

# GP(U E): GPU Gross-Pitaevskii Equation

0

Generated by Doxygen 1.8.6

Mon Jul 6 2015 17:26:33



# Contents



# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">BEC2D</a>	.....	??
<a href="#">FileIO</a>		
	Check source file for information on functions .....	??
<a href="#">hist3d</a>	.....	??
<a href="#">hist_it</a>	.....	??
<a href="#">Minions</a>	.....	??
<a href="#">observables</a>	.....	??
<a href="#">stats</a>	.....	??
<a href="#">Tracker</a>		
	See the source file for info on functions .....	??
<a href="#">vis</a>	.....	??
<a href="#">vort</a>	.....	??



## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">BEC2D::Wavefunction</a>	.....	??
-------------------------------------	-------	----





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

bin/ <a href="#">batch_run.sh</a>	??
bin/ <a href="#">path.sh</a>	??
bin/ <a href="#">run.sh</a>	??
bin/ <a href="#">sanity_test.sh</a>	??
bin/ <a href="#">upload_vids.sh</a>	??
bin/ <a href="#">zippit.sh</a>	??
include/ <a href="#">constants.h</a>	??
include/ <a href="#">ds.h</a>	??
include/ <a href="#">fileIO.h</a>	??
include/ <a href="#">kernels.h</a>	??
include/ <a href="#">minions.h</a>	??
include/ <a href="#">split_op.h</a>	??
include/ <a href="#">tracker.h</a>	??
py/ <a href="#">hist3d.py</a>	??
py/ <a href="#">hist_it.py</a>	??
py/ <a href="#">observables.py</a>	??
py/ <a href="#">stats.py</a>	??
py/ <a href="#">vis.py</a>	??
py/ <a href="#">vort.py</a>	??
src/ <a href="#">ds.cc</a>	??
src/ <a href="#">fileIO.cc</a>	??
src/ <a href="#">kernels.cu</a>	??
src/ <a href="#">minions.cc</a>	??
src/ <a href="#">multigpu.cu</a>	??
src/ <a href="#">split_op.cu</a>	??
src/ <a href="#">srt.cc</a>	??
src/ <a href="#">tracker.cc</a>	??
src/ <a href="#">wavefunction.cu</a>	??



## Chapter 4

# Namespace Documentation

### 4.1 BEC2D Namespace Reference

#### Classes

- class [Wavefunction](#)

#### Functions

- class [BEC2D::Wavefunction](#) [Wavefunction](#) ( )

#### 4.1.1 Function Documentation

##### 4.1.1.1 class [BEC2D::Wavefunction](#) [BEC2D::Wavefunction](#) ( )

Definition at line [54](#) of file [wavefunction.cu](#).

```
00054                                     {  
00055  
00056     }
```

### 4.2 FileIO Namespace Reference

Check source file for information on functions.

#### Functions

- double2 \* [readIn](#) (char \*, char \*, int, int)
- int [readState](#) (char \*)
- void [writeOut](#) (char \*, char \*, double2 \*, int, int)
- void [writeOutDouble](#) (char \*, char \*, double \*, int, int)
- void [writeOutInt](#) (char \*, char \*, int \*, int, int)
- void [writeOutInt2](#) (char \*, char \*, int2 \*, int, int)
- void [writeOutParam](#) (char \*, [Array](#), char \*)
- void [writeOutVortex](#) (char \*, char \*, struct [Tracker::Vortex](#) \*, int, int)

### 4.2.1 Detailed Description

Check source file for information on functions.

### 4.2.2 Function Documentation

#### 4.2.2.1 `double2 * FileIO::readIn ( char * fileR, char * fileI, int xDim, int yDim )`

Definition at line 45 of file `fileIO.cc`.

References `FILE`, `vis::i`, and `yDim`.

Referenced by `main()`.

```

00045                                     {
00046         FILE *f;
00047         f = fopen(fileR, "r");
00048         int i = 0;
00049         double2 *arr = (double2*) malloc(sizeof(double2)*xDim*yDim);
00050         double line;
00051         while (fscanf(f, "%le", &line) > 0) {
00052             arr[i].x = line;
00053             ++i;
00054         }
00055         fclose(f);
00056         f = fopen(fileI, "r");
00057         i = 0;
00058         while (fscanf(f, "%le", &line) > 0) {
00059             arr[i].y = line;
00060             ++i;
00061         }
00062         fclose(f);
00063         return arr;
00064     }

```

Here is the caller graph for this function:

#### 4.2.2.2 `int FileIO::readState ( char * name )`

Definition at line 156 of file `fileIO.cc`.

References `FILE`.

```

00156                                     {
00157         FILE *f;
00158         f = fopen(name, "r");
00159         fclose(f);
00160         return 0;
00161     }

```

#### 4.2.2.3 `void FileIO::writeOut ( char * buffer, char * file, double2 * data, int length, int step )`

Definition at line 84 of file `fileIO.cc`.

References `FILE`, `vis::i`, `x`, and `y`.

Referenced by `evolve()`, `initialise()`, and `optLatSetup()`.

```

00084                                     {
00085         FILE *f;
00086         sprintf (buffer, "%s_%d", file, step);
00087         f = fopen (buffer, "w");
00088         int i;
00089         for (i = 0; i < length; i++)
00090             fprintf (f, "%.16e\n", data[i].x);
00091         fclose (f);
00092
00093         sprintf (buffer, "%si_%d", file, step);
00094         f = fopen (buffer, "w");

```

```

00095         for (i = 0; i < length; i++)
00096             fprintf (f, "%.16e\n", data[i].y);
00097         fclose (f);
00098     }

```

Here is the caller graph for this function:

#### 4.2.2.4 void FileIO::writeOutDouble ( char \* *buffer*, char \* *file*, double \* *data*, int *length*, int *step* )

Definition at line 103 of file [fileIO.cc](#).

References [FILE](#), and [vis::i](#).

Referenced by [evolve\(\)](#), [initialise\(\)](#), and [main\(\)](#).

```

00103                                     {
00104         FILE *f;
00105         sprintf (buffer, "%s_%d", file, step);
00106         f = fopen (buffer, "w");
00107         int i;
00108         for (i = 0; i < length; i++)
00109             fprintf (f, "%.16e\n", data[i]);
00110         fclose (f);
00111     }

```

Here is the caller graph for this function:

#### 4.2.2.5 void FileIO::writeOutInt ( char \* *buffer*, char \* *file*, int \* *data*, int *length*, int *step* )

Definition at line 116 of file [fileIO.cc](#).

References [FILE](#), and [vis::i](#).

```

00116                                     {
00117         FILE *f;
00118         sprintf (buffer, "%s_%d", file, step);
00119         f = fopen (buffer, "w");
00120         int i;
00121         for (i = 0; i < length; i++)
00122             fprintf (f, "%d\n", data[i]);
00123         fclose (f);
00124     }

```

#### 4.2.2.6 void FileIO::writeOutInt2 ( char \* *buffer*, char \* *file*, int2 \* *data*, int *length*, int *step* )

Definition at line 129 of file [fileIO.cc](#).

References [FILE](#), [vis::i](#), [x](#), and [y](#).

```

00129                                     {
00130         FILE *f;
00131         sprintf (buffer, "%s_%d", file, step);
00132         f = fopen (buffer, "w");
00133         int i;
00134         for (i = 0; i < length; i++)
00135             fprintf (f, "%d,%d\n", data[i].x, data[i].y);
00136         fclose (f);
00137     }

```

#### 4.2.2.7 void FileIO::writeOutParam ( char \* *buffer*, Array *arr*, char \* *file* )

Definition at line 69 of file [fileIO.cc](#).

References [Array::array](#), [Param::data](#), [FILE](#), [vis::i](#), [Param::title](#), and [Array::used](#).

Referenced by [evolve\(\)](#), and [main\(\)](#).

```

00069                                     {
00070         FILE *f;
00071         sprintf(buffer, "%s", file);
00072         f = fopen(file, "w");
00073         fprintf(f, "[Params]\n");
00074         for (int i = 0; i < arr.used; ++i){
00075             fprintf(f, "%s=", arr.array[i].title);
00076             fprintf(f, "%e\n", arr.array[i].data);
00077         }
00078         fclose(f);
00079     }

```

Here is the caller graph for this function:

#### 4.2.2.8 void FileIO::writeOutVortex ( char \* *buffer*, char \* *file*, struct Tracker::Vortex \* *data*, int *length*, int *step* )

Definition at line 142 of file [fileIO.cc](#).

References [Tracker::Vortex::coords](#), [FILE](#), [vis::i](#), [Tracker::Vortex::sign](#), and [Tracker::Vortex::wind](#).

Referenced by [evolve\(\)](#).

```

00142                                     {
00143         FILE *f;
00144         sprintf (buffer, "%s_%d", file, step);
00145         f = fopen (buffer, "w");
00146         int i;
00147         fprintf (f, "#X,Y,WINDING,SIGN\n");
00148         for (i = 0; i < length; i++)
00149             fprintf (f, "%d,%d,%d,%d\n", data[i].coords.x, data[i].coords.y, data[i].
wind, data[i].sign);
00150         fclose (f);
00151     }

```

Here is the caller graph for this function:

## 4.3 hist3d Namespace Reference

### Functions

- def [plot\\_hist\\_pcolor](#)
- def [plot\\_xyz\\_histogram](#)

### Variables

- tuple [c](#) = ConfigParser.ConfigParser()
- tuple [dt](#) = (c.getfloat('Params','dt'))
- tuple [dx](#) = (c.getfloat('Params','dx'))
- tuple [evMaxVal](#) = int(c.getfloat('Params','esteps'))
- tuple [gndMaxVal](#) = int(c.getfloat('Params','gsteps'))
- tuple [incr](#) = int(c.getfloat('Params','print\_out'))
- int [num\\_vort](#) = 0
- tuple [sep](#) = (c.getfloat('Params','dx'))
- tuple [xDim](#) = int(c.getfloat('Params','xDim'))
- tuple [xMax](#) = (c.getfloat('Params','xMax'))
- tuple [yDim](#) = int(c.getfloat('Params','yDim'))
- tuple [yMax](#) = (c.getfloat('Params','yMax'))

### 4.3.1 Detailed Description

hist3d.py - GPUE: Split Operator based GPU solver for Nonlinear Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O’Riordan <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### 4.3.2 Function Documentation

#### 4.3.2.1 `def hist3d.plot_hist_pcolor ( start, fin, incr, barcolor )`

Definition at line 87 of file [hist3d.py](#).

Referenced by [vis.overlap\(\)](#).

```

00087
00088 def plot_hist_pcolor(start,fin,incr, barcolor):
00089     fig = plt.figure()
00090
00091     data = []
00092     for i in range(start, fin, incr):
00093         v_arr=genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',')
00094         datatmp=[]
00095         count=0
00096
00097         for i1 in range(0,v_arr.size/2):
00098             for i2 in range(i1,v_arr.size/2):
00099                 m_tmp = m.sqrt(abs(v_arr[i1][0]*sep - v_arr[i2][0]*sep)**2 + abs(v_arr[i1][1]*sep - v_arr
00100 [i2][1]*sep)**2 )
00101                 datatmp.append( m_tmp )
00102                 count = count + 1
00103             hist=np.histogram(datatmp,bins=np.arange(0.0,240.0,0.1))
00104             data.append(hist[:][0])
00105
00106     # print data
00107     ax = fig.add_subplot(111)
00108     ax.imshow(data)
00109     plt.gca().invert_yaxis()
00110     ax.set_aspect('auto')
00111     # plt.jet()
00112     fig.savefig("HIST_PCOLOR.pdf")
00113
00114 #plot_xyz_histogram(0,100000,100,'b')
00115 #plot_hist_pcolor(0,100000,100,'b')
00116

```

Here is the caller graph for this function:

#### 4.3.2.2 def hist3d.plot\_xyz\_histogram ( start, fin, incr, barcolor )

Definition at line 57 of file [hist3d.py](#).

```

00057
00058 def plot_xyz_histogram(start,fin,incr, barcolor):
00059     fig = plt.figure()
00060     ax = Axes3D(fig)
00061     data = []
00062     for i in range(start, fin, incr):
00063         v_arr=genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',')
00064         datatmp=[]
00065         count=0
00066
00067         for i1 in range(0,v_arr.size/2):
00068             for i2 in range(i1,v_arr.size/2):
00069                 datatmp.append(m.sqrt( abs(v_arr[i1][0]*sep - v_arr[i2][0]*sep)**2 + abs(v_arr[i1][1]*sep
- v_arr[i2][1]*sep)**2 ))
00070                 count = count + 1
00071                 hist=np.histogram(datatmp,bins=np.arange(1.0,m.sqrt(xDim**2 + yDim**2),1.0))
00072                 data.append(hist[:,0])
00073             """ Takes in a matrix (see structure above) and generate a pseudo-3D histogram by overlaying close,
semitransparent bars. """
00074         for time, occurrence in zip(range(len(data)), data):
00075             dist = range(len(occurrence))
00076             barband = range(-45, 45, 5)
00077             #for modifier in barband:
00078                 ax.bar(dist, occurrence, zs=time, zdir='y', color=np.random.rand(3,1), alpha=0.8)
00079                 #ax.bar(current, occurrence, zs=duration+(float(modifier)/100), zdir='y',
color=np.random.rand(3,1), alpha=0.6)
00080
00081         ax.set_xlabel('Dist')
00082         ax.set_ylabel('Time')
00083         ax.set_zlabel('Occurrences')
00084
00085         plt.savefig("HIST_N.pdf")
00086         plt.show()

```

### 4.3.3 Variable Documentation

#### 4.3.3.1 tuple hist3d.c = ConfigParser.ConfigParser()

Definition at line 41 of file [hist3d.py](#).

Referenced by [Minions.complexDiv\(\)](#), and [Minions.conj\(\)](#).

#### 4.3.3.2 tuple hist3d.dt = (c.getfloat('Params','dt'))

Definition at line 51 of file [hist3d.py](#).

#### 4.3.3.3 tuple hist3d.dx = (c.getfloat('Params','dx'))

Definition at line 50 of file [hist3d.py](#).

#### 4.3.3.4 tuple hist3d.evMaxVal = int(c.getfloat('Params','esteps'))

Definition at line 47 of file [hist3d.py](#).

#### 4.3.3.5 tuple hist3d.gndMaxVal = int(c.getfloat('Params','gsteps'))

Definition at line 46 of file [hist3d.py](#).

#### 4.3.3.6 tuple hist3d.incr = int(c.getfloat('Params','print\_out'))

Definition at line 48 of file [hist3d.py](#).



4.3.3.7 `int hist3d.num_vort = 0`

Definition at line 54 of file [hist3d.py](#).

4.3.3.8 `float hist3d.sep = (c.getfloat('Params','dx'))`

Definition at line 49 of file [hist3d.py](#).

4.3.3.9 `tuple hist3d.xDim = int(c.getfloat('Params','xDim'))`

Definition at line 44 of file [hist3d.py](#).

4.3.3.10 `tuple hist3d.xMax = (c.getfloat('Params','xMax'))`

Definition at line 52 of file [hist3d.py](#).

4.3.3.11 `tuple hist3d.yDim = int(c.getfloat('Params','yDim'))`

Definition at line 45 of file [hist3d.py](#).

4.3.3.12 `tuple hist3d.yMax = (c.getfloat('Params','yMax'))`

Definition at line 53 of file [hist3d.py](#).

## 4.4 hist\_it Namespace Reference

### 4.4.1 Detailed Description

hist\_it.py - GPUE: Split Operator based GPU solver for Nonlinear Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 4.5 Minions Namespace Reference

### Functions

- double2 [complexDiv](#) (double2 num, double2 den)
- double [complexMag](#) (double2 in)
- double [complexMag2](#) (double2 in)
- double2 [complexMult](#) (double2 in1, double2 in2)
- double2 [complexScale](#) (double2 comp, double scale)
- double2 [conj](#) (double2 c)
- void [coordSwap](#) (struct [Tracker::Vortex](#) \*vCoords, int src, int dest)
- double [flnvSqRt](#) (double)
- *id magic hackery*
- double [maxValue](#) (double \*, int)
- double [minValue](#) (double \*, int)
- double [psi2](#) (double2)
- double [sumAvg](#) (double \*in, int len)

### 4.5.1 Function Documentation

#### 4.5.1.1 double2 Minions::complexDiv ( double2 num, double2 den )

Definition at line 118 of file [minions.cc](#).

References [hist3d::c](#), [complexMag2\(\)](#), [complexMult\(\)](#), [complexScale\(\)](#), and [conj\(\)](#).

Referenced by [Tracker::findVortex\(\)](#), and [Tracker::phaseTest\(\)](#).

```
00118                                     {
00119         double2 c = conj(den);
00120         return complexScale(complexMult(num, c), (1.0/
00121         complexMag2(den)));
00121     }
```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 4.5.1.2 double Minions::complexMag ( double2 in )

Definition at line 90 of file [minions.cc](#).

Referenced by [Tracker::findVortex\(\)](#), and [Tracker::phaseTest\(\)](#).

```
00090                                     {
00091         return sqrt(in.x*in.x + in.y*in.y);
00092     }
```

Here is the caller graph for this function:

#### 4.5.1.3 double Minions::complexMag2 ( double2 in )

Definition at line 94 of file [minions.cc](#).

Referenced by [complexDiv\(\)](#).

```
00094                                     {
00095         return in.x*in.x + in.y*in.y;
00096     }
```

Here is the caller graph for this function:

4.5.1.4 double2 Minions::complexMult ( double2 *in1*, double2 *in2* )

Definition at line 98 of file [minions.cc](#).

References [result](#).

Referenced by [complexDiv\(\)](#).

```

00098                                     {
00099         double2 result;
00100         result.x = (in1.x*in2.x - in1.y*in2.y);
00101         result.y = (in1.x*in2.y + in1.y*in2.x);
00102         return result;
00103     }
```

Here is the caller graph for this function:

4.5.1.5 double2 Minions::complexScale ( double2 *comp*, double *scale* )

Definition at line 105 of file [minions.cc](#).

References [result](#).

Referenced by [complexDiv\(\)](#), [Tracker::findVortex\(\)](#), and [Tracker::phaseTest\(\)](#).

```

00105                                     {
00106         double2 result;
00107         result.x = comp.x*scale;
00108         result.y = comp.y*scale;
00109         return result;
00110     }
```

Here is the caller graph for this function:

4.5.1.6 double2 Minions::conj ( double2 *c* )

Definition at line 112 of file [minions.cc](#).

References [hist3d::c](#), and [result](#).

Referenced by [complexDiv\(\)](#).

```

00112                                     {
00113         double2 result = c;
00114         result.y = -result.y;
00115         return result;
00116     }
```

Here is the caller graph for this function:

4.5.1.7 void Minions::coordSwap ( struct Tracker::Vortex \* *vCoords*, int *src*, int *dest* )

Definition at line 84 of file [minions.cc](#).

Referenced by [Tracker::vortArrange\(\)](#).

```

00084                                     {
00085         struct Tracker::Vortex d = vCoords[dest];
00086         vCoords[dest] = vCoords[src];
00087         vCoords[src] = d;
00088     }
```

Here is the caller graph for this function:

#### 4.5.1.8 double Minions::flnvSqrt ( double *in* )

id magic hackery

Definition at line 69 of file [minions.cc](#).

References [in\(\)](#), and [l](#).

```

00069                                     {
00070         long long l;
00071         double in05, calc;
00072         const double threehalfs = 1.5;
00073
00074         in05 = in*0.5;
00075         calc=in;
00076         l = * (long long*) &calc;
00077         l = 0x5fe6eb50c7b537a9LL - (l >> 1);
00078         calc = *(double *) &l;
00079         calc = calc*( 1.5 - (in05*calc*calc) );
00080
00081         return calc;
00082     }
```

Here is the call graph for this function:

#### 4.5.1.9 double Minions::maxValue ( double \* *grid*, int *len* )

Definition at line 41 of file [minions.cc](#).

References [vis::i](#).

Referenced by [Tracker::findOLMaxima\(\)](#).

```

00041                                     {
00042         double max = grid[0];
00043         for (unsigned int i=1; i<len-1; ++i) {
00044             if (max<grid[i]) {
00045                 max=grid[i];
00046             }
00047         }
00048         return max;
00049     }
```

Here is the caller graph for this function:

#### 4.5.1.10 double Minions::minValue ( double \* *grid*, int *len* )

Definition at line 51 of file [minions.cc](#).

References [vis::i](#).

```

00051                                     {
00052         double min = grid[0];
00053         for (unsigned int i=1; i<len-1; ++i) {
00054             if (min>grid[i]) {
00055                 min=grid[i];
00056             }
00057         }
00058         return min;
00059     }
```

#### 4.5.1.11 double Minions::psi2 ( double2 *in* )

Definition at line 37 of file [minions.cc](#).

Referenced by [evolve\(\)](#).

```

00037                                     {
00038         return in.x*in.x + in.y*in.y;
00039     }
```

Here is the caller graph for this function:

## 4.5.1.12 double Minions::sumAvg ( double \* in, int len )

Definition at line 60 of file [minions.cc](#).

References [vis::i](#).

Referenced by [evolve\(\)](#).

```

00060                                     {
00061         double avg = 0.0;
00062
00063         for (unsigned int i=0; i<len; ++i){
00064             avg += in[i];
00065         }
00066         return avg/len;
00067     }

```

Here is the caller graph for this function:

## 4.6 observables Namespace Reference

### Functions

- def [ang\\_mom](#)
- def [dens\\_struct\\_fact](#)
- def [energy\\_kinetic](#)
- def [energy\\_potential](#)
- def [energy\\_total](#)
- def [expec\\_val\\_](#)
- def [expec\\_val\\_monopole](#)
- def [expec\\_val\\_quadrupole](#)
- def [kinertrum](#)
- *Kinetic energy spectrum = kinertrum.*
- def [kinertrum\\_loop](#)

### Variables

- tuple [c](#) = ConfigParser.ConfigParser()
- tuple [data](#) = numpy.ndarray(shape=([xDim](#),[yDim](#)))
- tuple [dkx](#) = (c.getfloat('Params','dpx'))
- tuple [dky](#) = (c.getfloat('Params','dpy'))
- tuple [dt](#) = (c.getfloat('Params','dt'))
- tuple [dx](#) = (c.getfloat('Params','dx'))
- tuple [dy](#) = (c.getfloat('Params','dy'))
- tuple [evMaxVal](#) = int(c.getfloat('Params','esteps'))
- tuple [g](#) = (0.5\*N)
- tuple [gndMaxVal](#) = int(c.getfloat('Params','gsteps'))
- float [HBAR](#) = 1.05457148e-34
- float [hbar](#) = 1.05457e-34
- tuple [incr](#) = int(c.getfloat('Params','print\_out'))
- tuple [K](#) = np.array(open('K\_0').read().splitlines(),dtype='f8')
- tuple [k\\_mag](#) = np.sqrt( [kx](#)\*\*2 + [ky](#)\*\*2 )
- tuple [km\\_mag](#) = np.sqrt( [kxm](#)\*\*2 + [kym](#)\*\*2 )
- tuple [kMax](#) = max(max([k\\_mag](#)))
- tuple [kx](#) = np.reshape( np.array( [np.linspace( 0, ([xDim](#)/2-1)\*[dkx](#), [xDim](#)/2), np.linspace( (-[xDim](#)/2-1)\*[dkx](#), -[dkx](#), [xDim](#)/2)]), ([xDim](#),1) )

- tuple `ky` = `np.reshape( np.array( [np.linspace( 0, (yDim/2-1)*dky, yDim/2), np.linspace( (-yDim/2-1)*dky, -dky, yDim/2)]), (yDim,1) )`
- float `m` = `1.4431607e-25`
- tuple `mass` = `(c.getfloat('Params','Mass'))`
- tuple `N` = `int(c.getfloat('Params','atoms'))`
- tuple `num_vort` = `int(c.getfloat('Params','Num_vort'))`
- tuple `omega` = `(c.getfloat('Params','omega'))`
- tuple `omegaX` = `(c.getfloat('Params','omegaX'))`
- tuple `omegaZ` = `(c.getfloat('Params','omegaZ'))`
- float `PI` = `3.141592653589793`
- tuple `V` = `np.array(open('V_0').read().splitlines(),dtype='f8')`
- tuple `x` = `np.asarray(open('x_0').read().splitlines(),dtype='f8')`
- tuple `xDim` = `int(c.getfloat('Params','xDim'))`
- tuple `xMax` = `(c.getfloat('Params','xMax'))`
- tuple `xPy` = `np.array(open('xPy_0').read().splitlines(),dtype='f8')`
- tuple `y` = `np.asarray(open('y_0').read().splitlines(),dtype='f8')`
- tuple `yDim` = `int(c.getfloat('Params','yDim'))`
- tuple `yMax` = `(c.getfloat('Params','yMax'))`
- tuple `yPx` = `np.array(open('yPx_0').read().splitlines(),dtype='f8')`

#### 4.6.1 Detailed Description

observables.py - GPUE: Split Operator based GPU solver for Nonlinear Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

#### 4.6.2 Function Documentation

##### 4.6.2.1 `def observables.ang_mom( dataName, initValue, finalValue, incr, ev_type, imgdpi )`

Definition at line 293 of file `observables.py`.

Referenced by `expec_val_()`.

```

00293
00294 def ang_mom(dataName, initValue, finalValue, incr, ev_type, imgdpi):
00295     xm, ym = np.meshgrid(x,y)
00296     pxm, pym = np.meshgrid(px,py)
00297     dx2=dx**2
00298     Lz = np.zeros( (finalValue/incr))
00299     for i in range(initValue,incr*(finalValue/incr),incr):
00300         if os.path.exists(dataName + '_' + str(i)):
00301             real=open(dataName + '_' + str(i)).read().splitlines()
00302             img=open(dataName + 'i_' + str(i)).read().splitlines()
00303             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00304             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00305             a = a_r[:] + 1j*a_i[:]
00306             wfc = np.reshape(a, (xDim,yDim))
00307             conjwfc = np.conj(wfc)
00308
00309             wfc_ypx = np.multiply(ym,np.fft.ifft(np.multiply(pxm,np.fft.fft(wfc,axis=1)),axis=1))
00310             wfc_xpy = np.multiply(xm,np.fft.ifft(np.multiply(pym,np.fft.fft(wfc,axis=0)),axis=0))
00311             result = np.sum( np.sum( np.multiply(conjwfc,wfc_xpy - wfc_ypx) ) ) * dx2
00312         else:
00313             print "Skipped " + dataName + "_" + str(i)
00314             result = np.nan
00315
00316             print i, incr
00317             Lz[(i/incr)] = np.real(result)
00318         type=""
00319         if ev_type == 0:
00320             type = "gnd"
00321         else:
00322             type = "ev"
00323         np.savetxt('Lz.csv',Lz,delimiter=',')
00324
00325         plt.plot(Lz)
00326         plt.savefig("Lz_"+type+".pdf",dpi=imgdpi)
00327         plt.axis('off')
00328         plt.savefig("Lz_"+type+"_axis0.pdf",bbox_inches='tight',dpi=imgdpi)
00329         plt.close()

```

Here is the caller graph for this function:

#### 4.6.2.2 def observables.dens\_struct\_fact( *dataName*, *initValue*, *finalValue*, *incr* )

Definition at line 194 of file [observables.py](#).

References [kinertrum\(\)](#).

Referenced by [expec\\_val\\_\(\)](#).

```

00194
00195 def dens_struct_fact(dataName, initValue, finalValue,incr):
00196     n_k=np.zeros(finalValue/incr)
00197     n_k_t=np.zeros((finalValue/incr,xDim,yDim),dtype=np.complex128)
00198     for i in range(initValue,incr*(finalValue/incr),incr):
00199         if os.path.exists(dataName + '_' + str(i)):
00200             real=open(dataName + '_' + str(i)).read().splitlines()
00201             img=open(dataName + 'i_' + str(i)).read().splitlines()
00202             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00203             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00204             a = a_r[:] + 1j*a_i[:]
00205             n = np.abs(a)**2
00206
00207             kinertrum(np.reshape(a, (xDim,yDim)),dx,i,0)
00208             sf = np.fft.fftfreq(np.fft.fft2(np.reshape(n, (xDim,yDim))))
00209             n_k_t[i/incr][:][:] = sf[:][:];
00210             n_k[i/incr]=(abs(np.sum(np.sum(sf)) * dx**2))
00211
00212             fig, ax = plt.subplots()
00213             f = plt.imshow(np.log10(abs(sf)), cmap=plt.get_cmap('gnuplot2'))
00214             cbar = fig.colorbar(f)
00215             plt.gca().invert_yaxis()
00216             plt.savefig("struct_" + str(i/incr) + ".png",vmin=0,vmax=12,dpi=200)
00217             plt.close()
00218             print i/incr
00219
00220             np.savetxt('Struct' + '.csv',n_k,delimiter=',')
00221             plt.plot(range(initValue,finalValue,incr),n_k)
00222             sp.io.savemat('Struct_t.mat',mdict={'n_k_t',n_k_t})
00223             plt.savefig("Struct.pdf",dpi=200)
00224             plt.close()

```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 4.6.2.3 def observables.energy\_kinetic ( *dataName*, *initValue*, *finalValue*, *increment* )

Definition at line 268 of file [observables.py](#).

Referenced by [expec\\_val\\_\(\)](#).

```

00268
00269 def energy_kinetic(dataName, initValue, finalValue, increment):
00270     px1 = np.fft.fftshift(px)
00271     py1 = np.fft.fftshift(py)
00272     dk=[]
00273     dk2[:]= (px1[:]**2 + py1[:]**2)
00274     Lz = np.zeros( (finalValue/incr))
00275     for i in range(initValue,incr*(finalValue/incr),incr):
00276         if os.path.exists(dataName + '_' + str(i)):
00277             real=open(dataName + '_' + str(i)).read().splitlines()
00278             img=open(dataName + 'i_' + str(i)).read().splitlines()
00279             a_r = numpy.asanyarray(real, dtype='f8') #64-bit double
00280             a_i = numpy.asanyarray(img, dtype='f8') #64-bit double
00281             a = a_r[:] + 1j*a_i[:]
00282             wfcp = np.fft.fft2(np.reshape(a, (xDim,yDim)))
00283             conjwfcp = np.conj(wfcp)
00284             E_k = np.zeros(len(px1))
00285             for ii in range(0,len(px1)):
00286                 E_k[ii] = np.sum( np.sum( np.multiply(wfcp,conjwfcp) ) )*dk2[ii]
00287
00288             np.savetxt('E_k_' + str(i) + '.csv',E_k,delimiter=',')
00289             print i

```

Here is the caller graph for this function:

#### 4.6.2.4 def observables.energy\_potential ( *dataName*, *initValue*, *finalValue*, *increment* )

Definition at line 290 of file [observables.py](#).

```

00290
00291 def energy_potential(dataName, initValue, finalValue, increment):
00292     print 'energy'

```

#### 4.6.2.5 def observables.energy\_total ( *dataName*, *initValue*, *finalValue*, *increment* )

Definition at line 235 of file [observables.py](#).

Referenced by [expec\\_val\\_\(\)](#).

```

00235
00236 def energy_total(dataName, initValue, finalValue, increment):
00237     E=np.zeros((finalValue,1))
00238     E_k=np.zeros((finalValue,1))
00239     E_vi=np.zeros((finalValue,1))
00240     E_l=np.zeros((finalValue,1))
00241     for i in range(initValue,incr*(finalValue/incr),incr):
00242         if os.path.exists(dataName + '_' + str(i)):
00243             real=open(dataName + '_' + str(i)).read().splitlines()
00244             img=open(dataName + 'i_' + str(i)).read().splitlines()
00245             a_r = np.array(real, dtype='f8') #64-bit double
00246             a_i = np.array(img, dtype='f8') #64-bit double
00247             wfcr = np.reshape(a_r[:] + 1j*a_i[:], (xDim,yDim))
00248             wfcp = np.array(np.fft.fft2(wfcr))
00249             wfcr_c = np.conj(wfcr)
00250
00251             E1 = np.fft.ifft2(K*wfcp)
00252             E2 = (V + 0.5*g*np.abs(wfcr)**2)*wfcr
00253             E3 = -(omega*omegaX)*(np.fft.ifft(xPy*np.fft.fft(wfcr,axis=0),axis=0) - np.fft.ifft(yPx*
np.fft.fft(wfcr,axis=1),axis=1) )
00254

```



```

00255         E_k[i/incr] = np.trapz(np.trapz(wfcr_c*E1))*dx*dy
00256         E_vi[i/incr] = np.trapz(np.trapz(wfcr_c*E2))*dx*dy
00257         E_l[i/incr] = np.trapz(np.trapz(wfcr_c*E3))*dx*dy
00258         E[i/incr] = E_k[i/incr] + E_vi[i/incr] + E_l[i/incr]
00259         print (i/float(evMaxVal))
00260     np.savetxt('E_'+ str(i) + '.csv',E,delimiter=',')
00261     np.savetxt('E_k_'+ str(i) + '.csv',E_k,delimiter=',')
00262     np.savetxt('E_vi_'+ str(i) + '.csv',E_vi,delimiter=',')
00263     np.savetxt('E_l_'+ str(i) + '.csv',E_l,delimiter=',')
00264     t = np.array(range(initValue,finalValue,incr))/dt
00265     plt.plot(t,E,'r-',t,E_k,'g-',t,E_vi,'b-',t,E_l,'y-')
00266     plt.savefig("EnergyVst.pdf",dpi=200)
00267     plt.close()

```

Here is the caller graph for this function:

#### 4.6.2.6 def observables.expec\_val\_( quant\_name, quantity, dataName, initValue, finalValue, incr )

Definition at line 382 of file [observables.py](#).

References [ang\\_mom\(\)](#), [dens\\_struct\\_fact\(\)](#), [energy\\_kinetic\(\)](#), [energy\\_total\(\)](#), [expec\\_val\\_monopole\(\)](#), [expec\\_val\\_quadrupole\(\)](#), and [kinertrum\\_loop\(\)](#).

```

00382
00383 def expec_val_(quant_name, quantity, dataName, initValue, finalValue, incr):
00384     x=np.asarray(open('x_0').read().splitlines(),dtype='f8')
00385     y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
00386     # px=open('px_0')
00387     # py=open('py_0')
00388     xm, ym = np.meshgrid(x, y)
00389     result = []
00390     for i in range(initValue,finalValue,incr):
00391         if not os.path.exists(dataName):
00392             real=open(dataName + '_' + str(i)).read().splitlines()
00393             img=open(dataName + 'i_' + str(i)).read().splitlines()
00394             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00395             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00396             a = a_r[:] + 1j*a_i[:]
00397             wfc = np.reshape(a, (xDim,yDim))
00398             conjwfc = np.conj(wfc)
00399
00400             d1 = np.multiply( quantity, wfc )
00401             d2 = np.multiply( conjwfc, d1)
00402             result.append( np.real( np.sum( np.sum( d2 ) ) ) *dx*dx )
00403             print str(100*float(i)/finalValue) + '%'
00404     np.savetxt(quant_name + '.csv',result,delimiter=',')
00405     plt.plot(range(initValue,finalValue,incr),result)
00406     plt.savefig(quant_name + ".pdf",dpi=200)
00407     plt.close()

```

Here is the call graph for this function:

#### 4.6.2.7 def observables.expec\_val\_monopole ( dataName, initValue, finalValue, incr )

Definition at line 330 of file [observables.py](#).

Referenced by [expec\\_val\\_\(\)](#).

```

00330
00331 def expec_val_monopole(dataName, initValue, finalValue, incr):
00332     x=np.asarray(open('x_0').read().splitlines(),dtype='f8')
00333     y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
00334     # px=open('px_0')
00335     # py=open('py_0')
00336     xm, ym = np.meshgrid(x, y)
00337     result = []
00338     for i in range(initValue,finalValue,incr):
00339         if not os.path.exists(dataName):
00340             real=open(dataName + '_' + str(i)).read().splitlines()
00341             img=open(dataName + 'i_' + str(i)).read().splitlines()
00342             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00343             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00344             a = a_r[:] + 1j*a_i[:]

```

```

00345         wfc = np.reshape(a, (xDim,yDim))
00346         conjwfc = np.conj(wfc)
00347
00348         d1 = np.multiply( np.square(xm) + np.square(ym), wfc )
00349         d2 = np.multiply( conjwfc, d1)
00350         result.append( np.real( np.sum( np.sum( d2 ) ) ) *dx*dx )
00351         print str(100*float(i)/finalValue) + '%'
00352         np.savetxt('monopole.csv',result,delimiter=',')
00353         plt.plot(range(initValue,finalValue,incr),result)
00354         plt.savefig("Monopole.png",dpi=200)
00355         plt.close()

```

Here is the caller graph for this function:

#### 4.6.2.8 def observables.expec\_val\_quadrupole( *dataName*, *initValue*, *finalValue*, *incr* )

Definition at line 356 of file [observables.py](#).

Referenced by [expec\\_val\(\)](#).

```

00356
00357 def expec_val_quadrupole(dataName, initValue, finalValue, incr):
00358     x=np.asarray(open('x_0').read().splitlines(),dtype='f8')
00359     y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
00360     # px=open('px_0')
00361     # py=open('py_0')
00362     xm, ym = np.meshgrid(x, y)
00363     result = []
00364     for i in range(initValue,finalValue,incr):
00365         if not os.path.exists(dataName):
00366             real=open(dataName + '_r' + str(i)).read().splitlines()
00367             img=open(dataName + '_i' + str(i)).read().splitlines()
00368             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00369             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00370             a = a_r[:] + 1j*a_i[:]
00371             wfc = np.reshape(a, (xDim,yDim))
00372             conjwfc = np.conj(wfc)
00373
00374             d1 = np.multiply( np.square(xm) - np.square(ym), wfc )
00375             d2 = np.multiply( conjwfc, d1)
00376             result.append( np.real( np.sum( np.sum( d2 ) ) ) *dx*dx )
00377             print str(100*float(i)/finalValue) + '%'
00378             np.savetxt('quadrupole.csv',result,delimiter=',')
00379             plt.plot(range(initValue,finalValue,incr),result)
00380             plt.savefig("Quadrupole.png",dpi=200)
00381             plt.close()

```

Here is the caller graph for this function:

#### 4.6.2.9 def observables.kinertrum( *Psi*, *dx*, *i*, *quOn* )

Kinetic energy spectrum = kinertrum.

Calculates the spectrum for compressible and incompressible kinetic energies.

##### Parameters

<i>Psi</i>	The wavefunction
<i>dx</i>	Increment along x
<i>i</i>	The current step number
<i>quOn</i>	Boolean to turn on quantum kinetic energy spectrum (includes phase term).

Definition at line 104 of file [observables.py](#).

Referenced by [dens\\_struct\\_fact\(\)](#), and [kinertrum\\_loop\(\)](#).

```

00104
00105 def kinertrum(Psi, dx, i, quOn):
00106
00107     kMax = np.max(np.max(kx))
00108     Psi[np.where(Psi==0)] = 1e-100

```

```

00109     n_r = np.abs(Psi)**2
00110     n_r[np.where(n_r==0)] = 1e-100
00111     cPsi = np.conj(Psi)
00112     phi = np.angle(Psi)
00113
00114     ph1 = np.unwrap(phi, axis=0)
00115     ph2 = np.unwrap(phi, axis=1)
00116
00117     vel_ph1_x, vel_ph1_y = np.gradient(ph1,dx,dy)
00118     vel_ph2_x, vel_ph2_y = np.gradient(ph2,dx,dy)
00119
00120     v_x = (hbar/m)*vel_ph1_x;
00121     v_y = (hbar/m)*vel_ph2_y;
00122     v_x[np.where(v_x==0)] = 1e-100
00123     v_y[np.where(v_y==0)] = 1e-100
00124
00125     u_x = np.multiply(np.abs(Psi),v_x)
00126     u_y = np.multiply(np.abs(Psi),v_y)
00127
00128     if quOn:
00129         u_x = np.multiply(u_x,np.exp(1j*np.angle(Psi)))
00130         u_y = np.multiply(u_y,np.exp(1j*np.angle(Psi)))
00131
00132     u_kx = np.fft.fftn(u_x)
00133     u_ky = np.fft.fftn(u_y)
00134
00135     uc_kx = ( kxm**2*u_kx + kxm*kym*u_ky ) / ( km_mag**2 + 1e-100 )
00136     uc_ky = ( kym*kxm*u_kx + kym**2*u_ky ) / ( km_mag**2 + 1e-100 )
00137
00138     ui_kx = u_kx - uc_kx
00139     ui_ky = u_ky - uc_ky
00140
00141     uc_x = np.fft.ifftn(uc_kx)
00142     uc_y = np.fft.ifftn(uc_ky)
00143     ui_x = np.fft.ifftn(ui_kx)
00144     ui_y = np.fft.ifftn(ui_ky)
00145
00146     Ec = 0.5*np.abs(np.square(uc_x) + np.square(uc_y))
00147     Ei = 0.5*np.abs(np.square(ui_x) + np.square(ui_y))
00148
00149     fig, ax = plt.subplots()
00150     f = plt.imshow((Ec),cmap=plt.get_cmap('gnuplot2'))
00151     cbar = fig.colorbar(f)
00152     plt.gca().invert_yaxis()
00153     plt.savefig("Ec_" + str(i/incr) + ".png",dpi=200)
00154     plt.close()
00155     fig, ax = plt.subplots()
00156     f = plt.imshow((Ei),cmap=plt.get_cmap('gnuplot2'))
00157     cbar = fig.colorbar(f)
00158     plt.gca().invert_yaxis()
00159     plt.savefig("Ei_" + str(i/incr) + ".png",dpi=200)
00160     plt.close()
00161
00162     print Ec
00163     #exit()
00164     ekc = np.zeros((xDim/2-1,1))
00165     eki = np.zeros((xDim/2-1,1))
00166     for i1 in np.arange(0,np.size(k_mag)/2 -2):
00167         iX = np.array(np.where(np.logical_and( k_mag[i1] >= km_mag, k_mag[i1+1] < km_mag )))
00168         # Ei_kx = np.sum(np.sum(np.abs(ui_kx[iX]**2*k[iX])))
00169         # Ei_ky = np.sum(np.sum(np.abs(ui_ky[iX]**2*k[iX])))
00170         ekc[i1] = (0.5*m*k_mag[i1]) * (np.sum(np.abs(uc_kx[iX]**2 + uc_ky[iX]**2))/len(iX)
00171         eki[i1] = (0.5*m*k_mag[i1]) * (np.sum(np.abs(ui_kx[iX]**2 + ui_ky[iX]**2))/len(iX)
00172     print i1
00173     np.savetxt('ekc_' + str(i) + '.csv',ekc,delimiter=',')
00174     np.savetxt('eki_' + str(i) + '.csv',eki,delimiter=',')
00175     fig, ax = plt.subplots()
00176     print eki[0:(xDim/2-2)]
00177     f = plt.loglog(np.ravel(k_mag[0:(xDim/2 -2)]),eki[0:(xDim/2-2)])
00178     plt.savefig("eki_" + str(i) + ".png",dpi=200)
00179     f = plt.loglog(np.ravel(k_mag[0:(xDim/2 -2)]),np.ravel(ekc[0:(xDim/2-2)]))
00180     plt.savefig("ekc_" + str(i) + ".png",dpi=200)
00181     plt.close()
00182

```

Here is the caller graph for this function:

#### 4.6.2.10 def observables.kinertrum\_loop( *dataName*, *initValue*, *finalValue*, *incr* )

Definition at line 183 of file [observables.py](#).

References [kinertrum\(\)](#).

Referenced by [expec\\_val\\_\(\)](#).

```

00183
00184 def kinertrum_loop(dataName, initValue, finalValue, incr):
00185     for i in range(initValue,incr*(finalValue/incr),incr):
00186         if os.path.exists(dataName + '_' + str(i)):
00187             real=open(dataName + '_' + str(i)).read().splitlines()
00188             img=open(dataName + 'i_' + str(i)).read().splitlines()
00189             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00190             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00191             a = a_r[:] + 1j*a_i[:]
00192
00193             kinertrum(np.reshape(a, (xDim,yDim)), dx,i,1)

```

Here is the call graph for this function:

Here is the caller graph for this function:

### 4.6.3 Variable Documentation

#### 4.6.3.1 tuple observables.c = ConfigParser.ConfigParser()

Definition at line 55 of file [observables.py](#).

#### 4.6.3.2 tuple observables.data = numpy.ndarray(shape=(xDim,yDim))

Definition at line 82 of file [observables.py](#).

#### 4.6.3.3 tuple observables.dkx = (c.getfloat('Params','dpx'))

Definition at line 66 of file [observables.py](#).

#### 4.6.3.4 tuple observables.dky = (c.getfloat('Params','dpy'))

Definition at line 67 of file [observables.py](#).

#### 4.6.3.5 tuple observables.dt = (c.getfloat('Params','dt'))

Definition at line 68 of file [observables.py](#).

#### 4.6.3.6 tuple observables.dx = (c.getfloat('Params','dx'))

Definition at line 64 of file [observables.py](#).

#### 4.6.3.7 tuple observables.dy = (c.getfloat('Params','dy'))

Definition at line 65 of file [observables.py](#).

#### 4.6.3.8 tuple observables.evMaxVal = int(c.getfloat('Params','esteps'))

Definition at line 61 of file [observables.py](#).

4.6.3.9 tuple `observables.g = (0.5*N)`

Definition at line 233 of file [observables.py](#).

Referenced by [Tracker.findVortex\(\)](#).

4.6.3.10 tuple `observables.gndMaxVal = int(c.getfloat('Params','gsteps'))`

Definition at line 60 of file [observables.py](#).

4.6.3.11 float `observables.HBAR = 1.05457148e-34`

Definition at line 51 of file [observables.py](#).

4.6.3.12 float `observables.hbar = 1.05457e-34`

Definition at line 96 of file [observables.py](#).

4.6.3.13 tuple `observables.incr = int(c.getfloat('Params','print_out'))`

Definition at line 62 of file [observables.py](#).

4.6.3.14 tuple `observables.K = np.array(open('K_0').read().splitlines(),dtype='f8')`

Definition at line 227 of file [observables.py](#).

4.6.3.15 tuple `observables.k_mag = np.sqrt( kx**2 + ky**2 )`

Definition at line 93 of file [observables.py](#).

Referenced by [optLatSetup\(\)](#).

4.6.3.16 tuple `observables.km_mag = np.sqrt( kxm**2 + kym**2 )`

Definition at line 92 of file [observables.py](#).

4.6.3.17 tuple `observables.kMax = max(max(k_mag))`

Definition at line 94 of file [observables.py](#).

4.6.3.18 tuple `observables.kx = np.reshape( np.array( [np.linspace( 0, (xDim/2-1)*dkx, xDim/2), np.linspace( (-xDim/2-1)*dkx, -dkx, xDim/2)] ), (xDim,1) )`

Definition at line 89 of file [observables.py](#).

4.6.3.19 tuple `observables.ky = np.reshape( np.array( [np.linspace( 0, (yDim/2-1)*dky, yDim/2), np.linspace( (-yDim/2-1)*dky, -dky, yDim/2)] ), (yDim,1) )`

Definition at line 90 of file [observables.py](#).

4.6.3.20 `float observables.m = 1.4431607e-25`

Definition at line 97 of file [observables.py](#).

4.6.3.21 `tuple observables.mass = (c.getfloat('Params','Mass'))`

Definition at line 72 of file [observables.py](#).

4.6.3.22 `tuple observables.N = int(c.getfloat('Params','atoms'))`

Definition at line 80 of file [observables.py](#).

4.6.3.23 `tuple observables.num_vort = int(c.getfloat('Params','Num_vort'))`

Definition at line 77 of file [observables.py](#).

4.6.3.24 `tuple observables.omega = (c.getfloat('Params','omega'))`

Definition at line 73 of file [observables.py](#).

4.6.3.25 `tuple observables.omegaX = (c.getfloat('Params','omegaX'))`

Definition at line 74 of file [observables.py](#).

4.6.3.26 `tuple observables.omegaZ = (c.getfloat('Params','omegaZ'))`

Definition at line 71 of file [observables.py](#).

4.6.3.27 `float observables.PI = 3.141592653589793`

Definition at line 52 of file [observables.py](#).

4.6.3.28 `tuple observables.V = np.array(open('V_0').read().splitlines(),dtype='f8')`

Definition at line 225 of file [observables.py](#).

4.6.3.29 `tuple observables.x = np.asarray(open('x_0').read().splitlines(),dtype='f8')`

Definition at line 84 of file [observables.py](#).

4.6.3.30 `tuple observables.xDim = int(c.getfloat('Params','xDim'))`

Definition at line 58 of file [observables.py](#).

4.6.3.31 `tuple observables.xMax = (c.getfloat('Params','xMax'))`

Definition at line 69 of file [observables.py](#).

4.6.3.32 tuple `observables.xPy` = `np.array(open('xPy_0').read().splitlines(),dtype='f8')`

Definition at line 229 of file `observables.py`.

4.6.3.33 tuple `observables.y` = `np.asarray(open('y_0').read().splitlines(),dtype='f8')`

Definition at line 85 of file `observables.py`.

4.6.3.34 tuple `observables.yDim` = `int(c.getfloat('Params','yDim'))`

Definition at line 59 of file `observables.py`.

4.6.3.35 tuple `observables.yMax` = `(c.getfloat('Params','yMax'))`

Definition at line 70 of file `observables.py`.

4.6.3.36 tuple `observables.yPx` = `np.array(open('yPx_0').read().splitlines(),dtype='f8')`

Definition at line 231 of file `observables.py`.

## 4.7 stats Namespace Reference

### Functions

- def `lsFit`

### Variables

- tuple `c` = `ConfigParser.ConfigParser()`
- tuple `incr` = `int(c.getfloat('Params','print_out'))`
- tuple `xDim` = `int(c.getfloat('Params','xDim'))`
- tuple `yDim` = `int(c.getfloat('Params','yDim'))`

### 4.7.1 Detailed Description

stats.py - GPUE: Split Operator based GPU solver for Nonlinear Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 4.7.2 Function Documentation

### 4.7.2.1 `def stats.lsFit( start, end, incr )`

Definition at line 57 of file [stats.py](#).

