GPuE

- GPU integrator for the Gross–Pitaevskii Equation -

Jake Glidden

June 12, 2020

---

1 Introduction

This is the documentation for the **`gpeSolver_RK4`** package for numerical integration of the Gross–Pitaevskii equation in three dimensions.

$$i\hbar\frac{\partial}{\partial t}\Psi = -\frac{\hbar^2}{2m}\nabla^2\Psi + V(x,y,z,t)\Psi + \frac{4\pi\hbar^2 a}{m}|\Psi|^2\Psi \tag{1}$$

The programme is specifically configured for a cylindrical finite-square-well trap, and uses the fourth-order Runge–Kutta method to compute the relevant increments. The programme uses the CUDA features of modern graphics processing units (GPUs) to efficiently compute various parts of the numerics in parallel, most notably making use of the cuFFT library for parallelised Fourier transforms, and cuBLAS for some basic matrix operations.

It includes a period of evolution in *imaginary time* at the start of the simulation, which is helpful to find the ground state. The subsequent evolution is for a potential made of two parts, the *static* and *time-varying* contributions, which may be written as

$$V_{\text{box}} = \begin{cases} 0\,, & (x^2+y^2 < R^2) \,\wedge\, (2|z| < L) \\ U_{\text{D}}\,, & \text{otherwise} \end{cases} \tag{2}$$

i.e., a finite square well of depth U_{D} for a cylindrical geometry of radius R and length L, and

$$V_t = Fz\sin(2\pi f t) \tag{3}$$

respectively. In addition, an (optional) absorbing boundary is placed at the edges of the grid, which is implemented as an imaginary potential with the form

$$V_{\text{bnd}} = -i\cdot A\cdot\max\left[\cosh^{-2}\left(\tfrac{1}{w}\cdot\left[1-\tfrac{|x|}{\max x}\right]\right),\cosh^{-2}\left(\tfrac{1}{w}\cdot\left[1-\tfrac{|y|}{\max y}\right]\right),\cosh^{-2}\left(\tfrac{1}{w}\cdot\left[1-\tfrac{|z|}{\max z}\right]\right)\right]\,. \tag{4}$$

Here w is a width parameter corresponding to the distance the absorbing boundary should extend into the computational grid, taken as a fraction of the total grid length, and A is the strength of the potential. Both parameters should be chosen such that (at at minimum) atoms confined within the trap are not lost on appropriate timescales.

The total potential is the sum of these three contributions, $V_{\text{box}} + V_t + V_{\text{bnd}}$.

The programme is configured to use single-precision floating point values for real quantities, and pairs of single-precision floating point values for complex quantities.

2 Dependencies

The **`gpeSolver_RK4`** package has been built with:

- CUDA 9.0
- the cuFFT library
- the cuBLAS library
- VisualStudio 2015 (CE)
- Windows 10 (64-bit architecture)

3 Installation

♣

4 Usage

Once the package has been installed, it can be run from the command line as:

```
$ [/path/to]gpeSolver_RK4.exe [opts]
```

with one or more options as specified in the Options section below.

4.1 Units

Unless otherwise specified, units are 'natural' for the field of ultracold Bose gases:

- Lengths are given in [μm] **except** for s-wave scattering lengths, which are given in units of the Bohr radius [a_0]
- Momenta are correspondingly specified by wavenumber k in [μm^{-1}]
- Energies are given in [nK]
- Times are given in [ms]

4.2 Options

The runtime configurable options are given below. In each case, they're passed with a single leading hyphen (-), followed by a space, and then their corresponding value. The order they are passed, with the exception of the NOutputs/OutputTimes parameters, are unimportant. The NOutputs/OutputTimes pair must be specified together, and in that order.

- **-AtomNumber**
  Default value: 1.0×10^5. Floating-point value for the atom number.
- **-ScatteringLength**
  Default value: 200.0. Floating-point value for the s-wave scattering length, in units of the Bohr radius a_0.
- **-Radius**
  Default value: 15.0. Floating-point value for the cylindrical box radius R as in Eq. (2), in units of [μm].
- **-Length**
  Default value: 50.0. Floating-point value for the cylindrical box length L as in Eq. (2), in units of [μm].
- **-Depth**
  Default value: 300.0. Floating-point value for the cylindrical box depth U_D as in Eq. (2), in units of [nK].
- **-ITSimulationLength**
  Default value: 20.0. Floating-point value for the [I]maginary [T]ime simulation length, in units of [ms].
- **-SimulationLength**
  Default value: 1000.0. Floating-point value for the real time simulation length, in units of [ms].
- **-Timestep**
  Default value: 1.0×10^{-2}. Floating-point value for the length of a single timestep in the RK4 algorithm, in units of [ms]. The algorithm divides this once more into a half-step internally.
- **-NX** (also -NY, -NZ)
  Default value: 128. Unsigned integer for the number of grid points along the corresponding grid axis.
- **-XRange** (also -YRange, -ZRange)
  Default value: 100.0. Accepts a floating-point value for the length of the grid along the corresponding axis, in units of [μm].
- **-ShakeAmplitude**
  Default value: 0.2. Floating-point value for forcing amplitude F in Eq. (3), specified in units of [nK.μm^{-1}].
- **-ShakeFrequency**
  Default value: 20.0. Floating-point value for the drive frequency f in Eq. (3), ***specified in Hz***.
- **-NOutputs**
  Default value: 2. Unsigned integer for the number of timesteps at which output information should be saved.

