \int_{\mathbb{R}^2} |\psi(t,\mathbf{x})|^2 d\mathbf{x} = \int_{\mathbb{R}^2} |\psi(0,\mathbf{x})|^2 d\mathbf{x} = ||\psi||_0^2 = 1, t > 0, \quad (2.3)$$

where $||\psi||_0$ is the $L^2(\mathbb{R}^2)$-norm of ψ. Another quantity which must be conserved is the energy $E_{\beta,0}$ that is given by (for a cubic nonlinearity here and a time-independent potential)

$$E_{\beta,\Omega}(\psi) = \int_{\mathbb{R}^2} \left[\frac{1}{2}|\nabla\psi|^2 + V(\mathbf{x})|\psi|^2 + \frac{\beta}{2}|\psi|^4 - \Omega\psi^* L_z\psi \right] d\mathbf{x}. \tag{2.4}$$

2.1.2 Stationary states

One important question in the numerical solution of GPE is the computation of stationary states. The problem consists in finding a solution

$$\psi(t,\mathbf{x}) = e^{-i\mu t}\phi(\mathbf{x}), \tag{2.5}$$

where μ is called the chemical potential of the condensate and ϕ is a time independent function. This solution is given as the solution to the nonlinear elliptic equation

$$\mu\phi(\mathbf{x}) = -\frac{1}{2}\Delta\phi(\mathbf{x}) + V(\mathbf{x})\phi(\mathbf{x}) + \beta|\phi(\mathbf{x})|^2\phi(\mathbf{x}) - \Omega L_z\phi(\mathbf{x}), \tag{2.6}$$

under the normalization constraint

$$||\phi||_0^2 = \int_{\mathbb{R}^2} |\phi(\mathbf{x})|^2 d\mathbf{x} = 1. \tag{2.7}$$

This nonlinear eigenvalue problem can be solved by computing the chemical potential

$$\mu_{\beta,\Omega}(\phi) = E_{\beta,\Omega}(\phi) + \frac{\beta}{4}\int_{\mathbb{R}^2} |\phi(\mathbf{x})|^4 d\mathbf{x}, \tag{2.8}$$

with

$$E_{\beta,\Omega}(\phi) = \int_{\mathbb{R}^2} \frac{1}{2}(|\nabla\phi|^2 + V|\phi|^2 + \beta|\phi|^4 - \Omega\phi^* L_z\phi) d\mathbf{x}. \tag{2.9}$$

This also means that the eigenfunctions are the critical points of the energy functional $E_{\beta,\Omega}$ over the unit sphere: $\mathbb{S} := \{||\phi||_0 = 1\}$. Furthermore, (2.6) can be seen as the Euler-Lagrange equations associated with the constraint minimization problem (2.7). Computing the global minimal solutions ϕ_g to the energy functional (2.9) under the normalization constraint

$$\phi_g = \underset{\phi\in\mathbb{S}}{\operatorname{argmin}}\ E_{\beta,\Omega}(\phi) \tag{2.10}$$

corresponds to obtain a ground state solution while local minima are excited (metastable) states.

2.2 Approximate solutions (as initial guess) and potentials

When one wants to compute numerically solutions to the minimization problem (2.9), then an iterative procedure is of course needed. This means that an initial guess has to be given to the method to initialize it and then the minimization process compute (or try to compute) a minimal solution through iterations. The aim of this section is to give informations concerning the choice of this initial guess which is often built analytically as an approximate minimal state for simplified problems.

For a non-rotating BEC, it can be proved that the global minimal solution is unique and gives a ground state $\phi_g \geq 0$ for a positive initial data ϕ_0. Therefore, one usually choose the solution to the linear Schrödinger equation with harmonic potential when we are under the critical frequency: $\Omega \ll \gamma_{xy}$, with $\gamma_{xy} = \min(\gamma_x;\gamma_y)$ for a harmonic trap

$$V(\mathbf{x}) = \frac{1}{2}(\gamma_x x^2 + \gamma_y y^2). \tag{2.11}$$

The initial data is then given by:

$$\phi(\mathbf{x}) = \frac{(\gamma_x\gamma_y)^{1/4}}{\sqrt{\pi}} e^{-(\gamma_x x^2+\gamma_y y^2)/2}. \tag{2.12}$$

This choice can also be considered for the (non-rotating) harmonic potential and a potential of a stirrer corresponding to a far-blue detuned Gaussian laser beam (toroidal trap) [16, 20, 22]

$$V(\mathbf{x}) = \frac{1}{2}(\gamma_x^2 x^2 + \gamma_y^2 y^2) + w_0 e^{-||\mathbf{x}-\mathbf{x}_0||^2/d^2}. \tag{2.13}$$

When a rotation is taken into account, the choice of the initial data is less clear. In [15], Bao *et al.* propose to choose, for $\gamma_x = \gamma_y = 1$,

$$\phi(\mathbf{x}) = \frac{(1-\Omega)\phi_{ho}(\mathbf{x}) + \Omega\phi_{ho}^v(\mathbf{x})}{||(1-\Omega)\phi_{ho}(\mathbf{x}) + \Omega\phi_{ho}^v(\mathbf{x})||_0}, \tag{2.14}$$

with

$$\phi_{ho}(\mathbf{x}) = \frac{1}{\sqrt{\pi}} e^{-(\gamma_x x^2+\gamma_y y^2)/2}, \qquad \phi_{ho}^v(\mathbf{x}) = \frac{(\gamma_x x + i\gamma_y y)}{\sqrt{\pi}} e^{-(\gamma_x x^2+\gamma_y y^2)/2} \tag{2.15}$$

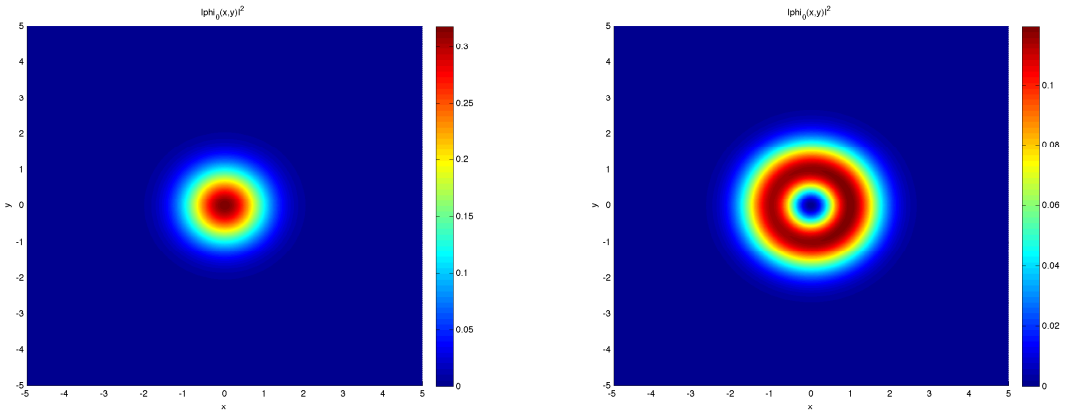
With the above initial data, ground states for rotating gazes can be obtained for $\Omega < \gamma_{xy}$ (while this is not e.g. the case with (2.12) when the rotating term is too large). In GPELab, all these possibilities can be found in the `GaussianInitialData2d` function where the following extended version is coded:

$$\phi(\mathbf{x}) = \sum_{\ell=1}^{p} \frac{(1-\Omega)\phi_{ho}(\mathbf{x}-\mathbf{x}_\ell) + \Omega\phi_{ho}^v(\mathbf{x}-\mathbf{x}_\ell)}{||(1-\Omega)\phi_{ho}(\mathbf{x}-\mathbf{x}_\ell) + \Omega\phi_{ho}^v(\mathbf{x}-\mathbf{x}_\ell)||_0}, \tag{2.16}$$

with

$$\phi_{ho}(\mathbf{x}) = \frac{(\gamma_x\gamma_y)^{1/4}}{\sqrt{\pi}} e^{-(\gamma_x x^2+\gamma_y y^2)/2}, \qquad \phi_{ho}^v(\mathbf{x}) = \frac{(\gamma_x\gamma_y)^{1/4}}{\sqrt{\pi}} (\gamma_x x + i\gamma_y y) e^{-(\gamma_x x^2+\gamma_y y^2)/2}. \tag{2.17}$$

Two examples of initial data using (2.16)-(2.17) are presented on Figure 2.1 (for the discrete L^2-norm defined by (2.58) and function `L2_norm2d`).



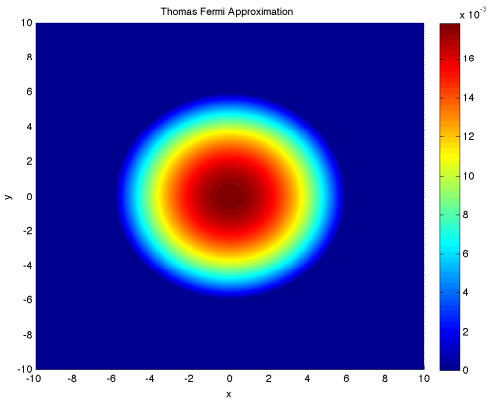
(a) $\Omega = 0$ (b) $\Omega = 0.99$

Figure 2.1: Two possible gaussian initial data for initializing the iterative scheme (with $\gamma_x = \gamma_y = 1$) and $\mathbf{x}_0 = (0, 0)$.

In the case of a strong linearity, one may also consider the Thomas-Fermi (TF) approximation of the ground state as initial data. For the 2d case and a quadratic potential, the TF approximate function is such that

$$\phi_\beta^{TF}(\mathbf{x}) = \begin{cases} \sqrt{(\mu_\beta^{TF} - V(\mathbf{x})/\beta_d}, & \text{if } \beta^{TF} > V(\mathbf{x}), \\ 0, & \text{otherwise.} \end{cases} \tag{2.18}$$

The eigenvalue approximation μ_β^{TF} is given by: $\mu_\beta^{TF} = (4\beta\gamma_x\gamma_y/\pi)^{1/2}/2$. The corresponding function is `Thomas_Fermi2d`. An example is given on figure 2.2 More details about these functions as well as `InitialData_Var2d` can be found in Section 3.6.6, page 53.



(a) $\beta = 1000$

Figure 2.2: Thomas-Fermi initial data for initializing the iterative scheme for potential (2.11) (with $\gamma_x = \gamma_y = 1$).

As already said, many kinds of potentials may be used. This is for example the case of a harmonic trap (2.11) (`quadratic_potential2d` function) with a possible added exponential term like in (2.13) (`quadratic_plus_exp_potential2d`). Examples of these two potentials are given below on Figure 2.4. Other possibilities include

- Quadratic plus quartic potential (`quadratic_plus_quartic_potential2d` function) [19]

$$V(\mathbf{x}) = (1-\alpha)\frac{1}{2}(\gamma_x^2x^2 + \gamma_y^2y^2) + \frac{\kappa}{4}(\gamma_x^2x^2 + \gamma_y^2y^2)^2. \tag{2.19}$$

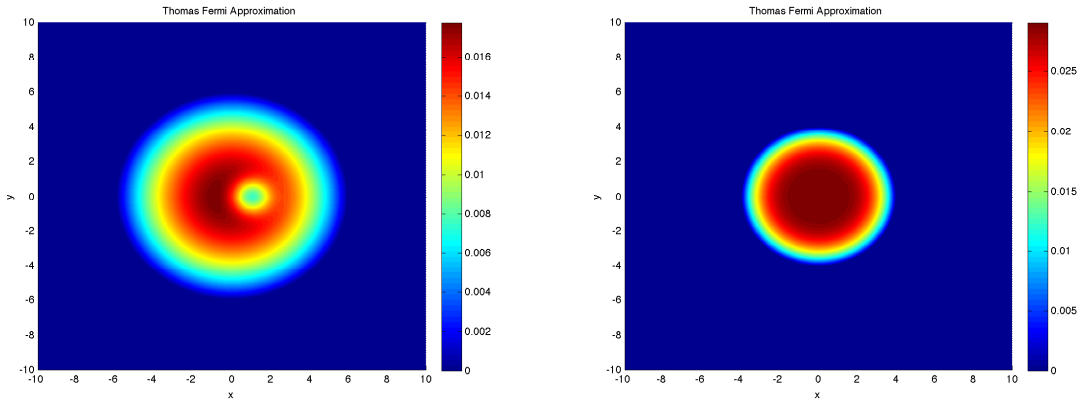
- Quadratic plus sin (optical) potential (`quadratic_plus_sin_potential2d` function) [19]

$$V(\mathbf{x}) = \frac{1}{2}(\gamma_x^2x^2 + \gamma_y^2y^2) + \frac{a_1}{2}\sin\left(\frac{\pi x}{d_1}\right)^2 + \frac{a_2}{2}\sin\left(\frac{\pi y}{d_2}\right)^2. \tag{2.20}$$

- Double well trapping potential (`double_well_trapping_potential2d` function) [22]

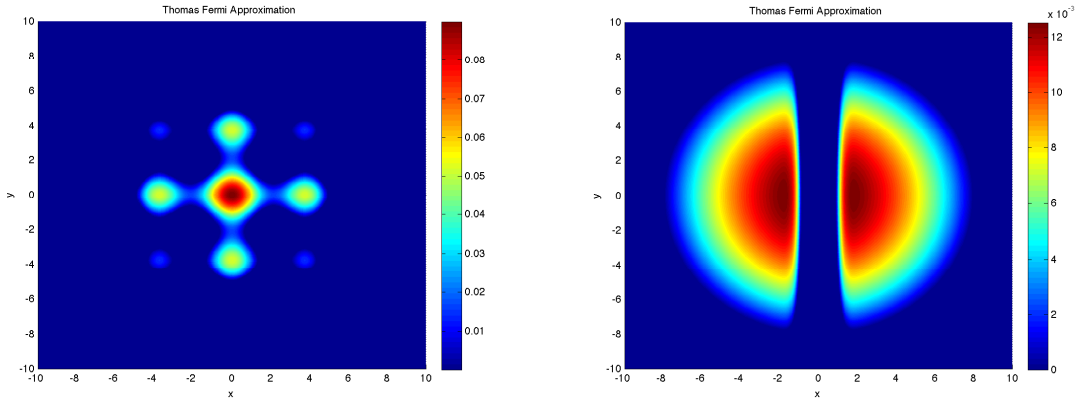
$$V(\mathbf{x}) = \frac{1}{2}\left(\gamma_x^2x^2 + \gamma_y^2y^2\right) + V_0e^{-x^2/2d^2}. \tag{2.21}$$

Examples of these potentials are given on Figures 2.4-2.3. Clearly, any new initial data or potential can be added by just following the way the functions are written. More details about the potential functions in GPELab are given in Subsection 3.6.2, page 45.



(a) $\gamma_x = \gamma_y = 1; \mathbf{x}_0 = (1,0); d = 1; w_0 = 10$ (b) $\gamma_x = \gamma_y = 1; \alpha = 1.2; \kappa = 0.3$

Figure 2.3: Examples Thomas-Fermi approximation for potentials (2.13) (left) and (2.19) (right).



(a) $\gamma_x = \gamma_y = 1; a_1 = a_2 = 25; d_1 = d_2 = 4$ (b) $\gamma_x = \gamma_y = 1/2;\ V_0 = 40;\ d = 0.5$

Figure 2.4: Examples Thomas-Fermi approximation for potentials (2.20) (left) and (2.21) (right).

2.3 Gradient flow formulation and discretization

One classical solution for computing the solution to (2.9)-(2.10) is through the *projected gradient method* which is also called *imaginary time method* in the Physics community. This is the basic method that is coded in GPELab for computing minimal solutions to (2.10).

The method consists in i) computing one step of a gradient method and then ii) project the solution onto the unit sphere $\mathbb{S}$. Let us denote by $t_0 < ... < t_n < ...$ the discrete times and by $\Delta t_n = t_{n+1} - t_n$ the local time step. The Continuous Normalized Gradient Flow (CNGF) is given by

$$\left\{ \begin{array}{l} \partial_t \phi = -\nabla_{\phi^*} E_{\beta,\Omega}(\phi) = \dfrac{1}{2}\Delta\phi - V\phi - \beta|\phi|^2\phi + \Omega L_z \phi, t_n < t < t_{n+1}, \\ \phi(\mathbf{x}, t_{n+1}) = \phi(\mathbf{x}, t_{n+1}^+) = \dfrac{\phi(\mathbf{x}, t_{n+1}^+)}{||\phi(\mathbf{x}, t_{n+1}^+)||_0} \\ \phi(\mathbf{x}, 0) = \phi_0(\mathbf{x}), \mathbf{x} \in \mathbb{R}^2, \text{ with } ||\phi||_0 = 1. \end{array} \right. \tag{2.22}$$

In the above equations, we set: $\phi(\mathbf{x}, t_{n+1}^{\pm}) := \lim_{t \to t_n^{\pm}} \phi(\mathbf{x}, t)$. Hence, iterations in times correspond to iterations in the projected gradient. It is proved in [16] that the CNGF is normalization conserving and energy diminishing if $\beta = 0$ and the potential is positive. Another interpretation is that the

CNGF is a first-order time splitting scheme with discontinuous coefficients. When t tends towards infinity, ϕ gives an approximation of the steady state solution which is a critical point of the energy when the assumption on V is fulfilled ($V \geq 0$). The initial guess ϕ_0 is chosen according to the possible choices provided in section 2.2. Finally, let us remark that, in our notations, we write $\phi(\mathbf{x}, t)$ and not $\phi(t, \mathbf{x})$ like for the dynamical case to insist on the fact that t is not a real time but rather a continuation parameter.

2.3.1 Time and space discretizations: the Backward Euler scheme

Different schemes can be considered for computing ground states. In [16], the authors show that the Time Splitting sine-Spectral (TSSP) and the Backward Euler Finite Difference schemes (BEFD) are well-adapted when no rotation is included. TSSP is supposed to be fast since this is an explicit scheme with FFT-based spatial discretization but it requires very small time steps when it is used for ground states computations. For this reason, we will not use this scheme for the stationary states (but it will be used for the dynamics). Here, we rather consider the BEFD scheme with rotating term. The scheme is implicit and therefore it requires at each step the solution to a linear system. It however can be solved efficiently by using a direct solver or a preconditioned Krylov subspace methods (e.g. a Bi Conjugate Gradient Stabilized (BiCGStab)). The interesting property is that the scheme is energy diminishing for the non rotating case and larger time steps can be used. The BEFD scheme is however only second-order accurate in space which is a limitation for computing fast rotating condensates. Higher order schemes could be used. This is the goal of subsection 2.3.3 where we present a new scheme (called BESP) based on BE in time but on a SPectral FFT scheme in space to capture accurately the creation of vortices for fast rotating condensates. The BESP is the scheme that you should prefer to use in GPELab when you consider fast rotating gazes.

2.3.2 Backward Euler Finite Difference (BEFD)

Concerning the time discretization of (2.22), the application of the Backward Euler scheme leads to the semi-discrete semi-implicit (linear) scheme (BE scheme)

$$\begin{cases} \dfrac{\tilde{\phi} - \phi^n}{\Delta t} = \dfrac{1}{2}\Delta\tilde{\phi} - V(\mathbf{x})\tilde{\phi} - \beta|\phi^n|^2\tilde{\phi} + \Omega L_z\tilde{\phi}, \quad 1 \leq n \leq N, \mathbf{x} \in \mathbb{R}^2, \\ \phi^{n+1} = \dfrac{\tilde{\phi}}{||\tilde{\phi}||_0}, \quad \mathbf{x} \in \mathbb{R}^2, \end{cases} \tag{2.23}$$

setting $M\Delta t = T_{\texttt{max}}$, where $T_{\texttt{max}}$ is the maximal time of computation and N is the number of time steps. Let us remark here that $T_{\texttt{max}}$ is not known *a priori* but rather fixed by a stopping criterion to check the convergence of the iterative scheme towards the ground state solution (see Eq. (2.31) for example). Until now, GPELab only includes uniform time stepping in time, for a fixed time step Δt.

For the numerical purpose, the scheme (2.23) still requires to be space discretized. To this end, we use a finite difference discretization here. Since the domain is $\mathbb{R}^2$, we have to set suitable boundary conditions. Here, we impose the homogeneous boundary condition $\tilde{\phi}(\mathbf{x}) = 0$ for $\mathbf{x}$ on the boundary of a large enough computational box: $\mathcal{O} :=]-L_x; L_x[\times]-L_y; L_y[$ assuming that the physics takes place inside this box. Moreover, we introduce the indices of the spatial grid points (x_j, y_k), for $(j,k) \in \mathcal{D}_{J,K}$, setting

$$\mathcal{D}_{J,K} = \left\{(j,k) \in \mathbb{N}^2; 1 \leq j \leq J-1 \text{ and } 1 \leq k \leq K-1\right\},$$

with $J, K \geq 3$ and uniform discretization steps h_x and h_y in the x- and y-directions, respectively. Therefore, for $1 < j \leq J-1$,

$$h_x = (x_j - x_{j-1}) = 2L_x/J,$$

and, for $1 < k \leq K-1$,

$$h_y = (y_k - y_{k-1}) = 2L_y/K.$$

The rotating term L_z is discretized by a two-point second-order scheme

$$[L_z]\phi^n_{j,k} := -i(x_j \delta_y \phi^n_{j,k} - y_k \delta_x \phi^n_{j,k}), \tag{2.24}$$

We associate a matrix $[L_z]$ to this discrete operator and denote by $\boldsymbol{\phi}^n := (\phi^n_{I(j,k)})_{(j,k)\in\mathcal{D}_{J,K}}$ the unknown vector where we assume that the global numbering is made by a local-to-global reordering procedure based on $I(j,k) = j + (J-1)(k-1)$ (which corresponds to using the `reshape` Matlab function when coding). Each discrete x- and y-derivative uses the two-points scheme adapted to the homogeneous Dirichlet boundary condition

$$\delta_x \phi^n_{j,k} = \frac{\phi^n_{j+1,k} - \phi^n_{j-1,k}}{2h_x}, \quad \delta_y \phi^n_{j,k} = \frac{\phi^n_{j,k+1} - \phi^n_{j,k-1}}{2h_y}. \tag{2.25}$$

Concerning the GPELab toolbox, the discrete rotational operator (2.24) can be found in the `Two_Points_Rotation2d` function. The derivative operators (2.25) are the `Two_Points_Gradientx2d` and `Two_Points_Gradienty2d` functions. These three operators provide square matrices in $\mathcal{M}_{M_\mathcal{D}}(\mathbb{C})$ with respect to the discretization, with $M_\mathcal{D} := (J-1)(K-1)$.

The Laplacian is discretized thanks to the five-points scheme with homogeneous Dirichlet boundary conditions. The interior scheme is based on

$$\begin{aligned} &\delta_x^2 \phi^n_{j,k} = \frac{\phi^n_{j+1,k} - 2\phi^n_{j,k} + \phi^n_{j-1,k}}{h_x^2}, \quad \delta_y^2 \phi^n_{j,k} = \frac{\phi^n_{j,k+1} - 2\phi^n_{j,k} + \phi^n_{j,k-1}}{h_y^2}, \\ &[\Delta]\phi^n_{j,k} = \delta_x^2 \phi^n_{j,k} + \delta_y^2 \phi^n_{j,k}, (j,k) \in \mathcal{D}_{J,K}. \end{aligned} \tag{2.26}$$

The corresponding function for $[\Delta]$ is named `FivePoints_Laplacian2d`. It provides the matrix $[\Delta] \in \mathcal{M}_{M_\mathcal{P}}(\mathbb{C})$ which must be applied to a (global) vector $\boldsymbol{\phi}^n := (\phi^n_{I(j,k)})_{(j,k)\in\mathcal{D}_{J,K}}$ of size $M_\mathcal{D}$. Finally, the potential is only considered at the discretization points and leads to a diagonal matrix $[V] \in \mathcal{M}_{M_\mathcal{P}}(\mathbb{C})$, with diagonal elements $[V]_{I(j,k)}$.

As a consequence, the spatial discretization of (2.23) leads to the $M_\mathcal{D} \times M_\mathcal{D}$ linear system with normalization step

$$\left\{ \begin{array}{l} [A]\tilde{\boldsymbol{\phi}} = \mathbf{b}^n, \\ \boldsymbol{\phi}^{n+1} = \dfrac{\tilde{\boldsymbol{\phi}}}{||\tilde{\boldsymbol{\phi}}||_0}, \end{array} \right. \tag{2.27}$$

with

$$\begin{aligned} &[A] := \frac{1}{\Delta t}[I] - \frac{1}{2}[\Delta] + [V] + \beta[|\boldsymbol{\phi}^n|^2] - \Omega[L_z], \\ &\mathbf{b}^n := \frac{\boldsymbol{\phi}^n}{\Delta t}. \end{aligned} \tag{2.28}$$

Hereabove, we set: $[I] \in \mathcal{M}_{M_\mathcal{P}}(\mathbb{C})$ as the identity matrix and $[|\boldsymbol{\phi}^n|^2] \in \mathcal{M}_{M_\mathcal{P}}(\mathbb{C})$ as the diagonal (potential) matrix with diagonal terms $[|\boldsymbol{\phi}^n|^2]_{I(j,k)} := |\boldsymbol{\phi}^n|^2_{I(j,k)}$. For the sake of conciseness, we denote by $||\cdot||_0$ the discrete 2-norm of a vector. In the finite difference context, the norm of a vector $\boldsymbol{\phi}$ is simply defined by

$$||\boldsymbol{\phi}||_0 := h_x^{1/2} h_y^{1/2} \Big(\sum_{(j,k)\in\mathcal{D}_{J,K}} |\phi_{j,k}|^2 \Big)^{1/2}. \tag{2.29}$$

The corresponding function is named `L2_norm2d`.

Now we come to the main function, `BEFD_CNGF2d`, which solves the ground states computation problem, by the BEFD scheme. Essentially, the description of the input parameters of the function follows the physics and the discretization. Furthermore, one can add drawing option as well as

solver choice. This last parameter proposes to solve the linear system by a direct gaussian solver or by a preconditioned restarted GMRES Krylov subspace solver. The preconditioner is based on the TF approximation related to the diagonal matrix

$$[M^{TF}] := \frac{1}{\Delta t}[I] + [V] + \beta[|\boldsymbol{\phi}^n|^2]. \tag{2.30}$$

The outputs are

- $\boldsymbol{\phi}_g$ the ground state solution at the final time according to the stopping criterion

$$||\boldsymbol{\phi}^{n+1} - \boldsymbol{\phi}^n||_\infty < \varepsilon \Delta t, \tag{2.31}$$

  wthere the tolerance is ε ($= 10^{-8}$ for example) and where $||\phi||_\infty$ is the infinity norm of the function defined at the discrete level by: $||\boldsymbol{\phi}||_\infty := \sup_{(j,k)\in\mathcal{D}_{J,K}} |\phi_{j,k}|$.

- `Time`: is the discrete vector of time steps

- $\boldsymbol{\phi}_g^2(0)$: the value of the square of the ground state at the origin (used in physics papers)

- $x_{\rm rms}$ and $y_{\rm rms}$: the radius mean square in the x- and y-directions according to

$$\alpha_{\rm rms} = ||\alpha \boldsymbol{\phi}_g||_0 \tag{2.32}$$

  The corresponding function is `alpha_rms`.

- $E_{\beta,\Omega}(\phi_g)$: The energy $E_{\beta,\Omega}(\phi_g)$ defined at the continuous level by

$$E_{\beta,\Omega}(\phi) := \int_{\mathbb{R}^2} \left[\frac{1}{2}|\nabla\phi(\mathbf{x})|^2 + V(\mathbf{x})|\phi(\mathbf{x})|^2 + \frac{\beta}{2}|\phi(\mathbf{x})|^4 - \Re(\Omega\phi^* L_z\phi)\right] d\mathbf{x} \tag{2.33}$$

  All the discretizations use the previous schemes at the interior nodes based on the trapezoidal rule. Our function is called: `energy_beta_psi_Diff2d`.

- $\mu_{\beta,\Omega}$: The chemical potential $\mu_{\beta,\Omega}$ is defined by

$$\mu_{\beta,\Omega} := E_{\beta,\Omega}(\phi) + \int_{\mathbb{R}^2} \frac{\beta}{2}|\phi(\mathbf{x})|^4 d\mathbf{x} \tag{2.34}$$

  The GPELab function is `chemical_potential` and the output is the chemical potential for the converged function ϕ. This function is coded in such a way that we do not have to precise the discretization scheme.

- $\mathcal{L}(\phi)$: The angular momentum $\mathcal{L}(\phi)$ is defined by

$$\mathcal{L}(\phi) := \int_{\mathbb{R}^2} i\left(x\partial_y - y\partial_x\right)\phi(x,y)dxdy \tag{2.35}$$

  The GPELab function is `Angularl_momentum_Diff2d` and the output is the angular momentum for the function ϕ at the converged state.

- The total cputime until convergence for computing the ground state solution by the scheme.

2.3.3 BESP: Spatial discretization/pseudospectral scheme based on FFTs

Rather than a finite difference scheme, pseudospectral approximation of the spatial derivatives can be used to get high-order accuracy. We consider in GPELab an approach based on Fourier series through FFTs. We now impose a periodic boundary condition on the boundary of a large enough computational box: $\mathcal{O} :=]-L_x; L_x[\times]-L_y; L_y[$ assuming that the physics takes place inside this box and the solution is confined in the box. Moreover, we introduce the indices of the spatial grid points (x_j, y_k), for $(j,k) \in \mathcal{P}_{J,K}$, setting

$$\mathcal{P}_{J,K} = \left\{(j,k) \in \mathbb{N}^2; 0 \leq j \leq J-1 \text{ and } 0 \leq k \leq K-1\right\},$$

with $J, K \geq 2$ and uniform discretization steps h_x and h_y in the x- and y-directions, respectively. Therefore, for $0 < j \leq J-1$,

$$h_x = (x_j - x_{j-1}) = 2L_x/J,$$

and, for $0 < k \leq K-1$,

$$h_y = (y_k - y_{k-1}) = 2L_y/K.$$

The partial Fourier pseudospectral discretizations in the x- and y-directions are respectively given by

$$\begin{aligned} \tilde{\phi}(x_j, y_k, t) &= \frac{1}{J} \sum_{p=-J/2}^{J/2-1} \widehat{\tilde{\phi}_p}(y_k, t) e^{i\mu_p(x_j + L_x)}, \\ \tilde{\phi}(x_j, y_k, t) &= \frac{1}{K} \sum_{q=-K/2}^{K/2-1} \widehat{\tilde{\phi}_q}(x_j, t) e^{i\lambda_q(y_k + L_y)}, \end{aligned} \tag{2.36}$$

where $\widehat{\tilde{\phi}_p}$ and $\widehat{\tilde{\phi}_q}$ are respectively the Fourier coefficients in the x- and y-directions

$$\begin{aligned} \widehat{\tilde{\phi}_p}(y_k, t) &= \sum_{j=0}^{J-1} \tilde{\phi}(x_j, y_k, t) e^{-i\mu_p(x_j + L_x)}, \\ \widehat{\tilde{\phi}_q}(x_j, t) &= \sum_{k=0}^{K-1} \tilde{\phi}(x_j, y_k, t) e^{-i\lambda_q(y_k + L_y)}, \end{aligned} \tag{2.37}$$

with $\mu_p = \frac{\pi p}{L_x}$ and $\lambda_q = \frac{\pi q}{L_y}$. For the backward Euler scheme, this implies that we have the following spatial approximation

$$\begin{cases} \mathbb{A}^{\mathrm{BE},n} \tilde{\boldsymbol{\phi}} = \mathbf{b}^{\mathrm{BE},n}, \\ \boldsymbol{\phi}^{n+1}(\mathbf{x}) = \dfrac{\tilde{\boldsymbol{\phi}}}{||\tilde{\boldsymbol{\phi}}||_0}, \end{cases} \tag{2.38}$$

where $\tilde{\boldsymbol{\phi}} = (\tilde{\phi}(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{P}_{J,K}}$ is the discrete unknown array in $\mathcal{M}_{M_\mathcal{P}}(\mathbb{C})$ and the right hand side is

$$\mathbf{b}^{\mathrm{BE},n} := \boldsymbol{\phi}^n / \Delta t,$$

with $\boldsymbol{\phi}^n = (\phi^n(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{P}_{J,K}} \in \mathcal{M}_M(\mathbb{C})$. Here, $\mathcal{M}_{M_\mathcal{P}}(\mathbb{C})$ designates the set of 2D (respectively 1D and 3D) arrays with complex coefficients, with $M_\mathcal{P} = JK$ (respectively $M_\mathcal{P} = J$ and $M_\mathcal{P} = JKL$) in 2D (respectively 1D and 3D). For conciseness, let us remark that we do not make any distinction between an array $\boldsymbol{\phi}$ in $\mathcal{M}_{M_\mathcal{P}}(\mathbb{C})$ and the corresponding reshaped vector in $\mathbb{C}^{M_\mathcal{P}}$.

The operator $\mathbb{A}^{\mathrm{BE},n}$ is given by the map which for any vector $\boldsymbol{\psi} \in \mathbb{C}^{M_\mathcal{P}}$, that is assumed to approximate $(\psi(\mathbf{x}_{j,k})) \in \mathbb{C}^{M_\mathcal{P}}$ for a function ψ, computes a vector $\boldsymbol{\Psi} \in \mathbb{C}^{M_\mathcal{P}}$ such that

$$\begin{aligned} \boldsymbol{\Psi} &:= \mathbb{A}^{\mathrm{BE},n}\boldsymbol{\psi} = \mathbb{A}^{\mathrm{BE},n}_{\mathrm{TF}}\boldsymbol{\psi} + \mathbb{A}^{BE}_{\Delta,\Omega}\boldsymbol{\psi}, \\ \mathbb{A}^{\mathrm{BE},n}_{\mathrm{TF}}\boldsymbol{\psi} &:= \left(\frac{[[I]]}{\Delta t} + [[V]] + \beta[[|\boldsymbol{\phi}^n|^2]]\right)\boldsymbol{\psi}, \\ \mathbb{A}^{\mathrm{BE}}_{\Delta,\Omega}\boldsymbol{\psi} &:= \left(-\frac{1}{2}[[\Delta]] - \Omega[[L_z]]\right)\boldsymbol{\psi}. \end{aligned} \tag{2.39}$$

The evaluation of the two above operators is made as follows. For $\mathbb{A}_{\mathrm{TF}}^{\mathrm{BE},n}$, the application is direct since it is realized pointwize in the physical space by setting

$$[[I]]_{j,k} := \delta_{j,k}, \qquad [[V]]_{j,k} := V(\mathbf{x}_{j,k}), \qquad [[|\boldsymbol{\psi}^n|^2]]_{j,k} = |\boldsymbol{\psi}^n|^2(\mathbf{x}_{j,k}), \tag{2.40}$$

for $(j,k) \in \mathcal{P}_{J,K}$. The symbol $\delta_{j,k}$ denotes the Dirac delta symbol which is equal to 1 if and only if $j = k$ and 0 otherwise. Let us note that the discrete operator $\mathbb{A}_{\mathrm{TF}}^{\mathrm{BE},n}$ is represented by a diagonal matrix after reshaping. The label TF refers to the fact that this operator is related to the discretization of the Thomas-Fermi approximation.

