FLBE
Fast Library for Boltzmann Equation
v0.101

Documentation v1.0

July 1, 2022

Petr F. Kartsev
E-mail: PFKartsev@mephi.ru

Ilya O. Kuznetsov
E-mail: ili-isk@mail.ru

National Research Nuclear University MEPhI
(Moscow Engineering Physics Institute)
115409, Russian Federation, Moscow, Kashirskoe hwy, 31

Contents

Chapter 1

Introduction

Main features of the library. This library allows to calculate collision integrals and relaxation times for wide spectrum of solid state problems in uniform space, i.e. for occupation numbers $n(\mathbf{k}, t)$ without space-coordinate dependency, on discrete momentum lattice $L \times L \times L$ (3D) or $L \times L$ (2D). Due to the application of original universal transformation [1, 2], the required number of operations can be estimated as $\sim L^5 \ln L$ regardless of the interaction type. It allows to study the kinetics of various solid state problems in the macroscopic limit using system size as large as $L \simeq 32$ as it takes less than 1 second per calculation step on modern PC with our library.

The system under consideration can consist of Fermi and Bose particles (electrons, phonons, excitons, photons *etc.*) with various energy spectrum $\varepsilon_{\mathbf{k}}$ and corresponding single-particle Hamiltonian:

$$\hat{H}_0 = \sum_{\alpha,\mathbf{k}} \varepsilon_{\alpha,\mathbf{k}} \hat{n}_{\alpha,\mathbf{k}},$$

where α denotes the sort of particles.

Interaction terms. The library allows arbitrary interaction terms written as a chain of secondary-quantization operators, e.g. electron-phonon interaction:

$$\hat{H}_{e-ph} = \sum_{\mathbf{kq}\sigma} M(\mathbf{q})\, \hat{a}^\dagger_{\mathbf{k}\sigma} \hat{a}_{\mathbf{k}-\mathbf{q},\sigma} \hat{c}_{\mathbf{q}} + H.c., \tag{1.1}$$

or pair interaction:

$$\hat{H}_{pair} = \sum_{\mathbf{kpq}} V(\mathbf{q})\, \hat{b}^\dagger_{\mathbf{k}} \hat{b}^\dagger_{\mathbf{p}} \hat{b}_{\mathbf{p}+\mathbf{q}} \hat{b}_{\mathbf{k}-\mathbf{q}}, \tag{1.2}$$

or some hypothetical three-particle fusion:

$$\hat{H}_{31} = T_0 \sum_{\mathbf{kpq}} \hat{a}^\dagger_{\mathbf{k}+\mathbf{p}+\mathbf{q}} \hat{b}_{\mathbf{k}} \hat{c}_{\mathbf{p}} \hat{d}_{\mathbf{q}} + H.c., \tag{1.3}$$

et cetera, depending on the specific physical problem under study.

Generally, any combination of operators is supported, with two restrictions: (i) we assume the momentum conservation; (ii) the matrix element can eigher depend only on the momentum change, like $M(\mathbf{q})$ in (1.1), $V(\mathbf{q})$ in (1.2), or be constant, like T_0 in (1.3).

More complicated interaction types with matrix elements depending on several momenta, like $T(\mathbf{k}, \mathbf{p}, \mathbf{q})$ in (1.3), are not supported currently in the library due to limitations of the analytic transformation used in the calculation of collision integrals. Nonetheless, even the momentum-independent matrix elements are still useful as they allow to obtain and demonstrate the general behaviour of interacting quantum systems [4, 5], or to formulate a simplified qualitative model [6, 7, 8] and therefore this restriction does not limit the research possibilities.

Collision integrals. The interaction terms generate the collision integrals $J^{(i)}_{\alpha,\mathbf{k}}$ in the Boltzmann equation:

$$\frac{dn_{\alpha,\mathbf{k}}}{dt} = \sum_i J^{(i)}_{\alpha,\mathbf{k}} \tag{1.4}$$

where $n_{\alpha,\mathbf{k}}$ are the time-dependent occupation numbers, and i denotes the corresponding interaction term in the Hamiltonian.

In particular, the collision integral for pair interaction (1.2) in the case of Bose statistics has the form:

$$J_{\mathbf{k}}^{(\mathrm{pair})} = 2\frac{2\pi}{\hbar}\sum_{\mathbf{pq}} |V(\mathbf{q})|^2 \left[(n_{\mathbf{k}}+1)(n_{\mathbf{p}}+1+\delta_{\mathbf{kp}})n_{\mathbf{p+q}}(n_{\mathbf{k-q}}-\delta_{\mathbf{p+q,k-q}}) - \right. \tag{1.5}$$
$$\left. n_{\mathbf{k}}(n_{\mathbf{p}}-\delta_{\mathbf{kp}})(n_{\mathbf{p+q}}+1)(n_{\mathbf{k-q}}+1+\delta_{\mathbf{p+q,k-q}})\right]\delta(\Delta\varepsilon)$$

where Kronecker symbols $\delta_{\mathbf{kp}}$, $\delta_{\mathbf{p+q,k-q}}$ take into account the possibility of coinciding momenta which is important in the case of finite discrete lattice, and $\delta(\Delta\varepsilon)$ corresponds to energy conservation: $\Delta\varepsilon = \varepsilon_{\mathbf{k}} + \varepsilon_{\mathbf{p}} - \varepsilon_{\mathbf{p+q}} - \varepsilon_{\mathbf{k-q}}$.

Natural scale of values. In FLBE, it is implied that all physical parameters are dimensionless. Interaction parameters and energies have the magnitude order around unity. The scale of time is then determined by $2\pi/\hbar \equiv 1$.

Performance. To calculate the collision integral (1.5) with direct summation on discrete momentum lattice $L \times L \times L$, one would require $\sim L^9$ operations, or even $\sim L^{12}$ in the case without momentum conservation, so the application of this primitive approach is limited to very small systems $L \simeq 6 \div 8$.

In FLBE, on the contrary, only $\sim L^5 \ln L$ operations are required for the calculation of collision integral due to accelerated formulas based on original FFT-based transformation described in [1, 2, 3], making possible to study system sizes as large as $L = 32 \ldots 64$. It makes possible to study the problem in continual limit.

Note that the primitive approach with direct summation is also realized in the library, as well as wide set of automatic tests, allowing to check and control the results of different formulas in case of doubt. Such variety can be especially useful in the case of nonstandard interaction not covered by the documentation.

