1 Introduction

Libgrid provides routines for efficiently accessing and manipulating 1-D, 2-D, and 3-D Cartesian real and complex grids on Linux-based systems without the need of considering the underlying specialized hardware (e.g., OpenMP, CUDA). In addition to the basic grid operations such as grid allocation, release, Fourier transform, grid arithmetics, etc., it has also specialized routines for propagating time-dependent Schrödinger equation in real or imaginary time. The latter routines are required by libdft library, which solves various types of non-linear Schrödinger equations that describe superfluid ^{4}He and Bose-Einstein condensates. Libgrid was written by Lauri Lehtovaara, David Mateo, and Jussi Eloranta, and is freely distributed according to GNU GENERAL PUBLIC LICENSE Version 3 (see doc/GPL.txt).

2 Installation

Installation of libgrid requires the following packages:

- git (a free and open source distributed version control system)
- GNU C compiler with OpenMP support (gcc)
- FFTW 3.x (Fast Fourier Transform package)
- LAPACK (Linear Algebra PACKage; optional)
- BLAS (Basic Linear Algebra Subprograms; optional)

To install these packages on Fedora linux, use (# implies execution with root privileges):

```
# dnf install git gcc fftw-* lapack lapack-devel blas blas-devel
```

If the system has NVIDIA GPUs, libgrid can use CUDA to accelerate the grid operations. For Fedora-based systems, the propriatary NVIDIA drivers and the CUDA libraries can be obtained from: `https://negativo17.org/nvidia-driver/`. This installation is compatible with the default settings in make.conf as described below. Currently, libgrid does not support OpenCL.

To copy the current version of libgrid to subdirectory libgrid, issue (% implies execution with normal user privileges):

```
% git clone https://github.com/jmeloranta/libgrid.git
```

Change to libgrid directory and review the configuration settings:

```
% cd libgrid
% more make.conf
```

The options specified in this file determine how libgrid will be compiled. OpenMP support is included automatically with COMPILER = gcc.

Option	**Description**
COMPILER	gcc (normal use) or gcc-debug (non-parallel debugging version)
REAL	Real floating point precision (float, double or quad precision). Note that for libdft use, single precision floats have limited use.
INT	Integer number size (int or long)
ROOT	Root installation directory (default /usr)
CUDA	Set to "yes" if CUDA is installed and "no" if not. The default is to auto-detect.
CUDA_DEV	Which CUDA device number to use by default (0 = first GPU). Note: GPU device number can be changed in the user code.
CUDAINC	Include file directory for CUDA C header files (default /usr/include/cuda).
CUDALIB	Directory containing the CUDA libraries (default /usr/lib64).
CUDA_HOSTCC	GCC version that is compatible with the current installation of CUDA (default /usr/bin/cuda-gcc).
CUDA_CC	NVIDIA GPU architecture (e.g., sm_50; default auto-detect).
CUDA_FASTMATH	"yes" implies NVCC options: –ftz=true –prec-div=false –prec-sqrt=false –fmad=true and "no" implies options: –ftz=false –prec-div=true –prec-sqrt=true –fmad=true (default no).
CUDA_TPB	Number of CUDA threads per block (GPU architecture dependent; default 6).
CUDA_THRADJ	CUDA Thread adjustment for 2-D operations (i.e., Crank -Nicolson). GPU architecture dependent value, 2 - 3 (default 3).
CUDA_DEBUG	Whether to include debug code in libggrid CUDA routines ("yes" or "no"; default no).
USE_LAPACK	Some routines in libgrid use LAPACK (not needed for libdft). Use "yes" to include them or "no" to leave them out.
AR	Archive program (default ar).

To compile the library, change to src subdirectory and issue make:

```
% cd src
% make -j
```

Provided that the compilation completed without errors, install the library (as root):

```
# make install
```

3 Accessing the library routines

To access libgrid functions in C program, the following header files should be included:

```
#include <grid/grid.h>
#include <grid/au.h>
```

where the first include file is required and the second optional include makes the following conversion factors available:

GRID_AUTOANG	Factor to convert from Bohr to Ånström (10^{-10} m).
GRID_AUTOM	Factor to convert from Bohr to meter.
GRID_AUTOK	Factor to convert from Hartee to Kelvin (energy; kT).
GRID_AUTOCM1	Factor to convert from Hartree to wavenumber.
GRID_HZTOCM1	Factor to convert from Hz to wavenumber.
GRID_AUTOAMU	Atomic unit mass (electron mass) to atomic mass unit (AMU).
GRID_AUTOFS	Atomic unit time to femtosecond.
GRID_AUTOS	Atomic unit time to second.
GRID_AUTOBAR	Atomic pressure unit (Hartree/Bohr^2) to bar.
GRID_AUTOPA	Atomic pressure unit to Pa (Pascal).
GRID_AUTOATM	Atomic pressure unit to atm.
GRID_AUTOMPS	Atomic velocity unit to m/s.
GRID_AUTON	Atomic force unit to Newton (N).
GRID_AUTOVPM	Atomic electric field strength to V/m.
GRID_AUTOPAS	Atomic viscosity unit to Pa s.
GRID_AUKB	Boltzmann constant in a.u. (k_B)

Note that wherever applicable, librid will use the atomic unit system and the above conversion factors are provided for converting to other systems. To convert from atomic unit to another unit, multiply by the predefined factor above or, divide by it in order to convert the other way around.