- **-OutputTimes**
  Default value: [*0.0, <SimulationLength>*]. Accepts a comma-separated list of <NOutputs> floating-point values at which the output information should be saved. <NOutputs> ***must*** have been specified before this option on the command line, as it is needed in order to correctly allocate memory for the list. This list ***must not*** contain spaces, or it will not be parsed correctly.
- **-BoundaryDepth**
  Default value: 20.0. Floating-point value for the strength A of the (imaginary) absorbing potential at the edges of the grid as in Eq. (4), in units of [nK].
- **-BoundaryWidth**
  Default value: 0.05. Floating-point fraction w of the the grid into which the absorbing boundary should extend, as in Eq. (4).

A fully-qualified example call to **`gpeSolver_RK4`** might then look like:

```
$ [/path/to/]gpeSolver_RK4.exe -AtomNumber 1.0e4 -ScatteringLength 300.0 -Radius 15.0 -Length 50.0 -Depth 300.0 -
ITSimulationLength -BoundaryDepth 20.0 -BoundaryWidth 0.05 -NX 128 -NY 128 -NZ 256 -XRange 50.0 -YRange 50.0 -
ZRange 100.0 -Timestep 5.0e-3 -ShakeAmplitude 1.0e-2 -ShakeFrequency 20.0 -SimulationLength 100.0 -NOutputs 5 -
OutputTimes 0.0,10.0,20.0,50.0,100.0
```

4.3 Batch processing

In its current form, the simulation is designed to run only one set of parameters at a time, which should run to completion. One convenient way to queue several different parameter sets is using a 'batch' wrapper, that will call **`gpeSolver_RK4`** with a set of arguments, wait for those arguments to complete, and then start the next set. One can, e.g., put the text below into a file with the '.bat' extension', and save that file in any directory you like. The directory will be the location that outputs are stored (although the program will automatically create a subdirectory called 'Outputs').

Note that the entries in a batch file are interpreted live, rather than checked in advance; it pays to exercise some caution to avoid typographical errors, etc. Where a command is executed several times with only a single parameter change, then one should consider using an appropriate batch loop construction.

```
@echo off
rem Lines starting with rem are comments
rem Lines starting with echo print to the console
rem @echo [on/off] switches whether every line is both executed and printed, or just executed
rem You can use a trailing carat (^) to continue a command onto the next line (like ... in MATLAB, or \ in various languages)
rem -- but make sure to leave a space before!
rem echo: (with the trailing colon) prints a blank line to the console

title Simulation manager

rem With this we can put %logg% at the end of a command and it will be both displayed on screen, and written to a log file.
set TEMPLOG="%TMP%\batchlog.txt"
set LOGFILE="200a0_batchlog.txt"
set logg=^>%TEMPLOG% ^&^& type %TEMPLOG% ^&^& type %TEMPLOG%^>^>%LOGFILE%

rem PARAMETERS FOR 200a0
echo ========================================== %logg%
echo * 200a0                          * %logg%
echo ========================================== %logg%

FOR %%G IN ("100.0,0.1" "200,0.2" "400,0.4") DO (
FOR /F "delims=, tokens=1*" %%H in (%%G) DO (
rem Now %%H, %%I, %%J, etc. have each element of the comma-separated list
echo ^>^> Running with scattering length %%H a0 and shake amplitude %%I nK per um ... %logg%
start /WAIT "%%H a0, amp. %%I nK per um" CMD.EXE /C ^("D:\Documents\Visual Studio 2015\Projects\gpeSolver_RK4\x64\
RK4_Release\gpeSolver_RK4.exe" ^
-AtomNumber 1.0e5 -ScatteringLength %%H -ITSimulationLength 50 -Radius 15.0 -Length 50.0 ^
-Depth 300.0 -Timestep 1.0e-2 -ShakeFrequency 15.6 -ShakeAmplitude %%I ^
-SimulationLength 3000 -NOutputs 15 -OutputTimes 0,64,128,192,256,320,641,961,1282,1602,1923,2243,2564,2884,3000 ^)
)
```

```
echo ^>^> Complete. %logg%
echo: %logg%
echo: %logg%
)

pause
```

5 Outputs

As soon as the programme starts, it should tell you the name of the timestamped output directory (within the directory from which the executable was called). This directory will contain a file called 'Log.txt' that details the parameters used to run the code. It also lists the relevant geometry elements, including vectors for the x, y, and z grids, as well as their corresponding momenta (with the prefix 'k').