Overall, this library is designed to help study various kinetic phenomena in modern problems of solid state physics. Recently, we used this approach to study the relaxation of nonequilibrium state in superconductor excited by femtosecond laser pulse [9, 10].

Chapter 2

Prerequisites

To use the library, you will need:

- GFortran compiler with OpenMP support
- fftw3 – library for Fast Fourier Transform

Also recommended:

- make

And for better output and debug info:

- text console of operating system or built-in in your preferred IDE (VS Code, mc, Far Manager *etc.*)
- Gnuplot – used by benchmarks to draw the graphs
- LaTeX system – to create PDF files with the equations for debug purposes. However it is not mandatory to use TeX output. You can choose HTML output viewable with a web browser.

Next we give details, depending on the operating system.

2.1 OS Windows

1. **GFortran** compiler in MinGW or MSYS2 builds, preferable 64-bit versions:

   - https://sourceforge.net/projects/mingw-w64/
   - https://www.msys2.org/

   We personally use and recommend build sjlj-v8.1.0 of mingw-w64. Recently it was updated to version 10.0.0 but we did not check it yet.

   (a) Download *.zip with mingw64 build
   (b) Extract to some folder (we use `D:/development/mingw64`)
   (c) Add `D:/development/mingw64/bin` to PATH variable *(Attention: restart existing console to use the updated PATH)*
   (d) Check the compiler availability by typing `gfortran` in command line. Expected response:

   ```
   gfortran.exe: fatal error: no input files
   compilation terminated.
   ```

   It means that the compiler is found and usable.

2. **FFTW**: standard library for Fast Fourier Transform.

   http://www.fftw.org/install/windows.html

   (a) Download *.zip with precompiled FFTW Windows DLLs (as of writing, version 3.3.5).
   (b) Extract the archive to some folder (we use `D:/development/fftw`).
   (c) Add `D:/development/fftw` to PATH variable, for visibility of DLL.

(d) Check the path to the library in Makefile or use the corresponding options in compilation command:

```
-I D:/development/fftw -L D:/development/fftw
```

3. **Make** – recommended for easier compilation:

   http://gnuwin32.sourceforge.net/packages/make.htm

   (a) Download the setup program (as of writing, version 3.81).

   (b) Run the setup and allow to update PATH variable, or do it manually *(Attention: restart existing console to use the updated PATH)*.

   (c) Check the utility availability by typing `make` in command line. Expected response:

   ```
   make: *** No targets specified and no makefile found.  Stop.
   ```

   It means that the utility is found and usable.

   (d) Now you can use the provided Makefile for easier compilation and customization!

   As an alternative to Make, you can use some long command with all the parameters, for example in our demo:

```
gfortran src/demo/minimal.f90 -I src/flbe  -lfftw3-3 -fopenmp  \\
 -I D:/development/fftw -L D:/development/fftw -O3 -J \%TEMP\% \\
 -o minimal.exe
```

4. **LaTeX** and **Gnuplot** are used in the tests and benchmarks to show graphs and equations, you could like to install them too:

   - https://tug.org/texlive/windows.html#install
   - http://gnuplot.info/download.html

5. **Text console** for better output and control: we use and recommend FAR Manager (https://farmanager.com) and/or VSCode (https://code.visualstudio.com), Atom (https://atom.io) or you can use your favorite IDE with text console.

2.2 OS Linux

To work in text console, we use Midnight Commander (`mc`) and/or VSCode.

Use the package manager of your OS to install the corresponding packages: `gfortran`, `fftw3`, `make`; optionally also `mc`, `vscode`, `texlive`, and `gnuplot`. Names of the packages may vary, depending on distribution and release/version.

- Ubuntu 20.04 LTS:

```
sudo apt-get gcc gfortran libgomp libfftw3-3 libfftw3-3-dev make
```

  Optionally:

```
sudo snap install --classic code
sudo snap install --classic atom
sudo apt-get mc gnuplot texlive
```

- Gentoo:

```
## USE keys: +fortran +openmp
emerge gcc fftw make
```

  Optionally:

```
emerge app-misc/mc vscode gnuplot texlive
```

Chapter 3

Library Components

There are several layers in FLBE.

3.1 Help layer

Class `Geometry` allows to create the discrete momentum lattice (Monkhorst-Pack grid) and use arithmetic operations (add, subtract, invert) keeping the result in the first Brillouin zone. Right now, the lattice is supposed to be simple cubic (1D, 2D, or 3D). Usually, lattice size L=32 is large enough to give reasonable macroscopic predictions.

Class `TableOfValues` allows to store the values for this grid: energies $\varepsilon_{\mathbf{k}}$, lifetimes $\tau_{\mathbf{k}}$, interaction amplitudes $V(\mathbf{q})$. More on this later.

Class `Output` is helpful to create debug pages with equations in HTML and TEX formats. It supports multi-line equations with normal and bold symbols, sums with limits, several layers of brackets and single level of upper and lower indices. The resulting files can be viewed with web browser (*.html) or compiled with `pdflatex` (*.tex) to PDF document.

3.2 Physical system

The physical system is described by the momentum lattice used in simulation, model Hamiltonian, and energy broadening factor.

Class `Hamiltonian` contains single-particle and perturbation terms:

$$\hat{H} = \hat{H}_0 + \sum_i \hat{H}_i. \tag{3.1}$$

Class `Efactor` describes the broadening factor $F(\Delta\varepsilon)$ for the energy conservation law used in the collision integrals, e.g. Eq. (1.5):

$$\begin{aligned} J_i &= \sum (\dots)\delta(\Delta\varepsilon) \\ &\to \sum (\dots)F(\Delta\varepsilon). \end{aligned}$$

The energy levels of most physical systems usually have a finite width, especially in condensed matter, due to thermal noise and various broadening mechanisms. Another reason to consider wider levels is the discrete momentum lattice used in our approach ($16 \times 16 \times 16$, $32 \times 32 \times 32$ *etc.*), resulting in sparse irregular spectrum which limits the scattering possibilities. By smoothing the energy spectrum, we can approach the macroscopic system behavior.

- `EfactorExact(`σ`)` returns 1 when $|\Delta\varepsilon| < \sigma$, and 0 otherwise.
- `EfactorLorentz(`σ`)` and `EfactorGauss(`σ`)` providing Lorentz and Gauss broadening factors.

3.2.1 Particles

The system can contain several subsystems of various particles of Bose and Fermi statistics: electrons, holes, phonons, *etc.* Right now, up to 4 subsystems are allowed in the library, but this amount can be easily expanded.

