Available online at www.sciencedirect.com

ScienceDirect

Computer Physics Communications 00 (2014) 1–37

~~Computer~~
Physics
Communi-
~~cations~~

GPELab, a Matlab Toolbox to solve Gross-Pitaevskii Equations I: computation of stationary solutions

Xavier Antoine[1,b], Romain Duboscq[1,c]

[a]Université de Lorraine, Institut Elie Cartan de Lorraine, UMR 7502, Vandoeuvre-lès-Nancy, F-54506, France
[b]Inria Nancy Grand-Est/IECL - ALICE
[c]Inria Nancy Grand-Est/IECL - CORIDA

Abstract

This paper presents GPELab (Gross-Pitaevskii Equation Laboratory), an advanced easy-to-use and flexible Matlab toolbox for numerically simulating many complex physics situations related to Bose-Einstein condensation. The model equation that GPELab solves is the Gross-Pitaevskii equation. The aim of this first part is to present the physical problems and the robust and accurate numerical schemes that are implemented for computing stationary solutions, to show a few computational examples and to explain how the basic GPELab functions work. Problems that can be solved include: 1d, 2d and 3d situations, general potentials, large classes of local and nonlocal nonlinearities, multi-components problems, fast rotating gazes. The toolbox is developed in such a way that other physics applications that require the numerical solution of general Schrödinger-type equations can be considered. GPELab also presents some functionalities to compute the dynamics of Bose-Einstein condensates and to numerically include some stochastic effects.

Keywords: Bose-Einstein condensates, nonlinear Schrödinger equation, Gross-Pitaevskii equation, stationary states, ground states, imaginary time, conjugate normalized gradient flow, numerical simulation, computational methods, pseudospectral method
PACS: 02.60.-x, 02.70.-c, 31.15.-p, 31.15.xf
2010 MSC: 35Q41, 81Q05, 65M06, 65M12, 65M70, 65Z05

Contents

Email addresses: xavier.antoine@univ-lorraine.fr (Xavier Antoine), romain.duboscq@univ-lorraine.fr (Romain Duboscq)

1. Introduction

`GPELab`[1] (Gross-Pitaevskii Equation Laboratory) is a flexible Matlab toolbox devoted to the *numerical computation of stationary and dynamical solutions of 1d-2d-3d Gross-Pitaevskii Equations (GPEs)* [41, 52, 53, 63, 64] arising in the modeling of Bose-Einstein Condensates (BECs) [10, 30, 34]. In particular, GPELab can handle advanced physical problems that consider general potentials [45, 46, 48, 74], local and nonlocal (dipole-dipole) nonlinearities [38, 39, 40, 49, 62], include rotation terms [4, 55, 56, 57, 66], stochastic effects [1, 2, 3, 36, 37, 70] and/or multi-components problems [6, 47, 50, 59, 61, 69, 75]. One of the main purposes of GPELab is to provide to physicists a generic, friendly and robust numerical modeling tool to compute theoretical data related to BECs through GPEs. The numerical methods implemented in GPELab are based on pseudospectral approximation techniques [14, 18, 76] that lead to highly accurate spatial solutions. In this first paper, we present the GPEs that GPELab can solve, the numerical schemes that are used for computing stationary solutions and the associated GPELab functions. Examples

[1]`http://gpelab.math.cnrs.fr/`

of GPELab scripts are fully developed to show some of its possibilities for physics problems of interest. A second paper [12] will present the numerical schemes that are included in GPELab for solving the deterministic [11] and stochastic dynamics of GPEs [1, 2, 3, 14, 36], the associated GPELab functions and a few numerical examples. Since the GPEs are Schrödinger-type equations, GPELab could also be used to solve other physics problems that need to numerically compute the solution to a Schrödinger equation like for example in nonlinear laser optics [7, 8, 33, 58].

Freely distributed computational physics codes for solving specific nonlinear Schrödinger equations already exist in the literature. For example, in [71], the authors propose a Fortran 90 code that solves the non rotating one-component GPE for a cubic nonlinearity and a quadratic potential by using the imaginary time method. In [60, 73], the authors distribute codes developed around finite difference methods for solving the GPE with a radial or a spherical potential for a single-component, without rotation. Furthermore, improvements concerning the parallelization of the codes by using OpenMP are realized in [73]. Other examples of codes (Fortran or Matlab codes) for numerically solving GPEs can be found in [25, 42, 54]. However, to the best of our knowledge, GPELab seems to be the most generic freely available toolbox that provides robust and efficient numerical methods for solving stationary and dynamical problems for a large class of Gross-Pitaevskii equations.

The paper is structured as follows. In Section 2, we introduce the standard Gross-Pitaevskii equation used to model Bose-Einstein condensates in a rotating frame and we derive the associated dimensionless form of the GPE that is used in GPELab. Section 3 is dedicated to the problem of characterizing the stationary states of the GPE and to present the robust and fast numerical methods to solve these problems. In particular, we recall the standard approximate steady state solutions that can be derived for the GPE since they play the role of initial guess for the iterative algorithms (e.g. imaginary time [5, 9, 14, 19, 24, 26, 27, 32, 35]) used to compute the stationary solutions. Examples of potentials and nonlinearities are also provided. We next describe in Section 4 the computational methods that are used in GPELab. Essentially, the techniques are based on a semi-implicit backward Euler scheme in time [19] and finite difference or (FFT-based) pseudospectral approximation schemes in space [18]. They are combined with robust Krylov subspace iterative solvers [67, 68, 72] accelerated by physics-based preconditioners [15]. We extend the methods in Section 5 to a large class of multi-components GPEs that state the foundations of GPELab. Section 6 gives a simple but complete example of GPELab source code for a two-dimensional BEC with rotation. This allows us to explain the general philosophy of GPELab and to show step-by-step how to build a GPELab script for a model problem. Section 7 describes with more details the different functions that are included in GPELab and how they must be used for stationary state problems. Section 8 provides two additional examples of simulations. Finally, Section 9 concludes.

2. The dimensionless rotating Gross-Pitaevskii equation

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) [63, 65] based on Gross-Pitaevskii Equations (GPEs) [41, 52, 53, 64]. We do not want here to describe the complex physics behind the BECs and GPE (see [63, 65]) but only to state a few well-known facts about GPE and explain how to rewrite the physical form of the GPE as a dimensionless GPE which is the model equation 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.

We consider a system of N atoms in a cold gas that are confined by using a trapping system. Let us furthermore assume that the temperature T is much smaller than the critical temperature T_c

$$T_c \approx \left(\frac{N}{\zeta(3/2)}\right)^{3/2} \frac{h^2}{2\pi m k_B},$$

where h is the Planck constant, m is the atomic mass, k_B is the Boltzmann's constant and ζ is the zeta function. We can describe a BEC under a rotation effect in the z-axis through a macroscopic wave function ψ which depends on the spatial variable $\mathbf{x} := (x, y, z) \in \mathbb{R}^3$ and time $t > 0$. This function has a dynamics which is governed by a specific nonlinear Schrödinger equation, the so-called Gross-Pitaevskii equation, given by

$$i\hbar\frac{\partial\psi}{\partial t} = \left(-\frac{\hbar^2}{2m}\Delta + V(t,\mathbf{x}) + (N-1)U_0|\psi|^2 - \Omega L_z\right)\psi, \tag{2.1}$$

where $\hbar = h/2\pi$ is the reduced Planck constant and Ω is the angular velocity of the condensate along the z-axis. The complex number i is the complex unit: $i^2 = -1$. 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) trap

$$V(\mathbf{x}) = \frac{m}{2}(\omega_x^2 x^2 + \omega_y^2 y^2 + \omega_z^2 z^2), \tag{2.2}$$

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

$$U_0 = \frac{4\pi\hbar^2 a_s}{m}, \tag{2.3}$$

describes the interaction between the 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

$$L_z = xp_y - yp_x = -i\hbar(x\partial_y - y\partial_x). \tag{2.4}$$

This is the z-component of the angular momentum $L = \mathbf{x} \times \mathbf{P}$, where the momentum operator is $\mathbf{P} = -i\hbar\nabla = (p_x, p_y, p_z)^T$. The energy of the functional is defined by [63, 65]

$$E(\psi) = \int_{\mathbb{R}^3} \left[\frac{\hbar^2}{2m}|\nabla\psi|^2 + V|\psi|^2 + \frac{NU_0}{2}|\psi|^4 - \Omega\psi^* L_z\psi \right] d\mathbf{x}. \tag{2.5}$$

The wave function is normalized

$$||\psi||_0^2 := \int_{\mathbb{R}^3} |\psi(t, \mathbf{x})| d\mathbf{x} = 1, \tag{2.6}$$

which corresponds to the mass conservation constraint.

2.2. The dimensionless GPE

Let us introduce the following changes of variables [21, 29, 43]

$$\begin{aligned} &t \to \frac{t}{\omega_m}, \quad \omega_m = \min(\omega_x, \omega_y, \omega_z), \quad \mathbf{x} \to \mathbf{x}a_0, \quad a_0 = \sqrt{\frac{\hbar}{m\omega_m}}, \\ &\psi \to \frac{\psi}{a_0^{3/2}}, \quad \Omega \to \Omega\omega_m, \quad E(\cdot) \to \hbar\omega_m E_{\beta,\Omega}(\cdot). \end{aligned} \tag{2.7}$$

This leads to the following dimensionless GPE

$$i\frac{\partial\psi}{\partial t} = \left(-\frac{1}{2}\Delta + V + \beta|\psi|^2 - \Omega L_z\right)\psi, \tag{2.8}$$

where

$$\beta = \frac{U_0 N}{a_0^3 \hbar\omega_m} = \frac{4\pi a_s N}{a_0}, \tag{2.9}$$

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

$$V(\mathbf{x}) = \frac{1}{2}(\gamma_x^2 x^2 + \gamma_y^2 y^2 + \gamma_z^2 z^2), \tag{2.10}$$

setting $\gamma_{x,y,z} = \omega_{x,y,z}/\omega_m$, where $\omega_m = \min_{x,y,z} \omega_{x,y,z}$. The dimensionless energy functional $E_{\beta,\Omega}$ is defined [20] by

$$E_{\beta,\Omega}(\psi) = \int_{\mathbb{R}^3} \left[\frac{1}{2}|\nabla\psi|^2 + V|\psi|^2 + \frac{\beta}{2}|\psi|^4 - \Omega\psi^* L_z\psi \right] d\mathbf{x}. \tag{2.11}$$

In the special case of a disk-shaped condensation, we consider that $\omega_x \approx \omega_y$ and $\omega_z \gg \omega_x$. This leads to: $\gamma_x = 1$, $\gamma_y \approx 1$, and $\gamma_z \gg 1$, with $\omega = \omega_x$. This means that excitations along the x- and y-axes require less energy than along the z-axis. Therefore, the BEC dynamics is mostly in the x- and y-directions whereas it is stationary in the z-direction. If we consider the following decomposition of the wave function [44, 51]: $\psi(t,\mathbf{x}) = \psi_2(t,x,y)\psi_3(z)$, where

$$\psi_3(z) = \left(\int_{\mathbb{R}^2} |\psi_0(x,y,z)|^2 dxdy\right)^{1/2},$$

with ψ_0 the ground state of the BEC, then, the 3d GPE reduces to a 2d GPE given by

$$i\frac{\partial\psi_2}{\partial t} = \left(-\frac{1}{2}\Delta + V_2 + \sigma + \beta_2|\psi_2|^2 - \Omega L_z\right)\psi_2, \tag{2.12}$$

where

$$\sigma = \frac{1}{2}\int_{\mathbb{R}} \left(\gamma_z^2 z^2|\psi_3(z)|^2 + |\partial_z\psi_3(z)|^2\right)dz, \quad \beta_2 = \beta\int_{\mathbb{R}} |\psi_3(z)|^4 dz, \quad V_2(x,y) = \frac{1}{2}(\gamma_x^2 x^2 + \gamma_y^2 y^2).$$

By using the gauge transformation $\psi_2(t,x,y) \to \psi_2(t,x,y)e^{-\frac{i\sigma}{2}t}$, we can integrate the constant σ and thus eliminate it from Equation (2.12). As a conclusion, one may write the GPE in dimension d as

$$i\frac{\partial\psi}{\partial t} = \left(-\frac{1}{2}\Delta + V_d + \beta_d|\psi|^2 - \Omega L_z\right)\psi, \tag{2.13}$$

for $\mathbf{x} \in \mathbb{R}^d$, $t > 0$, $\beta_3 = \beta$ and $V_3(x,y,z) = V(x,y,z)$. When $\Omega = 0$, we can also reduce the 2d GPE to a 1d GPE by using similar arguments when considering a cigar-shaped condensation [14, 21, 44, 51].