Referenced by [vis.overlap\(\)](#).

```

00057
00058 def lsFit(start,end,incr):
00059     L = np.matrix([
00060         [0,0,1],
00061         [1,0,1],
00062         [0,1,1],
00063         [1,1,1]
00064     ])
00065     LSQ = np.linalg.inv(np.transpose(L)*L)*np.transpose(L)
00066     for i in range(start,end,incr):
00067         v_arr=genfromtxt('vort_arr_' + str(i),delimiter=',')
00068         real=open('wfc_ev_' + str(i)).read().splitlines()
00069         img=open('wfc_evi_' + str(i)).read().splitlines()
00070         a_r = np.asanyarray(real,dtype='f8') #64-bit double
00071         a_i = np.asanyarray(img,dtype='f8') #64-bit double
00072         a = a_r[:] + 1j*a_i[:]
00073         wfc = (np.reshape(a, (xDim,yDim)))
00074
00075         indX = [row[0] for row in v_arr]
00076         indY = [row[1] for row in v_arr]
00077         wind = [row[2] for row in v_arr]
00078         sign = [row[3] for row in v_arr]
00079         data=[]
00080         for ii in range(0,len(indX)):
00081             p=np.matrix([[0],[0],[0],[0]],dtype=np.complex)
00082             p[0]=(wfc[indX[ii], indY[ii]])
00083             p[1]=(wfc[indX[ii]+1, indY[ii]])
00084             p[2]=(wfc[indX[ii], indY[ii]+1])
00085             p[3]=(wfc[indX[ii]+1, indY[ii]+1])
00086             rc = LSQ * np.real(p)
00087             ic = LSQ * np.imag(p)
00088
00089             A=np.squeeze([row[0:2] for row in [rc,ic]])
00090             B=-np.squeeze([row[2] for row in [rc,ic]])
00091             r=np.linalg.lstsq(A,B)[0]
00092             data.append([indX[ii]+r[0],indY[ii]+r[1],sign[ii]])
00093
00094 #         f = plt.imshow(abs(wfc)**2)
00095 #         plt.jet()
00096 #         plt.gca().invert_yaxis()
00097 #         plt.hold(True)
00098 #         X = [row[0] for row in data]
00099 #         Y = [row[1] for row in data]
00100 #         plt.scatter(Y,X,s=0.2,marker='.',c='red',lw=0)
00101 #         plt.scatter(indY,indX,s=0.2,marker='.',c='yellow',lw=0)
00102 #         plt.savefig("fig.png",dpi=1200)
00103 #         plt.close()
00104         np.savetxt('vort_lsq_' + str(i) + '.csv',data,delimiter=',')

```

Here is the caller graph for this function:

## 4.7.3 Variable Documentation

### 4.7.3.1 `tuple stats.c = ConfigParser.ConfigParser()`

Definition at line 50 of file [stats.py](#).



4.7.3.2 `tuple stats.incr = int(c.getfloat('Params','print_out'))`

Definition at line 53 of file [stats.py](#).

4.7.3.3 `tuple stats.xDim = int(c.getfloat('Params','xDim'))`

Definition at line 54 of file [stats.py](#).

4.7.3.4 `tuple stats.yDim = int(c.getfloat('Params','yDim'))`

Definition at line 55 of file [stats.py](#).

## 4.8 Tracker Namespace Reference

See the source file for info on functions.

### Classes

- struct [Vortex](#)  
*[Vortex](#) is used to track specific individual vortices. [More...](#)*

### Functions

- int [findOLMaxima](#) (int \*marker, double \*V, double radius, int xDim, double \*x)  
*Finds the maxima of the optical lattice.*
- int [findVortex](#) (int \*marker, double2 \*wfc, double radius, int xDim, double \*x, int timestep)  
*Phase winding method to determine vortex positions.*
- void [olPos](#) (int \*marker, int2 \*olLocation, int xDim)  
*Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.*
- int [phaseTest](#) (int2 vLoc, double2 \*wfc, int xDim)  
*Tests the phase winding of the wavefunction, looking for vortices.*
- double [sigVOL](#) (int2 \*vArr, int2 \*opLatt, double \*x, int numVort)
- double [sigVOL](#) (struct [Tracker::Vortex](#) \*vArr, int2 \*opLatt, double \*x, int numVort)  
*Sigma of vortex lattice and optical lattice.*
- double [vortAngle](#) (struct [Tracker::Vortex](#) \*vortCoords, struct [Vortex](#) central, int numVort)  
*Determines the angle of the vortex lattice relative to the x-axis.*
- void [vortArrange](#) (struct [Tracker::Vortex](#) \*vCoordsC, struct [Vortex](#) \*vCoordsP, int length)  
*Ensures the vortices are tracked and arranged in the right order based on minimum distance between previous and current positions.*
- struct [Vortex](#) [vortCentre](#) (struct [Tracker::Vortex](#) \*cArray, int length, int xDim)  
*Determines the coords of the vortex closest to the central position.*
- void [vortPos](#) (int \*marker, struct [Tracker::Vortex](#) \*vLocation, int xDim, double2 \*wfc)  
*Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.*
- struct [Vortex](#) \* [vortPosDelta](#) (int \*cMarker, int2 \*pMarker, double \*x, double tolerance, int numVortices, int xDim)
- double [vortSepAvg](#) (struct [Vortex](#) \*vArray, struct [Tracker::Vortex](#) centre, int length)  
*Determines the vortex separation at the centre of the lattice.*

## Variables

- char [bufferT](#) [1024]

### 4.8.1 Detailed Description

See the source file for info on functions. Contains all the glorious info you need to track vortices and see what they are up to.

### 4.8.2 Class Documentation

#### 4.8.2.1 struct Tracker::Vortex

[Vortex](#) is used to track specific individual vortices.

coords tracks x,y positions. sign indicates direction of vortex rotation. wind indicates the unit charge of the vortex.

Definition at line 54 of file [tracker.h](#).

Collaboration diagram for Tracker::Vortex:

#### Class Members

int2	coords	
int	sign	
int	wind	

### 4.8.3 Function Documentation

#### 4.8.3.1 int Tracker::findOLMaxima ( int \* marker, double \* Vopt, double radius, int xDim, double \* x )

Finds the maxima of the optical lattice.

Deprecated.

Definition at line 67 of file [tracker.cc](#).

References [vis::i](#), and [Minions::maxValue\(\)](#).

```

00067                                     {
00068         double gridValues[9];
00069         int2 mIndex[1024];
00070         int2 index;
00071         int i,j,found;
00072         found=0;
00073         for (i=1; i<xDim-1; ++i ){
00074             for(j=1; j<xDim-1;++j){
00075                 if(sqrt(x[i]*x[i] + x[j]*x[j]) < radius){
00076                     gridValues[0] = Vopt[(i-1)*xDim + (j-1)];
00077                     gridValues[1] = Vopt[(i-1)*xDim + j];
00078                     gridValues[2] = Vopt[(i-1)*xDim + (j+1)];
00079                     gridValues[3] = Vopt[i*xDim + (j-1)];
00080                     gridValues[4] = Vopt[i*xDim + j];
00081                     gridValues[5] = Vopt[i*xDim + (j+1)];
00082                     gridValues[6] = Vopt[(i+1)*xDim + (j-1)];
00083                     gridValues[7] = Vopt[(i+1)*xDim + j];
00084                     gridValues[8] = Vopt[(i+1)*xDim + (j+1)];
00085                     if(fabs((gridValues[4]-Minions::maxValue(gridValues,9))/gridValues[4])
00086 <= 1e-7){
00087                         //printf ("%d,%d\n",i,j);
00088                         (marker)[i*xDim + j] = 1;
00089                         index.x=i;
00089                         index.y=j;
00090                         mIndex[found] = index;
00091                         ++found;
00092                     }
00093                 }
00094             }
00095         }

```

```

00096         return found;
00097     }

```

Here is the call graph for this function:

#### 4.8.3.2 int Tracker::findVortex ( int \* marker, double2 \* wfc, double radius, int xDim, double \* x, int timestep )

Phase winding method to determine vortex positions.

Calculates the phase around a loop and checks if  $\sim \pm 2\pi$ .

Definition at line 136 of file [tracker.cc](#).

References [Minions::complexDiv\(\)](#), [Minions::complexMag\(\)](#), [Minions::complexScale\(\)](#), [observables::g](#), [vis::i](#), [PI](#), and [y](#).

Referenced by [evolve\(\)](#).

```

00136                                     {
00137         double2 *g = (double2*) malloc(sizeof(double2)*4);
00138         double *phiDelta = (double*) malloc(sizeof(double)*4);
00139         int i,j,found;
00140         int cond_x, cond_y;
00141         cond_x = 0; cond_y = 0;
00142         found = 0;
00143         long rnd_value = 0;
00144         double sum = 0.0;
00145         for ( i=0; i < xDim-1; ++i ){
00146             for ( j=0; j < xDim-1; ++j ){
00147                 if(sqrt(x[i]*x[i] + x[j]*x[j]) < radius){
00148                     g[0] = Minions::complexScale(
Minions::complexDiv( wfc[i*xDim + j], wfc[(i+1)*xDim + j] ), (
Minions::complexMag( wfc[(i+1)*xDim + j]) /
Minions::complexMag( wfc[i*xDim + j] ))) );
00149                     g[1] = Minions::complexScale(
Minions::complexDiv( wfc[(i+1)*xDim + j], wfc[(i+1)*xDim + (j+1)] ), (
Minions::complexMag( wfc[(i+1)*xDim + (j+1)]) /
Minions::complexMag( wfc[(i+1)*xDim + j] ))) );
00150                     g[2] = Minions::complexScale(
Minions::complexDiv( wfc[(i+1)*xDim + (j+1)], wfc[i*xDim + (j+1)] ), (
Minions::complexMag( wfc[i*xDim + (j+1)]) /
Minions::complexMag( wfc[(i+1)*xDim + (j+1)] ))) );
00151                     g[3] = Minions::complexScale(
Minions::complexDiv( wfc[i*xDim + (j+1)], wfc[i*xDim + j] ), (
Minions::complexMag( wfc[i*xDim + j]) /
Minions::complexMag( wfc[i*xDim + (j+1)] ))) );
00152
00153                     for (int k=0; k<4; ++k){
00154                         phiDelta[k] = atan2( g[k].y, g[k].x );
00155                         if(phiDelta[k] <= -PI){
00156                             phiDelta[k] += 2*PI;
00157                         }
00158                     }
00159                     sum = phiDelta[0] + phiDelta[1] + phiDelta[2] + phiDelta[3];
00160                     rnd_value = lround(sum/(2*PI));
00161                     if( sum >= 1.9*PI && cond_x <= 0 && cond_y <= 0 ){
00162                         marker[i*xDim + j] = rnd_value;
00163                         ++found;
00164                         sum = 0.0;
00165                         cond_x = 2; cond_y = 2;
00166                     }
00167                     else if( sum <= -1.9*PI && cond_x <= 0 && cond_y <= 0 ) {
00168                         marker[i*xDim + j] = -rnd_value;
00169                         ++found;
00170                         sum = 0.0;
00171                         cond_x = 2; cond_y = 2;
00172                     }
00173                 }
00174             }
00175             --cond_x;
00176             --cond_y;
00177         }
00178     }
00179     return found;
00180 }

```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 4.8.3.3 void Tracker::olPos ( int \* marker, int2 \* vLocation, int xDim )

Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.

Definition at line 186 of file [tracker.cc](#).

References [vort::counter](#), [vis::i](#), [xDim](#), and [y](#).

```
00186                                     {
00187     int i, j;
00188     unsigned int counter=0;
00189     for(i=0; i<xDim; ++i){
00190         for(j=0; j<xDim; ++j){
00191             if((marker)[i*xDim + j] == 1){
00192                 (olLocation)[ counter ].x=i;
00193                 (olLocation)[ counter ].y=j;
00194                 ++counter;
00195             }
00196         }
00197     }
00198 }
```

#### 4.8.3.4 int Tracker::phaseTest ( int2 vLoc, double2 \* wfc, int xDim )

Tests the phase winding of the wavefunction, looking for vortices.

Definition at line 203 of file [tracker.cc](#).

References [Minions::complexDiv\(\)](#), [Minions::complexMag\(\)](#), [Minions::complexScale\(\)](#), [PI](#), [result](#), and [y](#).

```
00203                                     {
00204     int result = 0;
00205     double2 gridValues[4];
00206     double phiDelta[4];
00207     double sum=0.0;
00208     int i=vLoc.x, j=vLoc.y;
00209     gridValues[0] = Minions::complexScale(
00210         Minions::complexDiv(wfc[i*xDim + j], wfc[(i+1)*xDim + j]), (
00211         Minions::complexMag(wfc[(i+1)*xDim + j]) /
00212         Minions::complexMag(wfc[i*xDim + j]));
00213     gridValues[1] = Minions::complexScale(
00214         Minions::complexDiv(wfc[(i+1)*xDim + j], wfc[(i+1)*
00215         xDim + (j+1)]), (Minions::complexMag(wfc[(i+1)*
00216         xDim + (j+1)]) / Minions::complexMag(wfc[(i+1)*xDim + j]));
00217     gridValues[2] = Minions::complexScale(
00218         Minions::complexDiv(wfc[(i+1)*xDim + (j+1)], wfc[i*
00219         xDim + (j+1)]), (Minions::complexMag(wfc[i*xDim + (j+1)]) /
00220         Minions::complexMag(wfc[(i+1)*xDim + (j+1)]));
00221     gridValues[3] = Minions::complexScale(
00222         Minions::complexDiv(wfc[i*xDim + (j+1)], wfc[i*xDim + j]), (
00223         Minions::complexMag(wfc[i*xDim + j]) /
00224         Minions::complexMag(wfc[i*xDim + (j+1)]));
00225     for (int k=0; k<4; ++k){
00226         phiDelta[k] = atan2(gridValues[k].y, gridValues[k].x);
00227         if(phiDelta[k] <= -PI){
00228             phiDelta[k] += 2*PI;
00229         }
00230     }
00231     sum = phiDelta[0] + phiDelta[1] + phiDelta[2] + phiDelta[3];
00232     if(sum >= 1.8*PI){
00233         result = 1;
00234     }
00235     return result;
00236 }
```

Here is the call graph for this function:

#### 4.8.3.5 double Tracker::sigVOL ( int2 \* vArr, int2 \* opLatt, double \* x, int numVort )

#### 4.8.3.6 double Tracker::sigVOL ( struct Tracker::Vortex \* vArr, int2 \* opLatt, double \* x, int numVort )

Sigma of vortex lattice and optical lattice.

Definition at line 312 of file [tracker.cc](#).

References [Tracker::Vortex::coords](#), [dx](#), and [vis::i](#).

```

00312                                     {
00313         double sigma = 0.0;
00314         double dx = abs(x[1]-x[0]);
00315         for (int i=0; i<numVort; ++i){
00316             sigma += pow( abs( sqrt( (vArr[i].coords.x - opLatt[i].x)*(vArr[i].
coords.x - opLatt[i].x) + (vArr[i].coords.y - opLatt[i].y)*(vArr[i].
coords.y - opLatt[i].y) ) * dx ), 2 );
00317         }
00318         sigma /= numVort;
00319         return sigma;
00320     }

```

#### 4.8.3.7 double Tracker::vortAngle ( struct Vortex \* vortCoords, struct Vortex central, int numVort )

Determines the angle of the vortex lattice relative to the x-axis.

Definition at line 291 of file [tracker.cc](#).

References [Tracker::Vortex::coords](#), [vis::i](#), and [PI](#).

Referenced by [evolve\(\)](#).

```

00291                                     {
00292         int location;
00293         double sign=1.0;
00294         double minVal=1e300; //(pow(central.x - vortCoords[0].x,2) + pow(central.y - vortCoords[0].y,2));
00295         for (int i=0; i < numVort; ++i){
00296             if (minVal > (pow(central.coords.x - vortCoords[i].coords.x,2) + pow(central.coords.y -
vortCoords[i].coords.y,2)) && abs(central.coords.x - vortCoords[i].coords.x) > 2e-6 && abs(central.coords.y -
vortCoords[i].coords.y) > 2e-6){
00297                 minVal = (pow(central.coords.x - vortCoords[i].coords.x,2) + pow(central.coords.y -
vortCoords[i].coords.y,2));
00298                 location = i;
00299             }
00300         }
00301         double ang=fmod(atan2( (vortCoords[location].coords.y - central.coords.y), (vortCoords[
location].coords.x - central.coords.x) ),PI/3));
00302         printf("Angle=%e\n",ang);
00303         return PI/3 - ang;
00304
00305         //return PI/2 + fmod(atan2(vortCoords[location].y-central.y, vortCoords[location].x - central.x),
PI/3);
00306         //return PI/2 - sign*acos( ( (central.x - vortCoords[location].x)*(central.x -
vortCoords[location].x) ) / ( minVal*(central.x - vortCoords[location].x) ) );
00307     }

```

Here is the caller graph for this function:

#### 4.8.3.8 void Tracker::vortArrange ( struct Vortex \* vCoordsC, struct Vortex \* vCoordsP, int length )

Ensures the vortices are tracked and arranged in the right order based on minimum distance between previous and current positions.

Definition at line 249 of file [tracker.cc](#).

References [Tracker::Vortex::coords](#), [Minions::coordSwap\(\)](#), [vort::dist\(\)](#), and [vis::i](#).

Referenced by [evolve\(\)](#).

```

00249                                     {
00250         int dist, dist_t;
00251         int i, j, index;
00252         for ( i = 0; i < length; ++i ){
00253             dist = 0x7FFFFFFF; //arbitrary big value
00254             index = i;
00255             for ( j = i; j < length ; ++j){
00256                 dist_t = ( (vCoordsP[i].coords.x - vCoordsC[j].coords.x)*(vCoordsP[i].
coords.x - vCoordsC[j].coords.x) + (vCoordsP[i].coords.y - vCoordsC[j].coords.y)*(vCoordsP[i].
coords.y - vCoordsC[j].coords.y) );
00257                 if(dist > dist_t ){

```

```

00258             dist = dist_t;
00259             index = j;
00260         }
00261     }
00262     Minions::coordSwap(vCoordsC,index,i);
00263 }
00264 }

```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 4.8.3.9 struct Vortex Tracker::vortCentre ( struct Tracker::Vortex \* cArray, int length, int xDim )

Determines the coords of the vortex closest to the central position.

Useful for centering the optical lattice over v. lattice\*

Definition at line 269 of file [tracker.cc](#).

References [Tracker::Vortex::coords](#), [vort::counter](#), and [vis::i](#).

Referenced by [evolve\(\)](#).

```

00269                                     {
00270         int i, j, counter=0;
00271         int valX, valY;
00272         double valueTest, value = 0.0;
00273         valX = (cArray)[0].coords.x - ((xDim/2)-1);
00274         valY = (cArray)[0].coords.y - ((xDim/2)-1);
00275         value = sqrt( valX*valX + valY*valY );//Calcs the sqrt(x^2+y^2) from central position. try to
00276         minimise this value
00277         for ( i=1; i<length; ++i ){
00278             valX = (cArray)[i].coords.x - ((xDim/2)-1);
00279             valY = (cArray)[i].coords.y - ((xDim/2)-1);
00280             valueTest = sqrt(valX*valX + valY*valY);
00281             if(value > valueTest){
00282                 value = valueTest;
00283                 counter = i;
00284             }
00285         }
00286         return (cArray)[counter];
00287     }

```

Here is the caller graph for this function:

#### 4.8.3.10 void Tracker::vortPos ( int \* marker, struct Vortex \* vLocation, int xDim, double2 \* wfc )

Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.

Definition at line 230 of file [tracker.cc](#).

References [vort::counter](#), [vis::i](#), and [xDim](#).

Referenced by [evolve\(\)](#).

```

00230                                     {
00231         int i, j;
00232         unsigned int counter=0;
00233         for(i=0; i<xDim; ++i){
00234             for(j=0; j<xDim; ++j){
00235                 if( abs((marker)[i*xDim + j]) >= 1){
00236                     (vLocation)[ counter ].coords.x=i;
00237                     (vLocation)[ counter ].coords.y=j;
00238                     (vLocation)[ counter ].sign = ( signbit(abs(marker[i*xDim + j])) == 0 ) ? 1 : -1;
00239                     (vLocation)[ counter ].wind = abs(marker[i*xDim + j]);
00240                     ++counter;
00241                 }
00242             }
00243         }
00244     }

```

Here is the caller graph for this function:

**4.8.3.11** `struct Vortex* Tracker::vortPosDelta ( int * cMarker, int2 * pMarker, double * x, double tolerance, int numVortices, int xDim )`

**4.8.3.12** `double Tracker::vortSepAvg ( struct Vortex * vArray, struct Vortex centre, int length )`

Determines the vortex separation at the centre of the lattice.

Definition at line 48 of file [tracker.cc](#).

References [Tracker::Vortex::coords](#), and [result](#).

Referenced by [evolve\(\)](#), and [optLatSetup\(\)](#).

```

00048
00049     double result=0.0; // = sqrt( pow(centre.x - v_array[0].x,2) + pow(centre.y -
    v_array[0].y,2));
00050     double min = 0.0;
00051     double min_tmp = 0.0;
00052     int index=0;
00053     min = sqrt( pow(centre.coords.x - vArray[0].coords.x,2) + pow(centre.coords.y - vArray[0].coords.y,
    2));
00054     for (int j=1; j<length; ++j){
00055         min_tmp = sqrt( pow(centre.coords.x - vArray[j].coords.x,2) + pow(centre.coords.y - vArray[j].
    coords.y,2));
00056         if(min > min_tmp && min_tmp > 1e-7){
00057             min = min_tmp;
00058             index = j;
00059         }
00060     }
00061     return min;
00062 }
```

Here is the caller graph for this function:

## 4.8.4 Variable Documentation

**4.8.4.1** `char Tracker::bufferT[1024]`

Definition at line 43 of file [tracker.cc](#).

## 4.9 vis Namespace Reference

### Functions

- def [delaunay](#)
- def [hist\\_gen](#)
- def [image\\_gen](#)
- def [image\\_gen\\_single](#)
- def [laplacian](#)
- def [opPot](#)
- def [overlap](#)
- def [scaleAxis](#)
- def [struct\\_fact](#)
- def [voronoi](#)
- def [vort\\_traj](#)

### Variables

- tuple `c` = `ConfigParser.ConfigParser()`
- list `CPUs` = `os.environ['SLURM_JOB_CPUS_PER_NODE']`
- tuple `data` = `numpy.ndarray(shape=(xDim,yDim))`

- tuple `dt` = (c.getfloat('Params','dt'))
- tuple `dx` = (c.getfloat('Params','dx'))
- list `ev_proc` = []
- list `evImgList` = []
- tuple `evMaxVal` = int(c.getfloat('Params','esteps'))
- list `gnd_proc` = []
- list `gndImgList` = []
- tuple `gndMaxVal` = int(c.getfloat('Params','gsteps'))
- tuple `i` = gndImgList.pop()
- tuple `incr` = int(c.getfloat('Params','print\_out'))
- int `num_vort` = 0
- tuple `p` = proc.pop()
- `proc` = `gnd_proc+ev_proc`
- tuple `sep` = (c.getfloat('Params','dx'))
- tuple `xDim` = int(c.getfloat('Params','xDim'))
- tuple `xMax` = (c.getfloat('Params','xMax'))
- tuple `yDim` = int(c.getfloat('Params','yDim'))
- tuple `yMax` = (c.getfloat('Params','yMax'))

#### 4.9.1 Detailed Description

vis.py - GPUE: Split Operator based GPU solver for Nonlinear Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

#### 4.9.2 Function Documentation

##### 4.9.2.1 `def vis.delaunay ( dataName, dataType, value )`

Definition at line 78 of file `vis.py`.

Referenced by `overlap()`.



```

00078
00079 def delaunay(dataName,dataType,value):
00080     v_arr=genfromtxt(dataName + str(value) + dataType,delimiter=',')
00081     data = np.array([[row[0],row[1]] for row in v_arr])
00082     dln = sp.spatial.Delaunay(data)
00083     plt.triplot(data[:,0],data[:,1],dln.simplices.copy(),linewidth=0.5,color='b',marker='.')
00084     plt.xlim(300,700);plt.ylim(300,700);
00085     plt.savefig('delaun_' + str(value) + '.png',dpi=200)
00086     print 'Saved Delaunay @ t=' + str(value)

```

Here is the caller graph for this function:

#### 4.9.2.2 def vis.hist\_gen ( name, value, num\_bins )

Definition at line 133 of file vis.py.

```

00133
00134 def hist_gen(name,value,num_bins):
00135     v_arr=genfromtxt('vort_arr_' + str(value),delimiter=',')
00136     H=[]
00137     count=0
00138
00139     for i1 in range(0,v_arr.size/2):
00140         for i2 in range(i1,v_arr.size/2):
00141             H.append(m.sqrt( abs(v_arr[i1][0]*sep - v_arr[i2][0]*sep)**2 + abs(v_arr[i1][1]*sep - v_arr[
00142 i2][1]*sep)**2 ))
00143             count = count + 1
00144             plt.title('Vortex lattice @ t=' + str(value*dt))
00145             plt.ticklabel_format(style='scientific')
00146             plt.ticklabel_format(style='scientific',axis='x', scilimits=(0,0))
00147             h = plt.hist(H, bins=num_bins)
00148             plt.savefig(name + "_" + str(value) + ".pdf")
00149             plt.close()

```

#### 4.9.2.3 def vis.image\_gen ( dataName, initValue, finalValue, increment, imgdpi )

Definition at line 149 of file vis.py.

```

00149
00150 def image_gen(dataName, initValue, finalValue, increment,imgdpi):
00151     for i in range(initValue,finalValue,increment):
00152         if not os.path.exists(dataName+"r_"+str(i)+"_abspsi2.png"):
00153             real=open(dataName + '_' + str(i)).read().splitlines()
00154             img=open(dataName + 'i_' + str(i)).read().splitlines()
00155             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00156             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00157             a = a_r[:] + 1j*a_i[:]
00158             b = np.reshape(a, (xDim,yDim))
00159             f = plt.imshow(abs(b)**2)
00160             plt.jet()
00161             plt.gca().invert_yaxis()
00162             plt.savefig(dataName+"r_"+str(i)+"_abspsi2.png",dpi=imgdpi)
00163             plt.close()
00164             g = plt.imshow(np.angle(b))
00165             plt.gca().invert_yaxis()
00166             plt.savefig(dataName+"r_"+str(i)+"_phi.png",dpi=imgdpi)
00167             plt.close()
00168             f = plt.imshow(abs(np.fft.fftshift(np.fft.fft2(b))**2))
00169             plt.gca().invert_yaxis()
00170             plt.jet()
00171             plt.savefig(dataName+"p_"+str(i)+"_abspsi2.png",dpi=imgdpi)
00172             plt.close()
00173             g = plt.imshow(np.angle(np.fft.fftshift(np.fft.fft2(b))))
00174             plt.gca().invert_yaxis()
00175             plt.savefig(dataName+"p_"+str(i)+"_phi.png",dpi=imgdpi)
00176             plt.close()
00177             print "Saved figure: " + str(i) + ".png"
00178             plt.close()
00179         else:
00180             print "File(s) " + str(i) + ".png already exist."

```

#### 4.9.2.4 def vis.image\_gen\_single( dataName, value, imgdpi, opmode )

Definition at line 181 of file `vis.py`.

References `laplacian()`, and `struct_fact()`.

```

00181
00182 def image_gen_single(dataName, value, imgdpi,opmode):
00183     real=open(dataName + '_' + str(0)).read().splitlines()
00184     img=open(dataName + 'i_' + str(0)).read().splitlines()
00185     al_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
00186     al_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
00187     al = al_r[:] + 1j*al_i[:]
00188     bl = np.reshape(al, (xDim,yDim))
00189
00190     if not os.path.exists(dataName+"r_"+str(value)+"_abspsi2.png"):
00191         real=open(dataName + '_' + str(value)).read().splitlines()
00192         img=open(dataName + 'i_' + str(value)).read().splitlines()
00193         a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
00194         a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
00195         a = a_r[:] + 1j*a_i[:]
00196         b = np.reshape(a, (xDim,yDim))
00197         m_val=np.max(np.abs(b)**2)
00198         #scaleAxis(b,dataName,"_abspsi2",value,imgdpi)
00199         if opmode & 0b100000 > 0:
00200             fig, ax = plt.subplots()
00201             #plt.rc('text',usetex=True)
00202             #plt.rc('font',family='serif')
00203             f = plt.imshow((abs(b)**2 - abs(bl)**2), cmap='gnuplot2', vmin=-6, vmax=6)
00204             plt.title(r'$\left(\rho(r,t) - \rho(r,t_0)\right)$', t='$' + str(value*dt))
00205             cbar = fig.colorbar(f)
00206             plt.gca().set_xlabel('x ' + str(dx))
00207             plt.gca().set_ylabel('y ' + str(dy))
00208             plt.gca().invert_yaxis()
00209             plt.savefig(dataName+"r_"+str(value)+"_diffabspsi2.png", dpi=imgdpi)
00210             plt.close()
00211             #plt.rc('text',usetex=True)
00212             #plt.rc('font',family='serif')
00213
00214             fig, ax = plt.subplots()
00215             f = plt.imshow((abs(b)**2), cmap='gnuplot2', vmin=0, vmax=1e7)
00216             plt.title('rho(r) @ t=' + str(value*dt))
00217             plt.title(r'$\rho\left(r,t\right)$', t='$' + str(value*dt))
00218
00219             #plugins.connect(fig, plugins.MousePosition(fontsize=14))
00220
00221             cbar = fig.colorbar(f)
00222             plt.gca().set_xlabel('x ' + str(dx))
00223             plt.gca().set_ylabel('y ' + str(dy))
00224             plt.gca().invert_yaxis()
00225             plt.savefig(dataName+"r_"+str(value)+"_abspsi2.png", dpi=imgdpi)
00226             plt.axis('off')
00227             plt.savefig(dataName+"r_"+str(value)+"_abspsi2_axis0.pdf", bbox_inches='tight', dpi=imgdpi)
00228             plt.close()
00229
00230         if opmode & 0b010000 > 0:
00231             fig, ax = plt.subplots()
00232             g = plt.imshow(np.angle(b))
00233             cbar = fig.colorbar(g)
00234             plt.gca().invert_yaxis()
00235             plt.title('theta(r) @ t=' + str(value*dt))
00236             plt.savefig(dataName+"r_"+str(value)+"_phi.png", dpi=imgdpi)
00237             plt.close()
00238
00239         if opmode & 0b001000 > 0:
00240             fig, ax = plt.subplots()
00241             f = plt.imshow(abs(np.fft.fftshift(np.fft.fft2(b))**2))
00242             cbar = fig.colorbar(f)
00243             plt.gca().invert_yaxis()
00244             plt.jet()
00245             plt.title('rho(p) @ t=' + str(value*dt))
00246             plt.savefig(dataName+"p_"+str(value)+"_abspsi2.png", dpi=imgdpi)
00247             plt.close()
00248
00249         if opmode & 0b000100 > 0:
00250             fig, ax = plt.subplots()
00251             g = plt.imshow(np.angle(np.fft.fftshift(np.fft.fft2(b))))
00252             cbar = fig.colorbar(g)
00253             plt.gca().invert_yaxis()
00254             plt.title('theta(p) @ t=' + str(value*dt))
00255             plt.savefig(dataName+"p_"+str(value)+"_phi.png", dpi=imgdpi)
00256             plt.close()
00257
00258         if opmode & 0b000010 > 0:
00259             struct_fact(abs(b)**2, dataName+"_" + str(value), imgdpi)

```

```

00260
00261         if opmode & 0b000001 > 0:
00262             laplacian(abs(b)**2, dataName+"_" + str(value), imgdpi)
00263
00264         print "Saved figure: " + str(value) + ".png"
00265         plt.close()
00266     else:
00267         print "File(s) " + str(value) + ".png already exist."

```

Here is the call graph for this function:

#### 4.9.2.5 def vis.laplacian ( density, name, imgdpi )

Definition at line 96 of file [vis.py](#).

Referenced by [image\\_gen\\_single\(\)](#).

```

00096
00097 def laplacian(density, name, imgdpi):
00098     gx, gy = np.gradient(density)
00099     g2x, gxgy = np.gradient(gx)
00100     gygx, g2y = np.gradient(gy)
00101     fig, ax = plt.subplots()
00102     #f = plt.quiver(gx, gy)
00103     f = plt.imshow((g2x**2 + g2y**2), cmap=plt.get_cmap('spectral'))
00104     cbar = fig.colorbar(f)
00105     plt.savefig(name + "_laplacian.png", dpi=imgdpi)
00106     plt.close()
00107     f = plt.imshow((gxgy - gygx), cmap=plt.get_cmap('spectral'))
00108     cbar = fig.colorbar(f)
00109     plt.savefig(name + "_dx dy.png", dpi=imgdpi)
00110     plt.close()

```

Here is the caller graph for this function:

#### 4.9.2.6 def vis.opPot ( dataName, imgdpi )

Definition at line 121 of file [vis.py](#).

Referenced by [overlap\(\)](#).

```

00121
00122 def opPot(dataName, imgdpi):
00123     data = open(dataName).read().splitlines()
00124     a = numpy.asanyarray(data, dtype='f8')
00125     b = np.reshape(a, (xDim, yDim))
00126     fig, ax = plt.subplots()
00127     f = plt.imshow((b))
00128     plt.gca().invert_yaxis()
00129     cbar = fig.colorbar(f)
00130     plt.jet()
00131     plt.savefig(dataName + ".png", dpi=imgdpi)
00132     plt.close()

```

Here is the caller graph for this function:

#### 4.9.2.7 def vis.overlap ( dataName, initValue, finalValue, increment )

Definition at line 309 of file [vis.py](#).

References [delaunay\(\)](#), [stats.IsFit\(\)](#), [opPot\(\)](#), [hist3d.plot\\_hist\\_pcolor\(\)](#), and [vort\\_traj\(\)](#).

```

00309
00310 def overlap(dataName, initValue, finalValue, increment):
00311     real=open(dataName + '_' + str(0)).read().splitlines()
00312     img=open(dataName + 'i_' + str(0)).read().splitlines()
00313     a_r = numpy.asanyarray(real, dtype='f8') #128-bit complex

```

```

00314     a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
00315     wfc0 = a_r[:] + 1j*a_i[:]
00316     for i in range(initValue,finalValue,increment):
00317         real=open(dataName + '_' + str(value)).read().splitlines()
00318         img=open(dataName + 'i_' + str(value)).read().splitlines()
00319         a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
00320         a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
00321         a = a_r[:] + 1j*a_i[:]
00322         b = np.dot(wfc0,a)
00323         print i, np.sum(b)

```

Here is the call graph for this function:

#### 4.9.2.8 def vis.scaleAxis( data, dataName, label, value, imgdpi )

Definition at line 298 of file [vis.py](#).

```

00298
00299 def scaleAxis(data,dataName,label,value,imgdpi):
00300     fig, ax = plt.subplots()
00301     ax.xaxis.set_major_locator(ScaledLocator(dx=dx))
00302     ax.xaxis.set_major_formatter(ScaledLocator(dx=dx))
00303     f = plt.imshow(abs(data)**2)
00304     cbar = fig.colorbar(f)
00305     plt.gca().invert_yaxis()
00306     plt.jet()
00307     plt.savefig(dataName+"r_"+str(value)+"_"+label + ".png",dpi=imgdpi)
00308     plt.close()

```

#### 4.9.2.9 def vis.struct\_fact( density, name, imgdpi )

Definition at line 111 of file [vis.py](#).

Referenced by [image\\_gen\\_single\(\)](#).

```

00111
00112 def struct_fact(density,name,imgdpi):
00113     fig, ax = plt.subplots()
00114     #f = plt.quiver(gx,gy)
00115     f = plt.imshow((np.abs(np.fft.fftshift(np.fft.fft2(density))))),cmap=plt.get_cmap('prism'))
00116     cbar = fig.colorbar(f)
00117     cbar.set_clim(1e6,1e11)
00118     plt.jet()
00119     plt.savefig(name + "_struct_log10.png",dpi=imgdpi)
00120     plt.close()

```

Here is the caller graph for this function:

#### 4.9.2.10 def vis.voronoi( dataName, dataType, value )

Definition at line 87 of file [vis.py](#).

```

00087
00088 def voronoi(dataName,dataType,value):
00089     v_arr=genfromtxt(dataName + str(value) + dataType,delimiter=',')
00090     data = [[row[0],row[1]] for row in v_arr]
00091     vor = Voronoi(data)
00092     voronoi_plot_2d(vor)
00093     plt.xlim(300,700);plt.ylim(300,700);
00094     plt.savefig('voronoi_' + str(value) + '.png',dpi=200)
00095     print 'Saved Voronoi @ t=' + str(value)

```

**4.9.2.11** `def vis.vort_traj( name, imgdpi )`

Definition at line 268 of file [vis.py](#).

Referenced by [overlap\(\)](#).

```

00268
00269 def vort_traj(name,imgdpi):
00270     evMaxVal_l = evMaxVal
00271     H=genfromtxt('vort_arr_0',delimiter=',')
00272     count=0
00273     for i1 in range(incr,evMaxVal_l,incr):
00274         try:
00275             v_arr=genfromtxt('vort_lsq_' + str(i1) + '.csv',delimiter=',')
00276             H=np.column_stack((H,v_arr))
00277         except:
00278             evMaxVal_l = i1
00279             break
00280     X=np.zeros((evMaxVal_l/incr),dtype=np.float64)
00281     Y=np.zeros((evMaxVal_l/incr),dtype=np.float64)
00282     H=np.reshape(H,([num_vort,2,evMaxVal_l/incr]),order='F')
00283     for i1 in range(0, num_vort):
00284         for i2 in range(0,evMaxVal_l/incr):
00285             X[i2]=(H[i1,0,i2]*dx) - xMax
00286             Y[i2]=(H[i1,1,i2]*dx) - yMax
00287         h = plt.plot(X,Y,color=(r.random(),r.random(),r.random(),0.85),linewidth=0.1)
00288     plt.axis('equal')
00289     plt.title('Vort(x,y) from t=0 to t='+str(evMaxVal_l*dt)+" s")
00290
00291     plt.axis((-xMax/2.0, xMax/2.0, -yMax/2.0, yMax/2.0))
00292     plt.ticklabel_format(style='scientific')
00293     plt.ticklabel_format(style='scientific',axis='x', scilimits=(0,0))
00294     plt.ticklabel_format(style='scientific',axis='y', scilimits=(0,0))
00295     plt.savefig(name+".pdf")
00296     plt.close()
00297     print "Trajectories plotted."

```

Here is the caller graph for this function:

**4.9.3** Variable Documentation**4.9.3.1** `tuple vis.c = ConfigParser.ConfigParser()`

Definition at line 59 of file [vis.py](#).

**4.9.3.2** `list vis.CPUs = os.environ['SLURM_JOB_CPUS_PER_NODE']`

Definition at line 34 of file [vis.py](#).

**4.9.3.3** `tuple vis.data = numpy.ndarray(shape=(xDim,yDim))`

Definition at line 76 of file [vis.py](#).

**4.9.3.4** `tuple vis.dt = (c.getfloat('Params','dt'))`

Definition at line 71 of file [vis.py](#).

**4.9.3.5** `tuple vis.dx = (c.getfloat('Params','dx'))`

Definition at line 70 of file [vis.py](#).

**4.9.3.6** `list vis.ev_proc = []`

Definition at line 342 of file [vis.py](#).

4.9.3.7 `list vis.evImgList = []`

Definition at line 336 of file [vis.py](#).

4.9.3.8 `tuple vis.evMaxVal = int(c.getfloat('Params','esteps'))`

Definition at line 67 of file [vis.py](#).

4.9.3.9 `list vis.gnd_proc = []`

Definition at line 341 of file [vis.py](#).

4.9.3.10 `list vis.gndImgList = []`

Definition at line 335 of file [vis.py](#).

4.9.3.11 `tuple vis.gndMaxVal = int(c.getfloat('Params','gsteps'))`

Definition at line 66 of file [vis.py](#).

4.9.3.12 `tuple vis.i = gndImgList.pop()`

Definition at line 344 of file [vis.py](#).

Referenced by [delta\\_define\(\)](#), [energy\\_angmom\(\)](#), [evolve\(\)](#), [Tracker.findOLMaxima\(\)](#), [Tracker.findVortex\(\)](#), [initialise\(\)](#), [Minions.maxValue\(\)](#), [Minions.minValue\(\)](#), [multipass\(\)](#), [Tracker.olPos\(\)](#), [optLatSetup\(\)](#), [pSum\(\)](#), [pSumT\(\)](#), [FileIO.readIn\(\)](#), [sepAvg\(\)](#), [Tracker.sigVOL\(\)](#), [Minions.sumAvg\(\)](#), [Tracker.vortAngle\(\)](#), [Tracker.vortArrange\(\)](#), [Tracker.vortCentre\(\)](#), [Tracker.vortPos\(\)](#), [FileIO.writeOut\(\)](#), [FileIO.writeOutDouble\(\)](#), [FileIO.writeOutInt\(\)](#), [FileIO.writeOutInt2\(\)](#), [FileIO.writeOutParam\(\)](#), and [FileIO.writeOutVortex\(\)](#).

4.9.3.13 `tuple vis.incr = int(c.getfloat('Params','print_out'))`

Definition at line 68 of file [vis.py](#).

4.9.3.14 `int vis.num_vort = 0`

Definition at line 74 of file [vis.py](#).

4.9.3.15 `tuple vis.p = proc.pop()`

Definition at line 360 of file [vis.py](#).

Referenced by [appendData\(\)](#), and [newParam\(\)](#).

4.9.3.16 `vis.proc = gnd_proc+ev_proc`

Definition at line 354 of file [vis.py](#).

4.9.3.17 `tuple vis.sep = (c.getfloat('Params','dx'))`

Definition at line 69 of file [vis.py](#).

4.9.3.18 `tuple vis.xDim = int(c.getfloat('Params','xDim'))`

Definition at line 64 of file [vis.py](#).

4.9.3.19 `tuple vis.xMax = (c.getfloat('Params','xMax'))`

Definition at line 72 of file [vis.py](#).

4.9.3.20 `tuple vis.yDim = int(c.getfloat('Params','yDim'))`

Definition at line 65 of file [vis.py](#).

4.9.3.21 `tuple vis.yMax = (c.getfloat('Params','yMax'))`

Definition at line 73 of file [vis.py](#).

## 4.10 vort Namespace Reference

### Classes

- class [Vortex](#)
- class [VtxList](#)

### Functions

- def [\\_\\_init\\_\\_](#)
- def [\\_\\_init\\_\\_](#)
- def [add](#)
- def [as\\_np](#)
- def [dist](#)
- def [do\\_the\\_thing](#)
- def [element](#)
- def [idx\\_min\\_dist](#)
- def [max\\_uid](#)
- def [remove](#)
- def [swap\\_uid](#)
- def [update\\_next](#)
- def [update\\_on](#)
- def [update\\_uid](#)
- def [vort\\_decrease](#)
- def [vort\\_increase](#)
- def [vtx\\_uid](#)
- def [write\\_out](#)

### Variables

- tuple [c](#) = `ConfigParser.ConfigParser()`
- int [counter](#) = 0
- tuple [current](#) = `self.element(pos-1)`
- list [data](#) = []
- tuple [dcp](#) = `set(uid_c)`

- tuple `dpc` = set(`uid_p`)
- tuple `dt` = (c.getfloat('Params','dt'))
- list `dtype` = [('x',float),('y',float),('sign',int),('uid',int),('isOn',int)]
- tuple `dx` = (c.getfloat('Params','dx'))
- tuple `evMaxVal` = int(c.getfloat('Params','esteps'))
- tuple `gndMaxVal` = int(c.getfloat('Params','gsteps'))
- int `i` = 0
- tuple `incr` = int(c.getfloat('Params','print\_out'))
- tuple `index_r` = vorts\_c.idx\_min\_dist(vorts\_p.element(i3))
- tuple `max_uid` = vorts\_p.max\_uid()
- int `pos` = 0
- int `pos_l` = 0
- tuple `r` = m.sqrt((self.x - vtx.x)\*\*2 + (self.y - vtx.y)\*\*2)
- `ret_idx` = counter
- list `uid_c` = [[a for a in b][3] for b in vorts\_c.as\_np()]
- list `uid_p` = [[a for a in b][3] for b in vorts\_p.as\_np()]
- tuple `v0c` = vorts\_c.element(`index_r`[0])
- tuple `v0p` = vorts\_p.element(i3)
- tuple `v1c` = vorts\_c.element(`index_r`[0])
- tuple `v_arr_c` = genfromtxt('vort\_lsq\_' + str(i) + '.csv',delimiter=',')
- tuple `v_arr_c_coords` = np.array([[a for a in v][:2] for v in v\_arr\_c])
- tuple `v_arr_c_sign` = np.array([[a for a in v][2] for v in v\_arr\_c])
- tuple `v_arr_p` = genfromtxt('vort\_lsq\_' + str(0) + '.csv',delimiter=',')
- `v_arr_p=genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')`
- tuple `v_arr_p_coords` = np.array([[a for a in v][:2] for v in v\_arr\_p])
- tuple `v_arr_p_sign` = np.array([[a for a in v][2] for v in v\_arr\_p])
- int `val` = 0
- tuple `vorts_c` = VtxList()
- tuple `vorts_c_update` = sorted(vorts\_c.as\_np(),key=lambda vtx: vtx[3])
- tuple `vorts_p` = VtxList()
- `vtx` = self.head
- tuple `vtx_c` = Vortex(-1-i2,v\_arr\_c\_coords[i2][0],v\_arr\_c\_coords[i2][1],True,sign=v\_arr\_c\_sign[i2])
- tuple `vtx_p` = Vortex(i1,v\_arr\_p\_coords[i1][0],v\_arr\_p\_coords[i1][1],True,sign=v\_arr\_p\_sign[i1])
- tuple `vtx_pos` = self.vtx\_uid(uid\_i)
- list `vtx_pos_c` = []
- list `vtx_pos_p` = []
- tuple `xDim` = int(c.getfloat('Params','xDim'))
- tuple `xMax` = (c.getfloat('Params','xMax'))
- tuple `yDim` = int(c.getfloat('Params','yDim'))
- tuple `yMax` = (c.getfloat('Params','yMax'))

#### 4.10.1 Detailed Description

vort.py - GPUE: Split Operator based GPU solver for Nonlinear Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.



3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 4.10.2 Class Documentation

### 4.10.2.1 class vort::Vortex

Definition at line 56 of file [vort.py](#).

Collaboration diagram for vort.Vortex:

### 4.10.2.2 class vort::VtxList

Definition at line 90 of file [vort.py](#).

Collaboration diagram for vort.VtxList:

## 4.10.3 Function Documentation

### 4.10.3.1 def vort.\_\_init\_\_( self, uid, x, y, isOn, sign = 1 )

Definition at line 59 of file [vort.py](#).

Referenced by [\\_\\_init\\_\\_\(\)](#).

```
00059
def __init__(self,uid,x,y,isOn,sign=1):
```

Here is the caller graph for this function:

### 4.10.3.2 def vort.\_\_init\_\_( self )

Definition at line 93 of file [vort.py](#).

References [\\_\\_init\\_\\_\(\)](#).

```
00093
def __init__(self):
```

Here is the call graph for this function:

### 4.10.3.3 def vort.add( self, Vtx, index = None )

Definition at line 141 of file [vort.py](#).

```
00141
def add(self,Vtx,index=None): #Add a vtx at index, otherwise end
```

#### 4.10.3.4 `def vort.as_np( self )`

Definition at line 157 of file [vort.py](#).

```
00157
def as_np(self): #Return numpy array with format x,y,sign,uid,isOn
```

#### 4.10.3.5 `def vort.dist( self, vtx )`

Definition at line 84 of file [vort.py](#).

Referenced by [Tracker.vortArrange\(\)](#).

```
00084
def dist(self,vtx): #Distance between self and vtx
```

Here is the caller graph for this function:

#### 4.10.3.6 `def vort.do_the_thing( start, fin, incr )`

Definition at line 236 of file [vort.py](#).

```
00236
def do_the_thing(start,fin,incr): #Performs the tracking
```

#### 4.10.3.7 `def vort.element( self, pos )`

Definition at line 100 of file [vort.py](#).

```
00100
def element(self,pos): #Get vtx at position pos
```

#### 4.10.3.8 `def vort.idx_min_dist( self, vortex, isSelf=False )`

Definition at line 175 of file [vort.py](#).

```
00175
def idx_min_dist(self,vortex, isSelf=False): #Closest vtx to self
```

#### 4.10.3.9 `def vort.max_uid( self )`

Definition at line 124 of file [vort.py](#).

References [max\\_uid](#).

```
00124
def max_uid(self): #Return position and value of largest uid
```

#### 4.10.3.10 `def vort.remove( self, pos )`

Definition at line 191 of file [vort.py](#).

```
00191
def remove(self,pos): #Remove vortices outside artificial boundary
```

## 4.10.3.11 def vort.swap\_uid ( self, uid\_i, uid\_f )

Definition at line 210 of file [vort.py](#).

```
00210
def swap_uid(self,uid_i,uid_f): #Swap uid between vtx
```

## 4.10.3.12 def vort.update\_next ( self, next )

Definition at line 79 of file [vort.py](#).

```
00079
def update_next(self,next): #Get next vortex
```

## 4.10.3.13 def vort.update\_on ( self, isOn )

Definition at line 74 of file [vort.py](#).

```
00074
def update_on(self,isOn): #Vortex is trackable
```

## 4.10.3.14 def vort.update\_uid ( self, uid )

Definition at line 69 of file [vort.py](#).

Referenced by [vort\\_increase\(\)](#).

```
00069
def update_uid(self,uid):
```

Here is the caller graph for this function:

## 4.10.3.15 def vort.vort\_decrease ( self, positions, vorts\_p )

Definition at line 217 of file [vort.py](#).

```
00217
def vort_decrease(self,positions,vorts_p): #Turn off vortex timeline
```

## 4.10.3.16 def vort.vort\_increase ( self, positions, vorts\_p )

Definition at line 227 of file [vort.py](#).