Class name	Dispersion relation	Comments
`DispersionRelationParabolic`	$\varepsilon_{\mathbf{k}} = \varepsilon_1 \mathbf{k}^2 = \varepsilon_1 \sum_{i=1}^{3} k_i^2$	electrons *etc.*
`DispersionRelationLinear`	$\varepsilon_{\mathbf{k}} = \varepsilon_1 \lvert\mathbf{k}\rvert = \varepsilon_1 \sqrt{\sum_{i=1}^{3} k_i^2}$	phonons *etc.*
`DispersionRelationConst`	$\varepsilon_{\mathbf{k}} = const = \varepsilon_1,$	e.g. optical phonons
`DispersionRelationTable`	array of arbitrary values $\{\varepsilon_{\mathbf{k}}\}$	general case

Table 3.1: Versions of `DispersionRelation`

The subsystems are introduced with class `Subsystem` with corresponding `DispersionRelation` ($\varepsilon_{\mathbf{k}}$) describing the following single-particle Hamiltonian

$$\hat{H}_0 = \sum_{\alpha,\mathbf{k}} \varepsilon_{\alpha,\mathbf{k}} \hat{n}_{\alpha,\mathbf{k}}, \tag{3.2}$$

where α denotes the sybsystem number.

Class `DispersionRelation` provides energy levels $\varepsilon_{\mathbf{k}}$ (see Table 3.1). The energy value is obtained using `getEnergy(k, geometry)`. The coefficient ε_1 (default `1.d0`), if needed, is set with `setCoeff(...)`.

Class `Lifetime` is used to add the relaxation terms describing the particles leaving the volume:

$$\frac{dn_{\alpha,\mathbf{k}}}{dt} = \cdots - \frac{n_{\alpha,\mathbf{k}} - n^{(\mathrm{eq})}_{\alpha,\mathbf{k}}}{\tau_{\alpha,\mathbf{k}}}. \tag{3.3}$$

As a result, the particle occupations $n_{\alpha,\mathbf{k}}$ will tend to achieve the equilibrium values $n^{(\mathrm{eq})}_{\alpha,\mathbf{k}}$ with characteristic times $\tau_{\alpha,\mathbf{k}}$.

Class `Occupations` is used to store the current (time-depending) $n_{\alpha,\mathbf{k}}$ or equilibrium $n^{(\mathrm{eq})}_{\alpha,\mathbf{k}}$ occupation numbers of all subsystems using 4D array with indices [$\mathbf{k}$, α].

3.2.2 Problem

Class `Problem` combines Hamiltonian with specific lattice size (`Geometry`) and energy broadening factor `Efactor`.

There are several constructors prepared in advance for some of the most demanded problems:

- `BoseGas` – no interaction, can contain several subsystems (amount is requested by optional parameter `numSubsystems`, default 1) with parabolic dispersion law ($\varepsilon_{\mathbf{k}} = \varepsilon_1 \mathbf{k}^2$):

$$\hat{H} = \hat{H}_0 = \varepsilon_1 \sum_{\alpha,\mathbf{k}} \mathbf{k}^2 \hat{n}_{\alpha,\mathbf{k}},$$

  It can serve as the template for later customization. You can add the desired interaction terms, more subsystems (e.g. phonons or photons), and change the dispersion law by setting the value of ε_1 or replacing the `DispersionRelation` object of the corresponding `Subsystem`.

  It is called from the next constructor:

- `InteractingBoseGas` – single Bose subsystem with contact pair interaction:

$$\hat{H} = \hat{H}_0 + \hat{H}_{\mathrm{pair}} = \varepsilon_1 \sum_{\mathbf{k}} \mathbf{k}^2 \hat{n}_{\mathbf{k}} + V_0 \sum_{\mathbf{kpq}} \hat{b}^\dagger_{\mathbf{k}} \hat{b}^\dagger_{\mathbf{p}} \hat{b}_{\mathbf{p+q}} \hat{b}_{\mathbf{k-q}}. \tag{3.4}$$

- `InteractingFermiGas` – single (spinless) or two (spin-up and spin-down) Fermi subsystems with contact pair interaction:

$$\hat{H} = \hat{H}_0 + \hat{H}_{\text{pair}} = \varepsilon_1 \sum_{\mathbf{k}} \mathbf{k}^2 \hat{n}_{\mathbf{k}} + V_0 \sum_{\mathbf{kpq}} \hat{a}^\dagger_{\mathbf{k}} \hat{a}^\dagger_{\mathbf{p}} \hat{a}_{\mathbf{p+q}} \hat{a}_{\mathbf{k-q}}$$

  or

$$\hat{H} = \hat{H}_0 + \hat{H}_{\text{pair}} = \varepsilon_1 \sum_{\mathbf{k}\sigma} \mathbf{k}^2 \hat{n}_{\mathbf{k}\sigma} + V_0 \sum_{\mathbf{kpq}} \hat{a}^\dagger_{\mathbf{k}\uparrow} \hat{a}^\dagger_{\mathbf{p},\downarrow} \hat{a}_{\mathbf{p+q},\downarrow} \hat{a}_{\mathbf{k-q},\uparrow}.$$

- `ElectronGasWithPhonons` – two subsystems: electrons and phonons. Interactions include electron-electron and electron-phonon terms:

$$\begin{aligned} \hat{H} = \hat{H}_0 + \hat{H}_{\text{pair}} + \hat{H}_{\text{e-ph}} = \\ = \varepsilon_1 \sum_{\mathbf{k}\sigma} \mathbf{k}^2 \hat{n}_{\mathbf{k}\sigma} + v_{\text{sound}} \sum_{\mathbf{k}} |\mathbf{k}| \hat{n}^{(\text{phon})}_{\mathbf{k}} + \\ + V_{\text{pair}} \sum_{\mathbf{kpq}} \hat{a}^\dagger_{\mathbf{k}\uparrow} \hat{a}^\dagger_{\mathbf{p},\downarrow} \hat{a}_{\mathbf{p+q},\downarrow} \hat{a}_{\mathbf{k-q},\uparrow} + M_0 \sum_{\mathbf{kq}\sigma} \left(\hat{a}^\dagger_{\mathbf{k}\sigma} \hat{a}_{\mathbf{k-q},\sigma} \hat{c}_{\mathbf{q}} + H.c. \right), \end{aligned}$$

  where $\hat{a}$, $\hat{a}^\dagger$ are the electron operators, $\hat{c}$, $\hat{c}^\dagger$ are the phonon operators, $\varepsilon_1 \mathbf{k}^2$ is the dispersion law of electrons (parabolic) and $v_{\text{sound}}|\mathbf{k}|$ is the dispersion law of acoustic phonons (linear), $\hat{n}_{\mathbf{k}\sigma} = \hat{a}^\dagger_{\mathbf{k}\sigma} \hat{a}_{\mathbf{k}\sigma}$ and $\hat{n}^{(\text{phon})}_{\mathbf{q}} = \hat{c}^\dagger_{\mathbf{q}} \hat{c}_{\mathbf{q}}$ are the operators for electron and phonon occupation numbers.