3. Stationary states - initial data - potentials - nonlinearities

Let us assume that we are now in the 2d case. We remark here that 3d (and 1d) problems can also be treated by GPELab but, for conciseness, we restrict ourselves to the 2d case, the extensions to the 1d and 3d situations being quite direct [13].

3.1. Stationary states

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

$$\psi(t,\mathbf{x}) = e^{-i\mu t}\phi(\mathbf{x}), \tag{3.14}$$

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

$$\mu\phi(\mathbf{x}) = -\frac{1}{2}\Delta\phi(\mathbf{x}) + V(\mathbf{x})\phi(\mathbf{x}) + \beta|\phi(\mathbf{x})|^2\phi(\mathbf{x}) - \Omega L_z\phi(\mathbf{x}), \tag{3.15}$$

under the normalization constraint

$$||\phi||_0^2 = \int_{\mathbb{R}^2} |\phi(\mathbf{x})|^2 d\mathbf{x} = 1. \tag{3.16}$$

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

$$\mu_{\beta,\Omega}(\phi) = E_{\beta,\Omega}(\phi) + \frac{\beta}{4}\int_{\mathbb{R}^2} |\phi(\mathbf{x})|^4 d\mathbf{x}, \tag{3.17}$$

with

$$E_{\beta,\Omega}(\phi) = \int_{\mathbb{R}^2}\left[\frac{1}{2}|\nabla\phi|^2 + V|\phi|^2 + \frac{\beta}{2}|\phi|^4 - \Omega\phi^* L_z\phi\right]d\mathbf{x}. \tag{3.18}$$

Furthermore, (3.15) can be seen as the Euler-Lagrange equations associated with the constraint minimization problem (3.16). This also means that the eigenfunctions are the critical points of the energy functional $E_{\beta,\Omega}$ over the unit sphere: $\mathbb{S} := \{||\phi||_0 = 1\}$ [44, 63]. Computing the global minimal solution(s) ϕ_g to the energy functional (3.18) under the normalization constraint

$$\phi_g = \underset{\phi\in\mathbb{S}}{\operatorname{argmin}}\ E_{\beta,\Omega}(\phi) \tag{3.19}$$

provides a ground state solution while local minima are excited (metastable) states.

3.2. *Initial data*

When one wants to compute numerically solutions to the minimization problem (3.18)-(3.19), 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 computes (or tries to compute) a minimal solution through iterations. The initial data is often (but not always) chosen as an Ansatz of the expected solution for a simplified problem.

For a non-rotating BEC, it can be proved [17, 53] that the global minimal solution is unique and gives a ground state $\phi_g \geq 0$ for a positive initial data ϕ_0. Therefore, for a weak nonlinear interaction, one may 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

$$V(\mathbf{x}) = \frac{1}{2}(\gamma_x^2 x^2 + \gamma_y^2 y^2). \tag{3.20}$$

The initial data is then given in 2d by

$$\phi(\mathbf{x}) = \frac{(\gamma_x\gamma_y)^{1/4}}{\sqrt{\pi}} e^{-(\gamma_x x^2+\gamma_y y^2)/2}. \tag{3.21}$$

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) [19, 22, 74]

$$V(\mathbf{x}) = \frac{1}{2}(\gamma_x^2 x^2 + \gamma_y^2 y^2) + w_0 e^{-\|\mathbf{x}-\mathbf{x}_0\|^2/d^2}. \tag{3.22}$$

When a rotation is taken into account, the choice of the initial data is less clear. In [23], Bao *et al.* propose to choose, for $\gamma_x = \gamma_y = 1$,

$$\phi(\mathbf{x}) = \frac{(1-\Omega)\phi_{\text{ho}}(\mathbf{x}) + \Omega\phi_{\text{ho}}^{v}(\mathbf{x})}{\|(1-\Omega)\phi_{\text{ho}}(\mathbf{x}) + \Omega\phi_{\text{ho}}^{v}(\mathbf{x})\|_0}, \tag{3.23}$$

with

$$\phi_{\text{ho}}(\mathbf{x}) = \frac{1}{\sqrt{\pi}} e^{-(\gamma_x x^2+\gamma_y y^2)/2}, \qquad \phi_{\text{ho}}^{v}(\mathbf{x}) = \frac{(\gamma_x x + i\gamma_y y)}{\sqrt{\pi}} e^{-(\gamma_x x^2+\gamma_y y^2)/2}. \tag{3.24}$$

With the above initial data, ground states for rotating gazes can be obtained for $\Omega < \gamma_{xy}$ by using imaginary time methods (see Section 4) (while this is not e.g. the case with (3.21) when the rotation speed Ω is too large).

In the case of a strong linearity, one may also consider the Thomas-Fermi (TF) approximation [19, 21] of the ground state as initial data. For the 2d case and a quadratic potential, the TF approximate function [31, 44] is such that

$$\phi_\beta^{TF}(\mathbf{x}) = \begin{cases} \sqrt{(\mu_\beta^{TF} - V(\mathbf{x}))/\beta_d}, & \text{if } \beta^{TF} > V(\mathbf{x}), \\ 0, & \text{otherwise.} \end{cases} \tag{3.25}$$

The eigenvalue approximation μ_β^{TF} is given by: $\mu_\beta^{TF} = (4\beta\gamma_x\gamma_y/\pi)^{1/2}/2$. More details about these functions as well as `InitialData_Var2d` can be found in Section 7.3.6. Let us remark that this function allows to use one of the above initial data but the user can also consider its own initial data (that then need to be defined).

3.3. *Potentials*

As already said, GPELab provides the possibility of considering many potentials as well as to define its own potentials. This is for example the case of a harmonic trap (3.20) (`quadratic_potential2d` function) with a possible added exponential term like in (3.22) (`quadratic_plus_exp_potential2d`). Other possibilities include

- Quadratic-plus-quartic potential (`quadratic_plus_quartic_potential2d` function) [46]

$$V(\mathbf{x}) = (1-\alpha)\frac{1}{2}(\gamma_x^2 x^2 + \gamma_y^2 y^2) + \frac{\kappa}{4}(\gamma_x^2 x^2 + \gamma_y^2 y^2)^2. \tag{3.26}$$

- Quadratic-plus-sin (optical) potential (`quadratic_plus_sin_potential2d` function) [28]

$$V(\mathbf{x}) = \frac{1}{2}(\gamma_x^2 x^2 + \gamma_y^2 y^2) + \frac{a_1}{2}\sin\left(\frac{\pi x}{d_1}\right)^2 + \frac{a_2}{2}\sin\left(\frac{\pi y}{d_2}\right)^2. \tag{3.27}$$

- Double-well trapping potential (`double_well_trapping_potential2d` function) [74]

$$V(\mathbf{x}) = \frac{1}{2}\left(\gamma_x^2 x^2 + \gamma_y^2 y^2\right) + V_0 e^{-x^2/2d^2}. \tag{3.28}$$

Any new initial data or potential can be added by just following the way the functions are written. This makes GPELab flexible to handle many physics configuration. Functions for the 1d and 3d cases are available as well.

3.4. Nonlinearities

In GPELab, a large class of nonlinearities can be considered in the Gross-Pitaevskii equation. We have already mentioned the standard cubic nonlinearity given by: $f(\psi)\psi = |\psi|^2\psi$, which describes the van der Waals interactions between the atoms in the condensate. This nonlinearity is related to the s-wave scattering length [63, 65] and is an effective local interaction. Other types of interactions can be taken into account in the Gross-Pitaevskii model like for the magnetic dipole-dipole interaction in BECs made of ^{52}Cr atoms [39, 40]. Due to the large magnetic moment of chromium atoms, the magnetic interaction between the atoms in the condensate cannot be neglected. This type of interaction is modeled by a nonlocal nonlinearity [38, 49, 62]. In the 3d case, it is given by

$$f(\psi)\psi = \frac{\mu\mu_0}{4\pi}\int_{\mathbb{R}^3} \frac{1-3\cos(\theta(\mathbf{y}))^2}{|\mathbf{x}-\mathbf{y}|^2}|\psi(t,\mathbf{y})|^2 d\mathbf{y},$$

where $\theta(\mathbf{y})$ is the angle between the direction of polarization and $\mathbf{y}$. The constant μ and μ_0 are the magnetic moment of the atoms and the magnetic permeability of the vacuum, respectively.

4. Conjugate Normalized Gradient Flow (CNGF) formulation and discretization (for the one-component BEC)

One classical solution for computing the solution to (3.18)-(3.19) is through the *projected gradient method* [19] which is also called *imaginary time method* in the Physics community [9]. This is the basic method that is implemented in GPELab for computing stationary solutions to GPEs. All the corresponding 2d GPELab functions are available in the subdirectory `Code2D`. For the 1d and 3d problems, the associated subdirectories are `Code1D` and `Code3D`, respectively.

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 [19]

$$\begin{cases} \partial_t\phi = -\nabla_{\phi^*}E_{\beta,\Omega}(\phi) = \frac{1}{2}\Delta\phi - V\phi - \beta|\phi|^2\phi + \Omega L_z\phi, t_n < t < t_{n+1}, \\ \phi(\mathbf{x}, t_{n+1}) = \phi(\mathbf{x}, t_{n+1}^+) = \dfrac{\phi(\mathbf{x}, t_{n+1}^+)}{\|\phi(\mathbf{x}, t_{n+1}^+)\|_0}, \\ \phi(\mathbf{x}, 0) = \phi_0(\mathbf{x}), \mathbf{x} \in \mathbb{R}^2, \text{ with } \|\phi\|_0 = 1. \end{cases} \tag{4.29}$$

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 [19] that the CNGF is normalization conserving and energy diminishing if $\beta = 0$ and if the potential is positive. When t tends towards infinity, ϕ gives an approximation of the steady state solution which is a critical point of the energy functional when $V \geq 0$. The initial guess ϕ_0 is chosen according to the possible choices provided in Section 3.2. Finally, let us remark that 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 (imaginary time).

4.1. Time and space discretizations: the Backward Euler (BE) scheme

Different schemes can be considered for computing ground states. In [19], the authors show that the Time Splitting sine-Spectral (TSSP) and the Backward Euler (BE) Finite Difference (FD) 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 do not use this scheme for the stationary states (but it is used in GPELab for the dynamics). Here, we rather consider the

BEFD scheme with rotating term (see Section 4.1.1). 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 iterative method (e.g. BiConjugate Gradient Stabilized (BiCGStab) [68, 72]). 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 must then be used. This is the goal of Section 4.1.2 where we present the BESP scheme which is based on BE in time but on a SPectral FFT scheme in space to capture accurately the creation of vortices for fast rotating condensates. BESP is the scheme that you should prefer to use in GPELab when you consider fast rotating gazes. BEFD (Section 4.1.1) and BESP (Section 4.1.2) are included in GPELab for 1d, 2d and 3d (not for BEFD) problems, general potentials and nonlinearities. As we see latter (Section 5), systems of BECs can also be considered by using these methods.