References [update\\_uid\(\)](#).

```
00227
def vort_increase(self,positions,vorts_p): #Add new vtx to tracking
```

Here is the call graph for this function:

## 4.10.3.17 def vort.vtx\_uid ( self, uid )

Definition at line 114 of file [vort.py](#).

```
00114
def vtx_uid(self,uid): #Get vtx with identifier uid
```

4.10.3.18 `def vort.write_out ( self, time, data )`

Definition at line 170 of file [vort.py](#).

```
00170
def write_out(self,time,data): #Write out CSV file as x,y,sign,uid,isOn
```

#### 4.10.4 Variable Documentation

4.10.4.1 `tuple vort.c = ConfigParser.ConfigParser()`

Definition at line 42 of file [vort.py](#).

4.10.4.2 `int vort.counter = 0`

Definition at line 177 of file [vort.py](#).

Referenced by [Tracker.olPos\(\)](#), [Tracker.vortCentre\(\)](#), and [Tracker.vortPos\(\)](#).

4.10.4.3 `vort.current = self.element(pos-1)`

Definition at line 194 of file [vort.py](#).

4.10.4.4 `list vort.data = []`

Definition at line 160 of file [vort.py](#).

4.10.4.5 `tuple vort.dcp = set(uid_c)`

Definition at line 274 of file [vort.py](#).

4.10.4.6 `tuple vort.dpc = set(uid_p)`

Definition at line 273 of file [vort.py](#).

4.10.4.7 `tuple vort.dt = (c.getfloat('Params','dt'))`

Definition at line 51 of file [vort.py](#).

4.10.4.8 `list vort.dtype = [('x',float),('y',float),('sign',int),('uid',int),('isOn',int)]`

Definition at line 159 of file [vort.py](#).

4.10.4.9 `tuple vort.dx = (c.getfloat('Params','dx'))`

Definition at line 50 of file [vort.py](#).

4.10.4.10 `tuple vort.evMaxVal = int(c.getfloat('Params','esteps'))`

Definition at line 48 of file [vort.py](#).

4.10.4.11 `tuple vort.gndMaxVal = int(c.getfloat('Params','gsteps'))`

Definition at line 47 of file [vort.py](#).

4.10.4.12 `int vort.i = 0`

Definition at line 161 of file [vort.py](#).

4.10.4.13 `tuple vort.incr = int(c.getfloat('Params','print_out'))`

Definition at line 49 of file [vort.py](#).

4.10.4.14 `tuple vort.index_r = vorts_c.idx_min_dist(vorts_p.element(i3))`

Definition at line 258 of file [vort.py](#).

4.10.4.15 `tuple vort.max_uid = vorts_p.max_uid()`

Definition at line 219 of file [vort.py](#).

Referenced by [max\\_uid\(\)](#).

4.10.4.16 `int vort.pos = 0`

Definition at line 117 of file [vort.py](#).

4.10.4.17 `int vort.pos_l = 0`

Definition at line 102 of file [vort.py](#).

4.10.4.18 `tuple vort.r = m.sqrt((self.x - vtx.x)**2 + (self.y - vtx.y)**2)`

Definition at line 86 of file [vort.py](#).

4.10.4.19 `vort.ret_idx = counter`

Definition at line 178 of file [vort.py](#).

4.10.4.20 `list vort.uid_c = [[a for a in b][3] for b in vorts_c.as_np()]`

Definition at line 269 of file [vort.py](#).

4.10.4.21 `list vort.uid_p = [[a for a in b][3] for b in vorts_p.as_np()]`

Definition at line 270 of file [vort.py](#).

4.10.4.22 `tuple vort.v0c = vorts_c.element(index_r[0])`

Definition at line 260 of file [vort.py](#).

4.10.4.23 `tuple vort.v0p = vorts_p.element(i3)`

Definition at line 261 of file [vort.py](#).

4.10.4.24 `tuple vort.v1c = vorts_c.element(index_r[0])`

Definition at line 262 of file [vort.py](#).

4.10.4.25 `tuple vort.v_arr_c = genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',')`

Definition at line 244 of file [vort.py](#).

4.10.4.26 `tuple vort.v_arr_c_coords = np.array([[a for a in v][:2] for v in v_arr_c])`

Definition at line 246 of file [vort.py](#).

4.10.4.27 `tuple vort.v_arr_c_sign = np.array([[a for a in v][2] for v in v_arr_c])`

Definition at line 248 of file [vort.py](#).

4.10.4.28 `tuple vort.v_arr_p = genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')`

`v_arr_p=genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')`

Definition at line 239 of file [vort.py](#).

4.10.4.29 `tuple vort.v_arr_p_coords = np.array([[a for a in v][:2] for v in v_arr_p])`

Definition at line 245 of file [vort.py](#).

4.10.4.30 `tuple vort.v_arr_p_sign = np.array([[a for a in v][2] for v in v_arr_p])`

Definition at line 247 of file [vort.py](#).

4.10.4.31 `vort.val = 0`

Definition at line 126 of file [vort.py](#).

4.10.4.32 `tuple vort.vorts_c = VtxList()`

Definition at line 242 of file [vort.py](#).

4.10.4.33 `tuple vort.vorts_c_update = sorted(vorts_c.as_np(),key=lambda vtx: vtx[3])`

Definition at line 285 of file [vort.py](#).

4.10.4.34 `tuple vort.vorts_p = VtxList()`

Definition at line 241 of file [vort.py](#).

4.10.4.35 `tuple vort.vtx = self.head`

Definition at line 104 of file [vort.py](#).

4.10.4.36 `tuple vort.vtx_c = Vortex(-1-i2,v_arr_c_coords[i2][0],v_arr_c_coords[i2][1],True,sign=v_arr_c_sign[i2])`

Definition at line 254 of file [vort.py](#).

4.10.4.37 `tuple vort.vtx_p = Vortex(i1,v_arr_p_coords[i1][0],v_arr_p_coords[i1][1],True,sign=v_arr_p_sign[i1])`

Definition at line 250 of file [vort.py](#).

4.10.4.38 `tuple vort.vtx_pos = self.vtx_uid(uid_i)`

Definition at line 212 of file [vort.py](#).

4.10.4.39 `tuple vort.vtx_pos_c = []`

Definition at line 276 of file [vort.py](#).

4.10.4.40 `tuple vort.vtx_pos_p = []`

Definition at line 275 of file [vort.py](#).

4.10.4.41 `tuple vort.xDim = int(c.getfloat('Params','xDim'))`

Definition at line 45 of file [vort.py](#).

4.10.4.42 `tuple vort.xMax = (c.getfloat('Params','xMax'))`

Definition at line 52 of file [vort.py](#).

4.10.4.43 `tuple vort.yDim = int(c.getfloat('Params','yDim'))`

Definition at line 46 of file [vort.py](#).

4.10.4.44 `tuple vort.yMax = (c.getfloat('Params','yMax'))`

Definition at line 53 of file [vort.py](#).





## Class Documentation

```
00057                                     {
00058
00059 }
```

### 5.1.3 Member Function Documentation

5.1.3.1 `int2 BEC2D::Wavefunction::getGridSize ( int xDim, int yDim )`

5.1.3.2 `double2& BEC2D::Wavefunction::getWfc ( )`

5.1.3.3 `double2 BEC2D::Wavefunction::initWfc ( )`

5.1.3.4 `BEC2D::Wavefunction::setGridSize ( int xDim, int yDim )`

Definition at line 60 of file [wavefunction.cu](#).

```
00060                                     {
00061
00062     }
```

### 5.1.4 Member Data Documentation

5.1.4.1 `double2 BEC2D::Wavefunction::dimMax [private]`

Definition at line 42 of file [wavefunction.cu](#).

5.1.4.2 `int2 BEC2D::Wavefunction::gridSize [private]`

Definition at line 41 of file [wavefunction.cu](#).

5.1.4.3 `double2* BEC2D::Wavefunction::wfc = new double2[xDim*yDim] [private]`

Definition at line 43 of file [wavefunction.cu](#).

The documentation for this class was generated from the following file:

- [src/wavefunction.cu](#)

## Chapter 6

# File Documentation

### 6.1 bin/batch\_run.sh File Reference

### 6.2 batch\_run.sh

```
00001 #GPUE: Split Operator based GPU solver for Nonlinear
00002 #Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O’Riordan
00003 #<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00004 #Redistribution and use in source and binary forms, with or without
00005 #modification, are permitted provided that the following conditions are
00006 #met:
00007 #
00008 #1. Redistributions of source code must retain the above copyright
00009 #notice, this list of conditions and the following disclaimer.
00010 #
00011 #2. Redistributions in binary form must reproduce the above copyright
00012 #notice, this list of conditions and the following disclaimer in the
00013 #documentation and/or other materials provided with the distribution.
00014 #
00015 #3. Neither the name of the copyright holder nor the names of its
00016 #contributors may be used to endorse or promote products derived from
00017 #this software without specific prior written permission.
00018 #
00019 #THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00020 #"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00021 #LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00022 #PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00023 #HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 #SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00025 #TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 #PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 #LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00028 #NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00029 #SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 #!/bin/bash
00031
00032 #SBATCH --job-name=
00033 #SBATCH --partition=
00034 #SBATCH --mem=
00035 #SBATCH --cpus-per-task=
00036 #SBATCH --ntasks=
00037 #SBATCH --error=
00038 #SBATCH --mail-type=ALL
00039 #SBATCH --mail-user=
```

### 6.3 bin/path.sh File Reference

### 6.4 path.sh

```
00001 #GPUE: Split Operator based GPU solver for Nonlinear
00002 #Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O’Riordan
00003 #<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00004 #Redistribution and use in source and binary forms, with or without
```

```

00005 #modification, are permitted provided that the following conditions are
00006 #met:
00007 #
00008 #1. Redistributions of source code must retain the above copyright
00009 #notice, this list of conditions and the following disclaimer.
00010 #
00011 #2. Redistributions in binary form must reproduce the above copyright
00012 #notice, this list of conditions and the following disclaimer in the
00013 #documentation and/or other materials provided with the distribution.
00014 #
00015 #3. Neither the name of the copyright holder nor the names of its
00016 #contributors may be used to endorse or promote products derived from
00017 #this software without specific prior written permission.
00018 #
00019 #THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00020 #"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00021 #LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00022 #PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00023 #HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 #SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00025 #TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 #PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 #LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00028 #NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00029 #SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 #!/bin/bash
00031 export PATH=$PATH:/usr/local/cuda/bin:/usr/local/cuda/open64/bin
00032 export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64

```

## 6.5 bin/run.sh File Reference

### 6.6 run.sh

```

00001 #GPUE: Split Operator based GPU solver for Nonlinear
00002 #Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 #<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00004 #Redistribution and use in source and binary forms, with or without
00005 #modification, are permitted provided that the following conditions are
00006 #met:
00007 #
00008 #1. Redistributions of source code must retain the above copyright
00009 #notice, this list of conditions and the following disclaimer.
00010 #
00011 #2. Redistributions in binary form must reproduce the above copyright
00012 #notice, this list of conditions and the following disclaimer in the
00013 #documentation and/or other materials provided with the distribution.
00014 #
00015 #3. Neither the name of the copyright holder nor the names of its
00016 #contributors may be used to endorse or promote products derived from
00017 #this software without specific prior written permission.
00018 #
00019 #THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00020 #"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00021 #LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00022 #PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00023 #HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 #SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00025 #TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 #PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 #LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00028 #NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00029 #SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 #!/bin/bash
00031 i=0
00032 EMAIL=mymail@addr.com
00033 count=0
00034 NAME=$1
00035 PARAMS=$2
00036 declare -a JOBS=(-1 -1 -1 -1 -1 -1 -1 -1)
00037 function run_gpue_test {
00038     echo $1
00039 }
00040
00041 function run_gpue {
00042     if [[ $(echo $1 | head -c 1) == "#" ]];then
00043         return;
00044     elif [[ $(echo $1 | head -c 1) == "" ]];then
00045         return
00046     fi
00047
00048     if [ -n "$NAME" ];then

```

```

00049     NAME=$(echo $NAME)_
00050     fi
00051     sleep 1
00052     A=$(date '+%y/%m/%d/%H_%M_%S')
00053     if [ -d ./A ]; then
00054         echo "Exists"
00055         A=$A-$i
00056         i=$((i+1))
00057     fi
00058     echo "$NAME$A"
00059     mkdir -p $NAME$A
00060     cp ./gpue ./NAME$A; cp -r ./src ./NAME$A; cp -r ./include ./NAME$A; cp ./Makefile ./NAME$A; cp -
r ./py ./NAME$A; cp -r ./bin ./NAME$A; cp ./wfc_load ./NAME$A; cp ./wfc_load ./NAME$A;
00061     cd ./NAME$A
00062     pwd >> result.log
00063     echo $1 >>result.log
00064     mail -s "#Started GPU Job# $A" lee.oriordan@oist.jp < result.log
00065     ./gpue $1 2>&1> result.log
00066     mkdir -p ./images
00067     #python ./py/vis.py >> result.log
00068     cp *.png ./images
00069     cd ./images
00070     ls | grep wfc_evr | grep _abs | grep png | sort -k3 -t _ -n > list1.txt;mencoder mf://@list1.txt -mf
w=1280:h=1024:fps=24:type=png -oac copy -ovc lavc -lavcopts
vcodec=mpeg4:mbd=2:mv0:trell:v4mv:cbp:last_pred=3:predia=2:dia=2:vmax_b_frames=2:vb_strategy=1:precmp=2:cmp=2:subcmp=2
wfc_${PWD##*/}.avi
00071     rm -rf ./*.png
00072     #python ./py/hist3d.py
00073     rm wfc*
00074     mail -s "#Completed GPU Job# $A" $EMAIL < $(echo $(cat result.log; cat ./Params.dat))
00075     cd ../../../../..
00076     sleep 1
00077 }
00078
00079 while read line ; do
00080     run_gpue "$line" &
00081     #echo "Running $line"
00082     JOBS[$count]=$!
00083     let count+=1
00084
00085     if [ $count -gt 7 ]; then
00086         wait
00087         count=0
00088     fi
00089 done < $PARAMS

```

## 6.7 bin/sanity\_test.sh File Reference

### Variables

- [FILE](#)
- do let POSITION if ["\$!"!="0.0000000000000000e+00"]

### 6.7.1 Variable Documentation

#### 6.7.1.1 FILE

#### Initial value:

```

=$1
COUNTER=0
POSITION=-1
ARR[0]=0
for i in $(cat $FILE)

```

Definition at line 32 of file [sanity\\_test.sh](#).

Referenced by [FileIO::readIn\(\)](#), [FileIO::readState\(\)](#), [FileIO::writeOut\(\)](#), [FileIO::writeOutDouble\(\)](#), [FileIO::writeOutInt\(\)](#), [FileIO::writeOutInt2\(\)](#), [FileIO::writeOutParam\(\)](#), and [FileIO::writeOutVortex\(\)](#).

### 6.7.1.2 do let POSITION iff["\$i"!="0.0000000000000000e+00"]

Definition at line 39 of file [sanity\\_test.sh](#).

## 6.8 sanity\_test.sh

```

00001 #GPUE: Split Operator based GPU solver for Nonlinear
00002 #Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O’Riordan
00003 #<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00004 #Redistribution and use in source and binary forms, with or without
00005 #modification, are permitted provided that the following conditions are
00006 #met:
00007 #
00008 #1. Redistributions of source code must retain the above copyright
00009 #notice, this list of conditions and the following disclaimer.
00010 #
00011 #2. Redistributions in binary form must reproduce the above copyright
00012 #notice, this list of conditions and the following disclaimer in the
00013 #documentation and/or other materials provided with the distribution.
00014 #
00015 #3. Neither the name of the copyright holder nor the names of its
00016 #contributors may be used to endorse or promote products derived from
00017 #this software without specific prior written permission.
00018 #
00019 #THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00020 #"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00021 #LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00022 #PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00023 #HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 #SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00025 #TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 #PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 #LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00028 #NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00029 #SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030
00031 #!/bin/bash
00032 FILE=$1
00033 COUNTER=0
00034 POSITION=-1
00035 ARR[0]=0
00036 for i in $(cat $FILE);
00037 do
00038     let POSITION++
00039     if [ "$i" != "0.0000000000000000e+00" ];
00040     then
00041         ARR[$COUNTER]=$POSITION
00042         let COUNTER++
00043     fi
00044 done
00045 echo Non-zero elements $COUNTER
00046 echo "Elements located at:"
00047 for item in ${ARR[*]}
00048 do
00049     printf "%s\n" $item
00050 done
00051 done

```

## 6.9 bin/upload\_vids.sh File Reference

### Functions

- do echo (if [[\$(basename \$(dirname \$i))=='images']];then cd \$(dirname \$i)/../bin;TITLE=\$(head -n 1 run\_ - params.conf) SUMMARY=\$(head -n 20 ../result.log) cd -google youtube post --category Tech \$i --title "\$TITLE" --summary "\$SUMMARY" --access=unlisted \$i fi)

### Variables

- OLDPWD

## 6.9.1 Function Documentation

**6.9.1.1** `do echo ( if];then cd $(dirname $i)/../bin; TITLE[$(basename $(dirname $i))==images'] = $(head -n 1 run_`  
`params.conf) SUMMARY=$(head -n 20 ../result.log) cd-google youtube post--category Tech`  
`)`

## 6.9.2 Variable Documentation

### 6.9.2.1 OLDPWD

**Initial value:**

```
= $(pwd)
for i in $(cat ./ogg.txt | grep wfc)
```

Definition at line 31 of file [upload\\_vids.sh](#).

## 6.10 upload\_vids.sh

```
00001 #GPUE: Split Operator based GPU solver for Nonlinear
00002 #Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 #<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00004 #Redistribution and use in source and binary forms, with or without
00005 #modification, are permitted provided that the following conditions are
00006 #met:
00007 #
00008 #1. Redistributions of source code must retain the above copyright
00009 #notice, this list of conditions and the following disclaimer.
00010 #
00011 #2. Redistributions in binary form must reproduce the above copyright
00012 #notice, this list of conditions and the following disclaimer in the
00013 #documentation and/or other materials provided with the distribution.
00014 #
00015 #3. Neither the name of the copyright holder nor the names of its
00016 #contributors may be used to endorse or promote products derived from
00017 #this software without specific prior written permission.
00018 #
00019 #THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00020 #"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00021 #LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00022 #PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00023 #HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 #SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00025 #TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 #PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 #LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00028 #NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00029 #SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 #!/bin/bash
00031 OLDPWD=$(pwd)
00032 for i in $(cat ./ogg.txt | grep wfc);
00033 do
00034     echo $(if [[ $(basename $(dirname $i))==images' ]];
00035     then
00036         cd $(dirname $i)/../bin;
00037         TITLE=$(head -n 1 run_params.conf)
00038         SUMMARY=$(head -n 20 ../result.log)
00039         cd -
00040         google youtube post --category Tech $i --title "$TITLE" --summary "$SUMMARY" --access=
unlisted $i
00041     fi);
00042 done
00043
```

## 6.11 bin/zippit.sh File Reference

### Functions

- `for i in (cat manifest.txt)`

## Variables

- do [echo](#) Working on \$i

### 6.11.1 Function Documentation

#### 6.11.1.1 for i in ( cat manifest. txt )

Referenced by [conjugate\(\)](#), and [Minions::flnvSqRt\(\)](#).

Here is the caller graph for this function:

### 6.11.2 Variable Documentation

#### 6.11.2.1 \$HOME builds bin pigz p r \$i

Definition at line 31 of file [zippit.sh](#).

## 6.12 zippit.sh

```

00001 #GPUE: Split Operator based GPU solver for Nonlinear
00002 #Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O’Riordan
00003 #<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00004 #Redistribution and use in source and binary forms, with or without
00005 #modification, are permitted provided that the following conditions are
00006 #met:
00007 #
00008 #1. Redistributions of source code must retain the above copyright
00009 #notice, this list of conditions and the following disclaimer.
00010 #
00011 #2. Redistributions in binary form must reproduce the above copyright
00012 #notice, this list of conditions and the following disclaimer in the
00013 #documentation and/or other materials provided with the distribution.
00014 #
00015 #3. Neither the name of the copyright holder nor the names of its
00016 #contributors may be used to endorse or promote products derived from
00017 #this software without specific prior written permission.
00018 #
00019 #THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00020 #"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00021 #LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00022 #PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00023 #HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 #SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00025 #TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 #PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 #LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00028 #NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00029 #SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 #!/bin/bash
00031 for i in $(cat manifest.txt); do echo 'Working on $i'; $HOME/builds/bin/pigz -
    p 24 -r $i; done

```

## 6.13 include/constants.h File Reference

This graph shows which files directly or indirectly include this file:

## Macros

- #define [EPSILON\\_0](#) 8.854187817620e-12
- #define [HBAR](#) 1.05457148e-34
- #define [INV\\_RT\\_2](#) 0.7071067811865475
- #define [MU\\_0](#) 4\*PI\*1e-7



- `#define MU_B 9.27400915e-24`
- `#define MU_N 5.05078324e-27`
- `#define PI 3.141592653589793`
- `#define Q 1.602176565e-19`
- `#define RT_2 1.4142135623730951`

### 6.13.1 Macro Definition Documentation

#### 6.13.1.1 `#define EPSILON_0 8.854187817620e-12`

Definition at line 42 of file [constants.h](#).

#### 6.13.1.2 `#define HBAR 1.05457148e-34`

Definition at line 37 of file [constants.h](#).

Referenced by [cMultDensity\(\)](#), [delta\\_define\(\)](#), [energyCalc\(\)](#), [evolve\(\)](#), [initialise\(\)](#), and [optLatSetup\(\)](#).

#### 6.13.1.3 `#define INV_RT_2 0.7071067811865475`

Definition at line 43 of file [constants.h](#).

#### 6.13.1.4 `#define MU_0 4*PI*1e-7`

Definition at line 41 of file [constants.h](#).

#### 6.13.1.5 `#define MU_B 9.27400915e-24`

Definition at line 39 of file [constants.h](#).

#### 6.13.1.6 `#define MU_N 5.05078324e-27`

Definition at line 38 of file [constants.h](#).

#### 6.13.1.7 `#define PI 3.141592653589793`

Definition at line 36 of file [constants.h](#).

Referenced by [cMultDensity\(\)](#), [evolve\(\)](#), [Tracker::findVortex\(\)](#), [initialise\(\)](#), [optLatSetup\(\)](#), [Tracker::phaseTest\(\)](#), and [Tracker::vortAngle\(\)](#).

#### 6.13.1.8 `#define Q 1.602176565e-19`

Definition at line 40 of file [constants.h](#).

#### 6.13.1.9 `#define RT_2 1.4142135623730951`

Definition at line 44 of file [constants.h](#).

## 6.14 constants.h

```

00001 /** constants.h - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033 #ifndef CONSTANTS_H
00034 #define CONSTANTS_H
00035
00036 #define PI 3.141592653589793
00037 #define HBAR 1.05457148e-34 // m^2 kg/s
00038 #define MU_N 5.05078324e-27 // J/T Nuclear magneton
00039 #define MU_B 9.27400915e-24 // J/T Bohr magneton
00040 #define Q 1.602176565e-19 // C Elementary charge of proton
00041 #define MU_0 4*PI*1e-7 // V*S/A*m or H/m or N/A^2 Vacuum permeability
00042 #define EPSILON_0 8.854187817620e-12 // F/m Vacuum permittivity
00043 #define INV_RT_2 0.7071067811865475 // 1/sqrt(2)
00044 #define RT_2 1.4142135623730951 // sqrt(2)
00045
00046 #endif

```

## 6.15 include/ds.h File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

Include dependency graph for ds.h: This graph shows which files directly or indirectly include this file:

### Classes

- struct [Array](#)
- struct [Param](#)

### Functions

- void [appendData](#) ([Array](#) \*arr, char \*t, double d)
- void [freeArray](#) ([Array](#) \*arr)
- void [initArr](#) ([Array](#) \*arr, size\_t initLen)
- [Param](#) [newParam](#) (char \*t, double d)

## 6.15.1 Class Documentation

### 6.15.1.1 struct Array

Definition at line 46 of file [ds.h](#).

Collaboration diagram for Array:

#### Class Members

<a href="#">Param *</a>	array	
<a href="#">size_t</a>	length	
<a href="#">size_t</a>	used	

### 6.15.1.2 struct Param

Definition at line 40 of file [ds.h](#).

Collaboration diagram for Param:

#### Class Members

<a href="#">double</a>	data	
<a href="#">char</a>	title[32]	

## 6.15.2 Function Documentation

### 6.15.2.1 void appendData ( Array \* arr, char \* t, double d )

Definition at line 42 of file [ds.cc](#).

References [Array::array](#), [Array::length](#), [newParam\(\)](#), [vis::p](#), and [Array::used](#).

Referenced by [evolve\(\)](#), [initialise\(\)](#), [optLatSetup\(\)](#), and [parseArgs\(\)](#).

```

00042                                     {
00043     Param p = newParam(t,d);
00044     if(arr->used == arr->length){
00045         arr->length *= 2;
00046         arr->array = (Param*)realloc(arr->array, arr->length*sizeof(
Param));
00047     }
00048     arr->array[arr->used] = p;
00049     arr->used = arr->used + 1;
00050 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

### 6.15.2.2 void freeArray ( Array \* arr )

Definition at line 52 of file [ds.cc](#).

References [Array::array](#), [Array::length](#), and [Array::used](#).

```

00052                                     {
00053     free(arr->array);
00054     arr->array = NULL;
00055     arr->used = 0;
00056     arr->length = 0;
00057 }
```

### 6.15.2.3 void initArr ( Array \* arr, size\_t initLen )

Definition at line 36 of file [ds.cc](#).

References [Array::array](#), [Array::length](#), and [Array::used](#).

Referenced by [main\(\)](#).

```
00036                                     {
00037     arr->array = (Param*) malloc(initLen*sizeof(Param));
00038     arr->used = 0;
00039     arr->length = initLen;
00040 }
```

Here is the caller graph for this function:

### 6.15.2.4 Param newParam ( char \* t, double d )

Definition at line 59 of file [ds.cc](#).

References [Param::data](#), [vis::p](#), and [Param::title](#).

Referenced by [appendData\(\)](#).

```
00059                                     {
00060     Param p;
00061     strcpy(p.title,t);
00062     p.data = d;
00063     return p;
00064 }
```

Here is the caller graph for this function:

## 6.16 ds.h

```
00001 /** ds.h - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034 #ifndef DS_H
00035 #define DS_H
00036 #include<stdio.h>
00037 #include<stdlib.h>
00038 #include<string.h>
```

```

00039
00040 struct Param{
00041     char title[32];
00042     double data;
00043 };
00044 typedef struct Param Param;
00045
00046 struct Array{
00047     Param *array;
00048     size_t length;
00049     size_t used;
00050 };
00051 typedef struct Array Array;
00052
00053 void initArr(Array *arr, size_t initLen);
00054 void appendData(Array *arr, char* t, double d);
00055 void freeArray(Array *arr);
00056 Param newParam(char* t,double d);
00057 #endif

```

## 6.17 include/fileIO.h File Reference

```

#include "../include/ds.h"
#include "../include/tracker.h"

```

Include dependency graph for fileIO.h: This graph shows which files directly or indirectly include this file:

### Namespaces

- [FileIO](#)

*Check source file for information on functions.*

### Functions

- double2 \* [FileIO::readIn](#) (char \*, char \*, int, int)
- int [FileIO::readState](#) (char \*)
- void [FileIO::writeOut](#) (char \*, char \*, double2 \*, int, int)
- void [FileIO::writeOutDouble](#) (char \*, char \*, double \*, int, int)
- void [FileIO::writeOutInt](#) (char \*, char \*, int \*, int, int)
- void [FileIO::writeOutInt2](#) (char \*, char \*, int2 \*, int, int)
- void [FileIO::writeOutParam](#) (char \*, [Array](#), char \*)
- void [FileIO::writeOutVortex](#) (char \*, char \*, struct [Tracker::Vortex](#) \*, int, int)

## 6.18 fileIO.h

```

00001 /*** fileIO.h - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

```

```

00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034 #ifndef FILEIO_H
00035 #define FILEIO_H
00036 #include "../include/ds.h"
00037 #include "../include/tracker.h"
00038
00039 namespace FileIO{
00040     double2* readIn(char*, char*, int, int);
00041     void writeOut(char*, char*, double2*, int, int );
00042     void writeOutDouble(char*, char*, double*, int, int);
00043     void writeOutInt(char*, char*, int*, int, int);
00044     void writeOutInt2(char*, char*, int2*, int, int);
00045     void writeOutVortex(char*, char*, struct Tracker::Vortex*, int, int);
00046     void writeOutParam(char*, char*, Array, char*);
00047     int readState(char*);
00048 }
00049 #endif
00050

```

## 6.19 include/kernels.h File Reference

#include <stdio.h>

Include dependency graph for kernels.h: This graph shows which files directly or indirectly include this file:

### Functions

- `__global__ void angularOp` (double, double, double2 \*, double \*, double2 \*)
- `__device__ double2 braKetMult` (double2 in1, double2 in2)
- `__global__ void cMult` (cufftDoubleComplex \*, cufftDoubleComplex \*, cufftDoubleComplex \*)
- `__global__ void cMultDensity` (double2 \*, double2 \*, double2 \*, double, double, double, int, int)
- `__device__ double complexMagnitudeSquared` (double2)
- `__device__ double2 conjugate` (double2 in)
- `__global__ void energyCalc` (double2 \*wfc, double2 \*op, double dt, double2 \*energy, int grnd\_state, int op\_space, double sqrt\_omegaz\_mass)
- `__device__ unsigned int getBid3d3d` ()
- `unsigned int getGid3d3d` ()
- `__device__ unsigned int getTid3d3d` ()
- `__global__ void multipass` (cufftDoubleComplex \*, cufftDoubleComplex \*, int)
- `__global__ void pinVortex` (cufftDoubleComplex \*, cufftDoubleComplex \*, cufftDoubleComplex \*)
- `__global__ void pSum` (double \*in1, double \*output, int pass)
- *Routine for parallel summation.*
- `__device__ double2 realCompMult` (double scalar, double2 comp)
- `__global__ void reduce` (double2 \*, double \*)
- `__global__ void scalarDiv` (double2 \*, double, double2 \*)
- *Divides both components of vector type "in", by the value "factor".*
- `__global__ void scalarDiv1D` (double2 \*, double2 \*)
- `__global__ void scalarDiv2D` (double2 \*, double2 \*)
- `__global__ void scalarDiv_wfcNorm` (double2 \*, double, double2 \*, double2 \*)
- *As above, but normalises for wfc.*

## 6.19.1 Function Documentation

### 6.19.1.1 `__global__ void angularOp ( double , double , double2 * , double * , double2 * )`

Definition at line 153 of file [kernels.cu](#).

References [getGid3d3d\(\)](#), and [result](#).

```
00153
00154     unsigned int gid = getGid3d3d();
00155     double2 result;
00156     double op;
00157     op = exp( -omega*xpyypx[gid]*dt);
00158     result.x=wfc[gid].x*op;
00159     result.y=wfc[gid].y*op;
00160     out[gid]=result;
00161 }
```

Here is the call graph for this function:

### 6.19.1.2 `__device__ double2 braKetMult ( double2 in1, double2 in2 ) [inline]`

Definition at line 88 of file [kernels.cu](#).

References [complexMultiply\(\)](#), and [conjugate\(\)](#).

Referenced by [energyCalc\(\)](#).

```
00089 {
00090     return complexMultiply(conjugate(in1),in2);
00091 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

### 6.19.1.3 `__global__ void cMult ( cufftDoubleComplex * , cufftDoubleComplex * , cufftDoubleComplex * )`

### 6.19.1.4 `__global__ void cMultDensity ( double2 * , double2 * , double2 * , double , double , double , int , int )`

Definition at line 104 of file [kernels.cu](#).

References [complexMagnitudeSquared\(\)](#), [HBAR](#), [mass](#), [PI](#), [result](#), [x](#), and [y](#).

```
00104
00105     {
00106     double2 result;
00107     double gDensity;
00108     int tid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x*blockDim.x + threadIdx.x;
00109     gDensity = N*complexMagnitudeSquared(in2[tid])*4*
00110     HBAR*HBAR*PI*(4.67e-9/mass)*sqrt(mass*(omegaZ)/(2*PI*
00111     HBAR));
00112
00113     if(gstate == 0){
00114         double tmp = in1[tid].x*exp(-gDensity*(dt/HBAR));
00115         result.x = (tmp)*in2[tid].x - (in1[tid].y)*in2[tid].y;
00116         result.y = (tmp)*in2[tid].y + (in1[tid].y)*in2[tid].x;
00117     }
00118     else{
00119         double2 tmp;
00120         tmp.x = in1[tid].x*cos(-gDensity*(dt/HBAR)) - in1[tid].y*sin(-gDensity*(
00121         dt/HBAR));
00122         tmp.y = in1[tid].y*cos(-gDensity*(dt/HBAR)) + in1[tid].x*sin(-gDensity*(
00123         dt/HBAR));
00124         result.x = (tmp.x)*in2[tid].x - (tmp.y)*in2[tid].y;
00125         result.y = (tmp.x)*in2[tid].y + (tmp.y)*in2[tid].x;
00126     }
00127     out[tid] = result;
00128 }
```

Here is the call graph for this function:

#### 6.19.1.5 \_\_device\_\_ double complexMagnitudeSquared ( double2 )

Definition at line 74 of file [kernels.cu](#).

Referenced by [cMultDensity\(\)](#), and [energyCalc\(\)](#).

```
00074                                     {
00075     return in.x*in.x + in.y*in.y;
00076 }
```

Here is the caller graph for this function:

#### 6.19.1.6 \_\_device\_\_ double2 conjugate ( double2 in )

Definition at line 56 of file [kernels.cu](#).

References [in\(\)](#), and [result](#).

Referenced by [braKetMult\(\)](#).

```
00056                                     {
00057     double2 result = in;
00058     result.y = -result.y;
00059     return result;
00060 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 6.19.1.7 \_\_global\_\_ void energyCalc ( double2 \* wfc, double2 \* op, double dt, double2 \* energy, int gnd\_state, int op\_space, double sqrt\_omegaz\_mass )

Definition at line 193 of file [kernels.cu](#).

References [braKetMult\(\)](#), [complexMagnitudeSquared\(\)](#), [dt](#), [gDenConst](#), [getGid3d3d\(\)](#), [HBAR](#), [realCompMult\(\)](#), [result](#), and [x](#).

```
00193                                     {
00194     unsigned int gid = getGid3d3d();
00195     double hbar_dt = HBAR/dt;
00196     double g_local = 0.0;
00197     double2 result;
00198     double opLocal;
00199     if(op_space)
00200         g_local = gDenConst*sqrt_omegaz_mass*complexMagnitudeSquared(
wfc[gid]);
00201     if(!gnd_state){
00202         opLocal = -log(op[gid].x + g_local)*hbar_dt;
00203     }
00204     else{
00205         opLocal = cos(op[gid].x + g_local)*hbar_dt;
00206     }
00207     result = braKetMult(wfc[gid], realCompMult(opLocal,
wfc[gid]));
00208     //printf("oplocal=%e Resx=%e Resy=%e\n",opLocal,result.x,result.y);
00209     energy[gid].x += result.x;
00210     energy[gid].y += result.y;
00211 }
```

Here is the call graph for this function:

#### 6.19.1.8 \_\_device\_\_ unsigned int getBid3d3d ( )

Definition at line 46 of file [kernels.cu](#).

```
00046                                     {
00047     return blockIdx.x + gridDim.x*(blockIdx.y + gridDim.y * blockIdx.z);
00048 }
```



## 6.19.1.9 unsigned int getGid3d3d ( )

Definition at line 41 of file [kernels.cu](#).

Referenced by [angularOp\(\)](#), [cMult\(\)](#), [energyCalc\(\)](#), [multipass\(\)](#), [pSum\(\)](#), [pSumT\(\)](#), [scalarDiv\(\)](#), and [scalarDiv\\_wfc-Norm\(\)](#).

```
00041                                     {
00042     return blockDim.x * ( ( blockDim.y * ( ( blockIdx.z * blockDim.z + threadIdx.z ) + blockIdx.y ) +
        threadIdx.y ) + blockIdx.x ) + threadIdx.x;
00043 }
```

Here is the caller graph for this function:

## 6.19.1.10 \_\_device\_\_ unsigned int getTid3d3d ( )

Definition at line 52 of file [kernels.cu](#).

```
00052                                     {
00053     return blockDim.x * ( blockDim.y * ( blockDim.z + ( threadIdx.z * blockDim.y ) ) + threadIdx.y ) +
        threadIdx.x;
00054 }
```

## 6.19.1.11 \_\_global\_\_ void multipass ( cufftDoubleComplex \*, cufftDoubleComplex \*, int )

## 6.19.1.12 \_\_global\_\_ void pinVortex ( cufftDoubleComplex \*, cufftDoubleComplex \*, cufftDoubleComplex \* )

## 6.19.1.13 \_\_global\_\_ void pSum ( double \* in1, double \* output, int pass )

Routine for parallel summation.

Can be looped over from host.

Definition at line 239 of file [kernels.cu](#).

References [getGid3d3d\(\)](#), and [vis::i](#).

```
00239                                     {
00240     unsigned int tid = threadIdx.x;
00241     unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x; // printf("bid0=%d\n",bid);
00242     unsigned int gid = getGid3d3d();
00243     extern __shared__ double sdata2[];
00244     for(int i = blockDim.x>>1; i > 0; i>>=1){
00245         if(tid < blockDim.x>>1){
00246             sdata2[tid] += sdata2[tid + i];
00247         }
00248         __syncthreads();
00249     }
00250     if(tid==0){
00251         output[bid] = sdata2[0];
00252     }
00253 }
```

Here is the call graph for this function:

## 6.19.1.14 \_\_device\_\_ double2 realCompMult ( double scalar, double2 comp )

Definition at line 62 of file [kernels.cu](#).

References [result](#).

Referenced by [energyCalc\(\)](#).

```
00062                                     {
00063     double2 result;
```

```

00064     result.x = scalar * comp.x;
00065     result.y = scalar * comp.y;
00066     return result;
00067 }

```

Here is the caller graph for this function:

**6.19.1.15** `__global__ void reduce ( double2 *, double * )`

**6.19.1.16** `__global__ void scalarDiv ( double2 * in, double factor, double2 * out )`

Divides both components of vector type "in", by the value "factor".

Results given with "out"

Definition at line 130 of file [kernels.cu](#).

References [getGid3d3d\(\)](#), and [result](#).

```

00130                                     {
00131     double2 result;
00132     //extern __shared__ double2 tmp_in[];
00133     unsigned int gid = getGid3d3d();
00134     result.x = (in[gid].x*factor);
00135     result.y = (in[gid].y*factor);
00136     out[gid] = result;
00137 }

```

Here is the call graph for this function:

**6.19.1.17** `__global__ void scalarDiv1D ( double2 *, double2 * )`

**6.19.1.18** `__global__ void scalarDiv2D ( double2 *, double2 * )`

**6.19.1.19** `__global__ void scalarDiv_wfcNorm ( double2 *, double, double2 *, double2 * )`

As above, but normalises for wfc.

Definition at line 142 of file [kernels.cu](#).

References [getGid3d3d\(\)](#), [result](#), [x](#), and [y](#).

```

00142                                     {
00143     unsigned int gid = getGid3d3d();
00144     double2 result;
00145     double norm = sqrt((pSum[0].x + pSum[0].y)*dr);
00146     result.x = (in[gid].x/norm);
00147     result.y = (in[gid].y/norm);
00148     out[gid] = result;
00149 }

```

Here is the call graph for this function:

## 6.20 kernels.h

```

00001 /** kernels.h - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012

```

```

00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034 #ifndef KERNELS_H
00035 #define KERNELS_H
00036 #include<stdio.h>
00037 /* CUDA function declarations */
00038
00039 unsigned int  getGid3d3d();
00040
00041 __device__ unsigned int  getBid3d3d();
00042 __device__ unsigned int  getTid3d3d();
00043 __device__ double  complexMagnitudeSquared(double2);
00044 __device__ double  complexMagnitudeSquared(double2);
00045 __global__ void  cMult(cufftDoubleComplex*, cufftDoubleComplex*, cufftDoubleComplex*);
00046 __global__ void  pinVortex(cufftDoubleComplex*, cufftDoubleComplex*, cufftDoubleComplex*);
00047 __global__ void  cMultDensity(double2*, double2*, double2*, double, double, double, int, int);
00048 __global__ void  scalarDiv(double2*, double, double2*);
00049 __global__ void  scalarDiv1D(double2*, double2*);
00050 __global__ void  scalarDiv2D(double2*, double2*);
00051 __global__ void  scalarDiv_wfcNorm(double2*, double, double2*, double2*);
00052 __global__ void  reduce(double2*, double*);
00053 __global__ void  multipass(cufftDoubleComplex*, cufftDoubleComplex*, int);
00054 __global__ void  angularOp(double, double, double2*, double*, double2*);
00055
00056
00057 //#####
00058 //
00059
00060 __device__ double2  conjugate(double2 in);
00061 __device__ double2  realCompMult(double scalar, double2 comp);
00062 __global__ void  energyCalc(double2 *wfc, double2 *op, double dt, double2 *energy, int
gnd_state, int op_space, double sqrt_omegaz_mass);
00063 inline __device__ double2  braKetMult(double2 in1, double2 in2);
00064 //template<typename T> __global__ void pSumT(T* in1, T* output, int pass);
00065 __global__ void pSum(double* in1, double* output, int pass);
00066 //template<double> __global__ void pSumT(double* in1, double* output, int pass);
00067
00068 #endif

```

## 6.21 include/minions.h File Reference

```

#include <cuda.h>
#include <stdio.h>
#include <math.h>
#include <cuda_runtime.h>
#include "tracker.h"

```

Include dependency graph for minions.h: This graph shows which files directly or indirectly include this file:

### Namespaces

- [Minions](#)

### Functions

- double2 [Minions::complexDiv](#) (double2 num, double2 den)

- double `Minions::complexMag` (double2 `in`)
  - double `Minions::complexMag2` (double2 `in`)
  - double2 `Minions::complexMult` (double2 `in1`, double2 `in2`)
  - double2 `Minions::complexScale` (double2 `comp`, double `scale`)
  - double2 `Minions::conj` (double2 `c`)
  - void `Minions::coordSwap` (struct `Tracker::Vortex` `*vCoords`, int `src`, int `dest`)
  - double `Minions::fInvSqRt` (double)
- id magic hackery*
- double `Minions::maxValue` (double `*`, int)
  - double `Minions::minValue` (double `*`, int)
  - double `Minions::psi2` (double2)
  - double `Minions::sumAvg` (double `*in`, int `len`)

## 6.22 minions.h

```

00001 /** minions.h - GPU: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034 #ifndef MINIONS_H
00035 #define MINIONS_H
00036
00037 #include <cuda.h>
00038 #include <stdio.h>
00039 #include <math.h>
00040 #include <cuda_runtime.h>
00041 #include "tracker.h"
00042
00043 namespace Minions{
00044     /* Returns |x|^2 of the double2 arg*/
00045     double psi2(double2);
00046
00047     /* Returns the minimum and maximum values in the array*/
00048     double minValue(double*,int);
00049     double maxValue(double*,int);
00050
00051     /* Computes average of the array*/
00052     double sumAvg(double* in, int len);
00053
00054     double fInvSqRt(double);
00055     //float fInvSqRt(float);
00056
00057     void coordSwap(struct Tracker::Vortex *vCoords, int src, int dest);
00058     double complexMag(double2 in);
00059     double complexMag2(double2 in);
00060     double2 complexMult(double2 in1, double2 in2);
00061     double2 complexScale(double2 comp, double scale);

```

```

00063     double2 conj(double2 c);
00064     double2 complexDiv(double2 num, double2 den);
00065 }
00066
00067 #endif
00068
00069
00070
00071
00072
00073
00074

```

## 6.23 include/split\_op.h File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <cufft.h>
#include <ctype.h>
#include <getopt.h>
#include "tracker.h"

```

Include dependency graph for split\_op.h: This graph shows which files directly or indirectly include this file:

### Functions

- double [energy\\_angmom](#) (double \*[Energy](#), double \*[Energy\\_gpu](#), double2 \*[V\\_op](#), double2 \*[K\\_op](#), double [dx](#), double [dy](#), double2 \*[gpuWfc](#), int [gState](#))  
*Calculates energy and angular momentum of current state.*
- int [isError](#) (int, char \*)
- void [optLatSetup](#) (struct [Tracker::Vortex](#) centre, double \*[V](#), struct [Tracker::Vortex](#) \*[vArray](#), int [num\\_vortices](#), double [theta\\_opt](#), double [intensity](#), double \*[v\\_opt](#), double \*[x](#), double \*[y](#))  
*Matches the optical lattice to the vortex lattice.*
- void [parSum](#) (double2 \*, double2 \*, int, int, int)

### Variables

- double [a\\_s](#)
- int [ang\\_mom](#) = 0
- long [atoms](#)
- double [dt](#)
- double [dx](#)
- double [dy](#)
- cufftDoubleComplex \* [AppliedField](#)
- cufftDoubleComplex \* [EK](#)
- double \* [Energy](#)
- double \* [Energy\\_gpu](#)
- cudaError\_t [err](#)
- long [esteps](#)
- cufftDoubleComplex \* [EV](#)
- cufftDoubleComplex \* [EV\\_opt](#)
- cufftDoubleComplex \* [ExPy](#)
- cufftDoubleComplex \* [EyPx](#)

- double [gdt](#)
- cufftDoubleComplex \* [GK](#)
- int [gpe](#) = 0
- dim3 [grid](#)
- long [gsteps](#)
- cufftDoubleComplex \* [GV](#)
- cufftDoubleComplex \* [GV\\_half](#)
- cufftDoubleComplex \* [GxPy](#)
- cufftDoubleComplex \* [GyPx](#)
- double [interaction](#)
- double \* [K](#)
- cufftDoubleComplex \* [K\\_gpu](#)
- double [l](#)
- double [laser\\_power](#)
- double [mass](#)
- double [omegaX](#)
- double [omegaY](#)
- double [omegaZ](#)
- cufftDoubleComplex \* [par\\_sum](#)
- struct Params \* [paramS](#)
- double \* [Phi](#)
- cufftHandle [plan\\_1d](#)
- cufftHandle [plan\\_2d](#)
- int [print](#)
- double \* [px](#)
- double \* [py](#)
- double \* [r](#)
- int [read\\_wfc](#)
- cufftResult [result](#)
- cudaStream\_t [streamA](#)
- cudaStream\_t [streamB](#)
- cudaStream\_t [streamC](#)
- cudaStream\_t [streamD](#)
- int [threads](#)
- double \* [V](#)
- cufftDoubleComplex \* [V\\_gpu](#)
- double \* [V\\_opt](#)
- cufftDoubleComplex \* [wfc](#)
- cufftDoubleComplex \* [wfc0](#)
- cufftDoubleComplex \* [wfc\\_backup](#)
- cufftDoubleComplex \* [wfc\\_gpu](#)
- int [write\\_it](#)
- double \* [x](#)
- int [xDim](#)
- double [xi](#)
- double [xMax](#)
- double \* [xp](#)
- double \* [xPy](#)
- double \* [xPy\\_gpu](#)
- double \* [y](#)
- int [yDim](#)
- double [yMax](#)
- double \* [yp](#)
- double \* [yPx](#)
- double \* [yPx\\_gpu](#)

### 6.23.1 Function Documentation

**6.23.1.1** `double energy_angmom ( double * Energy, double * Energy_gpu, double2 * V_op, double2 * K_op, double dx, double dy, double2 * gpuWfc, int gState )`

Calculates energy and angular momentum of current state.

Definition at line 655 of file [split\\_op.cu](#).

References [vis::i](#), [result](#), [xDim](#), and [yDim](#).

```
00655
00656         {
00657     double renorm_factor_2d=1.0/pow(xDim*yDim,0.5);
00658     double result=0;
00659     for (int i=0; i < xDim*yDim; ++i){
00660         Energy[i] = 0.0;
00661     }
00662
00663
00664 /*  cudaMalloc((void**) &energy_gpu, sizeof(double2) * xDim*yDim);
00665
00666     energyCalc<<<grid,threads>>>( gpuWfc, V_op, 0.5*dt, energy_gpu, gState,1,i 0.5*sqrt(omegaZ/mass));
00667     result = cufftExecZ2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_FORWARD );
00668     scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise
00669
00670     energyCalc<<<grid,threads>>>( gpuWfc, K_op, dt, energy_gpu, gState,0, 0.5*sqrt(omegaZ/mass));
00671     result = cufftExecZ2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_INVERSE );
00672     scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise
00673
00674     err=cudaMemcpy(energy, energy_gpu, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);
00675
00676     for(int i=0; i<xDim*yDim; i++){
00677         result += energy[i].x;
00678         //printf("En=%E\n",result*dx*dy);
00679     }
00680     return result*dx*dy;
00681 */
00682
00683 }
```

**6.23.1.2** `int isError ( int , char * )`

Definition at line 58 of file [split\\_op.cu](#).

References [result](#).

```
00058         {
00059     if(result!=0){printf("Error has occurred for method %s with return type %d\n",
00060         c,result);
00061         exit(result);
00062     }
00063     return result;
00064 }
```

**6.23.1.3** `void optLatSetup ( struct Tracker::Vortex centre, double * V, struct Tracker::Vortex * vArray, int num_vortices, double theta_opt, double intensity, double * v_opt, double * x, double * y )`

Matches the optical lattice to the vortex lattice.

Definition at line 596 of file [split\\_op.cu](#).

References [appendData\(\)](#), [buffer](#), [Tracker::Vortex::coords](#), [dt](#), [dx](#), [dy](#), [EV\\_opt](#), [HBAR](#), [vis::i](#), [observables::k\\_mag](#), [PI](#), [sepMinEpsilon](#), [Tracker::vortSepAvg\(\)](#), [FileIO::writeOut\(\)](#), [xDim](#), and [yDim](#).