The constructors have optional parameters: `chemicalPotential` (or `particleDensity`) and temperature `kT` allow to define equilibrium occupations which are used later in relaxation term (3.3).

After the creation of the `Problem`, the subsystems and interactions can be modified as needed: e.g. use particle mass (`call dispersionLaw % setCoeff(1/(m_eff*2))`), as well as `Efactor`.

The interaction terms are initially added in the form of momentum-independent version with unity amplitude. It can be changed to another value or replaced with `TableOfValues` containing momentum-dependent matrix elements $V(\mathbf{q})$.

Any other less standard `Problem` or `Hamiltonian`, if needed, can be created manually from scratch: `DispersionRelations` → `Subsystems` → `Perturbations` → `Hamiltonian` → `EFactor` → `Problem`.

3.2.3 Hamiltonian

Class `Hamiltonian` contains subsystems and perturbations.

Class `Perturbation` describes the specific term in the Hamiltonian made of several particle operators, for example pair interaction or electron-phonon interaction:

$$\hat{H}_{\text{pair}} = \sum_{\mathbf{kpq}} V(\mathbf{q})\, \hat{a}^\dagger_{\mathbf{k}} \hat{a}^\dagger_{\mathbf{p}} \hat{a}_{\mathbf{p+q}} \hat{a}_{\mathbf{k-q}}, \tag{3.5}$$

$$\hat{H}_{\text{e-ph}} = \sum_{\mathbf{kq}\sigma} M(\mathbf{q})\, \hat{b}^\dagger_{\mathbf{k}\sigma} \hat{b}_{\mathbf{k+q},\sigma} \hat{c}^\dagger_{\mathbf{q}} + H.c. \tag{3.6}$$

The `Perturbation` is created by calling the constructor and then adding creation and annihilation operators of the corresponding subsystems: `addSubsystem(s, numCreation, numAnnihilation)`. For example, the following code

```
dr => DispersionRelationParabolic()
s => BoseSubsystem( dr )
p => Perturbation()
call p % addSubsystem( s, 2, 2)
```

creates the interaction term with two creation operators and two annihilation operators and $V(\mathbf{q}) = V_0 = 1$:

$$\hat{H}_{\text{int}} = \sum_{\mathbf{kpq}} \hat{a}^\dagger_{\mathbf{k}} \hat{a}^\dagger_{\mathbf{p}} \hat{a}_{\mathbf{p+q}} \hat{a}_{\mathbf{k-q}},$$

Similarly, the command

```
call p % addSubsystem( s, 2, 1)
```

creates the interaction term with two creation operators and one annihilation operator and $V(\mathbf{q}) = V_0 = 1$:

$$\hat{H}_{\text{int}} = \sum_{\mathbf{kp}} \hat{a}^\dagger_{\mathbf{k}} \hat{a}^\dagger_{\mathbf{p}} \hat{a}_{\mathbf{k+p}} + H.c. = \sum_{\mathbf{kp}} \left(\hat{a}^\dagger_{\mathbf{k}} \hat{a}^\dagger_{\mathbf{p}} \hat{a}_{\mathbf{k+p}} + \hat{a}^\dagger_{\mathbf{k+p}} \hat{a}^\dagger_{\mathbf{p}} \hat{a}_{\mathbf{k}} \right). \tag{3.7}$$

(note the Hermitian conjugate term added automatically by the library).

The matrix elements can be assumed constant for simplicity then you can set value with method `setAmplitudeConst(...)`, or it can depend on the momentum change $\mathbf{q}$, in this case the array of values $V(\mathbf{q})$ is stored in `TableOfValues` and set with method `setAmplitudeTable(table, operatorsForQ)`).

The parameters `operatorsForQ` describe the positions of operators undergoing the momentum change $\mathbf{q}$, starting from zero. For example, in the term (3.5) the momentum change is given by $\mathbf{q} = \mathbf{k}_3 - \mathbf{k}_0 = \mathbf{k}_1 - \mathbf{k}_2$ and the matrix elements are set with `setAmplitudeTable(table, (/3, 0/), (/1, 2/)))`. In the term (3.6), $\mathbf{q} = \mathbf{k}_1 - \mathbf{k}_0 = \mathbf{k}_2$ and the matrix elements are set with `setAmplitudeTable(table, (/1, 0/), (2, -1))`, where negative index (here -1) means the lack of the term in the expression for $\mathbf{q}$.

The `Perturbations` are added to `Hamiltonian` with `addPerturbation(p)`. The `Subsystems` mentioned in `Perturbation` will be registered in the `Hamiltonian` and assigned the operator characters (i.e. $\hat{a}$, $\hat{b}$, ...) automatically, or if you prefer different numeration order, you can add them explicitly in advance using `addSubsystem(s)`.

After adding all `Perturbations`, the `Hamiltonian` should be switched to work mode using method `prepare()`. After that, any subsequent changes to the `Problem` parameters and structure are not allowed, lest you risk the inner data becoming inconsistent.

To check if the `Hamiltonian` and `Problem` are correct, you can use the method `show(Output o)` and see the formulas in .html and .tex files:

```
include 'flbe.f90'

use flbe

class( Geometry ), pointer :: sizes
class( Problem ), pointer :: prob
class( Hamiltonian ), pointer :: hamilt
class( Output ), pointer :: o

sizes => Geometry( 8, 8, 8 )
prob => InteractingBoseGas( sizes, particleDensity=2.d0, kT=10.d0 )
hamilt => prob % hamilt

o => OutputHTML( 'hamiltonian.html' )
call hamilt % show( o )
deallocate( o )
o => OutputTeX( 'everything.tex' )
call hamilt % show( o )
call prob % show( o ) ! you can output several equations
deallocate( o ) ! the output file is closed in destructor
deallocate( prob )
deallocate( sizes )

end program
```

and then you can view `hamiltonian.html` in your web browser / create `everything.pdf` using

```
pdflatex everything.tex
```

3.3 Collision integrals

The collision integrals are generated from Perturbations in the Problem following the standard rules, with help of class `QbeTerm`.