To compile and link a program using libgrid, it is most convenient to construct a makefile (note that the $(CC) line has TAB as the first character):

```
include /usr/include/grid/make.conf

test: test.o
    $(CC) $(CFLAGS) -o test test.o $(LDFLAGS)

test.o: test.c
```

This will compile the program specified in test.c and link the appropriate libraries automatically. Both CFLAGS and LDFLAGS are obtained automatically from libgrid's make.conf.

4 Data types

The libgrid header file defines the real and integer data types automatically according to those requested during the configuration step (make.conf). Instead of using float, double, int, long etc. directly, use REAL to represent a floating point number and INT to introduce the variables (e.g., REAL x, y; INT i;). To define complex number type, use REAL complex. Since the function names in the system math library vary depending on the floating point precision (e.g., expf(), exp(), expl()), libgrid defines the corresponding function in upper case such that is assigned to the function of the requested precision (e.g., EXP()). So, in order to call the exponential function, use, e.g., y = EXP(x);. Another place where the size of the REAL and INT data types are required is the scanf/printf standard I/O library routines. To print a REAL number with printf, use, for example, printf("value = " FMT_R "\n");. Or to print an INT use FMT_I.

Libgrid has built-in data types for 1-D, 2-D, and 3-D Cartesian REAL ("rgrid") and REAL complex ("cgrid") grids. To allocate such grids in a C program, first introduce them as pointers, e.g., rgrid *abc or cgrid *abc. Then use either rgrid_alloc() or cgrid_alloc() functions to allocate the space. For example:

```
cgrid *abc;
abc = cgrid_alloc(32, 32, 32, 1.0, CGRID3D_PERIODIC_BOUNDARY, NULL);
...
cgrid_free(abc);
```

would allocate a periodic 3-D grid with dimensions 32x32x32 and spatial grid step length of 1.0 (for more information on the arguments to cgrid_alloc() in Section 5). To get a 1-D grid, include it as the last dimension, e.g., 1x1x32, or for a 2-D grid use the two last dimensions, 1x32x32. It is important to follow this convention for performance reasons. In the code, coordinate x corresponds to the first index, y to the second, and z to the third. By default, the origin is placed to the center of the grid.

The basic properties of the above rgrid and cgrid data types are explained below.

4.1 Real grid data type (rgrid)

Data type rgrid corresponds to a structure with the following members:

Member	Description
REAL *value	Array holding the real grid point data.
char id[32]	String describing the grid (comment).
size_t grid_len	Number of bytes allocated for value array.
INT nx	Number of grid points along the 1st grid index (x).
INT ny	Number of grid points along the 2nd grid index (y).
INT nz	Number of grid points along the 3rd grid index (z).
INT nz2	This is equal to $2 \times (nz/2 + 1)$. Used for indexing *value.
REAL step	Grid step length (equal in all directions).
REAL x0	Grid origin x_0 (default 0.0).
REAL y0	Grid origin y_0 (default 0.0).
REAL z0	Grid origin z_0 (default 0.0).
REAL kx0	Grid origin $k_{x,0}$ in the reciprocal (momentum) space (default 0.0).
REAL ky0	Grid origin $k_{y,0}$ in the reciprocal (momentum) space (default 0.0).

REAL kz0	Grid origin $k_{z,0}$ in the reciprocal (momentum) space (default 0.0).
REAL (*value_outside) (rgrid *, INT, INT, INT)	Pointer to function returning values outside the grid.
void *outside_params_ptr	Pointer for passing additional data to value_outside function.
REAL default_outside_params	Default REAL value for outside_params_ptr.
fftwX_plan plan	FFTW plan for forward FFT transforming the grid (X = f, empty, l).
fftwX_plan iplan	FFTW plan for inverse FFT transforming the grid (X = f, empty, l).
cufftHandle cufft_handle_r2c	CUFFT handle for forward FFT transform (CUDA).
cufftHandle cufft_handle_c2r	CUFFT handle for inverse FFT transform (CUDA).
REAL fft_norm	Normalization factor for FFT (without including the step length).
REAL fft_norm2	Normalization factor for FFT (including the step length).

Note that rgrid is the same as struct rgrid_struct type. To reference the value at index (i, j, k) in grid->value, use:

```
grid->value[(i * grid->ny + j) * grid->nz2 + k]
```

The same indexing applies to 1-D (i = j = 0) and 2-D grids with (i = 0). This does not account for the grid boundary condition. Library function rgrid_value_at_index(rgrid *grid, INT i, INT j, INT k) can be used to retrieve the value subject to the chosen boundary condition automatically. To set value for a grid point at (i, j, k), library function rgrid_value_to_index(rgrid *grid, INT i, INT j, INT k, REAL value) can be used. This function does not consider the boundary condition assigned to the grid. Note that a different indexing scheme must be used if the grid is in the reciprocal (Fourier) space:

```
grid->value[(i * grid->ny + j) * (grid->nz / 2 + 1) + k]
```

Alternatively, function rgrid_cvalue_at_index(rgrid *grid, INT i, INT j, INT k) can be used.