Referenced by [evolve\(\)](#).

```
00596
00597     int i,j;
00598
00599     {
```

```

00598     double sepMin = Tracker::vortSepAvg(vArray,centre,num_vortices);
00599     sepMin = sepMin*(1 + sepMinEpsilon);
00600     appendData(&params,"Vort_sep", (double)sepMin);
00601     /*
00602     * Defining the necessary k vectors for the optical lattice
00603     */
00604     double k_mag = ((2*PI/(sepMin*dx))/(2)*(2/sqrt(3))); // Additional /2 as a result of lambda/2
    period
00605     double2* k = (double2*) malloc(sizeof(double2)*3);
00606     appendData(&params,"kmag", (double)k_mag);
00607     k[0].x = k_mag * cos(0*PI/3 + theta_opt);
00608     k[0].y = k_mag * sin(0*PI/3 + theta_opt);
00609     k[1].x = k_mag * cos(2*PI/3 + theta_opt);
00610     k[1].y = k_mag * sin(2*PI/3 + theta_opt);
00611     k[2].x = k_mag * cos(4*PI/3 + theta_opt);
00612     k[2].y = k_mag * sin(4*PI/3 + theta_opt);
00613
00614     double2 *r_opt = (double2*) malloc(sizeof(double2)*xDim);
00615
00616     /* for (int ii = 0; ii < xDim; ++ii){
00617         r_opt[ii].x = 0.0 + (xDim/sepMin)*PI*(ii-centre.coords.x)/(xDim-1);
00618         r_opt[ii].y = 0.0 + (xDim/sepMin)*PI*(ii-centre.coords.y)/(yDim-1);
00619     }
00620     */
00621     FileIO::writeOut(buffer,"r_opt",r_opt,xDim,0);
00622     appendData(&params,"k[0].x", (double)k[0].x);
00623     appendData(&params,"k[0].y", (double)k[0].y);
00624     appendData(&params,"k[1].x", (double)k[1].x);
00625     appendData(&params,"k[1].y", (double)k[1].y);
00626     appendData(&params,"k[2].x", (double)k[2].x);
00627     appendData(&params,"k[2].y", (double)k[2].y);
00628
00629     double x_shift = dx*(9+(0.5*xDim-1) - centre.coords.x);//sin(theta_opt)*(sepMin);
00630     double y_shift = dy*(0+(0.5*yDim-1) - centre.coords.y);//cos(theta_opt)*(sepMin);
00631
00632     printf("Xs=%e\nYs=%e\n",x_shift,y_shift);
00633
00634     /*#pragma omp parallel for private(j)
00635     for ( j=0; j<yDim; ++j ){
00636         for ( i=0; i<xDim; ++i ){
00637             v_opt[j*xDim + i] = intensity*(
00638                 pow( abs( cos( k[0].x*( x[i] + x_shift ) + k[0].
    y*( y[j] + y_shift ) ) ), 2)
00639                 + pow( abs( cos( k[1].x*( x[i] + x_shift ) + k[1].y*(
    y[j] + y_shift ) ) ), 2)
00640                 + pow( abs( cos( k[2].x*( x[i] + x_shift ) + k[2].y*(
    y[j] + y_shift ) ) ), 2)
00641                 /*
    y_shift ) ) ), 2)
00642                 + pow( abs( cos( k[0].x*( r_opt[i].x + x_shift ) + k[0].y*( r_opt[j].y +
    y_shift ) ) ), 2)
00643                 + pow( abs( cos( k[1].x*( r_opt[i].x + x_shift ) + k[1].y*( r_opt[j].y + y_shift )
    ) ), 2)
00644                 */
    );
00645     EV_opt[(j*xDim + i)].x=cos( -(V[(j*xDim + i)] + v_opt[j*xDim +
    i])*(dt/(2*HBAR)));
00646     EV_opt[(j*xDim + i)].y=sin( -(V[(j*xDim + i)] + v_opt[j*xDim + i])*(
    dt/(2*HBAR)));
00647     }
00648     }
00649
00650 }

```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 6.23.14 void parSum ( double2 \*, double2 \*, int, int, int )

Definition at line 572 of file `split_op.cu`.

References `dx`, `dy`, `threads`, and `yDim`.

Referenced by `evolve()`.

```

00572     {
00573         int grid_tmp = xDim*yDim;
00574         int block = grid_tmp/threads;
00575         int thread_tmp = threads;
00576         int pass = 0;
00577         while((double)grid_tmp/threads > 1.0){

```



```

00578         if(grid_tmp == xDim*yDim){
00579             multipass<<<block,threads,threads*sizeof(double2)>>>(&gpuWfc[0],&gpuParSum[0],pass);
00580         }
00581         else{
00582             multipass<<<block,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0],pass
00583         );
00584         }
00585         grid_tmp /= threads;
00586         block = (int) ceil((double)grid_tmp/threads);
00587         pass++;
00588     }
00589     thread_tmp = grid_tmp;
00590     multipass<<<1,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0], pass);
00591     scalarDiv_wfcNorm<<<grid,threads>>>(gpuWfc, dx*dy, gpuParSum, gpuWfc);
00592 }

```

Here is the caller graph for this function:

## 6.23.2 Variable Documentation

### 6.23.2.1 double a\_s

Definition at line 66 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), and [initialise\(\)](#).

### 6.23.2.2 int ang\_mom = 0

Definition at line 62 of file [split\\_op.h](#).

Referenced by [main\(\)](#), and [parseArgs\(\)](#).

### 6.23.2.3 long atoms

Definition at line 74 of file [split\\_op.h](#).

Referenced by [main\(\)](#), and [parseArgs\(\)](#).

### 6.23.2.4 double dt

Definition at line 70 of file [split\\_op.h](#).

Referenced by [delta\\_define\(\)](#), [energyCalc\(\)](#), [evolve\(\)](#), [initialise\(\)](#), [optLatSetup\(\)](#), and [parseArgs\(\)](#).

### 6.23.2.5 double dx

Definition at line 75 of file [split\\_op.h](#).

Referenced by [delta\\_define\(\)](#), [evolve\(\)](#), [initialise\(\)](#), [optLatSetup\(\)](#), [parSum\(\)](#), and [Tracker::sigVOL\(\)](#).

### 6.23.2.6 double dy

Definition at line 75 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), [initialise\(\)](#), [optLatSetup\(\)](#), and [parSum\(\)](#).

### 6.23.2.7 cufftDoubleComplex \* EappliedField

Definition at line 81 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#).

#### 6.23.2.8 `cufftDoubleComplex *` EK

Definition at line 81 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [main\(\)](#).

#### 6.23.2.9 `double*` Energy

Definition at line 82 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#).

#### 6.23.2.10 `double *` Energy\_gpu

Definition at line 82 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#).

#### 6.23.2.11 `cudaError_t` err

Definition at line 58 of file [split\\_op.h](#).

Referenced by [main\(\)](#).

#### 6.23.2.12 `long` esteps

Definition at line 74 of file [split\\_op.h](#).

Referenced by [main\(\)](#), and [parseArgs\(\)](#).

#### 6.23.2.13 `cufftDoubleComplex *` EV

Definition at line 81 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), [initialise\(\)](#), and [main\(\)](#).

#### 6.23.2.14 `cufftDoubleComplex *` EV\_opt

Definition at line 81 of file [split\\_op.h](#).

Referenced by [delta\\_define\(\)](#), [evolve\(\)](#), [initialise\(\)](#), and [optLatSetup\(\)](#).

#### 6.23.2.15 `cufftDoubleComplex *` ExPy

Definition at line 81 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [main\(\)](#).

#### 6.23.2.16 `cufftDoubleComplex *` EyPx

Definition at line 81 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [main\(\)](#).

#### 6.23.2.17 double gdt

Definition at line 70 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), [initialise\(\)](#), and [parseArgs\(\)](#).

#### 6.23.2.18 cufftDoubleComplex \* GK

Definition at line 81 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [main\(\)](#).

#### 6.23.2.19 int gpe = 0

Definition at line 63 of file [split\\_op.h](#).

Referenced by [main\(\)](#), and [parseArgs\(\)](#).

#### 6.23.2.20 dim3 grid

Definition at line 95 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#).

#### 6.23.2.21 long gsteps

Definition at line 74 of file [split\\_op.h](#).

Referenced by [main\(\)](#), and [parseArgs\(\)](#).

#### 6.23.2.22 cufftDoubleComplex \* GV

Definition at line 81 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [main\(\)](#).

#### 6.23.2.23 cufftDoubleComplex \* GV\_half

Definition at line 81 of file [split\\_op.h](#).

#### 6.23.2.24 cufftDoubleComplex \* GxPy

Definition at line 81 of file [split\\_op.h](#).

#### 6.23.2.25 cufftDoubleComplex \* GyPx

Definition at line 81 of file [split\\_op.h](#).

#### 6.23.2.26 double interaction

Definition at line 91 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), and [parseArgs\(\)](#).

#### 6.23.2.27 `double * K`

Definition at line 82 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#).

#### 6.23.2.28 `cufftDoubleComplex * K_gpu`

Definition at line 85 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [main\(\)](#).

#### 6.23.2.29 `double l`

Definition at line 99 of file [split\\_op.h](#).

Referenced by [Minions::fInvSqRt\(\)](#), [initialise\(\)](#), and [parseArgs\(\)](#).

#### 6.23.2.30 `double laser_power`

Definition at line 92 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), and [parseArgs\(\)](#).

#### 6.23.2.31 `double mass`

Definition at line 66 of file [split\\_op.h](#).

Referenced by [cMultDensity\(\)](#), [evolve\(\)](#), and [initialise\(\)](#).

#### 6.23.2.32 `double omegaX`

Definition at line 66 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), [main\(\)](#), and [parseArgs\(\)](#).

#### 6.23.2.33 `double omegaY`

Definition at line 66 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), [main\(\)](#), and [parseArgs\(\)](#).

#### 6.23.2.34 `double omegaZ`

Definition at line 66 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), [initialise\(\)](#), and [parseArgs\(\)](#).

#### 6.23.2.35 `cufftDoubleComplex * par_sum`

Definition at line 85 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [main\(\)](#).

**6.23.2.36 struct Params\* paramS**

Definition at line 50 of file [split\\_op.cu](#).

**6.23.2.37 double \* Phi**

Definition at line 82 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#).

**6.23.2.38 cufftHandle plan\_1d**

Definition at line 78 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), and [initialise\(\)](#).

**6.23.2.39 cufftHandle plan\_2d**

Definition at line 78 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), and [initialise\(\)](#).

**6.23.2.40 int print**

Definition at line 73 of file [split\\_op.h](#).

Referenced by [main\(\)](#), and [parseArgs\(\)](#).

**6.23.2.41 double \* px**

Definition at line 75 of file [split\\_op.h](#).

**6.23.2.42 double \* py**

Definition at line 75 of file [split\\_op.h](#).

**6.23.2.43 double \* r**

Definition at line 82 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#).

**6.23.2.44 int read\_wfc**

Definition at line 73 of file [split\\_op.h](#).

Referenced by [main\(\)](#), and [parseArgs\(\)](#).

**6.23.2.45 cufftResult result**

Definition at line 59 of file [split\\_op.h](#).

Referenced by [angularOp\(\)](#), [cMult\(\)](#), [cMultDensity\(\)](#), [Minions::complexMult\(\)](#), [complexMultiply\(\)](#), [Minions::complexScale\(\)](#), [Minions::conj\(\)](#), [conjugate\(\)](#), [energy\\_angmom\(\)](#), [energyCalc\(\)](#), [evolve\(\)](#), [initialise\(\)](#), [isError\(\)](#), [Tracker::phaseTest\(\)](#), [realCompMult\(\)](#), [scalarDiv\(\)](#), [scalarDiv\\_wfcNorm\(\)](#), [sepAvg\(\)](#), and [Tracker::vortSepAvg\(\)](#).

#### 6.23.2.46 `cudaStream_t streamA`

Definition at line 88 of file [split\\_op.h](#).

#### 6.23.2.47 `cudaStream_t streamB`

Definition at line 88 of file [split\\_op.h](#).

#### 6.23.2.48 `cudaStream_t streamC`

Definition at line 88 of file [split\\_op.h](#).

#### 6.23.2.49 `cudaStream_t streamD`

Definition at line 88 of file [split\\_op.h](#).

#### 6.23.2.50 `int threads`

Definition at line 96 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [parSum\(\)](#).

#### 6.23.2.51 `double * V`

Definition at line 82 of file [split\\_op.h](#).

Referenced by [delta\\_define\(\)](#), [evolve\(\)](#), and [initialise\(\)](#).

#### 6.23.2.52 `cufftDoubleComplex * V_gpu`

Definition at line 85 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), [initialise\(\)](#), and [main\(\)](#).

#### 6.23.2.53 `double * V_opt`

Definition at line 82 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), [initialise\(\)](#), and [main\(\)](#).

#### 6.23.2.54 `cufftDoubleComplex* wfc`

Definition at line 81 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), [initialise\(\)](#), and [main\(\)](#).

#### 6.23.2.55 `cufftDoubleComplex * wfc0`

Definition at line 81 of file [split\\_op.h](#).

**6.23.2.56** `cufftDoubleComplex * wfc_backup`

Definition at line 81 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#).

**6.23.2.57** `cufftDoubleComplex* wfc_gpu`

Definition at line 85 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [main\(\)](#).

**6.23.2.58** `int write_it`

Definition at line 73 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#), and [parseArgs\(\)](#).

**6.23.2.59** `double* x`

Definition at line 75 of file [split\\_op.h](#).

Referenced by [cMultDensity\(\)](#), [energyCalc\(\)](#), [evolve\(\)](#), [initialise\(\)](#), [main\(\)](#), [scalarDiv\\_wfcNorm\(\)](#), [FileIO::writeOut\(\)](#), and [FileIO::writeOutInt2\(\)](#).

**6.23.2.60** `int xDim`

Definition at line 73 of file [split\\_op.h](#).

Referenced by [delta\\_define\(\)](#), [energy\\_angmom\(\)](#), [evolve\(\)](#), [initialise\(\)](#), [main\(\)](#), [Tracker::olPos\(\)](#), [optLatSetup\(\)](#), [parseArgs\(\)](#), and [Tracker::vortPos\(\)](#).

**6.23.2.61** `double xi`

Definition at line 67 of file [split\\_op.h](#).

Referenced by [evolve\(\)](#).

**6.23.2.62** `double xMax`

Definition at line 75 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#).

**6.23.2.63** `double * xp`

Definition at line 75 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#).

**6.23.2.64** `double * xPy`

Definition at line 82 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [main\(\)](#).

**6.23.2.65 double \* xPy\_gpu**

Definition at line 82 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [main\(\)](#).

**6.23.2.66 double \* y**

Definition at line 75 of file [split\\_op.h](#).

Referenced by [cMultDensity\(\)](#), [evolve\(\)](#), [Tracker::findVortex\(\)](#), [initialise\(\)](#), [main\(\)](#), [Tracker::olPos\(\)](#), [Tracker::phase-Test\(\)](#), [scalarDiv\\_wfcNorm\(\)](#), [FileIO::writeOut\(\)](#), and [FileIO::writeOutInt2\(\)](#).

**6.23.2.67 int yDim**

Definition at line 73 of file [split\\_op.h](#).

Referenced by [delta\\_define\(\)](#), [energy\\_angmom\(\)](#), [evolve\(\)](#), [initialise\(\)](#), [main\(\)](#), [optLatSetup\(\)](#), [parseArgs\(\)](#), [par-Sum\(\)](#), and [FileIO::readIn\(\)](#).

**6.23.2.68 double yMax**

Definition at line 75 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#).

**6.23.2.69 double \* yp**

Definition at line 75 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#).

**6.23.2.70 double \* yPx**

Definition at line 82 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [main\(\)](#).

**6.23.2.71 double \* yPx\_gpu**

Definition at line 82 of file [split\\_op.h](#).

Referenced by [initialise\(\)](#), and [main\(\)](#).

**6.24 split\_op.h**

```

00001 /** split_op.h - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the

```



```

00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034 #ifndef SPLIT_OP_H
00035 #define SPLIT_OP_H
00036
00037 #include <stdio.h>
00038 #include <stdlib.h>
00039 #include <math.h>
00040 #include <string.h>
00041 #include <time.h>
00042 #include <cuda.h>
00043 #include <cuda_runtime.h>
00044 #include <cuFFT.h>
00045 #include <ctype.h>
00046 #include <getopt.h>
00047 #include "tracker.h"
00048 #ifdef __linux
00049     #include <omp.h>
00050 #elif __APPLE__
00051     //printf("OpenMP support disabled due to Clang/LLVM being behind the trend.");
00052 #endif
00053
00054 /* Keep track of all params for reading/writing to file*/
00055 extern struct Params *params;
00056
00057 /* Error variable & return variables */
00058 cudaError_t err;
00059 cuFFTResult result;
00060
00061 /* Define operating modes */
00062 int ang_mom = 0;
00063 int gpe = 0;
00064
00065 /* Allocating global variables */
00066 double mass, a_s, omegaX, omegaY, omegaZ;
00067 double xi; //Healing length minimum value defined at central density.
00068
00069 /* Evolution timestep */
00070 double dt, gdt;
00071
00072 /* Grid dimensions vector. xyz are dim length, w is total grid size (x*y*z) */
00073 int xDim, yDim, read_wfc, print, write_it;
00074 long gsteps, esteps, atoms;
00075 double *x,*y,*xp,*yp,*px,*py,dx,dy,xMax,yMax;
00076
00077 /* CuFFT plans for forward and inverse. May only need to use 1 for both */
00078 cuFFTHandle plan_2d, plan_1d;
00079
00080 /* Arrays for storing wavefunction, momentum and position op, etc */
00081 cuFFTDoubleComplex *wfc, *wfc0, *wfc_backup, *GK, *GV_half, *
    GV, *EK, *EV, *EV_opt, *GxPy, *GyPx, *ExPy, *EyPx, *
    EappliedField;
00082 double *Energy, *Energy_gpu, *r, *Phi, *V, *V_opt, *K, *
    xPy, *yPx, *xPy_gpu, *yPx_gpu;
00083
00084 /* CUDA data buffers for FFT */
00085 cuFFTDoubleComplex *wfc_gpu, *K_gpu, *V_gpu, *par_sum;
00086
00087 /* CUDA streams */
00088 cudaStream_t streamA, streamB, streamC, streamD;
00089
00090 /* Scaling the interaction */
00091 double interaction;
00092 double laser_power;
00093
00094 /* Define global dim3 and threads for grid and thread dim calculation */
00095 dim3 grid;
00096 int threads;
00097
00098 /* */

```

```

00099 double l;
00100 /* Function declarations */
00101 /*
00102  * arg1 = Function result code from CUDA CUFFT calls.
00103  * arg2 = String data for name of function called. Prints value to stdout.
00104  */
00105 int isError(int, char*); //Checks to see if an error has occurred.
00106
00107 void parSum(double2* , double2* , int , int , int );
00108 void optLatSetup(struct Tracker::Vortex centre, double*
    V, struct Tracker::Vortex *vArray, int num_vortices, double theta_opt, double intensity,
    double* v_opt, double *x, double *y);
00109
00110 double energy_angmom(double* Energy, double* Energy_gpu, double2 *V_op,
    double2 *K_op, double dx, double dy, double2 *gpuWfc, int gState);
00111 #endif
00112
00113
00114 /*class SplitOp{
00115
00116 }*/

```

## 6.25 include/tracker.h File Reference

```

#include <math.h>
#include <stdio.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <complex.h>

```

Include dependency graph for tracker.h: This graph shows which files directly or indirectly include this file:

### Classes

- struct [Tracker::Vortex](#)  
*Vortex is used to track specific individual vortices. [More...](#)*

### Namespaces

- [Tracker](#)  
*See the source file for info on functions.*

### Functions

- int [Tracker::findOLMaxima](#) (int \*marker, double \*V, double radius, int xDim, double \*x)  
*Finds the maxima of the optical lattice.*
- int [Tracker::findVortex](#) (int \*marker, double2 \*wfc, double radius, int xDim, double \*x, int timestep)  
*Phase winding method to determine vortex positions.*
- void [Tracker::olPos](#) (int \*marker, int2 \*olLocation, int xDim)  
*Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.*
- int [Tracker::phaseTest](#) (int2 vLoc, double2 \*wfc, int xDim)  
*Tests the phase winding of the wavefunction, looking for vortices.*
- double [Tracker::sigVOL](#) (int2 \*vArr, int2 \*opLatt, double \*x, int numVort)
- double [Tracker::vortAngle](#) (struct [Tracker::Vortex](#) \*vortCoords, struct Vortex central, int numVort)  
*Determines the angle of the vortex lattice relative to the x-axis.*
- void [Tracker::vortArrange](#) (struct [Tracker::Vortex](#) \*vCoordsC, struct Vortex \*vCoordsP, int length)  
*Ensures the vortices are tracked and arranged in the right order based on minimum distance between previous and current positions.*
- struct Vortex [Tracker::vortCentre](#) (struct [Tracker::Vortex](#) \*cArray, int length, int xDim)

*Determines the coords of the vortex closest to the central position.*

- void `Tracker::vortPos` (int \*marker, struct `Tracker::Vortex` \*vLocation, int xDim, double2 \*wfc)

*Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.*

- struct `Vortex` \* `Tracker::vortPosDelta` (int \*cMarker, int2 \*pMarker, double \*x, double tolerance, int num-Vortices, int xDim)
- double `Tracker::vortSepAvg` (struct `Vortex` \*vArray, struct `Tracker::Vortex` centre, int length)

*Determines the vortex separation at the centre of the lattice.*

## 6.25.1 Class Documentation

### 6.25.1.1 struct Tracker::Vortex

`Vortex` is used to track specific individual vortices.

coords tracks x,y positions. sign indicates direction of vortex rotation. wind indicates the unit charge of the vortex.

Definition at line 54 of file `tracker.h`.

Collaboration diagram for `Tracker::Vortex`:

#### Class Members

int2	coords	
int	sign	
int	wind	

## 6.26 tracker.h

```

00001 /** tracker.h - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034 #ifndef TRACKER_H
00035 #define TRACKER_H
00036 #ifdef __linux
00037     #include<omp.h>
00038 #elif __APPLE__
00039 #endif
00040 #include<math.h>
00041 #include<stdio.h>
00042 #include<cuda.h>

```

```

00043 #include<cuda_runtime.h>
00044 #include<complex.h>
00045
00047 namespace Tracker{
00048
00054     struct Vortex{
00055         int2 coords;
00056         int sign;
00057         int wind;
00058     };
00059
00060     int findVortex(int*,double2*, double, int, double*, int);
00061     void vortPos(int *marker, struct Vortex *vLocation, int xDim, double2*
wfc);
00062     void olPos(int *marker, int2 *vLocation, int xDim);
00063     struct Vortex* vortPosDelta(int *cMarker, int2 *pMarker, double*
x, double tolerance, int numVortices, int xDim);
00064     struct Vortex vortCentre(struct Vortex *cArray, int length, int
xDim);
00065     double vortAngle(struct Vortex *vortCoords, struct Vortex central, int numVort);
00066     double vortSepAvg(struct Vortex *vArray, struct Vortex centre, int length);
00067     double sigVOL(int2 *vArr, int2 *opLatt, double *x, int numVort);
00068
00072     int findOLMaxima(int *marker, double *V, double radius, int
xDim, double* x);
00073     void vortArrange(struct Vortex *vCoordsC, struct Vortex *vCoordsP, int length);
00074     int phaseTest(int2 vLoc, double2* wfc, int xDim);
00075 }
00076
00077 #endif

```

## 6.27 py/hist3d.py File Reference

### Namespaces

- [hist3d](#)

### Functions

- def [hist3d.plot\\_hist\\_pcolor](#)
- def [hist3d.plot\\_xyz\\_histogram](#)

### Variables

- tuple [hist3d.c](#) = ConfigParser.ConfigParser()
- tuple [hist3d.dt](#) = (c.getfloat('Params','dt'))
- tuple [hist3d.dx](#) = (c.getfloat('Params','dx'))
- tuple [hist3d.evMaxVal](#) = int(c.getfloat('Params','esteps'))
- tuple [hist3d.gndMaxVal](#) = int(c.getfloat('Params','gsteps'))
- tuple [hist3d.incr](#) = int(c.getfloat('Params','print\_out'))
- int [hist3d.num\\_vort](#) = 0
- tuple [hist3d.sep](#) = (c.getfloat('Params','dx'))
- tuple [hist3d.xDim](#) = int(c.getfloat('Params','xDim'))
- tuple [hist3d.xMax](#) = (c.getfloat('Params','xMax'))
- tuple [hist3d.yDim](#) = int(c.getfloat('Params','yDim'))
- tuple [hist3d.yMax](#) = (c.getfloat('Params','yMax'))

## 6.28 hist3d.py

```

00001 '''
00002 hist3d.py - GPUE: Split Operator based GPU solver for Nonlinear
00003 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00004 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00005

```

```

00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 '''
00033
00034 from mpl_toolkits.mplot3d import Axes3D
00035 import matplotlib.pyplot as plt
00036 import numpy as np
00037 from numpy import genfromtxt
00038 import math as m
00039 import ConfigParser
00040
00041 c = ConfigParser.ConfigParser()
00042 c.readfp(open(r'Params.dat'))
00043
00044 xDim = int(c.getfloat('Params','xDim'))
00045 yDim = int(c.getfloat('Params','yDim'))
00046 gndMaxVal = int(c.getfloat('Params','gsteps'))
00047 evMaxVal = int(c.getfloat('Params','esteps'))
00048 incr = int(c.getfloat('Params','print_out'))
00049 sep = (c.getfloat('Params','dx'))
00050 dx = (c.getfloat('Params','dx'))
00051 dt = (c.getfloat('Params','dt'))
00052 xMax = (c.getfloat('Params','xMax'))
00053 yMax = (c.getfloat('Params','yMax'))
00054 num_vort = 0#int(c.getfloat('Params','Num_vort'))
00055
00056 sep=1.0
00057 def plot_xyz_histogram(start,fin,incr, barcolor):
00058     fig = plt.figure()
00059     ax = Axes3D(fig)
00060     data = []
00061     for i in range(start, fin, incr):
00062         v_arr=genfromtxt('vort_lsq_ ' + str(i) + '.csv',delimiter=',')
00063         datatmp=[]
00064         count=0
00065
00066         for i1 in range(0,v_arr.size/2):
00067             for i2 in range(i1,v_arr.size/2):
00068                 datatmp.append(m.sqrt( abs(v_arr[i1][0]*sep - v_arr[i2][0]*sep)**2 + abs(v_arr[i1][1]*sep
- v_arr[i2][1]*sep)**2 ))
00069                 count = count + 1
00070             hist=np.histogram(datatmp,bins=np.arange(1.0,m.sqrt(xDim**2 + yDim**2),1.0))
00071             data.append(hist[:,0])
00072             """ Takes in a matrix (see structure above) and generate a pseudo-3D histogram by overlaying close,
semitransparent bars. """
00073         for time, occurrence in zip(range(len(data)), data):
00074             dist = range(len(occurrence))
00075             barband = range(-45, 45, 5)
00076             #for modifier in barband:
00077                 ax.bar(dist, occurrence, zs=time, zdir='y', color=np.random.rand(3,1), alpha=0.8)
00078                 #ax.bar(current, occurrence, zs=duraton+(float(modifier)/100), zdir='y',
color=np.random.rand(3,1), alpha=0.6)
00079
00080         ax.set_xlabel('Dist')
00081         ax.set_ylabel('Time')
00082         ax.set_zlabel('Occurrances')
00083
00084         plt.savefig("HIST_N.pdf")
00085         plt.show()
00086
00087 def plot_hist_pcolor(start,fin,incr, barcolor):
00088     fig = plt.figure()
00089

```

```

00090     data =[]
00091     for i in range(start, fin, incr):
00092         v_arr=genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',')
00093         datatmp=[]
00094         count=0
00095
00096         for i1 in range(0,v_arr.size/2):
00097             for i2 in range(i1,v_arr.size/2):
00098                 m_tmp = m.sqrt(abs(v_arr[i1][0]*sep - v_arr[i2][0]*sep)**2 + abs(v_arr[i1][1]*sep - v_arr
00099 [i2][1]*sep)**2 )
00099                 datatmp.append( m_tmp )
00100                 count = count + 1
00101                 hist=np.histogram(datatmp,bins=np.arange(0.0,240.0,0.1))
00102                 data.append(hist[:][0])
00103
00104             # print data
00105             ax = fig.add_subplot(111)
00106             ax.imshow(data)
00107             plt.gca().invert_yaxis()
00108             ax.set_aspect('auto')
00109             # plt.jet()
00110             fig.savefig("HIST_PCOLOR.pdf")
00111
00112 #plot_xyz_histogram(0,100000,100,'b')
00113 #plot_hist_pcolor(0,100000,100,'b')
00114

```

## 6.29 py/hist\_it.py File Reference

### Namespaces

- [hist\\_it](#)

## 6.30 hist\_it.py

```

00001 '''
00002 hist_it.py - GPUE: Split Operator based GPU solver for Nonlinear
00003 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00004 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 '''

```

## 6.31 py/observables.py File Reference

## Namespaces

- [observables](#)

## Functions

- def [observables.ang\\_mom](#)
- def [observables.dens\\_struct\\_fact](#)
- def [observables.energy\\_kinetic](#)
- def [observables.energy\\_potential](#)
- def [observables.energy\\_total](#)
- def [observables.expec\\_val\\_](#)
- def [observables.expec\\_val\\_monopole](#)
- def [observables.expec\\_val\\_quadrupole](#)
- def [observables.kinertrum](#)  
*Kinetic energy spectrum = kinertrum.*
- def [observables.kinertrum\\_loop](#)

## Variables

- tuple [observables.c](#) = ConfigParser.ConfigParser()
- tuple [observables.data](#) = numpy.ndarray(shape=(xDim,yDim))
- tuple [observables.dkx](#) = (c.getfloat('Params','dpx'))
- tuple [observables.dky](#) = (c.getfloat('Params','dpy'))
- tuple [observables.dt](#) = (c.getfloat('Params','dt'))
- tuple [observables.dx](#) = (c.getfloat('Params','dx'))
- tuple [observables.dy](#) = (c.getfloat('Params','dy'))
- tuple [observables.evMaxVal](#) = int(c.getfloat('Params','esteps'))
- tuple [observables.g](#) = (0.5\*N)
- tuple [observables.gndMaxVal](#) = int(c.getfloat('Params','gsteps'))
- float [observables.HBAR](#) = 1.05457148e-34
- float [observables.hbar](#) = 1.05457e-34
- tuple [observables.incr](#) = int(c.getfloat('Params','print\_out'))
- tuple [observables.K](#) = np.array(open('K\_0').read().splitlines(),dtype='f8')
- tuple [observables.k\\_mag](#) = np.sqrt( kx\*\*2 + ky\*\*2 )
- tuple [observables.km\\_mag](#) = np.sqrt( kxm\*\*2 + kym\*\*2 )
- tuple [observables.kMax](#) = max(max(k\_mag))
- tuple [observables.kx](#) = np.reshape( np.array( [np.linspace( 0, (xDim/2-1)\*dkx, xDim/2), np.linspace( -xDim/2-1)\*dkx, -dkx, xDim/2]] ), (xDim,1) )
- tuple [observables.ky](#) = np.reshape( np.array( [np.linspace( 0, (yDim/2-1)\*dky, yDim/2), np.linspace( -yDim/2-1)\*dky, -dky, yDim/2]] ), (yDim,1) )
- float [observables.m](#) = 1.4431607e-25
- tuple [observables.mass](#) = (c.getfloat('Params','Mass'))
- tuple [observables.N](#) = int(c.getfloat('Params','atoms'))
- tuple [observables.num\\_vort](#) = int(c.getfloat('Params','Num\_vort'))
- tuple [observables.omega](#) = (c.getfloat('Params','omega'))
- tuple [observables.omegaX](#) = (c.getfloat('Params','omegaX'))
- tuple [observables.omegaZ](#) = (c.getfloat('Params','omegaZ'))
- float [observables.PI](#) = 3.141592653589793
- tuple [observables.V](#) = np.array(open('V\_0').read().splitlines(),dtype='f8')
- tuple [observables.x](#) = np.asarray(open('x\_0').read().splitlines(),dtype='f8')
- tuple [observables.xDim](#) = int(c.getfloat('Params','xDim'))
- tuple [observables.xMax](#) = (c.getfloat('Params','xMax'))
- tuple [observables.xPy](#) = np.array(open('xPy\_0').read().splitlines(),dtype='f8')

- tuple `observables.y` = `np.asarray(open('y_0').read().splitlines(), dtype='f8')`
- tuple `observables.yDim` = `int(c.getfloat('Params', 'yDim'))`
- tuple `observables.yMax` = `(c.getfloat('Params', 'yMax'))`
- tuple `observables.yPx` = `np.array(open('yPx_0').read().splitlines(), dtype='f8')`

## 6.32 observables.py

```

00001 '''
00002 observables.py - GPU: Split Operator based GPU solver for Nonlinear
00003 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00004 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 '''
00033 import os
00034 from numpy import genfromtxt
00035 import math as m
00036 import matplotlib as mpl
00037 import numpy as np
00038 import scipy as sp
00039 import numpy.matlib
00040 mpl.use('Agg')
00041 import multiprocessing as mp
00042 from multiprocessing import Pool
00043 from multiprocessing import Process
00044 from matplotlib.ticker import ScalarFormatter
00045 import matplotlib.pyplot as plt
00046 import ConfigParser
00047 import random as r
00048 from decimal import *
00049 from scipy.spatial import Delaunay
00050
00051 HBAR = 1.05457148e-34
00052 PI = 3.141592653589793
00053
00054 getcontext().prec = 4
00055 c = ConfigParser.ConfigParser()
00056 c.readfp(open(r'Params.dat'))
00057
00058 xDim = int(c.getfloat('Params', 'xDim'))
00059 yDim = int(c.getfloat('Params', 'yDim'))
00060 gndMaxVal = int(c.getfloat('Params', 'gsteps'))
00061 evMaxVal = int(c.getfloat('Params', 'esteps'))
00062 incr = int(c.getfloat('Params', 'print_out'))
00063 #sep = (c.getfloat('Params', 'dx'))
00064 dx = (c.getfloat('Params', 'dx'))
00065 dy = (c.getfloat('Params', 'dy'))
00066 dkx = (c.getfloat('Params', 'dpx'))
00067 dky = (c.getfloat('Params', 'dpy'))
00068 dt = (c.getfloat('Params', 'dt'))
00069 xMax = (c.getfloat('Params', 'xMax'))
00070 yMax = (c.getfloat('Params', 'yMax'))
00071 omegaZ = (c.getfloat('Params', 'omegaZ'))
00072 mass = (c.getfloat('Params', 'Mass'))
00073 omega = (c.getfloat('Params', 'omega'))
00074 omegaX = (c.getfloat('Params', 'omegaX'))

```



```

00075
00076 try:
00077     num_vort = int(c.getfloat('Params','Num_vort'))
00078 except:
00079     print '!num_vort undefined!'
00080 N = int(c.getfloat('Params','atoms'))
00081
00082 data = numpy.ndarray(shape=(xDim,yDim))
00083
00084 x=np.asarray(open('x_0').read().splitlines(),dtype='f8')
00085 y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
00086 #kx=np.asarray(open('px_0').read().splitlines(),dtype='f8')
00087 #ky=np.asarray(open('py_0').read().splitlines(),dtype='f8')
00088
00089 kx = np.reshape( np.array( [np.linspace( 0, (xDim/2-1)*dkx, xDim/2), np.linspace( (-xDim/2-1)*dkx, -dkx,
xDim/2)]), (xDim,1) )
00090 ky = np.reshape( np.array( [np.linspace( 0, (yDim/2-1)*dky, yDim/2), np.linspace( (-yDim/2-1)*dky, -dky,
yDim/2)]), (yDim,1) )
00091 kxm, kym = np.meshgrid(kx,ky)
00092 km_mag = np.sqrt( kxm**2 + kym**2 )
00093 k_mag = np.sqrt( kx**2 + ky**2 )
00094 kMax = max(max(k_mag))
00095
00096 hbar = 1.05457e-34
00097 m = 1.4431607e-25
00098
00099 ## Kinetic energy spectrum = kinertrum. Calculates the spectrum for compressible and incompressible kinetic
energies.
00100 # @param Psi The wavefunction
00101 # @param dx Increment along x
00102 # @param i The current step number
00103 # @param quOn Boolean to turn on quantum kinetic energy spectrum (includes phase term).
00104 def kinertrum(Psi, dx, i, quOn):
00105
00106     kMax = np.max(np.max(kx))
00107     Psi[np.where(Psi==0)] = 1e-100
00108     n_r = np.abs(Psi)**2
00109     n_r[np.where(n_r==0)] = 1e-100
00110     cPsi = np.conj(Psi)
00111     phi = np.angle(Psi)
00112
00113     ph1 = np.unwrap(phi, axis=0)
00114     ph2 = np.unwrap(phi, axis=1)
00115
00116     vel_ph1_x, vel_ph1_y = np.gradient(ph1,dx,dy)
00117     vel_ph2_x, vel_ph2_y = np.gradient(ph2,dx,dy)
00118
00119     v_x = (hbar/m)*vel_ph1_x;
00120     v_y = (hbar/m)*vel_ph2_y;
00121     v_x[np.where(v_x==0)] = 1e-100
00122     v_y[np.where(v_y==0)] = 1e-100
00123
00124     u_x = np.multiply(np.abs(Psi),v_x)
00125     u_y = np.multiply(np.abs(Psi),v_y)
00126
00127     if quOn:
00128         u_x = np.multiply(u_x,np.exp(1j*np.angle(Psi)))
00129         u_y = np.multiply(u_y,np.exp(1j*np.angle(Psi)))
00130
00131     u_kx = np.fft.fftn(u_x)
00132     u_ky = np.fft.fftn(u_y)
00133
00134     uc_kx = ( kxm**2*u_kx + kxm*kym*u_ky ) / ( km_mag**2 + 1e-100 )
00135     uc_ky = ( kym*kxm*u_kx + kym**2*u_ky ) / ( km_mag**2 + 1e-100 )
00136
00137     ui_kx = u_kx - uc_kx
00138     ui_ky = u_ky - uc_ky
00139
00140     uc_x = np.fft.ifftn(uc_kx)
00141     uc_y = np.fft.ifftn(uc_ky)
00142     ui_x = np.fft.ifftn(ui_kx)
00143     ui_y = np.fft.ifftn(ui_ky)
00144
00145     Ec = 0.5*np.abs(np.square(uc_x) + np.square(uc_y))
00146     Ei = 0.5*np.abs(np.square(ui_x) + np.square(ui_y))
00147
00148     fig, ax = plt.subplots()
00149     f = plt.imshow((Ec),cmap=plt.get_cmap('gnuplot2'))
00150     cbar = fig.colorbar(f)
00151     plt.gca().invert_yaxis()
00152     plt.savefig("Ec_" + str(i/incr) + ".png",dpi=200)
00153     plt.close()
00154     fig, ax = plt.subplots()
00155     f = plt.imshow((Ei),cmap=plt.get_cmap('gnuplot2'))
00156     cbar = fig.colorbar(f)
00157     plt.gca().invert_yaxis()
00158     plt.savefig("Ei_" + str(i/incr) + ".png",dpi=200)

```

```

00159 plt.close()
00160
00161 print Ec
00162 #exit()
00163 ekc = np.zeros((xDim/2-1,1))
00164 eki = np.zeros((xDim/2-1,1))
00165 for i1 in np.arange(0,np.size(k_mag)/2 -2):
00166     iX = np.array(np.where(np.logical_and( k_mag[i1] >= km_mag, k_mag[i1+1] < km_mag )))
00167 #     Ei_kx = np.sum(np.sum(np.abs(ui_kx[iX]**2*k[iX])))
00168 #     Ei_ky = np.sum(np.sum(np.abs(ui_ky[iX]**2*k[iX])))
00169     ekc[i1] = (0.5*m*k_mag[i1]) * (np.sum(np.abs(uc_kx[iX]**2 + uc_ky[iX]**2))/len(iX))
00170     eki[i1] = (0.5*m*k_mag[i1]) * (np.sum(np.abs(ui_kx[iX]**2 + ui_ky[iX]**2))/len(iX))
00171 print i1
00172 np.savetxt('ekc_' + str(i) + '.csv',ekc,delimiter=',')
00173 np.savetxt('eki_' + str(i) + '.csv',eki,delimiter=',')
00174 fig, ax = plt.subplots()
00175 print eki[0:(xDim/2-2)]
00176 f = plt.loglog(np.ravel(k_mag[0:(xDim/2 -2)]),eki[0:(xDim/2-2)])
00177 plt.savefig("eki_" + str(i) + ".png",dpi=200)
00178 f = plt.loglog(np.ravel(k_mag[0:(xDim/2 -2)]),np.ravel(ekc[0:(xDim/2-2)]))
00179 plt.savefig("ekc_" + str(i) + ".png",dpi=200)
00180 plt.close()
00181
00182
00183 def kinertrum_loop(dataName, initValue, finalValue, incr):
00184     for i in range(initValue,incr*(finalValue/incr),incr):
00185         if os.path.exists(dataName + '_' + str(i)):
00186             real=open(dataName + '_' + str(i)).read().splitlines()
00187             img=open(dataName + 'i_' + str(i)).read().splitlines()
00188             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00189             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00190             a = a_r[:] + 1j*a_i[:]
00191
00192             kinertrum(np.reshape(a, (xDim,yDim)),dx,i,1)
00193
00194 def dens_struct_fact(dataName, initValue, finalValue,incr):
00195     n_k=np.zeros(finalValue/incr)
00196     n_k_t=np.zeros((finalValue/incr,xDim,yDim),dtype=np.complex128)
00197     for i in range(initValue,incr*(finalValue/incr),incr):
00198         if os.path.exists(dataName + '_' + str(i)):
00199             real=open(dataName + '_' + str(i)).read().splitlines()
00200             img=open(dataName + 'i_' + str(i)).read().splitlines()
00201             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00202             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00203             a = a_r[:] + 1j*a_i[:]
00204             n = np.abs(a)**2
00205
00206             kinertrum(np.reshape(a, (xDim,yDim)),dx,i,0)
00207             sf = np.fft.fftfreq(np.fft.fftfreq2(np.reshape(n, (xDim,yDim))))
00208             n_k_t[i/incr][:][:] = sf[:][:];
00209             n_k[i/incr]=(abs(np.sum(np.sum(sf)))*dkx**2)
00210
00211             fig, ax = plt.subplots()
00212             f = plt.imshow(np.log10(abs(sf)),cmap=plt.get_cmap('gnuplot2'))
00213             cbar = fig.colorbar(f)
00214             plt.gca().invert_yaxis()
00215             plt.savefig("struct_" + str(i/incr) + ".png",vmin=0,vmax=12,dpi=200)
00216             plt.close()
00217             print i/incr
00218
00219     np.savetxt('Struct' + '.csv',n_k,delimiter=',')
00220     plt.plot(range(initValue,finalValue,incr),n_k)
00221     sp.io.savemat('Struct_t.mat',mdict={'n_k_t',n_k_t})
00222     plt.savefig("Struct.pdf",dpi=200)
00223     plt.close()
00224
00225 V = np.array(open('V_0').read().splitlines(),dtype='f8')
00226 V = np.reshape(V, (xDim,yDim))
00227 K = np.array(open('K_0').read().splitlines(),dtype='f8')
00228 K = np.reshape(K, (xDim,yDim))
00229 xPy = np.array(open('xPy_0').read().splitlines(),dtype='f8')
00230 xPy = np.reshape(xPy, (xDim,yDim))
00231 yPx = np.array(open('yPx_0').read().splitlines(),dtype='f8')
00232 yPx = np.reshape(yPx, (xDim,yDim))
00233 g = (0.5*N)*4.0*HBAR*HBAR*PI*(4.67e-9/mass)*np.sqrt(mass*omegaZ/(2.0*PI*HBAR))
00234
00235 def energy_total(dataName, initValue, finalValue, increment):
00236     E=np.zeros((finalValue,1))
00237     E_k=np.zeros((finalValue,1))
00238     E_vi=np.zeros((finalValue,1))
00239     E_l=np.zeros((finalValue,1))
00240     for i in range(initValue,incr*(finalValue/incr),incr):
00241         if os.path.exists(dataName + '_' + str(i)):
00242             real=open(dataName + '_' + str(i)).read().splitlines()
00243             img=open(dataName + 'i_' + str(i)).read().splitlines()
00244             a_r = np.array(real,dtype='f8') #64-bit double
00245             a_i = np.array(img,dtype='f8') #64-bit double

```

```

00246         wfc_r = np.reshape(a_r[:] + 1j*a_i[:], (xDim,yDim))
00247         wfc_p = np.array(np.fft.fft2(wfc_r))
00248         wfc_r_c = np.conj(wfc_r)
00249
00250         E1 = np.fft.ifft2(K*wfc_p)
00251         E2 = (V + 0.5*g*np.abs(wfc_r)**2)*wfc_r
00252         E3 = -(omega*omegaX)*(np.fft.ifft(xPy*np.fft.fft(wfc_r,axis=0),axis=0) - np.fft.ifft(yPx*
np.fft.fft(wfc_r,axis=1),axis=1) )
00253
00254         E_k[i/incr] = np.trapz(np.trapz(wfc_r_c*E1))*dx*dy
00255         E_vi[i/incr] = np.trapz(np.trapz(wfc_r_c*E2))*dx*dy
00256         E_l[i/incr] = np.trapz(np.trapz(wfc_r_c*E3))*dx*dy
00257         E[i/incr] = E_k[i/incr] + E_vi[i/incr] + E_l[i/incr]
00258         print (i/float(evMaxVal))
00259         np.savetxt('E_'+ str(i) + '.csv',E,delimiter=',')
00260         np.savetxt('E_k_'+ str(i) + '.csv',E_k,delimiter=',')
00261         np.savetxt('E_vi_'+ str(i) + '.csv',E_vi,delimiter=',')
00262         np.savetxt('E_l_'+ str(i) + '.csv',E_l,delimiter=',')
00263         t = np.array(range(initValue,finalValue,incr))/dt
00264         plt.plot(t,E,'r-',t,E_k,'g-',t,E_vi,'b-',t,E_l,'y-')
00265         plt.savefig("EnergyVst.pdf",dpi=200)
00266         plt.close()
00267
00268 def energy_kinetic(dataName, initValue, finalValue, increment):
00269     px1 = np.fft.fftshift(px)
00270     py1 = np.fft.fftshift(py)
00271     dk=[]
00272     dk2[:] = (px1[:]**2 + py1[:]**2)
00273     Lz = np.zeros( (finalValue/incr))
00274     for i in range(initValue,incr*(finalValue/incr),incr):
00275         if os.path.exists(dataName + '_' + str(i)):
00276             real=open(dataName + '_' + str(i)).read().splitlines()
00277             img=open(dataName + 'i_' + str(i)).read().splitlines()
00278             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00279             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00280             a = a_r[:] + 1j*a_i[:]
00281             wfc_p = np.fft.fft2(np.reshape(a, (xDim,yDim)))
00282             conjwfc_p = np.conj(wfc_p)
00283             E_k = np.zeros(len(px1))
00284             for ii in range(0,len(px1)):
00285                 E_k[ii] = np.sum( np.sum( np.multiply(wfc_p,conjwfc_p) ) ) *dk2[ii]
00286
00287             np.savetxt('E_k_'+ str(i) + '.csv',E_k,delimiter=',')
00288             print i
00289
00290 def energy_potential(dataName, initValue, finalValue, increment):
00291     print 'energy'
00292
00293 def ang_mom(dataName, initValue, finalValue, incr, ev_type, imgdpi):
00294     xm, ym = np.meshgrid(x,y)
00295     pxm, pym = np.meshgrid(px,py)
00296     dx2=dx**2
00297     Lz = np.zeros( (finalValue/incr))
00298     for i in range(initValue,incr*(finalValue/incr),incr):
00299         if os.path.exists(dataName + '_' + str(i)):
00300             real=open(dataName + '_' + str(i)).read().splitlines()
00301             img=open(dataName + 'i_' + str(i)).read().splitlines()
00302             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00303             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00304             a = a_r[:] + 1j*a_i[:]
00305             wfc = np.reshape(a, (xDim,yDim))
00306             conjwfc = np.conj(wfc)
00307
00308             wfc_ypx = np.multiply(ym,np.fft.ifft(np.multiply(pxm,np.fft.fft(wfc,axis=1)),axis=1))
00309             wfc_xpy = np.multiply(xm,np.fft.ifft(np.multiply(pym,np.fft.fft(wfc,axis=0)),axis=0))
00310             result = np.sum( np.sum( np.multiply(conjwfc,wfc_xpy - wfc_ypx) ) ) *dx2
00311         else:
00312             print "Skipped " + dataName + "_" + str(i)
00313             result = np.nan
00314
00315         print i, incr
00316         Lz[(i/incr)] = np.real(result)
00317         type=""
00318         if ev_type == 0:
00319             type = "gnd"
00320         else:
00321             type = "ev"
00322         np.savetxt('Lz.csv',Lz,delimiter=',')
00323
00324         plt.plot(Lz)
00325         plt.savefig("Lz_"+type+".pdf",dpi=imgdpi)
00326         plt.axis('off')
00327         plt.savefig("Lz_"+type+"_axis0.pdf",bbox_inches='tight',dpi=imgdpi)
00328         plt.close()
00329
00330 def expec_val_monopole(dataName, initValue, finalValue, incr):
00331     x=np.asarray(open('x_0').read().splitlines(),dtype='f8')