The occupation numbers change with time following the system of differential equations:

$$\frac{dn_{\mathbf{k},\alpha}}{dt} = \sum_i J^{(i)}_{\mathbf{k},\alpha} = f_{\mathbf{k},\alpha}(\{n\}). \tag{3.8}$$

The class `BoltzmannEquation` calculates the right part $f_{\mathbf{k},\alpha}(\{n\})$ combining all required `QbeTerms` for given Problem:

```
be => BoltzmannEquation( prob, de, version )
...
call be % getRightPart( time, n, dn_dt )
```

Optional parameter `de` (default 1.0) allows to set the energy discretization for FFT-based calculator. Optional parameter `version` (default is VERSION_FAST) allows to choose the version of `QbeTermCalculator` which can be useful to control the analytic transformation in complicated cases (see usage in Ch. 4 Tests). Recommended versions are `VERSION_AS_IS_v1` (useful to output the equations to .html or .tex format), and `VERSION_FAST` (fast FFT version described in [1, 3]), while `VERSION_AS_IS_v2` and `VERSION_AS_IS_v3` are essentially just internal classes which provide required data to the fast version. There is also a hard-coded version (`v0`) for some of the standard `Problems` used for testing (see Table 4.2).

3.4 Solver

Finally, all the aforementioned classes allow us to simulate the kinetics in the system under study.

Class `FiniteDifferenceSolver` is designed to solve differential equations (3.8). There are several descendant classes realizing standard explicit methods:

- 1st order Euler: `FD_Euler`,
- 2nd order Runge-Kutta: `FD_RungeKutta2`,
- 4nd order Runge-Kutta: `FD_RungeKutta4`,
- 5th order Dormand-Prince: `FD_DormandPrince`.

The work variables are instances of class `Occupations`.

The right part for the solver is calculated with general class `FD_RightPart`. Aside from main descendant class `BoltzmannEquation`, there is also class `FD_RightPart_Test` used for testing purposes.

There are several more internal classes: `MultiDimensionalCycle`, `QbeTermCalculatorSimple`, and `QbeTermCalculatorFFT`. They are not supposed to be created outside of main workflow of the library, and only listed here for information.

Chapter 4

Tests

The library is backed with automated tests allowing to quickly check the validity of main functions during the development.

The code resides in the folder /tests/.

To build the test suite, type:

```
make tests.exe
```

There are four levels of tests.

4.1 Help layer

Class `Geometry` is tested to work with discrete momentum lattice (Monkhorst-Pack grid): two- or three-dimensional arrays of integers in the range $[0 \dots L-1]$. Addition and subtraction of two vectors are checked, as well as conversion to and from arrays in the symmetric range $[-L/2 \dots L/2-1]$ useful for data exchange and analysis.

Class `DispersionRelation` is checked to return valid values for three main versions in Section 3.2.1 with 2D and 3D geometries and random values of energy coefficient ε_1.

Class `Occupations` is checked to work with two types of constructors where the dimensions are (i) given explicitly, or (ii) extracted from another instance of `Occupations`, which is useful in `FDSolver`. In addition, the function `compareTo` is checked to calculate the correct difference between two instances of `Occupations` which is used in later tests.

4.2 Solvers' layer

The correctness of `FiniteDifferenceSolver` is checked for several problems with known results:

Equation	Initial condition (R=random value)	Solution
$dn/dt = 0$	$n(0) = R$	$n(t) = R$
$dn/dt = n$	$n(0) = R$	$n(t) = R\exp(t)$
$dn/dt = \cos(t)$	$n(0) = R$	$n(t) = R + \sin(t)$

The versions of `FiniteDifferenceSolver` realized in the library (specifically, `FD_Euler`, `FD_RK2`, `FD_RK4`, `FD_DormandPrince`) are used to solve the equation with time step $\Delta t = 10^{-4}$ and 1000 time steps, then the results are compared. As expected, the precision is good enough (the difference is 10^{-5} or lower, depending on the method).

4.3 Collision integrals

The algorithm used to calculate the collision integrals generated by various types of `Perturbation` is checked in several ways.

First, we check classes `QbeTermAsIs`, `QbeTermGeneralSum`, `QbeTermSumOfABCD` (in short `QbeTerm*`) which allow to calculate the terms corresponding to given `Perturbation`. For example, in the case of pair interaction between Bose particles $\hat{a}_1^\dagger \hat{a}_2^\dagger \hat{a}_3 \hat{a}_4$ the term is given by

$$[n_1(n_2 - \delta_{k_1,k_2})(n_3+1)(n_4+1+\delta_{k_1,k_2}) - (n_1+1)(n_2+1+\delta_{k_1,k_2})n_3(n_4 - +\delta_{k_3,k_4})]\,. \tag{4.1}$$

Note that here only the combination of occupation numbers is calculated, ignoring the momentum conservation and energy-conserving factor $\delta(\Delta\varepsilon)$. The momentum values $\mathbf{k}_i$, however, affect the formula due to possible case of coniciding states. Indeed, in Eq. (1.5) the operator $\hat{a}_1\hat{a}_2$ gives the factor $n_1(n_1-1)$ if the momenta k_1 and k_2 are equal, or $n_1 n_2$, otherwise. Both versions can be written universally as $n_1(n_2-\delta_{k_1,k_2})$.

The values of occupation numbers $\{n_i\}$ and corresponding momenta $\{\mathbf{k}_i\}$ are provided by random number generator.

The classes `QbeTerm*` employ different approaches: `QbeTermAsIs` calculates the value following Eq. (4.1) to the letter, while `QbeTermSumOfABCD` expands the brackets to arrive to the 'ABCD'-decomposition (see [3]) which is later used to build the FFT-based optimized calculator.

In this test, we compare the values given by all three versions to those calculated with explicitly written correct hard-coded relations.

The list of `Perturbations` used in the test are listed in Table 4.1.