4.1.1. *Backward Euler Finite Difference (BEFD) scheme*

Concerning the time discretization of (4.29), the application of the Backward Euler scheme leads to the semi-discrete semi-implicit (linear) scheme (BE scheme)

$$\begin{cases} \dfrac{\tilde{\phi}-\phi^n}{\delta t} = \dfrac{1}{2}\Delta\tilde{\phi} - V(\mathbf{x})\tilde{\phi} - \beta|\phi^n|^2\tilde{\phi} + \Omega L_z\tilde{\phi}, \quad 1 \le n \le N, \mathbf{x} \in \mathbb{R}^2, \\ \phi^{n+1} = \dfrac{\tilde{\phi}}{\|\tilde{\phi}\|_0}, \quad \mathbf{x} \in \mathbb{R}^2, \end{cases} \tag{4.30}$$

setting $M\delta t = T_{\mathrm{cvg}}$, where T_{cvg} is the maximal time of computation and N is the number of time steps. Let us remark here that T_{cvg} 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. In GPELab, the (strong) stopping criterion that is used is the following

$$\|\phi^{n+1} - \phi^n\|_\infty < \varepsilon\delta t, \tag{4.31}$$

where $\|\cdot\|_\infty$ is the uniform norm. There is also a (weak) stopping criterion associated with the evolution of the energy that is given by

$$|E_{\beta,\Omega}(\phi^{n+1}) - E_{\beta,\Omega}(\phi^n)| < \varepsilon\delta t. \tag{4.32}$$

We need to fix ε small enough to obtain a good accuracy (most particularly when considering highly accurate solutions based on pseudospectral spatial approximation like in Section 4.1.2) in the approximation of the stationary state. Until now, GPELab only includes uniform time stepping in time, for a fixed (user-defined) time step δt.

For the numerical purpose, the scheme (4.30) still requires to be space discretized. To this end, we use a second-order finite difference discretization here. Since the domain is $\mathbb{R}^2$, we have to set suitable boundary conditions on a fictitious boundary to get a finite computational domain. Here, we impose the homogeneous Dirichlet boundary condition: $\tilde{\phi}(\mathbf{x}) = 0$, for $\mathbf{x}$ on the boundary of a large enough computational box: $\mathcal{O} :=]-a_x; a_x[\times]-a_y; a_y[$, assuming that the physics takes place inside $\mathcal{O}$. Let us introduce 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 for uniform discretization steps h_x and h_y in the x- and y-directions, respectively. Therefore, for $1 < j \le J-1$, $h_x = (x_j - x_{j-1}) = 2a_x/J$, and, for $1 < k \le K-1$, $h_y = (y_k - y_{k-1}) = 2a_y/K$, the rotation term L_z is discretized by a two-points second-order centered scheme

$$[L_z]\phi^n_{j,k} := -i(x_j\delta_y\phi^n_{j,k} - y_k\delta_x\phi^n_{j,k}). \tag{4.33}$$

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

$$\delta_x\phi^n_{j,k} = \frac{\phi^n_{j+1,k} - \phi^n_{j-1,k}}{2h_x}, \quad \delta_y\phi^n_{j,k} = \frac{\phi^n_{j,k+1} - \phi^n_{j,k-1}}{2h_y}. \tag{4.34}$$

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

$$\begin{aligned}&\delta_x^2\phi_{j,k}^n = \frac{\phi_{j+1,k}^n - 2\phi_{j,k}^n + \phi_{j-1,k}^n}{h_x^2}, \quad \delta_y^2\phi_{j,k}^n = \frac{\phi_{j,k+1}^n - 2\phi_{j,k}^n + \phi_{j,k-1}^n}{h_y^2},\\ &[\Delta]\phi_{j,k}^n = \delta_x^2\phi_{j,k}^n + \delta_y^2\phi_{j,k}^n, (j,k) \in \mathcal{D}_{J,K}.\end{aligned} \tag{4.35}$$

This provides the matrix $[\Delta] \in \mathcal{M}_{M_{\mathcal{D}}}(\mathbb{C})$ which must be applied to a (global) vector $\boldsymbol{\phi}^n := (\phi_{I(j,k)}^n)_{(j,k)\in\mathcal{D}_{J,K}}$ of size $M_{\mathcal{D}}$, where $M_{\mathcal{D}} = (J-1)(K-1)$ (respectively $M_{\mathcal{D}} = J-1$ and $M_{\mathcal{D}} = (J-1)(K-1)(L-1)$) in 2d (respectively 1d and 3d). Finally, the potential is only considered at the interior discretization points leading to a diagonal matrix $[V] \in \mathcal{M}_{M_{\mathcal{D}}}(\mathbb{C})$, with diagonal elements $[V]_{I(j,k)}$.

The spatial discretization of (4.30) consists in solving the $M_{\mathcal{D}} \times M_{\mathcal{D}}$ linear system with normalization step

$$\begin{cases}[A]\tilde{\boldsymbol{\phi}} = \mathbf{b}^n,\\ \boldsymbol{\phi}^{n+1} = \dfrac{\tilde{\boldsymbol{\phi}}}{\|\tilde{\boldsymbol{\phi}}\|_0},\end{cases} \tag{4.36}$$

with

$$[A] := \frac{1}{\delta t}[I] - \frac{1}{2}[\Delta] + [V] + \beta[|\boldsymbol{\phi}^n|^2] - \Omega[L_z], \quad \mathbf{b}^n := \frac{\boldsymbol{\phi}^n}{\delta t}. \tag{4.37}$$

Hereabove, we set: $[I] \in \mathcal{M}_{M_{\mathcal{D}}}(\mathbb{C})$ as the identity matrix and $[|\boldsymbol{\phi}^n|^2] \in \mathcal{M}_{M_{\mathcal{D}}}(\mathbb{C})$ as the diagonal ("nonlinear potential") matrix with diagonal terms $[|\boldsymbol{\phi}^n|^2]_{I(j,k)} := |\phi^n|_{I(j,k)}^2$. 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

$$\|\boldsymbol{\phi}\|_0 := h_x^{1/2}h_y^{1/2}\Big(\sum_{(j,k)\in\mathcal{D}_{J,K}} |\phi_{j,k}|^2\Big)^{1/2}. \tag{4.38}$$

Furthermore, we define the discrete (strong) stopping criterion as

$$\|\boldsymbol{\phi}^{n+1} - \boldsymbol{\phi}^n\|_\infty < \varepsilon\delta t, \tag{4.39}$$

with the discrete uniform norm defined by: $\forall\boldsymbol{\phi} \in \mathbb{C}^{M_{\mathcal{D}}}$, $\|\boldsymbol{\phi}\|_\infty = \max_{(j,k)\in\mathcal{D}_{J,K}} |\phi_{I(j,k)}|$, and the discrete (weak) stopping criterion as

$$|E_{\beta,\Omega}(\boldsymbol{\phi}^{n+1}) - E_{\beta,\Omega}(\boldsymbol{\phi}^n)| < \varepsilon\delta t, \tag{4.40}$$

with the discrete energy

$$E_{\beta,\Omega}(\boldsymbol{\phi}) = h_x^{1/2}h_y^{1/2}\sum_{(j,k)\in\mathcal{D}_{J,K}} \Re\left\{\phi_{I(j,k)}^*\left(-\frac{1}{2}[\Delta] + [V] + \beta[|\phi_{I(j,k)}|^2] - \Omega[L_z]\right)\phi_{I(j,k)}\right\}.$$

BEFD can be used in GEPLab for 1d and 2d problems but does not exist for 3d problems. The reason is that BESP is preferred in practice for 3d problems because of its robustness and accuracy.

4.1.2. Backward Euler pseudoSPectral (BESP) scheme

Rather than a finite difference scheme, pseudospectral approximation of the spatial derivatives can be used to get high-order accuracy in space [14]. GPELab considers an approach based on Fourier series representations through FFTs. To this end, we impose a periodic boundary condition on the fictitious boundary of a large enough finite computational box: $\mathcal{O} :=] - a_x; a_x[\times] - a_y; a_y[$. We also introduce the set $\mathcal{P}_{J,K}$ of indices of the spatial grid points (x_j, y_k), for $(j,k) \in \mathcal{P}_{J,K}$, as

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

with $J, K \ge 2$, and for uniform discretization steps h_x and h_y in the x- and y-directions, respectively. The partial Fourier pseudospectral discretizations in the x- and y-directions are respectively given by

$$\tilde{\phi}(x_j, y_k, t) = \frac{1}{J}\sum_{p=-J/2}^{J/2-1} \widehat{\tilde{\phi}}_p(y_k, t)e^{i\mu_p(x_j+a_x)}, \quad \tilde{\phi}(x_j, y_k, t) = \frac{1}{K}\sum_{q=-K/2}^{K/2-1} \widehat{\tilde{\phi}}_q(x_j, t)e^{i\lambda_q(y_k+a_y)}, \tag{4.41}$$

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

$$\widehat{\tilde{\phi}}_p(y_k,t) = \sum_{j=0}^{J-1} \tilde{\phi}(x_j,y_k,t)e^{-i\mu_p(x_j+a_x)}, \quad \widehat{\tilde{\phi}}_q(x_j,t) = \sum_{k=0}^{K-1} \tilde{\phi}(x_j,y_k,t)e^{-i\lambda_q(y_k+a_y)}, \tag{4.42}$$

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

$$\begin{cases} \mathbb{A}^{\mathrm{BE},n}\tilde{\boldsymbol{\phi}} = \mathbf{b}^{\mathrm{BE},n}, \\ \boldsymbol{\phi}^{n+1}(\mathbf{x}) = \dfrac{\tilde{\boldsymbol{\phi}}}{\|\tilde{\boldsymbol{\phi}}\|_0}, \end{cases} \tag{4.43}$$

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 $\boldsymbol{\phi}^n = (\phi^n(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{P}_{J,K}} \in \mathcal{M}_{M_{\mathcal{P}}}(\mathbb{C})$. Here, $\mathcal{M}_{M_{\mathcal{P}}}(\mathbb{C})$ designates the set of complex-valued 2d (respectively 1d and 3d) arrays, 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

$$\begin{aligned} \boldsymbol{\Psi} &:= \mathbb{A}^{\mathrm{BE},n}\boldsymbol{\psi} = \mathbb{A}^{\mathrm{BE},n}_{\mathrm{TF}}\boldsymbol{\psi} + \mathbb{A}^{BE}_{\Delta,\Omega}\boldsymbol{\psi}, \\ \mathbb{A}^{\mathrm{BE},n}_{\mathrm{TF}}\boldsymbol{\psi} &:= \left(\frac{[[I]]}{\delta t} + [[V]] + \beta[[|\boldsymbol{\phi}^n|^2]]\right)\boldsymbol{\psi}, \\ \mathbb{A}^{\mathrm{BE}}_{\Delta,\Omega}\boldsymbol{\psi} &:= \left(-\frac{1}{2}[[\Delta]] - \Omega[[L_z]]\right)\boldsymbol{\psi}. \end{aligned} \tag{4.44}$$

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

$$[[I]]_{j,k} := \delta_{j,k}, \qquad [[V]]_{j,k} := V(\mathbf{x}_{j,k}), \qquad [[|\boldsymbol{\psi}^n|^2]]_{j,k} = |\boldsymbol{\psi}^n|^2(\mathbf{x}_{j,k}), \tag{4.45}$$

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 $\mathbb{A}^{\mathrm{BE},n}_{\mathrm{TF}}$ 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. By using (4.41) and (4.42), the partial differential operators in the x- and y-directions are discretized as

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

Therefore, we obtain the following pseudospectral approximation of the operator L_z on the spatial grid

$$\forall (j,k) \in \mathcal{P}_{J,K}, ([[L_z]]\tilde{\boldsymbol{\phi}})_{j,k} = -i\left(x_j([[\partial_y]]\tilde{\boldsymbol{\phi}})_{j,k} - y_k([[\partial_x]]\tilde{\boldsymbol{\phi}})_{j,k}\right). \tag{4.46}$$

Another differentiation leads to the discretization of the second-order differential operators in the x- or y-directions