```

```

00332     y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
00333     # px=open('px_0')
00334     # py=open('py_0')
00335     xm, ym = np.meshgrid(x, y)
00336     result = []
00337     for i in range(initValue,finalValue,incr):
00338         if not os.path.exists(dataName):
00339             real=open(dataName + '_' + str(i)).read().splitlines()
00340             img=open(dataName + 'i_' + str(i)).read().splitlines()
00341             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00342             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00343             a = a_r[:] + 1j*a_i[:]
00344             wfc = np.reshape(a, (xDim,yDim))
00345             conjwfc = np.conj(wfc)
00346
00347             d1 = np.multiply( np.square(xm) + np.square(ym), wfc )
00348             d2 = np.multiply( conjwfc, d1)
00349             result.append( np.real( np.sum( d2 ) ) *dx*dx )
00350             print str(100*float(i)/finalValue) + '%'
00351     np.savetxt('monopole.csv',result,delimiter=',')
00352     plt.plot(range(initValue,finalValue,incr),result)
00353     plt.savefig("Monopole.png",dpi=200)
00354     plt.close()
00355
00356 def expec_val_quadrupole(dataName, initValue, finalValue, incr):
00357     x=np.asarray(open('x_0').read().splitlines(),dtype='f8')
00358     y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
00359     # px=open('px_0')
00360     # py=open('py_0')
00361     xm, ym = np.meshgrid(x, y)
00362     result = []
00363     for i in range(initValue,finalValue,incr):
00364         if not os.path.exists(dataName):
00365             real=open(dataName + '_' + str(i)).read().splitlines()
00366             img=open(dataName + 'i_' + str(i)).read().splitlines()
00367             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00368             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00369             a = a_r[:] + 1j*a_i[:]
00370             wfc = np.reshape(a, (xDim,yDim))
00371             conjwfc = np.conj(wfc)
00372
00373             d1 = np.multiply( np.square(xm) - np.square(ym), wfc )
00374             d2 = np.multiply( conjwfc, d1)
00375             result.append( np.real( np.sum( d2 ) ) *dx*dx )
00376             print str(100*float(i)/finalValue) + '%'
00377     np.savetxt('quadrupole.csv',result,delimiter=',')
00378     plt.plot(range(initValue,finalValue,incr),result)
00379     plt.savefig("Quadrupole.png",dpi=200)
00380     plt.close()
00381
00382 def expec_val_(quant_name, quantity, dataName, initValue, finalValue, incr):
00383     x=np.asarray(open('x_0').read().splitlines(),dtype='f8')
00384     y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
00385     # px=open('px_0')
00386     # py=open('py_0')
00387     xm, ym = np.meshgrid(x, y)
00388     result = []
00389     for i in range(initValue,finalValue,incr):
00390         if not os.path.exists(dataName):
00391             real=open(dataName + '_' + str(i)).read().splitlines()
00392             img=open(dataName + 'i_' + str(i)).read().splitlines()
00393             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00394             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00395             a = a_r[:] + 1j*a_i[:]
00396             wfc = np.reshape(a, (xDim,yDim))
00397             conjwfc = np.conj(wfc)
00398
00399             d1 = np.multiply( quantity, wfc )
00400             d2 = np.multiply( conjwfc, d1)
00401             result.append( np.real( np.sum( d2 ) ) *dx*dx )
00402             print str(100*float(i)/finalValue) + '%'
00403     np.savetxt(quant_name + '.csv',result,delimiter=',')
00404     plt.plot(range(initValue,finalValue,incr),result)
00405     plt.savefig(quant_name + ".pdf",dpi=200)
00406     plt.close()
00407
00408 if __name__ == '__main__':
00409     kinertrum_loop('wfc_ev', 0, evMaxVal, incr)
00410     exit()
00411     energy_total('wfc_ev',0,evMaxVal,incr)
00412     dens_struct_fact('wfc_ev', 0, evMaxVal, 500)
00413
00414     energy_kinetic('wfc_ev', 0, evMaxVal, 200)
00415     # ang_mom('wfc_0_ramp', 0, gndMaxVal, incr, 0, 200)
00416     ang_mom('wfc_ev', 0, evMaxVal, incr, 1, 200)
00417     expec_val_monopole('wfc_ev',0,evMaxVal,incr)
00418     expec_val_quadrupole('wfc_ev',0,evMaxVal,incr)

```

## 6.33 py/stats.py File Reference

### Namespaces

- [stats](#)

### Functions

- `def` [stats.lsFit](#)

### Variables

- tuple [stats.c](#) = `ConfigParser.ConfigParser()`
- tuple [stats.incr](#) = `int(c.getfloat('Params','print_out'))`
- tuple [stats.xDim](#) = `int(c.getfloat('Params','xDim'))`
- tuple [stats.yDim](#) = `int(c.getfloat('Params','yDim'))`

## 6.34 stats.py

```

00001 '''
00002 stats.py - GPUE: Split Operator based GPU solver for Nonlinear
00003 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00004 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 '''
00033 import os
00034 from numpy import genfromtxt
00035 import math as m
00036 #import matplotlib as mpl
00037 import numpy as np
00038 import numpy.matlib
00039 #mpl.use('Agg')
00040 #import multiprocessing as mp
00041 #from multiprocessing import Pool
00042 #from multiprocessing import Process
00043 #from matplotlib.ticker import ScalarFormatter
00044 #import matplotlib.pyplot as plt
00045 import ConfigParser
00046 import random as r
00047 from decimal import *
00048
00049 #getcontext().prec = 4
00050 c = ConfigParser.ConfigParser()
00051 c.readfp(open(r'Params.dat'))
00052
00053 incr = int(c.getfloat('Params','print_out'))

```

```

00054 xDim = int(c.getfloat('Params','xDim'))
00055 yDim = int(c.getfloat('Params','yDim'))
00056
00057 def lsFit(start,end,incr):
00058     L = np.matrix([
00059         [0,0,1],
00060         [1,0,1],
00061         [0,1,1],
00062         [1,1,1]
00063     ])
00064     LSQ = np.linalg.inv(np.transpose(L)*L)*np.transpose(L)
00065     for i in range(start,end,incr):
00066         v_arr=genfromtxt('vort_arr_' + str(i),delimiter=',')
00067         real=open('wfc_ev_' + str(i)).read().splitlines()
00068         img=open('wfc_evi_' + str(i)).read().splitlines()
00069         a_r = np.asanyarray(real,dtype='f8') #64-bit double
00070         a_i = np.asanyarray(img,dtype='f8') #64-bit double
00071         a = a_r[:] + 1j*a_i[:]
00072         wfc = (np.reshape(a, (xDim,yDim)))
00073
00074         indX = [row[0] for row in v_arr]
00075         indY = [row[1] for row in v_arr]
00076         wind = [row[2] for row in v_arr]
00077         sign = [row[3] for row in v_arr]
00078         data=[]
00079         for ii in range(0,len(indX)):
00080             p=np.matrix([[0],[0],[0],[0]],dtype=np.complex)
00081             p[0]=(wfc[indX[ii], indY[ii]])
00082             p[1]=(wfc[indX[ii]+1, indY[ii]])
00083             p[2]=(wfc[indX[ii], indY[ii]+1])
00084             p[3]=(wfc[indX[ii]+1, indY[ii]+1])
00085             rc = LSQ * np.real(p)
00086             ic = LSQ * np.imag(p)
00087
00088             A=np.squeeze([row[0:2] for row in [rc,ic]])
00089             B=-np.squeeze([row[2] for row in [rc,ic]])
00090             r=np.linalg.lstsq(A,B)[0]
00091             data.append([indX[ii]+r[0],indY[ii]+r[1],sign[ii]])
00092
00093 #         f = plt.imshow(abs(wfc)**2)
00094 #         plt.jet()
00095 #         plt.gca().invert_yaxis()
00096 #         plt.hold(True)
00097 #         X = [row[0] for row in data]
00098 #         Y = [row[1] for row in data]
00099 #         plt.scatter(Y,X,s=0.2,marker='.',c='red',lw=0)
00100 #         plt.scatter(indY,indX,s=0.2,marker='.',c='yellow',lw=0)
00101 #         plt.savefig("fig.png",dpi=1200)
00102 #         plt.close()
00103         np.savetxt('vort_lsq_'+str(i)+'.csv',data,delimiter=',')

```

## 6.35 py/vis.py File Reference

### Namespaces

- [vis](#)

### Functions

- [def vis.delaunay](#)
- [def vis.hist\\_gen](#)
- [def vis.image\\_gen](#)
- [def vis.image\\_gen\\_single](#)
- [def vis.laplacian](#)
- [def vis.opPot](#)
- [def vis.overlap](#)
- [def vis.scaleAxis](#)
- [def vis.struct\\_fact](#)
- [def vis.voronoi](#)
- [def vis.vort\\_traj](#)

## Variables

- tuple `vis.c` = `ConfigParser.ConfigParser()`
- list `vis.CPUs` = `os.environ['SLURM_JOB_CPUS_PER_NODE']`
- tuple `vis.data` = `numpy.ndarray(shape=(xDim,yDim))`
- tuple `vis.dt` = `(c.getfloat('Params','dt'))`
- tuple `vis.dx` = `(c.getfloat('Params','dx'))`
- list `vis.ev_proc` = `[]`
- list `vis.evImgList` = `[]`
- tuple `vis.evMaxVal` = `int(c.getfloat('Params','esteps'))`
- list `vis.gnd_proc` = `[]`
- list `vis.gndImgList` = `[]`
- tuple `vis.gndMaxVal` = `int(c.getfloat('Params','gsteps'))`
- tuple `vis.i` = `gndImgList.pop()`
- tuple `vis.incr` = `int(c.getfloat('Params','print_out'))`
- int `vis.num_vort` = `0`
- tuple `vis.p` = `proc.pop()`
- `vis.proc` = `gnd_proc+ev_proc`
- tuple `vis.sep` = `(c.getfloat('Params','dx'))`
- tuple `vis.xDim` = `int(c.getfloat('Params','xDim'))`
- tuple `vis.xMax` = `(c.getfloat('Params','xMax'))`
- tuple `vis.yDim` = `int(c.getfloat('Params','yDim'))`
- tuple `vis.yMax` = `(c.getfloat('Params','yMax'))`

## 6.36 vis.py

```

00001 '''
00002 vis.py - GPUE: Split Operator based GPU solver for Nonlinear
00003 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00004 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 '''
00033 import os
00034 CPUs = os.environ['SLURM_JOB_CPUS_PER_NODE']
00035 print "Number of cores: " + str(CPUs)
00036 from numpy import genfromtxt
00037 import math as m
00038 import matplotlib as mpl
00039 import matplotlib.tri as tri
00040 import numpy as np
00041 import scipy as sp
00042 from scipy.spatial import Voronoi, voronoi_plot_2d
00043 import numpy.matlib
00044 mpl.use('Agg')

```

```

00045 import multiprocessing as mp
00046 from multiprocessing import Pool
00047 from multiprocessing import Process
00048 from matplotlib.ticker import ScalarFormatter
00049 import matplotlib.pyplot as plt
00050 import ConfigParser
00051 import random as r
00052 from decimal import *
00053 import stats
00054 import hist3d
00055 import mpld3
00056 from mpld3 import plugins
00057
00058 getcontext().prec = 4
00059 c = ConfigParser.ConfigParser()
00060 getcontext().prec = 4
00061 c = ConfigParser.ConfigParser()
00062 c.readfp(open(r'Params.dat'))
00063
00064 xDim = int(c.getfloat('Params','xDim'))
00065 yDim = int(c.getfloat('Params','yDim'))
00066 gndMaxVal = int(c.getfloat('Params','gsteps'))
00067 evMaxVal = int(c.getfloat('Params','esteps'))
00068 incr = int(c.getfloat('Params','print_out'))
00069 sep = (c.getfloat('Params','dx'))
00070 dx = (c.getfloat('Params','dx'))
00071 dt = (c.getfloat('Params','dt'))
00072 xMax = (c.getfloat('Params','xMax'))
00073 yMax = (c.getfloat('Params','yMax'))
00074 num_vort = 0#int(c.getfloat('Params','Num_vort'))
00075
00076 data = numpy.ndarray(shape=(xDim,yDim))
00077
00078 def delaunay(dataName,dataType,value):
00079     v_arr=genfromtxt(dataName + str(value) + dataType,delimiter=',')
00080     data = np.array([[row[0],row[1]] for row in v_arr])
00081     dln = sp.spatial.Delaunay(data)
00082     plt.triplot(data[:,0],data[:,1],dln.simplices.copy(),linewidth=0.5,color='b',marker='.')
00083     plt.xlim(300,700);plt.ylim(300,700);
00084     plt.savefig('delaun_' + str(value) + '.png',dpi=200)
00085     print 'Saved Delaunay @ t=' + str(value)
00086
00087 def voronoi(dataName,dataType,value):
00088     v_arr=genfromtxt(dataName + str(value) + dataType,delimiter=',')
00089     data = [[row[0],row[1]] for row in v_arr]
00090     vor = Voronoi(data)
00091     voronoi_plot_2d(vor)
00092     plt.xlim(300,700);plt.ylim(300,700);
00093     plt.savefig('voronoi_' + str(value) + '.png',dpi=200)
00094     print 'Saved Voronoi @ t=' + str(value)
00095
00096 def laplacian(density,name,imgdpi):
00097     gx,gy = np.gradient(density)
00098     g2x,gxgy = np.gradient(gx)
00099     gygx,g2y = np.gradient(gy)
00100     fig, ax = plt.subplots()
00101     #f = plt.quiver(gx,gy)
00102     f = plt.imshow((g2x**2 + g2y**2),cmap=plt.get_cmap('spectral'))
00103     cbar = fig.colorbar(f)
00104     plt.savefig(name + "_laplacian.png",dpi=imgdpi)
00105     plt.close()
00106     f = plt.imshow((gxgy - gygx),cmap=plt.get_cmap('spectral'))
00107     cbar = fig.colorbar(f)
00108     plt.savefig(name + "_dx dy.png",dpi=imgdpi)
00109     plt.close()
00110
00111 def struct_fact(density,name,imgdpi):
00112     fig, ax = plt.subplots()
00113     #f = plt.quiver(gx,gy)
00114     f = plt.imshow((np.abs(np.fft.fftshift(np.fft.fft2(density))))),cmap=plt.get_cmap('prism'))
00115     cbar = fig.colorbar(f)
00116     cbar.set_clim(1e6,1e11)
00117     plt.jet()
00118     plt.savefig(name + "_struct_log10.png",dpi=imgdpi)
00119     plt.close()
00120
00121 def opPot(dataName,imgdpi):
00122     data = open(dataName).read().splitlines()
00123     a = numpy.asanyarray(data, dtype='f8')
00124     b = np.reshape(a, (xDim,yDim))
00125     fig, ax = plt.subplots()
00126     f = plt.imshow((b))
00127     plt.gca().invert_yaxis()
00128     cbar = fig.colorbar(f)
00129     plt.jet()
00130     plt.savefig(dataName + ".png",dpi=imgdpi)
00131     plt.close()

```



```

00132
00133 def hist_gen(name,value,num_bins):
00134     v_arr=genfromtxt('vort_arr_' + str(value),delimiter=',' )
00135     H=[]
00136     count=0
00137
00138     for i1 in range(0,v_arr.size/2):
00139         for i2 in range(i1,v_arr.size/2):
00140             H.append(m.sqrt( abs(v_arr[i1][0]*sep - v_arr[i2][0]*sep)**2 + abs(v_arr[i1][1]*sep - v_arr[
i2][1]*sep)**2 ))
00141             count = count + 1
00142     plt.title('Vortex lattice @ t=' + str(value*dt))
00143     plt.ticklabel_format(style='scientific')
00144     plt.ticklabel_format(style='scientific',axis='x', scilimits=(0,0))
00145     h = plt.hist(H, bins=num_bins)
00146     plt.savefig(name + "_" + str(value) + ".pdf")
00147     plt.close()
00148
00149 def image_gen(dataName, initValue, finalValue, increment,imgdpi):
00150     for i in range(initValue,finalValue,increment):
00151         if not os.path.exists(dataName+"r_"+str(i)+"_abspsi2.png"):
00152             real=open(dataName + '_' + str(i)).read().splitlines()
00153             img=open(dataName + 'i_' + str(i)).read().splitlines()
00154             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00155             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00156             a = a_r[:] + 1j*a_i[:]
00157             b = np.reshape(a, (xDim,yDim))
00158             f = plt.imshow(abs(b)**2)
00159             plt.jet()
00160             plt.gca().invert_yaxis()
00161             plt.savefig(dataName+"r_"+str(i)+"_abspsi2.png",dpi=imgdpi)
00162             plt.close()
00163             g = plt.imshow(np.angle(b))
00164             plt.gca().invert_yaxis()
00165             plt.savefig(dataName+"r_"+str(i)+"_phi.png",dpi=imgdpi)
00166             plt.close()
00167             f = plt.imshow(abs(np.fft.fftshift(np.fft.fft(b))**2))
00168             plt.gca().invert_yaxis()
00169             plt.jet()
00170             plt.savefig(dataName+"p_"+str(i)+"_abspsi2.png",dpi=imgdpi)
00171             plt.close()
00172             g = plt.imshow(np.angle(np.fft.fftshift(np.fft.fft(b))))
00173             plt.gca().invert_yaxis()
00174             plt.savefig(dataName+"p_"+str(i)+"_phi.png",dpi=imgdpi)
00175             plt.close()
00176             print "Saved figure: " + str(i) + ".png"
00177             plt.close()
00178         else:
00179             print "File(s) " + str(i) + ".png already exist."
00180
00181 def image_gen_single(dataName, value, imgdpi,opmode):
00182     real=open(dataName + '_' + str(0)).read().splitlines()
00183     img=open(dataName + 'i_' + str(0)).read().splitlines()
00184     al_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
00185     al_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
00186     al = al_r[:] + 1j*al_i[:]
00187     bl = np.reshape(al, (xDim,yDim))
00188
00189     if not os.path.exists(dataName+"r_"+str(value)+"_abspsi2.png"):
00190         real=open(dataName + '_' + str(value)).read().splitlines()
00191         img=open(dataName + 'i_' + str(value)).read().splitlines()
00192         a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
00193         a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
00194         a = a_r[:] + 1j*a_i[:]
00195         b = np.reshape(a, (xDim,yDim))
00196         m_val=np.max(np.abs(b)**2)
00197         #scaleAxis(b,dataName,"_abspsi2",value,imgdpi)
00198         if opmode & 0b100000 > 0:
00199             # fig, ax = plt.subplots()
00200             # plt.rc('text',usetex=True)
00201             # plt.rc('font',family='serif')
00202             f = plt.imshow((abs(b)**2 - abs(bl)**2),cmap='gnuplot2',vmin=-6,vmax=6)
00203             plt.title(r'$\left(\rho(r,t) - \rho(r,t_0)\right)$,t=$' + str(value*dt))
00204             cbar = fig.colorbar(f)
00205             plt.gca().set_xlabel('x ' + str((dx)))
00206             plt.gca().set_ylabel('y ' + str(dy))
00207             plt.gca().invert_yaxis()
00208             plt.savefig(dataName+"r_"+str(value)+"_diffabspsi2.png",dpi=imgdpi)
00209             plt.close()
00210             # plt.rc('text',usetex=True)
00211             # plt.rc('font',family='serif')
00212
00213             fig, ax = plt.subplots()
00214             f = plt.imshow((abs(b)**2),cmap='gnuplot2',vmin=0,vmax=1e7)
00215             plt.title('rho(r) @ t=' + str(value*dt))
00216             plt.title(r'$\rho \left( r,t \right)$,t=$' + str(value*dt))
00217

```

```

00218         #plugins.connect(fig, plugins.MousePosition(fontsize=14))
00219
00220         cbar = fig.colorbar(f)
00221         plt.gca().set_xlabel('x ' + str((dx)))
00222         plt.gca().set_ylabel('x ' + str(dx))
00223         plt.gca().invert_yaxis()
00224         plt.savefig(dataName+"r_"+str(value)+"_abspsi2.png", dpi=imgdpi)
00225         plt.axis('off')
00226         plt.savefig(dataName+"r_"+str(value)+"_abspsi2_axis0.pdf", bbox_inches='tight', dpi=imgdpi)
00227         plt.close()
00228
00229         if opmode & 0b010000 > 0:
00230             fig, ax = plt.subplots()
00231             g = plt.imshow(np.angle(b))
00232             cbar = fig.colorbar(g)
00233             plt.gca().invert_yaxis()
00234             plt.title('theta(r) @ t=' + str(value*dt))
00235             plt.savefig(dataName+"r_"+str(value)+"_phi.png", dpi=imgdpi)
00236             plt.close()
00237
00238         if opmode & 0b001000 > 0:
00239             fig, ax = plt.subplots()
00240             f = plt.imshow(abs(np.fft.fftshift(np.fft.fft2(b)))*2)
00241             cbar = fig.colorbar(f)
00242             plt.gca().invert_yaxis()
00243             plt.jet()
00244             plt.title('rho(p) @ t=' + str(value*dt))
00245             plt.savefig(dataName+"p_"+str(value)+"_abspsi2.png", dpi=imgdpi)
00246             plt.close()
00247
00248         if opmode & 0b000100 > 0:
00249             fig, ax = plt.subplots()
00250             g = plt.imshow(np.angle(np.fft.fftshift(np.fft.fft2(b))))
00251             cbar = fig.colorbar(g)
00252             plt.gca().invert_yaxis()
00253             plt.title('theta(p) @ t=' + str(value*dt))
00254             plt.savefig(dataName+"p_"+str(value)+"_phi.png", dpi=imgdpi)
00255             plt.close()
00256
00257         if opmode & 0b000010 > 0:
00258             struct_fact(abs(b)**2, dataName+"_" + str(value), imgdpi)
00259
00260         if opmode & 0b000001 > 0:
00261             laplacian(abs(b)**2, dataName+"_" + str(value), imgdpi)
00262
00263         print "Saved figure: " + str(value) + ".png"
00264         plt.close()
00265     else:
00266         print "File(s) " + str(value) + ".png already exist."
00267
00268 def vort_traj(name, imgdpi):
00269     evMaxVal_l = evMaxVal
00270     H=genfromtxt('vort_arr_0', delimiter=',')
00271     count=0
00272     for i1 in range(incr, evMaxVal_l, incr):
00273         try:
00274             v_arr=genfromtxt('vort_lsq_' + str(i1) + '.csv', delimiter=',')
00275             H=np.column_stack((H, v_arr))
00276         except:
00277             evMaxVal_l = i1
00278             break
00279     X=np.zeros((evMaxVal_l/incr), dtype=np.float64)
00280     Y=np.zeros((evMaxVal_l/incr), dtype=np.float64)
00281     H=np.reshape(H, ([num_vort, 2, evMaxVal_l/incr]), order='F')
00282     for i1 in range(0, num_vort):
00283         for i2 in range(0, evMaxVal_l/incr):
00284             X[i2]=(H[i1, 0, i2]*dx) - xMax
00285             Y[i2]=(H[i1, 1, i2]*dx) - yMax
00286             h = plt.plot(X, Y, color=(r.random(), r.random(), 0.85), linewidth=0.1)
00287     plt.axis('equal')
00288     plt.title('Vort(x,y) from t=0 to t='+str(evMaxVal_l*dt)+" s")
00289
00290     plt.axis((-xMax/2.0, xMax/2.0, -yMax/2.0, yMax/2.0))
00291     plt.ticklabel_format(style='scientific')
00292     plt.ticklabel_format(style='scientific', axis='x', scilimits=(0,0))
00293     plt.ticklabel_format(style='scientific', axis='y', scilimits=(0,0))
00294     plt.savefig(name + ".pdf")
00295     plt.close()
00296     print "Trajectories plotted."
00297
00298 def scaleAxis(data, dataName, label, value, imgdpi):
00299     fig, ax = plt.subplots()
00300     ax.xaxis.set_major_locator(ScaledLocator(dx=dx))
00301     ax.xaxis.set_major_formatter(ScaledLocator(dx=dx))
00302     f = plt.imshow(abs(data)**2)
00303     cbar = fig.colorbar(f)
00304     plt.gca().invert_yaxis()

```

```

00305     plt.jet()
00306     plt.savefig(dataName+"r_"+str(value)+"_"+label +".png",dpi=imgdpi)
00307     plt.close()
00308
00309 def overlap(dataName, initValue, finalValue, increment):
00310     real=open(dataName + '_' + str(0)).read().splitlines()
00311     img=open(dataName + 'i_' + str(0)).read().splitlines()
00312     a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
00313     a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
00314     wfc0 = a_r[:] + 1j*a_i[:]
00315     for i in range(initValue,finalValue,increment):
00316         real=open(dataName + '_' + str(value)).read().splitlines()
00317         img=open(dataName + 'i_' + str(value)).read().splitlines()
00318         a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
00319         a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
00320         a = a_r[:] + 1j*a_i[:]
00321         b = np.dot(wfc0,a)
00322         print i, np.sum(b)
00323
00324 if __name__ == '__main__':
00325     try:
00326         delaunay('vort_arr_',0)
00327         stats.lsFit(0,evMaxVal,incr)
00328         hist3d.plot_hist_pcolor(0,evMaxVal,incr,'b')
00329         vort_traj('traj_plot',200)
00330     except:
00331         print "Unhandled error occurred. Blame Lee."
00332     opPot('V_opt_0',200)
00333     opPot('V_0',200)
00334     opPot('K_0',200)
00335     gndImgList=[]
00336     evImgList=[]
00337     for i in range(0,gndMaxVal,incr):
00338         gndImgList.append(i)
00339     for i in range(0,evMaxVal,incr):
00340         evImgList.append(i)
00341     gnd_proc = []
00342     ev_proc = []
00343     while gndImgList:
00344         i=gndImgList.pop()
00345         gnd_proc.append(Process(target=image_gen_single,args=("wfc_0_ramp",i,200,0b110000)))
00346         gnd_proc.append(Process(target=image_gen_single,args=("wfc_0_const",i,200,0b110000)))
00347     while evImgList:
00348         i=evImgList.pop()
00349         ev_proc.append(Process(target=image_gen_single,args=("wfc_ev",i,200,0b101000)))
00350         #ev_proc.append(Process(target=mpd3.show,))
00351         ev_proc.append(Process(target=delaunay,args=("vort_lsq_",'.csv',i)))
00352         ev_proc.append(Process(target=voronoi,args=("vort_lsq_",'.csv',i)))
00353         ev_proc.append(Process(target=hist_gen,args=("hist_ev",i,128)))
00354     proc = gnd_proc + ev_proc
00355     while proc:
00356         #if (mp.cpu_count()/2) > len(mp.active_children()):
00357         if int(CPUs) > len(mp.active_children()):
00358             print len(mp.active_children())
00359             try:
00360                 p=proc.pop()
00361                 p.start()
00362             except:
00363                 print "Failed to execute ", p

```

## 6.37 py/vort.py File Reference

### Classes

- class [vort.Vortex](#)
- class [vort.VtxList](#)

### Namespaces

- [vort](#)

### Functions

- def [vort.\\_\\_init\\_\\_](#)

- def `vort.__init__`
- def `vort.add`
- def `vort.as_np`
- def `vort.dist`
- def `vort.do_the_thing`
- def `vort.element`
- def `vort.idx_min_dist`
- def `vort.max_uid`
- def `vort.remove`
- def `vort.swap_uid`
- def `vort.update_next`
- def `vort.update_on`
- def `vort.update_uid`
- def `vort.vort_decrease`
- def `vort.vort_increase`
- def `vort.vtx_uid`
- def `vort.write_out`

## Variables

- tuple `vort.c` = `ConfigParser.ConfigParser()`
- int `vort.counter` = 0
- tuple `vort.current` = `self.element(pos-1)`
- list `vort.data` = []
- tuple `vort.dcp` = `set(uid_c)`
- tuple `vort.dpc` = `set(uid_p)`
- tuple `vort.dt` = `(c.getfloat('Params','dt'))`
- list `vort.dtype` = `[('x',float),('y',float),('sign',int),('uid',int),('isOn',int)]`
- tuple `vort.dx` = `(c.getfloat('Params','dx'))`
- tuple `vort.evMaxVal` = `int(c.getfloat('Params','esteps'))`
- tuple `vort.gndMaxVal` = `int(c.getfloat('Params','gsteps'))`
- int `vort.i` = 0
- tuple `vort.incr` = `int(c.getfloat('Params','print_out'))`
- tuple `vort.index_r` = `vorts_c.idx_min_dist(vorts_p.element(i3))`
- tuple `vort.max_uid` = `vorts_p.max_uid()`
- int `vort.pos` = 0
- int `vort.pos_l` = 0
- tuple `vort.r` = `m.sqrt((self.x - vtx.x)**2 + (self.y - vtx.y)**2)`
- `vort.ret_idx` = `counter`
- list `vort.uid_c` = `[[a for a in b][3] for b in vorts_c.as_np()]`
- list `vort.uid_p` = `[[a for a in b][3] for b in vorts_p.as_np()]`
- tuple `vort.v0c` = `vorts_c.element(index_r[0])`
- tuple `vort.v0p` = `vorts_p.element(i3)`
- tuple `vort.v1c` = `vorts_c.element(index_r[0])`
- tuple `vort.v_arr_c` = `genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',')`
- tuple `vort.v_arr_c_coords` = `np.array([[a for a in v][:2] for v in v_arr_c])`
- tuple `vort.v_arr_c_sign` = `np.array([[a for a in v][2] for v in v_arr_c])`
- tuple `vort.v_arr_p` = `genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')`  
`v_arr_p=genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')`
- tuple `vort.v_arr_p_coords` = `np.array([[a for a in v][:2] for v in v_arr_p])`
- tuple `vort.v_arr_p_sign` = `np.array([[a for a in v][2] for v in v_arr_p])`
- int `vort.val` = 0
- tuple `vort.vorts_c` = `VtxList()`
- tuple `vort.vorts_c_update` = `sorted(vorts_c.as_np(),key=lambda vtx: vtx[3])`

- tuple `vort.vorts_p` = `VtxList()`
- `vort.vtx` = `self.head`
- tuple `vort.vtx_c` = `Vortex(-1-i2,v_arr_c_coords[i2][0],v_arr_c_coords[i2][1],True,sign=v_arr_c_sign[i2])`
- tuple `vort.vtx_p` = `Vortex(i1,v_arr_p_coords[i1][0],v_arr_p_coords[i1][1],True,sign=v_arr_p_sign[i1])`
- tuple `vort.vtx_pos` = `self.vtx_uid(uid_i)`
- list `vort.vtx_pos_c` = `[]`
- list `vort.vtx_pos_p` = `[]`
- tuple `vort.xDim` = `int(c.getfloat('Params','xDim'))`
- tuple `vort.xMax` = `(c.getfloat('Params','xMax'))`
- tuple `vort.yDim` = `int(c.getfloat('Params','yDim'))`
- tuple `vort.yMax` = `(c.getfloat('Params','yMax'))`

### 6.37.1 Class Documentation

#### 6.37.1.1 class vort::Vortex

Definition at line 56 of file `vort.py`.

Collaboration diagram for `vort.Vortex`:

#### 6.37.1.2 class vort::VtxList

Definition at line 90 of file `vort.py`.

Collaboration diagram for `vort.VtxList`:

## 6.38 vort.py

```

00001 '''
00002 vort.py - GPUE: Split Operator based GPU solver for Nonlinear
00003 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00004 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 '''
00033 #####
00034 import os
00035 from numpy import genfromtxt
00036 import math as m
00037 import numpy as np
00038 import copy as cp
00039 import ConfigParser
00040
00041 #####

```

```

00042 c = ConfigParser.ConfigParser()
00043 c.readfp(open(r'Params.dat'))
00044
00045 xDim = int(c.getfloat('Params','xDim'))
00046 yDim = int(c.getfloat('Params','yDim'))
00047 gndMaxVal = int(c.getfloat('Params','gsteps'))
00048 evMaxVal = int(c.getfloat('Params','esteps'))
00049 incr = int(c.getfloat('Params','print_out'))
00050 dx = (c.getfloat('Params','dx'))
00051 dt = (c.getfloat('Params','dt'))
00052 xMax = (c.getfloat('Params','xMax'))
00053 yMax = (c.getfloat('Params','yMax'))
00054
00055 #####
00056 class Vortex: #Tracks individual vortices over time.
00057 #####
00058 #####
00059     def __init__(self,uid,x,y,isOn,sign=1):
00060 #####
00061         self.uid = uid
00062         self.x = x
00063         self.y = y
00064         self.sign = sign
00065         self.isOn = isOn
00066         self.next = None
00067
00068 #####
00069     def update_uid(self,uid):
00070 #####
00071         self.uid = uid
00072
00073 #####
00074     def update_on(self,isOn): #Vortex is trackable
00075 #####
00076         self.isOn = isOn
00077
00078 #####
00079     def update_next(self,next): #Get next vortex
00080 #####
00081         self.next = next
00082
00083 #####
00084     def dist(self,vtx): #Distance between self and vtx
00085 #####
00086         r = m.sqrt((self.x - vtx.x)**2 + (self.y - vtx.y)**2)
00087         return r
00088
00089 #####
00090 class VtxList: #Linked-list for tracking vortices
00091 #####
00092 #####
00093     def __init__(self):
00094 #####
00095         self.head = None
00096         self.tail = None
00097         self.length = 0
00098
00099 #####
00100     def element(self,pos): #Get vtx at position pos
00101 #####
00102         pos_l = 0
00103         if pos < self.length:
00104             vtx = self.head
00105             while pos_l < pos:
00106                 pos_l = pos_l + 1
00107                 vtx = vtx.next
00108         else:
00109             print "Out of bounds"
00110             exit(-1)
00111         return vtx
00112
00113 #####
00114     def vtx_uid(self,uid): #Get vtx with identifier uid
00115 #####
00116         vtx = self.head
00117         pos = 0
00118         while vtx.uid != uid:
00119             vtx = vtx.next
00120             pos = pos + 1
00121         return [vtx,pos]
00122
00123 #####
00124     def max_uid(self): #Return position and value of largest uid
00125 #####
00126         val = 0
00127         vtx = self.head
00128         val = vtx.uid

```

```

00129         pos = 0
00130         #while pos < self.length:
00131         while True:
00132             vtx = vtx.next
00133             if (vtx == None):
00134                 break
00135             if vtx.uid > val:
00136                 val = vtx.uid
00137             pos = pos + 1
00138         return [val,pos]
00139
00140 #####
00141 def add(self,Vtx,index=None): #Add a vtx at index, otherwise end
00142 #####
00143     if self.length == 0:
00144         self.head = Vtx
00145         self.tail = Vtx
00146         self.length = 1
00147     elif index == None:
00148         self.tail.next = Vtx
00149         self.tail = Vtx
00150         self.length = self.length + 1
00151     else:
00152         Vtx.next = self.element(index)
00153         self.element(index-1).next = Vtx
00154         self.length = self.length + 1
00155
00156 #####
00157 def as_np(self): #Return numpy array with format x,y,sign,uid,isOn
00158 #####
00159     dtype = [('x',float),('y',float),('sign',int),('uid',int),('isOn',int)]
00160     data = []# np.array([],dtype=dtype)
00161     i = 0
00162     vtx = self.head
00163     while vtx != None:
00164         data.append([vtx.x, vtx.y, vtx.sign, vtx.uid, vtx.isOn])
00165         vtx = vtx.next
00166         i = i+1
00167     return (data)
00168
00169 #####
00170 def write_out(self,time,data): #Write out CSV file as x,y,sign,uid,isOn
00171 #####
00172     np.savetxt('vort_ord_'+str(time)+'.csv',data,fmt='%10.5f,%10.5f,%i,%i,%i',delimiter=',')
00173
00174 #####
00175 def idx_min_dist(self,vortex, isSelf=False): #Closest vtx to self
00176 #####
00177     counter = 0
00178     ret_idx = counter
00179     vtx = self.head
00180     if vtx != None:
00181         r = vtx.dist(vortex)
00182         while vtx.next != None:
00183             vtx = vtx.next
00184             counter = counter + 1
00185             if r > vtx.dist(vortex):
00186                 r = vtx.dist(vortex)
00187                 ret_idx = counter
00188     return (ret_idx,r)
00189
00190 #####
00191 def remove(self,pos): #Remove vortices outside artificial boundary
00192 #####
00193     if self.length > 1 and pos > 1:
00194         current = self.element(pos-1).next
00195         self.element(pos - 1).next = current.next
00196         current.next = None
00197         self.length = self.length - 1
00198         return current
00199     elif pos == 0:
00200         current = self.head
00201         self.head = self.head.next
00202         self.length = self.length - 1
00203         return current
00204     else:
00205         self.head = None
00206         self.length = 0
00207         return None
00208
00209 #####
00210 def swap_uid(self,uid_i,uid_f): #Swap uid between vtx
00211 #####
00212     vtx_pos = self.vtx_uid(uid_i)
00213     self.remove(pos_i)
00214     self.add(vtx,index=pos_f)
00215

```

```

00216 #####
00217 def vort_decrease(self, positions, vorts_p): #Turn off vortex timeline
00218 #####
00219     max_uid = vorts_p.max_uid()
00220     for i4 in positions:
00221         vtx = cp.copy(i4)
00222         vtx.update_on(False)
00223         vtx.update_next(None)
00224     self.add(vtx)
00225
00226 #####
00227 def vort_increase(self, positions, vorts_p): #Add new vtx to tracking
00228 #####
00229     counter = 1
00230     max_uid = vorts_p.max_uid()
00231     for i4 in positions:
00232         self.element(i4).update_uid(max_uid[0] + counter)
00233     counter = counter+1
00234
00235 #####
00236 def do_the_thing(start, fin, incr): #Performs the tracking
00237 #####
00238     #v_arr_p=genfromtxt('vort_lsq_' + str(0) + '.csv', delimiter=',')
00239     v_arr_p=genfromtxt('vort_lsq_' + str(0) + '.csv', delimiter=',')
00240     for i in range(start+incr, fin+1, incr): #loop over samples in time
00241         vorts_p = VtxList()
00242         vorts_c = VtxList()
00243         #v_arr_c=genfromtxt('vort_lsq_' + str(i) + '.csv', delimiter=',')
00244         v_arr_c=genfromtxt('vort_lsq_' + str(i) + '.csv', delimiter=',')
00245         v_arr_p_coors = np.array([[a for a in v][2] for v in v_arr_p])
00246         v_arr_c_coors = np.array([[a for a in v][2] for v in v_arr_c])
00247         v_arr_p_sign = np.array([[a for a in v][2] for v in v_arr_p])
00248         v_arr_c_sign = np.array([[a for a in v][2] for v in v_arr_c])
00249         for i1 in range(0, v_arr_p_coors.size/2): #loop over coordinates for a given time
00250             vtx_p = Vortex(i1, v_arr_p_coors[i1][0], v_arr_p_coors[i1][1], True, sign=v_arr_p_sign[i1])
00251             #, v_arr_p[i1][2])
00252             vorts_p.add(vtx_p)
00253
00254             for i2 in range(0, v_arr_c_coors.size/2):
00255                 vtx_c = Vortex(-1-i2, v_arr_c_coors[i2][0], v_arr_c_coors[i2][1], True, sign=v_arr_c_sign[
00256                     i2]) #, v_arr_c[i2][2])
00257                 vorts_c.add(vtx_c)
00258
00259                 for i3 in range(0, vorts_p.length):
00260                     index_r = vorts_c.idx_min_dist(vorts_p.element(i3))
00261
00262                     v0c = vorts_c.element(index_r[0]).sign
00263                     v0p = vorts_p.element(i3).sign
00264                     v1c = vorts_c.element(index_r[0]).uid
00265                     if (index_r[1] < 7) and (vorts_c.element(index_r[0]).sign == vorts_p.element(i3).sign) and (
00266                         vorts_c.element(index_r[0]).uid < 0):
00267                         #if (index_r[1] < 2) and (vorts_c.element(index_r[0]).sign > 0) and
00268                         (vorts_c.element(index_r[0]).uid < 0):
00269                             vorts_c.element(index_r[0]).update_uid(vorts_p.element(i3).uid)
00270                             vorts_c.element(index_r[0]).update_on(True)
00271
00272                     #You will never remember why this works
00273                     uid_c = [[a for a in b][3] for b in vorts_c.as_np()]
00274                     uid_p = [[a for a in b][3] for b in vorts_p.as_np()]
00275
00276                     #Check the difference between current and previous vtx data
00277                     dpc = set(uid_p).difference(set(uid_c))
00278                     dcp = set(uid_c).difference(set(uid_p))
00279                     vtx_pos_p=[]
00280                     vtx_pos_c=[]
00281                     for i5 in dpc:
00282                         vtx_pos_p = np.append(vtx_pos_p, vorts_p.vtx_uid(i5)[0])
00283                     for i6 in dcp:
00284                         vtx_pos_c = np.append(vtx_pos_c, vorts_c.vtx_uid(i6)[1])
00285                     if len(dpc or dcp) >= 1:
00286                         vorts_c.vort_decrease(vtx_pos_p, vorts_p)
00287                         vorts_c.vort_increase(vtx_pos_c, vorts_p)
00288
00289                     vorts_c_update=sorted(vorts_c.as_np(), key=lambda vtx: vtx[3])
00290                     vorts_c.write_out(i, np.asarray(vorts_c_update))
00291                     print "[" + str(i) + "]", "Length of previous=" + str(len(v_arr_p_coors)), "Length of current=" +
00292                         str(len(vorts_c_update))
00293                     v_arr_p=genfromtxt('vort_ord_' + str(i) + '.csv', delimiter=',')
00294 #####
00295 #####
00296 do_the_thing(0, 200000, 500)

```



## 6.39 src/ds.cc File Reference

#include "../include/ds.h"  
Include dependency graph for ds.cc:

### Functions

- void [appendData](#) ([Array](#) \*arr, char \*t, double d)
- void [freeArray](#) ([Array](#) \*arr)
- void [initArr](#) ([Array](#) \*arr, size\_t initLen)
- [Param](#) [newParam](#) (char \*t, double d)

### 6.39.1 Function Documentation

#### 6.39.1.1 void [appendData](#) ( [Array](#) \* arr, char \* t, double d )

Definition at line 42 of file [ds.cc](#).

References [Array::array](#), [Array::length](#), [newParam\(\)](#), [vis::p](#), and [Array::used](#).

Referenced by [evolve\(\)](#), [initialise\(\)](#), [optLatSetup\(\)](#), and [parseArgs\(\)](#).

```
00042                                     {
00043     Param p = newParam(t,d);
00044     if(arr->used == arr->length){
00045         arr->length *= 2;
00046         arr->array = (Param*)realloc(arr->array, arr->length*sizeof(
00047             Param));
00048     }
00049     arr->array[arr->used] = p;
00050     arr->used = arr->used + 1;
00051 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 6.39.1.2 void [freeArray](#) ( [Array](#) \* arr )

Definition at line 52 of file [ds.cc](#).

References [Array::array](#), [Array::length](#), and [Array::used](#).

```
00052                                     {
00053     free(arr->array);
00054     arr->array = NULL;
00055     arr->used = 0;
00056     arr->length = 0;
00057 }
```

#### 6.39.1.3 void [initArr](#) ( [Array](#) \* arr, size\_t initLen )

Definition at line 36 of file [ds.cc](#).

References [Array::array](#), [Array::length](#), and [Array::used](#).

Referenced by [main\(\)](#).

```
00036                                     {
00037     arr->array = (Param*) malloc(initLen*sizeof(Param));
00038     arr->used = 0;
00039     arr->length = initLen;
00040 }
```

Here is the caller graph for this function:

#### 6.39.1.4 Param newParam ( char \* t, double d )

Definition at line 59 of file [ds.cc](#).

References [Param::data](#), [vis::p](#), and [Param::title](#).

Referenced by [appendData\(\)](#).

```
00059                                     {
00060     Param p;
00061     strcpy(p.title,t);
00062     p.data = d;
00063     return p;
00064 }
```

Here is the caller graph for this function:

## 6.40 ds.cc

```
00001  /** ds.cc - GPUE: Split Operator based GPU solver for Nonlinear
00002  Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003  <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004  All rights reserved.
00005
00006  Redistribution and use in source and binary forms, with or without
00007  modification, are permitted provided that the following conditions are
00008  met:
00009
00010  1. Redistributions of source code must retain the above copyright
00011  notice, this list of conditions and the following disclaimer.
00012
00013  2. Redistributions in binary form must reproduce the above copyright
00014  notice, this list of conditions and the following disclaimer in the
00015  documentation and/or other materials provided with the distribution.
00016
00017  3. Neither the name of the copyright holder nor the names of its
00018  contributors may be used to endorse or promote products derived from
00019  this software without specific prior written permission.
00020
00021  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024  PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025  HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027  TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028  PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029  LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030  NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031  SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032  */
00033
00034  #include "../include/ds.h"
00035
00036  void initArr(Array *arr, size_t initLen){
00037     arr->array = (Param*) malloc(initLen*sizeof(Param));
00038     arr->used = 0;
00039     arr->length = initLen;
00040 }
00041
00042  void appendData(Array *arr, char* t, double d){
00043     Param p = newParam(t,d);
00044     if(arr->used == arr->length){
00045         arr->length *= 2;
00046         arr->array = (Param*)realloc(arr->array, arr->length*sizeof(
Param));
00047     }
00048     arr->array[arr->used] = p;
00049     arr->used = arr->used + 1;
00050 }
00051
00052  void freeArray(Array *arr){
00053     free(arr->array);
00054     arr->array = NULL;
00055     arr->used = 0;
00056     arr->length = 0;
00057 }
00058
00059  Param newParam(char* t,double d){
00060     Param p;
```

```

00061     strcpy(p.title,t);
00062     p.data = d;
00063     return p;
00064 }

```

## 6.41 src/fileIO.cc File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cuda_runtime.h>
#include "../include/fileIO.h"
Include dependency graph for fileIO.cc:

```

### Namespaces

- [FileIO](#)

*Check source file for information on functions.*

### Functions

- `double2 * FileIO::readIn (char *, char *, int, int)`
- `int FileIO::readState (char *)`
- `void FileIO::writeOut (char *, char *, double2 *, int, int)`
- `void FileIO::writeOutDouble (char *, char *, double *, int, int)`
- `void FileIO::writeOutInt (char *, char *, int *, int, int)`
- `void FileIO::writeOutInt2 (char *, char *, int2 *, int, int)`
- `void FileIO::writeOutParam (char *, Array, char *)`
- `void FileIO::writeOutVortex (char *, char *, struct Tracker::Vortex *, int, int)`

## 6.42 fileIO.cc

```

00001 /** fileIO.c - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */

```

```

00033
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <cuda_runtime.h>
00038 #include "../include/fileIO.h"
00039
00040 namespace FileIO{
00041
00042     /*
00043     * Reads datafile into memory.
00044     */
00045     double2* readIn(char* fileR, char* fileI, int xDim, int yDim){
00046         FILE *f;
00047         f = fopen(fileR,"r");
00048         int i = 0;
00049         double2 *arr = (double2*) malloc(sizeof(double2)*xDim*yDim);
00050         double line;
00051         while(fscanf(f,"%lE",&line) > 0){
00052             arr[i].x = line;
00053             ++i;
00054         }
00055         fclose(f);
00056         f = fopen(fileI,"r");
00057         i = 0;
00058         while(fscanf(f,"%lE",&line) > 0){
00059             arr[i].y = line;
00060             ++i;
00061         }
00062         fclose(f);
00063         return arr;
00064     }
00065
00066     /*
00067     * Writes out the parameter file.
00068     */
00069     void writeOutParam(char* buffer, Array arr, char *file){
00070         FILE *f;
00071         sprintf(buffer, "%s", file);
00072         f = fopen(file,"w");
00073         fprintf(f,"[Params]\n");
00074         for (int i = 0; i < arr.used; ++i){
00075             fprintf(f,"%s=",arr.array[i].title);
00076             fprintf(f,"%e\n",arr.array[i].data);
00077         }
00078         fclose(f);
00079     }
00080
00081     /*
00082     * Writes out double2 complex data files.
00083     */
00084     void writeOut(char* buffer, char *file, double2 *data, int length, int step){
00085         FILE *f;
00086         sprintf (buffer, "%s_%d", file, step);
00087         f = fopen (buffer,"w");
00088         int i;
00089         for (i = 0; i < length; i++)
00090             fprintf (f, "%.16e\n",data[i].x);
00091         fclose (f);
00092
00093         sprintf (buffer, "%si_%d", file, step);
00094         f = fopen (buffer,"w");
00095         for (i = 0; i < length; i++)
00096             fprintf (f, "%.16e\n",data[i].y);
00097         fclose (f);
00098     }
00099
00100     /*
00101     * Writes out double type data files.
00102     */
00103     void writeOutDouble(char* buffer, char *file, double *
data, int length, int step){
00104         FILE *f;
00105         sprintf (buffer, "%s_%d", file, step);
00106         f = fopen (buffer,"w");
00107         int i;
00108         for (i = 0; i < length; i++)
00109             fprintf (f, "%.16e\n",data[i]);
00110         fclose (f);
00111     }
00112
00113     /*
00114     * Writes out int type data files.
00115     */
00116     void writeOutInt(char* buffer, char *file, int *data, int length, int step){
00117         FILE *f;
00118         sprintf (buffer, "%s_%d", file, step);

```

```

00119         f = fopen (buffer,"w");
00120         int i;
00121         for (i = 0; i < length; i++)
00122             fprintf (f, "%d\n",data[i]);
00123         fclose (f);
00124     }
00125
00126     /*
00127     * Writes out int2 data type.
00128     */
00129     void writeOutInt2(char* buffer, char *file, int2 *data, int length, int step){
00130         FILE *f;
00131         sprintf (buffer, "%s_%d", file, step);
00132         f = fopen (buffer,"w");
00133         int i;
00134         for (i = 0; i < length; i++)
00135             fprintf (f, "%d,%d\n",data[i].x,data[i].y);
00136         fclose (f);
00137     }
00138
00139     /*
00140     * Writes out tracked vortex data.
00141     */
00142     void writeOutVortex(char* buffer, char *file, struct
Tracker::Vortex *data, int length, int step){
00143         FILE *f;
00144         sprintf (buffer, "%s_%d", file, step);
00145         f = fopen (buffer,"w");
00146         int i;
00147         fprintf (f, "#X,Y,WINDING,SIGN\n");
00148         for (i = 0; i < length; i++)
00149             fprintf (f, "%d,%d,%d,%d\n",data[i].coords.x,data[i].coords.y,data[i].
wind,data[i].sign);
00150         fclose (f);
00151     }
00152
00153     /*
00154     * Opens and closes file. Nothing more. Nothing less.
00155     */
00156     int readState(char* name){
00157         FILE *f;
00158         f = fopen(name,"r");
00159         fclose(f);
00160         return 0;
00161     }
00162 }

```

## 6.43 src/kernels.cu File Reference

```
#include "../include/constants.h"
```

```
#include <stdio.h>
```

Include dependency graph for kernels.cu:

### Functions

- `__global__ void angularOp` (double `omega`, double `dt`, double2 \*`wfc`, double \*`xpyypx`, double2 \*`out`)
- `__device__ double2 braKetMult` (double2 `in1`, double2 `in2`)
- `__global__ void cMult` (double2 \*`in1`, double2 \*`in2`, double2 \*`out`)  
*Performs complex multiplication of in1 and in2, giving result as out.*
- `__global__ void cMultDensity` (double2 \*`in1`, double2 \*`in2`, double2 \*`out`, double `dt`, double `mass`, double `omegaZ`, int `gstate`, int `N`)
- `__device__ double complexMagnitude` (double2 `in`)
- `__host__ __device__ double complexMagnitudeSquared` (double2 `in`)
- `__host__ __device__ double2 complexMultiply` (double2 `in1`, double2 `in2`)
- `__device__ double2 conjugate` (double2 `in`)
- `__global__ void energyCalc` (double2 \*`wfc`, double2 \*`op`, double `dt`, double2 \*`energy`, int `gnd_state`, int `op_` - space, double `sqrt_omegaz_mass`)
- `__device__ unsigned int getBid3d3d` ()
- `__device__ unsigned int getGid3d3d` ()
- `__device__ unsigned int getTid3d3d` ()

- `__global__ void multipass (double2 *input, double2 *output, int pass)`  
*Routine for parallel summation.*
- `__global__ void pSum (double *in1, double *output, int pass)`  
*Routine for parallel summation.*
- `template<typename T >`  
`__global__ void pSumT (T *in1, T *output, int pass)`  
*Routine for parallel summation.*
- `__device__ double2 realCompMult (double scalar, double2 comp)`
- `__global__ void scalarDiv (double2 *in, double factor, double2 *out)`  
*Divides both components of vector type "in", by the value "factor".*
- `__global__ void scalarDiv_wfcNorm (double2 *in, double dr, double2 *pSum, double2 *out)`  
*As above, but normalises for wfc.*

## Variables

- `__constant__ double gDenConst = 2.535425438831619e-59`

### 6.43.1 Function Documentation

#### 6.43.1.1 `__global__ void angularOp ( double omega, double dt, double2 * wfc, double * xpyypx, double2 * out )`

Definition at line 153 of file [kernels.cu](#).