Version	Interaction	Comments
`'V_ee'`	$\hat{a}^\dagger_{\mathbf{k}_1\uparrow}\hat{a}^\dagger_{\mathbf{k}_2\downarrow}\hat{a}_{\mathbf{k}_3\downarrow}\hat{a}_{\mathbf{k}_4\uparrow}$	Fermi statistics
`'V_bb'`	$\hat{b}^\dagger_{\mathbf{k}_1}\hat{b}^\dagger_{\mathbf{k}_2}\hat{b}_{\mathbf{k}_3}\hat{b}_{\mathbf{k}_4}$	Bose statistics
`'V_20'`	$\hat{b}^\dagger_{\mathbf{k}_1}\hat{b}^\dagger_{\mathbf{k}_2} + H.c.$	Bose statistics
`'V_21'`	$\hat{b}^\dagger_{\mathbf{k}_1}\hat{b}^\dagger_{\mathbf{k}_2}\hat{b}_{\mathbf{k}_3} + H.c.$	Bose statistics
`'V_12'`	$\hat{b}^\dagger_{\mathbf{k}_1}\hat{b}_{\mathbf{k}_2}\hat{b}_{\mathbf{k}_3} + H.c.$	Bose statistics
`'V_31'`	$\hat{b}^\dagger_{\mathbf{k}_1}\hat{b}^\dagger_{\mathbf{k}_2}\hat{b}^\dagger_{\mathbf{k}_3}\hat{b}_{\mathbf{k}_4} + H.c.$	Bose statistics

Table 4.1: List of `Perturbations` used to check `QbeTerm*`

Second, we check the complete collision integrals calculated by summation of all corresponding terms.

By default, the library uses the FFT-based general approach described in [1, 2, 3]. The relations are generated automatically depending on details of `Perturbation`, such as particle statistics (Bose or Fermi), count of operators, count of subsystems, *etc.*

In this test, we check all branches of the algorithm using several selected `Perturbations` shown in Table 4.2, for which easy-to-read explicit Fortran code ('version 0') was written allowing to check the values returned by the library. If the difference is too large, the value is also calculated using direct summation of the terms given by `QbeTerm*`, which allows to trace the cause of divergence.

In case if the interaction term allows to introduce the momentum transfer (e.g. $\mathbf{q}=\mathbf{k}_1-\mathbf{k}_2$ for 'V_bb'), the test is performed for both momentum-intependent V_0 and momentum-dependent matrix elements $V(\mathbf{q})$.

The values of occupation numbers $\{n_\mathbf{k}\}$ and matrix elements $V(\mathbf{q})$ are provided by random number generator.

Next, we check the calculation of relaxation times describing the behaviour of occupation numbers near the equilibrium values due to linearization of kinetic equations:

$$\frac{dn_{\alpha,\mathbf{k}}}{dt} \simeq -\frac{n_{\alpha,\mathbf{k}} - n^{(\mathrm{eq})}_{\alpha,\mathbf{k}}}{\tau_{\alpha,\mathbf{k}}}. \tag{4.2}$$

The analytical approach used in the library allows to obtain the relaxation times in similar way to collision integrals (see [3]) after applying the related linearization.

The RTA coefficients $-1/\tau_{\alpha,\mathbf{k}}$ are calculated with `QbeTermCalculator % addValue(t, n, dn, rta)` using optional parameter `rta=.TRUE.`. The occupation numbers are provided by thermal equilibrium functions — Fermi-Dirac and Bose-Einstein, depending on the subsystem statistics.

In this test, we use the same set of `Perturbations` from Table 4.2, and again check the values using hard-coded 'Version 0' explicitly created for several selected Problems.

Version-0 function	Interaction term	Comments
`'V0_f_c2a2'`	$\hat{a}^\dagger_{\mathbf{k}_1\uparrow}\hat{a}^\dagger_{\mathbf{k}_2\downarrow}\hat{a}_{\mathbf{k}_3\downarrow}\hat{a}_{\mathbf{k}_4\uparrow}$	Fermi statistics, $\mathbf{k}_1+\mathbf{k}_2=\mathbf{k}_3+\mathbf{k}_4$
`'V0_b_c2a2'`	$\hat{b}^\dagger_{\mathbf{k}_1}\hat{b}^\dagger_{\mathbf{k}_2}\hat{b}_{\mathbf{k}_3}\hat{b}_{\mathbf{k}_4}$	Bose statistics, $\mathbf{k}_1+\mathbf{k}_2=\mathbf{k}_3+\mathbf{k}_4$
`'V0_bb2_c1a1c1a1'`	$\hat{b}^\dagger_{\mathbf{k}_1}\hat{b}_{\mathbf{k}_2}\hat{c}^\dagger_{\mathbf{p}_1}\hat{c}_{\mathbf{p}_2}$	Bose statistics, two subsystems, $\mathbf{k}_1+\mathbf{p}_1=\mathbf{k}_2+\mathbf{p}_2$
`'V0_b_c2a1'`	$\hat{b}^\dagger_{\mathbf{k}_1}\hat{b}^\dagger_{\mathbf{k}_2}\hat{b}_{\mathbf{k}_3}+H.c.$	Bose statistics, $\mathbf{k}_1+\mathbf{k}_2=\mathbf{k}_3$
`'V0_b_c3a1'`	$\hat{b}^\dagger_{\mathbf{k}_1}\hat{b}^\dagger_{\mathbf{k}_2}\hat{b}^\dagger_{\mathbf{k}_3}\hat{b}_{\mathbf{k}_4}+H.c.$	Bose statistics, $\mathbf{k}_1+\mathbf{k}_2+\mathbf{k}_3=\mathbf{k}_4$
`'V0_bb2_c2a1c2a1'`	$\hat{b}^\dagger_{\mathbf{k}_1}\hat{b}^\dagger_{\mathbf{k}_2}\hat{b}_{\mathbf{k}_3}\hat{c}^\dagger_{\mathbf{p}_1}\hat{c}^\dagger_{\mathbf{p}_2}\hat{c}_{\mathbf{p}_3}+H.c.$	Bose statistics, two subsystems, $\mathbf{k}_1+\mathbf{k}_2+\mathbf{p}_1+\mathbf{p}_2=\mathbf{k}_3+\mathbf{p}_3$

Table 4.2: List of `Perturbations` used to check the values of collision integrals and RTA coefficients with 'Version 0' code

Finally, we directly check the relaxation times calculated by the library, using numerical differentiation of collision integrals with small enough increment of occupation numbers $\Delta n_{\mathbf{k}} \simeq 10^{-5}$:

$$\frac{dn^{(\mathrm{eq})}_{\alpha,\mathbf{k}}}{dt} = J_{\alpha,\mathbf{k}}(\{n^{(\mathrm{eq})}\}) = 0, \tag{4.3}$$

$$\frac{d(n^{(\mathrm{eq})}_{\alpha,\mathbf{k}} + \delta n_{\alpha,\mathbf{k}})}{dt} = J_{\alpha,\mathbf{k}}(\{n\}) \simeq -\frac{\delta n_{\alpha,\mathbf{k}}}{\tau_{\alpha,\mathbf{k}}}, \tag{4.4}$$

$$\tau_{\alpha,\mathbf{k}} \simeq \frac{\delta n_{\alpha,\mathbf{k}}}{J_{\alpha,\mathbf{k}}(\{n\})}. \tag{4.5}$$

The values of $\tau_{\alpha,\mathbf{k}}$ are calculated for each momentum point $\mathbf{k}$, and compared to the values returned by FLBE. The relative difference is usually lower than 10^{-8}, which proves the usability of the library for calculation of relaxation times.