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

leading to the discrete Laplace operator Δ defined by

$$([[\Delta]]\tilde{\boldsymbol{\phi}})_{j,k} = \left([[\partial_x^2]]\tilde{\boldsymbol{\phi}} + [[\partial_y^2]]\tilde{\boldsymbol{\phi}}\right)_{j,k}. \tag{4.47}$$

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

$$\forall \boldsymbol{\phi} \in \mathbb{C}^{M_{\mathcal{P}}}, \|\boldsymbol{\phi}\|_0 := h_x^{1/2}h_y^{1/2}(\sum_{(j,k)\in\mathcal{P}_{J,K}} |\phi_{j,k}|^2)^{1/2}. \tag{4.48}$$

Morover, we define the discrete (strong) stopping criterion as

$$||\boldsymbol{\phi}^{n+1} - \boldsymbol{\phi}^n||_\infty < \varepsilon\delta t, \tag{4.49}$$

with the discrete uniform norm defined by: $\forall\boldsymbol{\phi} \in \mathbb{C}^{M_\mathcal{P}}$, $||\boldsymbol{\phi}||_\infty = \max_{(j,k)\in\mathcal{P}_{J,K}} |\phi_{j,k}|$, and the discrete (weak) stopping criterion as

$$|E_{\beta,\Omega}(\boldsymbol{\phi}^{n+1}) - E_{\beta,\Omega}(\boldsymbol{\phi}^n)| < \varepsilon\delta t, \tag{4.50}$$

with the discrete energy

$$E_{\beta,\Omega}(\boldsymbol{\phi}) = h_x^{1/2}h_y^{1/2} \sum_{(j,k)\in\mathcal{P}_{J,K}} \Re\left\{\phi^*_{j,k}\left(-\frac{1}{2}[\Delta] + [V] + \beta[|\phi_{j,k}|^2] - \Omega[L_z]\right)\phi_{j,k}\right\}.$$

In practice, the linear system (4.44) is efficiently solved by a Krylov solver (BiCGStab [68, 72]) preconditioned by either the TF operator $\mathbb{A}^{\mathrm{BE},n}_{\mathrm{TF}}$ or a Laplace-type preconditioner (see [15]). BESP is the default method used to solve the CNGF in GPELab. The output physical quantities are the same as those provided by the BEFD scheme.

Finally, another possibility provided in GPELab is semi-implicit Crank-Nicolson schemes [19] for both FD and SP discretizations (CNFD and CNSP schemes). However, it is proved in [19] that CNFD (and CNSP) leads to a CFL-type constraint linking the time and spatial steps. For this reason, we do not develop the method here and refer the interested reader to [13] for further details.

4.2. One- and three-dimensional problems

All 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, 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.

For the three-dimensional case, only the BESP and CNSP methods are coded. In the same spirit as for the one- and two-dimensional functions, the suffix is `3d` and the functions are available in the `Code3D` directory (see the GPELab documentation [13] for further informations).

5. Extension to the multi-components case

5.1. The multi-components GPE

The CNGF extends to the multi-components case [14], i.e. a system of coupled GPEs. 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

$$i\partial_t\Psi(t,\mathbf{x}) = -\frac{1}{2}\Delta\Psi(t,\mathbf{x}) + V(\mathbf{x})\Psi(t,\mathbf{x}) + \sum_{j=1}^{d} G^j(\mathbf{x})\partial_{x_j}\Psi(t,\mathbf{x}) + \beta F(\Psi(t,\mathbf{x}),\mathbf{x})\Psi(t,\mathbf{x}),\ (t,\mathbf{x}) \in \mathbb{R}^+ \times \mathbb{R}^d, \tag{5.51}$$

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

$$\partial_{x_j}\Psi(t,\mathbf{x}) = (\partial_{x_j}\psi_l(t,\mathbf{x}))_{l=1,...,N_c},$$

- and 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 define the energy

$$E(\Psi) := \sum_{j=1}^{N_c} \frac{1}{2}\int_{\mathbb{R}^d} |\nabla\psi_j(t,\mathbf{x})|^2 d\mathbf{x} + \int_{\mathbb{R}^d} \Re\left(\Psi(t,\mathbf{x})^*\left[V(\mathbf{x}) + \sum_{k=1}^{d} G^k\partial_{x_k} + \beta F_{\mathtt{energy}}(\Psi(t,\mathbf{x}),\mathbf{x})\right]\Psi(t,\mathbf{x})\right)d\mathbf{x}, \quad (5.52)$$

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

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

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

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

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

5.2. Stationary states - CNGF

Like the one-component case, we consider the problem of finding stationary states for system (5.51). 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

$$i\mu\Phi(\mathbf{x}) = -\frac{1}{2}\Delta\Phi(\mathbf{x}) + V(\mathbf{x})\Phi(\mathbf{x}) + \sum_{j=1}^{d} G^j(\mathbf{x})\partial_{x_j}\Phi(\mathbf{x}) + \beta F(\Phi(\mathbf{x}),\mathbf{x})\Phi(\mathbf{x}), \quad (5.53)$$

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

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

As for the one-component case, we propose to use the CNGF for the multi-components problem which is a direct extension

$$\begin{cases} \partial_t \Phi = -\nabla_{\Phi^*} E(\Phi) = \frac{1}{2}\Delta\Phi - V(\mathbf{x})\Phi - \sum_{j=1}^{d} G^j(\mathbf{x})\partial_{x_j}\Phi - \beta F(\Phi, \mathbf{x})\Phi, \; t_n < t < t_{n+1}, \\ \Phi(\mathbf{x}, t_{n+1}) = \frac{\Phi(\mathbf{x}, t^+_{n+1})}{||\Phi(\mathbf{x}, t^+_{n+1})||_0}, \\ \Phi(\mathbf{x}, 0) = \Phi_0(\mathbf{x}). \end{cases} \tag{5.54}$$

In the above equations, we set: $\Phi(\mathbf{x}, t^+_{n+1}) = \lim_{t\to t^+_{n+1}} \Phi(\mathbf{x}, t)$. When t tends towards infinity, Φ gives an approximation of the steady state which is solution to (5.53). The ground state is again computed as a solution of the minimization problem of the energy functional E under the normalization constraint $\Phi_g = \underset{||\Phi||_0=1}{\operatorname{argmin}} E(\Phi)$. Let us remark here that (5.51) provides a very general form of coupled nonlinear Schrödinger equations. Therefore, very complex systems of GPEs can be treated but not only. For example, GPELab could be used for nonlinear optics computations involving Schrödinger-type equations [7, 8].

5.3. Time and space discretizations

We essentially focus on schemes based on the Backward Euler time discretization, that is BEFD and BESP. By using the operators 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 (5.54) based on the semi-implicit Backward Euler scheme

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

setting $M\delta t = T_{\text{cvg}}$, where T_{cvg} is the time of computation to get a solution satisfying the convergence criterion and M is the number of related time steps. For the 2d spatial discretization ($d = 2$), we consider the discrete Laplacian and gradients introduced in the one-component case. The functions are evaluated pointwize 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 the notations for the one-component case. 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 set to zero) on $\partial\mathcal{O}$ and we can use discrete Fourier transforms. For BESP, the following approximation holds

$$\begin{cases} \mathbb{A}^{\text{BE},n}\tilde{\mathbf{\Phi}} = \mathbf{b}^{\text{BE},n}, \\ \mathbf{\Phi}^{n+1}(\mathbf{x}) = \frac{\tilde{\mathbf{\Phi}}}{||\tilde{\mathbf{\Phi}}||_0}, \end{cases} \tag{5.55}$$

where $\tilde{\mathbf{\Phi}} = ((\tilde{\phi}_1(\mathbf{x}_{j,k}))_{(j,k)\in\mathfrak{D}_{J,K}}, ..., (\tilde{\phi}_{N_c}(\mathbf{x}_{j,k}))_{(j,k)\in\mathfrak{D}_{J,K}})$ is the discrete unknown array in $\mathbb{C}^{MN_c}$ and the right-hand side is $\mathbf{b}^{\text{BE},n} := \mathbf{\Phi}^n/\delta t$, with $\mathbf{\Phi}^n = ((\Phi^n_1(\mathbf{x}_{j,k}))_{(j,k)\in\mathfrak{D}_{J,K}}, ..., (\Phi^n_{N_c}(\mathbf{x}_{j,k}))_{(j,k)\in\mathfrak{D}_{J,K}}) \in \mathbb{C}^{MN_c}$. The operator $\mathbb{A}^{\text{BE},n} : \mathbb{C}^{MN_c} \to \mathbf{\Psi} \in \mathbb{C}^{MN_c}$ is defined by

$$\begin{aligned} \mathbb{A}^{\text{BE},n}\mathbf{\Phi} &= \mathbb{A}^{\text{BE},n}_{\text{TF}}\mathbf{\Phi} + \mathbb{A}^{\text{BE}}_{\Delta,\Omega}\mathbf{\Phi}, \\ \mathbb{A}^{\text{BE},n}_{\text{TF}}\mathbf{\Phi} &:= \left(\frac{[[I_{N_c}]]}{\delta t} + [[V]] + \beta[[F(\mathbf{\Phi^n})]] \right)\mathbf{\Phi}, \\ \mathbb{A}^{\text{BE}}_{\Delta,\nabla}\mathbf{\Phi} &:= \left(-\frac{1}{2}[[\Delta]] + [[G^1]][[\partial_x]] + [[G^2]][[\partial_y]] \right)\mathbf{\Phi}. \end{aligned} \tag{5.56}$$

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

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

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

$$[[\partial_x]]\mathbf{\Phi} := ([[\partial_x\phi_l]])_{l=1,\ldots,N_c}, \quad [[\partial_y]]\mathbf{\Phi} := \left([[\partial_y\phi_l]]\right)_{l=1,\ldots,N_c}. \tag{5.57}$$

We also define

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

setting $[[G^k_{lm}]] = (G^k_{lm}(\mathbf{x}_{j,k}))_{(j,k)\in\mathcal{D}_{J,K}}$. Finally, the norm $||\cdot||_0$ of a discrete vector $\mathbf{\Phi}$ of N_c components is defined by

$$\forall\mathbf{\Phi} \in \mathbb{C}^{MN_c}, ||\mathbf{\Phi}||_0 := (\sum_{l=1}^{N_c} ||\phi_l||_0^2)^{1/2}, \tag{5.58}$$

where the discrete norm for each component is given by (4.48).

For solving the first equation of (5.55), we again use the preconditioned BiCGStab[68, 72]. The preconditioners are based on the diagonal part of the TF approximation of the multi-components system. We refer to [13] for further details.

6. 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 the user writes a GPELab script to 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 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 to: $\delta t = 0.5$, and the stopping criterion in (4.49) is chosen with $\varepsilon := 10^{-5}$. 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 $N_x = 2^7 + 1$ and $N_y = 2^7 + 1$. We can see in Table 1 the corresponding GPELab source code.