On another hand, using (2.36) and (2.37), the differential operators in the x or y direction are discretized as, $\forall (j,k) \in \mathcal{P}_{J,K}$,

$$([[\partial_x]]\tilde{\boldsymbol{\phi}})_{j,k} = \frac{1}{J} \sum_{p=-J/2}^{J/2-1} i\mu_p \widehat{\tilde{\phi}_p}(y_k,t) e^{i\mu_p(x_j+a_x)},$$
$$([[\partial_y]]\tilde{\boldsymbol{\phi}})_{j,k} = \frac{1}{K} \sum_{q=-K/2}^{K/2-1} i\lambda_q \widehat{\tilde{\phi}_q}(x_k,t) e^{i\lambda_q(y_k+a_y)}.$$

Therefore, we obtain the following pseudo-spectral approximation of the operator L_z on the spatial grid, $\forall (j,k) \in \mathcal{P}_{J,K}$

$$([[L_z]]\tilde{\boldsymbol{\phi}})_{j,k} = -i\left(x_j([[\partial_y]]\tilde{\boldsymbol{\phi}})_{j,k} - y_k([[\partial_x]]\tilde{\boldsymbol{\phi}})_{j,k}\right). \tag{2.41}$$

Another differentiation leads to the second order differential operators in the x or y direction, $\forall (j,k) \in \mathcal{P}_{J,K}$,

$$([[\partial_x^2]]\tilde{\boldsymbol{\phi}})_{j,k} = \frac{1}{J} \sum_{p=-J/2}^{J/2-1} -\mu_p^2 \widehat{\tilde{\phi}_p}(y_k,t) e^{i\mu_p(x_j+a_x)},$$
$$([[\partial_y^2]]\tilde{\boldsymbol{\phi}})_{j,k} = \frac{1}{K} \sum_{q=-K/2}^{K/2-1} -\lambda_q^2 \widehat{\tilde{\phi}_q}(x_k,t) e^{i\lambda_q(y_k+a_y)},$$

which gives the pseudo-spectral approximation of the Laplacian operator Δ

$$([[\Delta]]\tilde{\boldsymbol{\phi}})_{j,k} = \left([[\partial_x^2]]\tilde{\boldsymbol{\phi}} + [[\partial_y^2]]\tilde{\boldsymbol{\phi}}\right)_{j,k}. \tag{2.42}$$

The discrete Laplace operator $[[\Delta]]$ is diagonal in the Fourier space but not $[[L_z]]$. Finally, the discrete $||\cdot||_0$ norm is given by

$$\forall \boldsymbol{\phi} \in \mathbb{C}^{M_{\mathcal{P}}}, ||\boldsymbol{\phi}||_0 := h_x^{1/2} h_y^{1/2} \Big(\sum_{(j,k)\in\mathcal{P}_{J,K}} |\phi_{j,k}|^2 \Big)^{1/2} \tag{2.43}$$

In practice, the linear system (2.47) is efficiently solved by a Krylov solver (BiCGStab) preconditioned by the TF operator $\mathbb{A}_{\mathrm{TF}}^{\mathrm{BE},n}$ (see [23]).

In GPELab, the function for computing the ground state by BESP is `BESP_CNGF2d` for the two-dimensional case. The operator $[[\Delta]]$ applied to a vector is given in `Delta_Fourier2d`, the gradient operator is `Grad_Fourier2d`. Other Matlab files can be found in the directory `Code2D/FFT` but are nevertheless not useful when you use GPELab. They are transparent for the user and do not really need to be explained. They generally contain helpful functions for optimizing the BESP method (for instance preconditioners,...). BESP is the default method used to solve the CNGF in GPELab (see Section 3.5.1, page 41). The output physical quantities are the same as these provided by the BEFD scheme (see subsection 2.3.2, page 24).

2.4 Crank-Nicolson schemes

Another possibility is to use a semi-implicit Crank-Nicolson scheme [16]. This results in the discretization

$$\left\{ \begin{array}{c} A^{\mathrm{CN},n}\tilde{\phi} = b^{\mathrm{CN},n}, \\ \phi^{n+1} = \dfrac{\tilde{\phi}}{||\tilde{\phi}||_0}, \end{array} \right. \tag{2.44}$$

where $A^{\mathrm{CN},n}$ and $b^{\mathrm{CN},n}$ are such that

$$\begin{aligned} A^{\mathrm{CN},n} &:= \left(\frac{I}{\Delta t} - \frac{1}{4}\Delta + \frac{1}{2}V + \frac{1}{2}\beta|\phi^n|^2 - \frac{1}{2}\Omega L_z \right), \\ b^{\mathrm{CN},n} &:= \left(\frac{I}{\Delta t} + \frac{1}{4}\Delta - \frac{1}{2}V - \frac{1}{2}\beta|\phi^n|^2 + \frac{1}{2}\Omega L_z \right) \phi^n. \end{aligned} \tag{2.45}$$

In [16], the authors prove that the scheme is energy diminishing if and only if a CFL-type condition is satisfied which imposes a restriction on the time step. Indeed, Δt must be sufficiently small in this case. This is no such restriction for the Backward Euler scheme for the CNGF. Indeed, it can be proved that BESP is unconditionally energy diminishing (this is typically drastically different from what is usually met in a time dependent problem with a real time (and not an imaginary time)).

Concerning (2.44)-(2.45), the fully discrete system is

$$\left\{ \begin{array}{c} \mathbb{A}^{\mathrm{CN},n}\tilde{\boldsymbol{\phi}} = \mathbf{b}^{\mathrm{CN},n}, \\ \boldsymbol{\phi}^{n+1} = \dfrac{\tilde{\boldsymbol{\phi}}}{||\tilde{\boldsymbol{\phi}}||_0}, \end{array} \right. \tag{2.46}$$

where $\mathbb{A}^{\mathrm{CN},n}$ and $\mathbf{b}^{\mathrm{CN},n}$ are such that

$$\begin{aligned} \boldsymbol{\Psi} &:= \mathbb{A}^{\mathrm{CN},n}\boldsymbol{\psi} = \mathbb{A}_{\mathrm{TF}}^{\mathrm{CN},n}\boldsymbol{\psi} + \mathbb{A}_{\Delta,\Omega}^{\mathrm{CN},n}\boldsymbol{\psi}, \\ \mathbb{A}_{\mathrm{TF}}^{\mathrm{CN},n}\boldsymbol{\psi} &:= \left(\frac{[[I]]}{\Delta t} + \frac{1}{2}[[V]] + \frac{1}{2}\beta[[|\boldsymbol{\phi}^n|^2]] \right) \boldsymbol{\psi}, \\ \mathbb{A}_{\Delta,\Omega}^{\mathrm{CN},n}\boldsymbol{\psi} &:= \left(-\frac{1}{4}[[\Delta]] - \frac{1}{2}\Omega[[L_z]] \right) \boldsymbol{\psi}, \end{aligned} \tag{2.47}$$

and

$$\mathbf{b}^{\mathrm{CN},n} := \left(\frac{[[I]]}{\Delta t} + \frac{1}{4}[[\Delta]] - \frac{1}{2}[[V]] - \frac{1}{2}\beta[[|\boldsymbol{\phi}^n|^2]] + \frac{1}{2}\Omega[[L_z]] \right) \boldsymbol{\psi}. \tag{2.48}$$

The corresponding numerical method is coded in `CNSP_CNGF2d` for the two-dimensional case with a pseudospectral spatial discretization and `CNFD_CNGF2d` for the finite difference scheme. The output physical quantities are again the same as these provided by the BEFD scheme (see subsection 2.3.2, page 24).

2.5 One- and three-dimensional cases

2.5.1 One-dimensional case

Al the functions that are found in the two-dimensional case have been developed for the one-dimensional case. In this situation, there is clearly no rotation term. All functions can be found in the directory `Code1D` and have the same corresponding names as for the two-dimensional case but with the suffix `1d` instead of `2d`. BESP, BEFD, CNSP and CNFD methods are coded. From the user point of view, the example file (see subsection 4.1, page 63, and subsection 4.3, page 68) shows that considering a one- or a two-dimensional problem does not need a lot of modifications in the main GPELab file that is launched for the simulations.

2.5.2 Three-dimensional case

For the three-dimensional case, only BESP and CNSP methods are coded. In the same spirit as for the one- and two-dimensional functions, the suffix is `3d`. An example is given in subsection 4.7, page 79.

2.6 Extension to the multi-components case

2.6.1 The multi-components GPE

The continuous normalized gradient flow can also be extended to the multi-components case, i.e. a system of coupled Gross-Pitaevskii equations. For the sake of conciseness, the spatial variable $\mathbf{x}$ is defined by: $\mathbf{x} := (x_1, ..., x_d) \in \mathbb{R}^d$. We denote by $\Psi = (\psi_1, ..., \psi_{N_c})$, with $N_c \in \mathbb{N}^* := \mathbb{N} - \{0\}$, a vector of N_c wave functions and consider the following generic system of Gross-Pitaevskii equations

$$\begin{aligned} i\partial_t \Psi(t,\mathbf{x}) = -\frac{1}{2}\Delta\Psi(t,\mathbf{x}) + V(\mathbf{x})\Psi(t,\mathbf{x}) + \sum_{j=1}^{d} G^j(\mathbf{x})\partial_{x_j}\Psi(t,\mathbf{x}) \\ +\beta F(\Psi(t,\mathbf{x}),\mathbf{x})\Psi(t,\mathbf{x}),\ (t,\mathbf{x}) \in \mathbb{R}^+ \times \mathbb{R}^d, \end{aligned} \tag{2.49}$$

with initial condition: $\Psi(t=0,\mathbf{x}) := \Psi_0(\mathbf{x})$, and where the operators are defined by

- the diagonal Laplacian
$$\Delta\Psi(t,\mathbf{x}) = (\Delta\psi_j(t,\mathbf{x}))_{j=1,...,N_c},$$
- the potential matrix
$$V(\mathbf{x}) = \begin{pmatrix} V_{11}(\mathbf{x}) & V_{12}(\mathbf{x}) & \cdots & V_{1N_c}(\mathbf{x}) \\ V_{21}(\mathbf{x}) & V_{22}(\mathbf{x}) & \cdots & V_{2N_c}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ V_{N_c1}(\mathbf{x}) & V_{N_c2}(\mathbf{x}) & \cdots & V_{N_cN_c}(\mathbf{x}) \end{pmatrix},$$
- the variable coefficients matrices in front of the gradient
$$G^j(\mathbf{x}) = \begin{pmatrix} G^j_{11}(\mathbf{x}) & G^j_{12}(\mathbf{x}) & \cdots & G^j_{1N_c}(\mathbf{x}) \\ G^j_{21}(\mathbf{x}) & G^j_{22}(\mathbf{x}) & \cdots & G^j_{2N_c}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ G^j_{N_c1}(\mathbf{x}) & G^j_{N_c2}(\mathbf{x}) & \cdots & G^j_{N_cN_c}(\mathbf{x}) \end{pmatrix},$$
- the diagonal gradient
$$\partial_{x_j}\Psi(t,\mathbf{x}) = (\partial_{x_j}\psi_l(t,\mathbf{x}))_{l=1,...,N_c},$$
- the nonlinearity matrix
$$F(\Psi(t,\mathbf{x}),\mathbf{x}) = \begin{pmatrix} F_{11}(\Psi(t,\mathbf{x}),\mathbf{x}) & F_{12}(\Psi(t,\mathbf{x}),\mathbf{x}) & \cdots & F_{1N_c}(\Psi(t,\mathbf{x}),\mathbf{x}) \\ F_{21}(\Psi(t,\mathbf{x}),\mathbf{x}) & F_{22}(\Psi(t,\mathbf{x}),\mathbf{x}) & \cdots & F_{2N_c}(\Psi(t,\mathbf{x}),\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ F_{N_c1}(\Psi(t,\mathbf{x}),\mathbf{x}) & F_{N_c2}(\Psi(t,\mathbf{x}),\mathbf{x}) & \cdots & F_{N_cN_c}(\Psi(t,\mathbf{x}),\mathbf{x}) \end{pmatrix}.$$

Moreover, we have the following mass normalization constraint

$$N(\Psi) := \sum_{j=1}^{N_c} N(\psi_j) = \sum_{j=1}^{N_c} \int_{\mathbb{R}^d} |\psi_j(t,\mathbf{x})|^2 d\mathbf{x} = \sum_{j=1}^{N_c} \int_{\mathbb{R}^d} |\psi_j(0,\mathbf{x})|^2 d\mathbf{x} = \|\Psi\|_0^2 = 1.$$

We also introduce the energy

$$\begin{aligned} E(\Psi) := & \sum_{j=1}^{N_c} \frac{1}{2} \int_{\mathbb{R}^d} |\nabla \psi_j(t,\mathbf{x})|^2 d\mathbf{x} \\ & + \int_{\mathbb{R}^d} \Re \left(\Psi(t,\mathbf{x})^* \left[V(\mathbf{x}) + \sum_{k=1}^{d} G^k \partial_{x_k} + \beta F_{\texttt{energy}}(\Psi(t,\mathbf{x}),\mathbf{x}) \right] \Psi(t,\mathbf{x}) \right) d\mathbf{x}, \end{aligned} \tag{2.50}$$

where $F_{\texttt{energy}}$ is an operator related to the nonlinearity F by the differentiation relation

$$\frac{\delta \left(\Psi^* F_{\texttt{energy}}(\Psi) \right)}{\delta \Psi^*} = F(\Psi),$$

where δ designates the Gâteaux derivative. For example, in the case of a decoupled cubic nonlinearity, $F_{\texttt{energy}}$ is already defined in GPELab and is given by

$$F_{\texttt{energy}}(\Psi(t,\mathbf{x}),\mathbf{x}) = \frac{1}{2} \begin{pmatrix} |\psi_1(t,\mathbf{x})|^2 & 0 & \cdots & 0 \\ 0 & |\psi_2(t,\mathbf{x})|^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & |\psi_{N_c}(t,\mathbf{x})|^2 \end{pmatrix}.$$

For dipolar gazes, when a nonlocal integral form of the nonlinearity must be considered, then the user must define himself the corresponding function $F_{\texttt{energy}}$ in GPELab.

2.6.2 Stationary states and the CNGF

As in the single component case, we consider the problem of finding stationary states for the system (5.14). More specifically, we are looking for a solution Ψ such that

$$\Psi(t,\mathbf{x}) = e^{-it\mu} \Phi(\mathbf{x}),$$

where $\Phi = (\phi_1, ..., \phi_{N_c})$ is a time-independent function, which is a solution of the following problem

$$i\mu\Phi(\mathbf{x}) = -\frac{1}{2}\Delta\Phi(\mathbf{x}) + V(\mathbf{x})\Phi(\mathbf{x}) + \sum_{j=1}^{d} G^j(\mathbf{x}) \partial_{x_j} \Phi(\mathbf{x}) + \beta F(\Phi(\mathbf{x}),\mathbf{x})\Phi(\mathbf{x}), \tag{2.51}$$

under the total mass constraint $N(\Phi) = 1$ and where $\mu(\Phi)$ is the chemical potential given by the formula

$$\mu(\Phi) = \sum_{j=1}^{l} \frac{1}{2} \int_{\mathbb{R}^d} |\nabla \phi_j(\mathbf{x})|^2 d\mathbf{x} + \int_{\mathbb{R}^d} \Re \left(\Phi(\mathbf{x})^* \left[V(\mathbf{x}) + \sum_{k=1}^{d} G^k(\mathbf{x}) \partial_{x_k} + \beta F(\Phi(\mathbf{x}),\mathbf{x}) \right] \Phi(\mathbf{x}) \right) d\mathbf{x}.$$

Like in the one-component case, we propose to use the CNGF for the multi-components problem which is a direct extension. Let us denote by $t_0 < ... < t_n < ...$ the discrete times and by $\Delta t_n = t_{n+1} - t_n$ the local time step. The Continuous Normalized Gradient Flow is given by

$$\begin{cases} \partial_t \Phi = -\nabla_{\Phi^*} E(\Phi) = \dfrac{1}{2}\Delta\Phi - V(\mathbf{x})\Phi - \displaystyle\sum_{j=1}^{d} G^j(\mathbf{x}) \partial_{x_j} \Phi - \beta F(\Phi,\mathbf{x})\Phi, \ t_n < t < t_{n+1}, \\ \Phi(\mathbf{x}, t_{n+1}) = \dfrac{\Phi(\mathbf{x}, t_{n+1}^+)}{\|\Phi(\mathbf{x}, t_{n+1}^+)\|_0}, \\ \Phi(\mathbf{x}, 0) = \Phi_0(\mathbf{x}). \end{cases} \tag{2.52}$$

In the above equations, we set: $\Phi(\mathbf{x}, t_{n+1}^+) = \lim_{t \to t_{n+1}^+} \Phi(\mathbf{x}, t)$. Hence, iterations in times correspond to iterations in the projected gradient. When t tends towards infinity, Φ gives an approximation of the steady state which is solution to (2.51), that is supposed to exist here. The ground state is again computed as a solution of the minimization problem of the energy functional E under the normalization constraint

$$\Phi_g = \underset{\|\Phi\|_0=1}{\operatorname{argmin}} E(\Phi). \tag{2.53}$$

2.6.3 Time and space discretizations

For the same reason as in the single-component case, we focus on schemes based on the Backward Euler discretization, that is BEFD and BESP. Using the operators that we have introduced for the one-component case, the extension is direct, even from the point of view of the Krylov solver solution. We have the following time discretization of system (2.52) based on the semi-implicit Backward Euler scheme

$$\begin{cases} \dfrac{\tilde{\Phi} - \Phi^n}{\Delta t_n} = \dfrac{1}{2}\Delta\tilde{\Phi} - V(\mathbf{x})\tilde{\Phi} - \displaystyle\sum_{j=1}^{d} G^j(\mathbf{x})\partial_{x_j}\tilde{\Phi} - \beta F(\Phi^n, \mathbf{x})\tilde{\Phi}, \ 1 \le n \le M, \ \mathbf{x} \in \mathbb{R}^d, \\ \Phi^{n+1} = \dfrac{\tilde{\Phi}}{\|\tilde{\Phi}\|_0}, \ \mathbf{x} \in \mathbb{R}^d, \end{cases}$$

setting $M\Delta t = T_{\texttt{max}}$, where $T_{\texttt{max}}$ is the time of computation to get the solution satisfying the convergence criterion. Integer M is the number of related time steps. For the spatial discretization, we extend what is done in the one-component case to the multi-component case by considering the discrete Laplacian and gradients. The functions are evaluated pointwise on a rectangular uniform discretization grid, according to the space dimension. For the Finite Difference (respectively SPectral scheme), the resulting method is again called BEFD (respectively BESP). The semi-implicit Crank-Nicolson scheme is also implemented resulting in the CNFD and CNSP computational methods.

Let us consider for example the two-dimensional problem and let us denote $\mathbf{x}$ by: $\mathbf{x} = (x, y)$. We assume that the support of the evolving field of each component is inside a box

$$\mathcal{O} :=]-L_x; L_x[\times]-L_y; L_y[.$$

We consider the uniform grid

$$\mathcal{O}_{J,K} = \{\mathbf{x}_{j,k} := (x_j, y_k); 0 \le j \le J-1, 0 \le k \le K-1\},$$

J and K being two even positive integers. Furthermore, we introduce the indices of the spatial grid points (x_j, y_k), for $(j,k) \in \mathcal{D}_{J,K}$, setting

$$\mathcal{D}_{J,K} = \left\{(j,k) \in \mathbb{N}^2; 1 \le j \le J-1 \text{ and } 1 \le k \le K-1\right\},$$

with $J, K \ge 3$ and uniform discretization steps h_x and h_y in the x- and y-directions, respectively, given by

$$\begin{aligned} \Delta x_j &= x_{j+1} - x_j = h_x = 2L_x/J, \\ \Delta y_k &= y_{k+1} - y_k = h_y = 2L_y/K. \end{aligned} \tag{2.54}$$

Since we assume that all the components are compactly supported in $\mathcal{O}$, then each ϕ_l, $l = 1, ..., N_c$, satisfies a periodic boundary condition (which can in fact be put to zero) on $\partial\mathcal{O}$ and we can use discrete Fourier transforms. For BESP, the following approximation holds

$$\begin{aligned} \mathbb{A}^{\mathrm{BE},n}\tilde{\mathbf{\Phi}} &= \mathbf{b}^{\mathrm{BE},n}, \\ \mathbf{\Phi}^{n+1}(\mathbf{x}) &= \frac{\tilde{\mathbf{\Phi}}}{||\tilde{\mathbf{\Phi}}||_0}, \end{aligned} \tag{2.55}$$

where $\tilde{\mathbf{\Phi}} = ((\tilde{\phi}_1(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{D}_{J,K}}, ..., (\tilde{\phi}_{N_c}(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{D}_{J,K}})$ is the discrete unknown array in $\mathbb{C}^{MN_c}$ and the right hand side is

$$\mathbf{b}^{\mathrm{BE},n} := \mathbf{\Phi}^n/\Delta t,$$

with $\mathbf{\Phi}^n = ((\Phi_1^n(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{D}_{J,K}}, ..., (\Phi_{N_c}^n(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{D}_{J,K}}) \in \mathbb{C}^{MN_c}$.

The operator $\mathbb{A}^{\mathrm{BE},n} : \mathbb{C}^{MN_c} \to \mathbf{\Psi} \in \mathbb{C}^{MN_c}$ is defined by

$$\begin{aligned}
&\mathbb{A}^{\mathrm{BE},n}\mathbf{\Phi} = \mathbb{A}_{\mathrm{TF}}^{\mathrm{BE},n}\mathbf{\Phi} + \mathbb{A}_{\Delta,\Omega}^{\mathrm{BE}}\mathbf{\Phi},\\
&\mathbb{A}_{\mathrm{TF}}^{\mathrm{BE},n}\mathbf{\Phi} := \left(\frac{[[I_{N_c}]]}{\Delta t} + [[V]] + \beta[[F(\mathbf{\Phi}^{\boldsymbol{n}})]]\right)\mathbf{\Phi},\\
&\mathbb{A}_{\Delta,\nabla}^{\mathrm{BE}}\mathbf{\Phi} := \left(-\frac{1}{2}[[\Delta]] + [[G^1]][[\partial_x]] + [[G^2]][[\partial_y]]\right)\mathbf{\Phi}.
\end{aligned} \tag{2.56}$$

The finite dimensional operator $\mathbb{A}_{\mathrm{TF}}^{\mathrm{BE},n}$ is explicitly given through the matrices

$$[[I_{N_c}]] := \begin{pmatrix} [[I]] & 0 & \cdots & 0 \\ 0 & [[I]] & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & [[I]] \end{pmatrix}, \qquad [[V]] := \begin{pmatrix} [[V_{11}]] & [[V_{12}]] & \cdots & [[V_{1N_c}]] \\ [[V_{21}]] & [[V_{22}]] & \cdots & [[V_{2N_c}]] \\ \vdots & \vdots & \ddots & \vdots \\ [[V_{N_c1}]] & [[V_{2N_c}]] & \cdots & [[V_{N_cN_c}]] \end{pmatrix}$$

and

$$[[F(\mathbf{\Phi}^n)]] := \begin{pmatrix} [[F_{11}(\mathbf{\Phi}^n)]] & [[F_{12}(\mathbf{\Phi}^n)]] & \cdots & [[F_{1N_c}(\mathbf{\Phi}^n)]] \\ [[F_{21}(\mathbf{\Phi}^n)]] & [[F_{22}(\mathbf{\Phi}^n)]] & \cdots & [[F_{2N_c}(\mathbf{\Phi}^n)]] \\ \vdots & \vdots & \ddots & \vdots \\ [[F_{N_c1}(\mathbf{\Phi}^n)]] & [[F_{2N_c}(\mathbf{\Phi}^n)]] & \cdots & [[F_{N_cN_c}(\mathbf{\Phi}^n)]] \end{pmatrix}.$$

In the above equations, we set

$$[[F_{lm}(\mathbf{\Phi}^n)]] = \left(F_{lm}(\mathbf{\Phi}_{j,k}^n, \mathbf{x}_{j,k})\right)_{(j,k)\in\mathcal{D}_{J,K}},$$

where $\mathbf{\Phi}_{j,k}^n = (\phi_l(\mathbf{x}_{j,k}))_{l=1,...,N_c}$, and $[[V_{lm}]] = (V_{lm}(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{D}_{J,K}}$. The matrix $\mathbb{A}_{\Delta,\nabla}^{\mathrm{BE}}$ is implicitly given by the discrete differentiation operators *via* the FFT

$$[[\Delta]]\mathbf{\Phi} := ([[\Delta\phi_l]])_{l=1,...,N_c}$$

and

$$[[\partial_x]]\mathbf{\Phi} := ([[\partial_x\phi_l]])_{l=1,...,N_c}, \quad [[\partial_y]]\mathbf{\Phi} := ([[\partial_y\phi_l]])_{l=1,...,N_c}. \tag{2.57}$$

We also define

$$[[G^k]] := \begin{pmatrix} [[G_{11}^k]] & [[G_{12}^k]] & \cdots & [[G_{1N_c}^k]] \\ [[G_{21}^k]] & [[G_{22}^k]] & \cdots & [[G_{2N_c}^k]] \\ \vdots & \vdots & \ddots & \vdots \\ [[G_{N_c1}^k]] & [[G_{2N_c}^k]] & \cdots & [[G_{N_cN_c}^k]] \end{pmatrix}, \forall k = 1, 2,$$

setting $[[G_{lm}^k]] = (G_{lm}^k(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{D}_{J,K}}$.

Finally, the norm $||\cdot||_0$ of a discrete vector $\mathbf{\Phi}$ of N_c components is defined by

$$\forall \mathbf{\Phi} \in \mathbb{C}^{MN_c}, ||\mathbf{\Phi}||_0 := (\sum_{l=1}^{N_c} ||\boldsymbol{\phi}_l||_0^2)^{1/2}, \tag{2.58}$$

where the discrete norm for each component is given by (2.58).

For solving (5.28), we again use the preconditioned BiCGStab. Concerning the TF preconditioner, since we have coupling between gazes through $[[V]]$ and $[[F(\mathbf{\Phi}^n)]]$, $\mathbb{A}_{\mathrm{TF}}^{\mathrm{BE},n}$ is a nondiagonal matrix. We propose here to extract the diagonal part of $\mathbb{A}_{\mathrm{TF}}^{\mathrm{BE},n}$ to build the preconditioner, which means that we only consider potential and nonlinear effects in each single-component. Concretely, we build the following *diagonal* TF preconditioner $\mathbb{P}_{\mathrm{TF,diag}}^{\mathrm{BE},n}$ given by

$$\mathbb{P}_{\mathrm{TF,diag}}^{\mathrm{BE},n} := \frac{[[I_{N_c}]]}{\Delta t} + [[V_{\mathrm{diag}}]] + \beta[[F_{\mathrm{diag}}(\mathbf{\Phi}^{\boldsymbol{n}})]]$$

where $[[V_{\mathrm{diag}}]] := ([[V_{ll}]])_{l=1,\dots,N_c}$ and $[[F_{\mathrm{diag}}]] := ([[F_{ll}(\mathbf{\Phi}^n)]])_{l=1,\dots,N_c}$.

We can also directly inverse the matrix $\mathbb{A}_{\mathrm{TF}}^{\mathrm{BE},n}$. First, we remark that this matrix is composed of diagonal block matrices. The following formula can be used to directly compute the inverse of a block matrix

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} + A^{-1}BS^{-1}CA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}CA^{-1} & S^{-1} \end{pmatrix}$$

where A, B, C, D are diagonal matrices and $S = (D - CA^{-1}B)$ is the Schur complement of A, assuming that A can be inverted. We note that the result of this computation is still a matrix composed of diagonal block matrices. By using recursively this formula for the smaller blocks of the matrix

$$A_{p+1}^{-1} = \begin{pmatrix} A_p & B_p \\ C_p & D_p \end{pmatrix}^{-1} = \begin{pmatrix} A_p^{-1} + A_p^{-1}B_pS_p^{-1}C_pA_p^{-1} & -A_p^{-1}B_pS_p^{-1} \\ -S_p^{-1}C_pA_p^{-1} & S_p^{-1} \end{pmatrix}$$

where $S_p = (D_p - C_pA_p^{-1}B_p)$ and p is the level of recursity, we are able to compute the inverse of the block matrix $\mathbb{A}_{\mathrm{TF}}^{\mathrm{BE},n}$. Therefore, we can build the *full* TF preconditioner

$$\mathbb{P}_{\mathrm{TF,full}}^{\mathrm{BE},n} := \left(\frac{[[I_{N_c}]]}{\Delta t} + [[V]] + \beta[[F(\mathbf{\Phi}^{\boldsymbol{n}})]]\right)^{-1}$$

Chapter 3

How to use GPELab: stationary solutions

GPELab is a flexible Matlab toolbox which is able to compute the stationary solution and dynamics of Gross-Pitaevskii equations. The physical description of the equations that are defined and the mathematical algorithms are those described in the previous Chapters for the stationary situation. The code offers the possibility to compute solutions to multi-components BECs and to get physical important quantities. GPELab works through a main program that calls solvers corresponding to efficient and robust accurate numerical methods developed in the present document. From the point of view of the standard user, only this main program has to be modified. Furthermore, for more complex problems, the advanced user can defined his own physical inputs (potential, nonlinearity, number of components...) and outputs (new physical interesting quantities...). The user can also manipulate the computed quantities and draw figures or create movies in relations to its calculations, thanks to Matlab functions and already defined and well adapted GPELab visualization functions. Finally, GPELab also provides the possibility to include stochastic effects into the computations for example in the potential.

The aim of this chapter is to introduce the way GPELab works when you want to compute stationary solutions and to detail some functions related to this class of problems.

3.1 How to get and install GPELab

You can freely download the GPELab solver at the following address: `geplab.com` (a mettre correctement). The installation process is simple. You get GPELab from the website, save the archive `geplab.zip` on your laptop and unzip the file. Then, add the `GPELab` directory and its subdirectories in the Matlab setpath menu. You can launch Matlab which now knows the correct paths for using the GPELab functions. Once its done, GPELab can be directly used. You can open any of the examples files and test it to check that everything works well.

3.2 A simple but complete example

We now present how to compute the ground state of a one-component Gross-Pitaevskii equation with quadratic potential, cubic nonlinearity and rotational operator in 2D. The following program is an example of how you write a script in Matlab that will launch the computation of a ground state for such a physical configuration. The first part of the script consists in building two structures named `Method` and `Geometry2D` that will contain all the informations related to the method and the geometry respectively. In this example, we choose the BESP scheme to compute a ground state. Moreover, we fix the time step Δt such that: $\Delta t = 10^{-2}$ and the stopping criterion ε in (2.31) to:

$\varepsilon := 10^{-6}$. Concerning the geometry, the computational domain is $\mathcal{O} :=]-10, 10[\times]-10, 10[$ and the number of grid points (including the boundary points) is set to $N_x = 2^8 + 1$ and $N_y = 2^8 + 1$. We can see on Figure 3.1 how it is coded in GPELab.

```
Computation = 'Ground';
Ncomponents = 1;
Type = 'BESP';
Deltat = 1e-2;
Stop_time = [];
Stop_crit = 1e-6;
Method = Method_Var2d(Computation,Ncomponents, Type, Deltat, Stop_time ,
Stop_crit);
xmin = -10;
xmax = 10;
ymin = -10;
ymax = 10;
Nx = 2^8+1;
Ny = 2^8+1;
Geometry2D = Geometry2D_Var2d(xmin,xmax, ymin,ymax, Nx, Ny);
```

Table 3.1: An example of `Method` and `Geometry2D` in GPELab for computing a ground state.

The next step is to define the physical problem. In our case, we want to compute the ground state of the Gross-Pitaevskii equation

$$\begin{aligned} i\partial_t\Psi(x,y,t) &= \delta\Delta\Psi(x,y,t) + \frac{1}{2}\left(|x|^2+|y|^2\right)\Psi(x,y,t) + \beta|\Psi(x,y,t)|^2\Psi(x,y,t) \\ &\quad +i\Omega\left(y\partial_x - x\partial_y\right)\Psi(x,y,t), \end{aligned}$$

with $\delta = 0.5$, $\beta = 300$ and $\Omega = 0.7$. GPELab is designed in such a way that the user may define and add operators of the following types: a potential operator, a nonlinear operator and gradient operators. The potential operator and the nonlinear operator are functions of the space variables (and the wave function for the nonlinear operator) that are multiplied by the wave function. The gradient operators are defined by functions that are multiplied by the partial derivative of the wave function in the space directions. In our case, we identify

- the potential function: $V(x,y) = \frac{1}{2}\left(|x|^2+|y|^2\right)$,
- the nonlinear function: $F(\Psi,x,y) = |\Psi(t,x,y)|^2$,
- the gradient function in the x-direction: $G^1(x,y) = i\Omega y$,
- the gradient function in the y-direction: $G^2(x,y) = -i\Omega x$.

In this particular case, the operators are predefined in GPELab but for clarity we define them again in our script. To define the physical problem, we first need to build the `Physics2D` structure and set the values of the parameters δ, β and Ω. The `Physics2D` structure contains all the informations related to the physical problem, that is, among other, the functions related to the operators. Therefore, we also have to add the operators by using functions of GPELab to the `Physics2D` structure. We remark that, as we said, the operators are already predefined and set as default in the functions `Potential_Var2d`, `Nonlinearity_Var2d`, `Gradientx_Var2d` and `Gradienty_Var2d`. The resulting code in available in Table 3.2.

We now have to set the initial data. Initial data in GPELab are defined as a cell array, each cell containing a complex matrix which is the initial wave function of a component. GPELab users can

```
Delta = 0.5;
Beta = 300;
Omega = 0.7;
Physics2D = Physics2D_Var2d(Method,Delta,Beta);
Physics2D = Potential_Var2d(Method, Physics2D, @(x,y) (1/2)*(x.^2+y.^2));
Physics2D = Nonlinearity_Var2d(Method, Physics2D, @(phi,x,y) abs(phi).^2 );
Physics2D = Gradientx_Var2d(Method, Physics2D,@(x,y) 1i*Omega*y);
Physics2D = Gradienty_Var2d(Method, Physics2D,@(x,y) -1i*Omega*x);
```

Table 3.2: An example of how to define the Physics in GPELab through the `Physics2D` structure.

set the initial data themselves. However, the function `InitialData_Var2d` is helpful if one wants to use standard initial data like the centered gaussian or the Thomas-Fermi approximation (see Section 2.2, page 20). We would like in our case to use the Thomas-Fermi approximation as an initial wave function for the computation. This is done by setting the `InitialData_choice` variable to `2` as we can see this in Table 3.3.

```
InitialData_choice = 2 ;
Phi_0 = InitialData_Var2d(Method, Geometry2D, Physics2D,InitialData_choice);
```

Table 3.3: Initialization by the Thomas-Fermi approximation.