Chapter 5

Benchmarks

Here we present the data which would help you to estimate the system size and calculation performance on your equipment. The data was obtained on AMD Ryzen 7 3700X 8-Core CPU (16 threads), 64 GB RAM, OS Linux (kernel v5.15.41), gcc compiler v11.2.1. We simulated the particle kinetics in interacting Bose gas described by Hamiltonian (3.4).

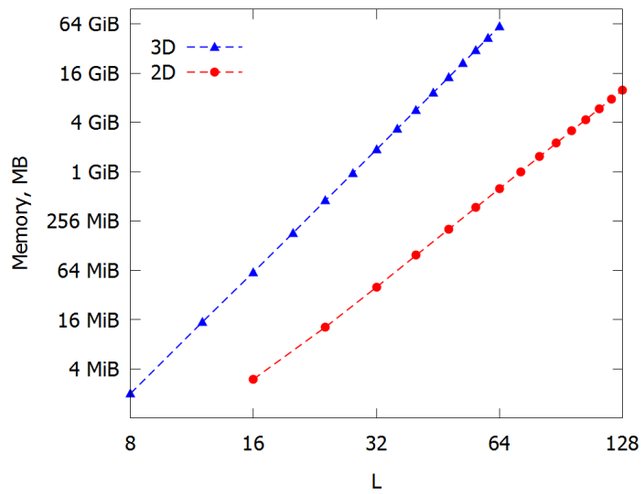
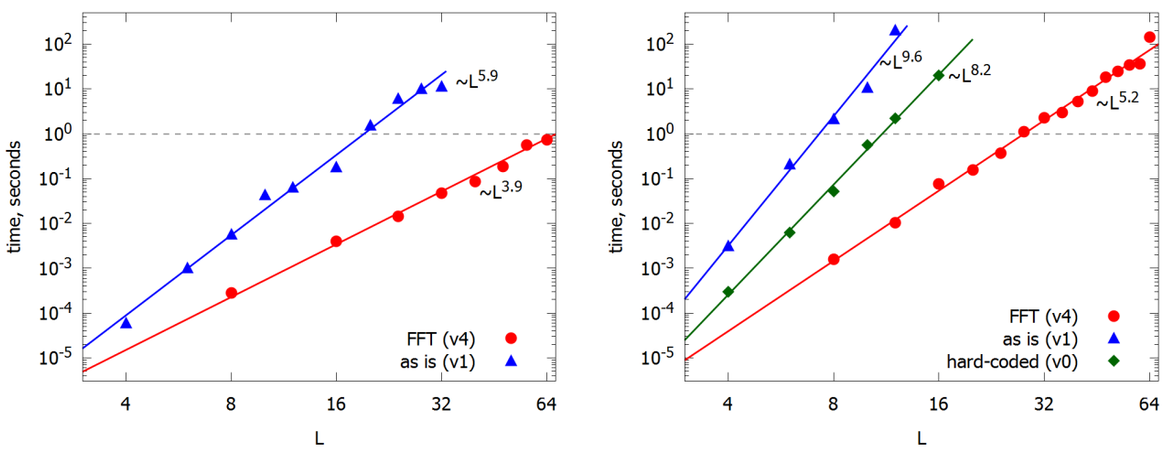


Figure 5.1: Memory consumption for the problem (3.4), depending on lattice size (2D and 3D cases)

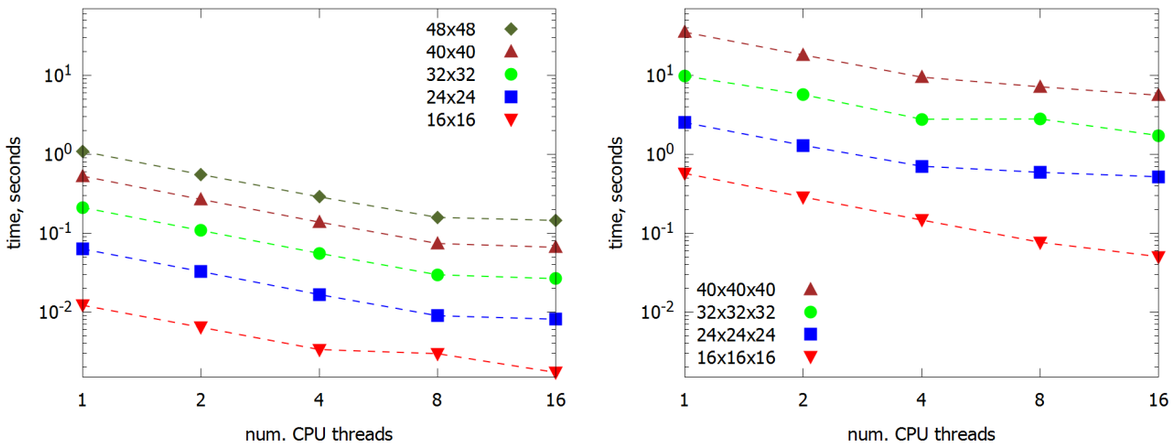


(a) 2D (b) 3D

Figure 5.2: Time per step for several algorithm versions, depending on lattice size: (a) 2D, (b) 3D. Solid lines show L^n fitting with power index indicated near to the line. Dashed line marks 1-second cutoff.

First, we present typical numbers for overall memory consumption (Figure 5.1). It is mostly related to internal arrays needed for Fourier transforms used in our accelerated approach. For a system with 64 GB RAM, we can take more than $L = 128$ for 2D (which takes approx. 10 GB RAM, and can be expanded even more) or up to $L = 64$ in 3D (taking approx. 61 GB RAM). However, it is reasonable to stick to smaller sizes to save the simulation time. See Figure 5.2: $L = 64 \ldots 80$ for 2D and $L = 28 \ldots 36$ for 3D are large enough and still take no more than a few seconds per calculation step.