4.2 Complex grid data type (cgrid)

Data type cgrid corresponds to a structure with the following members:

Member	Description
REAL complex *value	Array holding the complex grid point data.
char id[32]	String describing the grid (comment).
size_t grid_len;	Number of bytes allocated for the value array.
INT nx	Number of grid points along the 1st grid index (x).
INT ny	Number of grid points along the 2nd grid index (y).
INT nz	Number of grid points along the 3rd grid index (z).
REAL step	Grid step length (equal in all directions).
REAL x0	Grid origin x_0 (default 0.0).
REAL y0	Grid origin y_0 (default 0.0).
REAL z0	Grid origin z_0 (default 0.0).
REAL kx0	Grid origin $k_{x,0}$ in the reciprocal (momentum) space (default 0.0).
REAL ky0	Grid origin $k_{y,0}$ in the reciprocal (momentum) space (default 0.0).
REAL kz0	Grid origin $k_{z,0}$ in the reciprocal (momentum) space (default 0.0).
REAL omega	Rotation frequency around z-axis (rotating flow).
REAL complex (*value_outside)	Pointer to function returning values outside
(cgrid *grid, INT, INT, INT)	the grid.
void *outside_params_ptr	Pointer for passing additional data to value_outside function.
REAL complex default_outside_params	Default REAL complex value for outside_params_ptr.

fftwX_plan plan	FFTW plan for forward FFT transforming the grid (X = f, empty, l).
fftwX_plan iplan	FFTW plan for inverse FFT transforming the grid (X = f, empty, l).
fftwX_plan implan	Like plan but for non-periodic boundaries (untested).
fftwX_plan iimplan	Like iplan but for non-periodic boundaries (untested).
cufftHandle cufft_handle	CUFFT handle for FFT transforms using CUDA.
REAL fft_norm	Normalization factor for FFT (without including the step length).
REAL fft_norm2	Normalization factor for FFT (including the step length).

Indexing of the grid values follow the same convention as for rgrid data type.

4.3 Wave function data type (wf)

Wave functions are special objects that contain a complex grid (wave function values) and all other necessary parameters such that it can be propagated in time by using a non-linear Schrödinger equation. This data type is mostly used in libdft library.

Member	Description
cgrid *grid	Complex grid containing the wave function values.
REAL mass	Particle mass that is represented by this wave function.
REAL norm	Requested normalization of the wave function.
char boundary	Boundary condition to be used for time propagation. WF_DIRICHLET_BOUNDARY = Dirichlet boundary condition, WF_NEUMANN_BOUNDARY = Neumann boundary condition, WF_PERIODIC_BOUNDARY = Periodic boundary condition.
char propagator	Time propagator. WF_2ND_ORDER_PROPAGATOR = 2nd order in time, WF_4TH_ORDER_PROPAGATOR = 4th order in time.

5 Library functions

Functions in libgrid are divided into following classes: 1) rgrid_* (real valued grids; rgrid), 2) cgrid_* (complex valued grids; cgrid), and 3) grid_wf_* (grids describing a wavefunction; wf). These are described in the subsections below. The arguments are listed in the tables in the same order as they are passed to the corresponding functions.

5.1 Real grid routines

5.1.1 rgrid_alloc() – Allocate a real-valued grid

This function allocates memory for a real-valued grid. It takes the following arguments:

Argument	Description
INT nx	Number of points on the grid along x (1st index).
INT ny	Number of points on the grid along y (2nd index).
INT nz	Number of points on the grid along z (3rd index).
REAL step	Spatial step length for the grid.
REAL *(value_outside)()	Function for accessing boundary points: RGRID_DIRICHLET_BOUNDARY = Dirichlet boundary, RGRID_NEUMANN_BOUNDARY = Neumann boundary, RGRID_PERIODIC_BOUNDARY = Periodic boundary. This can also be a user supplied function.
void *outside_params_ptr	Pointer for passing parameters for the given boundary access function. Use NULL to with the predefined boundary functions.
char *id	String ID describing the grid.

The return value is a pointer to the allocated grid (rgrid *) or NULL on allocation error. *Note that the grid is kept in padded form, which can be directly used for in-place FFT.*

5.1.2 rgrid_set_origin() – Set grid origin

This function sets the origin (i.e., coordinates for the center of the grid). t takes the following arguments:

Argument	Description
rgrid *grid	Grid for which the origin is to be defined.
REAL x0	x coordinate for the origin.
REAL y0	y coordinate for the origin.
REAL z0	z coordinate for the origin.

This function does not return any value. The mapping between grid indices and coordinates is given by:

```
x(i)  = (i - nx / 2) * step - x0
y(j)  = (j - ny / 2) * step - y0
z(k)  = (k - nz / 2) * step - z0
```

for a grid with dimensions nx×ny×nz, spatial step length step, and origin set at (x0,y0,z0).

5.1.3 rgrid_shift_origin() – Shift grid origin

This function shifts the current origin by given offsets. It takes the following arguments:

Argument	Description
rgrid *grid	Grid for which the origin is to be defined.
REAL x0	Shift in x coordinate (i.e., add x0 to origin x).
REAL y0	Shift in y coordinate (i.e., add y0 to origin y).
REAL z0	Shift in z coordinate (i.e., add z0 to origin z).

This function does not return any value.

5.1.4 rgrid_set_momentum() – Set grid origin in reciprocal space

This function sets the grid origin in momentum space (or the velocity of the frame of reference). It takes the following arguments:

Argument	Description
rgrid *grid	Grid for which the momentum origin is to be defined.
REAL kx0	Momentum origin along the x axis.
REAL ky0	Momentum origin along the y axis.
REAL kz0	Momentum origin along the z axis.