The next step is to define the physical problem. In this example, we compute the ground state of the following GPE

$$i\partial_t\psi(x,y,t) = -\delta\Delta\psi(x,y,t) + \frac{1}{2}(|x|^2 + |y|^2)\psi(x,y,t) + \beta|\psi(x,y,t)|^2\psi(x,y,t) + i\Omega(y\partial_x - x\partial_y)\psi(x,y,t),$$

```
Computation = 'Ground';
Ncomponents = 1;
Type = 'BESP';
Deltat = 5e-1;
Stop_time = [];
Stop_crit = {'MaxNorm',1e-5};
Method = Method_Var2d(Computation,Ncomponents, Type, Deltat, Stop_time , Stop_crit);
xmin = -10;
xmax = 10;
ymin = -10;
ymax = 10;
Nx = 2^7+1;
Ny = 2^7+1;
Geometry2D = Geometry2D_Var2d(xmin,xmax, ymin,ymax, Nx, Ny);
```

Table 1. An example of the `Method` and `Geometry2D` structures in GPELab for computing a ground state.

with $\delta = 0.5, \beta = 500$ and $\Omega = 0.5$. 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 and the nonlinear operators 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: $V(x, y) = \frac{1}{2}(|x|^2 + |y|^2)$,
- 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 example, the operators are predefined in GPELab but for clarity we define them again in our script. To set the physical problem, we first need to build the `Physics2D` structure and fix the values of the parameters δ, β and Ω. The `Physics2D` structure contains all the informations related to the physical problem, that is, among others, the functions related to the operators. Therefore, we have to add the operators by using suitable GPELab functions to the `Physics2D` structure. As already said, the operators are predefined and are set as default arguments in the functions `Dispersion_Var2d`, `Potential_Var2d`, `Nonlinearity_Var2d`, `Gradientx_Var2d` and `Gradienty_Var2d`. The resulting code in available in Table 2.

```
Delta = 0.5;
Beta = 500;
Omega = 0.5;
Physics2D = Physics2D_Var2d(Method,Delta,Beta);
Physics2D = Dispersion_Var2d(Method, Physics2D, @(fftx,ffty) Delta*(fftx.^2+ffty.^2));
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 2. Definition of the physical problem in GPELab through the `Physics2D` structure.

We now provide the initial data for CNGF. 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. A GPELab user can also fix its initial data himself. However, the GPELab function `InitialData_Var2d` is helpful if one wants to use standard initial data like the centered gaussian (3.23) or the Thomas-Fermi approximation (3.25). Here, we consider the centered gaussian as an initial wave function. This is done by setting the `InitialData_choice` variable to `1` (see Table 3).

```
InitialData_choice = 1;
Phi_0 = InitialData_Var2d(Method, Geometry2D, Physics2D,InitialData_choice);
```

Table 3. Initialization by a centered gaussian.

Finally, let us consider now that we want to resolve our problem and 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 quantities like the energy or the mean-square radius are always defined and stored during a calculation. To print these informations when computing, the `Print` structure has to be built to define how to print the outputs. For example, in Table 4, we require that the informations are printed every 15 iterations and drawing the solution is forced.

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

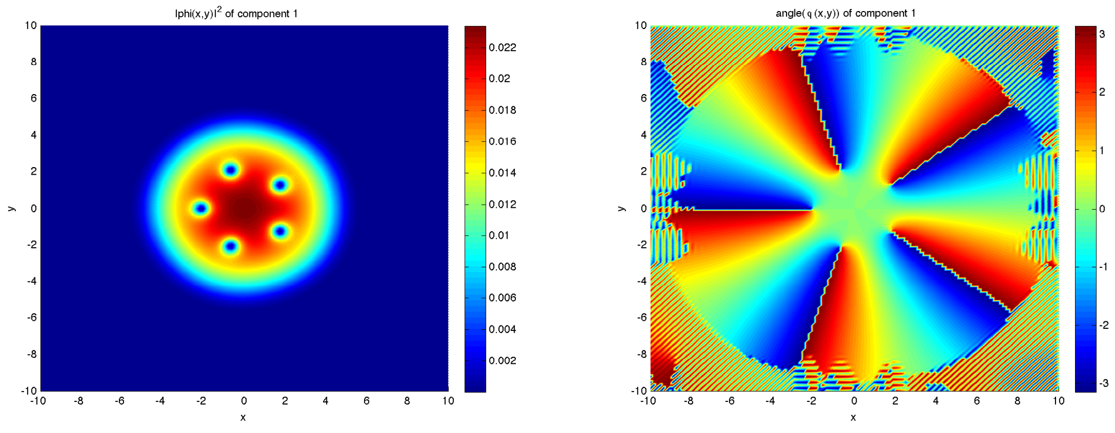
Table 4. Printing/drawing informations during the computations.

To effectively launch the simulation, we need to use the `GPELab2d` function which gathers all the previous structures (and thus the informations about the simulation). The GPELab command is given in Table 5.

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

Table 5. Launching the numerical computation of the solution.

At the end of the simulation, we obtain the modulus and phase of the wave function on the computational domain (see Figures 1(a)-1(b)). This model example shows that using GPELab is quite easy and direct to use.



(a) Modulus of the ground state

(b) Phase of the ground state

Figure 1. Ground state computed with GPELab by using the parameters from Section 6.

7. How to use GPELab

After the presentation of a GPELab model example in Section 6, we describe now more deeply the different functionalities that are defined in the toolbox. First, we begin with some notations and preliminary recalls (subsection 7.1). The main functions that are needed to run a full situation like the previous one are detailed in subsections 7.2 to 7.4. This description follows the way a GPELab script must be built. Section 8 provides some advanced additional examples.

7.1. Notations and preliminary remarks

GPELab uses the following Matlab data type: matrix, cell, function and structure. For more informations, we refer for example to the online Matlab user guide[2]. Let us now introduce some general notations to understand the types of the input and output arguments in the GPELab functions. Let us define

- 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 1, page 15, $\overline{N}_x = 2^8 + 1$ but the number of dof is $N_x = 2^8$ which optimizes FFTs computations.
- N_c is the number of components for the 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

  $$\texttt{f} : \texttt{(x1,x2,...,xN)} \to \texttt{@(x1,x2,...,xN) f(x1,x2,...,xN)}$$

  where `(x1,x2,...,xN)` $\in K_1 \times K_2 \times ... \times K_N$. More generally, we 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 $M \in \mathbb{N}$.
- $C_{N,M}\{K\}$ is a $N \times M$ (Matlab) cell array with values in K, with $N, M \in \mathbb{N}$.

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, this is designated in the sequel by the notation: `xj` $(K_j, \texttt{xj}^{\text{default}})$.

We essentially detail the Matlab functions for the two-dimensional case. Unless precised, 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, it is precised. 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 $K = \mathbb{C}$.

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

7.2. *Setting the numerical scheme and the geometry*

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

7.2.1. *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 6. The `Method_Var2d` function.

The `Method_Var2d` function (see Table 6) 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 [11, 12]), 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` has already some 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 4.1.1), `'CNFD'` to use the Crank-Nicolson Finite Difference scheme, `'BESP'` to use the Backward Euler SPectral discretization scheme (see Section 4.1.2) or `'CNSP'` to use the Crank-Nicolson SPectral discretization scheme.
- `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 dynamical problem.
- `Stop_crit` ($\{\mathbb{S},\mathbb{R}^+\}$,{`'MaxNorm'`,`1e-6`}) is a variable where the real number in the cell corresponds to the (strong) stopping criterion (4.49) when the string in the cell is set to `'MaxNorm'` and the (weak) stopping criterion (4.50) when set to `'Energy'` .
- `Max_iter` ($\mathbb{N}$, `1e6`) is a variable corresponding to the maximum number of iterations for a stationary state computation.
- `Preconditioner` ($\mathbb{S}$,`'FLaplace'`) is a variable that must be either `'None'` for a calculation without preconditioner, `'Laplace'` for the Laplace preconditioner, `'ThomasFermi'` for the Thomas Fermi preconditioner, `'FThomasFermi'` for a multi-components Thomas Fermi preconditioner and `'FLaplace'` for a multi-components Laplace 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.
- `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 residual 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, let us consider that 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 compute some outputs during the simulation. Then, this gives the code in Table 7.

```
Computation = 'Ground';
Ncomponents = 1;
Type = 'BESP';
Deltat = 1e-2;
Stop_time = [];
Stop_crit = {'Energy',1e-10};
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 7. An example of initialization and use of the `Method_Var2d` function.

7.2.2. *The `Geometry2D_Var2d.m` function*

The call to the function `Geometry2D_Var2d.m` is given in Table 8.

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

Table 8. 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 considers a computational box $]-15,15[\times]-15,15[$ with a number of grid points $N_x = N_y = 2^9 + 1$ for a spectral scheme, one builds the `Geometry2D` structure as in Table 9.

```
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 9. An example of the way to use the `Geometry2D_Var2d` function.

7.3. 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

$$i\partial_t \Psi(t,\mathbf{x}) = \mathbf{D}(-i\nabla)\Psi(t,\mathbf{x}) + \mathbf{V}(\mathbf{x})\Psi(t,\mathbf{x}) + \sum_{j=1}^{d} \mathbf{G}^j(\mathbf{x})\partial_{x_j}\Psi(t,\mathbf{x}) + \beta\mathbf{F}(\Psi(t,\mathbf{x}),\mathbf{x})\Psi(t,\mathbf{x}),\ (t,\mathbf{x}) \in \mathbb{R}^+ \times \mathbb{R}^d. \tag{7.59}$$

This system corresponds to the one developed in Section 5, page 11. Here, δ and β are two real-valued constants in $\mathbb{R}$. The energy for each component is

$$\mathbf{E}_j(\Psi) = \int_{\mathbb{R}^d} \Re\left(\psi(t,\mathbf{x})_j^* \left(\left[\mathbf{D}(i\nabla) + \mathbf{V}(\mathbf{x}) + \sum_{k=1}^{d}\mathbf{G}^k(\mathbf{x})\partial_{x_k} + \mathbf{F}_{energy}(\Psi(t,\mathbf{x}),\mathbf{x})\right]\Psi(t,\mathbf{x})\right)_j\right) d\mathbf{x}, \tag{7.60}$$

and the chemical potential is

$$\boldsymbol{\mu}_j(\Psi) = \int_{\mathbb{R}^d} \Re\left(\psi(t,\mathbf{x})_j^* \left(\left[\mathbf{D}(i\nabla) + \mathbf{V}(\mathbf{x}) + \sum_{k=1}^{d}\mathbf{G}^k(\mathbf{x})\partial_{x_k} + \mathbf{F}(\Psi(t,\mathbf{x}),\mathbf{x})\right]\Psi(t,\mathbf{x})\right)_j\right) d\mathbf{x}, \tag{7.61}$$

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

7.3.1. The `Physics2D_Var2d` function

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

Table 10. The `Physics2D_Var2d` function.

The `Physics2D_Var2d` function (Table 10) builds the `Physics2D` *structure* and enables the user to define the basic physical constants δ,β, and the rotation speed Ω (if the gradient operators are set as default (see Section 7.3.5)). 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 in front of the Laplace operator i.e. δ in the Laplace operator $-\delta\Delta$ (see Section 7.3.2) .
- `Beta` ($\mathbb{R}$,0) is a variable corresponding to the constant in front of the nonlinearity (β in equation (7.59)).
- `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 defined, then we have the following rotation operators

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

in the 2d physical problem, and

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

for the 3d case.

We remind that, for 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, a stronger potential must used (a quartic potential $V(\mathbf{x}) \approx |\mathbf{x}|^4$ for example). In Table 11, we show the corresponding GPELab code to 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 11. An example of the way the Physics2D_Var2d function is used.

7.3.2. *The Dispersion_Var2d function*

The Dispersion_Var2d function (see Table 12) defines a dispersion operator (e.g. the Laplace operator $-\delta\Delta$) in the problem by modifying the Physics2D structure.