In Figure 5.3, we show the performance for different number of OpenMP threads. As we see, the library benefits from parallelization and can use all CPU resources. The efficiency gain depends on the system size and is usually 5x-7x for 8 threads.



(a) 2D (b) 3D

Figure 5.3: Time per step, depending on number of CPU threads: (a) 2D, (b) 3D

Chapter 6

Caution!

There are several peculiarities you would want to beware:

- Required memory size can amount to tens of Gigabytes and more, depending on the problem dimensions. It is reasonable to check memory consumption with smaller lattice sizes, e.g. $L = 4$, $L = 6$, *etc.* For initial estimation, see graphs in the Benchmarks.
- There is possibility of memory leaks, due to simplified syntax for object creation. Usually it is not crucial here, as the object graph has few elements and is created only once before the calculation.
- Follow the recommended order of object creation (`Subsystem`, then `Problem`, then `BoltzmannEquation`, *etc.*) Take care not to change the internal objects in the middle of calculation, otherwise some secondary data can lose costistency leading to unexpected glitches.

Chapter 7

Example of calculation

Some minimal program, see `src/demo/minimal.f90`:

```
include 'flbe.f90'

use flbe
implicit none
integer, parameter :: L=4

class(Problem), pointer :: prob
class(Geometry), pointer :: sizes
class(BoltzmannEquation), pointer :: equation
class(FiniteDifferenceSolver), pointer :: solver

 call omp_set_num_threads( 4 )
 call flbe_init

 ! initial equilibrium distribution with given density and temperature
 sizes => Geometry( L, L, L )
 prob => InteractingBoseGas( sizes, particleDensity=2.d0, kT=10.d0 )
 equation => BoltzmannEquation( prob )
 solver => FD_Euler( equation )

 solver % n % values( :, :, :, :) = &
&  prob % equilibriumOccupations % values( :, :, :, :)

 ! offset from equilibrium in k=(1,1,1)
 solver % n % values( 1, 1, 1, :) = &
&  prob % equilibriumOccupations % values( 1, 1, 1, :) + 0.1d0

 print*, 'n(1,1,1)=', solver % n % values( 1, 1, 1, :)
 call solver % simulate( tmin=0.d0, tmax=1.0d-5, numSteps=1000 )
 print*, 'n(1,1,1)=', solver % n % values( 1, 1, 1, :)

 call flbe_done

end program
```

In the root of downloaded package, type `make minimal.exe` and run the created executable `minimal.exe`:

```
  n(111)=   2.1791087997341037
1000/1000 DONE in     0.97 seconds.
  n(111)=   2.1699662703842955
```

Chapter 8

Conclusion

8.1 Cheat sheet

`include 'flbe.f90'`	in your Fortran program
`use flbe`	in program blocks where you call the library functions
`call flbe_init`	after setting the number of OpenMP threads
`call flbe_done`	in the end, to release the resources
`-I D:/development/flbe`	mention the path to the library in the compilator keys

Table 8.1: Short list of required commands

8.2 Next steps of development

Program side, languages:

- Python interface and example code
- C/C++ interface and example code
- Using MPI for simulation on large clusters
- CUDA and OpenCL for GPU calculations

Program side, usability:

- Precompiled binaries (Windows and Linux) and headers, for easier integration
- Better isolation of library layers and checking to avoid data corruption
- Correct equations in output files (*.tex and *.html) for complicated cases

Physics side:

- Add classes for optical pumping, *e.g.* $\hat{a}^\dagger_{\mathbf{p}}\hat{a}^\dagger_{-\mathbf{p}} + H.c.$ with corresponding energy change $\pm\hbar\omega$
- More classes for standard Hamiltonians of solid state physics, e.g. multiband Hubbard and related models.
- Template programs for popular studies and modern experiments.

Bibliography

[1] I. O. Kuznetsov, P. F. Kartsev, "Method of numerical simulation of interacting quantum gas kinetics on finite momentum lattice", arXiv:2011.14173 [cond-mat.mes-hall], Nov. 2020.

[2] P. F. Kartsev, "Effective simulation of kinetic equations for bosonic system with two-particle interaction using OpenCL", Proceedings of IWOCL'17, 28 (2017), http://doi.org/10.1145/3078155.3078185.

[3] Currently prepared for publication.

[4] Ryo Hanai, Peter B. Littlewood, and Yoji Ohashi, "Photoluminescence and gain/absorption spectra of a driven-dissipative electron-hole-photon condensate", Phys. Rev. B **97**, 245302 (2018), https://doi.org/10.1103/PhysRevB.97.245302.

[5] V. V. Kabanov and A. S. Alexandrov, "Electron relaxation in metals: Theory and exact analytical solutions", Phys. Rev. B **78**, 174514 (2008), https://doi.org/10.1103/PhysRevB.78.174514.

[6] A. S. Kurdyubov, A. V. Trifonov, I. Ya. Gerlovin, B. F. Gribakin, P. S. Grigoryev, A. V. Mikhailov, I. V. Ignatiev, Yu. P. Efimov, S. A. Eliseev, V. A. Lovtcius, M.Aßmann, M. Bayer, and A. V. Kavokin. "Optical control of a dark exciton reservoir", Phys. Rev. B **104**, 035414 (2021), https://doi.org/10.1103/PhysRevB.104.035414.

[7] J. Demsar, "Non-equilibrium Phenomena in Superconductors Probed by Femtosecond Time-Domain Spectroscopy", J Low Temp Phys **201**, 676–709 (2020), https://doi.org/10.1007/s10909-020-02461-y.

[8] M. Król, R. Mirek, D. Stephan, K. Lekenta, J.-G. Rousset, W. Pacuski, A. V. Kavokin, M. Matuszewski, J. Szczytko, and B. Piętka, "Giant spin Meissner effect in a nonequilibrium exciton-polariton gas", Phys. Rev. B **99**, 115318 (2019), https://doi.org/10.1103/PhysRevB.99.115318.

[9] P.F. Kartsev, I.O. Kuznetsov, "Effect of transport current on suppression of superconductivity with ultrashort laser pulse", J. Phys.: Condens. Matter, vol. 33, Art. no. 295601, July 2021, http://doi.org/10.1088/1361-648X/abff91.

[10] I. O. Kuznetsov and P. F. Kartsev, "Numerical Simulation of Suppression of Superconductivity With Ultrashort Laser Pulse", IEEE Transactions on Applied Superconductivity, **32** (4), Art no. 9000805, June 2022, https://doi.org/10.1109/TASC.2022.3149458.