Finally, we want to launch the simulation. We would like to get informations about the wave function during the computations, for example to be sure that the energy is diminishing. Thus, we need to build the `Outputs` structure, that contains all the outputs computed during the simulation. Some outputs like the energy or the mean square radius are already defined and stored when computing. Moreover, to print these informations during the calculations, we have to build the `Print` structure that "explains" how to print the outputs. For example, in Table 3.4, we ask to print the informations every 15 iterations and to draw the solution.

```
Outputs = OutputsINI_Var2d(Method);
Printing = 1;
Evo = 15;
Draw = 1;
Print = Print_Var2d(Printing,Evo,Draw);
```

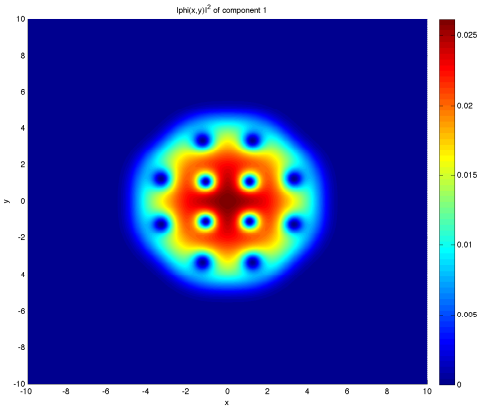
Table 3.4: Printing/drawing informations during the computations.

We are now ready to launch the simulation. This is done by using the `GPELab2d` function, which gather all the previous structure (and thus the informations about the simulation). The command is given in Table 3.5.

```
[Phi,Outputs]= GPELab2d(Phi_0,Method,Geometry2D,Physics2D,Outputs,[],Print);
```

Table 3.5: Launching the computation of the solution.

At the end of the simulation, we obtain the following figures



(a) Modulus of the ground state.

(b) Phase of the ground state.

Figure 3.1: Ground state computed with GPELab using the parameters from Section 3.2.

As we have seen with this example, using GPELab is quite easy and direct. We report more complex examples in Chapter 4 for different physical situations involving: ground states, dynamics, 1d-2d-3d problems, multi-components gazes...

3.3 Variables - types and various notions required for Matlab

To help the user who is not familiar with Matlab, we give quickly a few basic notions. For more informations, we refer for example to the Matlab user guide and help[1]. These examples are just to fix what is a matrix, a cell, a function or a structure and do not intend to give more informations. In practice, GPELab uses the following variables types

- **Matrix type**: Matrices are the basic variables of Matlab. A matrix is a multi-dimensional array of integers, real or complex numbers... To access a certain element of a matrix, the indexation must be done using parentheses

```
A = [1.3,-2;3.6,4.3];
A =
1.3 -2.
3.6 4.3
»
```

Table 3.6: An example of 2×2 real-valued matrix.

  Basic operations such as additions or multiplications for matrices are implemented in Matlab. Moreover, the element wise operations are done by adding a dot before the symbol of the operation. To initialize a matrix, one can choose the `zeros` function that will initialize a matrix of zeros.

- **Cell type**: Cells are variables that gather elements of Matlab such as matrices, functions or strings.

[1] `http://www.mathworks.fr/fr/help/matlab/`

```
A = [1,2;3,4];
B = [1,2;2,1];
A.*B
ans =
1 4
6 4
»
```

Table 3.7: An example of element wise multiplication for matrices.

```
A = {1,'string';3,[1,1;2,1]};
A =
[1] 'string'
[3] [2x2 double]
»
```

Table 3.8: An example of cell.

They use the same indexation as matrices and to access a certain element of a matrix, the indexation must be done using curly brackets. However, due to the fact that cells gather elements such as strings, basic operations are of course not implemented in Matlab.

```
A = {1,'string';3,[1,1;2,1]};
A{1,2}
ans =
string
»
```

Table 3.9: Accessing to an element of a cell.

- **Function**: Functions are scripts written in a simple way. They are defined as variables, making them more direct to create, to use and to manipulate than a script written in a `*.m` function file. To define a function, one has first to give the input arguments then write the script.

```
f = @(x,y,a,b) x*strcmp(a,b)+y;
f(2,3,'world','phone')
ans =
3
»
```

Table 3.10: An example of simple function.

- **Structure type**: Structures are variables that simply gather variables. Contrary to cells, there is no indexation and the access is done by directly naming the variables.

3.4 Notations and preliminary remarks

First, let us introduce some general notations to understand the types of the input and output arguments in the GPELab functions. Let us define

```
S.temperature = 32;
S.computer = 'office'
S =
temperature:  32
computer:  'office'
S.computer
ans =
'office'
»
```

Table 3.11: An example of structure.

- N_x, N_y, N_z: these parameters are equal to the number of degrees of freedom (dof) of the numerical method that is considered, in the x-, y- and z-directions, respectively. We emphasize here on the fact that these are *not* equal to $\overline{N}_x$, $\overline{N}_y$ and $\overline{N}_z$ which designate the total number of grid points, including the boundary points. For the FD scheme, the number of dof is $N_x = \overline{N}_x - 2$ in the x-direction and $N_x = \overline{N}_x - 1$ for the SP scheme. In example 3.1, page 36, $\overline{N}_x = 2^8 + 1$ but the number of dof is $N_x = 2^8$ which optimizes FFTs computations.
- N_c is the number of components in GPE.

Furthermore, let us consider the different sets of variables below that must be used when considering the corresponding Matlab variables in GPELab

- $\mathbb{N}$ denotes the positive integers,
- $\mathbb{R}$ designates the real numbers,
- $\mathbb{R}^+ := \mathbb{R} - \{0\}$ is the set of strictly positive real numbers,
- $\mathbb{C}$ denotes the set of complex numbers.

We also need the set of strings of characters that we designate by $\mathbb{S}$ and the set of Matlab structures denoted by $\mathcal{S}$.

We now introduce $K = \times_{j=1}^{N} K_j$ and $L = \times_{\ell=1}^{M} L_\ell$, where K_j and L_ℓ are two sets of variables like the ones defined above. In the sequel, we use the following notations

- $\mathbb{F}(K; L)$ is the set of Matlab functions `f` from $K \to L$ of the form

  `f : (x1,x2,...,xN)` $\to$ `@(x1,x2,...,xN) f(x1,x2,...,xN)`

  where $(x1, x2, ..., xN) \in K_1 \times K_2 \times ... \times K_N$. More generally, we also sometimes use the notation $\mathbb{F}(K^p; L) = \mathbb{F}(K, ..., K; L)$, if K is repeated p times.
- $\mathcal{M}_{N,M}(K)$ designates a $N \times M$ (Matlab) matrix with values in K, for N and $M \in \mathbb{N}$.
- $\mathcal{C}_{N,M}\{K\}$ is a $N \times M$ (Matlab) cell array with values in K, $N, M \in \mathbb{N}$.

Let us consider any input variable `xj` in a set K_j of a function `f`. In GPELab, all inputs of `f` have already default values $\texttt{xj}^{\text{default}}$ that can be modified. For clarity, in the sequel, we designate this by the notation: `xj` $(K_j, \texttt{xj}^{\text{default}})$.

We essentially detail the Matlab functions for the two-dimensional case. Unless precise, the extension from the 2d to the 1d and 3d cases is done by changing the functions' names. For example, the `Method_Var2d` function corresponds to the `Method_Var1d` function in 1d and to the `Method_Var3d` function in 3d. If changing the dimension implies any modification of the number or

the nature of the input or output arguments of the function, then we will precise it. Concerning the form of the variables x, y and z, we use the standard `meshgrid` ordering of variables. More precisely, this means that $x \in \mathcal{M}_{1,N_x}(K)$ in the 1d case, $x, y \in \mathcal{M}_{N_y,N_x}(K)$ in the 2d case and x, y, z are in $\mathcal{M}_{N_y,N_x,N_z}(K)$ for the 3d case. Here, $K = \mathbb{R}$. The same situation occurs when computing the set of frequencies (for example to compute nonlocal nonlinear interactions like for dipolar gazes) but $K = \mathbb{C}$.

3.5 Setting the numerical scheme and the geometry

First, the user has to define the geometry and the numerical method. There exists two variables that need to be defined: `Method` and `Geometry`. Those are created by using the two following functions: `Method_Var2d` and `Geometry2D_Var2d`.

3.5.1 The `Method_Var2d` function

```
Method = Method_Var2d(Computation,Ncomponents, Type, Deltat, Stop_time, Stop_crit,
Max_iter, Precond_type, Output, Splitting, BESP, Solver_FD, Iterative_tol,
Iterative_maxit);
```

Table 3.12: The `Method_Var2d` function.

The `Method_Var2d` function creates the `Method` *structure* that contains all the parameters relative to the method. By method, we mean the solver which is used to compute a solution. This includes the kind of computation (dynamics or ground state), the number of components, the type of scheme (BESP, BEFD, CNSP, CNFD for the ground state and Relaxation, Splitting for the dynamics), the semi discretization parameters and other inputs that we explain below. The only output is the structure `Method`. As seen above, the input variable of `Method_Var2d` have already default values that may be modified. The optional arguments are the following

- `Computation` ($\mathbb{S}$,`'Ground'`) is a variable that must be `'Ground'` to compute a ground state by using the Continuous Normalized Gradient Flow (imaginary time method).
- `Ncomponents` ($\mathbb{N}$,`1`) is a variable corresponding to the number of components that describe the condensate.
- `Type` ($\mathbb{S}$, `'BESP'`) is a variable corresponding to the scheme used in the computation. In the case of a ground state computation, it must be either `'BEFD'` to use the Backward Euler Finite Difference scheme (see section 2.3.2), `'CNFD'` to use the Crank-Nicolson Finite Difference scheme (see section 2.4), `'BESP'` to use the Backward Euler SPectral discretization scheme (see section 2.3.3) or `'CNSP'` to use the Crank-Nicolson SPectral discretization scheme (see section 2.4).
- `Deltat` ($\mathbb{R}^+$,`1e-3`) is a variable corresponding to the time step of the method. The time discretization is always uniform.
- `Stop_time` ($\mathbb{R}^+$,`1`) is a variable corresponding to the final time of computation in the case of a dynamic problem (see section 5.5.1).
- `Stop_crit` ($\mathbb{R}^+$,`1e-8`) is a variable corresponding to the stopping criterion (2.31).
- `Max_iter` ($\mathbb{N}$, `1e6`) is a variable corresponding to the maximum number of iterations for a stationary state computation.

- `Preconditioner` ($\mathbb{S}$,`'ThomasFermi'`) is a variable that must be either `'None'` for a calculation without preconditioner, `'Laplace'` for the Laplace preconditioner and `'ThomasFermi'` for the Thomas Fermi preconditioner.
- `Output` ($\mathbb{N}$,`1`) is a variable that must either be 1 if one computes outputs during the computations or 0 if not.
- `Splitting` ($\mathbb{S}$,`'Strang'`) is a variable corresponding to the type of splitting in the case of a dynamic computation (see section 5.5.1).
- `BESP` ($\mathbb{N}$,`0`) is a variable that must be either 1 if one uses the Jacobi method or 0 for the Krylov method, for the BESP scheme.
- `Solver_FD` ($\mathbb{N}$,`0`) is a variable that must be either 1 if one uses the direct Gauss solver from Matlab (i.e. backslash \) or 0 for the Krylov method.
- `Iterative_tol` ($\mathbb{R}^+$, `1e-9`) is a variable corresponding to the stopping criterion related to the difference between two successive iterates in the Krylov solver.
- `Iterative_maxit` ($\mathbb{N}$,`1e3`) is a variable corresponding to the stopping criterion related to the maximum number of iterations in the Krylov solver.

For example, we want to compute a stationary solution for a single-component BEC by using the BESP scheme. We choose a time step $\Delta t = 10^{-2}$ and a stopping criterion for $\varepsilon = 10^{-8}$. We set the maximal number of iterations to 10^6 and we choose to compute outputs during the simulation. This gives the code in table 3.13.

```
Computation = 'Ground';
Ncomponents = 1;
Type = 'BESP';
Deltat = 1e-2;
Stop_time = [];
Stop_crit = 1e-8;
Max_iter = 10e6;
Precond_type = 'None';
Output = 1;
Method = Method_Var2d(Computation,Ncomponents, Type, Deltat, Stop_time, Stop_crit,
Max_iter, Precond_type, Output;
```

Table 3.13: An example of initialization and use of the `Method_Var2d` function.

3.5.2 The `Geometry2D_Var2d.m` function

The call to the function `Geometry2D_Var2d.m` is

```
Geometry2D = Geometry2D_Var2d(xmin, xmax, ymin, ymax, Nx, Ny);
```

Table 3.14: The `Geometry2D_Var2d` function.

The aim of the `Geometry2D_Var2d.m` function is to create the `Geometry2D` *structure* which contains the size of the computational box and the number of points in each spatial direction (including the boundaries). Note that the spatial domain is always rectangular with a uniform mesh grid. The output is the `Geometry2D` structure. As for the `Method_Var2d` function, this function includes default values for the input arguments. The optional arguments are the following

- `xmin` ($\mathbb{R}$,`-10`) is a variable corresponding to the left endpoint of the computational domain in the x-direction.
- `xmax` ($\mathbb{R}$,`10`) is a variable corresponding to the right endpoint of the computational domain in the x-direction.
- `ymin` ($\mathbb{R}$,`-10`) is a variable corresponding to the lower endpoint of the computational domain in the y-direction.
- `ymax` ($\mathbb{R}$,`10`) is a variable corresponding to the upper endpoint of the computational domain in the y-direction.
- `Nx` ($\mathbb{N}$,`2^7+1`) is a variable corresponding to the number of points in the x-direction.
- `Ny` ($\mathbb{N}$,`2^7+1`) is a variable corresponding to the number of points in the y-direction.

In the case of a 1d simulation, one has to discard `ymin`, `ymax` and `Ny`. Moreover, in the case of a 3d simulation, one must add `zmin` and `zmax` after `ymax` and `Nz` after `Ny`.

If one wants to take a larger computational box $[-15, 15] \times [-15, 15]$ and a large number of grid points $N_x = N_y = 2^9 + 1$ to use a spectral scheme, then one builds the `Geometry2D` structure as in table 3.15.

```
xmin = -15;
xmax = 15;
ymin = -15;
ymax = 15;
Nx = 2^9+1;
Ny = 2^9+1;
Geometry2D = Geometry2D_Var2d(xmin, xmax, ymin, ymax, Nx, Ny);
```

Table 3.15: An example of how to use the `Geometry2D_Var2d` function.

3.6 Setting the physical problem

We now explain how to set the physical problem. We consider the following general GPE with N_c components, each one being defined in the d-dimensional space by

$$\begin{aligned} i\partial_t \Psi(t,\mathbf{x}) = -\delta \boldsymbol{\Delta} \Psi(t,\mathbf{x}) + \mathbf{V}(\mathbf{x})\Psi(t,\mathbf{x}) + \sum_{j=1}^{d} \mathbf{G}^j(\mathbf{x}) \partial_{x_j} \Psi(t,\mathbf{x}) \\ + \beta \mathbf{F}(\Psi(t,\mathbf{x}),\mathbf{x})\Psi(t,\mathbf{x}), \ (t,\mathbf{x}) \in \mathbb{R}^+ \times \mathbb{R}^d. \end{aligned} \tag{3.1}$$

This system corresponds to the one developed in Section 2.6, page 30. Here δ, β are two real-valued constants in $\mathbb{R}$. The energy for each component is set as

$$\begin{aligned}\mathbf{E}_j(\Psi) = \delta \int_{\mathbb{R}^d} |\nabla \psi_j(t,\mathbf{x})|^2 d\mathbf{x} \\ + \int_{\mathbb{R}^d} \Re\left(\psi(t,\mathbf{x})_j^* \left(\left[\mathbf{V}(\mathbf{x}) + \sum_{k=1}^{d} \mathbf{G}^k(\mathbf{x})\partial_{x_k} + \mathbf{F}_{energy}(\Psi(t,\mathbf{x}),\mathbf{x})\right]\Psi(t,\mathbf{x})\right)_j\right) d\mathbf{x},\end{aligned} \tag{3.2}$$

for each $j \in \{1, ..., N_c\}$, and the chemical potential for each component is set as

$$\begin{aligned}\boldsymbol{\mu}_j(\Psi) = \delta \int_{\mathbb{R}^d} |\nabla \psi_j(t,\mathbf{x})|^2 d\mathbf{x} \\ + \int_{\mathbb{R}^d} \Re\left(\psi(t,\mathbf{x})_j^* \left(\left[\mathbf{V}(\mathbf{x}) + \sum_{k=1}^{d} \mathbf{G}^k(\mathbf{x})\partial_{x_k} + \mathbf{F}(\Psi(t,\mathbf{x}),\mathbf{x})\right]\Psi(t,\mathbf{x})\right)_j\right) d\mathbf{x},\end{aligned} \tag{3.3}$$

for each $j \in \{1, ..., N_c\}$.

3.6.1 The `Physics2D_Var2d` function

```
Physics2D = Physics2D_Var2d(Method,Delta,Beta,Omega);
```

Table 3.16: The `Physics2D_Var2d` function.

The `Physics2D_Var2d` function builds the `Physics2D` *structure* and enables the user to define the basic physical constants δ, β and the rotation speed Ω (if the gradients operators are set as default (see section 3.6.5)). The `Physics2D` *structure* also contains the physical operators as explained below. The `Method` structure is a required argument and the optional arguments are the following

- `Delta` ($\mathbb{R}$,`1/2`) is a variable corresponding to the constant of the Laplacian operator i.e. δ in equation (3.1).
- `Beta` ($\mathbb{R}$,`0`) is a variable corresponding to the constant in front of the nonlinearity (β in equation (3.1)).
- `Omega` ($\mathbb{R}$,`0` or $\mathcal{M}_{1,3}(\mathbb{R})$,`0`) is a variable that defines the rotation speed if the default gradient operators ($(x\partial_y - y\partial_x)$ in 2d, or $(\mathbf{x} \times \nabla)$ in 3d) are present in the equation (otherwise, it has no effect). It is a real-valued parameter ($\Omega \in \mathbb{R}$) in the 2d case or a vector ($\boldsymbol{\Omega} \in \mathcal{M}_{1,3}(\mathbb{R})$) in the 3d case. We note that this variable does not exist in the 1d situation. If the default gradient operators are set, then we have the following rotation operators

  $$\sum_{j=1}^{2} \mathbf{G}^j(\mathbf{x})\partial_{x_j} = \begin{pmatrix} \Omega\,(x\partial_y - y\partial_x) & 0 & \cdots & 0 \\ 0 & \Omega\,(x\partial_y - y\partial_x) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Omega\,(x\partial_y - y\partial_x) \end{pmatrix},$$

  in the 2d physical problem, and

  $$\sum_{j=1}^{3} \mathbf{G}^j(\mathbf{x})\partial_{x_j} = \begin{pmatrix} \boldsymbol{\Omega}\cdot(\mathbf{x}\times\nabla) & 0 & \cdots & 0 \\ 0 & \boldsymbol{\Omega}\cdot(\mathbf{x}\times\nabla) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{\Omega}\cdot(\mathbf{x}\times\nabla) \end{pmatrix},$$

  for the 3d case.

We remind that in the case where one puts a rotational operator, with a rotation speed ranging from 0 to 1 excluded, then a quadratic potential ($V(\mathbf{x}) \approx |\mathbf{x}|^2$) is enough to compensate the centrifugal force. However, for a rotation speed larger than 1, one must use a stronger potential, for example a quartic potential ($V(\mathbf{x}) \approx |\mathbf{x}|^4$). In table 3.17, we show the code where we create the `Physics2D` structure with $\delta = \frac{1}{2}$, $\beta = 1000$ and $\Omega = 0.7$.

```
Delta = 0.5;
Beta = 1000;
Omega = 0.7;
Physics2D = Physics2D_Var2d(Method,Delta,Beta,Omega);
```

Table 3.17: An example to use the `Physics2D_Var2d` function.

3.6.2 The `Potential_Var2d` function

```
Physics2D = Potential_Var2d(Method, Physics2D, Potential, G );
```

Table 3.18: The `Potential_Var2d` function.

The `Potential_Var2d` function allows to define the time-independent potential operator (i.e. $\mathbf{V}(t, \mathbf{x}) = \mathbf{V}(\mathbf{x})$) in the problem by modifying the `Physics2D` structure. It must be provided with the `Method` and `Physics2D` structures. The optional arguments are the following

- `Potential`: If a function `Potential` in $\mathbb{F}(\mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))$ is provided, the physical potential is defined as follow, for each $j, k \in \{1, ..., N_c\}$,

$$\mathbf{V}_{j,k}(x, y) = \begin{cases} \texttt{Potential}(x, y) \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}$$

  If `Potential` is a cell array of functions in

$$\mathcal{C}_{N_c,N_c}\{\mathbb{F}(\mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))\},$$

  then the potential is defined by

$$\mathbf{V}_{j,k}(x, y) = \texttt{Potential}\{j, k\}(x, y),$$

  for $j, k \in \{1, ..., N_c\}$. The default argument is `quadratic_potential2d` which corresponds to

$$\mathbf{V}_{j,k}(x, y) = \begin{cases} \frac{1}{2}(x^2 + y^2) \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}$$

  Note that in the case of a stationary state computation, the potential operator should be time-independent.

- `G` ($\mathcal{M}_{N_c,N_c}(\mathbb{C})$, `ones(N_c)`) is a complex variable that multiplies the potential element-by-element, leading to the following potential

$$\mathbf{V}_{j,k}(x, y) = \texttt{G}(j, k)\texttt{Potential}\{j, k\}(x, y)$$

  for $j, k \in \{1, ..., N_c\}$.

For example, we want to set a quadratic potential for the computation of a ground state for a multi-components BEC with internal atomic Josephson junction, as in [17] where the system of two-components Bose-Einstein condensate is modeled by the following system of equations

$$\begin{cases} i\partial_t\psi_1 = \left[-\frac{1}{2}\Delta + V(\mathbf{x}) + \delta + (\beta_{11}|\psi_1|^2 + \beta_{12}|\psi_2|^2)\right]\psi_1 + \lambda\psi_2, \\ i\partial_t\psi_2 = \left[-\frac{1}{2}\Delta + V(\mathbf{x}) + (\beta_{22}|\psi_2|^2 + \beta_{12}|\psi_1|^2)\right]\psi_2 + \lambda\psi_1, \end{cases}$$

where δ is the detuning constant for the Raman transition, β_{jk} are the interactions constants and λ is the effective Rabi frequency.

Thus we have to build a potential operator, where the diagonal terms are quadratic potentials (plus the detuning constant δ for the first component) and the extradiagonal terms are the effective Rabi frequency λ. To this end, we have to create a cell array of functions and then we modify the `Physics2D` structure to define the potential operator, as it is done in table 3.19.

```
function P = Example_potential(Detuning_constant,Rabi_frequency)
P = cell(2);
P{1,1} = @(x,y) (1/2)*(x.^2+y.^2)+Detuning_constant;
P{1,2} = @(x,y) Rabi_frequency;
P{2,1} = @(x,y) Rabi_frequency;
P{2,2} = @(x,y) (1/2)*(x.^2+y.^2);
end
Detuning_constant = 1;
Rabi_frequency = -5;
Physics2D = Potential_Var2d(Method, Physics2D, ...
Example_potential(Detuning_constant,Rabi_frequency) );
```

Table 3.19: An example to use the `Potential_Var2d` function.

3.6.3 The `Nonlinearity_Var2d` function

```
Physics2D = Nonlinearity_Var2d(Method, Physics2D, Nonlinearity, G,
Nonlinearity_energy);
```

Table 3.20: The `Nonlinearity_Var2d` function.

The `Nonlinearity_Var2d` function allows to define the nonlinear term, i.e. $\mathbf{F}(\Psi(t,\mathbf{x}),\mathbf{x})$, in the problem by modifying the `Physics2D` structure. Note that in GPELab, the solution of the system is defined as a cell array of matrices ($\mathcal{C}_{N_c,N_c}\{\mathcal{M}_{Ny,Nx}(\mathbb{C})\}$). This function must be provided with the `Method` and `Physics2D` structures and it has the following optional arguments

- `Nonlinearity`: If a function `Nonlinearity` in

$$\mathbb{F}(\mathcal{C}_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\}, \mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))$$

is given, the physical nonlinearity will be defined as follows, for each $j,k \in \{1,...,N\}$,

$$\mathbf{F}_{j,k}(\Psi(t,\mathbf{x}),x,y) = \begin{cases} \texttt{Nonlinearity}(\Psi(t,\mathbf{x}),x,y) \text{ if } j=k \\ 0 \text{ if } j \neq k \end{cases}$$

If `Nonlinearity` is a cell array of functions in

$$\mathcal{C}_{N_c,N_c}\{\mathbb{F}(\mathcal{C}_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\},\mathcal{M}_{N_y,N_x}(\mathbb{R})^2;\mathcal{M}_{N_y,N_x}(\mathbb{C}))\},$$

then the nonlinear operator will be defined by

$$\mathbf{F}_{j,k}(\Psi(t,\mathbf{x}),x,y)=\texttt{Nonlinearity}\{j,k\}(\mathbf{\Psi}(t,\mathbf{x}),x,y),$$

for $j,k\in\{1,...,N\}$. The default argument is `Cubic2d` which corresponds to

$$\mathbf{F}_{j,k}(\Psi(t,\mathbf{x}),x,y)=\begin{cases}|\psi_j(t,\mathbf{x})|^2 \text{ if } j=k\\ 0 \text{ if } j\neq k\end{cases}$$

- `G` ($\mathcal{M}_{N_c,N_c}(\mathbb{C})$, `ones(N_c)`) is a complex-valued variable that multiplies the nonlinearity element-by-element, leading to the following nonlinearity definition

$$\mathbf{F}_{j,k}(\mathbf{\Psi}(t,\mathbf{x}),x,y)=\texttt{G}(j,k)\texttt{Nonlinearity}\{j,k\}(\mathbf{\Psi}(t,\mathbf{x}),x,y)$$

for $j,k\in\{1,...,N\}$.

- `Nonlinearity_energy` is a nonlinear operator used to compute the energy associated to the physical nonlinearity. It corresponds to $\mathbf{F}_{\texttt{energy}}(\Psi(t,\mathbf{x}),\mathbf{x})$ in the energy definition (3.2). Note that it must be the same type of variable as the variable `Nonlinearity`. If the variable `G` is defined, it will also be multiplied element by element by `Nonlinearity_energy`.

  If a function `Nonlinearity_energy` in

$$\mathbb{F}(\mathcal{C}_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\},\mathcal{M}_{N_y,N_x}(\mathbb{R})^2;\mathcal{M}_{N_y,N_x}(\mathbb{C}))$$

  is given, the nonlinear energy operator is defined as follows, for each $j,k\in\{1,...,N\}$,

$$(\mathbf{F}_{\texttt{energy}})_{j,k}(\Psi(t,\mathbf{x}),x,y)=\begin{cases}\texttt{Nonlinearity\_energy}(\Psi(t,\mathbf{x}),x,y) \text{ if } j=k\\ 0 \text{ if } j\neq k\end{cases}$$

  If `Nonlinearity_energy` is a cell array of function in

$$\mathcal{C}_{N_c,N_c}\{\mathbb{F}(\mathcal{C}_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\},\mathcal{M}_{N_y,N_x}(\mathbb{R})^2;\mathcal{M}_{N_y,N_x}(\mathbb{C}))\}$$

  then the nonlinear energy operator is

$$(\mathbf{F}_{\texttt{energy}})_{j,k}(\Psi(t,\mathbf{x}),x,y)=\texttt{Nonlinearity\_energy}\{j,k\}(\mathbf{\Psi}(t,\mathbf{x}),x,y)$$

  for $j,k\in\{1,...,N\}$. The default argument is `Cubic_energy2d` which corresponds to

$$(\mathbf{F}_{\texttt{energy}})_{j,k}(\Psi(t,\mathbf{x}),x,y)=\begin{cases}\frac{1}{2}|\psi_j(t,\mathbf{x})|^2 \text{ if } j=k\\ 0 \text{ if } j\neq k\end{cases}$$

This way of proceeding allows us to continue the example from Section 3.6.2, page 45, where we set a potential in the case of an internal atomic Josephson junction. We also need to define the coupled nonlinearities if we want to effectively take into account all the effects in the system of equations [17]. In the case of a two-components Gross-Pitaevskii equation with a Josephson junction, we have

$$\begin{cases}F_{11}(\mathbf{\Psi}(t,\mathbf{x}),\mathbf{x})=\beta_{11}|\psi_1|^2+\beta_{12}|\psi_2|^2,\\ F_{22}(\mathbf{\Psi}(t,\mathbf{x}),\mathbf{x})=\beta_{22}|\psi_2|^2+\beta_{12}|\psi_1|^2,\end{cases}$$

and $F_{12}(\mathbf{\Psi}(t,\mathbf{x}),\mathbf{x})=F_{21}(\mathbf{\Psi}(t,\mathbf{x}),\mathbf{x})=0$. This is done in table 3.21, where we create a cell array of functions corresponding to the previous nonlinearities and then define the nonlinear operator by using the `Nonlinearity_Var2d` function.

```
function NL = Example_nonlinearity(Beta_11,Beta_22,Beta_12)
NL = cell(2);
NL{1,1} = @(Phi,x,y) Beta_11*abs(Phi{1}).^2 + Beta_12*abs(Phi{2}).^2;
NL{2,2} = @(Phi,x,y) Beta_22*abs(Phi{2}).^2 + Beta_12*abs(Phi{1}).^2;
NL{1,2} = @(Phi,x,y) 0;
NL{2,1} = @(Phi,x,y) 0;
end
Beta_11 = 2;
Beta_12 = 1;
Beta_22 = 2;
Physics2D = Nonlinearity_Var2d(Method, Physics2D, ...
Example_nonlinearity(Beta_11,Beta_22,Beta_12));
```

Table 3.21: An example to use the `Nonlinearity_Var2d` function.

3.6.4 The `FFTNonlinearity_Var2d` function

```
Physics2D = FFTNonlinearity_Var2d(Method, Physics2D, FFTNonlinearity, G,
FFTNonlinearity_energy);
```

Figure 3.2: The `FFTNonlinearity_Var2d` function.

In the case where one wants to consider a nonlocal (integral-type) nonlinearity in the GPE, this can be done in GPELab by using the Fourier transform. The aim is to be able to define general nonlinear interactions through the representation formula

$$\mathbf{F}_{j,k}(\Psi(t,\mathbf{x}),x,y) := \texttt{ifft}(\mathcal{K}_{j,k}(\mathbf{x},\boldsymbol{\xi}).*\texttt{fft}(\mathbf{H}_{j,k}(\Psi)))(t,\mathbf{x})$$

for $j,k \in \{1,...,N_c\}$. For each j,k, the symbol $\mathcal{K}_{j,k}(\mathbf{x},\boldsymbol{\xi})$ is a Matlab function in

$$\mathbb{F}(\mathcal{M}_{N_y,N_x}(\mathbb{R})^2, \mathcal{M}_{N_y,N_x}(\mathbb{C})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C})).$$

Each function $\mathbf{H}_{j,k}(\Psi)$ is an element of

$$\mathbb{F}(\mathcal{C}_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\}; \mathcal{M}_{N_y,N_x}(\mathbb{C})).$$

Here above, `fft` and `ifft` designate the FFT and inverse FFT in Matlab, respectively. The symbol $.*$ is the point wise multiplication of arrays in Matlab. For example, if one wants to compute a convolution form (like in a one-component dipolar gaz)

$$\mathbf{F}(\psi(t,\mathbf{x}),x,y) := (K \star |\psi|^2)(t,\mathbf{x}),$$

then you can use the relation

$$\mathbf{F}(\psi(t,\mathbf{x}),x,y) := \texttt{ifft}(\mathcal{K}(\boldsymbol{\xi}).*\texttt{fft}(|\psi|^2))(t,\mathbf{x}),$$

where $\mathcal{K}$ is the Fourier transform of the kernel K. Another example is the derivation operator ∂_x which has symbol $-i\xi_x$. A full example is given in Section 4.7, page 79. We remark that this function is effective only in the case of spectral schemes.

The `FFTNonlinearity_Var2d` function allows to define the nonlinear operator (i.e. $\mathbf{F}(\Psi(t,\mathbf{x}),\mathbf{x})$ in our equation) in the problem. To this end, it modifies the `Physics2D` structure. The `Method` and `Physics2D` structures are required arguments and the optional arguments are the following

- `FFTNonlinearity`: If we have a function `FFTNonlinearity` in

$$\mathbb{F}(\mathcal{C}_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\},\mathcal{M}_{N_y,N_x}(\mathbb{R})^2,\mathcal{M}_{N_y,N_x}(\mathbb{C})^2;\mathcal{M}_{N_y,N_x}(\mathbb{C})),$$

the physical nonlocal nonlinearity is such that, for each $j,k\in\{1,...,N_c\}$,

$$\mathbf{F}_{j,k}(\Psi(t,\mathbf{x}),x,y)=\left\{\begin{array}{l}\texttt{FFTNonlinearity}(\Psi(t,\mathbf{x}),x,y,\xi_x,\xi_y)\text{ if }j=k\\0\text{ if }j\neq k\end{array}\right.$$

where ξ_x and ξ_y are the discrete Fourier frequencies in the x- and y-directions, respectively.

If `FFTNonlinearity` is given by a cell array of functions in

$$\mathcal{C}_{N_c,N_c}\{\mathbb{F}(\mathcal{C}_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\},\mathcal{M}_{N_y,N_x}(\mathbb{R})^2,\mathcal{M}_{N_y,N_x}(\mathbb{C})^2;\mathcal{M}_{N_y,N_x}(\mathbb{C}))\},$$

the nonlocal nonlinearity is

$$\mathbf{F}_{j,k}(\Psi(t,\mathbf{x}),x,y)=\texttt{FFTNonlinearity}\{j,k\}(\Psi(t,\mathbf{x}),x,y,\xi_x,\xi_y)$$

for $j,k\in\{1,...,N_c\}$. The default argument is `Cubic2d` which corresponds to

$$\mathbf{F}_{j,k}(\Psi(t,\mathbf{x}),x,y)=\left\{\begin{array}{l}|\psi_j(t,\mathbf{x})|^2\text{ if }j=k\\0\text{ if }j\neq k\end{array}\right.$$

- `G` ($\mathcal{M}_{N_c,N_c}(\mathbb{C})$, `ones(N_c)`) is a variable that multiplies the nonlocal nonlinearity element-by-element, leading to

$$\mathbf{F}_{j,k}(\mathbf{\Psi}(t,\mathbf{x}),x,y)=\texttt{G}(j,k)\texttt{FFTNonlinearity}\{j,k\}(\Psi(t,\mathbf{x}),x,y,\xi_x,\xi_y)$$

for $j,k\in\{1,...,N_c\}$.