Here kx0, ky0 and kz0 can be any real numbers but keep in mind that the grid will only contain the component $k = 0$ if they are multiples of:

```
kx0min = 2 * M_PI / (NX * STEP)
ky0min = 2 * M_PI / (NY * STEP)
kz0min = 2 * M_PI / (NZ * STEP)
```

where (NX, NY, NZ) specify the grid dimensions and STEP is the grid spatial step length. In terms of velocities, this means that velocities must be multiples of:

```
kx0min = 2 * M_PI * HBAR / (NX * STEP * MASS)
ky0min = 2 * M_PI * HBAR / (NY * STEP * MASS)
kz0min = 2 * M_PI * HBAR / (NZ * STEP * MASS)
```

where MASS is the particle mass, and HBAR is $\hbar$. This function does not return any value.

5.1.5 rgrid_free() – Free real-valued grid

This function takes a pointer to the grid (rgrid *) to be freed as the argument. It has not return value. Note that all allocated memory is automatically released when the program terminates.

5.1.6 rgrid_write() – Write grid on disk in binary format

This function will write the contents of a given grid to the disk. It takes the following arguments:

Argument	Description
rgrid *grid	Grid to be written to disk.
FILE *out	File handle for the file I/O.

This is a binary format file where the data is stored in the following order:

```
nx
ny
nz
step
(nx * ny * nz2) number of grid point values
```

where the grid dimensions are (nx,ny,nz) and nz2 = 2 * (nz / 2 + 1). Note that nz2 must be used as we store the grid in padded format. This function does not return any value.

5.1.7 rgrid_read() – Read grid from disk in binary format

This function will read grid data from disk. It takes the following arguments:

Argument	Description
rgrid *grid	Grid where the data is to be read. If NULL, a grid with the correct dimensions will be allocated. Note that the boundary condition will assigned to periodic by default.
FILE *in	File handle for reading the file I/O.

This function returns a pointer to the grid or NULL pointer on error.

5.1.8 rgrid_copy() – Copy grid

Copy a grid to another where the source and destination grids are determined by the arguments:

Argument	Description
rgrid *copy	Destination grid.
rgrid *grid	Source grid.

This function has no return value.

5.1.9 rgrid_shift() – Shift grid

This function shifts a grid spatially by a given amount. It takes the following arguments:

Argument	Description
rgrid *shifted	Destination grid for the operation.
rgrid *grid	Source grid for the operation.
REAL xs	Shift grid spatially by this amount along x.
REAL ys	Shift grid spatially by this amount along y.
REAL zs	Shift grid spatially by this amount along z.

The new positions for the grid are:

```
x(new) = x - xs
y(new) = y - ys
z(new) = z - zs
```

So, the the point (xs, ys, zs) moves to the center of the grid. This function does not return any value.

5.1.10 rgrid_zero() - Zero grid values

This function sets all values in a given grid to zero. The only argument it takes is a pointer (rgrid *) to the grid to be zeroed. It returns no value.

5.1.11 rgrid_constant – Set grid to constant value

This function is similar to rgrid_zero() but it allows for an additional argument to specify the value that will be written to the grid. The arguments are as follows:

Argument	Description
rgrid *grid	Grid to be set to a constant value.
REAL c	Constant value.

This function does not return any value.

5.1.12 rgrid_product_func() – Multiply grid by function

This function multiples a given grid by the values returned by a user specified function. It takes the following arguments:

Argument	Description
rgrid *grid	Destination grid for the operation.
REAL (*func)()	Function providing the multiplication values. See below for the arguments to this function.
void *farg	Pointer to user specified data that is passed to (*func).

Function REAL (*func)() takes the following arguments:

Argument	Description

void *arg	Pointer to user data (provided by *farg above).
REAL val	Value at the current grid point.
REAL x	Current x coordinate.
REAL y	Current y coordinate.
REAL z	Current z coordinate.

rgrid_product_func() function has no return value.

5.1.13 rgrid_map – Map function on grid

This function maps a user specified function onto a given grid. It takes the following arguments: Function REAL (*func)() takes the following arguments:

Argument	Description
rgrid *grid	Destination grid for the operation.
REAL (*func)()	User specified function providing the mapping. For description of the arguments, see below.
void *farg	Pointer to user specified data for (*func)().

User specified function REAL (*func)() takes the following arguments:

Argument	Description
void *arg	Pointer to user data (provided by *farg above).
REAL x	Current x coordinate.
REAL y	Current y coordinate.
REAL z	Current z coordinate.

rgrid_map function has no return value.

5.1.14 rgrid_smooth_map – Smooth map function onto grid

This function is like rgrid_map() but performs linear smoothing on the function. This can be used to weight the values at grid points to produce more accurate integration over the grid. This function takes the following arguments:

Argument	Description
rgrid *grid	Destination grid for the operation.

REAL (*func)()	Function providing the mapping. The arguments are same as for rgrid_map() function.
void *farg	Pointer to user specified data.
INT ns	Number of intermediate points to be used in smoothing.

This function has no return value.

5.1.15 rgrid_adaptive_map() – Adaptive smooth map function onto grid

This function is like rgrid_smooth_map() but limits for intermediate steps and requested tolerance can be specified. It takes the following arguments:

Argument	Description
rgrid *grid	Destination grid for the operation.
REAL (*func)()	Function providing the mapping. It takes the same arguments as rgrid_map().
void *farg	Pointer to user specified data for (*func)().
INT min_ns	Minimum number of intermediate points to be used in smoothing.
INT max_ns	Maximum number of intermediate points to be used in smoothing.
REAL tol	Tolerance for the converge of integral over the function (REAL; input).

This function has no return value.