References [getGid3d3d\(\)](#), and [result](#).

```
00153                                     {
00154     unsigned int gid = getGid3d3d();
00155     double2 result;
00156     double op;
00157     op = exp( -omega*xpyypx[gid]*dt);
00158     result.x=wfc[gid].x*op;
00159     result.y=wfc[gid].y*op;
00160     out[gid]=result;
00161 }
```

Here is the call graph for this function:

#### 6.43.1.2 `__device__ double2 braKetMult ( double2 in1, double2 in2 ) [inline]`

Definition at line 88 of file [kernels.cu](#).

References [complexMultiply\(\)](#), and [conjugate\(\)](#).

Referenced by [energyCalc\(\)](#).

```
00089 {
00090     return complexMultiply(conjugate(in1), in2);
00091 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 6.43.1.3 `__global__ void cMult ( double2 * in1, double2 * in2, double2 * out )`

Performs complex multiplication of in1 and in2, giving result as out.

Definition at line 96 of file [kernels.cu](#).

References [getGid3d3d\(\)](#), and [result](#).

```

00096                                     {
00097     double2 result;
00098     unsigned int gid = getGid3d3d();
00099     result.x = (in1[gid].x*in2[gid].x - in1[gid].y*in2[gid].y);
00100     result.y = (in1[gid].x*in2[gid].y + in1[gid].y*in2[gid].x);
00101     out[gid] = result;
00102 }

```

Here is the call graph for this function:

#### 6.43.1.4 `__global__ void cMultDensity ( double2 * in1, double2 * in2, double2 * out, double dt, double mass, double omegaZ, int gstate, int N )`

Definition at line 104 of file [kernels.cu](#).

References [complexMagnitudeSquared\(\)](#), [HBAR](#), [mass](#), [PI](#), [result](#), [x](#), and [y](#).

```

00104                                     {
00105     double2 result;
00106     double gDensity;
00107     int tid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x*blockDim.x + threadIdx.x;
00108     gDensity = N*complexMagnitudeSquared(in2[tid])*4*
HBAR*HBAR*PI*(4.67e-9/mass)*sqrt(mass*(omegaZ)/(2*PI*
HBAR));
00109
00110     if(gstate == 0){
00111         double tmp = in1[tid].x*exp(-gDensity*(dt/HBAR) );
00112         result.x = (tmp)*in2[tid].x - (in1[tid].y)*in2[tid].y;
00113         result.y = (tmp)*in2[tid].y + (in1[tid].y)*in2[tid].x;
00114     }
00115     else{
00116         double2 tmp;
00117         tmp.x = in1[tid].x*cos(-gDensity*(dt/HBAR)) - in1[tid].y*sin(-gDensity*(
dt/HBAR));
00118         tmp.y = in1[tid].y*cos(-gDensity*(dt/HBAR)) + in1[tid].x*sin(-gDensity*(
dt/HBAR));
00119
00120         result.x = (tmp.x)*in2[tid].x - (tmp.y)*in2[tid].y;
00121         result.y = (tmp.x)*in2[tid].y + (tmp.y)*in2[tid].x;
00122     }
00123     out[tid] = result;
00124 }

```

Here is the call graph for this function:

#### 6.43.1.5 `__device__ double complexMagnitude ( double2 in )`

Definition at line 70 of file [kernels.cu](#).

```

00070                                     {
00071     return sqrt(in.x*in.x + in.y*in.y);
00072 }

```

#### 6.43.1.6 `__host__ __device__ double complexMagnitudeSquared ( double2 in )`

Definition at line 74 of file [kernels.cu](#).

Referenced by [cMultDensity\(\)](#), and [energyCalc\(\)](#).

```

00074                                     {
00075     return in.x*in.x + in.y*in.y;
00076 }

```

Here is the caller graph for this function:

#### 6.43.1.7 `__host__ __device__ double2 complexMultiply ( double2 in1, double2 in2 )`

Definition at line 78 of file [kernels.cu](#).

References [result](#).

Referenced by [braKetMult\(\)](#).

```
00078                                     {
00079     double2 result;
00080     result.x = (in1.x*in2.x - in1.y*in2.y);
00081     result.y = (in1.x*in2.y + in1.y*in2.x);
00082     return result;
00083 }
```

Here is the caller graph for this function:

#### 6.43.1.8 `__device__ double2 conjugate ( double2 in )`

Definition at line 56 of file [kernels.cu](#).

References [in\(\)](#), and [result](#).

Referenced by [braKetMult\(\)](#).

```
00056                                     {
00057     double2 result = in;
00058     result.y = -result.y;
00059     return result;
00060 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 6.43.1.9 `__global__ void energyCalc ( double2 * wfc, double2 * op, double dt, double2 * energy, int gnd_state, int op_space, double sqrt_omegaz_mass )`

Definition at line 193 of file [kernels.cu](#).

References [braKetMult\(\)](#), [complexMagnitudeSquared\(\)](#), [dt](#), [gDenConst](#), [getGid3d3d\(\)](#), [HBAR](#), [realCompMult\(\)](#), [result](#), and [x](#).

```
00193                                     {
00194     unsigned int gid = getGid3d3d();
00195     double hbar_dt = HBAR/dt;
00196     double g_local = 0.0;
00197     double2 result;
00198     double opLocal;
00199     if (op_space)
00200         g_local = gDenConst*sqrt_omegaz_mass*complexMagnitudeSquared(
00201             wfc[gid]);
00202     if (!gnd_state){
00203         opLocal = -log(op[gid].x + g_local)*hbar_dt;
00204     }
00205     else{
00206         opLocal = cos(op[gid].x + g_local)*hbar_dt;
00207     }
00208     result = braKetMult(wfc[gid], realCompMult(opLocal,
00209         wfc[gid]));
00210     //printf("oplocal=%e    Resx=%e Resy=%e\n", opLocal, result.x, result.y);
00211     energy[gid].x += result.x;
00212     energy[gid].y += result.y;
00213 }
```

Here is the call graph for this function:



6.43.1.10 `__device__ unsigned int getBid3d3d ( )`

Definition at line 46 of file [kernels.cu](#).

```
00046     {
00047     return blockIdx.x + gridDim.x*(blockIdx.y + gridDim.y * blockIdx.z);
00048 }
```

6.43.1.11 `__device__ unsigned int getGid3d3d ( )`

Definition at line 41 of file [kernels.cu](#).

Referenced by [angularOp\(\)](#), [cMult\(\)](#), [energyCalc\(\)](#), [multipass\(\)](#), [pSum\(\)](#), [pSumT\(\)](#), [scalarDiv\(\)](#), and [scalarDiv\\_wfc-Norm\(\)](#).

```
00041     {
00042     return blockDim.x * ( ( blockDim.y * ( ( blockIdx.z * blockDim.z + threadIdx.z ) + blockIdx.y ) +
threadIdx.y ) + blockIdx.x ) + threadIdx.x;
00043 }
```

Here is the caller graph for this function:

6.43.1.12 `__device__ unsigned int getTid3d3d ( )`

Definition at line 52 of file [kernels.cu](#).

```
00052     {
00053     return blockDim.x * ( blockDim.y * ( blockDim.z + ( threadIdx.z * blockDim.y ) ) + threadIdx.y ) +
threadIdx.x;
00054 }
```

6.43.1.13 `__global__ void multipass ( double2 * input, double2 * output, int pass )`

Routine for parallel summation.

Can be looped over from host.

Definition at line 166 of file [kernels.cu](#).

References [getGid3d3d\(\)](#), and [vis::i](#).

```
00166     {
00167     unsigned int tid = threadIdx.x;
00168     unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x; // printf("bid0=%d\n",bid);
00169     unsigned int gid = getGid3d3d();
00170     extern __shared__ double2 sdata[];
00171     sdata[tid] = input[gid];
00172     if(pass == 0){
00173         sdata[tid].x *= sdata[tid].x;
00174         sdata[tid].y *= sdata[tid].y;
00175     }
00176     __syncthreads();
00177     for(int i = blockDim.x>>1; i > 0; i>>=1){
00178         if(tid < blockDim.x>>1){
00179             sdata[tid].x += sdata[tid + i].x;
00180             sdata[tid].y += sdata[tid + i].y;
00181         }
00182         __syncthreads();
00183     }
00184     if(tid==0){
00185         output[bid] = sdata[0];
00186     }
00187 }
```

Here is the call graph for this function:

#### 6.43.1.14 `__global__ void pSum ( double * in1, double * output, int pass )`

Routine for parallel summation.

Can be looped over from host.

Definition at line 239 of file [kernels.cu](#).

References [getGid3d3d\(\)](#), and [vis::i](#).

```
00239                                     {
00240         unsigned int tid = threadIdx.x;
00241         unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x; // printf("bid0=%d\n",bid);
00242         unsigned int gid = getGid3d3d();
00243         extern __shared__ double sdata2[];
00244         for(int i = blockDim.x>>1; i > 0; i>>=1){
00245             if(tid < blockDim.x>>1){
00246                 sdata2[tid] += sdata2[tid + i];
00247             }
00248             __syncthreads();
00249         }
00250         if(tid==0){
00251             output[bid] = sdata2[0];
00252         }
00253 }
```

Here is the call graph for this function:

#### 6.43.1.15 `template<typename T> __global__ void pSumT ( T * in1, T * output, int pass )`

Routine for parallel summation.

Can be looped over from host.

Definition at line 220 of file [kernels.cu](#).

References [getGid3d3d\(\)](#), and [vis::i](#).

```
00220                                     {
00221         unsigned int tid = threadIdx.x;
00222         unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x; // printf("bid0=%d\n",bid);
00223         unsigned int gid = getGid3d3d();
00224         extern __shared__ T sdata[];
00225         for(int i = blockDim.x>>1; i > 0; i>>=1){
00226             if(tid < blockDim.x>>1){
00227                 sdata[tid] += sdata[tid + i];
00228             }
00229             __syncthreads();
00230         }
00231         if(tid==0){
00232             output[bid] = sdata[0];
00233         }
00234 }
```

Here is the call graph for this function:

#### 6.43.1.16 `__device__ double2 realCompMult ( double scalar, double2 comp )`

Definition at line 62 of file [kernels.cu](#).

References [result](#).

Referenced by [energyCalc\(\)](#).

```
00062                                     {
00063         double2 result;
00064         result.x = scalar * comp.x;
00065         result.y = scalar * comp.y;
00066         return result;
00067 }
```

Here is the caller graph for this function:

#### 6.43.1.17 `__global__ void scalarDiv ( double2 * in, double factor, double2 * out )`

Divides both components of vector type "in", by the value "factor".

Results given with "out"

Definition at line 130 of file [kernels.cu](#).

References [getGid3d3d\(\)](#), and [result](#).

```
00130                                     {
00131     double2 result;
00132     //extern __shared__ double2 tmp_in[];
00133     unsigned int gid = getGid3d3d();
00134     result.x = (in[gid].x*factor);
00135     result.y = (in[gid].y*factor);
00136     out[gid] = result;
00137 }
```

Here is the call graph for this function:

#### 6.43.1.18 `__global__ void scalarDiv_wfcNorm ( double2 * in, double dr, double2 * pSum, double2 * out )`

As above, but normalises for wfc.

Definition at line 142 of file [kernels.cu](#).

References [getGid3d3d\(\)](#), [result](#), [x](#), and [y](#).

```
00142                                     {
00143     unsigned int gid = getGid3d3d();
00144     double2 result;
00145     double norm = sqrt((pSum[0].x + pSum[0].y)*dr);
00146     result.x = (in[gid].x/norm);
00147     result.y = (in[gid].y/norm);
00148     out[gid] = result;
00149 }
```

Here is the call graph for this function:

### 6.43.2 Variable Documentation

#### 6.43.2.1 `__constant__ double gDenConst = 2.535425438831619e-59`

Definition at line 38 of file [kernels.cu](#).

Referenced by [energyCalc\(\)](#).

## 6.44 kernels.cu

```
00001 /** kernels.cu - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
```

```

00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034 #include "../include/constants.h"
00035 #include <stdio.h>
00036
00037
00038 __constant__ double gDenConst = 2.535425438831619e-59;//Evaluted in MATLAB:
    HBAR*(4.67e-9)*sqrt(8*HBAR*PI)*;
00039 //inline __device__ unsigned int getGid3d3d(){
00040
00041 __device__ unsigned int getGid3d3d(){
00042     return blockDim.x * ( ( blockDim.y * ( ( blockIdx.z * blockDim.z + threadIdx.z ) + blockIdx.y ) +
    threadIdx.y ) + blockIdx.x ) + threadIdx.x;
00043 }
00044
00045 //inline __device__ unsigned int getBid3d3d(){
00046 __device__ unsigned int getBid3d3d(){
00047     return blockIdx.x + gridDim.x*(blockIdx.y + gridDim.y * blockIdx.z);
00048 }
00049
00050
00051 //inline __device__ unsigned int getTid3d3d(){
00052 __device__ unsigned int getTid3d3d(){
00053     return blockDim.x * ( blockDim.y * ( blockDim.z + ( threadIdx.z * blockDim.y ) ) + threadIdx.y ) +
    threadIdx.x;
00054 }
00055
00056 __device__ double2 conjugate(double2 in){
00057     double2 result = in;
00058     result.y = -result.y;
00059     return result;
00060 }
00061
00062 __device__ double2 realCompMult(double scalar, double2 comp){
00063     double2 result;
00064     result.x = scalar * comp.x;
00065     result.y = scalar * comp.y;
00066     return result;
00067 }
00068
00069 //inline __device__ double complexMagnitude(double2 in){
00070 __device__ double complexMagnitude(double2 in){
00071     return sqrt(in.x*in.x + in.y*in.y);
00072 }
00073
00074 __host__ __device__ double complexMagnitudeSquared(double2
    in){
00075     return in.x*in.x + in.y*in.y;
00076 }
00077
00078 __host__ __device__ double2 complexMultiply(double2 in1, double2 in2){
00079     double2 result;
00080     result.x = (in1.x*in2.x - in1.y*in2.y);
00081     result.y = (in1.x*in2.y + in1.y*in2.x);
00082     return result;
00083 }
00084
00085 /*
00086 * Used to perform conj(in1)*in2; == < in1 | in2 >
00087 */
00088 inline __device__ double2 braKetMult(double2 in1, double2 in2)
    {
00089     return complexMultiply(conjugate(in1),in2);
00090 }
00091
00092
00096 __global__ void cMult(double2* in1, double2* in2, double2* out){
00097     double2 result;
00098     unsigned int gid = getGid3d3d();
00099     result.x = (in1[gid].x*in2[gid].x - in1[gid].y*in2[gid].y);
00100     result.y = (in1[gid].x*in2[gid].y + in1[gid].y*in2[gid].x);
00101     out[gid] = result;
00102 }
00103
00104 __global__ void cMultDensity(double2* in1, double2* in2, double2* out, double
    dt, double mass,double omegaZ, int gstate, int N){

```

```

00105     double2 result;
00106     double gDensity;
00107     int tid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x*blockDim.x + threadIdx.x;
00108     gDensity = N*complexMagnitudeSquared(in2[tid])*4*HBAR*
HBAR*PI*(4.67e-9/mass)*sqrt(mass*(omegaZ)/(2*PI*HBAR));
00109
00110     if(gstate == 0){
00111         double tmp = in1[tid].x*exp(-gDensity*(dt/HBAR));
00112         result.x = (tmp)*in2[tid].x - (in1[tid].y)*in2[tid].y;
00113         result.y = (tmp)*in2[tid].y + (in1[tid].y)*in2[tid].x;
00114     }
00115     else{
00116         double2 tmp;
00117         tmp.x = in1[tid].x*cos(-gDensity*(dt/HBAR)) - in1[tid].y*sin(-gDensity*(dt/
HBAR));
00118         tmp.y = in1[tid].y*cos(-gDensity*(dt/HBAR)) + in1[tid].x*sin(-gDensity*(dt/
HBAR));
00119
00120         result.x = (tmp.x)*in2[tid].x - (tmp.y)*in2[tid].y;
00121         result.y = (tmp.x)*in2[tid].y + (tmp.y)*in2[tid].x;
00122     }
00123     out[tid] = result;
00124 }
00125
00130 __global__ void scalarDiv(double2* in, double factor, double2* out){
00131     double2 result;
00132     //extern __shared__ double2 tmp_in[];
00133     unsigned int gid = getGid3d3d();
00134     result.x = (in[gid].x*factor);
00135     result.y = (in[gid].y*factor);
00136     out[gid] = result;
00137 }
00138
00142 __global__ void scalarDiv_wfcNorm(double2* in, double dr, double2*
pSum, double2* out){
00143     unsigned int gid = getGid3d3d();
00144     double2 result;
00145     double norm = sqrt((pSum[0].x + pSum[0].y)*dr);
00146     result.x = (in[gid].x/norm);
00147     result.y = (in[gid].y/norm);
00148     out[gid] = result;
00149 }
00150
00153 __global__ void angularOp(double omega, double dt, double2* wfc, double* xpyypx, double2
* out){
00154     unsigned int gid = getGid3d3d();
00155     double2 result;
00156     double op;
00157     op = exp(-omega*xpyypx[gid]*dt);
00158     result.x=wfc[gid].x*op;
00159     result.y=wfc[gid].y*op;
00160     out[gid]=result;
00161 }
00162
00166 __global__ void multipass(double2* input, double2* output, int pass){
00167     unsigned int tid = threadIdx.x;
00168     unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x; // printf("bid0=%d\n",bid);
00169     unsigned int gid = getGid3d3d();
00170     extern __shared__ double2 sdata[];
00171     sdata[tid] = input[gid];
00172     if(pass == 0){
00173         sdata[tid].x *= sdata[tid].x;
00174         sdata[tid].y *= sdata[tid].y;
00175     }
00176     __syncthreads();
00177     for(int i = blockDim.x>>1; i > 0; i>=1){
00178         if(tid < blockDim.x>>1){
00179             sdata[tid].x += sdata[tid + i].x;
00180             sdata[tid].y += sdata[tid + i].y;
00181         }
00182         __syncthreads();
00183     }
00184     if(tid==0){
00185         output[bid] = sdata[0];
00186     }
00187 }
00188
00189
00190 /*
00191 * Calculates all of the energy of the current state. sqrt_omegaz_mass = sqrt(omegaZ/mass), part of the
nonlin interaction term
00192 */
00193 __global__ void energyCalc(double2 *wfc, double2 *op, double dt, double2 *energy, int
gnd_state, int op_space, double sqrt_omegaz_mass){
00194     unsigned int gid = getGid3d3d();
00195     double hbar_dt = HBAR/dt;
00196     double g_local = 0.0;

```

```

00197     double2 result;
00198     double opLocal;
00199     if(op_space)
00200         g_local = gDenConst*sqrt_omegaz_mass*complexMagnitudeSquared(wfc[
gid]);
00201     if(!gnd_state){
00202         opLocal = -log(op[gid].x + g_local)*hbar_dt;
00203     }
00204     else{
00205         opLocal = cos(op[gid].x + g_local)*hbar_dt;
00206     }
00207     result = braKetMult(wfc[gid], realCompMult(opLocal,wfc[gid]));
00208     //printf("oplocal=%e    Resx=%e Resy=%e\n",opLocal,result.x,result.y);
00209     energy[gid].x += result.x;
00210     energy[gid].y += result.y;
00211 }
00212
00213
00214 //#####
00215 //#####
00216
00220 template<typename T> __global__ void pSum(T* in1, T* output, int pass){
00221     unsigned int tid = threadIdx.x;
00222     unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x; // printf("bid0=%d\n",bid);
00223     unsigned int gid = getGid3d3d();
00224     extern __shared__ T sdata[];
00225     for(int i = blockDim.x>>1; i > 0; i>>=1){
00226         if(tid < blockDim.x>>1){
00227             sdata[tid] += sdata[tid + i];
00228         }
00229         __syncthreads();
00230     }
00231     if(tid==0){
00232         output[bid] = sdata[0];
00233     }
00234 }
00235
00239 __global__ void pSum(double* in1, double* output, int pass){
00240     unsigned int tid = threadIdx.x;
00241     unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x; // printf("bid0=%d\n",bid);
00242     unsigned int gid = getGid3d3d();
00243     extern __shared__ double sdata2[];
00244     for(int i = blockDim.x>>1; i > 0; i>>=1){
00245         if(tid < blockDim.x>>1){
00246             sdata2[tid] += sdata2[tid + i];
00247         }
00248         __syncthreads();
00249     }
00250     if(tid==0){
00251         output[bid] = sdata2[0];
00252     }
00253 }
00254
00255
00256
00257 //#####
00258 //#####

```

## 6.45 src/minions.cc File Reference

#include "../include/minions.h"  
Include dependency graph for minions.cc:

### Namespaces

- [Minions](#)

### Functions

- double2 [Minions::complexDiv](#) (double2 num, double2 den)
- double [Minions::complexMag](#) (double2 in)
- double [Minions::complexMag2](#) (double2 in)
- double2 [Minions::complexMult](#) (double2 in1, double2 in2)

- double2 `Minions::complexScale` (double2 comp, double scale)
- double2 `Minions::conj` (double2 c)
- void `Minions::coordSwap` (struct `Tracker::Vortex` \*vCoords, int src, int dest)
- double `Minions::flnvSqRt` (double)
- *id magic hackery*
- double `Minions::maxValue` (double \*, int)
- double `Minions::minValue` (double \*, int)
- double `Minions::psi2` (double2)
- double `Minions::sumAvg` (double \*in, int len)

## 6.46 minions.cc

```

00001 /** minions.cc - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034 #include "../include/minions.h"
00035
00036 namespace Minions{
00037     double psi2(double2 in){
00038         return in.x*in.x + in.y*in.y;
00039     }
00040
00041     double maxValue(double* grid,int len){
00042         double max = grid[0];
00043         for (unsigned int i=1;i<len-1;++i){
00044             if(max<grid[i]){
00045                 max=grid[i];
00046             }
00047         }
00048         return max;
00049     }
00050
00051     double minValue(double* grid,int len){
00052         double min = grid[0];
00053         for (unsigned int i=1;i<len-1;++i){
00054             if(min>grid[i])
00055                 min=grid[i];
00056         }
00057         return min;
00058     }
00059
00060     double sumAvg(double* in, int len){
00061         double avg = 0.0;
00062
00063         for (unsigned int i=0; i<len; ++i){
00064             avg += in[i];
00065         }
00066         return avg/len;

```

```

00067     }
00068
00069     double fInvSqrt(double in){
00070         long long l;
00071         double in05, calc;
00072         const double threehalfs = 1.5;
00073
00074         in05 = in*0.5;
00075         calc=in;
00076         l = * (long long*) &calc;
00077         l = 0x5fe6eb50c7b537a9LL - (l >> 1);
00078         calc = *(double *) &l;
00079         calc = calc*( 1.5 - (in05*calc*calc) );
00080
00081         return calc;
00082     }
00083
00084     void coordSwap(struct Tracker::Vortex *vCoords, int src, int dest){
00085         struct Tracker::Vortex d = vCoords[dest];
00086         vCoords[dest] = vCoords[src];
00087         vCoords[src] = d;
00088     }
00089
00090     double complexMag(double2 in){
00091         return sqrt(in.x*in.x + in.y*in.y);
00092     }
00093
00094     double complexMag2(double2 in){
00095         return in.x*in.x + in.y*in.y;
00096     }
00097
00098     double2 complexMult(double2 in1, double2 in2){
00099         double2 result;
00100         result.x = (in1.x*in2.x - in1.y*in2.y);
00101         result.y = (in1.x*in2.y + in1.y*in2.x);
00102         return result;
00103     }
00104
00105     double2 complexScale(double2 comp, double scale){
00106         double2 result;
00107         result.x = comp.x*scale;
00108         result.y = comp.y*scale;
00109         return result;
00110     }
00111
00112     double2 conj(double2 c){
00113         double2 result = c;
00114         result.y = -result.y;
00115         return result;
00116     }
00117
00118     double2 complexDiv(double2 num, double2 den){
00119         double2 c = conj(den);
00120         return complexScale(complexMult(num,c), (1.0/
complexMag2(den)));
00121     }
00122 }
00123
00124 /*
00125 int qSort(int2 *vCoords, int *vCoordsP int index, int length){
00126     if(index < 2){
00127         return 0;
00128     }
00129     int2 pivot;
00130     int l = 0;
00131     int r = length - 1;
00132     while (l <= r){
00133         0;
00134     }
00135 }
00136 */

```

## 6.47 src/multigpu.cu File Reference

## 6.48 multigpu.cu



## 6.49 src/split\_op.cu File Reference

```
#include "../include/split_op.h"
#include "../include/kernels.h"
#include "../include/constants.h"
#include "../include/fileIO.h"
#include "../include/tracker.h"
#include "../include/minions.h"
#include "../include/ds.h"
```

Include dependency graph for split\_op.cu:

### Functions

- void [delta\\_define](#) (double \*x, double \*y, double x0, double y0, double \*delta)
- double [energy\\_angmom](#) (double \*Energy, double \*Energy\_gpu, double2 \*V\_op, double2 \*K\_op, double dx, double dy, double2 \*gpuWfc, int gState)

*Calculates energy and angular momentum of current state.*

- int [evolve](#) (cufftDoubleComplex \*gpuWfc, cufftDoubleComplex \*gpuMomentumOp, cufftDoubleComplex \*gpuPositionOp, void \*gpu1dyPx, void \*gpu1dxPy, cufftDoubleComplex \*gpuParSum, int [gridSize](#), int numSteps, int [threads](#), unsigned int gstate, int lz, int nonlin, int printSteps, int N, unsigned int ramp)
- int [initialise](#) (double [omegaX](#), double [omegaY](#), int N)
- int [isError](#) (int [result](#), char \*c)
- int [main](#) (int argc, char \*\*argv)
- void [optLatSetup](#) (struct [Tracker::Vortex](#) centre, double \*V, struct [Tracker::Vortex](#) \*vArray, int num\_vortices, double theta\_opt, double intensity, double \*v\_opt, double \*x, double \*y)

*Matches the optical lattice to the vortex lattice.*

- int [parseArgs](#) (int argc, char \*\*argv)
- void [parSum](#) (double2 \*gpuWfc, double2 \*gpuParSum, int xDim, int yDim, int [threads](#))
- template<typename T >  
void [parSum](#) (T \*gpuToSumArr, T \*gpuParSum, int xDim, int yDim, int [threads](#))

### Variables

- double [a0x](#)
- double [a0y](#)
- double [angle\\_sweep](#)
- char [buffer](#) [100]
- int [device](#)
- double [gammaY](#)
- int [kick\\_it](#)
- double [omega](#)
- Params \* [paramS](#)
- Array [params](#)
- double [Rxy](#)
- double [sepMinEpsilon](#) =0.0
- double [timeTotal](#)
- int [verbose](#)
- double [x0\\_shift](#)
- double [y0\\_shift](#)

## 6.49.1 Function Documentation

### 6.49.1.1 void delta\_define ( double \* x, double \* y, double x0, double y0, double \* delta )

Definition at line 865 of file [split\\_op.cu](#).

References [dt](#), [dx](#), [EV\\_opt](#), [HBAR](#), [vis::i](#), [V](#), [xDim](#), and [yDim](#).

```
00865
00866     for (unsigned int i=0; i<xDim; ++i){
00867         for (unsigned int j=0; j<yDim; ++j){
00868             delta[j*xDim + i] = 1e6*HBAR*exp( -( pow( x[i] - x0, 2) + pow(
00869                 y[j] - y0, 2) )/(5*dx*dx) );
00869             EV_opt[(j*xDim + i)].x=cos( -(V[(j*xDim + i)] + delta[j*xDim +
00870                 i])*(dt/(2*HBAR))) );
00870             EV_opt[(j*xDim + i)].y=sin( -(V[(j*xDim + i)] + delta[j*xDim +
00871                 i])*(dt/(2*HBAR))) );
00871         }
00872     }
00873 }
```

### 6.49.1.2 double energy\_angmom ( double \* Energy, double \* Energy\_gpu, double2 \* V\_op, double2 \* K\_op, double dx, double dy, double2 \* gpuWfc, int gState )

Calculates energy and angular momentum of current state.

Definition at line 655 of file [split\\_op.cu](#).

References [vis::i](#), [result](#), [xDim](#), and [yDim](#).

```
00655
00656     {
00657         double renorm_factor_2d=1.0/pow(xDim*yDim,0.5);
00658         double result=0;
00659         for (int i=0; i < xDim*yDim; ++i){
00660             Energy[i] = 0.0;
00661         }
00662
00663         /* cudaMalloc((void**) &energy_gpu, sizeof(double2) * xDim*yDim);
00664
00665         energyCalc<<<grid,threads>>>( gpuWfc, V_op, 0.5*dt, energy_gpu, gState,1,i 0.5*sqrt(omegaZ/mass));
00666         result = cufftExecZ2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_FORWARD );
00667         scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise
00668
00669         energyCalc<<<grid,threads>>>( gpuWfc, K_op, dt, energy_gpu, gState,0, 0.5*sqrt(omegaZ/mass));
00670         result = cufftExecZ2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_INVERSE );
00671         scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise
00672
00673         err=cudaMemcpy(energy, energy_gpu, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);
00674
00675         for(int i=0; i<xDim*yDim; i++){
00676             result += energy[i].x;
00677             //printf("En=%E\n",result*dx*dy);
00678         }
00679         return result*dx*dy;
00680     }
00681 */
00682
00683 }
```

### 6.49.1.3 int evolve ( cufftDoubleComplex \* gpuWfc, cufftDoubleComplex \* gpuMomentumOp, cufftDoubleComplex \* gpuPositionOp, void \* gpu1dyPx, void \* gpu1dxPy, cufftDoubleComplex \* gpuParSum, int gridSize, int numSteps, int threads, unsigned int gstate, int lz, int nonlin, int printSteps, int N, unsigned int ramp )

```
** #####
**
** HERE BE DRAGONS OF THE MOST DANGEROUS KIND! **
** #####
**
```

```

** #####
**
** #####
**
** More F'n' Dragons! **
** #####
**
** #####
**

```

Definition at line 293 of file `split_op.cu`.

References `a_s`, `angle_sweep`, `appendData()`, `buffer`, `Tracker::Vortex::coords`, `dt`, `dx`, `dy`, `EV`, `EV_opt`, `Tracker::findVortex()`, `gdt`, `HBAR`, `vis::i`, `interaction`, `kick_it`, `laser_power`, `mass`, `omega`, `omegaX`, `omegaY`, `omegaZ`, `optLatSetup()`, `parSum()`, `PI`, `plan_1d`, `plan_2d`, `Minions::psi2()`, `result`, `sepAvg()`, `Tracker::Vortex::sign`, `Minions::sumAvg()`, `V`, `V_gpu`, `V_opt`, `Tracker::vortAngle()`, `Tracker::vortArrange()`, `Tracker::vortCentre()`, `Tracker::vortPos()`, `Tracker::vortSepAvg()`, `wfc`, `Tracker::Vortex::wind`, `write_it`, `FileIO::writeOut()`, `FileIO::writeOutDouble()`, `FileIO::writeOutParam()`, `FileIO::writeOutVortex()`, `x`, `xDim`, `xi`, `y`, and `yDim`.

Referenced by `main()`.

```

00300
00301
00302 //Because no two operations are created equally. Multiplimultiplication is faster than divisions.
00303 double renorm_factor_2d=1.0/pow(gridSize,0.5);
00304 double renorm_factor_1d=1.0/pow(xDim,0.5);
00305
00306 clock_t begin, end;
00307 double time_spent;
00308 double Dt;
00309 if(gstate==0){
00310     Dt = gdt;
00311     printf("Timestep for grounstate solver set as: %E\n",Dt);
00312 }
00313 else{
00314     Dt = dt;
00315     printf("Timestep for evolution set as: %E\n",Dt);
00316 }
00317 begin = clock();
00318 double omega_0=omega*omegaX;
00319
00320 #if 0
00321
00322 int gridSum = 1<<6;
00323 double *densitySubset = (double*) malloc(sizeof(double)*gridSum);
00324 #pragma omp parallel for private(k)
00325 for (int j=0; j<gridSum; ++j){
00326     for (int k=0; k<gridSum; ++k){
00327         densitySubset[j*gridSum + k] = Minions::psi2(wfc[ ( (
yDim/2) - (gridSum/2) + j ) *yDim + ( (xDim/2) - (gridSum/2) + k )]);
00328     }
00329 }
00330 xi = 1/sqrt(8*PI*a_s*Minions::sumAvg(densitySubset,gridSum)/(
dx*dy)); //defined central condensate density
00331 printf("Avg healing length at centre=%E\n",xi);
00332 #endif
00333
00334 //Double buffering and will attempt to thread free and calloc operations to hide time penalty. Or may
not bother.
00335 int num_vortices[2] = {0,0};
00336 int num_latt_max = 0;
00337 int* vortexLocation; //binary matrix of size xDim*yDim, 1 for vortex at specified index, 0 otherwise
00338 int* olMaxLocation = (int*) calloc(xDim*yDim,sizeof(int));
00339
00340 struct Tracker::Vortex central_vortex; //vortex closest to the central position
00341 double vort_angle; //Angle of vortex lattice. Add to optical lattice for alignment.
00342 struct Tracker::Vortex *vortCoords = NULL; //array of vortex coordinates from
vortexLocation 1's
00343 struct Tracker::Vortex *vortCoordsP = NULL; //Previous array of vortex coordinates from
vortexLocation 1's
00344 int2 *olCoords = NULL; //array of vortex coordinates from vortexLocation 1's
00345 int2 *vortDelta = NULL;
00346
00347 double vortOLSigma=0.0;
00348 double sepAvg = 0.0;
00349
00350 int num_kick = 0;

```

```

00355     double t_kick = (2*PI/omega_0)/(6*Dt);
00356
00357     for(int i=0; i < numSteps; ++i){
00358         if ( ramp == 1 ){
00359             omega_0=omegaX*((omega-0.39)*((double)i/(double)(numSteps)) + 0.39); //Adjusts omega for
the appropriate trap frequency.
00360         }
00361         if(i % printSteps == 0){
00362             printf("Step: %d      Omega: %lf\n",i,omega_0/omegaX);
00363             cudaMemcpy(wfc, gpuWfc, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);
00364             end = clock();
00365             time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
00366             printf("Time spent: %lf\n",time_spent);
00367             char* fileName = "";
00368             printf("ramp=%d      gstate=%d      rg=%d      \n",ramp,gstate,ramp | (gstate<<1));
00369             switch ( ramp | (gstate<<1) ){
00370                 case 0:
00371                     fileName = "wfc_0_const";
00372                     break;
00373                 case 1:
00374                     fileName = "wfc_0_ramp";
00375                     break;
00376                 case 2:
00377                     fileName = "wfc_ev";
00378                     vortexLocation = (int*) calloc(xDim*yDim,sizeof(int));
00379                     num_vortices[0] = Tracker::findVortex(vortexLocation,
wfc, 1e-4, xDim, x, i);
00380                     if(i==0){
00381                         vortCoords = (struct Tracker::Vortex*) malloc(sizeof(struct
Tracker::Vortex)*(2*num_vortices[0]));
00382                         vortCoordsP = (struct Tracker::Vortex*) malloc(sizeof(struct
Tracker::Vortex)*(2*num_vortices[0]));
00383                         Tracker::vortPos(vortexLocation, vortCoords,
xDim, wfc);
00384                         central_vortex = Tracker::vortCentre(vortCoords, num_vortices[0]
, xDim);
00385                         vort_angle = Tracker::vortAngle(vortCoords,central_vortex,
num_vortices[0]);
00386                         appendData(&params,"Vort_angle",vort_angle);
00387                         optLatSetup(central_vortex, V, vortCoords, num_vortices[0], vort_angle
+ PI*angle_sweep/180.0, laser_power*HBAR*sqrt(omegaX*
omegaY), V_opt, x, y);
00388                         sepAvg = Tracker::vortSepAvg(vortCoords,central_vortex,
num_vortices[0]);
00389                         if(kick_it == 2){
00390                             printf("Kicked it 1\n");
00391                             cudaMemcpy(V_gpu, EV_opt, sizeof(cufftDoubleComplex)*
xDim*yDim, cudaMemcpyHostToDevice);
00392                         }
00393                         FileIO::writeOutDouble(buffer,"V_opt_1",
V_opt,xDim*yDim,0);
00394                         FileIO::writeOut(buffer,"EV_opt_1",
EV_opt,xDim*yDim,0);
00395                         appendData(&params,"Central_vort_x", (double)central_vortex.coords.x
);
00396                         appendData(&params,"Central_vort_y", (double)central_vortex.coords.y
);
00397                         appendData(&params,"Central_vort_winding", (double)central_vortex.
wind);
00398                         appendData(&params,"Central_vort_sign", (double)central_vortex.sign)
;
00399                         appendData(&params,"Num_vort", (double)num_vortices[0]);
00400                         FileIO::writeOutParam(buffer,
params, "Params.dat");
00401                     }
00402                     else if(num_vortices[0] > num_vortices[1]){
00403                         printf("Number of vortices changed from %d to %d\n",num_vortices[1],num_vortices[0]
);
00404                         Tracker::vortPos(vortexLocation, vortCoords,
xDim,wfc);
00405                     }
00406                     else{
00407                         Tracker::vortPos(vortexLocation, vortCoords,
xDim,wfc);
00408                         Tracker::vortArrange(vortCoords, vortCoordsP, num_vortices[0]);
00409                     }
00410                     /* num_latt_max = Tracker::findOLMaxima(olMaxLocation, V_opt, 1e-4, xDim, x);
00411                     if(num_latt_max == num_vortices[0]){
00412                         olCoords = (int2*) malloc(sizeof(int2)*num_latt_max);
00413                         Tracker::olPos(olMaxLocation, olCoords, xDim);
00414                         vortOLSigma = Tracker::sigVOL(vortCoords, olCoords, x, num_latt_max);
00415                         FileIO::writeOutInt2(buffer, "opt_max_arr", olCoords, num_latt_max, i);
00416                         free(olCoords);
00417                     }*/
00418                     FileIO::writeOutVortex(buffer, "vort_arr", vortCoords,
num_vortices[0], i);
00419                     printf("Located %d vortices\n",num_vortices[0]);

```

```

00420         printf("Sigma=%e\n",vortOLSigma);
00421         free(vortexLocation);
00422         num_vortices[1] = num_vortices[0];
00423         memcpy(vortCoordsP,vortCoords,sizeof(int2)*num_vortices[0]);
00424         break;
00425     case 3:
00426         fileName = "wfc_ev_ramp";
00427         break;
00428     default:
00429         break;
00430     }
00431     if(write_it)
00432         FileIO::writeOut(buffer, fileName, wfc,
00433             xDim*yDim, i);
00434     //printf("Energy[t%d]=%E\n",i,energy_angmom(gpuPositionOp, gpuMomentumOp, dx, dy,
00435         gpuWfc,gstate));
00436     /*
00437         cudaMemcpy(V_gpu, V, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
00438         cudaMemcpy(K_gpu, K, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
00439         cudaMemcpy(V_gpu, , sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
00440         cudaMemcpy(K_gpu, K, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
00441     */
00442     }
00443     if(i % ((int)t_kick+1) == 0 && num_kick<=6 && gstate==1 && kick_it == 1 ){
00444         cudaMemcpy(V_gpu, EV_opt, sizeof(cufftDoubleComplex)*xDim*yDim,
00445             cudaMemcpyHostToDevice);
00446         ++num_kick;
00447     }
00448     /*
00449         * U_r(dt/2)*wfc
00450     */
00451     if(nonlin == 1){
00452         cMultDensity<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc,0.5*Dt,
00453             mass,omegaZ,gstate,N*interaction);
00454     }
00455     else {
00456         cMult<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc);
00457     }
00458     /*
00459         * U_p(dt)*fft2(wfc)
00460     */
00461     result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD);
00462     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc); //Normalise
00463     cMult<<<grid,threads>>>(gpuMomentumOp,gpuWfc,gpuWfc);
00464     result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE);
00465     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc); //Normalise
00466     /*
00467         * U_r(dt/2)*wfc
00468     */
00469     if(nonlin == 1){
00470         cMultDensity<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc,Dt*0.5,
00471             mass,omegaZ,gstate,N*interaction);
00472     }
00473     else {
00474         cMult<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc);
00475     }
00476     if( (i % (int)(t_kick+1) == 0 && num_kick<=6 && gstate==1) || (kick_it >= 1 &&
00477         i==0) ){
00478         cudaMemcpy(V_gpu, EV, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
00479         printf("Got here: Cuda memcpy EV into GPU\n");
00480     }
00481     /*
00482         Angular momentum xPy-yPx
00483     */
00484     if(lz == 1){
00485         switch(i%2 | (gstate<<1)){
00486             case 0: //Groundstate solver, even step
00487                 result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
00488                 scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
00489                 angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpuldxPy, gpuWfc);
00490                 result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE);
00491                 scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
00492
00493                 result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
00494                 scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
00495                 result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
00496                 scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
00497                 angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpuldyPx, gpuWfc);
00498                 result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
00499                 scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
00500                 result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
00501                 scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
00502                 break;
00503             case 1: //Groundstate solver, odd step
00504                 result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
00505                 scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
00506

```

```

00507         result = cufftExecZ2Z(plan_1d, gpuWfc, gpuWfc, CUFFT_INVERSE); //1D inverse to wfc_yPx
00508         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_1d, gpuWfc);
00509         angularOp<<<grid, threads>>>(omega_0, Dt, gpuWfc, (double*) gpuldyPx, gpuWfc);
00510         result = cufftExecZ2Z(plan_1d, gpuWfc, gpuWfc, CUFFT_FORWARD); // wfc_PxPy
00511         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_1d, gpuWfc);
00512         result = cufftExecZ2Z(plan_2d, gpuWfc, gpuWfc, CUFFT_INVERSE); //2D Inverse
00513         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_2d, gpuWfc);
00514
00515         result = cufftExecZ2Z(plan_1d, gpuWfc, gpuWfc, CUFFT_FORWARD); // wfc_xPy
00516         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_1d, gpuWfc);
00517         angularOp<<<grid, threads>>>(omega_0, Dt, gpuWfc, (double*) gpuldxPy, gpuWfc);
00518         result = cufftExecZ2Z(plan_1d, gpuWfc, gpuWfc, CUFFT_INVERSE);
00519         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_1d, gpuWfc);
00520         break;
00521
00522         case 2: //Real time evolution, even step
00523         result = cufftExecZ2Z(plan_1d, gpuWfc, gpuWfc, CUFFT_FORWARD); // wfc_xPy
00524         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_1d, gpuWfc);
00525         cMult<<<grid, threads>>>(gpuWfc, (cufftDoubleComplex*) gpuldxPy, gpuWfc);
00526         result = cufftExecZ2Z(plan_1d, gpuWfc, gpuWfc, CUFFT_INVERSE);
00527         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_1d, gpuWfc);
00528
00529         result = cufftExecZ2Z(plan_2d, gpuWfc, gpuWfc, CUFFT_FORWARD); //2D forward
00530         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_2d, gpuWfc);
00531         result = cufftExecZ2Z(plan_1d, gpuWfc, gpuWfc, CUFFT_INVERSE); //1D inverse to wfc_yPx
00532         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_1d, gpuWfc);
00533         cMult<<<grid, threads>>>(gpuWfc, (cufftDoubleComplex*) gpuldyPx, gpuWfc);
00534         result = cufftExecZ2Z(plan_1d, gpuWfc, gpuWfc, CUFFT_FORWARD); // wfc_PxPy
00535         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_1d, gpuWfc);
00536         result = cufftExecZ2Z(plan_2d, gpuWfc, gpuWfc, CUFFT_INVERSE); //2D Inverse
00537         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_2d, gpuWfc);
00538         break;
00539
00540         case 3: //Real time evolution, odd step
00541         result = cufftExecZ2Z(plan_2d, gpuWfc, gpuWfc, CUFFT_FORWARD); //2D forward
00542         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_2d, gpuWfc);
00543         result = cufftExecZ2Z(plan_1d, gpuWfc, gpuWfc, CUFFT_INVERSE); //1D inverse to wfc_yPx
00544         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_1d, gpuWfc);
00545         cMult<<<grid, threads>>>(gpuWfc, (cufftDoubleComplex*) gpuldyPx, gpuWfc);
00546         result = cufftExecZ2Z(plan_1d, gpuWfc, gpuWfc, CUFFT_FORWARD); // wfc_PxPy
00547         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_1d, gpuWfc);
00548         result = cufftExecZ2Z(plan_2d, gpuWfc, gpuWfc, CUFFT_INVERSE); //2D Inverse
00549         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_2d, gpuWfc);
00550
00551         result = cufftExecZ2Z(plan_1d, gpuWfc, gpuWfc, CUFFT_FORWARD); // wfc_xPy
00552         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_1d, gpuWfc);
00553         cMult<<<grid, threads>>>(gpuWfc, (cufftDoubleComplex*) gpuldxPy, gpuWfc);
00554         result = cufftExecZ2Z(plan_1d, gpuWfc, gpuWfc, CUFFT_INVERSE);
00555         scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_1d, gpuWfc);
00556         break;
00557     }
00558 }
00559 }
00560 /*****
00561
00562     if(gstate==0){
00563         parSum(gpuWfc, gpuParSum, xDim, yDim, threads);
00564     }
00565 }
00566 return 0;
00567 }

```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 6.49.1.4 int initialise ( double omegaX, double omegaY, int N )

Definition at line 64 of file [split\\_op.cu](#).

References [a0x](#), [a0y](#), [a\\_s](#), [appendData\(\)](#), [buffer](#), [dt](#), [dx](#), [dy](#), [EappliedField](#), [EK](#), [Energy](#), [Energy\\_gpu](#), [EV](#), [EV\\_opt](#), [ExPy](#), [EyPx](#), [gammaY](#), [gdt](#), [GK](#), [grid](#), [GV](#), [HBAR](#), [vis::i](#), [K](#), [K\\_gpu](#), [l](#), [mass](#), [omega](#), [omegaZ](#), [par\\_sum](#), [Phi](#), [PI](#), [plan\\_1d](#), [plan\\_2d](#), [r](#), [result](#), [Rxy](#), [threads](#), [V](#), [V\\_gpu](#), [V\\_opt](#), [wfc](#), [wfc\\_backup](#), [wfc\\_gpu](#), [FileIO::writeOut\(\)](#), [FileIO::writeOutDouble\(\)](#), [x](#), [xDim](#), [xMax](#), [xp](#), [xPy](#), [xPy\\_gpu](#), [y](#), [yDim](#), [yMax](#), [yp](#), [yPx](#), and [yPx\\_gpu](#).