- `FFTNonlinearity_energy` is a nonlinear operator used to compute the energy associated to the physical nonlocal nonlinearity (i.e. it corresponds to $\mathbf{F}_{\texttt{energy}}(\Psi(t,\mathbf{x}),\mathbf{x})$ in the energy (3.2)). Note that it must be the same type of variable as `FFTNonlinearity`. Again, if `G` is defined, it is multiplied element by element by `FFTNonlinearity_energy`.

If a function `FFTNonlinearity_energy` in

$$\mathbb{F}(\mathcal{C}_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\},\mathcal{M}_{N_y,N_x}(\mathbb{R})^2,\mathcal{M}_{N_y,N_x}(\mathbb{C})^2;\mathcal{M}_{N_y,N_x}(\mathbb{C}))$$

is given, the nonlocal nonlinear energy operator is, for each $j,k\in\{1,...,N_c\}$,

$$(\mathbf{F}_{\texttt{energy}})_{j,k}(\Psi(t,\mathbf{x}),x,y)=\left\{\begin{array}{l}\texttt{FFTNonlinearity\_energy}(\Psi(t,\mathbf{x}),x,y,\xi_x,\xi_y)\text{ if }j=k\\0\text{ if }j\neq k\end{array}\right.$$

If `FFTNonlinearity_energy` is a cell array of function in

$$\mathcal{C}_{N_c,N_c}\{\mathbb{F}(\mathcal{C}_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\},\mathcal{M}_{N_y,N_x}(\mathbb{R})^2,\mathcal{M}_{N_y,N_x}(\mathbb{C})^2;\mathcal{M}_{N_y,N_x}(\mathbb{C}))\}$$

then the nonlocal nonlinear energy operator is defined by

$$(\mathbf{F}_{\texttt{energy}})_{j,k}(\Psi(t,\mathbf{x}),x,y)=\texttt{FFTNonlinearity\_energy}\{j,k\}(\Psi(t,\mathbf{x}),x,y,\xi_x,\xi_y)$$

for $j,k\in\{1,...,N_c\}$. The default argument is `Cubic_energy2d`

$$(\mathbf{F}_{\texttt{energy}})_{j,k}(\Psi(t,\mathbf{x}),x,y)=\left\{\begin{array}{l}\frac{1}{2}|\psi_j(t,\mathbf{x})|^2\text{ if }j=k\\0\text{ if }j\neq k\end{array}\right.$$

A standard example of nonlocal nonlinearity comes from the dipolar interaction. This interaction is usually described in 3d, however, a reduction to the 2d case is possible. The dipolar interaction in the 2d case with a single component ψ is given by (see [18])

$$F(\psi(t,\mathbf{x}),\mathbf{x}) = \mathcal{F}^{-1}\left(\left(2-3\sqrt{\pi}|\boldsymbol{\xi}|e^{|\boldsymbol{\xi}|^2}\mathrm{erfc}(|\boldsymbol{\xi}|)\right)\mathcal{F}\left(|\psi(t,\mathbf{x})|^2\right)\right),$$

where erfc is the complementary error function[2]. This nonlinearity can be easily implemented by using a Matlab script. This is done in table 3.22 where we define the previous nonlocal nonlinearity which is then added to the `Physics2D` structure by using the `FFTNonlinearity_Var2d.m` function.

```
function Dipolar2d = Example_fftnonlinearity(phi,x,y,fftx,ffty)
square_xi = fftx.^2+ffty.^2;
K = 2-3*sqrt(pi)*sqrt(square_xi).*exp(square_xi).*erfc(sqrt(square_xi));
Dipolar2d = ifft2(K.*fft2(abs(phi).^2));
end
Physics2D = FFTNonlinearity_Var2d(Method, Physics2D,@(phi,x,y,fftx,ffty)
Example_nonlinearity(phi,x,y,fftx,ffty));
```

Table 3.22: An example of application of the `FFTNonlinearity_Var2d` function with the dipolar operator.

[2] `http://www.mathworks.fr/fr/help/matlab/ref/erfc.html`

3.6.5 The gradient functions

```
Physics2D = Gradientx_Var2d(Method, Physics2D, Gradientx, G);
```

Table 3.23: The `Gradientx_Var2d` function.

The gradient functions allow to define the derivation operators $\sum_{j=1}^{d} \mathbf{G}^j(\mathbf{x})\partial_{x_j}$ in the problem by modifying the `Physics2D` structure. Here, we take for example the function `Gradientx_Var2d`, as the other gradient functions work similarly. We remark that we can only define `Gradientx` in 1d, `Gradientx` and `Gradienty` in 2d and `Gradientx`, `Gradienty` and `Gradientz` in 3d. The `Method` and `Physics2D` structures are required arguments. It is possible to include the following optional arguments

- `Gradientx`: Let us provide a function `Gradientx` in

$$\mathbb{F}(\mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C})),$$

then the variable coefficients in front of the gradient are defined as

$$\mathbf{G}^1_{j,k}(x,y) = \begin{cases} \texttt{Gradientx}(x,y) \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}$$

for each $j, k \in \{1, ..., N_c\}$. If `Gradientx` is a cell array of functions in

$$\mathcal{C}_{N_c,N_c}\{\mathbb{F}(\mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))\}$$

then the variable coefficients are

$$\mathbf{G}^1_{j,k}(x,y) = \texttt{Gardientx}\{j,k\}(x,y)$$

for $j, k \in \{1, ..., N_c\}$. The default argument is the part of the rotational operator corresponding to

$$\mathbf{G}^1_{j,k}(x,y) = \begin{cases} i\Omega y \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}$$

for the `Gradientx_Var2d` function, where Ω is to the rotational speed defined in the `Physics2D` structure, i.e. `Omega` ($\in \mathbb{R}$). In the case of the `Gradienty_Var2d` function, we have

$$\mathbf{G}^2_{j,k}(x,y) = \begin{cases} -i\Omega x \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}$$

Note that in the 1d case, the default argument is

$$\mathbf{G}^1_{j,k}(x) = 0.$$

In the 3d situation, the default operator is the following rotational operator

$$\mathbf{G}^1_{j,k}(x,y,z) = \begin{cases} i(\mathbf{\Omega}_3 y - \mathbf{\Omega}_2 z) \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}$$

$$\mathbf{G}^2_{j,k}(x,y,z) = \begin{cases} i(\mathbf{\Omega}_1 z - \mathbf{\Omega}_3 x) \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}$$

$$\mathbf{G}^3_{j,k}(x,y,z) = \begin{cases} i(\mathbf{\Omega}_2 x - \mathbf{\Omega}_1 y) \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}$$

where $\mathbf{\Omega}$ corresponds to the rotation vector defined in the `Physics3D` structure, i.e. `Omega` ($\in \mathcal{M}_{1,3}(\mathbb{R})$).

- `G` ($\mathcal{M}_{N_c,N_c}(\mathbb{C})$, `ones(N_c)`) is a variable that multiplies the gradient operator element by element
$$\mathbf{G}^1_{j,k}(x,y) = \texttt{G}(j,k)\texttt{Gradientx}\{j,k\}(x,y)$$
for $j,k \in \{1,...,N_c\}$.

An interesting case of coupling between Gross-Pitaevskii equations is the Rashba coupling. For example, in the case of a system of two Gross-Pitaevskii equations with a quadratic potential, a coupled cubic nonlinearity and a Rashba coupling, we obtain the following equations

$$\begin{cases} i\partial_t\psi_1 = \left[-\frac{1}{2}\Delta + V(\mathbf{x}) + (\beta_{11}|\psi_1|^2 + \beta_{12}|\psi_2|^2)\right]\psi_1 - \kappa\left(i\partial_x + \partial_y\right)\psi_2 \\ i\partial_t\psi_2 = \left[-\frac{1}{2}\Delta + V(\mathbf{x}) + (\beta_{22}|\psi_2|^2 + \beta_{12}|\psi_1|^2)\right]\psi_2 - \kappa\left(i\partial_x - \partial_y\right)\psi_1 \end{cases}$$

where $V(\mathbf{x}) = \frac{1}{2}\left(x^2+y^2\right)$ is the quadratic potential, β_{jk} are the interactions constants and κ is the intensity of the Rashba coupling. For defining the operator effect, we have to set the derivation operators such that the gradient operators are

$$\begin{aligned} \mathbf{G}^1(x,y) &= \begin{pmatrix} 0 & -i\kappa \\ -i\kappa & 0 \end{pmatrix}, \\ \mathbf{G}^2(x,y) &= \begin{pmatrix} 0 & -\kappa \\ \kappa & 0 \end{pmatrix}. \end{aligned}$$

In GPELab, we thus have to create a cell array of functions and use the `Gradientx_Var2d` and `Gradienty_Var2d` functions like in Table 3.24 to add the Rashba coupling to a system of two Gross-Pitaevskii equations.

```
function Gradx = Example_gradientx(Kappa)
Gradx = cell(2);
Gradx {1,1} = @(x,y) 0;
Gradx {1,2} = @(x,y) -1i*Kappa;
Gradx {2,1} = @(x,y) -1i*Kappa;
Gradx {2,2} = @(x,y) 0;
end
function Grady = Example_gradienty(Kappa)
Grady = cell(2);
Grady {1,1} = @(x,y) 0;
Grady {1,2} = @(x,y) -Kappa;
Grady {2,1} = @(x,y) Kappa;
Grady {2,2} = @(x,y) 0;
end
Kappa = 1;
Physics2D = Gradientx_Var2d(Method, Physics2D, Example_gradientx(Kappa));
Physics2D = Gradienty_Var2d(Method, Physics2D, Example_gradienty(Kappa));
```

Table 3.24: An example to use the `Gradientx_Var2d.m` and `Gradienty_Var2d` functions.

3.6.6 The `InitialData_Var2d` function

```
Phi_0 = InitialData_Var2d(Method, Geometry2D, Physics2D, InitialData_Choice, X0,
Y0, gamma_x, gamma_y);
```

Table 3.25: The `InitialData_Var2d` function.

The `InitialData_Var2d` function builds an initial wave function (i.e. $\Psi_0(\mathbf{x})$) for the simulations. Already defined initial data corresponding to the Thomas-Fermi approximation or the centered Gaussian are existing in GPELab. Note that the user can also create its own initial wave function without using this function. The `Method`, `Geometry2D` and `Physics2D` structures are needed arguments for the function. Optional arguments are

- `InitialData_Choice` ($\mathbb{N}$,1) is a variable that must be either 1 if one uses centered gaussians or 2 for Thomas-Fermi approximations as initial data. The option 3 allows to use the imaginary-time method with the BESP scheme to compute ground-states for each component where the operators are restricted to their diagonal parts (i.e. the components are decoupled).
- `X0`,`Y0` ($\mathcal{M}_{1,N_c}(\mathbb{R})$, 0) are variables corresponding to the coordinates of the center of the gaussian or Thomas-Fermi approximation as initial data. We note that, in the 1d case, we only have to define `X0` and, in the 3d case, `Z0` is required.
- `gamma_x`, `gamma_y` ($\mathbb{R}$, 1) are variables corresponding to the parameters of the centered gaussian. We note that, in the 1d case, we only have to define `gamma_x` and, in the 3d case, we have to add `gamma_z`.

For example, if we want to compute a Thomas-Fermi approximation for initial data, we proceed as in table 3.26.

```
InitialData_Choice = 2;
Phi_0 = InitialData_Var2d(Method, Geometry2D, Physics2D, InitialData_Choice);
```

Table 3.26: An example to use the `InitialData_Var2d` function.

3.7 Launching the simulation, setting the outputs and informations on the computation

3.7.1 The `OutputsINI_Var2d` function

```
Outputs = OutputsINI_Var2d(Method, Evo_outputs, save, userdef_outputs,
userdef_outputs_names, globaluserdef_outputs, globaluserdef_outputs_names);
```

Table 3.27: The `OutputsINI_Var2d` function

The `OutputsINI_Var2d` function initializes the outputs of a simulation by building the `Outputs` *structure*. Outputs are scalar values computed using each component of the wave function during the simulation. In GPELab, the predefined outputs are: the modulus of the wave function at the center of the domain, the root mean-square in each direction, the energy, the chemical potential and the angular momentum. More outputs can be computed by using user-defined functions. The outputs are computed and displayed in the command window at each iteration incremented by the value of the `Evo_outputs` variable. They are also stored after the simulation in the `Outputs`

structure (see the `GPELab2d` function, Section 3.7.6, page 60). The `Method` structure is a required argument of this function. Concerning the optional arguments, we have

- `Evo_outputs` ($\mathbb{N}$, `5`) is a variable corresponding to the number of iterations between each computation of the outputs. It must be smaller or equal to `Evo` from the `Print_Var2d` (see Section 3.7.3, page 58)

- `save` ($\mathbb{N}$,`0`) is a variable corresponding to the choice of wherever or not to save the computed wave functions in the output structure every `Evo`. It must be either `1` if one saves the wave functions or `0` otherwise.

- `userdef_outputs` is a cell array of functions in

$$\mathcal{C}_{1,n^{\texttt{Lout}}}\{\mathbb{F}(\mathcal{M}_{N_y,N_x}(\mathbb{C}),\mathcal{M}_{N_y,N_x}(\mathbb{R})^2,\mathcal{M}_{N_y,N_x}(\mathbb{C})^2;\mathbb{R})\}$$

  that allows the user to define itself $n^{\texttt{Lout}}$ relevant physical output quantities. These quantities are computed through $n^{\texttt{Lout}}$ Matlab functions that the user must write himself under the form

$$(\psi_\ell(t,\mathbf{x}),x,y,\xi_x,\xi_y)\rightarrow \texttt{userdef\_outputs}\{j\}(\psi_\ell(t,\mathbf{x}),x,y,\xi_x,\xi_y)$$

  for $j\in\{1,...,n^{\texttt{Lout}}\}$, where ξ_x and ξ_y are the discrete Fourier frequencies in the x- and y-directions. We remark that `userdef_outputs` must have $(\psi_\ell(t,\mathbf{x}),x,y,\xi_x,\xi_y)$ as arguments only in the case where a spectral scheme is used. Otherwise, the arguments are $(\psi_\ell(t,\mathbf{x}),x,y)$. By default, there is no other output computed than the predefined ones.

- `userdef_outputs_names` ($\mathcal{C}_{1,n^{\texttt{Lout}}}\{\mathbb{S}\}$,`'User defined function'`) is a cell array of character strings, where the j-th component corresponds to the name displayed in the command window of the j-th physical quantity appearing in `userdef_outputs`.

- `globaluserdef_outputs` is a cell array of functions in

$$\mathcal{C}_{1,n^{\texttt{Gout}}}\{\mathbb{F}(\mathcal{C}_{1,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\},\mathcal{M}_{N_y,N_x}(\mathbb{R})^2,\mathcal{M}_{N_y,N_x}(\mathbb{C})^2;\mathbb{R})\}$$

  that defines $n^{\texttt{Gout}}$ relevant physical output quantities. We remark that, compared with the previous variable `userdef_outputs`, these physical quantities can be defined through expressions involving the full wave function Ψ and not only its one-by-one components. They are evaluated through $n^{\texttt{Gout}}$ Matlab functions that must be of the form

$$(\Psi(t,\mathbf{x}),x,y,\xi_x,\xi_y)\rightarrow \texttt{globaluserdef\_outputs}\{j\}(\Psi(t,\mathbf{x}),x,y,\xi_x,\xi_y)$$

  for $j\in\{1,...,n^{\texttt{Gout}}\}$, where ξ_x and ξ_y are the discrete Fourier frequencies in the x- and y-directions. We remark that `globaluserdef_outputs` must have $(\psi_\ell(t,\mathbf{x}),x,y,\xi_x,\xi_y)$ as arguments only in the case where a spectral scheme is used. Otherwise, the arguments are $(\psi_\ell(t,\mathbf{x}),x,y)$. By default, there is no predefined output quantity in GPELab which means that the user must define its own functions.

- `globaluserdef_outputs_names` ($\mathcal{C}_{1,n^{\texttt{Gout}}}\{\mathbb{S}\}$,`'User defined function'`) is a variable that has the same role as `userdef_outputs_names` but for `globaluserdef_outputs`.

Let us assume that we launch a simulation that ends after N_{iter} iterations. Therefore, the outputs are computed

$$N_{\text{out}}=\text{Int}\left[\left[\frac{N_{\text{iter}}}{\texttt{Evo\_outputs}}\right]\right]+1$$

times at $t_k:=k$ `Evo_outputs` Δt, $1\leq k\leq N_{\text{out}}$, and $t_{N_{\text{out}}+1}=N_{\text{iter}}\Delta t$. In the above equation, Int$[[r]]$ designates the integer part of a real-valued number r. The resulting `Outputs` structure has the following variables

- `Solution` ($\mathcal{C}_{1,N_{\text{out}}}\{\mathcal{C}_{1,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\}\}$) contain the computed solutions for times t_k if `save` = `1`.

- `phi_abs_0` ($\mathcal{C}_{1,N_c}\{\mathcal{M}_{1,N_{\text{out}}}(\mathbb{R})\}$) is a cell array of vectors that contains the values of the square modulus of each wave function ψ_ℓ at the center of the domain for times t_k

$$\texttt{phi\_abs\_0}\{\ell\}(k) = \left|\psi_\ell\left(t_k, \frac{\texttt{x\_max}+\texttt{x\_min}}{2}, \frac{\texttt{y\_max}+\texttt{y\_min}}{2}\right)\right|^2$$

  where `x_max`, `x_min`, `y_max` and `y_min` have been defined by the `Geometry2D_Var2d` function (see subsection 3.5.2, page 43).

- `x_rms`, `y_rms` ($\mathcal{C}_{1,N_c}\{\mathcal{M}_{1,N_{\text{out}}}(\mathbb{R})\}$) is a cell array of vectors containing the values of the root mean-square of each wave function ψ_ℓ with respect to the x- and y-directions. They are computed by

$$\texttt{x\_rms}\{\ell\}(k) = \left(\int_{\mathcal{O}} x^2|\psi_\ell(t_k,x,y)|^2 dxdy\right)^{1/2}$$

  and

$$\texttt{y\_rms}\{\ell\}(k) = \left(\int_{\mathcal{O}} y^2|\psi_\ell(t_k,x,y)|^2 dxdy\right)^{1/2}.$$

- `Energy` ($\mathcal{C}_{1,N_c}\{\mathcal{M}_{1,N_{\text{out}}}(\mathbb{R})\}$) is a cell array of vectors such that

$$\texttt{Energy}\{\ell\}(k) = \mathbf{E}_\ell(\Psi)(t_k)$$

  (see Equation (3.2), page 44).

- `Chemical_potential` ($\mathcal{C}_{1,N_c}\{\mathcal{M}_{1,N_{\text{out}}}(\mathbb{R})\}$) is a cell array of vectors such that

$$\texttt{Chemical\_potential}\{\ell\}(k) = \boldsymbol{\mu}_\ell(\Psi)(t_k)$$

  (see Equation (3.3), page 44).

- `User_defined_local` ($\mathcal{C}_{1,n^{\texttt{Lout}}}\{\mathcal{M}_{1,N_{\text{out}}}(\mathbb{R})\}$) are the user defined functions `userdef_outputs`.

- `User_defined_global` ($\mathcal{C}_{1,n^{\texttt{Gout}}}\{\mathcal{M}_{1,N_{\text{out}}}(\mathbb{R})\}$) are the user defined functions `globaluserdef_outputs`.

For example, if one wants to compute the L^2-norm of the gradient of each component of a Bose-Einstein condensate on the computational domain

$$\texttt{Grad\_norm} = \int_{\mathcal{O}} |\nabla\psi(t,\mathbf{x})|^2 dxdy,$$

one has to first define a function that computes the L^2-norm of the gradient by using a FFT and then create the `Outputs` structure by using the `OutputsINI_Var2d` function with the function as argument. This is done in table 3.28.

```
function Grad_norm = Example_outputs(Geometry2D,phi,x,y,fftx,ffty)
Grad_x = ifft2(1i*fftx.*fft2(phi));
Grad_y = ifft2(1i*ffty.*fft2(phi));
Grad_x_norm = sqrt((Geometry2D.dx*Geometry2D.dy)*sum(sum(abs(Grad_x).^2)));
Grad_y_norm = sqrt((Geometry2D.dx*Geometry2D.dy)*sum(sum(abs(Grad_y).^2)));
Grad_norm = Grad_x_norm + Grad_y_norm;
end
Outputs = OutputsINI_Var2d(Method, 1, @(phi,x,y,fftx,ffty)
Example_outputs(Geometry2D,phi,x,y,fftx,ffty));
```

Table 3.28: An example to use the `OutputsINI_Var2d` function for a user-defined function with a single-component.

However, if one wants to compute the root mean-square of the sum of two components

$$\texttt{RMS} = \int_{\mathcal{O}} (|x|^2 + |y|^2)|\psi_1(t,\mathbf{x}) + \psi_2(t,\mathbf{x})|^2 dxdy,$$

one has to proceed differently because a function computing the root mean-square of the sum of two components takes the cell vector of the two wave functions as argument. Therefore, we have to use `globaluserdef_outputs`. We show how to do this in table 3.29.

```
function RMS = Example_outputs(Geometry2D,Phi,x,y,fftx,ffty)
RMS_local = (x.^2+y.^2).*abs(Phi{1}+Phi{2}).^2;
RMS = sqrt((Geometry2D.dx*Geometry2D.dy)*sum(sum(abs(RMS_local).^2)));
end
Outputs = OutputsINI_Var2d(Method, 1, [],[],@(Phi,x,y,fftx,ffty)
Example_outputs(Geometry2D,Phi,x,y,fftx,ffty));
```

Table 3.29: An example to use the `OutputsINI_Var2d` function for a user-defined function with multi-components.

3.7.2 The `Continuation_Var2d` function

```
Continuation = Continuation_Var2d(Coefficient_name,Coefficient, Continuation);
```

Table 3.30: The `Continuation_Var2d` function.

The `Continuation_Var2d` function allows to define what continuation means in GPELab when you want to compute a stationary state. By continuation, we mean that the computation is initialized by an initial state, and then doing loops on a chosen parameter $p \in \mathcal{C}_{1,n^{\texttt{cont}}}\{\mathbb{C}\}$ or $\mathcal{C}_{1,n^{\texttt{cont}}}\{\mathcal{M}_{N_c,N_c}(\mathbb{C})\}$, where $n^{\texttt{cont}}$ is the total number of iterations. At each iteration k, the new initial state is taken to be equal to the stationary state computed at the previous iteration $k-1$. The continuation continues until the end of the loop and gives a stationary state solution at the final iteration $n^{\texttt{cont}}$. This way of proceeding is expected to provide at the end of the calculations a ground state solution through small increments of the parameter p at the iteration k.

In addition, the `Continuation_Var2d` function allows to define a multi-parameter continuation path. More precisely, let us assume that the `Continuation_Var2d` function is called successively $n^{\texttt{param}}$ times with $n^{\texttt{param}}$ different continuation parameters $p_j \in \mathcal{C}_{1,n^{\texttt{cont}}}\{\mathbb{C}\}$ or $\mathcal{C}_{1,n^{\texttt{cont}}}\{\mathcal{M}_{N_c,N_c}(\mathbb{C})\}$, $j \in n^{\texttt{param}}$, where $n^{\texttt{cont}}$ must be the same for all p_j. Then, the multi-parameter continuation path is defined by

$$\texttt{path} \in (\mathbb{R}^{\texttt{param}})^{n^{\texttt{cont}}},$$

with $\texttt{path}_k := (p_1\{k\}, ..., p_{n^{\texttt{param}}}\{k\})$ and $1 \leq k \leq n^{\texttt{cont}}$.

Concerning the Matlab function itself, the following arguments are needed

- `Coefficient_name` ($\mathbb{S}$) identifies the parameter on which the continuation applies. This parameter can be one of the following arguments
  - `'Delta'`, `'Beta'` or `'Omega'`, (see Section 3.6.1, page 44),
  - `'GPotential'`, where `G` is the matrix associated to the potential (see Section 3.6.2, page 45),
  - `'GNonlinearity'`, with `G` is the matrix related to the nonlinearity (see Section 3.6.3, page 46),
  - `'GFFTNonlinearity'`, `G` is the matrix for the FFT-based nonlinearity (see Section 3.6.4, page 48),
  - or `'GGradientx'` (or `'GGradienty'`) if one wants to change the `G` matrix for the gradients (see Section 3.6.5, page 51).
- `Coefficient` ($\mathcal{C}_{1,n^{\texttt{cont}}}\{\mathbb{C}\}$ or $\mathcal{C}_{1,n^{\texttt{cont}}}\{\mathcal{M}_{N_c,N_c}(\mathbb{C})\}$) is the p variable containing the values of the parameter that will be changed during the continuation method. For a scalar parameter (`'Delta'`, `'Beta'` or `'Omega'`), it must be a cell array of scalars. In the case of a matrix parameters (`'GPotential'`, `'GNonlinearity'`, `'GFFTNonlinearity'`, `'GGradientx'` or `'GGradienty'`), it must be a cell array of matrices.

We have the following optional argument

- `Continuation` ($\mathcal{S}$, `[]`): the continuation structure must be taken as an argument if one wants to add parameters in the continuation method for multi-parameter paths.

A simple example for the continuation method is to increment a single parameter. We know that, if one wants to compute the ground state of a fast rotating Bose-Einstein condensate, one should do this by a continuation method on the rotation speed. In GPELab, the rotation speed is denoted by Ω, which corresponds to the `Omega` parameter in the `Physics2D` structure (see 3.6.1, page 44). Therefore, a continuation on Ω, where one wants to increase it from 1 to 5 for example, could be defined like in Table 3.31, where we have set the evolution of the parameter as $\Omega = 1, 2, 3, 3.5, 4, 4.5, 5$.

```
Continuation = Continuation_Var2d('Omega',{1,2,3,3.5,4,4.5,5});
```

Table 3.31: An example for using the `Continuation_Var2d` function when the parameter is a scalar.

Another interesting case is when there is a coupling between multiple components. In this scenario, a continuation method could be interesting to smoothly couple the two components. For example, if the two components have coupled nonlinearities, one can use a continuation method on the `G` matrix of the nonlinearity to increment the extradiagonal terms. This is done in Table 3.33, where we have defined a cell vector of `G` matrices with increasing extradiagonal terms from 0 to 1 in 5 steps.

```
GNL1 = [1,0;0,1];
GNL2 = [0,1;1,0];
for i=1:5
GNL{i} = GNL1 + (i/5)*GNL2
end
Continuation = Continuation_Var2d('GNonlinearity',GNL);
```

Table 3.32: An example for using the `Continuation_Var2d` function when the parameter is a matrix.

Finally, let us consider, for example, the case of a coupled fast rotating two-components Bose-Einstein condensate. We can proceed by using a continuation simultaneously on the `G` matrix for the coupling and on the `Omega` parameter. Thus, we first have to create the `continuation` structure for a parameter and then update it by inserting a second parameter. This is done in Table 3.33, where we first define the continuation on the `Omega` parameter and then we add the `G` matrix of the nonlinearity.

```
Continuation = Continuation_Var2d('Omega',{1,2,3,3.5,4});
GNL1 = [1,0;0,1];
GNL2 = [0,1;1,0];
for i=1:5
GNL{i} = GNL1 + (i/5)*GNL2
end
Continuation = Continuation_Var2d('GNonlinearity',GNL,Continuation);
```

Table 3.33: An example to use the `Continuation_Var2d` function for 2 parameters.

3.7.3 The `Print_Var2d` function

```
Print = Print_Var2d(Printing,Evo,Draw);
```

Table 3.34: The `Print_Var2d` function.

The `Print_Var2d` function builds the `Print` *structure*. The aim is to provide to the program the printing informations displayed during the computation. The following optional arguments are

- `Printing` (ℕ,1) is a variable equals to 1 for printing informations during the computation and 0 otherwise.
- `Evo` (ℕ,5) is a variable corresponding to the number of iterations between each displayed information (including drawing some figures). It must be bigger or equal to `Evo_outputs` from the `OutputsINI_Var2d` function (see Section 3.7.1, page 53).
- `Draw` (ℕ,1) is a variable equal to 1 if the modulus and the phase of the wave functions are drawn during the simulation and 0 if not.

For example, if one wants to print informations every 10 iterations but does not want to slow the program by drawing the wave function's modulus and angle, then one can define the `Print` structure by using the `Print_Var2d` function like in table 3.35.

```
Printing = 1;
Evo = 10;
Draw = 0;
Print = Print_Var2d(Printing,Evo,Draw);
```

Table 3.35: An example of use for the `Print_Var2d` function.

3.7.4 The `Figure_Var2d` function

```
Figure = Figure_Var2d(map);
```

Table 3.36: The `Figure_Var2d` function.

The `Figure_Var2d` function builds the `Figure` *structure* which contains informations needed to draw figures in 2d. We have the following optional argument

- `map` ($\mathbb{S}$,`'jet'`) is a variable corresponding to the colormap of the figures. It must be either `'jet'`, `'hsv'`, `'hot'`, `'cool'`, `'spring'`, `'summer'`, `'autumn'`, `'winter'`, `'gray'`, `'bone'`, `'copper'`, `'pink'` or `'lines'` (see the Matlab documentation for further informations about colormap[3]).

If one wants to draw figures using the `'hot'` colormap for example, then it can be done by defining the `Figure` structure as in table 3.39.

```
map = 'hot';
Figure = Figure_Var2d(map);
```

Table 3.37: An example of how to use the `Figure_Var2d` function.

3.7.5 The `Figure_Var3d` function

```
Figure = Figure_Var3d(view,iso,alpha,aspect,Sx,Sy,Sz,map);
```

Table 3.38: The `Figure_Var3d` function.

The `Figure_Var3d` function builds the `Figure` *structure* which contains informations needed to display the wave functions in 3d. The square modulus of the wave function is drawn in 3d by using an isovalues surface, and the phase of the wave function which is displayed by using some slices along the x-, y- and z-directions. The function has the following optional arguments

- `view` ($\mathbb{N}$ or $\mathcal{M}_{1,3}(\mathbb{R})$,`3`) is a variable corresponding to the viewing angle. For more informations see the `view`[4] Matlab function.
- `iso` ($\mathbb{R}^+$,`0.001`) is the isovalue chosen for drawing the modulus of the wave functions.
- `alpha` ($\mathbb{R}^+$,`0.6`) is the transparency of the isovalue surface [5].

[3] `http://www.mathworks.fr/fr/help/matlab/ref/colormap.html`
[4] `http://www.mathworks.fr/fr/help/matlab/ref/view.html`
[5] `http://www.mathworks.fr/fr/help/matlab/ref/alpha.html`

- `aspect` ($\mathcal{M}_{1,3}(\mathbb{R})$,`[1,1,1]`) is a variable corresponding to the data aspect ratio. For more informations see the `daspect`[6] Matlab function.
- `Sx` ($\mathbb{R}$,`0`) (respectively `Sy`, `Sz`) is a coordinate on the x-axis where an orthogonal slice to the x-axis (respectively y-axis, z-axis) will be drawn.
- `map` ($\mathbb{S}$,`'jet'`) is a variable corresponding to the colormap of the figures. It must be either `'jet'`, `'hsv'`, `'hot'`, `'cool'`, `'spring'`, `'summer'`, `'autumn'`, `'winter'`, `'gray'`, `'bone'`, `'copper'`, `'pink'` or `'lines'`.

For example, suppose that one wants to compute a rotating Bose-Einstein condensate, where the rotation is along the z-direction, and then to see the condensate along the z-axis to get a view from above. Thus, the view angle, as set by Matlab, should be $[0, 0, 1]$. Moreover, it would be convenient to set a slightly larger isovalue than the default one for the surface of the modulus of the wave function, say 0.01, to better distinguish the vortices. Then the `Figure` structure should be defined the same way as in table 3.39.

```
view = [0,0,1];
iso = 0.01;
Figure = Figure_Var3d(view,iso);
```

Table 3.39: An example to use the `Figure_Var3d` function.

3.7.6 The `GPELab2d` function

```
[Phi,Outputs] = GPELab2d(Phi_0,Method,Geometry2D,Physics2D,Outputs,Continuation,
Print,Figure);
```

Table 3.40: The `GPELab2d` function.

The `GPELab2d` function is the main function to launch a full simulation with respect to the given configuration. The output arguments are

- `Phi` ($\mathcal{C}_{1,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\}$), the wave functions computed at the final time (with respect to a stopping criterion for a stationary state or a fixed time for a dynamical computation)
- and `Outputs` ($\mathcal{S}$) which is a structure that contains all the outputs computed during the simulation.

The initial data `Phi_0` ($\mathcal{C}_{1,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\}$) and the `Method`, `Geometry2D`, `Physics2D` and `Outputs` structures are required arguments. The optional arguments that can be considered are the following

- `Continuation` ($\mathcal{S}$,`[]`) is the continuation structure if one wants to use a continuation method.
- `Print` ($\mathcal{S}$,`Print_Var2d`) is the printing structure.
- `Figure` ($\mathcal{S}$,`Figure_Var2d.`) is the structure that fixes the parameters for drawing the figures.

We report in Table 3.41 a model example of use of the `GPELab2d` function.

[6] `http://www.mathworks.fr/fr/help/matlab/ref/daspect.html`

```
[Phi,Outputs] = GPELab2d(Phi_0,Method,Geometry2D,Physics2D,Outputs);
```

Table 3.41: An example of call to the GPELab2d function.

3.7.7 The MakeVideo2d function

```
MakeVideo2d(Method,Geometry2D,Outputs, Function,VideoName,Figure);
```

Table 3.42: The MakeVideo2d function.

The MakeVideo2d function creates a movie from the saved wave functions during the computation. It assembles snapshots from each saved component's wave function and creates a video. Due to the fact that Matlab takes snapshots of the figures, the user must be extremely careful when building the movie by not covering the figure during the construction process. This function must be provided with the Method structure, the Geometry2D structure and the Outputs structure. We have the following optional arguments

- Function: If a function Function in

$$\mathbb{F}(\mathcal{M}_{N_y,N_x}(\mathbb{C}),\mathcal{M}_{N_y,N_x}(\mathbb{R})^2;\mathcal{M}_{N_y,N_x}(\mathbb{R}))$$

  is provided, then the snapshots are considered by applying Function to each component

$$(\psi_j(t,\mathbf{x}),x,y)\rightarrow \texttt{Function}(\psi_j(t,\mathbf{x}),x,y)$$

  for $j\in\{1,...,N_c\}$. If Function is a cell array of functions in

$$\mathcal{C}_{1,N_c}\{\mathbb{F}(\mathcal{M}_{N_y,N_x}(\mathbb{C}),\mathcal{M}_{N_y,N_x}(\mathbb{R})^2;\mathcal{M}_{N_y,N_x}(\mathbb{R}))\}$$

  then the snapshot are taken from the following matrix

$$(\psi_j(t,\mathbf{x}),x,y)\rightarrow \texttt{Function}\{j\}(\psi_j(t,\mathbf{x}),x,y)$$

  for $j\in\{1,...,N_c\}$. The default value is @(phi,X,Y)abs(phi).^2.