5.1.16 rgrid_sum() – Add two grids

This function adds two grids and stores the output in a third grid ("gridc = grida + gridb"). It takes the grid pointers as arguments:

Argument	Description
rgrid *gridc	Destination grid.
rgrid *grida	1st grid for the summation.
rgrid *gridb	2nd grid for the summation.

This function has no return value.

5.1.17 rgrid_difference() – Subtract two grids

This function subtracts two grids and stores the output in a third grid ("gridc = grida - gridb"). It takes the grid pointers as arguments :

Argument	Description
rgrid *gridc	Destination grid.
rgrid *grida	1st grid for the difference.
rgrid *gridb	2nd grid for the difference.

This function has no return value.

5.1.18 rgrid_product() – Product of two grids

This function takes a product of two grids and stores the output in a third grid ("gridc = grida * gridb"). It takes the grid pointers as arguments:

Argument	Description
rgrid *gridc	Destination grid.
rgrid *grida	1st grid for the product.
rgrid *gridb	2nd grid for the product.

This function has no return value.

5.1.19 rgrid_power() – Rise grid to power

This function rises a grid to specified power and stores the output in a third grid ("gridb = grida$^{\text{exponent}}$"). It takes the grid pointers as arguments:

Argument	Description
rgrid *gridb	Destination grid.
rgrid *grida	Source grid.
REAL exponent	Exponent for power operation.

This function has no return value. Note this function uses math library routine POW().

5.1.20 rgrid_abs_power() – Rise absolute grid to power

This function rises absolute value of a grid to specified power and stores the output in a third grid ("gridb = |grida|$^{\text{exponent}}$"). It takes the grid pointers as arguments:

Argument	Description
rgrid *gridb	Destination grid.
rgrid *grida	Source grid.
REAL exponent	Exponent for power operation.

This function has no return value. Note this function uses math library routine POW().

5.1.21 rgrid_division() – Divide two grids

This function divides two grids and stores the output in a third grid ("gridc = grida / gridb"). It takes the grid pointers as arguments:

Argument	Description
rgrid *gridc	Destination grid.
rgrid *grida	1st grid for the division.
rgrid *gridb	2nd grid for the division.

This function has no return value.

5.1.22 rgrid_division_eps() – Safe divide two grids

This function "safely" divides two grids and stores the output in a third grid ("gridc = grida / (gridb + eps)"). It takes the grid pointers as arguments:

Argument	Description
rgrid *gridc	Destination grid.
rgrid *grida	1st grid for the division.
rgrid *gridb	2nd grid for the division.
REAL eps	Small number to be added to the denominator.

This function has no return value.

5.1.23 rgrid_add() - Add constant to grid

This function adds a constant number to a grid ("grid = grid + c"). It takes the following arguments:

Argument	Description
rgrid *grid	Grid where the constant is added.
REAL c	Number to be added.

This function has no return value.

5.1.24 rgrid_multiply() – Multiply grid by constant

This function multiplied a grid by constant ("grid = grid * c"). It takes the following arguments:

Argument	Description
rgrid *grid	Grid to be multiplied.
REAL c	Number to multiply with.

This function has no return value.

5.1.25 rgrid_add_and_multiply() – Add and multiply grid

This function adds a constant to grid and multiplies it with another constant ("grid = (grid + ca) * cm"). It takes the following arguments:

Argument	Description
rgrid *grid	Grid to be multiplied.
REAL ca	Constant to be added to the grid.
REAL cm	Constant to be multiplied with the grid.

This function has no return value. Note that the adding and multiplying are done in different order as in rgrid_multiply_and_add().

5.1.26 rgrid_multiply_and_add() – Multiply and add grid

This function adds a constant to grid and multiplies it with another constant ("grid = cm * grid + ca"). It takes the following arguments:

Argument	Description

rgrid *grid	Grid to be multiplied.
REAL cm	Constant to be multiplied with the grid.
REAL ca	Constant to be added to the grid.

This function has no return value. Note that the adding and multiplying are done in different order as in rgrid_add_and_multiply().

5.1.27 rgrid_add_scaled() – Add scaled grids

This function adds two grids such that the second grid is scaled ("gridc = gridc + d * grida"). It takes the following arguments:

Argument	Description
rgrid *gridc	Source/destination grid for the operation.
REAL d	Multiplier for grida.
rgrid *grida	Source grid for the operation.

This function has no return value.

5.1.28 rgrid_add_scaled_product() – Add scaled product of grids

This function performs operation: "gridc = gridc + d * grida * gridb". It takes the following arguments:

Argument	Description
rgrid *gridc	Source/destination grid.
REAL d	Constant multiplier.
rgrid *grida	2nd source grid.
rgrid *gridb	3rd source grid.

This function has no return value.

5.1.29 rgrid_operate_one() – Operate on grid

Operate on a grid by a given operator: gridc = O(grida). It takes the following arguments:

Argument	Description
rgrid *gridc	Destination grid.

rgrid *grida	Source grid.
REAL (*operator)()	Function operating on grida. Arguments for this user specified function are given below.

The user specified operator (*operator)() maps the value at a grid point to another real number. The arguments to this function pointer are:

Argument	Description
rgrid *gridc	Destination grid.
rgrid *grida	Source grid.
REAL (*operator)()	Function operating on grida. Arguments for this user specified function are given below.
void *params	Parameters for operator (pointer).

The arguments to (*operator)() are:

Argument	Description
REAL val	Value at the current grid point.
void *params	Pointer to use supplied parameters.

rgrid_operate_one() function has no return value.