Referenced by [main\(\)](#).

```

00064     {
00065         //*****
00066         unsigned int xD=1,yD=1,zD=1;

```

```

00067     threads = 128;
00068     unsigned int b = xDim*yDim/threads; //number of blocks in simulation
00069     unsigned long long maxElements = 65536*65536ULL; //largest number of elements
00070
00071     if( b < (1<<16) ){
00072         xD = b;
00073     }
00074     else if( (b >= (1<<16) ) && (b <= (maxElements)) ){
00075         int t1 = log(b)/log(2);
00076         float t2 = (float) t1/2;
00077         t1 = (int) t2;
00078         if(t2 > (float) t1){
00079             xD <= t1;
00080             yD <= (t1 + 1);
00081         }
00082         else if(t2 == (float) t1){
00083             xD <= t1;
00084             yD <= t1;
00085         }
00086     }
00087     else{
00088         printf("Outside range of supported indexing");
00089         exit(-1);
00090     }
00091     printf("Compute grid dimensions chosen as X=%d Y=%d\n",xD,yD);
00092
00093     grid.x=xD;
00094     grid.y=yD;
00095     grid.z=zD;
00096     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00097
00098     unsigned int i,j; //Used in for-loops for indexing
00099
00100     unsigned int gSize = xDim*yDim;
00101     double xOffset, yOffset;
00102     xOffset=0.0; //5.0e-6;
00103     yOffset=0.0; //5.0e-6;
00104
00105     mass = 1.4431607e-25; //Rb 87 mass, kg
00106     appendData(&params,"Mass",mass);
00107     a_s = 4.67e-9;
00108     appendData(&params,"a_s",a_s);
00109
00110     double sum = 0.0;
00111
00112     a0x = sqrt(HBAR/(2*mass*omegaX));
00113     a0y = sqrt(HBAR/(2*mass*omegaY));
00114     appendData(&params,"a0x",a0x);
00115     appendData(&params,"a0y",a0y);
00116
00117     Rxy = pow(15,0.2)*pow(N*a_s*sqrt(mass*omegaZ/HBAR),0.2);
00118     appendData(&params,"Rxy",Rxy);
00119     //Rxy = pow(15,0.2)*pow(N*4.67e-9*sqrt(mass*pow(omegaX*omegaY,0.5)/HBAR),0.2);
00120     double bec_length = sqrt( HBAR/(mass*sqrt( omegaX*omegaX * ( 1 -
00121     omega*omega ) ) ) );
00121     xMax = 6*Rxy*a0x; //10*bec_length; //6*Rxy*a0x;
00122     yMax = 6*Rxy*a0y; //10*bec_length; //
00123     appendData(&params,"xMax",xMax);
00124     appendData(&params,"yMax",yMax);
00125
00126     double pxMax, pyMax;
00127     pxMax = (PI/xMax)*(xDim>>1);
00128     pyMax = (PI/yMax)*(yDim>>1);
00129     appendData(&params,"pyMax",pyMax);
00130     appendData(&params,"pxMax",pxMax);
00131
00132     dx = xMax/(xDim>>1);
00133     dy = yMax/(yDim>>1);
00134     appendData(&params,"dx",dx);
00135     appendData(&params,"dy",dy);
00136
00137     double dpx, dpy;
00138     dpx = PI/(xMax);
00139     dpy = PI/(yMax);
00140     appendData(&params,"dpx",dpx);
00141     appendData(&params,"dpy",dpy);
00142
00143     //printf("a0x=%e a0y=%e \n dx=%e dy=%e \n R_xy=%e\n",a0x,a0y,dx,dy,Rxy);
00144     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00145
00146     //double *x,*y,*xp,*yp;
00147     x = (double *) malloc(sizeof(double) * xDim);
00148     y = (double *) malloc(sizeof(double) * yDim);
00149     xp = (double *) malloc(sizeof(double) * xDim);
00150     yp = (double *) malloc(sizeof(double) * yDim);
00151
00152     /*

```

```

00153     * Pos and Mom grids
00154     */
00155     for(i=0; i<xDim/2; ++i){
00156         x[i] = -xMax + (i+1)*dx;
00157         x[i + (xDim/2)] = (i+1)*dx;
00158
00159         y[i] = -yMax + (i+1)*dy;
00160         y[i + (yDim/2)] = (i+1)*dy;
00161
00162         xp[i] = (i+1)*dpx;
00163         xp[i + (xDim/2)] = -pxMax + (i+1)*dpx;
00164
00165         yp[i] = (i+1)*dpy;
00166         yp[i + (yDim/2)] = -pyMax + (i+1)*dpy;
00167     }
00168
00169     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00170
00171     /* Initialise wavefunction, momentum and position operators on host */
00172     Energy = (double*) malloc(sizeof(double) * gSize);
00173     r = (double *) malloc(sizeof(double) * gSize);
00174     Phi = (double *) malloc(sizeof(double) * gSize);
00175     wfc = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00176     wfc_backup = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * (gSize/
threads));
00177     K = (double *) malloc(sizeof(double) * gSize);
00178     V = (double *) malloc(sizeof(double) * gSize);
00179     V_opt = (double *) malloc(sizeof(double) * gSize);
00180     GK = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00181     GV = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00182     EK = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00183     EV = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00184     EV_opt = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00185     xPy = (double *) malloc(sizeof(double) * gSize);
00186     yPx = (double *) malloc(sizeof(double) * gSize);
00187     ExPy = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00188     EyPx = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00189     EappliedField = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00190
00191     /* Initialise wfc, EKp, and EVr buffers on GPU */
00192     cudaMalloc((void**) &Energy_gpu, sizeof(double) * gSize);
00193     cudaMalloc((void**) &wfc_gpu, sizeof(cufftDoubleComplex) * gSize);
00194     cudaMalloc((void**) &K_gpu, sizeof(cufftDoubleComplex) * gSize);
00195     cudaMalloc((void**) &V_gpu, sizeof(cufftDoubleComplex) * gSize);
00196     cudaMalloc((void**) &xPy_gpu, sizeof(cufftDoubleComplex) * gSize);
00197     cudaMalloc((void**) &yPx_gpu, sizeof(cufftDoubleComplex) * gSize);
00198     cudaMalloc((void**) &par_sum, sizeof(cufftDoubleComplex) * (gSize/
threads));
00199     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00200
00201     #ifdef __linux
00202     int cores = omp_get_num_procs();
00203     appendData(&params, "Cores_Total", cores);
00204     appendData(&params, "Cores_Max", cores/2);
00205     omp_set_num_threads(cores/2);
00206     #pragma omp parallel for private(j)
00207     #endif
00208     for( i=0; i < xDim; i++){
00209         for( j=0; j < yDim; j++){
00210             Phi[(i*yDim + j)] = fmod(1*atan2(y[j], x[i]), 2*PI);
00211
00212             wfc[(i*yDim + j)].x = exp(-( pow((x[i])/(Rxy*a0x), 2) + pow((
y[j])/(Rxy*a0y), 2) ) ) * cos(Phi[(i*xDim + j)]);
00213             wfc[(i*yDim + j)].y = -exp(-( pow((x[i])/(Rxy*a0x), 2) + pow((
y[j])/(Rxy*a0y), 2) ) ) * sin(Phi[(i*xDim + j)]);
00214
00215             V[(i*yDim + j)] = 0.5*mass*( pow(omegaX*(x[i]+xOffset), 2) + pow(
gammaY*omegaY*(y[j]+yOffset), 2) );
00216             K[(i*yDim + j)] = (HBAR*HBAR/(2*mass))*(xp[i]*xp[i] +
yp[j]*yp[j]);
00217
00218             GV[(i*yDim + j)].x = exp( -V[(i*xDim + j)]*(gdt/(2*HBAR)));
00219             GK[(i*yDim + j)].x = exp( -K[(i*xDim + j)]*(gdt/HBAR));
00220             GV[(i*yDim + j)].y = 0.0;
00221             GK[(i*yDim + j)].y = 0.0;
00222
00223             xPy[(i*yDim + j)] = x[i]*yp[j];
00224             yPx[(i*yDim + j)] = -y[j]*xp[i];
00225
00226             EV[(i*yDim + j)].x = cos( -V[(i*xDim + j)]*(dt/(2*HBAR)));
00227             EV[(i*yDim + j)].y = sin( -V[(i*xDim + j)]*(dt/(2*HBAR)));
00228             EK[(i*yDim + j)].x = cos( -K[(i*xDim + j)]*(dt/HBAR));
00229             EK[(i*yDim + j)].y = sin( -K[(i*xDim + j)]*(dt/HBAR));
00230
00231             ExPy[(i*yDim + j)].x = cos(-omega*omegaX*xPy[(i*xDim + j)]*
dt);
00232             ExPy[(i*yDim + j)].y = sin(-omega*omegaX*xPy[(i*xDim + j)]*

```



```

    dt);
00233         EyPx[(i*yDim + j)].x=cos(-omega*omegaX*yPx[(i*xDim + j)]*
dt);
00234         EyPx[(i*yDim + j)].y=sin(-omega*omegaX*yPx[(i*xDim + j)]*
dt);
00235
00236         sum+=sqrt(wfc[(i*xDim + j)].x*wfc[(i*xDim + j)].x + wfc[(i*xDim + j)].
y*wfc[(i*xDim + j)].y);
00237     }
00238 }
00239 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00240 //hdfWriteDouble(xDim, V, 0, "V_0");
00241 //hdfWriteComplex(xDim, wfc, 0, "wfc_0");
00242 FileIO::writeOutDouble(buffer, "V", V, xDim*yDim, 0);
00243 FileIO::writeOutDouble(buffer, "V_opt", V_opt, xDim*yDim, 0);
00244 FileIO::writeOutDouble(buffer, "K", K, xDim*yDim, 0);
00245 FileIO::writeOutDouble(buffer, "xPy", xPy, xDim*yDim, 0);
00246 FileIO::writeOutDouble(buffer, "yPx", yPx, xDim*yDim, 0);
00247 FileIO::writeOutDouble(buffer, "WFC", wfc, xDim*yDim, 0);
00248 FileIO::writeOutDouble(buffer, "ExPy", ExPy, xDim*yDim, 0);
00249 FileIO::writeOutDouble(buffer, "EyPx", EyPx, xDim*yDim, 0);
00250 FileIO::writeOutDouble(buffer, "Phi", Phi, xDim*yDim, 0);
00251 FileIO::writeOutDouble(buffer, "x", r, xDim*yDim, 0);
00252 FileIO::writeOutDouble(buffer, "x", x, xDim, 0);
00253 FileIO::writeOutDouble(buffer, "y", y, yDim, 0);
00254 FileIO::writeOutDouble(buffer, "px", xp, xDim, 0);
00255 FileIO::writeOutDouble(buffer, "py", yp, yDim, 0);
00256 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00257
00258 //free(V);
00259 free(K); free(r); //free(Phi);
00260
00261 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00262
00263 sum=sqrt(sum*dx*dy);
00264 //#pragma omp parallel for reduction(+:sum) private(j)
00265 for (i = 0; i < xDim; i++){
00266     for (j = 0; j < yDim; j++){
00267         wfc[(i*yDim + j)].x = (wfc[(i*yDim + j)].x)/(sum);
00268         wfc[(i*yDim + j)].y = (wfc[(i*yDim + j)].y)/(sum);
00269     }
00270 }
00271
00272 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00273
00274 result = cufftPlan2d(&plan_2d, xDim, yDim, CUFFT_Z2Z);
00275 if(result != CUFFT_SUCCESS){
00276     printf("Result:=%d\n", result);
00277     printf("Error: Could not execute cufftPlan2d(%s ,%d, %d).\n", "plan_2d", (unsigned int)xDim, (
unsigned int)yDim);
00278     return -1;
00279 }
00280
00281 result = cufftPlan1d(&plan_1d, xDim, CUFFT_Z2Z, yDim);
00282 if(result != CUFFT_SUCCESS){
00283     printf("Result:=%d\n", result);
00284     printf("Error: Could not execute cufftPlan3d(%s ,%d, %d).\n", "plan_1d", (unsigned int)xDim, (
unsigned int)yDim);
00285     return -1;
00286 }
00287
00288 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00289
00290 return 0;
00291 }

```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 6.49.1.5 int isError ( int result, char \* c )

Definition at line 58 of file [split\\_op.cu](#).

References [result](#).

```

00058     {
00059         if(result!=0){printf("Error has occurred for method %s with return type %d\n",
c,result);
00060             exit(result);
00061         }
00062         return result;

```

```
00063 }
```

#### 6.49.1.6 int main ( int argc, char \*\* argv )

Definition at line 876 of file `split_op.cu`.

References `ang_mom`, `atoms`, `buffer`, `device`, `EK`, `err`, `esteps`, `EV`, `evolve()`, `ExPy`, `EyPx`, `GK`, `gpe`, `gsteps`, `GV`, `initArr()`, `initialise()`, `K_gpu`, `omegaX`, `omegaY`, `par_sum`, `parseArgs()`, `print`, `read_wfc`, `FileIO::readIn()`, `timeTotal`, `V_gpu`, `V_opt`, `wfc`, `wfc_gpu`, `FileIO::writeOutDouble()`, `FileIO::writeOutParam()`, `x`, `xDim`, `xPy`, `xPy_gpu`, `y`, `yDim`, `yPx`, and `yPx_gpu`.

```
00876                                     {
00877
00878     time_t start,fin;
00879     time(&start);
00880     printf("Start: %s\n", ctime(&start));
00881     initArr(&params,32);
00882     //appendData(&params,ctime(&start),0.0);
00883     parseArgs(argc,argv);
00884     cudaSetDevice(device);
00885     //*****//
00886     /*
00887     * Initialise the Params data structure to track params and variables
00888     */
00889     //*****//
00890     //paramS = (Params *) malloc(sizeof(Params));
00891     //strcpy(paramS->data,"INIT");
00892     //paramS->next=NULL;
00893
00894     initialise(omegaX,omegaY,atoms);
00895     timeTotal = 0.0;
00896     //*****//
00897     /*
00898     * Groundstate finder section
00899     */
00900     //*****//
00901     FileIO::writeOutParam(buffer, params, "Params.dat");
00902     if(read_wfc == 1){
00903         printf("Loading wavefunction...");
00904         wfc=FileIO::readIn("wfc_load","wfc_load",xDim,
00905 yDim);
00906         printf("Wavefunction loaded.\n");
00907     }
00908     double2 ph;
00909     double x_0,y_0;
00910     x_0 = 0;//(0.5*xDim)*dx;
00911     y_0 = 0;//(0.5*yDim)*dy;
00912     /* for(int i=0; i < xDim; i++){
00913         for(int j=0; j < yDim; j++){
00914             ph.x = cos( fmod( 0*atan2( y[j] - y_0, x[i] - x_0 ), 2*PI) );
00915             ph.y = -sin( fmod( 0*atan2( y[j] - y_0, x[i] - x_0 ), 2*PI) );
00916             wfc[(i*yDim + j)] = Minions::complexMult( wfc[(i*yDim + j)], ph );
00917         }
00918     }
00919     printf("l=%e\n",l);
00920     */ if(gsteps > 0){
00921         err=cudaMemcpy(K_gpu, GK, sizeof(cufftDoubleComplex)*xDim*
00922 yDim, cudaMemcpyHostToDevice);
00923         if(err!=cudaSuccess)
00924             exit(1);
00925         err=cudaMemcpy(V_gpu, GV, sizeof(cufftDoubleComplex)*xDim*
00926 yDim, cudaMemcpyHostToDevice);
00927         if(err!=cudaSuccess)
00928             exit(1);
00929         err=cudaMemcpy(xPy_gpu, xPy, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
00930         if(err!=cudaSuccess)
00931             exit(1);
00932         err=cudaMemcpy(yPx_gpu, yPx, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
00933         if(err!=cudaSuccess)
00934             exit(1);
00935         err=cudaMemcpy(wfc_gpu, wfc, sizeof(cufftDoubleComplex)*
00936 xDim*yDim, cudaMemcpyHostToDevice);
00937         if(err!=cudaSuccess)
00938             exit(1);
00939         evolve(wfc_gpu, K_gpu, V_gpu, yPx_gpu,
00940 xPy_gpu, par_sum, xDim*yDim, gsteps, 128, 0, ang_mom,
```

```

    gpe, print, atoms, 0);
00938     cudaMemcpy(wfc, wfc_gpu, sizeof(cufftDoubleComplex)*xDim*
yDim, cudaMemcpyDeviceToHost);
00939 }
00940
00941     free(GV); free(GK); free(xPy); free(yPx);
00942
00943     //*****//
00944     /*
00945     * Evolution
00946     */
00947     //*****//
00948     if(esteps > 0){
00949         err=cudaMemcpy(xPy_gpu, ExPy, sizeof(cufftDoubleComplex)*
xDim*yDim, cudaMemcpyHostToDevice);
00950         if(err!=cudaSuccess)
00951             exit(1);
00952         err=cudaMemcpy(yPx_gpu, EyPx, sizeof(cufftDoubleComplex)*
xDim*yDim, cudaMemcpyHostToDevice);
00953         if(err!=cudaSuccess)
00954             exit(1);
00955         err=cudaMemcpy(xPy_gpu, ExPy, sizeof(cufftDoubleComplex)*
xDim*yDim, cudaMemcpyHostToDevice);
00956         if(err!=cudaSuccess)
00957             exit(1);
00958         err=cudaMemcpy(yPx_gpu, EyPx, sizeof(cufftDoubleComplex)*
xDim*yDim, cudaMemcpyHostToDevice);
00959         if(err!=cudaSuccess)
00960             exit(1);
00961         err=cudaMemcpy(K_gpu, EK, sizeof(cufftDoubleComplex)*xDim*
yDim, cudaMemcpyHostToDevice);
00962         if(err!=cudaSuccess)
00963             exit(1);
00964         err=cudaMemcpy(V_gpu, EV, sizeof(cufftDoubleComplex)*xDim*
yDim, cudaMemcpyHostToDevice);
00965         if(err!=cudaSuccess)
00966             exit(1);
00967         err=cudaMemcpy(wfc_gpu, wfc, sizeof(cufftDoubleComplex)*
xDim*yDim, cudaMemcpyHostToDevice);
00968         if(err!=cudaSuccess)
00969             exit(1);
00970
00971         //delta_define(x, y, (523.6667 - 512 + x0_shift)*dx, (512.6667 - 512 + y0_shift)*dy, V_opt);
00972         FileIO::writeOutDouble(buffer, "V_opt", V_opt,
xDim*yDim, 0);
00973         evolve(wfc_gpu, K_gpu, V_gpu, yPx_gpu,
xPy_gpu, par_sum, xDim*yDim, esteps, 128, 1, ang_mom,
gpe, print, atoms, 0);
00974     }
00975     free(EV); free(EK); free(ExPy); free(EyPx);
00976     free(x); free(y);
00977     cudaFree(wfc_gpu); cudaFree(K_gpu); cudaFree(V_gpu); cudaFree(
yPx_gpu); cudaFree(xPy_gpu); cudaFree(par_sum);
00979     time(&fin);
00980     //appendData(&params, ctime(&fin), 0.0);
00981     printf("Finish: %s\n", ctime(&fin));
00982     printf("Total time: %ld seconds\n ", (long)fin-start);
00983     //appendData(&params, "t_duration", fin-start);
00984     return 0;
00985 }
00986 }

```

Here is the call graph for this function:

**6.49.1.7** void optLatSetup ( struct Tracker::Vortex centre, double \* V, struct Tracker::Vortex \* vArray, int num\_vortices, double theta\_opt, double intensity, double \* v\_opt, double \* x, double \* y )

Matches the optical lattice to the vortex lattice.

Definition at line 596 of file split\_op.cu.

References [appendData\(\)](#), [buffer](#), [Tracker::Vortex::coords](#), [dt](#), [dx](#), [dy](#), [EV\\_opt](#), [HBAR](#), [vis::i](#), [observables::k\\_mag](#), [PI](#), [sepMinEpsilon](#), [Tracker::vortSepAvg\(\)](#), [FileIO::writeOut\(\)](#), [xDim](#), and [yDim](#).

Referenced by [evolve\(\)](#).

```

00596
00597     int i, j;

```

```

00598     double sepMin = Tracker::vortSepAvg(vArray,centre,num_vortices);
00599     sepMin = sepMin*(1 + sepMinEpsilon);
00600     appendData(&params,"Vort_sep",(double)sepMin);
00601     /*
00602     * Defining the necessary k vectors for the optical lattice
00603     */
00604     double k_mag = ((2*PI/(sepMin*dx))/2)*(2/sqrt(3)); // Additional /2 as a result of lambda/2
    period
00605     double2* k = (double2*) malloc(sizeof(double2)*3);
00606     appendData(&params,"kmag",(double)k_mag);
00607     k[0].x = k_mag * cos(0*PI/3 + theta_opt);
00608     k[0].y = k_mag * sin(0*PI/3 + theta_opt);
00609     k[1].x = k_mag * cos(2*PI/3 + theta_opt);
00610     k[1].y = k_mag * sin(2*PI/3 + theta_opt);
00611     k[2].x = k_mag * cos(4*PI/3 + theta_opt);
00612     k[2].y = k_mag * sin(4*PI/3 + theta_opt);
00613
00614     double2 *r_opt = (double2*) malloc(sizeof(double2)*xDim);
00615
00616     /* for (int ii = 0; ii < xDim; ++ii){
00617         r_opt[ii].x = 0.0 + (xDim/sepMin)*PI*(ii-centre.coords.x)/(xDim-1);
00618         r_opt[ii].y = 0.0 + (xDim/sepMin)*PI*(ii-centre.coords.y)/(yDim-1);
00619     }
00620     */
00621     FileIO::writeOut(buffer,"r_opt",r_opt,xDim,0);
00622     appendData(&params,"k[0].x",(double)k[0].x);
00623     appendData(&params,"k[0].y",(double)k[0].y);
00624     appendData(&params,"k[1].x",(double)k[1].x);
00625     appendData(&params,"k[1].y",(double)k[1].y);
00626     appendData(&params,"k[2].x",(double)k[2].x);
00627     appendData(&params,"k[2].y",(double)k[2].y);
00628
00629     double x_shift = dx*(9+(0.5*xDim-1) - centre.coords.x);//sin(theta_opt)*(sepMin);
00630     double y_shift = dy*(0+(0.5*yDim-1) - centre.coords.y);//cos(theta_opt)*(sepMin);
00631
00632     printf("Xs=%e\nYs=%e\n",x_shift,y_shift);
00633
00634     /*#pragma omp parallel for private(j)
00635     for ( j=0; j<yDim; ++j ){
00636         for ( i=0; i<xDim; ++i ){
00637             v_opt[j*xDim + i] = intensity*(
00638                 pow( abs( cos( k[0].x*( x[i] + x_shift ) + k[0].
    y*( y[j] + y_shift ) ) ), 2)
00639                 + pow( abs( cos( k[1].x*( x[i] + x_shift ) + k[1].y*(
    y[j] + y_shift ) ) ), 2)
00640                 + pow( abs( cos( k[2].x*( x[i] + x_shift ) + k[2].y*(
    y[j] + y_shift ) ) ), 2)
00641                 /*
    y_shift ) ) ), 2)
00642                 + pow( abs( cos( k[0].x*( r_opt[i].x + x_shift ) + k[0].y*( r_opt[j].y +
    y_shift ) ) ), 2)
00643                 + pow( abs( cos( k[1].x*( r_opt[i].x + x_shift ) + k[1].y*( r_opt[j].y + y_shift )
    ) ), 2)
00644                 */
    );
00645     EV_opt[(j*xDim + i)].x=cos( -(V[(j*xDim + i)] + v_opt[j*xDim +
    i])*(dt/(2*HBAR))) );
00646     EV_opt[(j*xDim + i)].y=sin( -(V[(j*xDim + i)] + v_opt[j*xDim + i])*(
    dt/(2*HBAR))) );
00647     }
00648     }
00649
00650 }

```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 6.49.1.8 int parseArgs ( int argc, char \*\* argv )

Definition at line 714 of file `split_op.cu`.

References `ang_mom`, `angle_sweep`, `appendData()`, `atoms`, `device`, `dt`, `esteps`, `gammaY`, `gdt`, `gpe`, `gsteps`, `interaction`, `kick_it`, `l`, `laser_power`, `omega`, `omegaX`, `omegaY`, `omegaZ`, `print`, `read_wfc`, `sepMinEpsilon`, `write_it`, `x0_shift`, `xDim`, `y0_shift`, and `yDim`.

Referenced by `main()`.

```

00714     {
00715     int opt;
00716     while ((opt = getopt (argc, argv, "d:x:y:w:G:g:e:T:t:n:p:r:o:L:l:s:i:P:X:Y:O:k:W:U:V:S:")) != -1) {

```

```

00717     switch (opt)
00718     {
00719         case 'x':
00720             xDim = atoi(optarg);
00721             printf("Argument for x is given as %d\n", xDim);
00722             appendData(&params, "xDim", (double)xDim);
00723             break;
00724         case 'y':
00725             yDim = atoi(optarg);
00726             printf("Argument for y is given as %d\n", yDim);
00727             appendData(&params, "yDim", (double)yDim);
00728             break;
00729         case 'w':
00730             omega = atof(optarg);
00731             printf("Argument for OmegaRotate is given as %E\n", omega);
00732             appendData(&params, "omega", omega);
00733             break;
00734         case 'G':
00735             gammaY = atof(optarg);
00736             printf("Argument for gamma is given as %E\n", gammaY);
00737             appendData(&params, "gammaY", gammaY);
00738             break;
00739         case 'g':
00740             gsteps = atof(optarg);
00741             printf("Argument for Groundsteps is given as %ld\n", gsteps);
00742             appendData(&params, "gsteps", gsteps);
00743             break;
00744         case 'e':
00745             esteps = atof(optarg);
00746             printf("Argument for EvSteps is given as %ld\n", esteps);
00747             appendData(&params, "esteps", esteps);
00748             break;
00749         case 'T':
00750             gdt = atof(optarg);
00751             printf("Argument for groundstate Timestep is given as %E\n", gdt);
00752             appendData(&params, "gdt", gdt);
00753             break;
00754         case 't':
00755             dt = atof(optarg);
00756             printf("Argument for Timestep is given as %E\n", dt);
00757             appendData(&params, "dt", dt);
00758             break;
00759         case 'd':
00760             device = atoi(optarg);
00761             printf("Argument for device is given as %d\n", device);
00762             appendData(&params, "device", device);
00763             break;
00764         case 'n':
00765             atoms = atof(optarg);
00766             printf("Argument for atoms is given as %ld\n", atoms);
00767             appendData(&params, "atoms", atoms);
00768             break;
00769         case 'r':
00770             read_wfc = atoi(optarg);
00771             printf("Argument for ReadIn is given as %d\n", read_wfc);
00772             appendData(&params, "read_wfc", (double)read_wfc);
00773             break;
00774         case 'p':
00775             print = atoi(optarg);
00776             printf("Argument for Printout is given as %d\n", print);
00777             appendData(&params, "print_out", (double)print);
00778             break;
00779         case 'L':
00780             l = atof(optarg);
00781             printf("Vortex winding is given as : %E\n", l);
00782             appendData(&params, "winding", l);
00783             break;
00784         case 'l':
00785             ang_mom = atoi(optarg);
00786             printf("Angular Momentum mode engaged: %d\n", ang_mom);
00787             appendData(&params, "corotating", (double)ang_mom);
00788             break;
00789         case 's':
00790             gpe = atoi(optarg);
00791             printf("Non-linear mode engaged: %d\n", gpe);
00792             appendData(&params, "gpe", gpe);
00793             break;
00794         case 'o':
00795             omegaZ = atof(optarg);
00796             printf("Argument for OmegaZ is given as %E\n", omegaZ);
00797             appendData(&params, "omegaZ", omegaZ);
00798             break;
00799         case 'i':
00800             interaction = atof(optarg);
00801             printf("Argument for interaction scaling is %E\n", interaction);
00802             appendData(&params, "int_scaling", interaction);
00803             break;

```

```

00804         case 'P':
00805             laser_power = atof(optarg);
00806             printf("Argument for laser power is %E\n", laser_power);
00807             appendData(&params, "laser_power", laser_power);
00808             break;
00809         case 'X':
00810             omegaX = atof(optarg);
00811             printf("Argument for omegaX is %E\n", omegaX);
00812             appendData(&params, "omegaX", omegaX);
00813             break;
00814         case 'Y':
00815             omegaY = atof(optarg);
00816             printf("Argument for omegaY is %E\n", omegaY);
00817             appendData(&params, "omegaY", omegaY);
00818             break;
00819         case 'O':
00820             angle_sweep = atof(optarg);
00821             printf("Argument for angle_sweep is %E\n", angle_sweep);
00822             appendData(&params, "angle_sweep", angle_sweep);
00823             break;
00824         case 'k':
00825             kick_it = atoi(optarg);
00826             printf("Argument for kick_it is %i\n", kick_it);
00827             appendData(&params, "kick_it", kick_it);
00828             break;
00829         case 'W':
00830             write_it = atoi(optarg);
00831             printf("Argument for write_it is %i\n", write_it);
00832             appendData(&params, "write_it", write_it);
00833             break;
00834         case 'U':
00835             x0_shift = atof(optarg);
00836             printf("Argument for x0_shift is %lf\n", x0_shift);
00837             appendData(&params, "x0_shift", x0_shift);
00838             break;
00839         case 'V':
00840             y0_shift = atof(optarg);
00841             printf("Argument for y0_shift is %lf\n", y0_shift);
00842             appendData(&params, "y0_shift", y0_shift);
00843             break;
00844         case 'S':
00845             sepMinEpsilon = atof(optarg);
00846             printf("Argument for sepMinEpsilon is %lf\n", sepMinEpsilon);
00847             appendData(&params, "sepMinEpsilon", sepMinEpsilon);
00848             break;
00849         case '?':
00850             if (optopt == 'c') {
00851                 fprintf(stderr, "Option -%c requires an argument.\n", optopt);
00852             } else if (isprint (optopt)) {
00853                 fprintf(stderr, "Unknown option '-%c'.\n", optopt);
00854             } else {
00855                 fprintf(stderr, "Unknown option character '\\%x'.\n", optopt);
00856             }
00857             return -1;
00858         default:
00859             abort ();
00860     }
00861 }
00862 return 0;
00863 }

```

Here is the call graph for this function:

Here is the caller graph for this function:

#### 6.49.1.9 void parSum ( double2 \* gpuWfc, double2 \* gpuParSum, int xDim, int yDim, int threads )

Definition at line 572 of file [split\\_op.cu](#).

References [dx](#), [dy](#), [threads](#), and [yDim](#).

Referenced by [evolve\(\)](#).

```

00572                                     {
00573         int grid_tmp = xDim*yDim;
00574         int block = grid_tmp/threads;
00575         int thread_tmp = threads;
00576         int pass = 0;
00577         while((double)grid_tmp/threads > 1.0){
00578             if(grid_tmp == xDim*yDim){
00579                 multipass<<<block, threads, threads*sizeof(double2)>>>(&gpuWfc[0], &gpuParSum[0], pass);

```

```

00580         }
00581         else{
00582             multipass<<<block,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0],pass
);
00583         }
00584         grid_tmp /= threads;
00585         block = (int) ceil((double)grid_tmp/threads);
00586         pass++;
00587     }
00588     thread_tmp = grid_tmp;
00589     multipass<<<1,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0], pass);
00590     scalarDiv_wfcNorm<<<grid,threads>>>(gpuWfc, dx*dy, gpuParSum, gpuWfc);
00591 }

```

Here is the caller graph for this function:

#### 6.49.1.10 `template<typename T> void parSum ( T * gpuToSumArr, T * gpuParSum, int xDim, int yDim, int threads )`

Definition at line 692 of file [split\\_op.cu](#).

References [dx](#), [dy](#), [threads](#), and [yDim](#).

```

00692                                     {
00693         int grid_tmp = xDim*yDim;
00694         int block = grid_tmp/threads;
00695         int thread_tmp = threads;
00696         int pass = 0;
00697         while((double)grid_tmp/threads > 1.0){
00698             if(grid_tmp == xDim*yDim){
00699                 multipass<<<block,threads,threads*sizeof(T)>>>(&gpuToSumArr[0],&gpuParSum[0
],pass);
00700             }
00701             else{
00702                 multipass<<<block,thread_tmp,thread_tmp*sizeof(T)>>>(&gpuParSum[0],&
gpuParSum[0],pass);
00703             }
00704             grid_tmp /= threads;
00705             block = (int) ceil((double)grid_tmp/threads);
00706             pass++;
00707         }
00708         thread_tmp = grid_tmp;
00709         multipass<<<1,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0], pass);
00710         scalarDiv_wfcNorm<<<grid,threads>>>(gpuToSumArr, dx*dy, gpuParSum, gpuToSumArr);
00711 }

```

## 6.49.2 Variable Documentation

### 6.49.2.1 `double a0x`

Definition at line 54 of file [split\\_op.cu](#).

Referenced by [initialise\(\)](#).

### 6.49.2.2 `double a0y`

Definition at line 54 of file [split\\_op.cu](#).

Referenced by [initialise\(\)](#).

### 6.49.2.3 `double angle_sweep`

Definition at line 49 of file [split\\_op.cu](#).

Referenced by [evolve\(\)](#), and [parseArgs\(\)](#).

### 6.49.2.4 `char buffer[100]`

Definition at line 42 of file [split\\_op.cu](#).

Referenced by [evolve\(\)](#), [initialise\(\)](#), [main\(\)](#), and [optLatSetup\(\)](#).

#### 6.49.2.5 int device

Definition at line 44 of file [split\\_op.cu](#).

Referenced by [main\(\)](#), and [parseArgs\(\)](#).

#### 6.49.2.6 double gammaY

Definition at line 46 of file [split\\_op.cu](#).

Referenced by [initialise\(\)](#), and [parseArgs\(\)](#).

#### 6.49.2.7 int kick\_it

Definition at line 45 of file [split\\_op.cu](#).

Referenced by [evolve\(\)](#), and [parseArgs\(\)](#).

#### 6.49.2.8 double omega

Definition at line 47 of file [split\\_op.cu](#).

Referenced by [evolve\(\)](#), [initialise\(\)](#), and [parseArgs\(\)](#).

#### 6.49.2.9 Params\* paramS

Definition at line 50 of file [split\\_op.cu](#).

#### 6.49.2.10 Array params

Definition at line 51 of file [split\\_op.cu](#).

#### 6.49.2.11 double Rxy

Definition at line 53 of file [split\\_op.cu](#).

Referenced by [initialise\(\)](#).

#### 6.49.2.12 double sepMinEpsilon =0.0

Definition at line 55 of file [split\\_op.cu](#).

Referenced by [optLatSetup\(\)](#), and [parseArgs\(\)](#).

#### 6.49.2.13 double timeTotal

Definition at line 48 of file [split\\_op.cu](#).

Referenced by [main\(\)](#).

#### 6.49.2.14 int verbose

Definition at line 43 of file [split\\_op.cu](#).



## 6.49.2.15 double x0\_shift

Definition at line 52 of file [split\\_op.cu](#).

Referenced by [parseArgs\(\)](#).

## 6.49.2.16 double y0\_shift

Definition at line 52 of file [split\\_op.cu](#).

Referenced by [parseArgs\(\)](#).

## 6.50 split\_op.cu

```

00001  /*
00002  * split_op.cu - GPUE: Split Operator based GPU solver for Nonlinear
00003  Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00004  <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00005
00006  Redistribution and use in source and binary forms, with or without
00007  modification, are permitted provided that the following conditions are
00008  met:
00009
00010  1. Redistributions of source code must retain the above copyright
00011  notice, this list of conditions and the following disclaimer.
00012
00013  2. Redistributions in binary form must reproduce the above copyright
00014  notice, this list of conditions and the following disclaimer in the
00015  documentation and/or other materials provided with the distribution.
00016
00017  3. Neither the name of the copyright holder nor the names of its
00018  contributors may be used to endorse or promote products derived from
00019  this software without specific prior written permission.
00020
00021  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024  PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025  HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027  TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028  PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029  LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030  NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031  SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032  */
00033
00034  #include "../include/split_op.h"
00035  #include "../include/kernels.h"
00036  #include "../include/constants.h"
00037  #include "../include/fileIO.h"
00038  #include "../include/tracker.h"
00039  #include "../include/minions.h"
00040  #include "../include/ds.h"
00041
00042  char buffer[100];
00043  int verbose;
00044  int device;
00045  int kick_it;
00046  double gammaY;
00047  double omega;
00048  double timeTotal;
00049  double angle_sweep;
00050  Params *paramS;
00051  Array params;
00052  double x0_shift, y0_shift;
00053  double Rxy;
00054  double a0x, a0y;
00055  double sepMinEpsilon=0.0;
00056  /* Buffer and FILE for IO */
00057
00058  int isError(int result, char* c){
00059      if(result!=0){printf("Error has occurred for method %s with return type %d\n",c,result);
00060          exit(result);
00061      }
00062      return result;
00063  }
00064  int initialise(double omegaX, double omegaY, int N){

```

```

00065 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
00066 unsigned int xD=1,yD=1,zD=1;
00067 threads = 128;
00068 unsigned int b = xDim*yDim/threads; //number of blocks in simulation
00069 unsigned long long maxElements = 65536*65536ULL; //largest number of elements
00070
00071 if( b < (1<<16) ){
00072     xD = b;
00073 }
00074 else if( (b >= (1<<16) ) && (b <= (maxElements)) ){
00075     int t1 = log(b)/log(2);
00076     float t2 = (float) t1/2;
00077     t1 = (int) t2;
00078     if(t2 > (float) t1){
00079         xD <= t1;
00080         yD <= (t1 + 1);
00081     }
00082     else if(t2 == (float) t1){
00083         xD <= t1;
00084         yD <= t1;
00085     }
00086 }
00087 else{
00088     printf("Outside range of supported indexing");
00089     exit(-1);
00090 }
00091 printf("Compute grid dimensions chosen as X=%d Y=%d\n",xD,yD);
00092
00093 grid.x=xD;
00094 grid.y=yD;
00095 grid.z=zD;
00096 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
00097
00098 unsigned int i,j; //Used in for-loops for indexing
00099
00100 unsigned int gSize = xDim*yDim;
00101 double xOffset, yOffset;
00102 xOffset=0.0;//5.0e-6;
00103 yOffset=0.0;//5.0e-6;
00104
00105 mass = 1.4431607e-25; //Rb 87 mass, kg
00106 appendData(&params,"Mass",mass);
00107 a_s = 4.67e-9;
00108 appendData(&params,"a_s",a_s);
00109
00110 double sum = 0.0;
00111
00112 a0x = sqrt(HBAR/(2*mass*omegaX));
00113 a0y = sqrt(HBAR/(2*mass*omegaY));
00114 appendData(&params,"a0x",a0x);
00115 appendData(&params,"a0y",a0y);
00116
00117 Rxy = pow(15,0.2)*pow(N*a_s*sqrt(mass*omegaZ/HBAR),0.2);
00118 appendData(&params,"Rxy",Rxy);
00119 //Rxy = pow(15,0.2)*pow(N*4.67e-9*sqrt(mass*pow(omegaX*omegaY,0.5)/HBAR),0.2);
00120 double bec_length = sqrt( HBAR/(mass*sqrt( omegaX*omegaX * ( 1 -
omega*omega ) ) ) );
00121 xMax = 6*Rxy*a0x;//10*bec_length;//6*Rxy*a0x;
00122 yMax = 6*Rxy*a0y;//10*bec_length;//
00123 appendData(&params,"xMax",xMax);
00124 appendData(&params,"yMax",yMax);
00125
00126 double pxMax, pyMax;
00127 pxMax = (PI/xMax)*(xDim>>1);
00128 pyMax = (PI/yMax)*(yDim>>1);
00129 appendData(&params,"pyMax",pyMax);
00130 appendData(&params,"pxMax",pxMax);
00131
00132 dx = xMax/(xDim>>1);
00133 dy = yMax/(yDim>>1);
00134 appendData(&params,"dx",dx);
00135 appendData(&params,"dy",dy);
00136
00137 double dpx, dpy;
00138 dpx = PI/(xMax);
00139 dpy = PI/(yMax);
00140 appendData(&params,"dpx",dpx);
00141 appendData(&params,"dpy",dpy);
00142
00143 //printf("a0x=%e a0y=%e \n dx=%e dy=%e\n R_xy=%e\n",a0x,a0y,dx,dy,Rxy);
00144 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
00145
00146 //double *x,*y,*xp,*yp;
00147 x = (double *) malloc(sizeof(double) * xDim);
00148 y = (double *) malloc(sizeof(double) * yDim);
00149 xp = (double *) malloc(sizeof(double) * xDim);
00150 yp = (double *) malloc(sizeof(double) * yDim);

```

```

00151
00152  /*
00153   * Pos and Mom grids
00154   */
00155  for(i=0; i<xDim/2; ++i){
00156      x[i] = -xMax + (i+1)*dx;
00157      x[i + (xDim/2)] = (i+1)*dx;
00158
00159      y[i] = -yMax + (i+1)*dy;
00160      y[i + (yDim/2)] = (i+1)*dy;
00161
00162      xp[i] = (i+1)*dpx;
00163      xp[i + (xDim/2)] = -pxMax + (i+1)*dpx;
00164
00165      yp[i] = (i+1)*dpy;
00166      yp[i + (yDim/2)] = -pyMax + (i+1)*dpy;
00167  }
00168
00169  //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
00170
00171  /* Initialise wavefunction, momentum and position operators on host */
00172  Energy = (double*) malloc(sizeof(double) * gSize);
00173  r = (double *) malloc(sizeof(double) * gSize);
00174  Phi = (double *) malloc(sizeof(double) * gSize);
00175  wfc = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00176  wfc_backup = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * (gSize/
00177  threads));
00178  K = (double *) malloc(sizeof(double) * gSize);
00179  V = (double *) malloc(sizeof(double) * gSize);
00180  V_opt = (double *) malloc(sizeof(double) * gSize);
00181  GK = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00182  GV = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00183  EK = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00184  EV = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00185  EV_opt = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00186  xPy = (double *) malloc(sizeof(double) * gSize);
00187  yPx = (double *) malloc(sizeof(double) * gSize);
00188  ExPy = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00189  EyPx = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00190  EappliedField = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
00191
00192  /* Initialise wfc, EKp, and EVr buffers on GPU */
00193  cudaMalloc((void**) &Energy_gpu, sizeof(double) * gSize);
00194  cudaMalloc((void**) &wfc_gpu, sizeof(cufftDoubleComplex) * gSize);
00195  cudaMalloc((void**) &K_gpu, sizeof(cufftDoubleComplex) * gSize);
00196  cudaMalloc((void**) &V_gpu, sizeof(cufftDoubleComplex) * gSize);
00197  cudaMalloc((void**) &xPy_gpu, sizeof(cufftDoubleComplex) * gSize);
00198  cudaMalloc((void**) &yPx_gpu, sizeof(cufftDoubleComplex) * gSize);
00199  cudaMalloc((void**) &par_sum, sizeof(cufftDoubleComplex) * (gSize/
00200  threads));
00201  //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
00202
00203  #ifdef __linux
00204  int cores = omp_get_num_procs();
00205  appendData(&params,"Cores_Total",cores);
00206  appendData(&params,"Cores_Max",cores/2);
00207  omp_set_num_threads(cores/2);
00208  #pragma omp parallel for private(j)
00209  #endif
00210  for( i=0; i < xDim; i++){
00211      for( j=0; j < yDim; j++){
00212          Phi[(i*yDim + j)] = fmod(l*atan2(y[j], x[i]),2*PI);
00213
00214          wfc[(i*yDim + j)].x = exp(-( pow((x[i])/(Rxy*a0x),2) + pow((
00215          y[j])/(Rxy*a0y),2) ) *cos(Phi[(i*xDim + j)]));
00216          wfc[(i*yDim + j)].y = -exp(-( pow((x[i])/(Rxy*a0x),2) + pow((
00217          y[j])/(Rxy*a0y),2) ) *sin(Phi[(i*xDim + j)]));
00218
00219          V[(i*yDim + j)] = 0.5*mass*( pow(omegaX*(x[i]+xOffset),2) + pow(
00220          gammaY*omegaY*(y[j]+yOffset),2) );
00221          K[(i*yDim + j)] = (HBAR*HBAR/(2*mass))*(xp[i]*xp[i] +
00222          yp[j]*yp[j]);
00223
00224          GV[(i*yDim + j)].x = exp( -V[(i*xDim + j)]*(gdt/(2*HBAR)));
00225          GK[(i*yDim + j)].x = exp( -K[(i*xDim + j)]*(gdt/HBAR));
00226          GV[(i*yDim + j)].y = 0.0;
00227          GK[(i*yDim + j)].y = 0.0;
00228
00229          xPy[(i*yDim + j)] = x[i]*yp[j];
00230          yPx[(i*yDim + j)] = -y[j]*xp[i];
00231
00232          EV[(i*yDim + j)].x=cos( -V[(i*xDim + j)]*(dt/(2*HBAR)));
00233          EV[(i*yDim + j)].y=sin( -V[(i*xDim + j)]*(dt/(2*HBAR)));
00234          EK[(i*yDim + j)].x=cos( -K[(i*xDim + j)]*(dt/HBAR));
00235          EK[(i*yDim + j)].y=sin( -K[(i*xDim + j)]*(dt/HBAR));
00236
00237          ExPy[(i*yDim + j)].x=cos(-omega*omegaX*xPy[(i*xDim + j)]*dt);

```

```

00232         ExPy[(i*yDim + j)].y=sin(-omega*omegaX*xPy[(i*xDim + j)]*dt);
00233         EyPx[(i*yDim + j)].x=cos(-omega*omegaX*yPx[(i*xDim + j)]*dt);
00234         EyPx[(i*yDim + j)].y=sin(-omega*omegaX*yPx[(i*xDim + j)]*dt);
00235
00236         sum+=sqrt(wfc[(i*xDim + j)].x*wfc[(i*xDim + j)].x + wfc[(i*xDim + j)].
y*wfc[(i*xDim + j)].y);
00237     }
00238 }
00239 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00240 //hdfWriteDouble(xDim, V, 0, "V_0");
00241 //hdfWriteComplex(xDim, wfc, 0, "wfc_0");
00242 FileIO::writeOutDouble(buffer, "V", V, xDim*yDim, 0);
00243 //FileIO::writeOutDouble(buffer, "V_opt", V_opt, xDim*yDim, 0);
00244 FileIO::writeOutDouble(buffer, "K", K, xDim*yDim, 0);
00245 FileIO::writeOutDouble(buffer, "xPy", xPy, xDim*yDim, 0);
00246 FileIO::writeOutDouble(buffer, "yPx", yPx, xDim*yDim, 0);
00247 FileIO::writeOutDouble(buffer, "WFC", wfc, xDim*yDim, 0);
00248 FileIO::writeOutDouble(buffer, "ExPy", ExPy, xDim*yDim, 0);
00249 FileIO::writeOutDouble(buffer, "EyPx", EyPx, xDim*yDim, 0);
00250 FileIO::writeOutDouble(buffer, "Phi", Phi, xDim*yDim, 0);
00251 FileIO::writeOutDouble(buffer, "x", x, xDim, 0);
00252 FileIO::writeOutDouble(buffer, "x", x, xDim, 0);
00253 FileIO::writeOutDouble(buffer, "y", y, yDim, 0);
00254 FileIO::writeOutDouble(buffer, "px", xp, xDim, 0);
00255 FileIO::writeOutDouble(buffer, "py", yp, yDim, 0);
00256 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00257
00258 //free(V);
00259 free(K); free(r); //free(Phi);
00260
00261 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00262
00263 sum=sqrt(sum*dx*dy);
00264 //#pragma omp parallel for reduction(+:sum) private(j)
00265 for (i = 0; i < xDim; i++){
00266     for (j = 0; j < yDim; j++){
00267         wfc[(i*yDim + j)].x = (wfc[(i*yDim + j)].x)/(sum);
00268         wfc[(i*yDim + j)].y = (wfc[(i*yDim + j)].y)/(sum);
00269     }
00270 }
00271
00272 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00273
00274 result = cufftPlan2d(&plan_2d, xDim, yDim, CUFFT_Z2Z);
00275 if(result != CUFFT_SUCCESS){
00276     printf("Result:=%d\n", result);
00277     printf("Error: Could not execute cufftPlan2d(%s ,%d, %d).\n", "plan_2d", (unsigned int)xDim, (
unsigned int)yDim);
00278     return -1;
00279 }
00280
00281 result = cufftPlan1d(&plan_1d, xDim, CUFFT_Z2Z, yDim);
00282 if(result != CUFFT_SUCCESS){
00283     printf("Result:=%d\n", result);
00284     printf("Error: Could not execute cufftPlan3d(%s ,%d ,%d).\n", "plan_1d", (unsigned int)xDim, (
unsigned int)yDim);
00285     return -1;
00286 }
00287
00288 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00289
00290 return 0;
00291 }
00292
00293 int evolve( cufftDoubleComplex *gpuWfc,
00294             cufftDoubleComplex *gpuMomentumOp,
00295             cufftDoubleComplex *gpuPositionOp,
00296             void *gpuldypx,
00297             void *gpuldxpy,
00298             cufftDoubleComplex *gpuParSum,
00299             int gridSize, int numSteps, int threads,
00300             unsigned int gstate, int lz, int nonlin, int printSteps, int N, unsigned int ramp){
00301
00302     //Because no two operations are created equally. Multiplimultiplication is faster than divisions.
00303     double renorm_factor_2d=1.0/pow(gridSize,0.5);
00304     double renorm_factor_1d=1.0/pow(xDim,0.5);
00305
00306     clock_t begin, end;
00307     double time_spent;
00308     double Dt;
00309     if(gstate==0){
00310         Dt = gdt;
00311         printf("Timestep for grounstate solver set as: %E\n",Dt);
00312     }
00313     else{
00314         Dt = dt;
00315         printf("Timestep for evolution set as: %E\n",Dt);