- VideoName ($\mathbb{S}$ or $\mathcal{C}_{1,N_c}(\mathbb{S})$,'MyVideo') is a variable corresponding to the name of the video of each component. If a single character string is provided, the names of the videos are, for each $j\in\{1,...,N_c\}$, VideoName followed by j. If a cell vector of character strings is given, the names of the videos are, for each $j\in\{1,...,N_c\}$, VideoName$\{j\}$.

- Figure ($\mathcal{S}$,Figure_Var2d) is the figure structure.

If one wants to make a video of the angle of a wave function after a simulation, then one should proceed as in table 3.43.

```
Function = @(phi,X,Y) angle(phi);
MakeVideo2d(Method,Geometry2D,Outputs, Function);
```

Table 3.43: An example to use the MakeVideo2d function.

Chapter 4

Examples of simulations for stationary solutions

This Section is devoted to different physical examples treated by GPELab for computing stationary states. The user has a direct access to the source codes in the subdirectories `1d/examples`, `2d/examples` and `3d/examples` of GPELab. We will be happy to add some of your examples that are different from the ones given here.

4.1 Ground state of a 1d Gross-Pitaevskii equation with an optical potential and a cubic nonlinearity

We now show how to compute the ground state of a Gross-Pitaevskii equation with an optical potential and a cubic nonlinearity in 1d. This example comes from [21]. First, we have to build the `Method` and `Geometry1D` structures. We use the BESP scheme and a time step equal to 5.10^{-2}, with a spatial grid of $2^{10}+1$ points in the interval $[-16, 16]$. Moreover, we set the stopping criterion to 10^{-8}. We refer to Table 4.1 below for the corresponding GPELab code.

```
Computation = 'Ground';
Ncomponents = 1;
Type = 'BESP';
Deltat = 5e-2;
Stop_time = [];
Stop_crit = 1e-8;
Method = Method_Var1d(Computation,Ncomponents, Type, Deltat, Stop_time ,
Stop_crit);
xmin = -16;
xmax = 16;
Nx = 2^10+1;
Geometry1D = Geometry1D_Var1d(xmin,xmax, Nx);
```

Table 4.1: Defining the `Method` and `Geometry1D` structures.

We now define the physics of the problem to compute the ground state of the following Gross-Pitaevskii equation

$$i\partial_t \psi(t,x) = \frac{1}{2}\Delta\psi(t,x) + \left(\frac{|x|^2}{2} + 25\sin^2\left(\frac{\pi x}{4}\right)\right)\psi(t,x) + \beta|\psi(t,x)|^2\psi(t,x),$$

with $\beta = 250$. Let us remark that since we consider the cubic case, this is the default nonlinearity in GPELab and we therefore do not have to define it. We have

- to build the `Physics1D` structure with the desired coefficients,
- to define and add the optical potential,
- and finally to add the default nonlinear operator to the physics of the problem.

We follow this construction in Table 4.2 where we directly define the optical potential in the arguments of the `Potential_Var1d` function.

```
Delta = 0.5;
Beta = 250;
Physics1D = Physics1D_Var1d(Method,Delta,Beta);
Physics1D = Potential_Var1d(Method, Physics1D, @(x) x.^2 /2 + 25*sin(pi*x/4).^2);
Physics1D = Nonlinearity_Var1d(Method, Physics1D);
```

Table 4.2: Creating the `Physics1D` structures.

We then set the initial function to use for the computation. We choose a centered gaussian by setting the `InitialData_choice` to 1 in the `InitialData_Var1d` function, as in Table 4.3.

```
InitialData_choice = 1 ;
Phi_0 = InitialData_Var1d(Method, Geometry1D, Physics1D,InitialData_choice);
```

Table 4.3: Building the initial data.

We finally set the outputs and the printing informations and then launch the simulation. We choose to print informations in the command window every 15 iterations and to draw a figure of the square of the modulus of the solution. This is done in Table 4.4.

```
Outputs = OutputsINI_Var1d(Method);
Printing = 1;
Evo = 15;
Draw = 1;
Print = Print_Var1d(Printing,Evo,Draw);
[Phi,Outputs]= GPELab1d(Phi_0,Method,Geometry1D,Physics1D,Outputs,[],Print);
```

Table 4.4: Setting the outputs, the `Print` structure and launching the computation.

At the end of the simulation, we obtain the following informations on the command window of Matlab.

```
--------------------
Iteration 249 on 1000000
--Outputs of component 1--------
Square at the origin:  0.15144001489396
x-radius mean square:  3.36090790724700
Energy:  26.08386211010322
Chemical potential:  38.06922564389800
Energy evolution:  0.00000000000002
--------------------
CPU time:  7.36
»
```

The solution as also been printed out during the simulation. At the end, we obtain the solution given on Figure 4.1.

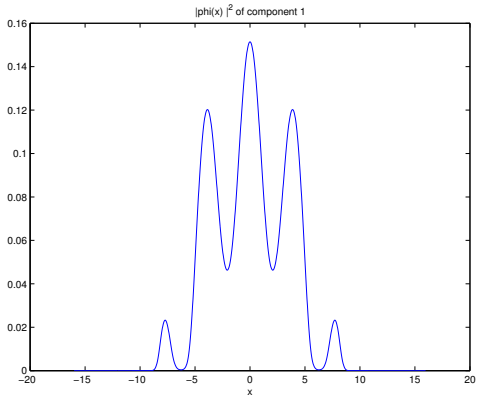


Figure 4.1: Modulus of the ground state.

Moreover, we can draw the evolution of the energy and the chemical potential during the computation by plotting `Outputs.Energy{1}` and `Outputs.Chemical_potential{1}` (see Figure 4.2).

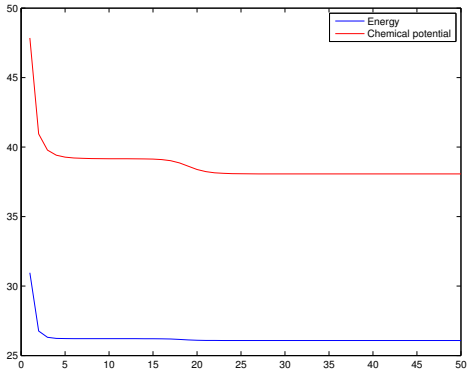


Figure 4.2: Evolution of the energy and the chemical potential during the computation.

4.2 Ground state of a system of 1d Gross-Pitaevskii equations with a quadratic potential and a Josephson junction

We would like to produce the numerical results in [17] where a system of two Gross-Pitaevskii equations coupled with a Josephson junction is considered. The system of Gross-Pitaevskii equations is the following

$$\begin{cases} i\partial_t\psi_1 = \left[-\frac{1}{2}\Delta + V(\mathbf{x}) + \delta + (\beta_{11}|\psi_1|^2 + \beta_{12}|\psi_2|^2)\right]\psi_1 + \lambda\psi_2, \\ i\partial_t\psi_2 = \left[-\frac{1}{2}\Delta + V(\mathbf{x}) + (\beta_{22}|\psi_2|^2 + \beta_{12}|\psi_1|^2)\right]\psi_2 + \lambda\psi_1, \end{cases} \tag{4.1}$$

where δ is the detuning constant for the Raman transition, β_{jk} are the interactions constants and λ is the effective Rabi frequency. We can already identify the operators that we will need to define: the potential operator with the quadratic potential, the detuning constant, the effective Rabi frequency and the nonlinear operator with the cubic nonlinearities. We already have defined these operators by using the script in tables 3.19 and 3.21 for the 2d case. We create two similar scripts for our case in tables 4.5 and 4.6.

```
function P = quadratic_potential_Josephson1d(Detuning_constant,Rabi_frequency)
P = cell(2);
P{1,1} = @(x) (1/2)*x.^2+Detuning_constant;
P{1,2} = @(x) Rabi_frequency;
P{2,1} = @(x) Rabi_frequency;
P{2,2} = @(x) (1/2)*x.^2;
end
```

Table 4.5: The potential function used for the Josephson junction.

```
function NL = Josephson_Nonlinearity(Beta_11,Beta_22,Beta_12)
NL = cell(2);
NL{1,1} = @(Phi,x) Beta_11*abs(Phi{1}).^2 + Beta_12*abs(Phi{2}).^2;
NL{2,2} = @(Phi,x) Beta_22*abs(Phi{2}).^2 + Beta_12*abs(Phi{1}).^2;
NL{1,2} = @(Phi,x) 0;
NL{2,1} = @(Phi,x) 0;
end
```

Table 4.6: The nonlinearity function used for the Josephson junction.

Moreover, we define in Table 4.7 the energy (2.50) (see page 31) which is associated to the nonlinearity.

```
function NLE = Josephson_Nonlinearity_Energy(Beta_11,Beta_22,Beta_12)
NLE = cell(2);
NLE{1,1} = @(Phi,x) (1/2)*Beta_11*abs(Phi{1}).^2 + (1/2)*Beta_12*abs(Phi{2}).^2;
NLE{2,2} = @(Phi,x) (1/2)*Beta_22*abs(Phi{2}).^2 + (1/2)*Beta_12*abs(Phi{1}).^2;
NLE{1,2} = @(Phi,x) 0;
NLE{2,1} = @(Phi,x) 0;
end
```

Table 4.7: The function computing the energy (2.50), page 31, associated to the nonlinearity used for the Josephson junction.

We now show how to compute the ground state of this system of coupled Gross-Pitaevskii equations by using similar parameters as in [17]. First, we have to build the `Method` and `Geometry1D`

structures. We use the BESP scheme and a time step equal to 10^{-1}, with a spatial grid of $2^{10}+1$ points in $[-16, 16]$. Moreover, the stopping criterion is fixed to 10^{-6} (see Table 4.8).

```
Computation = 'Ground';
Ncomponents = 2;
Type = 'BESP';
Deltat = 1e-1;
Stop_time = [];
Stop_crit = 1e-6;
Method = Method_Var1d(Computation,Ncomponents, Type, Deltat, Stop_time ,
Stop_crit);
xmin = -16;
xmax = 16;
Nx = 2^10+1;
Geometry1D = Geometry1D_Var1d(xmin,xmax, Nx);
```

Table 4.8: Building the `Method` and `Geometry1D` structures.

We now define the physics of the problem. We build the `Physics1D` structure according to equation (4.1). Moreover, one wishes to reproduce the numerical results in [17] with the set of physical parameters

$$\begin{aligned} \lambda &= -1 \qquad \delta = 0 \\ \beta &= 500 \qquad \beta_{11} = \beta \\ \beta_{12} &= 0.94\beta \qquad \beta_{22} = 0.97\beta \end{aligned}$$

We proceed as in table 4.9 where we use the two operators defined in Tables 4.5 and 4.6, and the function defining the energy associated to the nonlinearity in Table 4.7.

```
Delta = 0.5;
Rabi_frequency = -1;
Detuning_constant = 0;
Beta = 500;
Beta_11 = 1;
Beta_12 = 0.94;
Beta_22 = 0.97;
Physics1D = Physics1D_Var1d(Method,Delta,Beta);
Physics1D = Potential_Var1d(Method, Physics1D,...
Josephson_Potential(Detuning_constant,Rabi_frequency));
Physics1D = Nonlinearity_Var1d(Method, Physics1D,...
Josephson_Nonlinearity(Beta_11,Beta_22,Beta_12),[],...
Josephson_Nonlinearity_Energy(Beta_11,Beta_22,Beta_12));
```

Table 4.9: Building the `Physics1D` structure.

We then set the initial function to use for the computation that we choose as a centered gaussian (see Table 4.10).

```
InitialData_choice = 1 ;
Phi_0 = InitialData_Var1d(Method, Geometry1D, Physics1D,InitialData_choice);
```

Table 4.10: Gaussian initial data.

We finally set the outputs and the printing informations and launch the simulation. We display the informations in the command window every 15 iterations. We furthermore draw a figure of the square of the modulus of the solution. The corresponding GPELab code is detailed in Table 4.11.

```
Outputs = OutputsINI_Var1d(Method);
Printing = 1;
Evo = 15;
Draw = 1;
Print = Print_Var1d(Printing,Evo,Draw);
[Phi,Outputs]= GPELab1d(Phi_0,Method,Geometry1D,Physics1D,Outputs,[],Print);
```

Table 4.11: Main GPELab code.

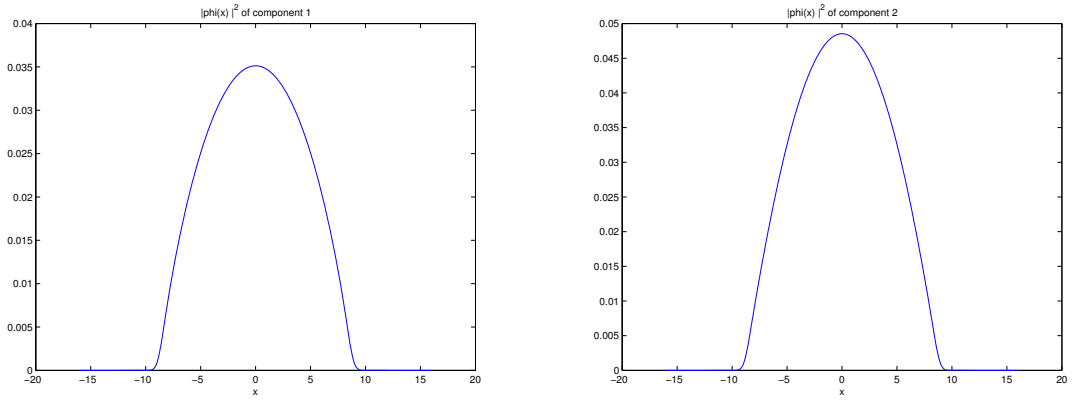
At the end of the simulation, we obtain the informations concerning the ground state in Table 4.12 and the modulus of each component in figure 4.3. Moreover, we can plot the two modulii on the same figure and obtain the same result as in [17] (see Figure 4.4).

```
--------------------
Iteration 164 on 1000000
--Outputs of component 1--------
Square at the origin:  0.03512761887793
x-radius mean square:  2.67309309073190
Energy:  9.97806793424214
Chemical potential:  32.84198217411518
Energy evolution:  0.00000000000000
--Outputs of component 2--------
Square at the origin:  0.04853615539806
x-radius mean square:  2.99189318517412
Energy:  13.16187704973455
Chemical potential:  38.16552799804442
Energy evolution:  0.00000000000000
--------------------
CPU time:  8.28
»
```

Table 4.12: Printed outputs for the 1d Josephson problem.

4.3 Ground state of a 2d Gross-Pitaevskii equation with an optical potential and a cubic nonlinearity

We now consider the computation of the ground state of a Gross-Pitaevskii equation with an optical potential and a cubic nonlinearity in the two-dimensional case. We use the parameters that are chosen in [21]. First, we build the `Method` and `Geometry2D` structures. We use the BESP scheme. The time step is $\Delta t = 10^{-1}$ and the spatial grid involves $2^8 + 1$ points both in the x- and y-directions. The computational domain is: $[-16, 16] \times [-16, 16]$. Moreover, the stopping criterion is 10^{-6}. We refer to Table 4.13 for the details.



(a) Component 1 of the ground state

(b) Component 2 of the ground state

Figure 4.3: Ground state obtained at the end of the simulation.

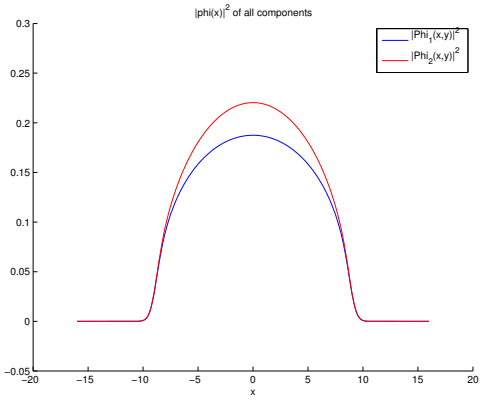


Figure 4.4: Both components of the ground state.

```
Computation = 'Ground';
Ncomponents = 1;
Type = 'BESP';
Deltat = 1e-2;
Stop_time = [];
Stop_crit = 1e-6;
Method = Method_Var2d(Computation,Ncomponents, Type, Deltat, Stop_time ,
Stop_crit);
xmin = -15;
xmax = 15;
ymin = -15;
ymax = 15;
Nx = 2^8+1;
Ny = 2^8+1;
Geometry2D = Geometry2D_Var2d(xmin,xmax, ymin,ymax, Nx, Ny);
```

Table 4.13: Creating the `Method` and `Geometry1D` structures.

We now define the physics of the problem. Let us consider the computation of the ground state of the following Gross-Pitaevskii equation

$$\begin{aligned} i\partial_t\psi(t,x,y) &= \frac{1}{2}\Delta\psi(t,x,y) + \left[\frac{|x|^2+|y|^2}{2} + \kappa\left(\sin^2\left(\frac{\pi x}{4}\right) + \sin^2\left(\frac{\pi y}{4}\right)\right)\right]\psi(t,x,y) \\ &\quad +\beta|\psi(t,x,y)|^2\psi(t,x,y), \end{aligned}$$

with $\kappa = 100$ and $\beta = 1000$. Let us recall that, since the default nonlinearity is the cubic nonlinearity, we do not have to redefine it. Therefore, we only have to impose the optical potential and then to build the `Physics2D` structure. We end by adding the potential operator to `Physics2D` which is defined as the optical potential and the default nonlinear operator (see Table 4.14).

```
Delta = 0.5;
Beta = 1000;
Kappa = 100;
Optical_Potential = @(x,y) (1/2)*(x.^2 + y.^2) + ...
Kappa*(sin(pi*x/4).^2+sin(pi*y/4).^2);
Physics2D = Physics2D_Var2d(Method,Delta,Beta);
Physics2D = Potential_Var2d(Method, Physics2D,Optical_Potential)
Physics2D = Nonlinearity_Var2d(Method, Physics2D);
```

Table 4.14: Construction of the `Physics2D` structure.

We use a centered gaussian as initial function (see Table 4.5).

```
InitialData_choice = 2 ;
Phi_0 = InitialData_Var2d(Method, Geometry2D, Physics2D,InitialData_choice);
```

Figure 4.5: Setting the initial data.

The outputs and the printing informations are then defined (see Table 4.15).

```
Outputs = OutputsINI_Var2d(Method);
Printing = 1;
Evo = 15;
Draw = 1;
Print = Print_Var2d(Printing,Evo,Draw);
[Phi,Outputs]= GPELab2d(Phi_0,Method,Geometry2D,Physics2D,Outputs,[],Print);
```

Table 4.15: Setting the outputs and the printing of informations, and then launching the simulation.

The results of the simulation are reported in Table 4.16 and the solution is given in Figure 4.6. The evolution of the energy and the chemical potential are plotted in Figure 4.7.

4.4 Ground state of a 2d Gross-Pitaevskii equation with a quadratic-plus-quartic potential, a cubic nonlinearity and a rotational operator

We consider the computation of the ground state of a 2d Gross-Pitaevskii equation with a quadratic plus quartic potential, a cubic nonlinearity and a rotational operator. This is a typical example of

```
--------------------
Iteration 157 on 1000000
--Outputs of component 1--------
Square at the origin:  0.06528267039451
x-radius mean square:  3.96548348047485
y-radius mean square:  3.96548346927846
Energy:  51.22028604002639
Chemical potential:  66.24901258432952
Angular momentum:  -0.00000000000000
Energy evolution:  -0.00000000003103
--------------------
CPU time:  172.47
»
```

Table 4.16: Printed outputs for the 2d GPE with cubic nonlinearity and optical lattice.

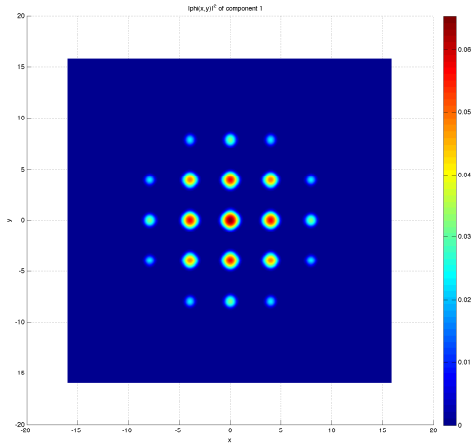


Figure 4.6: Modulus of the ground state.

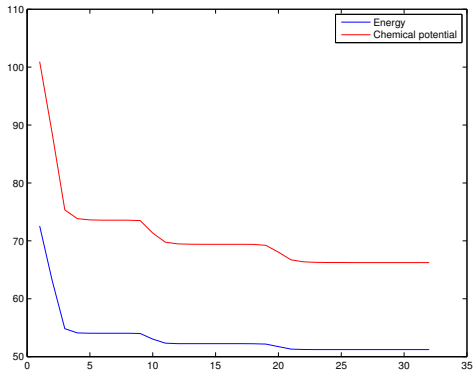


Figure 4.7: Evolution of the energy and the chemical potential during the computation.

computation of a ground state where a fast rotation can be considered. We begin by constructing the `Method` and `Geometry2D` structures. We use the BESP scheme, with a time step equal to 10^{-3}, to solve the problem on a spatial grid involving $2^8 + 1$ points in the x- and y- directions for the

domain $[-10, 10] \times [-10, 10]$. The stopping criterion is fixed to 10^{-5} (see Table 4.17).

```
Computation = 'Ground';
Ncomponents = 1;
Type = 'BESP';
Deltat = 1e-3;
Stop_time = [];
Stop_crit = 1e-5;
Method = Method_Var2d(Computation,Ncomponents, Type, Deltat, Stop_time ,
Stop_crit);
xmin = -10;
xmax = 10;
ymin = -10;
ymax = 10;
Nx = 2^8+1;
Ny = 2^8+1;
Geometry2D = Geometry2D_Var2d(xmin,xmax, ymin,ymax, Nx, Ny);
```

Table 4.17: Setting the `Method` and `Geometry2D` structures.

In this section, the Gross-Pitaevskii equation that we consider is

$$\begin{aligned} i\partial_t\psi(t,x,y) &= \frac{1}{2}\Delta\psi(t,x,y) + \left[\frac{1-\alpha}{2}\left(\gamma_x|x|^2+\gamma_y|y|^2\right) + \frac{\kappa}{4}\left(\gamma_x|x|^2+\gamma_y|y|^2\right)^2\right]\psi(t,x,y) \\ &\quad +\beta|\psi(t,x,y)|^2\psi(t,x,y) + i\Omega\left(y\partial_x - x\partial_y\right)\psi(t,x,y), \end{aligned}$$

with the parameters $\alpha = 1.2$, $\kappa = 0.3$, $\gamma_x = \gamma_y = 1$, $\beta = 1000$ and $\Omega = 3.5$. We know that the default nonlinearity is the cubic nonlinearity and the default gradients define the rotational operator. Therefore, we only have to define the quadratic-plus-quartic potential by using a Matlab script (Table 4.18) and then construct the `Physics2D` structure with the desired coefficients.

```
function [Potential] = quadratic_plus_quartic_potential2d(gamma_x,
gamma_y,alpha,kappa,X,Y)
Potential = (1-alpha)*(gamma_x*X.^2 + gamma_y*Y.^2)/2 + kappa*((gamma_x*X.^2 +
gamma_y*Y.^2)/2).^2;
```

Table 4.18: Defining the quadratic-plus-quartic potential.

We can now construct the `Physics2D` structure by using the `Physics2D_Var2d` function and then add the quadratic-plus-quartic potential that we have defined, the default nonlinear operators and the default gradients to the `Physics2D` structure, thus including them in the physics of the problem (see Table 4.19).

```
Delta = 0.5;
Beta = 1000;
Omega = 3.5;
Physics2D = Physics2D_Var2d(Method,Delta,Beta,Omega);
Alpha = 1.2;
Kappa = 0.3;
Gamma_x = 1;
Gamma_y = 1;
Physics2D = Potential_Var2d(Method, Physics2D,...
@(X,Y) quadratic_plus_quartic_potential2d(Gamma_x, Gamma_y,Alpha,Kappa,X,Y));
Physics2D = Nonlinearity_Var2d(Method, Physics2D);
Physics2D = Gradientx_Var2d(Method, Physics2D);
Physics2D = Gradienty_Var2d(Method, Physics2D);
```

Table 4.19: Building and defining the `Physics2D` structure.

We choose the Thomas-Fermi approximation as initial data (see Table 4.20).

```
InitialData_choice = 2 ;
Phi_0 = InitialData_Var2d(Method, Geometry2D, Physics2D,InitialData_choice);
```

Table 4.20: Building the initial data.

The outputs and printing options are defined in Table 4.21.

```
Outputs = OutputsINI_Var2d(Method);
Printing = 1;
Evo = 15;
Draw = 1;
Print = Print_Var2d(Printing,Evo,Draw);
[Phi,Outputs]= GPELab2d(Phi_0,Method,Geometry2D,Physics2D,Outputs,[],Print);
```

Table 4.21: Defining the `Outputs` and `Print` structures and then launching the computation using `GPELab2d`.

We obtain in Table 4.22 the informations on the ground state at the end of the computation.

```
--------------------
Iteration 46766 on 1000000
--Outputs of component 1--------
Square at the origin:  0.00000000000000
x-radius mean square:  4.57951169686043
y-radius mean square:  4.57951071463754
Energy:  115.52164061561449
Chemical potential:  122.58168418655728
Angular momentum:  146.32747911959200
Energy evolution:  -0.00000000141087
»
```

Table 4.22: Printed outputs for the 2d GPE with cubic nonlinearity, rotation and quadratic-plus-quartic potential.

We report on figure 4.8 the modulus of the stationary state solution at the end of the computations. We can observe the creation of an annulus with uniformly spaced vortices inside.

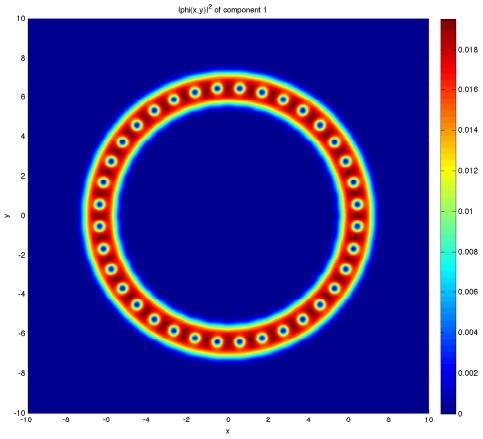


Figure 4.8: Modulus of the ground state.

4.5 Ground state of a system of 2d Gross-Pitaevskii equations with quadratic potentials, rotational operators and coupled cubic nonlinearites

The aim of this section is to consider the computation of the ground state of a 2d system composed of two Gross-Pitaevskii equations with quadratic potentials, rotational operators and coupled cubic nonlinearities. The two first structures that need to be defined are the `Method` and `Geometry2D` structures. In our case, we have to set GPELab to simulate two components. Moreover, we use the BESP scheme for a time step equal to 10^{-2} and a spatial grid of $2^8 + 1$ points in each direction of the domain $[-15, 15] \times [-15, 15]$. We set the stopping criterion to 10^{-6} (see Table 4.23).

```
Computation = 'Ground';
Ncomponents = 2;
Type = 'BESP';
Deltat = 1e-2;
Stop_time = [];
Stop_crit = 1e-6;
Method = Method_Var2d(Computation,Ncomponents, Type, Deltat, Stop_time ,
Stop_crit);
xmin = -10;
xmax = 10;
ymin = -10;
ymax = 10;
Nx = 2^8+1;
Ny = 2^8+1;
Geometry2D = Geometry2D_Var2d(xmin,xmax, ymin,ymax, Nx, Ny);
```

Table 4.23: Using the `Method_Var2d` and `Geometry2D_Var2d` functions to build the `Method` and `Geometry2D` structures.

We now define the physics of the problem. Let us consider the problem of computing the ground state of the following system of Gross-Pitaevskii equations

$$\begin{cases} i\partial_t\psi_1(t,x,y) = \frac{1}{2}\Delta\psi_1(t,x,y) + \left(\frac{1}{2}\left(|x|^2+|y|^2\right) + \beta_1|\psi_1(t,x,y)|^2 + \beta_{12}|\psi_2(t,x,y)|^2\right)\psi_1(t,x,y) \\ \qquad\qquad +i\Omega\left(y\partial_x - x\partial_y\right)\psi_1(t,x,y), \\ i\partial_t\psi_2(t,x,y) = \frac{1}{2}\Delta\psi_2(t,x,y) + \left(\frac{1}{2}\left(|x|^2+|y|^2\right) + \beta_2|\psi_2(t,x,y)|^2 + \beta_{12}|\psi_1(t,x,y)|^2\right)\psi_2(t,x,y) \\ \qquad\qquad +i\Omega\left(y\partial_x - x\partial_y\right)\psi_2(t,x,y), \end{cases}$$

with $\beta_1 = \beta_2 = 400$, $\beta_{12} = 200$ and $\Omega = 0.8$. We recall that, since the default potential is the quadratic potential and the default gradients define the rotational operator, we only have to define the coupled cubic nonlinearity. This is done in Table 4.24.

```
function [CoupledCubicNonlinearity] = Coupled_Cubic2d(Beta_coupled)
CoupledCubicNonlinearity = cell(2);
CoupledCubicNonlinearity{1,1} = @(Phi,X,Y) Beta_coupled(1,1)*abs(Phi{1}).^2+...
Beta_coupled(1,2)*abs(Phi{2}).^2;
CoupledCubicNonlinearity{2,2} = @(Phi,X,Y) Beta_coupled(2,2)*abs(Phi{2}).^2+...
Beta_coupled(2,1)*abs(Phi{1}).^2;
CoupledCubicNonlinearity{1,2} = @(Phi,X,Y) 0;
CoupledCubicNonlinearity{2,1} = @(Phi,X,Y) 0;
```

Table 4.24: Defining the coupled nonlinearity.

We also have to impose the energy associated to the coupled cubic nonlinearity (Table 4.25).

```
function [CoupledCubicEnergy] = Coupled_Cubic_energy2d(Method,Beta_coupled)
CoupledCubicEnergy = cell(2);
CoupledCubicEnergy{1,1} = @(Phi,X,Y) (1/2)*Beta_coupled(1,1)*abs(Phi{1}).^2+...
(1/2)*Beta_coupled(1,2)*abs(Phi{2}).^2;
CoupledCubicEnergy{2,2} = @(Phi,X,Y) (1/2)*Beta_coupled(2,2)*abs(Phi{2}).^2+...
(1/2)*Beta_coupled(2,1)*abs(Phi{1}).^2;
CoupledCubicEnergy{1,2} = @(Phi,X,Y) 0;
CoupledCubicEnergy{2,1} = @(Phi,X,Y) 0;
```

Table 4.25: Defining the energy associated with the coupled nonlinearity.

We can now build the `Physics2D` structure accordingly to our problem. We set the coefficients for the Laplacian, the nonlinearity and the rotational operator by using the `Physics2D_Var2d` function. Then we add the default potential, the default gradients and the coupled cubic nonlinearity to the physics of the problem thanks to the functions associated to each operator (see Table 4.26).

```
Delta = 0.5;
Beta = 200;
Beta_coupled= [2,1;1,1];
Omega = 0.8;
Physics2D = Physics2D_Var2d(Method,Delta,[],Omega);
Physics2D = Potential_Var2d(Method, Physics2D);
Physics2D = Nonlinearity_Var2d(Method, Physics2D,...
Coupled_Cubic2d(Method,Beta_coupled),...
[],Coupled_Cubic_energy2d(Method,Beta_coupled));
Physics2D = Gradientx_Var2d(Method, Physics2D);
Physics2D = Gradienty_Var2d(Method, Physics2D);
```

Table 4.26: Building the `Physics2D` structure.

We then set the initial function as the Thomas-Fermi approximation (see Table 4.27).

```
InitialData_choice = 2 ;
Phi_0 = InitialData_Var2d(Method, Geometry2D, Physics2D,InitialData_choice);
```

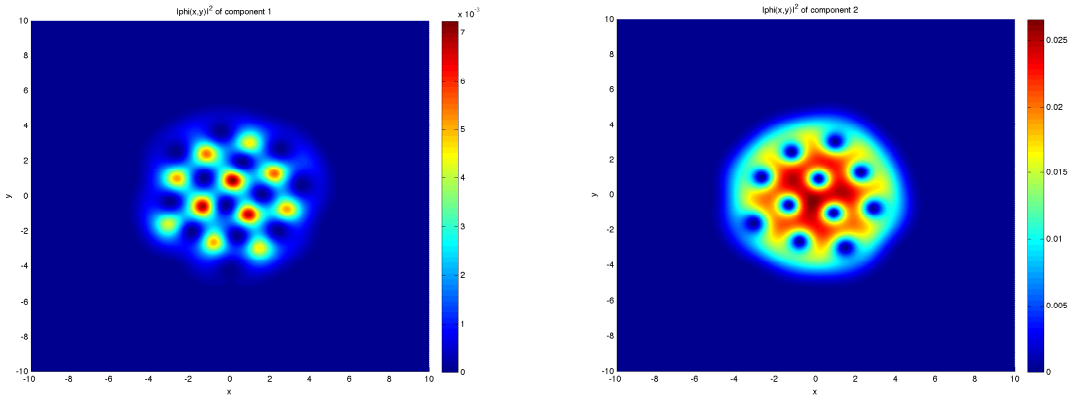
Table 4.27: Constructing the initial data.

Table 4.28 provides the informations for the outputs and printing options.

```
Outputs = OutputsINI_Var2d(Method);
Printing = 1;
Evo = 15;
Draw = 1;
Print = Print_Var2d(Printing,Evo,Draw);
[Phi,Outputs]= GPELab2d(Phi_0,Method,Geometry2D,Physics2D,Outputs,[],Print);
```

Table 4.28: Printing options and launching the computation.

We report on figure 4.9 the moduli of the stationary state solutions at the end of the computations.



(a) Component 1 of the ground state (b) Component 2 of the ground state

Figure 4.9: Ground state obtained at the end of the simulation.