5.1.30 rgrid_operate_one_product() – Operate and multiply grids

This function operates on a grid by a given operator and then multiplies by another grid: "gridc = gridb * O(grida)". The arguments to this function are:

Argument	Description
rgrid *gridc	Destination grid.
rgrid *gridb	Multiply by this grid.
rgrid *grida	Multiply by O(grida).
REAL (*operator)()	Function operating on grida. Arguments for this user specified function are given below.

The arguments to (*operator)() are:

Argument	Description
REAL val	Value at the current grid point.
void *params	Pointer to use supplied parameters.

rgrid_operate_one() function has no return value.

5.1.31 rgrid_operate_two() – Operate on two grids

This function operates on a grid by a given operator that depends on two grids: "gridc = O(grida, gridb)". The arguments to this function are:

Argument	Description
rgrid *gridc	Destination grid.
rgrid *grida	1st source grid for operator.
rgrid *gridb	2nd source grid for operator.
REAL (*operator)()	Function operating on grida and gridb. Arguments for this user specified function are given below.

The arguments to (*operator)() are:

Argument	Description
REAL aval	Value at the current grid point (grida).
REAL bval	Value at the current grid point (gridb).

rgrid_operate_two() function has no return value.

5.1.32 rgrid_transform_one() – Transform grid

This function operates directly on a grid by a given operator: "O(grid)". The arguments to this function are:

Argument	Description
rgrid *grid	Source/destination grid.
void (*operator)()	Function operating on grid. Arguments for this user specified function are given below.

The arguments to (*operator)() are:

Argument	Description
REAL *val	Value at the current grid points (note: pointer).

rgrid_transform_one() function has no return value.

5.1.33 rgrid_transform_two() – Transform two grids

This function operates directly on two grids by a given operator: "O(grida,gridb)". The arguments to this function are:

Argument	Description
rgrid *grida	1st grid to be operated.
rgrid *gridb	2nd grid to be operated.
void *(operator)	Operator for grid and gridb. The arguments to this function are given below.

The arguments to (*operator)() are:

Argument	Description
REAL *avalue	Pointer to current grida value.
REAL *bvalue	Pointer to current gridb value.

rgrid_transform_two() function has no return value.

5.1.34 rgrid_integral() – Integrate over grid

This function integrates over the given grid. It takes the grid pointer (rgrid *) as the only argument and returns the value of the integral.

5.1.35 rgrid_integral_region() – Integrate over grid region

This function integrates over the given grid region. It takes the following arguments:

Argument	Description
rgrid *grid	Grid to be integrated.
REAL xl	Lower limit for x.
REAL xu	Upper limit for x.
REAL yl	Lower limit for y.
REAL yu	Upper limit for y.
REAL zl	Lower limit for z.
REAL zu	Upper limit for z.

This function returns the value of the integral.

5.1.36 rgrid_integral_of_square() – Integrate grid squared

This function integrates over the given grid squared ("grid^2"). It takes the grid pointer (rgrid *) as the only argument and returns the value of the integral.

5.1.37 rgrid_integral_of_product() – Overlap between grids

Calculate the overlap between two grids (i.e., integral over the product of the grids). This function takes the following arguments:

Argument	Description
rgrid *grida	1st grid for integration.
rgrid *gridb	2nd grid for integration.

This function returns the value of the overlap integral (REAL).

5.1.38 rgrid_grid_expectation_value() – Grid expectation value

Calculate the expectation value of a grid over a given probability density: integral(gridb * |grida|^2). This function takes the following arguments:

Argument	Description
rgrid *grida	1st grid for integration (|grida|^2).
rgrid *gridb	2nd grid for integration.

This function returns the value of the overlap integral (REAL).

5.1.39 rgrid_grid_expectation_value_func() – Expectation value of function over grid squared

Calculate the expectation value of a function over grid squared: int grida func() grida = int func() grida^2. This function takes the following arguments:

Argument	Description
REAL (*func)()	Function to be averaged. The arguments to this functions are given below.
rgrid *grida	Grid giving the probability density (grida^2).

The arguments to (*func)() are:

Argument	Description
void *arg	Pointer to use supplied data to the function.
REAL val	Value of grida at the current point.
REAL x	Current value of x.
REAL y	Current value of y.
REAL z	Current value of z.

This user supplied function return the function value to be averaged (REAL). Function rgrid_grid_expectation_value_func() returns the expectation value (REAL).

5.1.40 rgrid_weighted_integral() – Integral of grid over weighting function

Integrate over grid multiplied by weighting function: int grid w(x,y,z). This function takes the arguments as:

Argument	Description
rgrid *grid	Grid to be integrated over.
REAL (*weight)() = Function defining the weight. The arguments to this function are given below.	
void *farg	Use supplied argument to the weight function.

The weight function takes the following arguments:

Argument	Description
void *farg	User supplied data to (*weight)().
x	Current x coordinate.
y	Current y coordinate.
z	Current z coordinate.

Function rgrid_weighted_integral() returns the value of the integral (REAL).

5.1.41 rgrid_weighted_integral_of_square() – Integral of grid squared over weighting function

Integrate over grid squared multiplied by weighting function: int grid^2 w(x,y,z). This function takes the arguments as:

Argument	Description
rgrid *grid	Grid to be integrated over.
REAL (*weight)() = Function defining the weight. The arguments to this function are given below.	
void *farg	Use supplied argument to the weight function.