```

```

00316     }
00317     begin = clock();
00318     double omega_0=omega*omegaX;
00319
00320     #if 0
00321
00322     int gridSum = 1<<6;
00323     double *densitySubset = (double*) malloc(sizeof(double)*gridSum);
00324     #pragma omp parallel for private(k)
00325     for (int j=0; j<gridSum; ++j){
00326         for (int k=0; k<gridSum; ++k){
00327             densitySubset[j*gridSum + k] = Minions::psi2(wfc[ ( (
yDim/2) - (gridSum/2) + j ) * yDim + ( (xDim/2) - (gridSum/2) + k )]);
00328         }
00329     }
00330     xi = 1/sqrt(8*PI*a_s*Minions::sumAvg(densitySubset,gridSum)/(
dx*dy)); //defined central condensate density
00331     printf("Avg healing length at centre=%E\n",xi);
00332     #endif
00333
00338     //Double buffering and will attempt to thread free and calloc operations to hide time penalty. Or may
not bother.
00339     int num_vortices[2] = {0,0};
00340     int num_latt_max = 0;
00341     int* vortexLocation; //binary matrix of size xDim*yDim, 1 for vortex at specified index, 0 otherwise
00342     int* olMaxLocation = (int*) calloc(xDim*yDim,sizeof(int));
00343
00344     struct Tracker::Vortex central_vortex; //vortex closest to the central position
00345     double vort_angle; //Angle of vortex lattice. Add to optical lattice for alignment.
00346     struct Tracker::Vortex *vortCoords = NULL; //array of vortex coordinates from
vortexLocation 1's
00347     struct Tracker::Vortex *vortCoordsP = NULL; //Previous array of vortex coordinates from
vortexLocation 1's
00348     int2 *olCoords = NULL; //array of vortex coordinates from vortexLocation 1's
00349     int2 *vortDelta = NULL;
00350
00351     double vortOLSigma=0.0;
00352     double sepAvg = 0.0;
00353
00354     int num_kick = 0;
00355     double t_kick = (2*PI/omega_0)/(6*Dt);
00356
00357     for(int i=0; i < numSteps; ++i){
00358         if ( ramp == 1 ){
00359             omega_0=omegaX*((omega-0.39)*((double)i/(double)(numSteps)) + 0.39); //Adjusts omega for
the appropriate trap frequency.
00360         }
00361         if(i % printSteps == 0){
00362             printf("Step: %d      Omega: %lf\n",i,omega_0/omegaX);
00363             cudaMemcpy(wfc, gpuWfc, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);
00364             end = clock();
00365             time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
00366             printf("Time spent: %lf\n",time_spent);
00367             char* fileName = "";
00368             printf("ramp=%d      gstate=%d      rg=%d      \n",ramp,gstate,ramp | (gstate<<1));
00369             switch ( ramp | (gstate<<1) ){
00370                 case 0:
00371                     fileName = "wfc_0_const";
00372                     break;
00373                 case 1:
00374                     fileName = "wfc_0_ramp";
00375                     break;
00376                 case 2:
00377                     fileName = "wfc_ev";
00378                     vortexLocation = (int*) calloc(xDim*yDim,sizeof(int));
00379                     num_vortices[0] = Tracker::findVortex(vortexLocation,
wfc, 1e-4, xDim, x, i);
00380                     if(i==0){
00381                         vortCoords = (struct Tracker::Vortex*) malloc(sizeof(struct
Tracker::Vortex)*(2*num_vortices[0]));
00382                         vortCoordsP = (struct Tracker::Vortex*) malloc(sizeof(struct
Tracker::Vortex)*(2*num_vortices[0]));
00383                         Tracker::vortPos(vortexLocation, vortCoords,
xDim, wfc);
00384                         central_vortex = Tracker::vortCentre(vortCoords, num_vortices[0]
, xDim);
00385                         vort_angle = Tracker::vortAngle(vortCoords,central_vortex,
num_vortices[0]);
00386                         appendData(&params,"Vort_angle",vort_angle);
00387                         optLatSetup(central_vortex, V, vortCoords, num_vortices[0], vort_angle
+ PI*angle_sweep/180.0, laser_power*HBAR*sqrt(omegaX*
omegaY), V_opt, x, y);
00388                         sepAvg = Tracker::vortSepAvg(vortCoords,central_vortex,
num_vortices[0]);
00389                         if(kick_it == 2){
00390                             printf("Kicked it 1\n");
00391                             cudaMemcpy(V_gpu, EV_opt, sizeof(cufftDoubleComplex)*

```

```

        xDim*yDim, cudaMemcpyHostToDevice);
00392     }
00393     FileIO::writeOutDouble(buffer, "V_opt_1",
V_opt, xDim*yDim, 0);
00394     FileIO::writeOut(buffer, "EV_opt_1",
EV_opt, xDim*yDim, 0);
00395     appendData(&params, "Central_vort_x", (double)central_vortex.
coords.x);
00396     appendData(&params, "Central_vort_y", (double)central_vortex.
coords.y);
00397     appendData(&params, "Central_vort_winding", (double)central_vortex.
wind);
00398     appendData(&params, "Central_vort_sign", (double)central_vortex.
sign);
00399     appendData(&params, "Num_vort", (double)num_vortices[0]);
00400     FileIO::writeOutParam(buffer, params, "Params.dat");
00401     }
00402     else if(num_vortices[0] > num_vortices[1]){
00403         printf("Number of vortices changed from %d to %d\n", num_vortices[1], num_vortices[0]
);
00404         Tracker::vortPos(vortexLocation, vortCoords,
xDim, wfc);
00405     }
00406     else{
00407         Tracker::vortPos(vortexLocation, vortCoords,
xDim, wfc);
00408         Tracker::vortArrange(vortCoords, vortCoordsP, num_vortices[0]);
00409     }
00410     /* num_latt_max = Tracker::findOLMaxima(olMaxLocation, V_opt, 1e-4, xDim, x);
00411     if(num_latt_max == num_vortices[0]){
00412         olCoords = (int2*) malloc(sizeof(int2)*num_latt_max);
00413         Tracker::olPos(olMaxLocation, olCoords, xDim);
00414         vortOLSigma = Tracker::sigVOL(vortCoords, olCoords, x, num_latt_max);
00415         FileIO::writeOutInt2(buffer, "opt_max_arr", olCoords, num_latt_max, i);
00416         free(olCoords);
00417     }*/
00418     FileIO::writeOutVortex(buffer, "vort_arr", vortCoords,
num_vortices[0], i);
00419     printf("Located %d vortices\n", num_vortices[0]);
00420     printf("Sigma=%e\n", vortOLSigma);
00421     free(vortexLocation);
00422     num_vortices[1] = num_vortices[0];
00423     memcpy(vortCoordsP, vortCoords, sizeof(int2)*num_vortices[0]);
00424     break;
00425     case 3:
00426         fileName = "wfc_ev_ramp";
00427         break;
00428     default:
00429         break;
00430     }
00431     if(write_it)
00432         FileIO::writeOut(buffer, fileName, wfc,
xDim*yDim, i);
00433     //printf("Energy[t@%d]=%E\n", i, energy_angmom(gpuPositionOp, gpuMomentumOp, dx, dy,
gpuWfc, gstate));
00434     /* cudaMemcpy(V_gpu, V, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
00435     cudaMemcpy(K_gpu, K, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
00436     cudaMemcpy(V_gpu, , sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
00437     cudaMemcpy(K_gpu, K, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
00438     */
00439     if(i % ((int)t_kick+1) == 0 && num_kick<=6 && gstate==1 && kick_it == 1 ){
00440         cudaMemcpy(V_gpu, EV_opt, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
00441         ++num_kick;
00442     }
00443     /*
00444     * U_r(dt/2)*wfc
00445     */
00446     if(nonlin == 1){
00447         cMultDensity<<<grid, threads>>>(gpuPositionOp, gpuWfc, gpuWfc, 0.5*Dt,
mass, omegaZ, gstate, N*interaction);
00448     }
00449     else {
00450         cMult<<<grid, threads>>>(gpuPositionOp, gpuWfc, gpuWfc);
00451     }
00452     /*
00453     * U_p(dt)*fft2(wfc)
00454     */
00455     result = cufftExecZ2Z(plan_2d, gpuWfc, gpuWfc, CUFFT_FORWARD);
00456     scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_2d, gpuWfc); //Normalise
00457     cMult<<<grid, threads>>>(gpuMomentumOp, gpuWfc, gpuWfc);
00458     result = cufftExecZ2Z(plan_2d, gpuWfc, gpuWfc, CUFFT_INVERSE);
00459     scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_2d, gpuWfc); //Normalise
00460     */
00461     /*
00462     */
00463     result = cufftExecZ2Z(plan_2d, gpuWfc, gpuWfc, CUFFT_FORWARD);
00464     scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_2d, gpuWfc); //Normalise
00465     cMult<<<grid, threads>>>(gpuMomentumOp, gpuWfc, gpuWfc);
00466     result = cufftExecZ2Z(plan_2d, gpuWfc, gpuWfc, CUFFT_INVERSE);
00467     scalarDiv<<<grid, threads>>>(gpuWfc, renorm_factor_2d, gpuWfc); //Normalise
00468     */
00469     /*

```

```

00470         * U_r(dt/2)*wfc
00471         */
00472         if(nonlin == 1){
00473             cMultDensity<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc,Dt*0.5,
mass,omegaZ,gstate,N*interaction);
00474         }
00475         else {
00476             cMult<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc);
00477         }
00478         if( (i % (int)(t_kick+1)) == 0 && num_kick<=6 && gstate==1) || (kick_it >= 1 &&
i==0) ){
00479             cudaMemcpy(V_gpu, EV, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
00480             printf("Got here: Cuda memcpy EV into GPU\n");
00481         }
00482         /*****
00483         /* Angular momentum xPy-yPx */
00484         if(lz == 1){
00485             switch(i%2 | (gstate<1)){
00486                 case 0: //Groundstate solver, even step
00487                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
00488                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00489                     angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpuldxPy, gpuWfc);
00490                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_INVERSE);
00491                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00492
00493                     result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
00494                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
00495                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
00496                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00497                     angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpuldyPx, gpuWfc);
00498                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
00499                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00500                     result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
00501                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
00502                     break;
00503
00504                 case 1: //Groundstate solver, odd step
00505                     result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
00506                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
00507                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
00508                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00509                     angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpuldyPx, gpuWfc);
00510                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
00511                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00512                     result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
00513                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
00514
00515                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
00516                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00517                     angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpuldxPy, gpuWfc);
00518                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_INVERSE);
00519                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00520                     break;
00521
00522                 case 2: //Real time evolution, even step
00523                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
00524                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00525                     cMult<<<grid,threads>>>(gpuWfc, (cufftDoubleComplex*) gpuldxPy, gpuWfc);
00526                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_INVERSE);
00527                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00528
00529                     result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
00530                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
00531                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
00532                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00533                     cMult<<<grid,threads>>>(gpuWfc, (cufftDoubleComplex*) gpuldyPx, gpuWfc);
00534                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
00535                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00536                     result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
00537                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
00538                     break;
00539
00540                 case 3: //Real time evolution, odd step
00541                     result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
00542                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
00543                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
00544                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00545                     cMult<<<grid,threads>>>(gpuWfc, (cufftDoubleComplex*) gpuldyPx, gpuWfc);
00546                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
00547                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00548                     result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
00549                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
00550
00551                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
00552                     scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00553                     cMult<<<grid,threads>>>(gpuWfc, (cufftDoubleComplex*) gpuldxPy, gpuWfc);
00554                     result = cufftExecZ2Z(plan_ld,gpuWfc,gpuWfc,CUFFT_INVERSE);

```

```

00555         scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_ld,gpuWfc);
00556         break;
00557     }
00558 }
00559 }
00560 /*
00561 */
00562     if(gstate==0){
00563         parSum(gpuWfc, gpuParSum, xDim, yDim, threads);
00564     }
00565 }
00566 return 0;
00567 }
00568
00569 /*
00570 * Used to perform parallel summation on WFC and normalise
00571 */
00572 void parSum(double2* gpuWfc, double2* gpuParSum, int xDim, int yDim, int
threads){
00573     int grid_tmp = xDim*yDim;
00574     int block = grid_tmp/threads;
00575     int thread_tmp = threads;
00576     int pass = 0;
00577     while((double)grid_tmp/threads > 1.0){
00578         if(grid_tmp == xDim*yDim){
00579             multipass<<<block,threads,threads*sizeof(double2)>>>(&gpuWfc[0],&gpuParSum[0],pass);
00580         }
00581         else{
00582             multipass<<<block,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0],pass
);
00583         }
00584         grid_tmp /= threads;
00585         block = (int) ceil((double)grid_tmp/threads);
00586         pass++;
00587     }
00588     thread_tmp = grid_tmp;
00589     multipass<<<1,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0], pass);
00590     scalarDiv_wfcNorm<<<grid,threads>>>(gpuWfc, dx*dy, gpuParSum, gpuWfc);
00591 }
00592
00593 void optLatSetup(struct Tracker::Vortex centre, double*
V, struct Tracker::Vortex *vArray, int num_vortices, double theta_opt, double intensity,
double* v_opt, double *x, double *y){
00594     int i,j;
00595     double sepMin = Tracker::vortSepAvg(vArray,centre,num_vortices);
00596     sepMin = sepMin*(1 + sepMinEpsilon);
00597     appendData(&params,"Vort_sep",(double)sepMin);
00598     /*
00599     * Defining the necessary k vectors for the optical lattice
00600     */
00601     double k_mag = ((2*PI/(sepMin*dx))/2)*(2/sqrt(3)); // Additional /2 as a result of lambda/2
period
00602     double2* k = (double2*) malloc(sizeof(double2)*3);
00603     appendData(&params,"k_mag",(double)k_mag);
00604     k[0].x = k_mag * cos(0*PI/3 + theta_opt);
00605     k[0].y = k_mag * sin(0*PI/3 + theta_opt);
00606     k[1].x = k_mag * cos(2*PI/3 + theta_opt);
00607     k[1].y = k_mag * sin(2*PI/3 + theta_opt);
00608     k[2].x = k_mag * cos(4*PI/3 + theta_opt);
00609     k[2].y = k_mag * sin(4*PI/3 + theta_opt);
00610
00611     double2 *r_opt = (double2*) malloc(sizeof(double2)*xDim);
00612
00613     /* for (int ii = 0; ii < xDim; ++ii){
00614         r_opt[ii].x = 0.0 + (xDim/sepMin)*PI*(ii-centre.coords.x)/(xDim-1);
00615         r_opt[ii].y = 0.0 + (xDim/sepMin)*PI*(ii-centre.coords.y)/(yDim-1);
00616     }
00617     */
00618     FileIO::writeOut(buffer,"r_opt",r_opt,xDim,0);
00619     appendData(&params,"k[0].x",(double)k[0].x);
00620     appendData(&params,"k[0].y",(double)k[0].y);
00621     appendData(&params,"k[1].x",(double)k[1].x);
00622     appendData(&params,"k[1].y",(double)k[1].y);
00623     appendData(&params,"k[2].x",(double)k[2].x);
00624     appendData(&params,"k[2].y",(double)k[2].y);
00625
00626     double x_shift = dx*(9+(0.5*xDim-1) - centre.coords.x);//sin(theta_opt)*(sepMin);
00627     double y_shift = dy*(0+(0.5*yDim-1) - centre.coords.y);//cos(theta_opt)*(sepMin);
00628
00629     printf("Xs=%e\nYs=%e\n",x_shift,y_shift);
00630
00631     //pragma omp parallel for private(j)
00632     for ( j=0; j<yDim; ++j ){
00633         for ( i=0; i<xDim; ++i ){
00634             v_opt[j*xDim + i] = intensity*(
00635                 pow( abs( cos( k[0].x*( x[i] + x_shift ) + k[0].y*( y[j] + y_shift ) ) ), 2)
00636                 + pow( abs( cos( k[1].x*( x[i] + x_shift ) + k[1].y*( y[j] + y_shift ) ) ), 2)
00637             );
00638         }
00639     }

```



```

00640          + pow( abs( cos( k[2].x*( x[i] + x_shift ) + k[2].y*( y[j] + y_shift ) ) ), 2)
00641          /*          pow( abs( cos( k[0].x*( r_opt[i].x + x_shift ) + k[0].y*( r_opt[j].y +
y_shift ) ) ), 2)
00642          + pow( abs( cos( k[1].x*( r_opt[i].x + x_shift ) + k[1].y*( r_opt[j].y + y_shift )
) ), 2)
00643          + pow( abs( cos( k[2].x*( r_opt[i].x + x_shift ) + k[2].y*( r_opt[j].y + y_shift )
) ), 2)
00644          );
00645          EV_opt[(j*xDim + i)].x=cos( -(V[(j*xDim + i)] + v_opt[j*xDim +
i])*(dt/(2*HBAR)));
00646          EV_opt[(j*xDim + i)].y=sin( -(V[(j*xDim + i)] + v_opt[j*xDim + i])*(
dt/(2*HBAR)));
00647      }
00648  }
00649
00650 }
00651
00655 double energy_angmom(double *Energy, double* Energy_gpu, double2 *V_op,
double2 *K_op, double dx, double dy, double2 *gpuWfc, int gState){
00656     double renorm_factor_2d=1.0/pow(xDim*yDim,0.5);
00657     double result=0;
00658
00659     for (int i=0; i < xDim*yDim; ++i){
00660         Energy[i] = 0.0;
00661     }
00662
00663
00664 /*  cudaMalloc((void**) &energy_gpu, sizeof(double2) * xDim*yDim);
00665
00666     energyCalc<<<grid,threads>>>( gpuWfc, V_op, 0.5*dt, energy_gpu, gState,1,i 0.5*sqrt(omegaZ/mass));
00667     result = cuFFTExec2Z2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_FORWARD );
00668     scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise
00669
00670     energyCalc<<<grid,threads>>>( gpuWfc, K_op, dt, energy_gpu, gState,0, 0.5*sqrt(omegaZ/mass));
00671     result = cuFFTExec2Z2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_INVERSE );
00672     scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise
00673
00674     err=cudaMemcpy(energy, energy_gpu, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);
00675
00676     for(int i=0; i<xDim*yDim; i++){
00677         result += energy[i].x;
00678         //printf("En=%E\n",result*dx*dy);
00679     }
00680     return result*dx*dy;
00681 */
00682 }
00683
00684
00685
00686 //
00687 //
00688 //
00689 /*
00690 * Used to perform parallel summation using templates from c++
00691 */
00692 template<typename T> void parSum(T *gpuToSumArr, T *gpuParSum, int xDim, int
yDim, int threads){
00693     int grid_tmp = xDim*yDim;
00694     int block = grid_tmp/threads;
00695     int thread_tmp = threads;
00696     int pass = 0;
00697     while((double)grid_tmp/threads > 1.0){
00698         if(grid_tmp == xDim*yDim){
00699             multipass<<<block,threads,threads*sizeof(T)>>>(&gpuToSumArr[0],&gpuParSum[0
],pass);
00700         }
00701         else{
00702             multipass<<<block,thread_tmp,thread_tmp*sizeof(T)>>>(&gpuParSum[0],&
gpuParSum[0],pass);
00703         }
00704         grid_tmp /= threads;
00705         block = (int) ceil((double)grid_tmp/threads);
00706         pass++;
00707     }
00708     thread_tmp = grid_tmp;
00709     multipass<<<1,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0], pass);
00710     scalarDiv_wfcNorm<<<grid,threads>>>(gpuToSumArr, dx*dy, gpuParSum, gpuToSumArr);
00711 }
00712 //
00713 //
00714 int parseArgs(int argc, char** argv){
00715     int opt;
00716     while ((opt = getopt (argc, argv, "d:x:y:w:G:g:e:T:t:n:p:r:o:L:l:s:i:P:X:Y:O:k:W:U:V:S:")) != -1) {

```

```

00717     switch (opt)
00718     {
00719         case 'x':
00720             xDim = atoi(optarg);
00721             printf("Argument for x is given as %d\n", xDim);
00722             appendData(&params, "xDim", (double) xDim);
00723             break;
00724         case 'y':
00725             yDim = atoi(optarg);
00726             printf("Argument for y is given as %d\n", yDim);
00727             appendData(&params, "yDim", (double) yDim);
00728             break;
00729         case 'w':
00730             omega = atof(optarg);
00731             printf("Argument for OmegaRotate is given as %E\n", omega);
00732             appendData(&params, "omega", omega);
00733             break;
00734         case 'G':
00735             gammaY = atof(optarg);
00736             printf("Argument for gamma is given as %E\n", gammaY);
00737             appendData(&params, "gammaY", gammaY);
00738             break;
00739         case 'g':
00740             gsteps = atof(optarg);
00741             printf("Argument for Groundsteps is given as %ld\n", gsteps);
00742             appendData(&params, "gsteps", gsteps);
00743             break;
00744         case 'e':
00745             esteps = atof(optarg);
00746             printf("Argument for EvSteps is given as %ld\n", esteps);
00747             appendData(&params, "esteps", esteps);
00748             break;
00749         case 'T':
00750             gdt = atof(optarg);
00751             printf("Argument for groundstate Timestep is given as %E\n", gdt);
00752             appendData(&params, "gdt", gdt);
00753             break;
00754         case 't':
00755             dt = atof(optarg);
00756             printf("Argument for Timestep is given as %E\n", dt);
00757             appendData(&params, "dt", dt);
00758             break;
00759         case 'd':
00760             device = atoi(optarg);
00761             printf("Argument for device is given as %d\n", device);
00762             appendData(&params, "device", device);
00763             break;
00764         case 'n':
00765             atoms = atof(optarg);
00766             printf("Argument for atoms is given as %ld\n", atoms);
00767             appendData(&params, "atoms", atoms);
00768             break;
00769         case 'r':
00770             read_wfc = atoi(optarg);
00771             printf("Argument for ReadIn is given as %d\n", read_wfc);
00772             appendData(&params, "read_wfc", (double) read_wfc);
00773             break;
00774         case 'p':
00775             print = atoi(optarg);
00776             printf("Argument for Printout is given as %d\n", print);
00777             appendData(&params, "print_out", (double) print);
00778             break;
00779         case 'L':
00780             l = atof(optarg);
00781             printf("Vortex winding is given as : %E\n", l);
00782             appendData(&params, "winding", l);
00783             break;
00784         case 'l':
00785             ang_mom = atoi(optarg);
00786             printf("Angular Momentum mode engaged: %d\n", ang_mom);
00787             appendData(&params, "corotating", (double) ang_mom);
00788             break;
00789         case 's':
00790             gpe = atoi(optarg);
00791             printf("Non-linear mode engaged: %d\n", gpe);
00792             appendData(&params, "gpe", gpe);
00793             break;
00794         case 'o':
00795             omegaZ = atof(optarg);
00796             printf("Argument for OmegaZ is given as %E\n", omegaZ);
00797             appendData(&params, "omegaZ", omegaZ);
00798             break;
00799         case 'i':
00800             interaction = atof(optarg);
00801             printf("Argument for interaction scaling is %E\n", interaction);
00802             appendData(&params, "int_scaling", interaction);
00803             break;

```

```

00804     case 'P':
00805         laser_power = atof(optarg);
00806         printf("Argument for laser power is %E\n", laser_power);
00807         appendData(&params, "laser_power", laser_power);
00808         break;
00809     case 'X':
00810         omegaX = atof(optarg);
00811         printf("Argument for omegaX is %E\n", omegaX);
00812         appendData(&params, "omegaX", omegaX);
00813         break;
00814     case 'Y':
00815         omegaY = atof(optarg);
00816         printf("Argument for omegaY is %E\n", omegaY);
00817         appendData(&params, "omegaY", omegaY);
00818         break;
00819     case 'O':
00820         angle_sweep = atof(optarg);
00821         printf("Argument for angle_sweep is %E\n", angle_sweep);
00822         appendData(&params, "angle_sweep", angle_sweep);
00823         break;
00824     case 'k':
00825         kick_it = atoi(optarg);
00826         printf("Argument for kick_it is %i\n", kick_it);
00827         appendData(&params, "kick_it", kick_it);
00828         break;
00829     case 'W':
00830         write_it = atoi(optarg);
00831         printf("Argument for write_it is %i\n", write_it);
00832         appendData(&params, "write_it", write_it);
00833         break;
00834     case 'U':
00835         x0_shift = atof(optarg);
00836         printf("Argument for x0_shift is %lf\n", x0_shift);
00837         appendData(&params, "x0_shift", x0_shift);
00838         break;
00839     case 'V':
00840         y0_shift = atof(optarg);
00841         printf("Argument for y0_shift is %lf\n", y0_shift);
00842         appendData(&params, "y0_shift", y0_shift);
00843         break;
00844     case 'S':
00845         sepMinEpsilon = atof(optarg);
00846         printf("Argument for sepMinEpsilon is %lf\n", sepMinEpsilon);
00847         appendData(&params, "sepMinEpsilon", sepMinEpsilon);
00848         break;
00849     case '?':
00850         if (optopt == 'c') {
00851             fprintf(stderr, "Option -%c requires an argument.\n", optopt);
00852         } else if (isprint(optopt)) {
00853             fprintf(stderr, "Unknown option '-%c'.\n", optopt);
00854         } else {
00855             fprintf(stderr, "Unknown option character '\\x%x'.\n", optopt);
00856         }
00857         return -1;
00858     default:
00859         abort ();
00860 }
00861 }
00862 return 0;
00863 }
00864
00865 void delta_define(double *x, double *y, double x0, double y0, double *delta){
00866     for (unsigned int i=0; i<xDim; ++i){
00867         for (unsigned int j=0; j<yDim; ++j){
00868             delta[j*xDim + i] = 1e6*HBAR*exp( -( pow( x[i] - x0, 2) + pow( y[j] - y0, 2) )/(5*
00869                 dx*dx) );
00870             EV_opt[(j*xDim + i)].x=cos( -(V[(j*xDim + i)] + delta[j*xDim +
00871                 i])*(dt/(2*HBAR)));
00872             EV_opt[(j*xDim + i)].y=sin( -(V[(j*xDim + i)] + delta[j*xDim +
00873                 i])*(dt/(2*HBAR)));
00874         }
00875     }
00876 }
00877
00878 int main(int argc, char **argv){
00879     time_t start, fin;
00880     time(&start);
00881     printf("Start: %s\n", ctime(&start));
00882     initArr(&params, 32);
00883     //appendData(&params, ctime(&start), 0.0);
00884     parseArgs(argc, argv);
00885     cudaSetDevice(device);
00886     //*****
00887     /*
00888     * Initialise the Params data structure to track params and variables

```

```

00888      */
00889      //*****//
00890      //paramS = (Params *) malloc(sizeof(Params));
00891      //strcpy(paramS->data,"INIT");
00892      //paramS->next=NULL;
00893
00894      initialise(omegaX,omegaY,atoms);
00895      timeTotal = 0.0;
00896      //*****//
00897      /*
00898      * Groundstate finder section
00899      */
00900      //*****//
00901      FileIO::writeOutParam(buffer, params, "Params.dat");
00902      if(read_wfc == 1){
00903          printf("Loading wavefunction...\n");
00904          wfc=FileIO::readIn("wfc_load","wfci_load",xDim,
yDim);
00905          printf("Wavefunction loaded.\n");
00906      }
00907
00908      double2 ph;
00909      double x_0,y_0;
00910      x_0 = 0; //(0.5*xDim)*dx;
00911      y_0 = 0; //(0.5*yDim)*dy;
00912      /* for(int i=0; i < xDim; i++){
00913          for(int j=0; j < yDim; j++){
00914              ph.x = cos( fmod( 0*atan2( y[j] - y_0, x[i] - x_0 ), 2*PI ) );
00915              ph.y = -sin( fmod( 0*atan2( y[j] - y_0, x[i] - x_0 ), 2*PI ) );
00916              wfc[(i*yDim + j)] = Minions::complexMult( wfc[(i*yDim + j)], ph );
00917          }
00918      }
00919      printf("l=%e\n",l);
00920      /* if(gsteps > 0){
00921          err=cudaMemcpy(K_gpu, GK, sizeof(cufftDoubleComplex)*xDim*
yDim, cudaMemcpyHostToDevice);
00922          if(err!=cudaSuccess)
00923              exit(1);
00924          err=cudaMemcpy(V_gpu, GV, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
00925          if(err!=cudaSuccess)
00926              exit(1);
00927          err=cudaMemcpy(xPy_gpu, xPy, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
00928          if(err!=cudaSuccess)
00929              exit(1);
00930          err=cudaMemcpy(yPx_gpu, yPx, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
00931          if(err!=cudaSuccess)
00932              exit(1);
00933          err=cudaMemcpy(wfc_gpu, wfc, sizeof(cufftDoubleComplex)*
xDim*yDim, cudaMemcpyHostToDevice);
00934          if(err!=cudaSuccess)
00935              exit(1);
00936
00937          evolve(wfc_gpu, K_gpu, V_gpu, yPx_gpu,
xPy_gpu, par_sum, xDim*yDim, gsteps, 128, 0, ang_mom,
gpe, print, atoms, 0);
00938          cudaMemcpy(wfc, wfc_gpu, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost
);
00939      }
00940
00941      free(GV); free(GK); free(xPy); free(yPx);
00942
00943      //*****//
00944      /*
00945      * Evolution
00946      */
00947      //*****//
00948      if(esteps > 0){
00949          err=cudaMemcpy(xPy_gpu, ExPy, sizeof(cufftDoubleComplex)*
xDim*yDim, cudaMemcpyHostToDevice);
00950          if(err!=cudaSuccess)
00951              exit(1);
00952          err=cudaMemcpy(yPx_gpu, EyPx, sizeof(cufftDoubleComplex)*
xDim*yDim, cudaMemcpyHostToDevice);
00953          if(err!=cudaSuccess)
00954              exit(1);
00955          err=cudaMemcpy(xPy_gpu, ExPy, sizeof(cufftDoubleComplex)*
xDim*yDim, cudaMemcpyHostToDevice);
00956          if(err!=cudaSuccess)
00957              exit(1);
00958          err=cudaMemcpy(yPx_gpu, EyPx, sizeof(cufftDoubleComplex)*
xDim*yDim, cudaMemcpyHostToDevice);
00959          if(err!=cudaSuccess)
00960              exit(1);
00961          err=cudaMemcpy(K_gpu, EK, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
00962          if(err!=cudaSuccess)

```

```

00963         exit(1);
00964         err=cudaMemcpy(V_gpu, EV, sizeof(cufftDoubleComplex)*xDim*yDim,
        cudaMemcpyHostToDevice);
00965         if(err!=cudaSuccess)
00966             exit(1);
00967         err=cudaMemcpy(wfc_gpu, wfc, sizeof(cufftDoubleComplex)*
        xDim*yDim, cudaMemcpyHostToDevice);
00968         if(err!=cudaSuccess)
00969             exit(1);
00970
00971         //delta_define(x, y, (523.6667 - 512 + x0_shift)*dx, (512.6667 - 512 + y0_shift)*dy, V_opt);
00972         FileIO::writeOutDouble(buffer, "V_opt", V_opt,
        xDim*yDim, 0);
00973         evolve(wfc_gpu, K_gpu, V_gpu, yPx_gpu,
        xPy_gpu, par_sum, xDim*yDim, esteps, 128, 1, ang_mom,
        gpe, print, atoms, 0);
00974     }
00975     free(EV); free(EK); free(ExPy); free(EyPx);
00976     free(x); free(y);
00977     cudaFree(wfc_gpu); cudaFree(K_gpu); cudaFree(V_gpu); cudaFree(
        yPx_gpu); cudaFree(xPy_gpu); cudaFree(par_sum);
00979     time(&fin);
00980     //appendData(&params, ctime(&fin), 0.0);
00981     printf("Finish: %s\n", ctime(&fin));
00982     printf("Total time: %ld seconds\n ", (long)fin-start);
00983     //appendData(&params, "t_duration", fin-start);
00984     return 0;
00985 }
00986

```

## 6.51 src/srt.cc File Reference

### Functions

- double [sepAvg](#) (int2 \*vArray, int2 centre, int length)

#### 6.51.1 Function Documentation

##### 6.51.1.1 double sepAvg ( int2 \* vArray, int2 centre, int length )

Definition at line 34 of file [srt.cc](#).

References [vis::i](#), and [result](#).

Referenced by [evolve\(\)](#).

```

00034         {
00035         double result=0.0; // = sqrt( pow(centre.x - v_array[0].x,2) + pow(centre.y - v_array[0].y,2));
00036         for (int i=0; i<length; ++i){
00037             result += sqrt( pow(centre.x - v_array[i].x,2) + pow(centre.y - v_array[i].y,2));
00038         }
00039         return result/length;
00040     }

```

Here is the caller graph for this function:

## 6.52 srt.cc

```

00001 /** srt.cc - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.

```

```

00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034 double sepAvg(int2 *vArray, int2 centre, int length){
00035     double result=0.0; // = sqrt( pow(centre.x - v_array[0].x,2) + pow(centre.y - v_array[0].y,2));
00036     for (int i=0; i<length; ++i){
00037         result += sqrt( pow(centre.x - v_array[i].x,2) + pow(centre.y - v_array[i].y,2));
00038     }
00039     return result/length;
00040 }

```

## 6.53 src/tracker.cc File Reference

```

#include "../include/tracker.h"
#include "../include/fileIO.h"
#include "../include/minions.h"
#include "../include/constants.h"

```

Include dependency graph for tracker.cc:

### Namespaces

- [Tracker](#)

*See the source file for info on functions.*

### Functions

- int [Tracker::findOLMaxima](#) (int \*marker, double \*V, double radius, int xDim, double \*x)  
*Finds the maxima of the optical lattice.*
- int [Tracker::findVortex](#) (int \*marker, double2 \*wfc, double radius, int xDim, double \*x, int timestep)  
*Phase winding method to determine vortex positions.*
- void [Tracker::olPos](#) (int \*marker, int2 \*olLocation, int xDim)  
*Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.*
- int [Tracker::phaseTest](#) (int2 vLoc, double2 \*wfc, int xDim)  
*Tests the phase winding of the wavefunction, looking for vortices.*
- double [Tracker::sigVOL](#) (struct [Tracker::Vortex](#) \*vArr, int2 \*opLatt, double \*x, int numVort)  
*Sigma of vortex lattice and optical lattice.*
- double [Tracker::vortAngle](#) (struct [Tracker::Vortex](#) \*vortCoords, struct [Vortex](#) central, int numVort)  
*Determines the angle of the vortex lattice relative to the x-axis.*
- void [Tracker::vortArrange](#) (struct [Tracker::Vortex](#) \*vCoordsC, struct [Vortex](#) \*vCoordsP, int length)  
*Ensures the vortices are tracked and arranged in the right order based on minimum distance between previous and current positions.*
- struct [Vortex](#) [Tracker::vortCentre](#) (struct [Tracker::Vortex](#) \*cArray, int length, int xDim)

*Determines the coords of the vortex closest to the central position.*

- void `Tracker::vortPos` (int \*marker, struct `Tracker::Vortex` \*vLocation, int xDim, double2 \*wfc)

*Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.*

- double `Tracker::vortSepAvg` (struct `Vortex` \*vArray, struct `Tracker::Vortex` centre, int length)

*Determines the vortex separation at the centre of the lattice.*

## Variables

- char `Tracker::bufferT` [1024]

## 6.54 tracker.cc

```
00001 /** tracker.cc - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034 #include "../include/tracker.h"
00035 #include "../include/fileIO.h"
00036 #include "../include/minions.h"
00037 #include "../include/constants.h"
00038
00042 namespace Tracker {
00043     char bufferT[1024];
00044
00048     double vortSepAvg(struct Vortex *vArray, struct
Tracker::Vortex centre, int length){
00049         double result=0.0; // = sqrt( pow(centre.x - v_array[0].x,2) + pow(centre.y -
v_array[0].y,2));
00050         double min = 0.0;
00051         double min_tmp = 0.0;
00052         int index=0;
00053         min = sqrt( pow(centre.coords.x - vArray[0].coords.x,2) + pow(centre.
coords.y - vArray[0].coords.y,2));
00054         for (int j=1; j<length; ++j){
00055             min_tmp = sqrt( pow(centre.coords.x - vArray[j].coords.x,2) + pow(centre.
coords.y - vArray[j].coords.y,2));
00056             if(min > min_tmp && min_tmp > 1e-7){
00057                 min = min_tmp;
00058                 index = j;
00059             }
00060         }
00061         return min;
00062     }
00063
00067     int findOLMaxima(int *marker, double *Vopt, double radius, int
xDim, double* x){
00068         double gridValues[9];
```

```

00069     int2 mIndex[1024];
00070     int2 index;
00071     int i,j,found;
00072     found=0;
00073     for (i=1; i<xDim-1; ++i ){
00074         for(j=1; j<xDim-1;++j){
00075             if(sqrt(x[i]*x[i] + x[j]*x[j]) < radius){
00076                 gridValues[0] = Vopt[(i-1)*xDim + (j-1)];
00077                 gridValues[1] = Vopt[(i-1)*xDim + j];
00078                 gridValues[2] = Vopt[(i-1)*xDim + (j+1)];
00079                 gridValues[3] = Vopt[i*xDim + (j-1)];
00080                 gridValues[4] = Vopt[i*xDim + j];
00081                 gridValues[5] = Vopt[i*xDim + (j+1)];
00082                 gridValues[6] = Vopt[(i+1)*xDim + (j-1)];
00083                 gridValues[7] = Vopt[(i+1)*xDim + j];
00084                 gridValues[8] = Vopt[(i+1)*xDim + (j+1)];
00085                 if(fabs((gridValues[4]-Minions::maxValue(gridValues,9))/gridValues[4])
<= 1e-7){
00086                     //printf ("%d,%d\n",i,j);
00087                     (marker)[i*xDim + j] = 1;
00088                     index.x=i;
00089                     index.y=j;
00090                     mIndex[found] = index;
00091                     ++found;
00092                 }
00093             }
00094         }
00095     }
00096     return found;
00097 }
00098
00099 #ifdef VORT_MIN
00100 int findVortex(int *marker, double2* wfc, double radius, int
xDim, double* x, int timestep){
00101     double gridValues[9];
00102     int2 vIndex[1024];
00103     int2 index;
00104     int i,j,found;
00105     found=0;
00106     // #pragma omp parallel for private(j)
00107     for (i=1; i<xDim-1; ++i ){
00108         for(j=1; j<xDim-1;++j){
00109             if(sqrt(x[i]*x[i] + x[j]*x[j]) < radius){
00110                 gridValues[0] = Minions::psi2(wfc[(i-1)*xDim + (j-1)]);
00111                 gridValues[1] = Minions::psi2(wfc[(i-1)*xDim + j]);
00112                 gridValues[2] = Minions::psi2(wfc[(i-1)*xDim + (j+1)]);
00113                 gridValues[3] = Minions::psi2(wfc[(i)*xDim + (j-1)]);
00114                 gridValues[4] = Minions::psi2(wfc[(i)*xDim + j]);
00115                 gridValues[5] = Minions::psi2(wfc[(i)*xDim + (j+1)]);
00116                 gridValues[6] = Minions::psi2(wfc[(i+1)*xDim + (j-1)]);
00117                 gridValues[7] = Minions::psi2(wfc[(i+1)*xDim + j]);
00118                 gridValues[8] = Minions::psi2(wfc[(i+1)*xDim + (j+1)]);
00119                 if(fabs((gridValues[4]-Minions::minValue(gridValues,9))/gridValues[4])
< 1e-7){
00120                     //printf ("%d,%d\n",i,j);
00121                     (marker)[i*xDim + j] = 1;
00122                     index.x=i;
00123                     index.y=j;
00124                     vIndex[found] = index;
00125                     found++;
00126                 }
00127             }
00128         }
00129     }
00130     return found;
00131 }
00132 #else
00133
00136 int findVortex(int *marker, double2* wfc, double radius, int xDim, double *x, int timestep){
00137     double2 *g = (double2*) malloc(sizeof(double2)*4);
00138     double *phiDelta = (double*) malloc(sizeof(double)*4);
00139     int i,j,found;
00140     int cond_x, cond_y;
00141     cond_x = 0; cond_y = 0;
00142     found = 0;
00143     long rnd_value = 0;
00144     double sum = 0.0;
00145     for ( i=0; i < xDim-1; ++i ){
00146         for ( j=0; j < xDim-1; ++j ){
00147             if(sqrt(x[i]*x[i] + x[j]*x[j]) < radius){
00148                 g[0] = Minions::complexScale(
Minions::complexDiv( wfc[i*xDim + j], wfc[(i+1)*xDim + j] ), (
Minions::complexMag( wfc[(i+1)*xDim + j])
Minions::complexMag( wfc[i*xDim + j] )));
00149                 g[1] = Minions::complexScale(
Minions::complexDiv( wfc[(i+1)*xDim + j], wfc[(i+1)*xDim + (j+1)] ), (
Minions::complexMag( wfc[(i+1)*xDim + (j+1)] )

```



```

    Minions::complexMag( wfc[(i+1)*xDim + j] ));
00150     g[2] = Minions::complexScale(
    Minions::complexDiv( wfc[(i+1)*xDim + (j+1)], wfc[i*xDim + (j+1)] ), (
    Minions::complexMag( wfc[i*xDim + (j+1)] ) /
    Minions::complexMag( wfc[(i+1)*xDim + (j+1)] )););
00151     g[3] = Minions::complexScale(
    Minions::complexDiv( wfc[i*xDim + (j+1)], wfc[i*xDim + j] ), (
    Minions::complexMag( wfc[i*xDim + j] ) /
    Minions::complexMag( wfc[i*xDim + (j+1)] )););
00152
00153     for (int k=0; k<4; ++k){
00154         phiDelta[k] = atan2( g[k].y, g[k].x );
00155         if(phiDelta[k] <= -PI){
00156             phiDelta[k] += 2*PI;
00157         }
00158     }
00159     sum = phiDelta[0] + phiDelta[1] + phiDelta[2] + phiDelta[3];
00160     rnd_value = lround(sum/(2*PI));
00161     if( sum >= 1.9*PI && cond_x <= 0 && cond_y <= 0 ){
00162         marker[i*xDim + j] = rnd_value;
00163         ++found;
00164         sum = 0.0;
00165         cond_x = 2; cond_y = 2;
00166     }
00167     else if( sum <= -1.9*PI && cond_x <= 0 && cond_y <= 0 ) {
00168         marker[i*xDim + j] = -rnd_value;
00169         ++found;
00170         sum = 0.0;
00171         cond_x = 2; cond_y = 2;
00172     }
00173     }
00174     --cond_x;
00175     --cond_y;
00176     }
00177     }
00178     }
00179     return found;
00180 }
00181 #endif
00182
00183 void olPos(int *marker, int2 *olLocation, int xDim){
00184     int i,j;
00185     unsigned int counter=0;
00186     for(i=0; i<xDim; ++i){
00187         for(j=0; j<xDim; ++j){
00188             if((marker)[i*xDim + j] == 1){
00189                 (olLocation)[ counter ].x=i;
00190                 (olLocation)[ counter ].y=j;
00191                 ++counter;
00192             }
00193         }
00194     }
00195 }
00196
00197
00198
00199
00200 int phaseTest(int2 vLoc, double2* wfc, int xDim){
00201     int result = 0;
00202     double2 gridValues[4];
00203     double phiDelta[4];
00204     double sum=0.0;
00205     int i=vLoc.x, j=vLoc.y;
00206     gridValues[0] = Minions::complexScale(
    Minions::complexDiv(wfc[i*xDim + j],wfc[(i+1)*xDim + j]), (
    Minions::complexMag(wfc[(i+1)*xDim + j]) /
    Minions::complexMag(wfc[i*xDim + j])););
00210     gridValues[1] = Minions::complexScale(
    Minions::complexDiv(wfc[(i+1)*xDim + j],wfc[(i+1)*xDim + (j+1)]), (
    Minions::complexMag(wfc[(i+1)*xDim + (j+1)]) /
    Minions::complexMag(wfc[(i+1)*xDim + j])););
00211     gridValues[2] = Minions::complexScale(
    Minions::complexDiv(wfc[(i+1)*xDim + (j+1)],wfc[i*xDim + (j+1)]), (
    Minions::complexMag(wfc[i*xDim + (j+1)]) /
    Minions::complexMag(wfc[(i+1)*xDim + (j+1)])););
00212     gridValues[3] = Minions::complexScale(
    Minions::complexDiv(wfc[i*xDim + (j+1)],wfc[i*xDim + j]), (
    Minions::complexMag(wfc[i*xDim + j]) /
    Minions::complexMag(wfc[i*xDim + (j+1)])););
00213
00214     for (int k=0; k<4; ++k){
00215         phiDelta[k] = atan2(gridValues[k].y,gridValues[k].x);
00216         if(phiDelta[k] <= -PI){
00217             phiDelta[k] += 2*PI;
00218         }
00219     }
00220     sum = phiDelta[0] + phiDelta[1] + phiDelta[2] + phiDelta[3];
00221     if(sum >=1.8*PI){
00222         result = 1;
00223     }

```

```

00224         return result;
00225     }
00226
00230 void vortPos(int *marker, struct Tracker::Vortex *vLocation, int xDim, double2 *
wfc){
00231     int i,j;
00232     unsigned int counter=0;
00233     for(i=0; i<xDim; ++i){
00234         for(j=0; j<xDim; ++j){
00235             if( abs((marker)[i*xDim + j]) >= 1){
00236                 (vLocation)[ counter ].coords.x=i;
00237                 (vLocation)[ counter ].coords.y=j;
00238                 (vLocation)[ counter ].sign = ( signbit(abs(marker[i*xDim + j])) == 0 ) ? 1 : -1;
00239                 (vLocation)[ counter ].wind = abs(marker[i*xDim + j]);
00240                 ++counter;
00241             }
00242         }
00243     }
00244 }
00245
00249 void vortArrange(struct Tracker::Vortex *vCoordsC, struct
Vortex *vCoordsP, int length){
00250     int dist, dist_t;
00251     int i, j, index;
00252     for ( i = 0; i < length; ++i ){
00253         dist = 0x7FFFFFFF; //arbitrary big value
00254         index = i;
00255         for ( j = i; j < length ; ++j){
00256             dist_t = ( (vCoordsP[i].coords.x - vCoordsC[j].coords.x)*(vCoordsP[i].coords.x
- vCoordsC[j].coords.x) + (vCoordsP[i].coords.y - vCoordsC[j].
coords.y)*(vCoordsP[i].coords.y - vCoordsC[j].coords.y) );
00257             if(dist > dist_t ){
00258                 dist = dist_t;
00259                 index = j;
00260             }
00261         }
00262         Minions::coordSwap(vCoordsC,index,i);
00263     }
00264 }
00265
00269 struct Vortex vortCentre(struct Tracker::Vortex *cArray, int length, int xDim){
00270     int i, j, counter=0;
00271     int valX, valY;
00272     double valueTest, value = 0.0;
00273     valX = (cArray)[0].coords.x - ((xDim/2)-1);
00274     valY = (cArray)[0].coords.y - ((xDim/2)-1);
00275     value = sqrt( valX*valX + valY*valY );//Calcs the sqrt(x^2+y^2) from central position. try to
minimise this value
00276     for ( i=1; i<length; ++i ){
00277         valX = (cArray)[i].coords.x - ((xDim/2)-1);
00278         valY = (cArray)[i].coords.y - ((xDim/2)-1);
00279         valueTest = sqrt(valX*valX + valY*valY);
00280         if(value > valueTest){
00281             value = valueTest;
00282             counter = i;
00283         }
00284     }
00285     return (cArray)[counter];
00286 }
00287
00291 double vortAngle(struct Tracker::Vortex *vortCoords, struct
Vortex central, int numVort){
00292     int location;
00293     double sign=1.0;
00294     double minVal=1e300;//(pow(central.x - vortCoords[0].x,2) + pow(central.y - vortCoords[0].y,2));
00295     for (int i=0; i < numVort; ++i){
00296         if (minVal > (pow(central.coords.x - vortCoords[i].coords.x,2) + pow(central.
coords.y - vortCoords[i].coords.y,2)) && abs(central.coords.x - vortCoords[i].
coords.x) > 2e-6 && abs(central.coords.y - vortCoords[i].coords.y) > 2e-6){
00297             minVal = (pow(central.coords.x - vortCoords[i].coords.x,2) + pow(central.
coords.y - vortCoords[i].coords.y,2));
00298             location = i;
00299         }
00300     }
00301     double ang=(fmod(atan2( (vortCoords[location].coords.y - central.coords.y), (vortCoords[
location].coords.x - central.coords.x) ),PI/3));
00302     printf("Angle=%e\n",ang);
00303     return PI/3 - ang;
00304
00305     //return PI/2 + fmod(atan2(vortCoords[location].y-central.y, vortCoords[location].x - central.x),
PI/3);
00306     //return PI/2 - sign*acos( ( (central.x - vortCoords[location].x)*(central.x -
vortCoords[location].x) ) / ( minVal*(central.x - vortCoords[location].x) ) );
00307 }
00308
00312 double sigVOL(struct Tracker::Vortex *vArr, int2 *opLatt, double *x, int numVort){
00313     double sigma = 0.0;

```

```

00314         double dx = abs(x[1]-x[0]);
00315         for (int i=0; i<numVort; ++i){
00316             sigma += pow( abs( sqrt( (vArr[i].coords.x - opLatt[i].x)*(vArr[i].
coords.x - opLatt[i].x) + (vArr[i].coords.y - opLatt[i].y)*(vArr[i].
coords.y - opLatt[i].y) ) * dx), 2);
00317         }
00318         sigma /= numVort;
00319         return sigma;
00320     }
00321 }
00322 }

```

## 6.55 src/wavefunction.cu File Reference

### Classes

- class [BEC2D::Wavefunction](#)

### Namespaces

- [BEC2D](#)

### Functions

- int2 [getGridSize](#) (int xDim, int yDim)
- double2 & [getWfc](#) ()
- double2 [initWfc](#) ()
- bool [setGridSize](#) (int xDim, int yDim)
- class [BEC2D::Wavefunction](#) [BEC2D::Wavefunction](#) ()
- [Wavefunction](#) ()
- [Wavefunction](#) (int xDim, int yDim, double xMax, double yMax)

### Variables

- double2 [dimMax](#)
- int2 [gridSize](#)
- double2 \* [wfc](#) = new double2[xDim\*yDim]

### 6.55.1 Function Documentation

6.55.1.1 int2 [Wavefunction::getGridSize](#) ( int xDim, int yDim )

6.55.1.2 double2& [Wavefunction::getWfc](#) ( )

6.55.1.3 double2 [Wavefunction::initWfc](#) ( )

6.55.1.4 bool [Wavefunction::setGridSize](#) ( int xDim, int yDim )

6.55.1.5 [Wavefunction::Wavefunction](#) ( )

Definition at line 54 of file [wavefunction.cu](#).

```

00054         {
00055
00056     }

```

6.55.1.6 `Wavefunction::Wavefunction ( int xDim, int yDim, double xMax, double yMax )`

## 6.55.2 Variable Documentation

6.55.2.1 `double2 dimMax`

Definition at line 315 of file [wavefunction.cu](#).

6.55.2.2 `int2 gridSize`

Definition at line 314 of file [wavefunction.cu](#).

6.55.2.3 `double2* wfc = new double2[xDim*yDim]`

Definition at line 316 of file [wavefunction.cu](#).

## 6.56 `wavefunction.cu`

```

00001 /** wavefunction.cu - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034
00035 /*
00036 Unused. Future extension.
00037 */
00038 namespace BEC2D{
00039     class Wavefunction{
00040     private:
00041         int2 gridSize;
00042         double2 dimMax;
00043         double2 *wfc = new double2[xDim*yDim];
00044
00045     public:
00046         Wavefunction();
00047         Wavefunction(int xDim, int yDim, double xMax, double
00048 yMax);
00049         bool setGridSize(int xDim, int yDim);
00050         int2 getGridSize(int xDim, int yDim);
00051         double2 initWfc();
00052         double2 &getWfc();
00053     }
00054     Wavefunction::Wavefunction() {

```

```
00055
00056     }
00057     Wavefunction::Wavefunction(int xDim, int yDim, double
00058         xMax, double yMax){
00059     }
00060     Wavefunction::setGridSize(int xDim, int yDim){
00061     }
00062 }
00063 }
00064
00065
```