```
Physics2D = Dispersion_Var2d(Method, Physics2D, Dispersion, G);
```

Table 12. The Dispersion_Var2d function.

It must be provided with the Method and Physics2D structures. The optional arguments are the following

- Dispersion: If a function Dispersion, in $\mathbb{F}(\mathcal{M}_{N_y,N_x}(\mathbb{C})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))$ is provided, the dispersion is defined as follows, for $j,k \in \{1,...,N_c\}$,

$$\mathbf{D}_{j,k}(\xi_x,\xi_y) = \begin{cases} \texttt{Potential}(\xi_x,\xi_y) \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}$$

  where ξ_x and ξ_y are the discrete Fourier frequencies in the x- and y-directions, respectively. If Dispersion is a cell array of functions in $C_{N_c,N_c}\{\mathbb{F}(\mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))\}$, then the dispersion is defined by

$$\mathbf{D}_{j,k}(\xi_x,\xi_y) = \texttt{Dispersion}\{j,k\}(\xi_x,\xi_y),$$

  for $j,k \in \{1,...,N_c\}$. The default argument is the Laplace operator $-\Delta$ which corresponds to

$$\mathbf{D}_{j,k}(\xi_x,\xi_y) = \begin{cases} \delta(\xi_x^2+\xi_y^2) \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}$$

  where δ is parameter defined in Physics2D structure, i.e. Delta ($\in \mathbb{R}$).

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

$$\mathbf{D}_{j,k}(\xi_x,\xi_y) = \mathtt{G}(j,k)\mathtt{Dispersion}\{j,k\}(\xi_x,\xi_y)$$

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

An interesting case of coupling between two Gross-Pitaevskii equations is the spin-orbit coupling (see [6] and Section 8.1). For example, in the case of a system of two GPEs with a quadratic potential, a coupled cubic nonlinearity and a spin-orbit coupling, we obtain the following set of equations

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

where $V(\mathbf{x}) = \frac{1}{2}(x^2+y^2)$ is the quadratic potential, β_{jk} are the interactions constants and κ is the intensity of the spin-orbit coupling. To define the effect of the operator, we have to set the dispersion operator such that

$$\mathbf{D}(\xi_x,\xi_y) = \begin{pmatrix} \frac{1}{2}(\xi_x^2+\xi_y^2) & \kappa(\xi_x - i\xi_y) \\ \kappa(\xi_x + i\xi_y) & \frac{1}{2}(\xi_x^2+\xi_y^2) \end{pmatrix}.$$

In GPELab, we thus have to create a cell array of functions and to use the `Dispersion_Var2d` function like in Table 13 to add the spin-orbit coupling to a system of two Gross-Pitaevskii equations.

```
function Dispersion = Example_Dispersion(Kappa)
Dispsersion = cell(2);
Dispsersion {1,1} = @(fftx,ffty) (1/2)*(fftx^2+ffty.^2);
Dispsersion {1,2} = @(fftx,ffty) Kappa*(fftx-1i*ffty);
Dispsersion {2,1} = @(fftx,ffty) Kappa*(fftx+1i*ffty);
Dispsersion {2,2} = @(fftx,ffty) (1/2)*(fftx^2+ffty.^2);
end
Kappa = 1;
Physics2D = Dispersion_Var2d(Method, Physics2D, Example_Dispersion(Kappa));
```

Table 13. An example of the way the `Dispersion_Var2d.m` function must be used.

7.3.3. *The `Potential_Var2d` function*

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

Table 14. The `Potential_Var2d` function.

The `Potential_Var2d` function (Table 14) 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 follows, for each $j,k \in \{1,...,N_c\}$,

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

If `Potential` is a cell array of functions in

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

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, let us consider that we want to define a quadratic potential for the computation of a ground state for a multi-components BEC with internal atomic Josephson junction [16]. The two-components BECs is modeled by the system

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

where δ is the detuning constant of 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 off-diagonal terms are the effective Rabi frequency λ. To this end, we create a cell array of functions and then we modify the `Physics2D` structure to define the potential operator, resulting in the code given in Table 15.

```
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 15. An example of how to use the `Potential_Var2d` function.

7.3.4. The `Nonlinearity_Var2d` function

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

Table 16. The `Nonlinearity_Var2d` function.

The Nonlinearity_Var2d function (see Table 16) 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 ($C_{N_c,N_c}\{\mathcal{M}_{Ny,Nx}(\mathbb{C})\}$). This function must be provided with the Method and Physics2D structures and has the following optional arguments

- Nonlinearity: If a function Nonlinearity in $\mathbb{F}(C_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\}, \mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))$, is given, the physical nonlinearity is 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 $C_{N_c,N_c}\{\mathbb{F}(C_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\}, \mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))\}$, then the nonlinear operator is defined by

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

  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 definition of the nonlinearity

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

  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 (7.60). Note that it must be the same type of variable as the variable Nonlinearity. If the variable G is defined, it is also multiplied element-by-element by Nonlinearity_energy. If a function Nonlinearity_energy in $\mathbb{F}(C_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\}, \mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))$ 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 $C_{N_c,N_c}\{\mathbb{F}(C_{N_c,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\}, \mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))\}$, then the nonlinear energy operator is

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

  for $j,k \in \{1,...,N\}$. The default argument is Cubic_energy2d and 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 develop the example from Section 7.3.3, page 22, where we fix 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 [16]. In the case of a two-components Gross-Pitaevskii equation with a Josephson junction, we have

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

and $F_{12}(\boldsymbol{\Psi}(t,\mathbf{x}),\mathbf{x}) = F_{21}(\boldsymbol{\Psi}(t,\mathbf{x}),\mathbf{x}) = 0$. This results in the code given in Table 17, 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 17. An example of application of the `Nonlinearity_Var2d` function.

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

Table 18. The `Gradientx_Var2d` function.

7.3.5. *The gradient functions*

The gradient functions 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` (see Table 18), 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 by

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

for each $j,k \in \{1,...,N_c\}$. If `Gradientx` is a cell array of functions in $C_{N_c,N_c}\{\mathbb{F}(\mathcal{M}_{N_y,N_x}(\mathbb{R})^2; \mathcal{M}_{N_y,N_x}(\mathbb{C}))\}$, 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 the rotation 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 $j,k \in \{1,...,N_c\}$.

7.3.6. The *InitialData_Var2d* function

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

Table 19. The `InitialData_Var2d` function.

The `InitialData_Var2d` function (see Table 19) 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 a centered gaussian 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 as initial data, we proceed as in Table 20.

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

Table 20. An example to use the `InitialData_Var2d` function.

7.4. Launching a simulation, setting the outputs and informations related to the computation

7.4.1. The *OutputsINI_Var2d* function

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

Table 21. The `OutputsINI_Var2d` function.

The `OutputsINI_Var2d` function (see Table 21) initializes the outputs of a simulation by building the `Outputs` *structure*. Outputs are scalar values computed by 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 7.4.4, page 30). 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 7.4.2, page 29).
- `save` ($\mathbb{N}$,0) is a variable corresponding to the choice of whether 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 $C_{1,n^{\text{Lout}}}\{\mathbb{F}(\mathcal{M}_{N_y,N_x}(\mathbb{C}),\mathcal{M}_{N_y,N_x}(\mathbb{R})^2,\mathcal{M}_{N_y,N_x}(\mathbb{C})^2;\mathbb{R})\}$ that allows the user to define himself n^{Lout} relevant physical output quantities. These quantities are computed through n^{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 $j\in\{1,...,n^{\text{Lout}}\}$, 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` ($C_{1,n^{\text{Lout}}}\{\mathbb{S}\}$,`'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 $C_{1,n^{\text{Gout}}}\{\mathbb{F}(C_{1,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\},\mathcal{M}_{N_y,N_x}(\mathbb{R})^2,\mathcal{M}_{N_y,N_x}(\mathbb{C})^2;\mathbb{R})\}$ that defines n^{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^{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 $j\in\{1,...,n^{\text{Gout}}\}$, 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` ($C_{1,n^{\text{Gout}}}\{\mathbb{S}\}$,`'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 $t_k := k\ \texttt{Evo\_outputs}\ \Delta t$, $1\leq k\leq N_{\text{out}}$, 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` ($C_{1,N_{\text{out}}}\{C_{1,N_c}\{\mathcal{M}_{N_y,N_x}(\mathbb{C})\}\}$) contains the computed solutions for times t_k if `save` = `1`.
- `phi_abs_0` ($C_{1,N_c}\{\mathcal{M}_{1,N_{\text{out}}}(\mathbb{R})\}$) 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 7.2.2, page 19).

- `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 (see Equation (7.60), page 20)

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

- `Chemical_potential` ($\mathcal{C}_{1,N_c}\{\mathcal{M}_{1,N_{\text{out}}}(\mathbb{R})\}$) is a cell array of vectors such that (see Equation (7.61), page 20)

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

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

For example, to compute the L^2-norm of the gradient of each component of a Bose-Einstein condensate on the computational domain $\mathcal{O}$

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

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 (see Table 22).

```
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 22. An example showing how to use the `OutputsINI_Var2d` function for a user-defined function (single-component BEC).

However, 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, `globaluserdef_outputs` must be used (see Table 23).

```
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 23. An example to use the `OutputsINI_Var2d` function for a user-defined function (multi-components BEC).

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

Table 24. The `Print_Var2d` function.

7.4.2. *The `Print_Var2d` function*

The `Print_Var2d` function builds the `Print` *structure* (Table 24). The aim is to provide to the program the printing informations displayed during the computation. The following optional arguments are

- `Printing` ($\mathbb{N}$,`1`) is a variable equals to 1 for printing informations during the computation and 0 otherwise.
- `Evo` ($\mathbb{N}$,`5`) is a variable corresponding to the number of iterations between each displayed information (including drawing some figures). It must be larger than or equal to `Evo_outputs` from the `OutputsINI_Var2d` function (see Section 7.4.1, page 26).
- `Draw` ($\mathbb{N}$,`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 modulus and phase of the wave function, one defines the `Print` structure by using the `Print_Var2d` function (see Table 25).

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

Table 25. Using the `Print_Var2d` function.

7.4.3. *The `Figure_Var2d` function*

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

Table 26. The `Figure_Var2d` function.

The `Figure_Var2d` function builds the `Figure` *structure* which contains informations needed to draw figures in 2d (see Table 26). 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 27.

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

Table 27. An example for the `Figure_Var2d` function.

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

Table 28. The `GPELab2d` function.

7.4.4. The `GPELab2d` function

The `GPELab2d` function (see Table 28) is the main function to launch a full simulation with respect to a 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 [12]).
- `Outputs` ($\mathcal{S}$) which is a structure containing 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 setting the parameters to draw the figures.

We report in Table 29 a model example to call the `GPELab2d` function.

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

Table 29. An example of call to the `GPELab2d` function.

8. Two additional examples

8.1. Ground state of a system of 2d GPEs modeling a spin-orbit-coupled BEC under rotation

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, coupled cubic nonlinearities and coupled gradients. This system of GPEs models a spin-orbit-coupled BEC under rotation [6]. 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 with $\delta t = 0.5$ and a spatial grid with $2^9 + 1$ points in each direction for a computational domain $]-10, 10[\times]-10, 10[$. We use the strong stopping criterion that we fix to 10^{-5} (see Table 30).

Let us consider the problem of computing the ground state of the following system of Gross-Pitaevskii equations

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

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

```
Computation = 'Ground';
Ncomponents = 2;
Type = 'BESP';
Deltat = 5e-1;
Stop_time = [];
Stop_crit = {'MaxNorm',1e-5};
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 30. Calling the `Method_Var2d` and `Geometry2D_Var2d` functions to build the `Method` and `Geometry2D` structures.

with $\beta_1 = \beta_2 = 1000$, $\beta_{12} = 2000$, $\Omega = 0.1$ and $\kappa = 1.75$. Since the default potential is the quadratic potential and the default gradients define the rotational operator, we only have to provide the coupled cubic nonlinearity and the spin-orbit coupling operators. The coupled cubic nonlinearities are defined by using a script (see Table 31).

```
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 31. Defining the coupled nonlinearity.

We also have to define the energy associated to the coupled cubic nonlinearity (Table 32). Then, we use the `Dispersion_Var2d` function to define the Laplacian and spin-orbit coupling operators (Table 33). This definition of the dispersion operator enables GPELab to construct the Laplace preconditioner with respect to the Laplacian and the spin-orbit coupling operator. 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 34).

```
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 32. Defining the energy associated with the coupled nonlinearity.

We then set the initial data as the Thomas-Fermi approximation (see Table 35).

```
function Dispersion = Dispersion_SpinOrbit(Kappa)
Dispsersion = cell(2);
Dispsersion {1,1} = @(fftx,ffty) (1/2)*(fftx^2+ffty.^2);
Dispsersion {1,2} = @(fftx,ffty) Kappa*(fftx-1i*ffty);
Dispsersion {2,1} = @(fftx,ffty) Kappa*(fftx+1i*ffty);
Dispsersion {2,2} = @(fftx,ffty) (1/2)*(fftx^2+ffty.^2);
end
```

Table 33. An example of the way the `Dispersion_Var2d.m` function has to be used.