The weight function takes the following arguments:

Argument	Description
void *farg	User supplied data to (*weight)().
x	Current x coordinate.
y	Current y coordinate.
z	Current z coordinate.

Function rgrid_weighted_integral_of_square() returns the value of the integral (REAL).

5.1.42 rgrid_fd_gradient_x() – Differentiate grid with respect to x

Differentiate a grid with respect to x (central difference). The function takes the following arguments:

Argument	Description
rgrid *grid	Grid to be differentiated.
rgrid *gradient	Grid containing the derivative (output).

This function does not return any value.

5.1.43 rgrid_fd_gradient_y() – Differentiate grid with respect to y

Differentiate a grid with respect to y (central difference). The function takes the following arguments:

Argument	Description
rgrid *grid	Grid to be differentiated.
rgrid *gradient	Grid containing the derivative (output).

This function does not return any value.

5.1.44 rgrid_fd_gradient_z() – Differentiate grid with respect to z

Differentiate a grid with respect to z (central difference). The function takes the following arguments:

Argument	Description
rgrid *grid	Grid to be differentiated.
rgrid *gradient	Grid containing the derivative (output).

This function does not return any value.

5.1.45 rgrid_fd_gradient() – gradient of a grid

Calculate gradient of a grid (finite difference). The function takes the following arguments:

Argument	Description
rgrid *grid	grid to be differentiated.
rgrid *gradient_x	x component derivative grid.
rgrid *gradient_y	y component derivative grid.
rgrid *gradient_z	z component derivative grid.

This function does not return any value.

5.1.46 rgrid_fd_laplace() – Laplace of a grid

Calculate Laplacian of a grid (finite difference). The function takes the following arguments:

Argument	Description
rgrid *grid	grid to be differentiated.
rgrid *laplace	Grid containing the Laplacian.

This function does not return any value.

5.1.47 rgrid_fd_laplace_x() – 2nd derivative of a grid (x)

Calculate 2nd derivative of a grid with respect to x (finite difference). The function takes the following arguments:

Argument	Description
rgrid *grid	grid to be differentiated.
rgrid *laplacex	Grid containing the 2nd derivative.

This function does not return any value.

5.1.48 rgrid_fd_laplace_y() – 2nd derivative of a grid (y)

Calculate 2nd derivative of a grid with respect to y (finite difference). The function takes the following arguments:

Argument	Description
rgrid *grid	grid to be differentiated.
rgrid *laplacey	Grid containing the 2nd derivative.

This function does not return any value.

5.1.49 rgrid_fd_laplace_z() – 2nd derivative of a grid (z)

Calculate 2nd derivative of a grid with respect to z (finite difference). The function takes the following arguments:

Argument	Description
rgrid *grid	grid to be differentiated.
rgrid *laplacez	Grid containing the 2nd derivative.

This function does not return any value.

5.1.50 rgrid_fd_gradient_dot_gradient() – Dot product of gradient of grid with itself

This function calculates dot product of the gradient of the grid with itself: ∇ grid $\cdot$ ∇ grid (finite difference). It takes the following arguments:

Argument	Description

rgrid *grid	Source grid for the operation.
rgrid *grad_dot_grad	Destination grid.

This function does not return any value.

5.1.51 rgrid_print() – Print contents of grid in ASCII

This function print a given grid into file in ASCII format. The arguments are as follows:

Argument	Description
rgrid *grid	Grid to be printed.
FILE *out	Output file pointer.

This function does not return any value.

5.1.52 rgrid_fft() – Fast forward Fourier transform of grid

Perform forward Fourier transformation of a grid. The grid for FFT is given as the only argument (rgrid *). The grid is overwritten by the result. No normalization of the output is performed. Note that the indexing of the complex data is different from the real data (see FFTW manual). The function does not return any value.

5.1.53 rgrid_inverse_fft() – Fast inverse Fourier transform of grid

Perform inverse Fourier transformation of a grid. The grid for FFT is given as the only argument (rgrid *). The grid is overwritten by the result. No normalization of the output is performed. The function does not return any value.

5.1.54 rgrid_scaled_inverse_fft() – Scaled fast inverse Fourier transform of grid

Perform scaled inverse Fourier transformation of a grid. The grid for FFT is given as the first argument (rgrid *) followed by normalization constant (i.e., transformed data multiplied by this value). The grid is overwritten by the result. The function does not return any value.

5.1.55 rgrid_inverse_fft_norm() – Normalized fast inverse Fourier transform of grid

Perform normalized inverse Fourier transformation of a grid. The grid for FFT is given as the only argument (rgrid *). The grid is overwritten by the result. The output is normalized (i.e., doing forward FFT followed by this yields the original data). The function does not return any value.

5.1.56 rgrid_fft_convolute() – Convolution in reciprocal space

Multiply two grids that are in reciprocal space. Convolution between grida and gridb can be obtained by:

```
rgrid_fft(grida);
rgrid_fft(gridb);
rgrid_convolute(gridc, grida, gridb);
rgrid_inverse_fft(gridc);
```

The output will be in gridc. This function takes three arguments as follows:

Argument	Description
rgrid *grida	1st grid to be convoluted.
rgrid *gridb	2nd grid to be convoluted.
rgrud *gridc	Output.

This function does not return any value.

5.1.57 rgrid_multiply_fft() – Multiply grid in reciprocal space by constant

Multiply grid in reciprocal space by a constant (grid->value is complex). The arguments are:

Argument	Description
rgrid *grid	Grid to be multiplied. Note that the values are actually complex.
REAL c	Multiply the grid by this value.