4.6 Ground state of a 3d Gross-Pitaevskii equation with a quadratic potential, a cubic nonlinearity and a rotational operator

In this section, we consider the problem of computing the ground state of a Gross-Pitaevskii equation with quadratic potential, cubic nonlinearity and a rotational operator in 3d. We first build the `Method` and `Geometry3D` structures. We use a BESP scheme and a step time of 10^{-2}, with a spatial grid of 2^6+1 points in the x-, y- and z-directions on the domain $[-10,10]\times[-10,10]\times[-10,10]$. Moreover, we set the stopping criterion to 10^{-6}. We can see in Table 4.29 how to proceed to set these parameters in the `Method` and `Geometry3D` structures by using the `Method_Var3d` and `Geometry3D_Var3d` functions.

```
Computation = 'Ground';
Ncomponents = 1;
Type = 'BESP';
Deltat = 1e-2;
Stop_time = [];
Stop_crit = 1e-4;
Method = Method_Var3d(Computation,Ncomponents, Type, Deltat, Stop_time ,
Stop_crit);
xmin = -10;
xmax = 10;
ymin = -10;
ymax = 10;
zmin = -10;
zmax = 10;
Nx = 2^6+1;
Ny = 2^6+1;
Nz = 2^6+1;
Geometry3D = Geometry3D_Var3d(xmin,xmax, ymin,ymax, zmin,zmax, Nx, Ny, Nz);
```

Table 4.29: Building the `Method` and `Geometry3D` structures.

The ground state that we want to compute is associated to the following Gross-Pitaevskii equation

$$\begin{aligned} i\partial_t\Psi(t,x,y,z) &= \frac{1}{2}\Delta\Psi(t,x,y,z)+\frac{1}{2}\left(\gamma_x|x|^2+\gamma_y|y|^2+\gamma_z|z|^2\right)\Psi(t,x,y,z) \\ &\quad +\beta|\Psi(t,x,y,z)|^2\Psi(t,x,y,z)+\mathbf{\Omega}\cdot(\mathbf{x}\times\nabla)\,\Psi(t,x,y,z), \end{aligned}$$

with $\gamma_x=\gamma_y=\gamma_z=1$, $\beta=500$ and $\mathbf{\Omega}=(0,0,0.9)$. We keep in mind that the default nonlinearity is the cubic nonlinearity, the default potential is the quadratic potential and the default gradients operators are the rotational operators. This implies that we only have to build the `Physics3D` structure with the desired coefficients and then add each operator (see Table 4.30 for more details).

```
Delta = 0.5;
Beta = 500;
Omega = [0,0,0.9];
Physics3D = Physics3D_Var3d(Method,Delta,Beta,Omega);
Physics3D = Potential_Var3d(Method, Physics3D);
Physics3D = Gradientx_Var3d(Method, Physics3D);
Physics3D = Gradienty_Var3d(Method, Physics3D);
Physics3D = Gradientz_Var3d(Method, Physics3D);
Physics3D = Nonlinearity_Var3d(Method, Physics2D);
```

Table 4.30: Setting the coefficients and adding the default operators to the `Physics3D` structure.

We then set the initial function by using the `InitialData_Var3d`. We choose the Thomas-Fermi approximation (see Table 4.31).

```
InitialData_choice = 2 ;
Phi_0 = InitialData_Var3d(Method, Geometry3D, Physics3D,InitialData_choice);
```

Table 4.31: Building the initial data as the Thomas-Fermi approximation.

We finally set the outputs and the printing informations, then we launch the simulation following Table 4.32.

```
Outputs = OutputsINI_Var3d(Method);
Printing = 1;
Evo = 15;
Draw = 1;
Print = Print_Var3d(Printing,Evo,Draw);
[Phi,Outputs]= GPELab3d(Phi_0,Method,Geometry3D,Physics3D,Outputs,[],Print);
```

Table 4.32: Creating the `Outputs` and `Print` structure and launching the computation.

At the end of the computation, we obtain on Figure 4.10 the isovalues of the modulus of the stationary state solution.

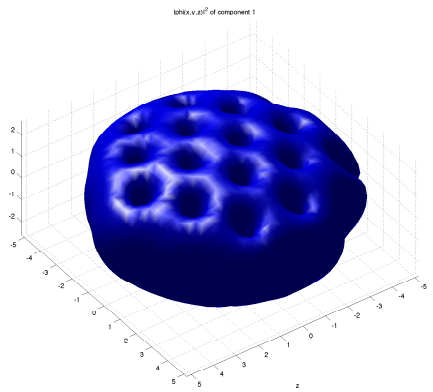


Figure 4.10: 10^{-3}-isovalues of modulus of the ground state.

4.7 Ground state of a 3d Gross-Pitaevskii equation with quadratic potential, cubic nonlinearity and dipole-dipole interaction

We now show how to compute the ground state of a Gross-Pitaevskii equation with quadratic potential, cubic nonlinearity and dipole-dipole interaction in 3d. We first build the `Method` and `Geometry3D` structures. In Table 4.33, we set GPELab to use the BESP scheme for a time step Δt equal to 10^{-2}, with a spatial grid of 2^6+1 points in the x-, y- and z-direction on the domain $[-15,15]\times[-15,15]\times[-15,15]$. The stopping criterion is fixed to 10^{-6}.

```
Computation = 'Ground';
Ncomponents = 1;
Type = 'BESP';
Deltat = 1e-2;
Stop_time = [];
Stop_crit = 1e-6;
Method = Method_Var3d(Computation,Ncomponents, Type, Deltat, Stop_time ,
Stop_crit);
xmin = -10;
xmax = 10;
ymin = -10;
ymax = 10;
zmin = -10;
zmax = 10;
Nx = 2^6+1;
Ny = 2^6+1;
Nz = 2^6+1;
Geometry3D = Geometry3D_Var3d(xmin,xmax, ymin,ymax, zmin,zmax, Nx, Ny, Nz);
```

Table 4.33: Setting the `Method` and `Geometry3D`.

We consider here the following Gross-Pitaevskii equation with a dipole-dipole interaction

$$\begin{aligned} i\partial_t\Psi(t,x,y,z) &= \frac{1}{2}\Delta\Psi(t,x,y,z)+\frac{1}{2}\left(\gamma_x|x|^2+\gamma_y|y|^2+\gamma_z|z|^2\right)\Psi(t,x,y,z) \\ &\quad +\beta|\Psi(t,x,y,z)|^2\Psi(t,x,y,z)+\left(d^2\int_{\mathbb{R}^3}\frac{1-3\cos^2(\widehat{\mathbf{a},\tilde{\mathbf{x}}})}{||(x,y,z)-\tilde{\mathbf{x}}||^3}|\Psi(t,\tilde{\mathbf{x}})|^2d\tilde{\mathbf{x}}\right)\Psi(t,x,y,z), \end{aligned}$$

with $\gamma_x=\gamma_y=\gamma_z=1$, $\beta=2000$ and $\mathbf{a}=(0,0,1)$. In GPELab, the default nonlinearity is the cubic nonlinearity and the default potential is the quadratic potential. We only have to define the dipolar interaction which can be computed by using a FFT *via*

$$d^2\int_{\mathbb{R}^3}\frac{1-3\cos^2(\widehat{\mathbf{a},\tilde{\mathbf{x}}})}{||(x,y,z)-\tilde{\mathbf{x}}||^3}|\Psi(t,\tilde{\mathbf{x}})|^2d\tilde{\mathbf{x}}=\mathcal{F}^{-1}\left(\frac{4\pi}{3}d^2(3\cos^2(\widehat{\mathbf{a},\boldsymbol{\xi}})-1)\mathcal{F}\left(|\Psi(t,x,y,z)|^2\right)(\boldsymbol{\xi})\right)(x,y,z).$$

Using the previous formula, we are able to efficiently compute the dipole-dipole interaction and we use the `FFTNonlinearity_Var3d` function to add it to the `Physics3d` structure. In Table 4.34, we can see how to define this type of nonlinearity in Matlab.

We can now proceed to build the `Physics3D` structure with the desired coefficients, add the default nonlinear operator, the quadratic-plus-quartic potential and the defined dipolar interaction to the `Physics3D` structure. We can see in Table 4.35 how we add each operator to the `Physics3D` structure by using the `Potential_Var3d`, `Nonlinearity_Var3d` and `FFTNonlinearity_Var3d` functions.

We choose the Thomas-Fermi approximation as the initial function (see Table 4.36).

```
function [Dipolar_interaction_nonlinearity] = Dipolar_interaction3d(Phi, FFTX,
FFTY, FFTZ, Dipolar_direction, d)
Cross_norm = sqrt((FFTY*Dipolar_direction(3)-FFTZ*Dipolar_direction(2)).^2+...
+(FFTZ*Dipolar_direction(1)-FFTX*Dipolar_direction(3)).^2...
+(FFTX*Dipolar_direction(2)-FFTY*Dipolar_direction(1)).^2);
Scalar_prod = FFTX*Dipolar_direction(1)+FFTY*Dipolar_direction(2)...
+FFTZ*Dipolar_direction(3);
Angle = atan2(Cross_norm,Scalar_prod);
NLFFT = fftn(abs(Phi).^2);
V = d^2*(4/3)*pi*(3*cos(Angle).^2-1);
Dipolar_interaction_nonlinearity = ifftn(V.*NLFFT);
```

Table 4.34: Defining the dipolar interaction *via* a FFT.

```
Delta = 0.5;
Beta = 1000;
Dipolar_direction = [0,0,1];
d = 0.5;
Physics3D = Physics3D_Var3d(Method,Delta,Beta);
Physics3D = Potential_Var3d(Method, Physics3D);
Physics3D = Nonlinearity_Var3d(Method, Physics2D);
Physics3D = FFTNonlinearity_Var3d(Method, Physics3D,...
@(Phi,X,Y,Z,FFTX,FFTY,FFTZ)Dipolar_interaction3d(Phi, FFTX, FFTY, FFTZ,
Dipolar_direction , d));
```

Table 4.35: Constructing the physics of the problem.

```
InitialData_choice = 2 ;
Phi_0 = InitialData_Var3d(Method, Geometry3D, Physics3D,InitialData_choice);
```

Table 4.36: Choosing the initial data.

We finally set the outputs and the printing informations, then we launch the simulation (see Table 4.37).

```
Outputs = OutputsINI_Var3d(Method);
Printing = 1;
Evo = 15;
Draw = 1;
Print = Print_Var3d(Printing,Evo,Draw);
[Phi,Outputs]= GPELab3d(Phi_0,Method,Geometry3D,Physics3D,Outputs,[],Print);
```

Table 4.37: Initializing the outputs, setting the informations to print and launching the computation.

At the end of the computation, we obtain (see Figure 4.11) the isovalues of the modulus of the stationary state solution.

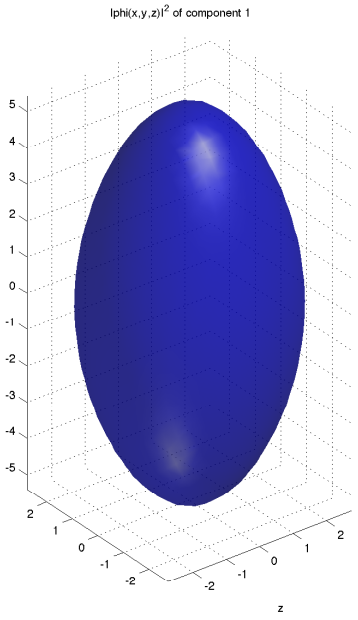


Figure 4.11: 10^{-3}-isovalues of the modulus of the ground state.

Chapter 5

Computation of the dynamics of GPE: methods, functions, examples

After having computed the stationary states of a GPE, one can be interested in the dynamics of the stationary solutions for a modified GPE (for example by changing the potential). More generally, the dynamics of an initial data is very important to analyze the dynamical properties of a BEC.

For this reason, we first introduce the main schemes that are included in GPELab. The first class of methods is Time Splitting SPectral (TSSP) schemes. We briefly explain how these schemes work in Section 5.1 and how an increased accuracy can be obtained by using suitable weighted approximations of the exponential operators appearing in the splitting formulae. Another scheme that is discussed is the relaxation scheme in time, coupled with pseudospectral approximation methods based on FFTs. This approach is described in Section 5.2. In Section 5.4, we explain the Richardson's extrapolation method that is coded in GPELab. This simple technique allows to increase easily the order of the time scheme by linear combinations between computations for fractional uniform time steps. We next provide in Section 5.5 the different functions that are met in GPELab to compute the dynamics of a given (system of) GP equation(s). Finally, in Section 5.6, we end the Chapter by a few computational examples to show how to use the GPELab functions for the dynamics and provide the solutions that are expected.

5.1 Alternate Direction Implicit (ADI)-Time Splitting pseudo SPectral (ADI-TSSP) schemes for the rotating GPE

Let us introduce A and B, two self-adjoint operators such that: $\mathcal{D}(A) \subset L^2$, $\mathcal{D}(B) \subset L^2$ and $A+B$ is a self-adjoint operator on $\mathcal{D}(A) \cap \mathcal{D}(B)$. We designate by $\mathcal{D}(A)$ and $\mathcal{D}(B)$ the domains of the operators A and B, respectively. We consider the following time-dependent Partial Differential Equation (PDE)

$$\begin{cases} \partial_t \psi(t,\mathbf{x}) = A\psi(t,\mathbf{x}) + B\psi(t,\mathbf{x}), \\ \psi(0,\mathbf{x}) = \psi_0(\mathbf{x}), \end{cases}$$

and we denote by $\psi(t,\mathbf{x}) = e^{(A+B)t}\psi_0(\mathbf{x})$ its solution, $t > 0$ and $\mathbf{x} \in \mathbb{R}^d$. The Time-Splitting (TS) schemes consist in approximating the solution of this PDE by the exponential operators of A and B, which means that we approximately split A and B into the exponential representation. In fact, a general approximation of the solution by a time splitting scheme can be written in the following form

$$\psi(t_n + \Delta t, \mathbf{x}) \approx \psi^{n+1}(\mathbf{x}) = e^{a_1 A\Delta t} e^{b_1 B\Delta t} e^{a_2 A\Delta t} e^{b_2 B\Delta t} ... e^{a_p A\Delta t} e^{b_p B\Delta t} \psi^n(\mathbf{x}),$$

where $\{a_k, b_k\}_{1\leq k\leq p} \subset \mathbb{R}$ are some weights that must be computed in such a way that the approximation of $e^{(A+B)\Delta t}$ is of a given order for a time step Δt which is supposed to be relatively small.

Two of the most commonly used time-splitting schemes are the Lie ($a_1 = b_1 = 1$) (cf. subsection 5.1.1) and the Strang ($a_1 = a_2 = 1/2$ and $b_1 = 1, b_2 = 0$) (cf. subsection 5.1.2) schemes which are first- and second-order time schemes, respectively. Higher-order schemes [1] can be constructed for appropriately chosen weights $\{a_k, b_k\}_{1\leq k\leq p}$. The main idea behind the splitting of operators is that each equation associated with each operator A or B can be solved easily.

5.1.1 The Lie ADI-TSSP scheme for the rotating GPE

The Lie TSSP scheme

Let us first begin by the Lie splitting scheme. Let us recall the rotating cubic GPE equation in 2d

$$i\partial_t\psi(t,\mathbf{x}) = -\frac{1}{2}\Delta\psi(t,\mathbf{x}) + V(t,\mathbf{x})\psi(t,\mathbf{x}) + \beta|\psi(t,\mathbf{x})|^2\psi(t,\mathbf{x}) - \Omega L_z\psi, t > 0, \mathbf{x}\in\mathbb{R}^2. \tag{5.1}$$

In terms of splitting, a natural decomposition of the above equation consists in solving

1) the partial differential equation related to

$$A = \frac{i}{2}\Delta + i\Omega L_z,$$

2) and next the ODE with respect to the potential and nonlinear terms

$$B = -iV(t,\mathbf{x}) - i\beta|\psi(t,\mathbf{x})|^2.$$

Indeed, as we will see later, the equation associated to the partial differential operator can be solved with the help of spectral methods. Furthermore, the equation related to the nonlinearity and the potential parts can be integrated exactly. Indeed, for a time step Δt, it can be proved that an approximation of the solution is given by

$$\psi^{n+1}(\mathbf{x}) = e^{i(\frac{1}{2}\Delta + i\Omega L_z)\Delta t}e^{-i(V(t,\mathbf{x})+\beta|\psi(t,\mathbf{x})|^2)\Delta t}\psi^n(\mathbf{x}).$$

Hence, for solving (5.1), we can proceed in two successive steps

1) let $\Delta t > 0$, $n\in\mathbb{N} - \{0\}$ and $n\Delta t < t \leq (n+1)\Delta t$, we solve

$$\begin{cases} i\partial_t\psi_1(t,\mathbf{x}) = -\frac{1}{2}\Delta\psi_1(t,\mathbf{x}) - \Omega L_z\psi_1(t,\mathbf{x}),\ n\Delta t < t \leq (n+1)\Delta t, \\ \psi_1(n\Delta t,\mathbf{x}) = \psi^n(\mathbf{x}), \end{cases} \tag{5.2}$$

2) and next

$$\begin{cases} i\partial_t\psi_2(t,\mathbf{x}) = V(t,\mathbf{x})\psi_2(t,\mathbf{x}) + \beta|\psi_2(t,\mathbf{x})|^2\psi_2(t,\mathbf{x}),\ n\Delta t < t \leq (n+1)\Delta t, \\ \psi_2(n\Delta t,\mathbf{x}) = \psi_1((n+1)\Delta t,\mathbf{x}). \end{cases} \tag{5.3}$$

Then we set: $\psi^{n+1}(\mathbf{x}) := \psi_2((n+1)\Delta t,\mathbf{x})$.

[1] http://techmath.uibk.ac.at/mecht/research/SpringSchool/manuscript_thalhammer.pdf

Time discretization and the ADI technique

Throughout this section, we consider the bounded domain $\mathcal{O} =]-L_x, L_x[\times]-L_y, L_y[$ used in the BESP scheme and the time discretization, $t_0 < t_1 < ... < t_n < ...$, with $\Delta t = \Delta t_n = t_{n+1} - t_n$.

Let us fix $n \in \mathbb{N}$ and $\psi^n \in L^2(\mathcal{O})$ that we assume to be ("numerically") compactly supported inside $\mathcal{O}$. The first step (5.2) of the Lie splitting consists in solving, between times $t = t_n$ and $t = t_{n+1}$,

$$\begin{cases} i\partial_t \psi_1(t,\mathbf{x}) = -\dfrac{1}{2}\Delta\psi_1(t,\mathbf{x}) - \Omega L_z \psi_1(t,\mathbf{x}), \\ \psi_1(t_n,\mathbf{x}) = \psi^n(\mathbf{x}), \end{cases} \tag{5.4}$$

in $\mathcal{O}$. For a non rotating BEC ($\Omega = 0$), this can be done efficiently by means of the FFT since the Laplacian can be inverted in the Fourier space. However, for $\Omega \neq 0$ and since the coefficients of L_z are not constant, this can no longer be done. Here, we apply the solution proposed in Bao *et al.* [20] based on a ADI scheme. The idea is to decouple the global 2d equation as two one-dimensional equations where the coefficients are now constant with respect to the directional derivatives. Therefore, this leads to splitting the equation (5.4) as

1.a) first, for $t = t_n$ to $t = t_{n+1}$,

$$\begin{cases} i\partial_t \psi^{(1)}(t,\mathbf{x}) = -\dfrac{1}{2}\partial_{xx}\psi^{(1)}(t,\mathbf{x}) - i\Omega y \partial_x \psi^{(1)}(t,\mathbf{x}), \\ \psi^{(1)}(t_n,\mathbf{x}) = \psi^n(\mathbf{x}), \end{cases} \tag{5.5}$$

1.b) and then, for $t = t_n$ to $t = t_{n+1}$,

$$\begin{cases} i\partial_t \psi^{(2)}(t,\mathbf{x}) = -\dfrac{1}{2}\partial_{yy}\psi^{(2)}(t,\mathbf{x}) + i\Omega x \partial_y \psi^{(2)}(t,\mathbf{x}), \\ \psi^{(2)}(t_n,\mathbf{x}) = \psi^{(1)}(t_{n+1},\mathbf{x}), \end{cases} \tag{5.6}$$

A second step (which corresponds in fact to consider (5.3)) is to solve the following PDE, from time $t = t_n$ to $t = t_{n+1}$

$$\begin{cases} i\partial_t \psi^{(3)}(t,\mathbf{x}) = V(t,\mathbf{x})\psi^{(3)}(t,\mathbf{x}) + \beta|\psi^{(3)}(t,\mathbf{x})|^2\psi^{(3)}(t,\mathbf{x}), \\ \psi^{(3)}(t_n,\mathbf{x}) = \psi^{(2)}(t_{n+1},\mathbf{x}). \end{cases} \tag{5.7}$$

We know that this equation can be solved exactly and the solution is given by,

$$\psi^{(3)}(t,\mathbf{x}) = \psi^{(2)}(t_{n+1},\mathbf{x}) e^{-i\beta|\psi^{(2)}(t_{n+1},\mathbf{x})|^2(t-t_n) - i\int_{t_n}^{t} V(s,\mathbf{x})ds} \tag{5.8}$$

which provides an approximation of $\psi^{n+1}(\mathbf{x}) \approx \psi^{(3)}(t_{n+1},\mathbf{x})$.

We remark that the ADI technique implies a loss of symmetry of the scheme when solving the partial differential operators of Eq. (5.2). Indeed, we first integrate in the x-direction in (5.5) and next in the y-direction according to (5.6). To avoid this problem and symmetrize the scheme, we alternate the ordering of the derivative directions any two time steps. Most specifically, from t_n to t_{n+1}, we solve (5.5) and next equation (5.6) followed by (5.7). For the next step, i.e. from t_{n+1} to t_{n+2}, we first solve (5.6), and then equation (5.5), and finally again Eq. (5.7).

Space discretization in 2D and implementation

We consider now the notations related to the direct and inverse Fourier series transforms (2.36)-(2.37), page 27. Fourier transforming Eq. (5.5) with respect to the x variable and by using the orthogonality of the Fourier functions, we obtain

$$i\partial_t \widehat{\psi}_p^{(1)}(t,y) = \left(\frac{1}{2}\mu_p^2 + \Omega y \mu_p\right)\widehat{\psi}_p^{(1)}(t,y),\ 1 - J/2 \leq p \leq J/2,\ t_n \leq t \leq t_{n+1},$$

This ODE can be integrated in time exactly and we obtain, $\forall t \in [t_n, t_{n+1}]$ and $\forall p$, $1-J/2 \leq p \leq J/2$,

$$\widehat{\psi}_p^{(1)}(t,y) = e^{-i\left(\frac{1}{2}\mu_p^2+\Omega y\mu_p\right)(t-t_n)}\widehat{\psi}_p^{(1)}(t_n,y).$$

Similarly, for Eq. (5.6), one gets, $\forall t \in [t_n, t_{n+1}]$ and $\forall q$, $1-K/2 \leq q \leq K/2$,

$$\widehat{\psi}_q^{(2)}(t,x) = e^{-i\left(\frac{1}{2}\lambda_q^2-\Omega x\lambda_q\right)(t-t_n)}\widehat{\psi}_q^{(2)}(t_n,x).$$

Thus, the first step of the Lie scheme, where we solve equation (5.5) and then equation (5.6) on the time interval $[t_n, t_{n+1}]$ and the domain $\mathcal{O}_{J,K}$, will be implemented in the following way

$$\psi^{(1)}(t_{n+1},x_j,y_k) = \frac{1}{J}\sum_{p=-J/2}^{J/2-1} e^{-i\left(\frac{1}{2}\mu_p^2+\Omega y_k\mu_p\right)(t_{n+1}-t_n)}\widehat{\psi_p^n}(y_k)e^{i\mu_p(x_j+L_x)},$$

$$\psi^{(2)}(t_{n+1},x_j,y_k) = \frac{1}{K}\sum_{q=-K/2}^{K/2-1} e^{-i\left(\frac{1}{2}\lambda_q^2-\Omega x_j\lambda_q\right)(t_{n+1}-t_n)}\widehat{\psi}_q^{(1)}(t_{n+1},x_j)e^{i\lambda_q(y_k+L_y)}.$$

Of course these operations are based on the `fft()` and `ifft()` Matlab functions. The frequency dependent quantities $(\mu_p^2/2+\Omega y_k\mu_p)$ and $(\lambda_q^2/2-\Omega x_j\lambda_q)$ are obtained through the `Delta_grad_Fourier()` (to compute μ_p and λ_q) and the `meshgrid()` functions. Then, the exponential matrix is computed by the usual exponential Matlab function. To discretize (5.8), we use the standard Simpson's quadrature rule in time

$$\begin{aligned}\int_{t_n}^{t_{n+1}} V(s,x_j,y_k)ds &\approx \frac{1}{6}\left(V(t_n,x_j,y_k)+6V\left(t_{n+1/2},x_j,y_k\right)+V(t_{n+1},x_j,y_k)\right)(t_{n+1}-t_n)\\ &:= \tilde{V}_n(x_j,y_k)(t_{n+1}-t_n),\end{aligned}$$

with $t_{n+1/2} = (t_n+t_{n+1})/2$, leading to

$$\psi^{(3)}(t_{n+1},x_j,y_k) = \psi^{(2)}(t_{n+1},x_j,y_k)e^{-i\Delta t\left(\beta|\psi^{(2)}(t_{n+1},x_j,y_k)|^2+\tilde{V}_n(x_j,y_k)\right)}.$$

This corresponds to only change the phase of the solution by a suitable shift. Let us also remark that all we have done above extend directly to the case of a general nonlinearity $f(|\psi|,\mathbf{x})$. This scheme is globally first-order in time and spectral in space.

5.1.2 The Strang ADI-TSSP scheme for the rotating GPE

Let us briefly explain the Strang splitting scheme, the implementation being quite similar to the Lie scheme for a rotating BEC. The approximation of the solution, on a time step Δt, will either be $e^{A(\Delta t/2)}e^{B\Delta t}e^{A(\Delta t/2)}$ or $e^{B(\Delta t/2)}e^{A\Delta t}e^{B(\Delta t/2)}$, setting $A = \frac{i}{2}\Delta + i\Omega L_z$ and $B = -iV(t,\mathbf{x}) - i\beta|\psi(t,\mathbf{x})|^2$. Using the ADI method and the first exponential splitting, the TSSP Strang scheme is the following

1) solve from $t = t_n$ to $t = t_{n+1/2}$

$$\begin{cases} i\partial_t\psi^{(1)}(t,\mathbf{x}) = -\frac{1}{2}\partial_{xx}\psi^{(1)}(t,\mathbf{x}) - i\Omega y\partial_x\psi^{(1)}(t,\mathbf{x}),\\ \psi^{(1)}(t_n,\mathbf{x}) = \psi_n(\mathbf{x}).\end{cases} \tag{5.9}$$

2) Solve from $t = t_n$ to $t = t_{n+1/2}$

$$\begin{cases} i\partial_t\psi^{(2)}(t,\mathbf{x}) = -\frac{1}{2}\partial_{yy}\psi^{(2)}(t,\mathbf{x}) + i\Omega x\partial_y\psi^{(2)}(t,\mathbf{x}),\\ \psi^{(2)}(t_n,\mathbf{x}) = \psi^{(1)}(t_{n+1/2},\mathbf{x}).\end{cases} \tag{5.10}$$

3) Solve from $t = t_n$ to $t = t_{n+1}$

$$\begin{cases} i\partial_t \psi^{(3)}(t,\mathbf{x}) = V(t,\mathbf{x})\psi^{(3)}(t,\mathbf{x}) + \beta|\psi^{(3)}(t,\mathbf{x})|^2\psi^{(3)}(t,\mathbf{x}), \\ \psi^{(3)}(t_n,\mathbf{x}) = \psi^{(2)}(t_{n+1/2},\mathbf{x}), \end{cases} \tag{5.11}$$

by the change of phase.

4) Solve from $t = t_n$ to $t = t_{n+1/2}$

$$\begin{cases} i\partial_t \psi^{(4)}(t,\mathbf{x}) = -\frac{1}{2}\partial_{yy}\psi^{(4)}(t,\mathbf{x}) + i\Omega x\partial_y \psi^{(4)}(t,\mathbf{x}), \\ \psi^{(4)}(t_n,\mathbf{x}) = \psi^{(3)}(t_{n+1},\mathbf{x}). \end{cases} \tag{5.12}$$

5) Solve from $t = t_n$ to $t = t_{n+1/2}$

$$\begin{cases} i\partial_t \psi^{(5)}(t,\mathbf{x}) = -\frac{1}{2}\partial_{xx}\psi^{(5)}(t,\mathbf{x}) - i\Omega y\partial_x \psi^{(5)}(t,\mathbf{x}), \\ \psi^{(5)}(t_n,\mathbf{x}) = \psi^{(4)}(t_{n+1/2},\mathbf{x}). \end{cases} \tag{5.13}$$

This last step finally gives $\psi^{n+1}(\mathbf{x}) := \psi^{(5)}(t_{n+1/2},\mathbf{x})$. Each partial differential equation with respect to x or y is solve through FFTs and iFFTs.

The Lie ADI-TSSP scheme is first-order in time and spectral in space, while the Strang ADI-TSSP scheme is second-order in time and spectrally accurate in space. Their computational costs are $\mathcal{O}(M \log M)$. The extension to the three-dimensional case is direct. Other properties are related to the fact that these schemes are time reversible, mass conserving (if it is true at the continuous level), time transverse invariant and the dispersion relation holds. However, there is no energy conservation. Indeed, it can be proved that only the energy of a closely related system is conserved. These explicit schemes are unconditionally stable. More details can be found in [**?**].

5.1.3 Extension to the multi-components case

We consider the notations of Section 2.6, page 30. The TSSP schemes can also be extended to the multi-components case, i.e. a system of N_c coupled GPEs

$$\begin{aligned} i\partial_t \Psi(t,\mathbf{x}) = -\frac{1}{2}\Delta\Psi(t,\mathbf{x}) + V(\mathbf{x})\Psi(t,\mathbf{x}) + \sum_{j=1}^{d} G^j(\mathbf{x}\backslash x_j)\partial_{x_j}\Psi(t,\mathbf{x}) \\ +\beta F(\Psi(t,\mathbf{x}),\mathbf{x})\Psi(t,\mathbf{x}),\ t>0, \mathbf{x}\in\mathbb{R}^d, \end{aligned} \tag{5.14}$$

where we set $|\Psi(t,\mathbf{x})|^2 = \sum_{m=1}^{N_c} |\psi_m(t,\mathbf{x})|^2$ and $\mathbf{x}\backslash x_j = (x_1,...,x_{j-1},x_{j+1},...,x_d)$. The operators are defined in Section 2.6. We furthermore assume that V and F are symmetric operators to get the mass conservation property: $V_{\ell,m} = V_{m,\ell}$ and $F_{\ell,m} = F_{m,\ell}$, $1 \le \ell, m \le N_c$. Let us remark that, for the TSSP scheme, we assume that F only depends on the modulus of Ψ, e.g. $F(\Psi(t,\mathbf{x}),\mathbf{x}) := F(|\Psi(t,\mathbf{x})|,\mathbf{x})$. The main difference is related to the form of the variable coefficients matrices in front of the gradients

$$G^j(\mathbf{x}\backslash x_j) = \begin{pmatrix} G^j_{11}(\mathbf{x}\backslash x_j) & G^j_{12}(\mathbf{x}\backslash x_j) & \cdots & G^j_{1N_c}(\mathbf{x}\backslash x_j) \\ G^j_{21}(\mathbf{x}\backslash x_j) & G^j_{22}(\mathbf{x}\backslash x_j) & \cdots & G^j_{2N_c}(\mathbf{x}\backslash x_j) \\ \vdots & \vdots & \ddots & \vdots \\ G^j_{N_c1}(\mathbf{x}\backslash x_j) & G^j_{N_c2}(\mathbf{x}\backslash x_j) & \cdots & G^j_{N_cN_c}(\mathbf{x}\backslash x_j) \end{pmatrix}.$$

The initial data is $\Psi(t=0,\mathbf{x}) = \Psi_0(\mathbf{x})$. To derive the TSSP scheme for this system, we choose the same kind of splitting strategy. For the Lie scheme, this leads to solving the equation in two steps. Let $\Delta t > 0$ and Ψ^n the approximation of the solution at time t_n. Then the splitting yields

1) solve from t_n to t_{n+1}

$$\left\{ \begin{array}{l} i\partial_t \Psi^{(1)}(t,\mathbf{x}) = -\frac{1}{2}\Delta\Psi^{(1)}(t,\mathbf{x}) + \sum_{j=1}^{d} G^j(\mathbf{x}\backslash x_j)\partial_{x_j}\Psi^{(1)}(t,\mathbf{x}), \\ \Psi^{(1)}(t_n,\mathbf{x}) = \Psi^n(\mathbf{x}). \end{array} \right. \tag{5.15}$$

2) Solve from t_n to t_{n+1}

$$\left\{ \begin{array}{l} i\partial_t \Psi^{(2)}(t,\mathbf{x}) = V(\mathbf{x})\Psi^{(2)}(t,\mathbf{x}) + \beta F(|\Psi^{(2)}(t,\mathbf{x})|,\mathbf{x})\Psi^{(2)}(t,\mathbf{x}), \\ \Psi^{(2)}(t_n,\mathbf{x}) = \Psi^{(1)}(t_{n+1},\mathbf{x}). \end{array} \right. \tag{5.16}$$

We furthermore have: $\Psi^0(\mathbf{x}) := \Psi_0(\mathbf{x})$. We remark now that (5.16) leads to: $\forall t > t_n$, $|\Psi^{(2)}(t,\mathbf{x})| = |\Psi^{(2)}(t_n,\mathbf{x})|$. Indeed, we have

$$\begin{aligned} \sum_{m=1}^{N_c} \partial_t |\Psi_m^{(2)}(t,\mathbf{x})|^2 &= 2\sum_{m=1}^{N_c} \Re\left(\Psi_m^{(2)}(t,\mathbf{x})^* \partial_t \Psi_m^{(2)}(t,\mathbf{x})\right) \\ &= -2\sum_{m,o=1}^{N_c} \Im\left(\Psi_m^{(2)}(t,\mathbf{x})^* (V_{mo}(\mathbf{x}) + F_{mo}(|\Psi^{(2)}(t,\mathbf{x})|,\mathbf{x}))\Psi_o^{(2)}(t,\mathbf{x})\right). \end{aligned}$$

Using the fact that $V_{mo}(\mathbf{x}) = V_{om}(\mathbf{x})$ and $F_{mo}(|\Psi^{(2)}(t,\mathbf{x})|,\mathbf{x}) = F_{om}(|\Psi^{(2)}(t,\mathbf{x})|,\mathbf{x})$, we obtain that

$$\begin{aligned} &\sum_{m=1}^{N_c} \partial_t |\Psi_m^{(2)}(t,\mathbf{x})|^2 = \\ &\quad -2\sum_{N_c \geq o > m \geq 1} \Im\left((V_{mo}(\mathbf{x}) + F_{mo}(|\Psi^{(2)}(t,\mathbf{x})|,\mathbf{x}))(\Psi_m^{(2)}(t,\mathbf{x})^*\Psi_o^{(2)}(t,\mathbf{x}) + \Psi_o^{(2)}(t,\mathbf{x})^*\Psi_m^{(2)}(t,\mathbf{x}))\right) \\ &\qquad -2\sum_{N_c \geq m \geq 1} \Im\left((V_{mm}(\mathbf{x}) + F_{mm}(|\Psi^{(2)}(t,\mathbf{x})|,\mathbf{x}))|\Psi_m^{(2)}(t,\mathbf{x})|^2\right) \\ &\qquad = -4\sum_{N_c \geq o > m \geq 1} \Im\left((V_{mo}(\mathbf{x}) + F_{mo}(|\Psi^{(2)}(t,\mathbf{x})|,\mathbf{x}))\Re(\Psi_m^{(2)}(t,\mathbf{x})^*\Psi_o^{(2)}(t,\mathbf{x}))\right) = 0. \end{aligned} \tag{5.17}$$

As a conclusion, we have that: $|\Psi^{(2)}(t,\mathbf{x})| = |\Psi^{(2)}(t_n,\mathbf{x})|$, for $t > t_n$.