```
Delta = 0.5;
Beta = 1000;
Beta_coupled= [1,2;2,1];
Omega = 0.1;
Kappa = 1.75;
Physics2D = Physics2D_Var2d(Method,Delta,[],Omega);
Physics2D = Dispersion_Var2d(Method, Physics2D, Dispersion_SpinOrbit(Kappa));
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 34. Building the `Physics2D` structure.

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

Table 35. Constructing the initial data.

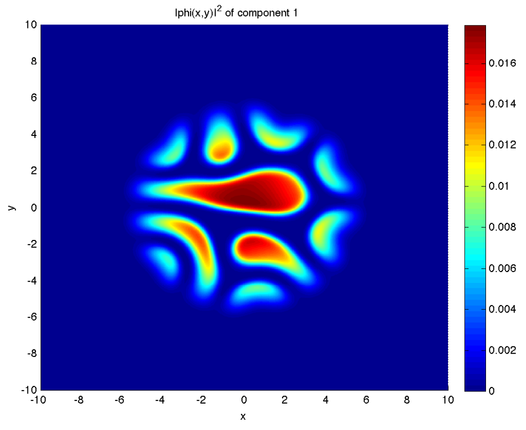
Finally, Table 36 provides the informations for the outputs and printing options. We report on Figures 2(a)-2(d) the moduli and the phases of the stationary state solutions at the end of the computations for the two-components. We observe that the solution has a very complex structure.

```
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);
```

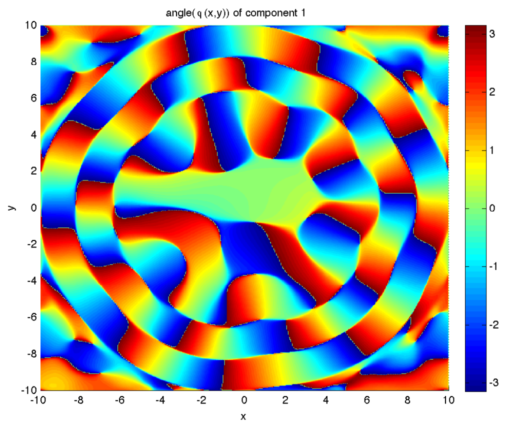
Table 36. Printing options and launching the computation.

8.2. *Ground state of a 3d GPE with a quadratic potential, a cubic nonlinearity and a rotational operator*

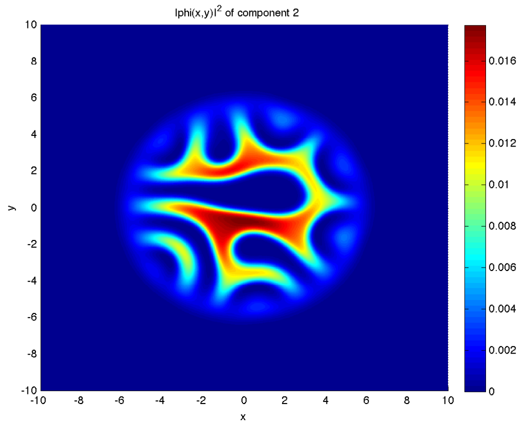
In this Section, we consider the problem of computing the ground state of a GPE with a quadratic potential, a cubic nonlinearity and a rotational operator in 3d. We first build the `Method` and `Geometry3D` structures. We use the BESP scheme with $\delta t = 0.5$, a spatial grid with $2^7 + 1$ points in the x-, y- and z-directions for $\mathcal{O} :=]-10,10[\times]-10,10[\times]-15,15[$. Moreover, we set the strong stopping criterion to 10^{-6}. We can see in Table 37 how to proceed to set these parameters in the `Method` and `Geometry3D` structures by using the `Method_Var3d` and `Geometry3D_Var3d`



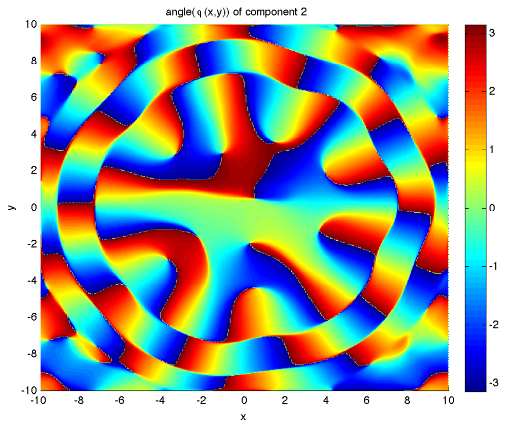
(a) Modulus of component 1 of the ground state



(b) Phase of component 1 of the ground state



(c) Modulus of component 2 of the ground state



(d) Phase of component 2 of the ground state

Figure 2. Ground state obtained at the end of the simulation for the spin-orbit-coupled BEC.

functions. We want to compute the ground state of the following GPE

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

with $\gamma_x = \gamma_y = 1$, $\gamma_z = 1/2$, $\beta = 500$ and $\mathbf{\Omega} = (0,0,0.7)$. We keep in mind that the default dispersion is the Laplacian, the default nonlinearity is the cubic nonlinearity and the default gradients operators are the rotational operators. This implies that we only have to build the `Physics3D` structure with the required coefficients and then to add each operator (see Table 38 for more details). In this case, the quadratic potential has to be defined in the `Potential_Var3d` function. We then fix the initial data by using `InitialData_Var3d`. We choose the Thomas-Fermi approximation (see Table 39). We finally set the outputs and the printing informations, then we launch the simulation following Table 40.

At the end of the computation, we draw on Figure 3(a) the 10^{-3}-isovalues of the modulus of the stationary state solution. In particular we observe the creation of vortex lines inside the BEC. Furthermore, we represent on Figure 3(b) the phase of the ground state in the (x, y)-plane.

```
Computation = 'Ground';
Ncomponents = 1;
Type = 'BESP';
Deltat = 5e-1;
Stop_time = [];
Stop_crit = {'MaxNorm',1e-6};
Method = Method_Var3d(Computation,Ncomponents, Type, Deltat, Stop_time , Stop_crit);
xmin = -10;
xmax = 10;
ymin = -10;
ymax = 10;
zmin = -15;
zmax = 15;
Nx = 2^7+1;
Ny = 2^7+1;
Nz = 2^7+1;
Geometry3D = Geometry3D_Var3d(xmin,xmax, ymin,ymax, zmin,zmax, Nx, Ny, Nz);
```

Table 37. Building the `Method` and `Geometry3D` structures.

```
Delta = 0.5;
Beta = 500;
Omega = [0,0,0.7];
gamma_x = 1;
gamma_y = 1;
gamma_z = 1/2;
Physics3D = Physics3D_Var3d(Method,Delta,Beta,Omega);
Physics3D = Dispersion_Var3d(Method,Physics3D);
Physics3D = Potential_Var3d(Method, Physics3D, @(X,Y,Z) quadratic_potential3d(gamma_x,
gamma_y,gamma_z,X,Y,Z));
Physics3D = Gradientx_Var3d(Method, Physics3D);
Physics3D = Gradienty_Var3d(Method, Physics3D);
Physics3D = Gradientz_Var3d(Method, Physics3D);
Physics3D = Nonlinearity_Var3d(Method, Physics3D);
```

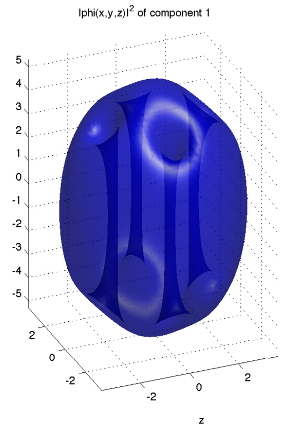
Table 38. Setting the coefficients and adding the default operators to the `Physics3D` structure.