The full k-space wavefunction is saved as a kpsi.*.bin[ary] dump (which is not currently very helpful, but can be converted later if useful) for each of your <OutputTimes>.

Once-integrated 'images' are also saved in .csv files, where an integral has been performed over the density along either the axis of the cylinder ($\int \mathrm{d}z \cdots$) or one of the orthogonal directions ($\int \mathrm{d}x \cdots$). These files exist both in momentum space and in coordinate space. Their filenames might seem daunting, but are to be read as in the following example:

I_kdens_dkz.t_+4.000e+01.csv - [I]ntegral of the [k/momentum] density along [dkz] at time [t] equal to 4.0×10^1 ms.

I've included a MATLAB excerpt on the next page that reads the geometry information from 'Log.txt' (helper function at the end of the definition) and constructs a four-panel plot containing the momentum and coordinate space images, integrated either 'horizontally' or 'vertically' as we would say in our experiments. The second argument is currently used to select the time (in [ms]), e.g. to preview the outputs in the directory 'gpeSolver_RK4-2020-06-11T18-37-33' at time $t = 5.0$ ms one would call

```
gpeanalyse('gpeSolver_RK4−2020−06−11T18−37−33', 5.0)
```

```
function [g] = gpeanalyse(myfilename, varargin)

  if (~exist(myfilename, 'dir'))
     myfilename = strsplit(myfilename, '\');
     myfilename = myfilename{end};
     myfilename = ['D:\Documents\Visual Studio 2015\Projects\gpeSolver_RK4\gpeSolver_RK4\Outputs\' ...
        myfilename];
  end

  mycmapname = 'jet';

  if nargin > 1
     mytimestring = num2str(varargin{1}, '%+.3e');
  else
     mytimestring = num2str(0, '%+.3e');
  end

  g = loadgeometry([myfilename '\Log.txt']);

  figure('WindowStyle', 'docked');

  x = dlmread([myfilename '\I_kdens_dkx.t_' mytimestring '.csv']);
  ax = subplot(2,2,1);
  imagesc(ax, g.ky, g.kz, log(x'));
  axis(ax,'tight');
  pbaspect(ax, [range(g.ky) range(g.kz) 1]);
  set(ax,'TickDir','out','TickLength',[0.03,0.03],'LineWidth',1)
  colormap(ax, mycmapname);

  x = dlmread([myfilename '\I_kdens_dkz.t_' mytimestring '.csv']);
  ax = subplot(2,2,2);
  imagesc(ax, g.kx, g.ky, log(x'));
  axis(ax,'tight');
  pbaspect(ax, [range(g.kx) range(g.ky) 1]);
  set(ax,'TickDir','out','TickLength',[0.03,0.03],'LineWidth',1)
  colormap(ax, mycmapname);

  x = dlmread([myfilename '\I_dens_dx.t_' mytimestring '.csv']);
  ax = subplot(2,2,3);
  imagesc(ax, g.y, g.z, x');
  axis(ax,'tight');
  set(ax,'TickDir','out','TickLength',[0.03,0.03],'LineWidth',1)
  pbaspect(ax, [range(g.y) range(g.z) 1]);
  colormap(ax, mycmapname);
  caxis(ax, [0 142])

  x = dlmread([myfilename '\I_dens_dz.t_' mytimestring '.csv']);
  ax = subplot(2,2,4);
  imagesc(ax, g.x, g.y, x');
  axis(ax,'tight');
  set(ax,'TickDir','out','TickLength',[0.03,0.03],'LineWidth',1)
  pbaspect(ax, [range(g.x) range(g.y) 1]);
  colormap(ax, mycmapname);
  caxis(ax, [0 142])
  fprintf('%.3e: %10.4e\n', varargin{1}, sum(x(:)) * g.dx * g.dy);

  suptitle(num2str(varargin{1}))

  function res = loadgeometry(f)
     res = struct();

     myfile = fopen(f, 'r');
     lastline = fgetl(myfile);
```

```
    while (~strcmp(lastline,'== Vectors')), lastline = fgetl(myfile); end % skip to the vectors

    contflag = true;
    while (contflag)
      lastline = fgetl(myfile);
      while (strcmp(lastline,'')) % skip newlines
        lastline = fgetl(myfile);
        if (lastline == -1), contflag=false; break; end
      end
      if (~contflag), break; end
      varname = lastline;
      lastline = fgetl(myfile);
      if lastline == -1, contflag = false; break; end
      vals = cellfun(@str2num,strsplit(lastline, ','));
      res.(varname) = vals;
      res.(['d' varname]) = mean(diff(vals));
    end

    fclose(myfile);
  end

end
```