Let $n \in \mathbb{N}$ and $\Psi^n(\mathbf{x}) \in L^2(\mathcal{O})^{N_c}$, compactly supported in $\mathcal{O}$. For the sake of simplicity, we explicit the method in the two-dimensional case, e.g. $d = 2$. A first step is to solve, from $t = t_n$ to $t = t_{n+1}$,

$$\left\{ \begin{array}{l} i\partial_t \Psi^{(1)}(t,\mathbf{x}) = -\frac{1}{2}\Delta\Psi^{(1)}(t,\mathbf{x}) + \sum_{j=1}^{2} G^j(\mathbf{x}\backslash x_j)\partial_{x_j}\Psi^{(1)}(t,\mathbf{x}), \\ \Psi^{(1)}(t_n,\mathbf{x}) = \Psi^n(\mathbf{x}). \end{array} \right. \tag{5.18}$$

As in the one-component case, an ADI method must be used to decouple the effects of G^j. Therefore, we split Eq. (5.15) in two equations, and solve from $t = t_n$ to $t = t_{n+1}$,

$$\left\{ \begin{array}{l} i\partial_t \Psi^{(1,1)}(t,\mathbf{x}) = -\frac{1}{2}\partial_{xx}\Psi^{(1,1)}(t,\mathbf{x}) + G^1(y)\partial_x\Psi^{(1,1)}(t,\mathbf{x}), \\ \Psi^{(1,1)}(t_n,\mathbf{x}) = \Psi^n(\mathbf{x}), \end{array} \right. \tag{5.19}$$

and next, from $t = t_n$ to $t = t_{n+1}$

$$\left\{ \begin{array}{l} i\partial_t \Psi^{(1,2)}(t,\mathbf{x}) = -\frac{1}{2}\partial_{yy}\Psi^{(1,2)}(t,\mathbf{x}) + G^2(x)\Psi^{(1,2)}(t,\mathbf{x}), \\ \Psi^{(1,2)}(t_n,\mathbf{x}) = \Psi^{(1,1)}(t_{n+1},\mathbf{x}). \end{array} \right. \tag{5.20}$$

A second step is to solve Eq. (5.16) for a well-chosen initial data, from $t = t_n$ to $t = t_{n+1}$, e.g.

$$\left\{ \begin{array}{l} i\partial_t \Psi^{(2)}(t,\mathbf{x}) = V(\mathbf{x})\Psi^{(2)}(t,\mathbf{x}) + \beta F(|\Psi^{(2)}(t,\mathbf{x})|,\mathbf{x})\Psi^{(2)}(t,\mathbf{x}), \\ \Psi^{(2)}(t_n,\mathbf{x}) = \Psi^{(1,2)}(t_{n+1},\mathbf{x}). \end{array} \right. \tag{5.21}$$

This equation, that can be solved exactly, has for solution

$$\Psi^{(2)}(t,\mathbf{x}) = e^{-i\beta F(|\Psi^{(1,2)}(t_{n+1},\mathbf{x})|,\mathbf{x})(t-t_n) - iV(\mathbf{x})(t-t_n)}\Psi^{(1,2)}(t_{n+1},\mathbf{x}). \tag{5.22}$$

This finally gives: $\Psi^{n+1}(\mathbf{x}) \approx \Psi^{(2)}(t_{n+1},\mathbf{x})$. Let us remark that we have to compute the exponential of a matrix to effectively calculate (5.22).

For the full approximation, we adapt to each component the spectral approximation based on `fft` and `ifft` in space. For both the Lie and Strang schemes, the spectral approximation is written under a symmetrical form as for the one-component case.

5.2 Relaxation pseudo SPectral scheme (ReSP)

Introduced by Besse in [], the relaxation method is a scheme that looks like the Crank-Nicolson scheme but avoiding the resolution of the nonlinear term through a fixed point or a Newton-Raphson method.

5.2.1 A Relaxation pseudo SPectral scheme (ReSP) for the one-component case

If we consider the 2d cubic GPE

$$i\partial_t\psi(t,\mathbf{x}) = -\frac{1}{2}\Delta\psi(t,\mathbf{x}) + V(\mathbf{x},t)\psi(t,\mathbf{x}) + \beta|\psi(t,\mathbf{x})|^2\psi(t,\mathbf{x}) - \Omega L_z\psi, \mathbf{x}\in\mathbb{R}^2, t>0, \tag{5.23}$$

then the relaxation scheme is

$$\left\{ \begin{array}{l} \dfrac{\phi^{n+1/2}+\phi^{n-1/2}}{2} = \beta|\psi^n|^2, \\ i\dfrac{\psi^{n+1}-\psi^n}{\Delta t} = (-\dfrac{1}{2}\Delta - \Omega L_z)\left(\dfrac{\psi^{n+1}+\psi^n}{2}\right) + \dfrac{V^{n+1}\psi^{n+1}+V^n\psi^n}{2} \\ \qquad\qquad +\phi^{n+1/2}\left(\dfrac{\psi^{n+1}+\psi^n}{2}\right), \end{array} \right. \tag{5.24}$$

with the uniform time step $\Delta t = t_{n+1} - t_n$, and where $\phi^{n+1/2} = \phi(t_{n+1/2},\mathbf{x})$, $\psi^n = \psi(t_n,\mathbf{x})$ and $V^n = V(t_n,\mathbf{x})$. The initial conditions are $\psi^0(\mathbf{x}) = \psi_0(\mathbf{x})$ and $\phi^{-1/2}(\mathbf{x}) = \beta|\psi_0(\mathbf{x})|^2$. The extension to a general nonlinearity is direct.

We now have to discretize the operator $(-\Delta - \Omega L_z)$. To this end, we use the pseudospectral approximation of the spatial derivatives based on the Fourier series transforms (2.36)-(2.37), page 27. Keeping the same notations, we have the following discretization

$$\left\{ \begin{array}{l} \boldsymbol{\phi}^{n+1/2} = \mathbf{c}^{\mathrm{Re},n}, \\ \mathbb{A}^{\mathrm{RE},n}\boldsymbol{\psi} = \mathbf{b}^{\mathrm{Re},n}, \end{array} \right. \tag{5.25}$$

where $\mathbb{A}^{\mathrm{Re},n}$, $\mathbf{b}^{\mathrm{Re},n}$ and $\mathbf{c}^{\mathrm{Re},n}$ are such that

$$\begin{array}{l} \mathbb{A}^{\mathrm{Re},n} := \left(i\dfrac{\mathbb{I}}{\Delta t} + \dfrac{1}{4}[[\Delta]] - \dfrac{1}{2}\mathbb{V}^{n+1} - \dfrac{1}{2}[[\boldsymbol{\phi}^{n+1/2}]] + \dfrac{1}{2}\Omega\mathbb{L}_z\right), \\ \mathbf{b}^{\mathrm{Re},n} := \left(i\dfrac{\mathbb{I}}{\Delta t} - \dfrac{1}{4}[[\Delta]] + \dfrac{1}{2}\mathbb{V}^{n} + \dfrac{1}{2}[[\boldsymbol{\phi}^{n+1/2}]] - \dfrac{1}{2}\Omega L_z\right)\boldsymbol{\psi}^n, \\ \mathbf{c}^{\mathrm{Re},n} := 2\beta|\boldsymbol{\psi}^n|^2 - \boldsymbol{\phi}^{n-1/2}. \end{array} \tag{5.26}$$

Like for the stationary case, the evaluation of the differential operators is made through the FFT/iFFTs while the diagonal matrices in the physical space are applied directly. The linear system in (5.27) is solved at each iteration with the BiCGStab. A preconditioner, in the spirit of the Laplace/Thomas-Fermi preconditioners (see []), is used to accelerate the convergence of the iterative Krylov subspace solver.

The resulting scheme is called Relaxation pseudo SPectral scheme (ReSP). The scheme is second-order in time, and spectrally accurate in space. It computational cost is $\mathcal{O}(M \log M)$. The extension to the three-dimensional case and other nonlinearities is direct. Other properties are related to the fact that it is time reversible, mass and energy (for a cubic nonlinearity) conserving when the property holds at the continuous level. It is not time transverse invariant and the dispersion relation is not preserved. The scheme is unconditionally stable. We refer to [**?**] for more details.

5.2.2 Extension of the ReSP scheme to the multi-components case

The relaxation scheme can be extended to the multi-components situation in a similar way as the CNGF (2.6). We consider the same notations as before (see also page 87, system (5.14)). We have the following time discretization of system (5.14) based on the relaxation scheme (for a general nonlinearity)

$$\begin{aligned}
&\frac{\Phi^{n+1/2}+\Phi^{n-1/2}}{2} = \beta F(\Psi^n, \mathbf{x}),\ 1 \le n \le M,\ \mathbf{x} \in \mathbb{R}^d,\\
&\frac{\Psi^{n+1}-\Psi^n}{\Delta t} = -i\left(-\frac{1}{2}\Delta + V(\mathbf{x}) + \sum_{j=1}^{d} G^j(\mathbf{x})\partial_{x_j} + \Phi^{n+1/2}\right)\frac{\Psi^{n+1}+\Psi^n}{2},\ \mathbf{x} \in \mathbb{R}^d
\end{aligned}$$

for $n \ge 0$. Concerning the spatial discretization, we use again a pseudo spectral method based on the FFTs/iFFTs. Let us detail the two-dimensional case, e.g. $d = 2$. For the relaxation scheme, we have

$$\begin{aligned}
&\mathbb{M}^{\mathrm{Re},n+1/2} = 2\beta\mathbb{F}(\mathbf{\Psi}^n, \mathbf{x}) - \mathbb{M}^{\mathrm{Re},n-1/2},\\
&\mathbb{A}^{\mathrm{Re},n}\mathbf{\Psi}^{n+1} = \mathbb{B}^{\mathrm{Re},n}\mathbf{\Psi}^n,
\end{aligned} \tag{5.27}$$

where $\mathbf{\Psi}^n = ((\psi_1^n(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{O}_{J,K}}, ..., (\psi_{N_c}^n(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{O}_{J,K}})$ is the discrete unknown array in $\mathbb{C}^{MN_c}$, with $M := JK$.

The nonlinear operator $\mathbb{M}^{\mathrm{Re},n+1/2} \in \mathcal{M}_{MN_c}(\mathbb{C})$ is updated through the matrix

$$\mathbb{F}(\mathbf{\Psi}^n, \mathbf{x}) := \begin{pmatrix}
\mathbb{F}_{11}(\mathbf{\Psi}^n, \mathbf{x}) & \mathbb{F}_{12}(\mathbf{\Psi}^n, \mathbf{x}) & \cdots & \mathbb{F}_{1N_c}(\mathbf{\Psi}^n, \mathbf{x}) \\
\mathbb{F}_{21}(\mathbf{\Psi}^n, \mathbf{x}) & \mathbb{F}_{22}(\mathbf{\Psi}^n, \mathbf{x}) & \cdots & \mathbb{F}_{2N_c}(\mathbf{\Psi}^n, \mathbf{x}) \\
\vdots & \vdots & \ddots & \vdots \\
\mathbb{F}_{N_c1}(\mathbf{\Psi}^n, \mathbf{x}) & \mathbb{F}_{2N_c}(\mathbf{\Psi}^n, \mathbf{x}) & \cdots & \mathbb{F}_{N_cN_c}(\mathbf{\Psi}^n, \mathbf{x})
\end{pmatrix},$$

setting $\mathbb{F}_{\ell m}(\mathbf{\Psi}^n, \mathbf{x}) = \left(F_{\ell m}(\mathbf{\Psi}^n_{j,k}, \mathbf{x}_{j,k})\right)_{(j,k)\in\mathcal{O}_{J,K}}$, $1 \le \ell, m \le N_c$. To be consistent with the one-component case, we consider: $\mathbb{M}^{\mathrm{Re},-1/2} = \beta\mathbb{F}(\Psi_0(\mathbf{x}), \mathbf{x})$.

The operator $\mathbb{A}^{\mathrm{Re},n} \in \mathcal{M}_{MN_c}(\mathbb{C})$ is defined by

$$\begin{aligned}
&\mathbb{A}^{\mathrm{Re},n}\mathbf{\Psi} = \mathbb{A}^{\mathrm{Re},n}_{\mathrm{TF}}\mathbf{\Psi} + \mathbb{A}^{\mathrm{Re}}_{\Delta,\nabla}\mathbf{\Psi},\\
&\mathbb{A}^{\mathrm{Re},n}_{\mathrm{TF}}\mathbf{\Psi} := i\frac{\mathbb{I}_{N_c}}{\Delta t}\mathbf{\Psi} - \frac{1}{2}\left(\mathbb{V} + \mathbb{M}^{\mathrm{Re},n+1/2}\right)\mathbf{\Psi},\\
&\mathbb{A}^{\mathrm{Re}}_{\Delta,\nabla}\mathbf{\Psi} := \frac{1}{2}\left(\frac{1}{2}[[\Delta]] - \mathbb{G}^1[[\partial_x]] - \mathbb{G}^2[[\partial_y]]\right)\mathbf{\Psi}.
\end{aligned} \tag{5.28}$$

The finite dimensional operator $\mathbb{A}_{\mathrm{TF}}^{\mathrm{Re},n} \in \mathcal{M}_{MN_c}(\mathbb{C})$ is explicitly given through the matrices

$$\mathbb{I}_{N_c} := \begin{pmatrix} \mathbb{I} & 0 & \cdots & 0 \\ 0 & \mathbb{I} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbb{I} \end{pmatrix} \in \mathcal{M}_{MN_c}(\mathbb{R}), \qquad \mathbb{V} := \begin{pmatrix} \mathbb{V}_{11} & \mathbb{V}_{12} & \cdots & \mathbb{V}_{1N_c} \\ \mathbb{V}_{21} & \mathbb{V}_{22} & \cdots & \mathbb{V}_{2N_c} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{V}_{N_c1} & \mathbb{V}_{2N_c} & \cdots & \mathbb{V}_{N_cN_c} \end{pmatrix} \in \mathcal{M}_{MN_c}(\mathbb{R}),$$

where the diagonal matrices are defined by: $\mathbb{I}_{\ell m} = (\delta_{j,k})_{(j,k)\in\mathcal{O}_{J,K}} \in \mathcal{M}_M(\mathbb{R})$ and $\mathbb{V}_{\ell m}$ $= (V_{\ell m}(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{O}_{J,K}} \in \mathcal{M}_M(\mathbb{R})$. The block diagonal matrix $\mathbb{A}_{\Delta,\nabla}^{\mathrm{Re}}$ in (5.28) is implicitly given by the discrete differentiation operators *via* the FFTs/iFFTs

$$[[\Delta]]\mathbf{\Psi} := ([[\Delta\psi_\ell]])_{\ell=1,\dots,N_c} \in \mathbb{C}^{MN_c},$$

and

$$[[\partial_x]]\mathbf{\Psi} := ([[\partial_x\psi_\ell]])_{\ell=1,\dots,N_c} \in \mathbb{C}^{MN_c}, \quad [[\partial_y]]\mathbf{\Psi} := ([[\partial_y\psi_\ell]])_{\ell=1,\dots,N_c} \in \mathbb{C}^{MN_c}. \tag{5.29}$$

For $k = 1, 2$, we also define

$$\mathbb{G}^k := \begin{pmatrix} \mathbb{G}_{11}^k & \mathbb{G}_{12}^k & \cdots & \mathbb{G}_{1N_c}^k \\ \mathbb{G}_{21}^k & \mathbb{G}_{22}^k & \cdots & \mathbb{G}_{2N_c}^k \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{G}_{N_c1}^k & \mathbb{G}_{2N_c}^k & \cdots & \mathbb{G}_{N_cN_c}^k \end{pmatrix} \in \mathcal{M}_{MN_c}(\mathbb{C}),$$

setting $\mathbb{G}_{\ell m}^k = (G_{\ell m}^k(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{O}_{J,K}} \in \mathcal{M}_M(\mathbb{C})$.

The right hand side operator $\mathbb{B}^{\mathrm{Re},n} : \mathbb{C}^{MN_c} \to \mathbb{C}^{MN_c}$ is defined by

$$\begin{aligned} \mathbb{B}^{\mathrm{Re},n}\mathbf{\Psi} &= \mathbb{B}_{\mathrm{TF}}^{\mathrm{Re},n}\mathbf{\Psi} + \mathbb{B}_{\Delta,\Omega}^{\mathrm{Re}}\mathbf{\Psi}, \\ \mathbb{B}_{\mathrm{TF}}^{\mathrm{Re},n}\mathbf{\Psi} &:= i\frac{\mathbb{I}_{N_c}}{\Delta t}\mathbf{\Psi} + \frac{1}{2}\left(\mathbb{V} + \mathbb{M}^{\mathrm{Re},n+1/2}\right)\mathbf{\Psi}, \\ \mathbb{B}_{\Delta,\nabla}^{\mathrm{Re}}\mathbf{\Psi} &:= \frac{1}{2}\left(-\frac{1}{2}[[\Delta]] + \mathbb{G}^1[[\partial_x]] + \mathbb{G}^2[[\partial_y]]\right)\mathbf{\Psi}. \end{aligned} \tag{5.30}$$

For solving the second equation of (5.27), we again use the preconditioned BiCGStab. Concerning the TF(-like) preconditioner, since we have some coupling effects between the gazes through $\mathbb{V}$ and $\mathbb{M}^{\mathrm{Re},n+1/2}$, the matrix $\mathbb{A}_{\mathrm{TF}}^{\mathrm{Re},n}$ is non diagonal. We propose here to only keep the diagonal part of $\mathbb{A}_{\mathrm{TF}}^{\mathrm{Re},n}$ for preconditioning, that is, to include the potential and nonlinear self-interactions in each gas. Concretely, we build the following *diagonal* TF preconditioner $\mathbb{P}_{\mathrm{TF,diag}}^{\mathrm{Re},n}$ given by

$$\mathbb{P}_{\mathrm{TF,diag}}^{\mathrm{Re},n} := \left(i\frac{\mathbb{I}_{N_c}}{\Delta t} - \frac{1}{2}\mathbb{V}_{\mathrm{diag}} - \frac{1}{2}\mathbb{M}_{\mathrm{diag}}^{\mathrm{Re},n+1/2}\right)^{-1},$$

where $\mathbb{V}_{\mathrm{diag}} := (\mathbb{V}_{\ell\ell})_{\ell=1,\dots,N_c}$ and $\mathbb{M}_{\mathrm{diag}}^{\mathrm{Re},n+1/2} := (\mathbb{M}_{\ell\ell}^{\mathrm{Re},n+1/2})_{\ell=1,\dots,N_c}$.

We can also build the *full* TF preconditioner $\mathbb{P}_{\mathrm{TF,full}}^{\mathrm{Re},n}$ by including the full potential and nonlinear operators and inverting the matrix. This preconditioner is therefore given by

$$\mathbb{P}_{\mathrm{TF,full}}^{\mathrm{Re},n} := \left(i\frac{\mathbb{I}_{N_c}}{\Delta t} - \frac{1}{2}\mathbb{V} - \frac{1}{2}\mathbb{M}^{\mathrm{Re},n+1/2}\right)^{-1}.$$

Finally, we have the Laplace preconditioner $\mathbb{P}_{\Delta}^{\mathrm{Re}}$ which is built by inverting the laplacian, which is a diagonal operator in the frequencies space, by using a fast Fourier transform. It is given by

$$\mathbb{P}_{\Delta}^{\mathrm{Re}} := \left(i\frac{\mathbb{I}_{N_c}}{\Delta t} + \frac{1}{4}[[\Delta]]\right)^{-1}.$$

5.3 Integration of a stochastic potential

In this section, we consider a Gross-Pitaevskii equation with a stochastic potential. We are interested on how to discretize in time equations of the form

$$i\partial_t\psi(t,\mathbf{x}) = -\frac{1}{2}\Delta\psi(t,\mathbf{x}) + V(\dot{w}(t),\mathbf{x})\psi(t,\mathbf{x}) + \beta|\psi(t,\mathbf{x})|^2\psi(t,\mathbf{x}) - \Omega L_z\psi, \mathbf{x}\in\mathbb{R}^2, t>0,$$

where we define the stochastic potential as

$$V(\dot{w}(t),\mathbf{x}) := \sum_{j=1}^{N_{\text{stoch}}} V_j(\mathbf{x})\dot{w}_j(t)$$

and $\{\dot{w}(t)\}_{t\geq 0}$ is the time derivative of a multidimensional noise $w(t) = (w_1(t), ..., w_{N_{\text{stoch}}}(t))$. Here, we will focus ourselves on the two previous schemes that we have introduced to solve such equations: the time splitting scheme and the relaxation scheme.

5.3.1 The time splitting scheme

In the time splitting scheme (see section 5.1, page 83), we essentially split the equation in two equations that we solve separately. Here, we follow the same procedure. In our case, given a time discretization $t_0 < t_1 < ... < t_n < ...$, with $\Delta t = \Delta t_n = t_{n+1} - t_n$, and an initial wave function $\psi^0(\mathbf{x})$, we obtain the following scheme

1) let $\Delta t > 0$, $n \in \mathbb{N} - \{0\}$ and $n\Delta t < t \leq (n+1)\Delta t$, we solve

$$\begin{cases} i\partial_t\psi_1(t,\mathbf{x}) = -\frac{1}{2}\Delta\psi_1(t,\mathbf{x}) - \Omega L_z\psi_1(t,\mathbf{x}),\ n\Delta t < t \leq (n+1)\Delta t, \\ \psi_1(n\Delta t,\mathbf{x}) = \psi^n(\mathbf{x}), \end{cases} \tag{5.31}$$

2) and next

$$\begin{cases} i\partial_t\psi_2(t,\mathbf{x}) = V(\dot{w}(t),\mathbf{x})\psi_2(t,\mathbf{x}) + \beta|\psi_2(t,\mathbf{x})|^2\psi_2(t,\mathbf{x}),\ n\Delta t < t \leq (n+1)\Delta t, \\ \psi_2(n\Delta t,\mathbf{x}) = \psi_1((n+1)\Delta t,\mathbf{x}). \end{cases} \tag{5.32}$$

Equation (5.31) is solved using the ADI technique and fast Fourier transforms (see section 5.1.1, page 85). Concerning equation (5.32), we have seen in section 5.1.1 that we can exactly integrate the nonlinearity and the time-dependent potential in time. Therefore we obtain, for all $n\Delta t < t \leq (n+1)\Delta t$,

$$\psi_2(t,\mathbf{x}) = \psi_1((n+1)\Delta t,\mathbf{x})e^{-i\beta|\psi_1((n+1)\Delta t,\mathbf{x})|^2(t-t_n) - i\int_{t_n}^t V(\dot{w}(s),\mathbf{x})ds}.$$

The time integration of the stochastic potential gives in fact

$$\int_{t_n}^t V(\dot{w}(s),\mathbf{x})ds = \sum_{j=1}^{N_{\text{stoch}}}\int_{t_n}^t V_j(\mathbf{x})\dot{w}_j(s)ds = \sum_{j=1}^{N_{\text{stoch}}} V_j(\mathbf{x})(w_j(t) - w_j(t_n)) = V(w(t) - w(t_n),\mathbf{x}).$$

This leads to the following integration for equation (5.32)

$$\psi_2(t,\mathbf{x}) = \psi_1((n+1)\Delta t,\mathbf{x})e^{-i\beta|\psi_1((n+1)\Delta t,\mathbf{x})|^2(t-t_n) - iV(w(t)-w(t_n),\mathbf{x})}.$$

5.3.2 The relaxation scheme

In section 5.2, page 89, we have introduced the relaxation scheme. The main problem in the case of our stochastic equation lies in the time discretization of the noise. Considering a time discretization $t_0 < t_1 < ... < t_n < ...$, with $\Delta t = \Delta t_n = t_{n+1} - t_n$, and an initial wave function $\psi^0(\mathbf{x})$, we propose the following time discretization

$$\begin{aligned}\int_{t_n}^{t_{n+1}} V(\dot{w}(s),\mathbf{x})\psi(s,\mathbf{x})ds &= \sum_{j=1}^{N_{\text{stoch}}} V_j(\mathbf{x}) \int_{t_n}^{t_{n+1}} \dot{w}_j(s)\psi(s,\mathbf{x})ds \\ &\approx \sum_{j=1}^{N_{\text{stoch}}} V_j(\mathbf{x}) \frac{\psi(t_{n+1},\mathbf{x}) + \psi(t_n,\mathbf{x})}{2}(w_j(t_{n+1}) - w_j(t_n)).\end{aligned}$$

Therefore, for our stochastic Gross-Pitaevskii equation, we obtain the following relaxation scheme

$$\begin{cases} \dfrac{\phi^{n+1/2} + \phi^{n-1/2}}{2} = \beta|\psi^n|^2, \\ i\dfrac{\psi^{n+1} - \psi^n}{\Delta t} = (-\dfrac{1}{2}\Delta - \Omega L_z + V^n)\left(\dfrac{\psi^{n+1} + \psi^n}{2}\right) + \phi^{n+1/2}\left(\dfrac{\psi^{n+1} + \psi^n}{2}\right), \end{cases} \tag{5.33}$$

where $\phi^{n+1/2} = \phi(t_{n+1/2},\mathbf{x})$, $\psi^n = \psi(t_n,\mathbf{x})$ and $V^n = V\left(\frac{w(t_{n+1})-w(t_n)}{\Delta t},\mathbf{x}\right)$. The initial conditions are $\psi^0(\mathbf{x}) = \psi_0(\mathbf{x})$ and $\phi^{-1/2}(\mathbf{x}) = \beta|\psi_0(\mathbf{x})|^2$.

5.4 Richardson's extrapolation

The Richardson's extrapolation is a simple method that linearly combines two solutions given by a numerical scheme on two different time steps to obtain a new solution with an increased accuracy. Let us assume that we can numerically compute an approximation $u_{\Delta t}$ of an exact quantity u by using a scheme of order $p \in \mathbb{N}$

$$u_{\Delta t} = u + C_p(\Delta t)^p + C_{p+1}(\Delta t)^{p+1} + o((\Delta t)^{p+1}). \tag{5.34}$$

Similarly, for a time step $\Delta t/2$, one gets

$$u_{\Delta t/2} = u + C_p\frac{(\Delta t)^p}{2^p} + C_{p+1}\frac{(\Delta t)^{p+1}}{2^{p+1}} + o((\Delta t)^{p+1}). \tag{5.35}$$

The Richardson's extrapolation method consists in combining the approximations (5.34) and (5.35) to eliminate the error term $C_p(\Delta t)^p$. It is easy to see that

$$v_{\Delta t/2} = \frac{2^p u_{\Delta t/2} - u_{\Delta t}}{2^p - 1} = u - C_{p+1}\frac{(\Delta t)^{p+1}}{2(2^p - 1)} + o((\Delta t)^{p+1}).$$

This equation provides an approximation $v_{\Delta t/2}$ which is at least of order $p+1$. In fact, depending on the scheme used, it is possible to even gain a higher order of accuracy. This process can be used iteratively to get high-order schemes. This results in the following triangular table

Approximations		Extrapolations	
Order p	Order $p+1$	Order $p+2$	Order $p+3$
$u_{\Delta t}$			
$u_{\Delta t/2}$	$v_{\Delta t/2} = \frac{2^p u_{\Delta t/2} - u_{\Delta t}}{2^p - 1}$		
$u_{\Delta t/4}$	$v_{\Delta t/4} = \frac{2^p u_{\Delta t/4} - u_{\Delta t/2}}{2^p - 1}$	$w_{\Delta t/4} = \frac{2^{p+1} v_{\Delta t/4} - v_{\Delta t/2}}{2^{p+1} - 1}$	
$u_{\Delta t/8}$	$v_{\Delta t/8} = \frac{2^p u_{\Delta t/8} - u_{\Delta t/4}}{2^p - 1}$	$w_{\Delta t/8} = \frac{2^{p+1} v_{\Delta t/8} - v_{\Delta t/4}}{2^{p+1} - 1}$	$x_{\Delta t/8} = \frac{2^{p+2} w_{\Delta t/8} - w_{\Delta t/4}}{2^{p+2} - 1}$

In GPELab, the Richardson's extrapolation method is used for the TSSP and ReSP schemes to get an increased accuracy in time.

5.5 GPELab functions for the dynamics

As for the computation of ground states, GPELab contains functions that are used to define variables or functions in order to set the problem in the first place then to compute interesting quantities. A typical script to compute a dynamical problem is very similar to a script to compute a ground state problem. However, some changes have to be made. In this section, we'll state the functions that need to be used and how they differ from the one used in the stationary case.

First, we have to define the `Method` and `Geometry` variables. This is done by using the `Method_Vard2d` and `Geometry2D_Var2d` functions. The `Geometry2D_Var2d` is handled in the same way as in the stationary state, we refer to section 3.5.2, page 43, on how to use it. The `Method_Var2d` function is the same as the one in the stationary state (see section 3.5.1, page 41), however we recall the arguments that are important when computing a dynamical problem.

5.5.1 The `Method_Var2d` function

```
Method = Method_Var2d(Computation,Ncomponents, Type, Deltat, Stop_time, Stop_crit,
Max_iter, Precond_type, Output, Splitting, BESP, Solver_FD, Iterative_tol,
Iterative_maxit);
```

Table 5.1: The `Method_Var2d` function.

The `Method_Var2d` function creates the `Method` *structure* that contains all the parameters relative to the method. We refer to section 3.5.1, page 41, for further details. The optional arguments are the following

- `Computation` ($\mathbb{S}$,`'Ground'`) is a variable that must be `'Dynamic'` to compute the dynamic of the GPE.
- `Ncomponents` ($\mathbb{N}$,`1`) is a variable corresponding to the number of components that describe the condensate.
- `Type` ($\mathbb{S}$, `'BESP'`) is a variable corresponding to the scheme used in the computation. In the case of a dynamical computation, it must either be `'Splitting'` to use a splitting scheme (see section 5.1, page 83) or `'Relaxation'` to use the relaxation scheme (see section 5.2, page 89).
- `Deltat` ($\mathbb{R}^+$,`1e-3`) is a variable corresponding to the time step of the method. The time discretization is always uniform.

- `Stop_time` ($\mathbb{R}^+$,`1`) is a variable corresponding to the final time of computation in the case of a dynamic problem. This corresponds to the following stopping criterion

```
Iter = 0
while { Deltat*Iter < Stop_time }
 Compute the solution of the dynamic problem at iteration Iter for Deltat
 Iter = Iter + 1
end
```

- `Stop_crit` ($\mathbb{R}^+$,`1e-8`) is a variable corresponding to the stopping criterion in the case of a ground state computation (see section 3.5.1, page 41).

- `Max_iter` ($\mathbb{N}$, `1e6`) is a variable corresponding to the maximum number of iterations for a stationary state computation.

- `Preconditioner` ($\mathbb{S}$,`'ThomasFermi'`) is a variable that must be either `'None'` for a calculation without preconditioner, `'Laplace'` for the Laplace preconditioner and `'ThomasFermi'` for the Thomas Fermi preconditioner.

- `Output` ($\mathbb{N}$,`1`) is a variable that must either be 1 if one computes outputs during the computations or 0 if not.

- `Splitting` ($\mathbb{S}$,`'Strang'`) is a variable that must be either `'Lie'` to use the Lie type of splitting, `'Strang'` to use the Strang type of splitting or `'Fourth'` to use the Fourth-order type of splitting for the splitting scheme.

- `BESP` ($\mathbb{N}$,`0`) is a variable corresponding to the type of method used to invert the linear system in BESP (see section 3.5.1, page 41).

- `Solver_FD` ($\mathbb{N}$,`0`) is a variable corresponding to the type of method used to invert the linear system in BEFD (see section 3.5.1, page 41)

- `Iterative_tol` ($\mathbb{R}^+$, `1e-9`) is a variable corresponding to the stopping criterion related to the difference between two successive iterates in the Krylov solver.

- `Iterative_maxit` ($\mathbb{N}$,`1e3`) is a variable corresponding to the stopping criterion related to the maximum number of iterations in the Krylov solver.

For example, we want to compute the dynamic solution for a single-component BEC by using a splitting scheme. We choose a time step $\Delta t = 10^{-3}$, a stopping time $T = 1$ and the Laplace preconditioner. Moreover we choose to compute outputs during the simulation. This gives the code in table 5.2.

We now want to set the physical problem. This is done exactly like in section 3.6, page 43. Here we list some of the physical operators that are used only in a dynamic problem.

5.5.2 The `TimePotential_Var2d` function

In the case where one wants to compute the dynamics of GPE, GPELab offers the possibility of handling a time dependent potential. The `TimePotential_Var2d` function allows to define the time-dependent potential operator (i.e. $\mathbf{V}(t, \mathbf{x})$) in the problem by modifying the `Physics2D` structure. It must be provided with the `Method` and `Physics2D` structures and takes the following optional arguments

```
Computation = 'Dynamic;
Ncomponents = 1;
Type = 'Splitting';
Deltat = 1e-3;
Stop_time = 1;
Stop_crit = [];
Max_iter = [];
Precond_type = 'Laplace';
Output = 1;
Method = Method_Var2d(Computation,Ncomponents, Type, Deltat, Stop_time, Stop_crit,
Max_iter, Precond_type, Output;
```

Table 5.2: An example of initialization and use of the `Method_Var2d` function.

```
Physics2D = TimePotential_Var2d(Method, Physics2D, TimePotential, G );
```

Table 5.3: The `TimePotential_Var2d` function.

- `TimePotential`: If a function `TimePotential` in $\mathbb{F}(\mathbb{R}^+, \mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))$ is provided, the physical time-dependent potential will be defined as follows, for each $j,k \in \{1,...,N_c\}$,

$$\mathbf{V}_{j,k}(t,x,y) = \begin{cases} \texttt{TimePotential}(t,x,y) \text{ if } j=k \\ 0 \text{ if } j \neq k \end{cases}$$

  If `TimePotential` is a cell array of functions in

$$\mathcal{C}_{N_c,N_c}\{\mathbb{F}(\mathbb{R}^+, \mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))\},$$

  then the potential will be defined by

$$\mathbf{V}_{j,k}(t,x,y) = \texttt{TimePotential}\{j,k\}(t,x,y)$$

  for $j,k \in \{1,...,N_c\}$. The default argument is `quadratic_potential2d` which corresponds to

$$\mathbf{V}_{j,k}(t,x,y) = \begin{cases} \frac{1}{2}(x^2+y^2) \text{ if } j=k \\ 0 \text{ if } j \neq k \end{cases}$$

  Note that in the case of a stationary state computation, the potential operator should be time-independent.