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

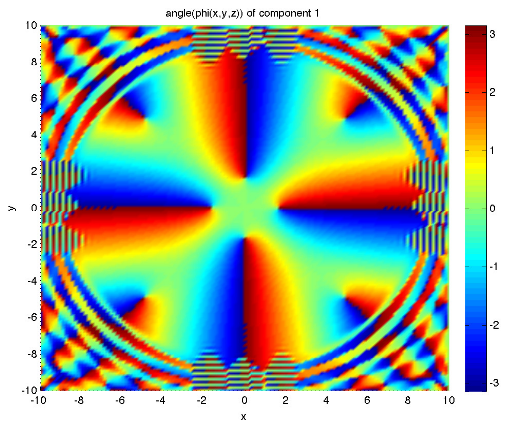
Table 39. Building the initial data as the Thomas-Fermi approximation.

```
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 40. Creating the `Outputs` and `Print` structures and launching the computation.



(a) 10^{-3}-isovalues of the modulus of the ground state



(b) Slice in the (x, y)-plane of the phase of the ground state

Figure 3. Ground state obtained at the end of the simulation.

9. Conclusion

In this paper, we presented GPELab which is a flexible and robust Matlab toolbox for the computation of stationary states (and dynamics) of BECs by using Gross-Pitaevskii equations. This toolbox provides a helpful tool to numerically simulate the solution to various kinds of nontrivial GPEs as shown on a few examples. Furthermore, GPELab integrates some possibilities for computing the dynamics of BECs as well as considering stochastic effects [12, 14].

Acknowledgements. This work was partially supported by the French ANR grant ANR-12-MONU-0007-02 BECASIM (”Modèles Numériques” call).

References

[1] F. Kh. Abdullaev, B. B. Baizakov, and V. V. Konotop. Dynamics of a Bose-Einstein condensate in optical trap. In *Nonlinearity and Disorder: Theory and Applications*, volume 45 of *NATO Science Series*, pages 69–78. Springer Netherlands, 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: Nonlinear Phenomena*, 184(1-4):319 – 332, 2003.
[3] F. Kh. Abdullaev, J. C Bronski, and G. Papanicolaou. Soliton perturbations and the random Kepler problem. *Physica D: Nonlinear Phenomena*, 135(3-4):369 – 386, 2000.
[4] J. R. Abo-Shaeer, C. Raman, J. M. Vogels, and W. Ketterle. Observation of vortex lattices in Bose-Einstein condensates. *Science*, 292(5516):476–479, 2001.
[5] S. K. Adhikari. Numerical solution of the two-dimensional Gross-Pitaevskii equation for trapped interacting atoms. *Physics Letters A*, 265(1):91–96, 2000.
[6] A. Aftalion and P. Mason. Phase diagrams and Thomas-Fermi estimates for spin-orbit-coupled Bose-Einstein condensates under rotation. *Phys. Rev. A*, 88:023610, Aug 2013.
[7] G. Agrawal. *Nonlinear fiber optics*. Springer, 2000.
[8] G. Agrawal. *Applications of Nonlinear Fiber Optics*. Optics and photonics. Elsevier Science, 2001.
[9] P. Amara, D. Hsu, and J. E. Straub. Global energy minimum searches using an approximate solution of the imaginary time Schrödinger equation. *The Journal of Physical Chemistry*, 97(25):6715–6721, 1993.
[10] M. H. Anderson, J. R. Ensher, M. R. Matthews, C. E. Wieman, and E. A. Cornell. Observation of Bose-Einstein condensation in a dilute atomic vapor. *Science*, 269(5221):198–201, 1995.
[11] X. Antoine, W. Bao, and C. Besse. Computational methods for the dynamics of the nonlinear Schrödinger/Gross-Pitaevskii equations. *Computer Physics Communications*, 184(12):2621 – 2633, 2013.
[12] X. Antoine and R. Duboscq. GPELab, a Matlab Toolbox to solve Gross-Pitaevskii Equations II: dynamics and stochastics. *In preparation*.
[13] X. Antoine and R. Duboscq. GPELab User Guide.
[14] X. Antoine and R. Duboscq. *Modeling and computation of Bose-Einstein condensates: stationary states, nucleation, dynamics, stochasticity*. Lecture Notes in Mathematics. Springer, to appear, 2014.
[15] X. Antoine and R. Duboscq. Robust and efficient preconditioned Krylov spectral solvers for computing the ground states of fast rotating and strongly interacting Bose-Einstein condensates. *Journal of Computational Physics*, 258C:509–523, 2014.

[16] W. Bao and Y. Cai. Ground states of two-component Bose-Einstein condensates with an internal atomic Josephson junction. *East Asian J. Appl. Math*, 1:49–81, 2011.

[17] W. Bao and Y. Cai. Mathematical theory and numerical methods for Bose-Einstein condensation. *Kinetic and Related Models*, 6(1):1–135, Mar 2013.

[18] 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. *Journal of Computational Physics*, 219(2):836–854, 2006.

[19] W. Bao and Q. Du. Computing the ground state solution of Bose-Einstein condensates by a normalized gradient flow. *SIAM Journal on Scientific Computing*, 25(5):1674–1697, 2004.

[20] W. Bao, Q. Du, and Y. Zhang. Dynamics of rotating Bose-Einstein condensates and its efficient and accurate numerical computation. *SIAM Journal on Applied Mathematics*, 66(3):758–786, 2006.

[21] W. Bao and W. Tang. Ground-state solution of Bose-Einstein condensate by directly minimizing the energy functional. *Journal of Computational Physics*, 187(1):230–254, 2003.

[22] W. Bao and H. Wang. An efficient and spectrally accurate numerical method for computing dynamics of rotating Bose-Einstein condensates. *Journal of Computational Physics*, 217(2):612–626, 2006.

[23] W. Bao, H. Wang, and P. A. Markowich. Ground, symmetric and central vortex states in rotating Bose-Einstein condensates. *Communications in Mathematical Sciences*, 3(1):57–88, 2005.

[24] D. Baye and J-M. Sparenberg. Resolution of the Gross-Pitaevskii equation with the imaginary-time method on a Lagrange mesh. *Physical Review E*, 82(5):056701, 2010.

[25] M. Caliari and S. Rainer. GSGPEs: A matlab code for computing the ground state of systems of Gross-Pitaevskii equations. *Computer Physics Communications*, 184(3):812 – 823, 2013.

[26] M. M. Cerimele, M. L. Chiofalo, F. Pistella, S. Succi, and M. P. Tosi. Numerical solution of the Gross-Pitaevskii equation using an explicit finite-difference scheme: An application to trapped Bose-Einstein condensates. *Physical Review E*, 62(1):1382, 2000.

[27] M. L. Chiofalo, S. Succi, and M. P. Tosi. Ground state of trapped interacting Bose-Einstein condensates by an explicit imaginary-time algorithm. *Physical Review E*, 62(5):7438, 2000.

[28] D.-I. Choi and Q. Niu. Bose-Einstein condensates in an optical lattice. *Phys. Rev. Lett.*, 82:2022–2025, Mar 1999.

[29] F. Dalfovo and S. Stringari. Bosons in anisotropic traps: Ground state and vortices. *Phys. Rev. A*, 53:2477–2485, Apr 1996.

[30] K. B. Davis, M-O. Mewes, M. R. van Andrews, N. J. Van Druten, D. S. Durfee, D. M. Kurn, and W. Ketterle. Bose-Einstein condensation in a gas of sodium atoms. *Physical Review Letters*, 75(22):3969–3973, 1995.

[31] V. Dunjko, V. Lorent, and M. Olshanii. Bosons in cigar-shaped traps: Thomas-Fermi regime, Tonks-Girardeau regime, and in between. *Phys. Rev. Lett.*, 86:5413–5416, Jun 2001.

[32] M. Edwards and K. Burnett. Numerical solution of the nonlinear Schrödinger equation for small samples of trapped neutral atoms. *Physical Review A*, 51:1382–1386, 1995.

[33] J-P. Fouque. *Wave propagation and time reversal in randomly layered media*, volume 56. Springer, 2007.

[34] D. G. Fried, T. C. Killian, L. Willmann, D. Landhuis, S. C. Moss, D. Kleppner, and T. J. Greytak. Bose-Einstein condensation of atomic hydrogen. *Physical Review Letters*, 81:3811–3814, Nov 1998.

[35] A. Gammal, T. Frederico, and L. Tomio. Improved numerical approach for the time-independent Gross-Pitaevskii nonlinear Schrödinger equation. *Physical Review E*, 60(2):2421, 1999.

[36] J. Garnier, F. Kh. Abdullaev, and B. B. Baizakov. Collapse of a Bose-Einstein condensate induced by fluctuations of the laser intensity. *Physical Review A*, 69:053607, May 2004.

[37] M. E. Gehm, K. M. O'Hara, T. A. Savard, and J. E. Thomas. Dynamics of noise-induced heating in atom traps. *Physical Review A*, 58:3914–3921, Nov 1998.

[38] S. Giovanazzi, A. Görlitz, and T. Pfau. Tuning the dipolar interaction in quantum gases. *Phys. Rev. Lett.*, 89:130401, Sep 2002.

[39] K. Góral, K. Rzążewski, and T. Pfau. Bose-Einstein condensation with magnetic dipole-dipole forces. *Physical Review A*, 61:051601, Mar 2000.

[40] A. Griesmaier, J. Werner, S. Hensler, J. Stuhler, and T. Pfau. Bose-Einstein condensation of chromium. *Physical Review Letters*, 94:160401, Apr 2005.

[41] E. P. Gross. Structure of a quantized vortex in boson systems. *Il Nuovo Cimento Series 10*, 20(3):454–477, 1961.

[42] U. Hohenester. OCTBEC: A Matlab toolbox for optimal quantum control of Bose-Einstein condensates. *Computer Physics Communications*, 2013.

[43] M. J. Holland, D. S. Jin, M. L. Chiofalo, and J. Cooper. Emergence of interaction effects in Bose-Einstein condensation. *Phys. Rev. Lett.*, 78:3801–3805, May 1997.

[44] A. D. Jackson, G. M. Kavoulakis, and C. J. Pethick. Solitary waves in clouds of Bose-Einstein condensed atoms. *Physical Review A*, 58:2417–2422, Sep 1998.

[45] B. Jackson, J. F. McCann, and C. S. Adams. Vortex formation in dilute inhomogeneous Bose-Einstein condensates. *Physical Review Letters*, 80:3903–3906, 1998.

[46] K. Kasamatsu, M. Tsubota, and M. Ueda. Giant hole and circular superflow in a fast rotating Bose-Einstein condensate. *Phys. Rev. A*, 66:053606, Nov 2002.

[47] Y. Kawaguchi and M. Ueda. Spinor Bose-Einstein condensates. *Physics Reports*, 2012.

[48] T. Kuga, Y. Torii, N. Shiokawa, T. Hirano, Y. Shimizu, and H. Sasada. Novel optical trap of atoms with a doughnut beam. *Phys. Rev. Lett.*, 78:4713–4716, Jun 1997.

[49] T. Lahaye, C. Menotti, L. Santos, M. Lewenstein, and T. Pfau. The physics of dipolar bosonic quantum gases. *Reports on Progress in Physics*, 72(12):126401, 2009.

[50] C. K. Law, H. Pu, and N. P. Bigelow. Quantum spins mixing in spinor Bose-Einstein condensates. *Phys. Rev. Lett.*, 81:5257–5261, Dec 1998.

[51] P. Leboeuf and N. Pavloff. Bose-Einstein beams: Coherent propagation through a guide. *Physical Review A*, 64:033602, Aug 2001.

[52] M. Lewin, P. T. Nam, and N. Rougerie. Derivation of Hartree's theory for generic mean-field Bose systems. *arXiv preprint arXiv:1303.0981*,

2013.
[53] E. H. Lieb and R. Seiringer. Derivation of the Gross-Pitaevskii equation for rotating Bose gases. *Communications in Mathematical Physics*, 264(2):505–537, 2006.
[54] E. J. M. Madarassy and V. T. Toth. Numerical simulation code for self-gravitating Bose-Einstein condensates. *Computer Physics Communications*, 184(4):1339 – 1343, 2013.
[55] K. W. Madison, F. Chevy, V. Bretin, and J. Dalibard. Stationary states of a rotating Bose-Einstein condensate: routes to vortex nucleation. *Physical Review Letters*, 86(20):4443–4446, 2001.
[56] K. W. Madison, F. Chevy, W. Wohlleben, and J. Dalibard. Vortex formation in a stirred Bose-Einstein condensate. *Physical Review Letters*, 84(5):806–809, 2000.
[57] K. W. Madison, F. Chevy, W. Wohlleben, and J. Dalibard. Vortices in a stirred Bose-Einstein condensate. *Journal of Modern Optics*, 47(14-15):2715–2723, 2000.
[58] R. Marty. On a splitting scheme for the nonlinear Schroedinger equation in a random medium. *Communications in Mathematical Sciences*, 4(4):363 – 376, 2006.
[59] H.-J. Miesner, D. M. Stamper-Kurn, J. Stenger, S. Inouye, A. P. Chikkatur, and W. Ketterle. Observation of metastable states in spinor Bose-Einstein condensates. *Phys. Rev. Lett.*, 82:2228–2231, Mar 1999.
[60] P. Muruganandam and S. K. Adhikari. Fortran programs for the time-dependent Gross-Pitaevskii equation in a fully anisotropic trap. *Computer Physics Communications*, 180(10):1888 – 1912, 2009.
[61] C. J. Myatt, E. A. Burt, R. W. Ghrist, E. A. Cornell, and C. E. Wieman. Production of two overlapping Bose-Einstein condensates by sympathetic cooling. *Physical Review Letters*, 78:586–589, Jan 1997.
[62] P. Pedri and L. Santos. Two-dimensional bright solitons in dipolar Bose-Einstein condensates. *Physical Review Letters*, 95:200404, Nov 2005.
[63] C. J. Pethick and H. Smith. *Bose-Einstein condensation in dilute gases*. Cambridge University Press, 2002.
[64] L. P. Pitaevskii. Vortex lines in an imperfect bose gas. *Soviet Physics JETP-USSR*, 13(2), 1961.
[65] L. P. Pitaevskii and S. Stringari. *Bose-Einstein condensation*, volume 116. Clarendon press, 2003.
[66] C. Raman, J. R. Abo-Shaeer, J. M. Vogels, K. Xu, and W. Ketterle. Vortex nucleation in a stirred Bose-Einstein condensate. *Physical Review Letters*, 87(21):210402, 2001.
[67] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.
[68] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.
[69] L. E. Sadler, J. M. Higbie, S. R. Leslie, M. Vengalattore, and D. M. Stamper-Kurn. Spontaneous symmetry breaking in a quenched ferromagnetic spinor Bose-Einstein condensate. *Nature*, 443(7109):312–315, 2006.
[70] T. A. Savard, K. M. O'Hara, and J. E. Thomas. Laser-noise-induced heating in far-off resonance optical traps. *Physical Review A*, 56:R1095–R1098, Aug 1997.
[71] R. P. Tiwari and A. Shukla. A basis-set based fortran program to solve the Gross-Pitaevskii equation for dilute bose gases in harmonic and anharmonic traps. *Computer Physics Communications*, 174(12):966 – 982, 2006.
[72] H. A. Van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.
[73] D. Vudragović, I. Vidanović, A. Balaž, P. Muruganandam, and S. K. Adhikari. C programs for solving the time-dependent Gross-Pitaevskii equation in a fully anisotropic trap. *Computer Physics Communications*, 183(9):2021 – 2025, 2012.
[74] L. Wen, H. Xiong, and B. Wu. Hidden vortices in a Bose-Einstein condensate in a rotating double-well potential. *Physical Review A*, 82(5):053627, 2010.
[75] X.-Q. Xu and J. H. Han. Spin-orbit coupled Bose-Einstein condensate under rotation. *Phys. Rev. Lett.*, 107:200401, Nov 2011.
[76] R. Zeng and Y. Zhang. Efficiently computing vortex lattices in rapid rotating Bose-Einstein condensates. *Computer Physics Communications*, 180(6):854–860, 2009.