- `G` ($\mathcal{M}_{N_c,N_c}(\mathbb{C})$, `ones(N_c)`) is a complex-valued variable that is used to multiply each component of the potential

$$\mathbf{V}_{j,k}(t,x,y) = \texttt{G}(j,k)\texttt{TimePotential}\{j,k\}(t,x,y)$$

  for $j,k \in \{1,...,N_c\}$.

For example, we want to set a quadratic potential with an intensity which varies in time. We would like to set $V(t,\mathbf{x}) = (\frac{3}{2}+\cos(t))|\mathbf{x}|^2$ for each component. To this end, we first need to create a square cell array of functions such that the diagonal part is the desired potential and 0 otherwise. This is done in table 5.4 where we set the cell array of functions and then modify the `Physics2D` by using the `TimePotential_Var2d` function.

```
function P = Example_timepotential(Method)
Ncomp = Method.Ncomponents;
P = cell(Ncomp);
for j = 1:Ncomp
for k = 1:Ncomp
if (j==k)
P{j,k} = @(t,x,y) (3/2+cos(t))*(x.^2+y.^2)
else
P{j,k} = @(t,x,y) 0
end
end
end
Physics2D = TimePotential_Var2d(Method, Physics2D, Example_timepotential(Method)
);
```

Table 5.4: An example to use the `TimePotential_Var2d` function.

```
Physics2D = StochasticPotential_Var2d(Method, Physics2D, StochasticPotential, G ,
StochasticProcess);
```

Table 5.5: The `StochasticPotential_Var2d` function.

5.5.3 The `StochasticPotential_Var2d` function

In the case where one wants to compute the dynamics of GPE, GPELab offers the possibility of handling a stochastic potential. That is, a potential that is defined using the derivative of a stochastic processes (denoted here by $\dot{w}(t)$). The `StochasticPotential_Var2d` function allows to define the stochastic potential operator (i.e. $\mathbf{V}(\dot{w}(t), \mathbf{x})$) in the problem by modifying the `Physics2D` structure. It must be provided with the `Method` and `Physics2D` structures and takes the following optional arguments

- `StochasticPotential`: If a function `StochasticPotential` in $\mathbb{F}(\mathbb{R}^+, \mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))$ is provided, the physical time-dependent potential will be defined as follows, for each $j, k \in \{1, ..., N_c\}$,

$$\mathbf{V}_{j,k}(\dot{w}(t), x, y) = \begin{cases} \texttt{StochasticPotential}(\dot{w}(t), x, y) \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}$$

  If `StochasticPotential` is a cell array of functions in

$$\mathcal{C}_{N_c,N_c}\{\mathbb{F}(\mathbb{R}^+, \mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))\},$$

  then the potential will be defined by

$$\mathbf{V}_{j,k}(\dot{w}(t), x, y) = \texttt{StochasticPotential}\{j, k\}(\dot{w}(t), x, y)$$

  for $j, k \in \{1, ..., N_c\}$. The default argument is `quadratic_potential2d` which corresponds to

$$\mathbf{V}_{j,k}(\dot{w}(t), x, y) = \begin{cases} \frac{1}{2}(x^2 + y^2) \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}.$$

- `G` ($\mathcal{M}_{N_c,N_c}(\mathbb{C})$, `ones(N_c)`) is a complex-valued variable that is used to multiply each component of the potential

$$\mathbf{V}_{j,k}(\dot{w}(t)x, y) = \texttt{G}(j, k)\texttt{StochasticPotential}\{j, k\}(\dot{w}(t), x, y)$$

  for $j, k \in \{1, ..., N_c\}$.

- `StochasticProcess` is a function or a cell vector of functions that is used as the stochastic process $w(t)$ when computing the `StochasticPotential`. If a function is defined, the `StochasticPotential` will be computed using a scalar value $\dot{w}(t)$. If a cell vector of functions is defined, the `StochasticPotential` will be computed using a cell vector of scalar value $\dot{w}_j(t)$.

For example, we want to set a stochastic quadratic potential. We would like to set $V(\dot{w}(t), \mathbf{x}) = 1/2(x^2\dot{w}_1(t) + y^2\dot{w}_2(t))$ for each component, where w_1 and w_2 are brownian motions. To this end, we first need to create a square cell array of functions such that the diagonal part is the desired potential and 0 otherwise. This is done in table 5.4 where we set the cell array of functions, then create a cell vector of two brownian motions and then modify the `Physics2D` by using the `StochasticPotential_Var2d` function.

```
function P = Example_stochasticpotential(Method)
Ncomp = Method.Ncomponents;
P = cell(Ncomp);
for j = 1:Ncomp
for k = 1:Ncomp
if (j==k)
P{j,k} = @(W,x,y(1/2)*(x.^2*W{1}+y.^2 W{2})
else
P{j,k} = @(W,x,y) 0
end
end
end
functionW = Example_stochasticprocess(Method)
W = cell(1,2);
W{1} = Brownian_Process2d(Method);
W{2} = Brownian_Process2d(Method);
end
Physics2D = TimePotential_Var2d(Method, Physics2D, ...
Example_stochasticpotential(Method), [] , Example_stochasticprocess(Method) );
```

Table 5.6: An example to use the `StochasticPotential_Var2d` function.

The initial data can be set using the `InitialData_Var2d` function (see section 3.6.6, page 53) just like in the stationary case. The outputs of a computation are set using the `OutputsINI_Var2d` function (see section 3.7.1, page 53). We remark that the outputs are computed like in the stationary case, that is we have to set the variable `Evo_outputs` which corresponds to the number of iterations between each computation of the outputs. Here the number of iterations corresponds to the number of time steps. We also need the `Print` and `Figure` structures, which are built using the `Print_Var2d` function (see section 3.7.3, page 58) and `Figure_Var2d` function (see sections 3.7.4, page 59, and 3.7.5, page 59) respectively. Finally, we use the `GPELab2d` function to launch the simulation (see section 3.7.6, page 60).

5.6 Examples of computations

5.6.1 Dynamic of a bright soliton for the Gross-Pitaevskii equation with cubic nonlinearity in 1D

We now show how to compute the dynamic of a soliton for Gross-Pitaevskii equation with cubic nonlinearity in 1D with a single component. First, we have to build the `Method` and `Geometry1D`

structures. We want to use a 'Splitting' scheme and a step time of 10^{-3}, with a spatial grid of $2^{10}+1$ points on the interval $[-15, 15]$. Moreover, we want to set the stopping time to 1. All of this is coded in Table 5.7.

```
Computation = 'Dynamic';
Ncomponents = 1;
Type = 'Splitting';
Deltat = 1e-3;
Stop_time = 1;
Stop_crit = [];
Method = Method_Var1d(Computation,Ncomponents, Type, Deltat, Stop_time ,
Stop_crit);
xmin = -15;
xmax = 15;
Nx = 2^10+1;
Geometry1D = Geometry1D_Var1d(xmin,xmax, Nx);
```

Table 5.7: Building the `Method` and `Geometry1D` structures.

We now have to define the physics of the problem. We want to compute the soliton for the following Gross-Pitaevskii equation.

$$i\partial_t \Psi(t,x) = \Delta\Psi(t,x) + \beta|\Psi(t,x)|^2\Psi(t,x)$$

with $\beta = -10$. We know that the default nonlinearity is the cubic nonlinearity. Therefore, we only have to build the `Physics1D` structure with the desired coefficients and to add the default non linear operator to the physics of the problem. We report on Table 5.8 how it is coded.

```
Delta = 1;
Beta = -10;
Physics1D = Physics1D_Var1d(Method,Delta,Beta);
Physics1D = Nonlinearity_Var1d(Method, Physics1D);
```

Table 5.8: Setting the `Physics1D` structure and adding the nonlinear operator.

We then set the initial function to use for the computation. We want to simulate a soliton and we would like the initial data to be like

$$\Psi_0(x) = \sqrt{\frac{2a}{100}}\mathrm{sech}(\sqrt{a}x)\exp\left(i\frac{c}{2}x + i\theta_0\right)$$

To do so, we have to use the mesh grid contained in the geometry structure and define the initial data. We choose the following parameters: $a = 1$, $c = 5$ and $\theta_0 = 0$. This is done in Table 5.9.

```
a = 1 ;
c = 5 ;
theta0 = 0 ;
X = Geometry1D.X ;
Phi_0{1} = sqrt(2*a/100)*sech(sqrt(a)*X).*exp(1i*c*X/2+1i*theta0);
```

Table 5.9: Building the initial data.

We want to save the position of the center of the soliton, corresponding to the position of its highest module, as an output. Therefore, we define the function that locates the center of the soliton, which done in Table 5.10.

```
function [X_center] = Soliton_center(Phi,X)
[Max_Phi,I_center] = max(abs(Phi));
X_center = X(I_center);
```

Table 5.10: Creating a function to locate the center of the soliton.

We finally set the outputs and the printing informations, then we launch the simulation. We choose to print informations in the command window every 15 iterations and to draw a figure of the square of the module of the solution. Moreover, we add the previous function as an output to compute with the name `'Center of the soliton'` and we want it to be computed each 10 iterations. We do not save the solution during the simulation. This is done in Table 5.11.

```
Solution_save = 0;
Outputs_iterations = 10;
Output_function{1} = @(Phi,X)Soliton_center(Phi,X);
Output_name{1} = 'Center of the soliton';
Outputs = OutputsINI_Var1d(Method,Outputs_iterations,Solution_save,Output_function,...
Output_name);
Printing = 1;
Evo = 15;
Draw = 1;
Print = Print_Var1d(Printing,Evo,Draw);
[Phi,Outputs]= GPELab1d(Phi_0,Method,Geometry1D,Physics1D,Outputs,[],Print);
```

Table 5.11: Setting the outputs and the `Print` structure then launching the simulation.

At the end of the simulation, we can retrieve the informations about the soliton using the `Outputs` structure. For instance, we have computed the center of the soliton at each 10 iterations. We can print it using the `plot` function from Matlab. This is coded in Table 5.12.

```
Time = [0:0.01:1];
plot(Time, Outputs.User_defined_local{1})
xlabel('Time')
ylabel('Center of the soliton')
```

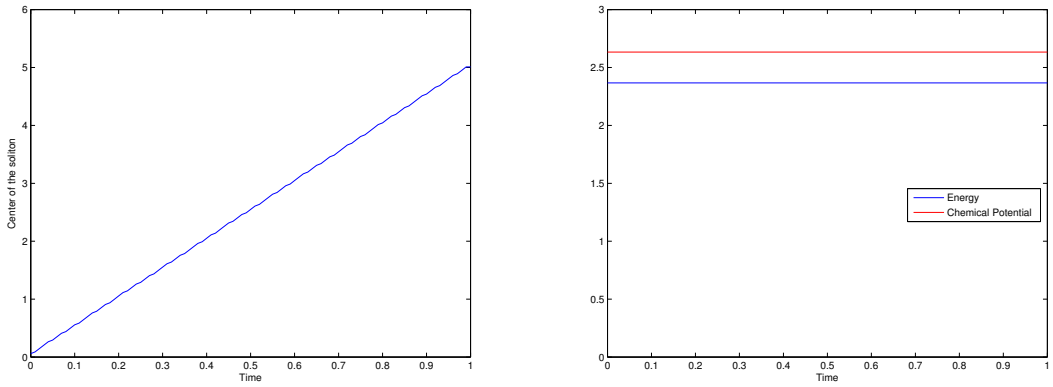
Table 5.12: Plotting the evolution of the solition center.

We obtain Figure 5.1(a). We can also plot the evolution of the energy and the chemical potential, which are conserved quantities, and can be seen on Figure 5.1(b).

5.6.2 Dynamic of a dark soliton in a Bose-Einstein condensate in 2D

We now show how to compute the dynamic of a dark soliton for Gross-Pitaevskii equation with cubic nonlinearity in 2D with a single component. We first compute a ground state for the Gross-Pitaevskii equation with quadratic potential and cubic nonlinearity, which corresponds to a stationary Bose-Einstein condensate. We have to build the `Method` and `Geometry2D` structures. Here, we choose to use the BESP scheme to compute a stationary state. We fix the time step $\Delta t = 10^{-1}$ and the stopping criterion $\varepsilon = 10^{-8}$. The geometry is set on a computational domain $\mathcal{O} =]-10, 10[\times]-10, 10[$ and the number of grid points $J = K = 2^9$. This is coded in Table 5.13.

Now, we have to define the physical problem. In this case, we want to compute a stationary state corresponding to a Bose-Einstein condensate in a quadratic trap. This corresponds to the



(a) Evolution of the center of the soliton.

(b) Evolution of the energy of the soliton.

Figure 5.1: Some outputs computed during the simulation.

```
Computation = 'Ground';
Ncomponents = 1;
Type = 'BESP';
Deltat = 1e-1;
Stop_time = [];
Stop_crit = 1e-8;
Method = Method_Var2d(Computation,Ncomponents, Type, Deltat, Stop_time ,
Stop_crit);
xmin = -10;
xmax = 10;
ymin = -10;
ymax = 10;
Nx = 2^9+1;
Ny = 2^9+1;
Geometry2D = Geometry2D_Var2d(xmin,xmax, ymin,ymax, Nx, Ny);
```

Table 5.13: Building the `Method` and `Geometry2D` structures for the computation of a stationary state.

following Gross-Piteavskii equation

$$i\partial_t \Psi(x,y,t) = \frac{1}{2}\Delta\Psi(x,y,t) + \frac{1}{2}\left(|x|^2 + |y|^2\right)\Psi(x,y,t) + \beta|\Psi(x,y,t)|^2\Psi(x,y,t)$$

where we choose $\beta = 10000$. Therefore, we have to add the potential and nonlinear operators using the `Potential_Var2d` and `Nonlinearity_Var2d`. The resulting code in available in Table 5.14.

```
Delta = 0.5;
Beta = 10000;
Physics2D = Physics2D_Var2d(Method,Delta,Beta);
Physics2D = Potential_Var2d(Method, Physics2D, @(x,y) (1/2)*(x.^2+y.^2));
Physics2D = Nonlinearity_Var2d(Method, Physics2D, @(phi,x,y) abs(phi).^2 );
```

Table 5.14: Setting the `Physics2D` structure to compute the stationary state.

We wish to use the Thomas-Fermi approximation as an initial wave function for the computation.

Therefore, we set the initial data using the `InitialData_Var2d` as in Table 5.15

```
InitialData_choice = 2 ;
Phi_0 = InitialData_Var2d(Method, Geometry2D, Physics2D,InitialData_choice);
```

Table 5.15: Initialization by the Thomas-Fermi approximation.

We finally launch the simulation. We set the defaults `Outputs` using the `OutputsINI_Var2d` function and the default `Print` structure using the `Print_Var2d` function. Then we launch the computation using the `GPELab2d` function and store the ground state under the variable `Phi_1`. This is done in Table 5.16

```
Outputs = OutputsINI_Var2d(Method);
Printing = 1;
Evo = 15;
Draw = 1;
Print = Print_Var2d(Printing,Evo,Draw);
[Phi_1,Outputs]= GPELab2d(Phi_0,Method,Geometry2D,Physics2D,Outputs,[],Print);
```

Table 5.16: Launching the computation of the ground state.

At the end of the computation, we obtain the ground state whose modulus is depicted in Figure 5.2.

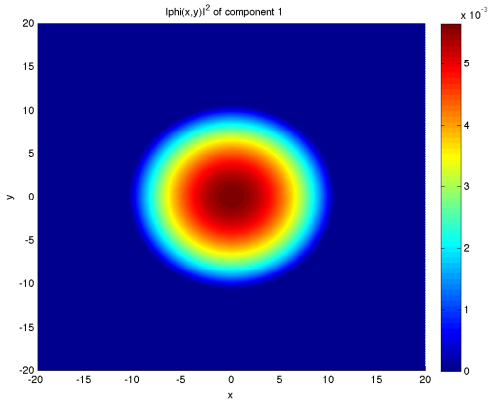


Figure 5.2: Ground state computed with GPELab using the parameters from Section 5.6.2.

It is now possible to phase imprint a dark soliton in the Bose-Einstein condensate and simulate its time evolution. First, we have to rebuild the `Method` structure for this dynamical problem. We want to use a 'Relaxation' scheme and a step time of 10^{-3}. Moreover, we want to set the stopping time to 1.5. We coded this in GPELab as in Table 5.17.

We now have to phase-imprint the dark soliton [24]. This is done by multiplying the initial data by

$$\xi(x,y) = e^{i\frac{\Delta\theta_0}{2}\left(1+\tanh\left(\frac{x-x_0}{s}\right)\right)}$$

where $\Delta\theta_0 = \pi/3$, $x_0 = 5$ and $s = 0.2$. This is done in Table 5.18 where we multiply the ground state `Phi_1` by the previous function.

We finally set the outputs and the printing informations, then we launch the simulation. We choose to print informations in the command window every 15 iterations and to draw a figure of

```
Computation = 'Dynamic';
Ncomponents = 1;
Type = 'Relaxation';
Deltat = 1e-3;
Stop_time = 1.5;
Stop_crit = [];
Method = Method_Var1d(Computation,Ncomponents, Type, Deltat, Stop_time ,
Stop_crit);
```

Table 5.17: Building the `Method` structure for a dynamical problem.

```
X_0 = 5;
ăDelta_theta_0 = pi/3;
s = 0.2;
ăPhi_1{1} = Phi_1{1}.*exp(1i*(Delta_theta_0/2)*(1+tanh((Geometry2D.X-X_0)./s)));
```

Table 5.18: Phase-imprinting the ground state with a dark soliton.

```
save = 1;
Evo_save = 100;
Outputs = OutputsINI_Var1d(Method,save,Evo_save);
Printing = 1;
Evo = 15;
Draw = 1;
Print = Print_Var1d(Printing,Evo,Draw);
Figure = Figure_Var1d('hot');
[Phi,Outputs]= GPELab1d(Phi_1,Method,Geometry1D,Physics1D,Outputs,[],Print,Figure);
```

the square of the module of the solution. Moreover, we choose to draw the figure using the `'hot'` colormap, which is done using the function `Figure_Var2d` and to save the solution.

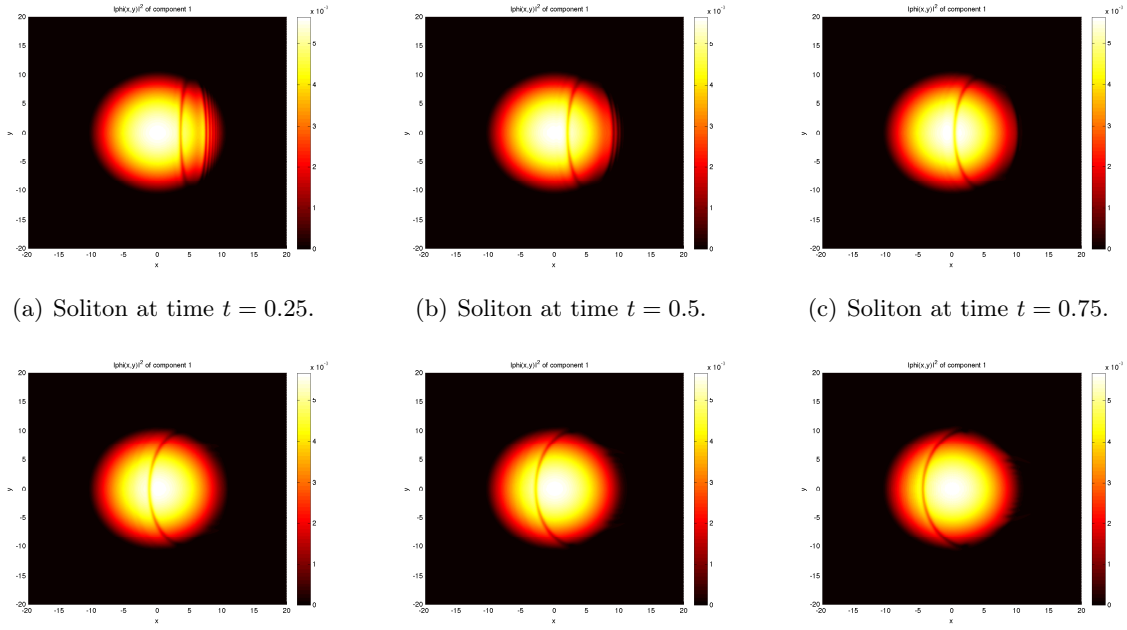
At the end of the simulation, we obtain the saved solution in the outputs which can be used to print the modulus of the solution and show the evolution of the dark soliton in the Bose-Einstein condensate. This can be seen on Figure 5.3. We can also plot the evolution of the energy

5.6.3 Dynamic of a rotating Bose-Einstein condensate in 2D

We now show how to compute the dynamic of a fast rotating Bose-Einstein condensate when changing the intensity of the potential. The initial data for this simulation is the ground state computed in section 4.4. Therefore, we assume that `Phi_1` is the ground state that is already computed.

First, we build the `Method` and `Geometry2D` structures. We remark that the geometry must be the same as in section 4.4. We choose to use the fourth order splitting scheme to compute the dynamic of the solution. This is done by directly changing the variable `Splitting` in the `Method` structure. We fix the time step $\Delta t = 10^{-3}$ and the stopping time $T = 1$. The geometry is set on a computational domain $\mathcal{O} =]-10, 10[\times]-10, 10[$ and the number of grid points $J = K = 2^8$. This is coded in Table 5.19.

Now, we define the physical problem. In this case, we want to keep the same physical operators as the one used for the computation of the stationary state but we change the parameters of the



(a) Soliton at time $t = 0.25$. (b) Soliton at time $t = 0.5$. (c) Soliton at time $t = 0.75$.

(d) Soliton at time $t = 1$. (e) Soliton at time $t = 1.25$. (f) Soliton at time $t = 1.5$.

Figure 5.3: Evolution of a dark soliton in a Bose-Einstein condensate.

```
Computation = 'Dynamic';
Ncomponents = 1;
Type = 'Splitting';
Deltat = 1e-3;
Stop_time = 1;
Method = Method_Var2d(Computation,Ncomponents, Type, Deltat, Stop_time);
Method.Splitting = 'Fourth';
xmin = -10;
xmax = 10;
ymin = -10;
ymax = 10;
Nx = 2^8+1;
Ny = 2^8+1;
Geometry2D = Geometry2D_Var2d(xmin,xmax, ymin,ymax, Nx, Ny);
```

Table 5.19: Building the `Method` and `Geometry2D` structures for the simulation.

potential. We consider the following Gross-Piteavskii equation

$$\begin{aligned} i\partial_t\psi(t,x,y) &= \frac{1}{2}\Delta\psi(t,x,y) + \left[\frac{1-\alpha}{2}\left(\gamma_x|x|^2+\gamma_y|y|^2\right) + \frac{\kappa}{4}\left(\gamma_x|x|^2+\gamma_y|y|^2\right)^2\right]\psi(t,x,y) \\ &\quad +\beta|\psi(t,x,y)|^2\psi(t,x,y) + i\Omega\left(y\partial_x - x\partial_y\right)\psi(t,x,y), \end{aligned}$$

with the parameters $\alpha = 1.2$, $\kappa = 0.7$, $\gamma_x = \gamma_y = 1$, $\beta = 1000$ and $\Omega = 3.5$. Here, we have changed the parameter κ from 0.3 for the stationary to 0.7 for the dynamical problem. We define the potential operator and add the defaults nonlinear and gardients operators. The resulting code can be seen in Table 5.20.

We finally launch the simulation. We set the defaults `Outputs` that we compute each 100 iterations and store the solution using the `OutputsINI_Var2d` function. Moreover, we build the

```
Delta = 0.5;
Beta = 1000;
Omega = 3.5;
Physics2D = Physics2D_Var2d(Method,Delta,Beta,Omega);
Alpha = 1.2;
Kappa = 0.7;
Gamma_x = 1;
Gamma_y = 1;
Physics2D = Potential_Var2d(Method, Physics2D,...
@(X,Y) quadratic_plus_quartic_potential2d(Gamma_x, Gamma_y,Alpha,Kappa,X,Y));
Physics2D = Nonlinearity_Var2d(Method, Physics2D);
Physics2D = Gradientx_Var2d(Method, Physics2D);
Physics2D = Gradienty_Var2d(Method, Physics2D);
```

Table 5.20: Building and defining the `Physics2D` structure.

default `Print` structure using the `Print_Var2d` function. Then we launch the computation using the `GPELab2d` function. This is done in Table 5.21.

```
Outputs_iterations = 100;
Outputs_save = 1:
Outputs = OutputsINI_Var2d(Method,Outputs_iterations,Outputs_save);
Printing = 1;
Evo = 15;
Draw = 1;
Print = Print_Var2d(Printing,Evo,Draw);
[Phi_2,Outputs]= GPELab2d(Phi_1,Method,Geometry2D,Physics2D,Outputs,[],Print);
```

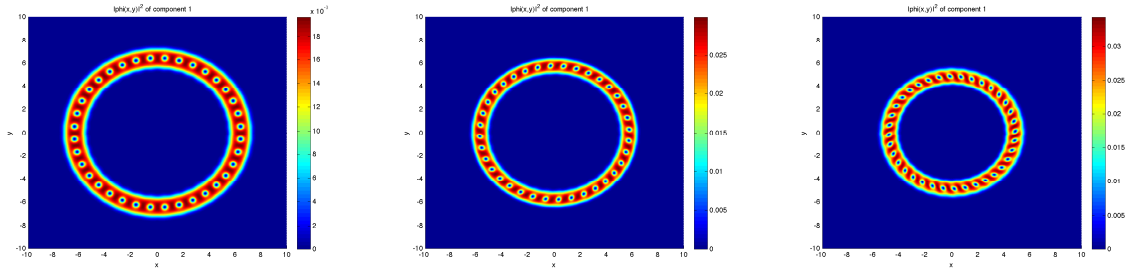
Table 5.21: Launching the simulation.

At the end of the simulation, we obtain the saved solution in the outputs which can be used to show the evolution of the rotating Bose-Einstein condensate. This evolution can be seen on Figure 5.4.

We can also plot the evolution of the `x_rms` and `y_rms`. This is done in Table 5.22. The resulting figures can be seen on Figure 5.5.

```
Time = [0:0.01:1];
plot(Time, Outputs.x_rms{1})
xlabel('Time')
ylabel('Root Mean Square in the x-direction')
```

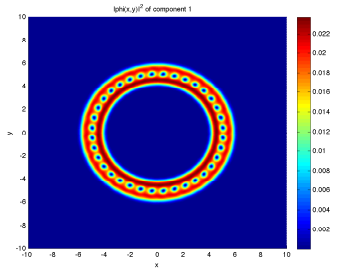
Table 5.22: Plotting the evolution of x-,y-rms.



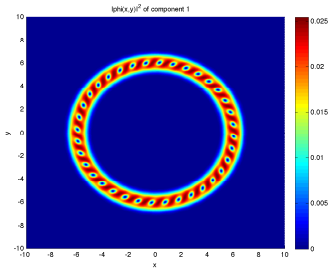
(a) BEC at time $t = 0$.

(b) BEC at time $t = 0.12$.

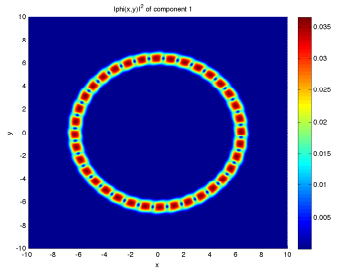
(c) BEC at time $t = 0.23$.



(d) BEC at time $t = 0.34$.

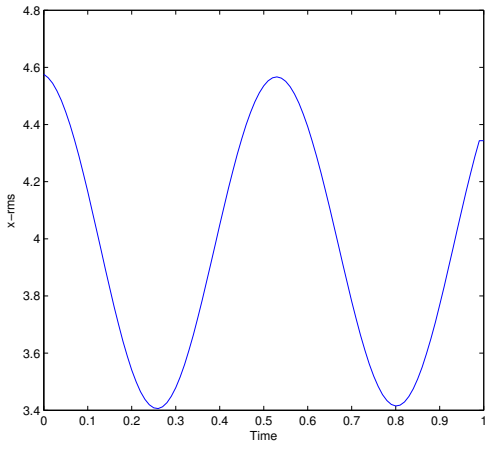


(e) BEC at time $t = 0.45$.

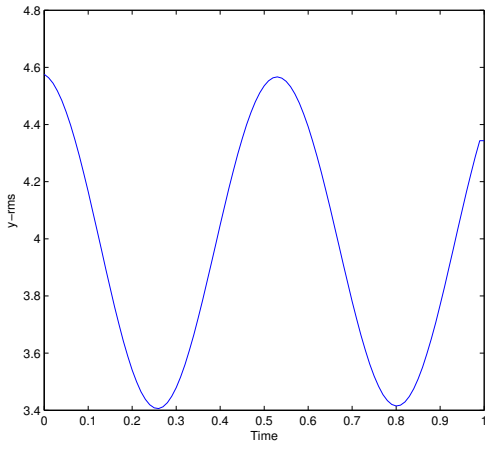


(f) BEC at time $t = 0.56$.

Figure 5.4: Evolution of a fast rotating Bose-Einstein condensate when changing the intensity of the potential.



(a) Evolution of the mean root square in the x-direction.



(b) Evolution of the mean root square in the y-direction.

Figure 5.5: Evolution of the root mean square in the x- and y-direction.

Appendix A

Copyrights & credits

Xavier Antoine
<xavier.antoine at univ-lorraine.fr>

and

Romain Duboscq
<romain.duboscq at univ-lorraine.fr>

Appendix B

License

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Preamble.

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This Licens" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you"."Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and

(2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that

prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work

need not make them do so. A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d. A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is

expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors. All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this

License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM ÒAS ISÓ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see: http://www.gnu.org/licenses/.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read http://www.gnu.org/philosophy/why-not-lgpl.html.

Bibliography

[1] F. Kh. Abdullaev, B.B. Baizakov and V.V. Konotop, *Dynamics of a Bose-Einstein condensate in optical trap*, Nonlinearity and Disorder: Theory and Applications, edited by F. Kh. Abdullaev, O. Bang and M.P. Soerensen, NATO Science Series vol. 45, Kluwer Dodrecht (2001).

[2] F. Kh. Abdullaev, J.C. Bronski and R.M. Galimzyanov, *Dynamics of a trapped 2D Bose-Einstein condensate with periodically and randomly varying atomic scattering length*, Physica D vol. 184, pp. 319-332 (2003).

[3] J. Garnier, F. Kh. Abdullaev and B.B. Baizakov, *Collapse of a Bose-Einstein condensate induced by fluctuations of laser intensity*, Phy. Rev. A vol. 69, 053607, pp. 369-386 (2004).

[4] R. Marty, *On a splitting scheme for the nonlinear Schrödinger equation in a random media*, Commun. Math. Sci. 4 679-705 (2006).

[5] Engel, Klaus-Jochen; Nagel, Rainer, *One-parameter semigroups for linear evolution equations*, Springer (2000).

[6] C. Besse, B. Bidégaray and S. Descombes, *Order estimates in time of splitting methods for the nonlinear schrödinger equation*, SIAM J. Numer. Anal. Vol 40, No. 1, pp. 26-40 (2002).

[7] C. Besse, *A relaxation scheme for the non-linear schrödinger equation*, SIAM J. Numer. Anal. Vol 42, No. 3, pp. 934-952 (2004).

[8] G. Da Prato and J. Zabczyk, *Stochastic equations in infinite dimensions*, Encyclopedia of mathematics and its applications, Cambridge University Press: Cambridge, England (1992).

[9] A. De Bouard and R. Fukuizumi, *Stochastic fluctuations in the Gross-Pitaevskii equation*, Nonlinearity 20, pp. 2821-2844 (2007).

[10] A. De Bouard and A. Debussche, *The stochastic nonlinear Schrödinger equation in H1*, Stochastic Analysis and Appl., vol. 21, p. 97-126 (2003).

[11] A. De Bouard and R. Fukuizumi, *Representation formula for stochastic Schrodinger evolution equations and applications*, preprint.

[12] W. Bao and H. Wang, *An efficient and spectrally accurate numerical method for computing dynamics of rotating Bose-Einstein condensates*, J. of Comp. Physics, Vol 217, pp. 612-626 (2006).

[13] J. Lee and B. Fornberg, *Some unconditionally stable time stepping methods for the 3-D Maxwell's equations*, J. of Comp. and Applied Mathematics, Vol 166, Issue 2, pp. 497-523 (2004).

[14] E. Faou, *Analysis of splitting methods for reaction-diffusion problems using stochastic calculus*, Math. Comp. Vol. 78, pp 1467-1483 (2009).

[15] W. Bao, H. Wang and P. Markowich, *Ground, symmetric and central vortex states in rotating Bose-Einstein condensates*, Comm. Math. Sci. 3 (1), pp.57-88 (2005).

[16] W. Bao and Q. Du, *Computing the ground state solution of Bose-Einstein condensates by a normalized gradient flow*, SIAM J. Sci. Comput. 25 (5), pp.1674-1697 (2004).

[17] W. Bao and Y. Cai, *Ground States of Two-component Bose-Einstein Condensates with an Internal Atomic Josephson Junction*, East Asia Journal on Applied Mathematics, Vol. 1, pp. 49-81, (2010).

[18] P. Pedri and L. Santos, Phys. Rev. Lett. **95**, 200404 (2005).

[19] R. Zeng and Y. Zhang, *Efficiently computing vortex lattices in rapid rotating Bose-Einstein condensates*, Comput. Phys. Comm. 180, pp.854-860 (2009).

[20] W. Bao and H. Wang, *An efficient and spectrally accurate numerical method for computing dynamics of rotating Bose-Einstein condensates*, J. Comp. Phys. 217, pp.612-626 (2006).

[21] W. Bao, I-L. Chern and F.Y. Lim, *Efficient and spectrally accurate numerical methods for computing ground and first excited states in Bose-Einstein condensates*, J. Comp. Phys. 219, pp 836-854 (2006). *An efficient and spectrally accurate numerical method for computing dynamics of rotating Bose-Einstein condensates*, J. Comp. Phys. 217, pp.612-626 (2006).

[22] L. Wen, H. Xiong and B. Wu, *Hidden vortices in a Bose-Einstein condensate in a rotating double-well potential*, Phys. Rev. A, 82, 053627 (2010).

[23] X. Antoine and R. Duboscq, *In preparation*, 2012.

[24] Denschlag, J and Simsarian, JE and Feder, DL and Clark, Charles W and Collins, LA and Cubizolles, J and Deng, L and Hagley, EW and Helmerson, K and Reinhardt, WP and others, *Generating solitons by phase engineering of a Bose-Einstein condensate*, Science 287, pp 97-101 (2000).