This function does not return any value.

5.1.58 rgrid_value_at_index() – Access grid point at given index

This function returns grid point value at a given index (follows boundary condition). The arguments are:

Argument	Description
rgrid *grid	Grid to be accessed.
INT i	Index along x.
INT j	Index along y.
INT k	Index along z.

This function returns the grid point value (REAL). Note that in CUDA this function is very slow as it transfers every element individually.

5.1.59 rgrid_value_to_index() – Set value to grid at given index

This function puts a specified value to a given index location in a grid. It takes the following arguments:

Argument	Description
rgrid *grid	Grid to be accessed.
INT i	1st index (x).
INT j	2nd index (y).
INT k	3rd index (z).
REAL value	Value to be set at grid location (i, j, k).

This function does not return any value. Note that in CUDA this function is very slow as it transfers every element individually.

5.1.60 rgrid_cvalue_at_index() – Access grid point in reciprocal space

This function returns (complex) grid point value at a given index when the grid is in reciprocal space (i.e., it has been Fourier transformed). It takes arguments as follows:

Argument	Description
rgrid *grid	grid to be accessed. The data is complex (after FFT).
INT i	1st index (x).

INT j	2nd index (y).
INT k	3rd index (z).

The function returns (REAL complex) value at grid point (i, j, k). Note that in CUDA this function is very slow as it transfers every element individually.

5.1.61 rgrid_value() – Access grid point at (x, y, z)

This function returns grid point at given (x, y, z) point where linear interpolation is used between the grid points. The arguments are as follows:

Argument	Description
rgrid *grid	Grid to be accessed.
REAL x	x value.
REAL y	y value.
REAL z	z value.

This function returns (REAL) value at grid position (x, y, z). Note that in CUDA this function is very slow as it transfers every element individually.

5.1.62 rgrid_extrapolate() – Extrapolate between two grid sizes

This function extrapolates between grids of different sizes (both size and step) using linear interpolation. It takes the following arguments:

Argument	Description
rgrid *dest	Destination grid.
rgrid *src	Source grid.

This function does not return any value.

5.1.63 rgrid_rotate_z() – Rotate contents of grid around z-axis

Rotate contents of a grid by a given angle around the z-axis. The function takes the following arguments:

Argument	Description
rgrid *in	The original grid (to be rotated).
rgrid *out	Output grid (rotated grid).
REAL th	Rotation angle in radians.

This function does not return any value.

5.1.64 rgrid_max() – Largest value on grid

Return the largest value contained in a grid (REAL). This function takes only one argument (rgrid *grid), which specifies the grid to be searched.

5.1.65 rgrid_max() – Smallest value on grid

Return the smallest value contained in a grid (REAL). This function takes only one argument (rgrid *grid), which specifies the grid to be searched.

5.1.66 rgrid_random() – Add random noise to grid

Add random numbers to given grid within [-scale,+scale[. The arguments are:

Argument	Description
rgrid *grid	Grid where random numbers will be added.
REAL scale	Specify the scale for random numbers.

This function does not return any value.

5.1.67 rgrid_poisson() – Solve Poisson equation

This routine solves Poisson equation ($\Delta f = u$) subject to periodic boundaries. It uses finite difference for Laplacian (7 point) and FFT. It takes one grid (rgrid *) as argument, which on entry describes the right hand side function (u). On exit, this grid is replaced by the solution (f). The function does not return any value.

5.1.68 rgrid_div() – Divergence of vector field

Calculate divergence of a given vector field. The arguments are:

Argument	Description
rgrid *div	Result grid.
rgrid *fx	x component of the vector field.
rgrid *fy	y component of the vector field.
rgrid *fz	z component of the vector field.

This function does not return any value.

5.1.69 rgrid_rot() – Rot of vector field

Calculate rot of a given vector field. The arguments are:

Argument	Description
rgrid *rotx	Result grid (x component).
rgrid *roty	Result grid (y component).
rgrid *rotz	Result grid (z component).
rgrid *fx	x component of the vector field.
rgrid *fy	y component of the vector field.
rgrid *fz	z component of the vector field.

This function does not return any value.

5.1.70 rgrid_abs_rot() – Norm of rot of vector field

Calculate norm of rot of a given vector field (|rot|). The arguments are:

Argument	Description
rgrid *rot	Result grid.
rgrid *fx	x component of the vector field.
rgrid *fy	y component of the vector field.
rgrid *fz	z component of the vector field.

This function does not return any value.

5.1.71 rgrid_dealias23() – Dealias in Fourier space (2/3 rule)

This function performs de-alias of a grid in Fourier space by the 2/3 rule: zero high wavenumber components between: [n/2 - n/3, n/2 + n/3] in each direction (except the last dimension is [nz - nz/3, nz]). The only argument is the grid (rgrid *; in reciprocal space) that will be operated on. The function does not return any value.

5.1.72 rgrid_ipower() – Rise grid to integer power

Raise given grid to a specified integer power (fast – not using POW()). The following arguments are required:

Argument	Description
rgrid *dst	Destination grid.
rgrid *src	Source grid for operation.
INT exponent	Exponent to be used. This value can be negative.

This function does not return any value.

5.1.73 rgrid_threshold_clear() – Set grid value based on threshold values

Set a value to given grid points based on upper/lower limit thresholds of another grid (which may correspond to same grids). This function takes the following arguments:

Argument	Description
rgrid *dest	Destination grid.
rgrid *src	Source grid for evaluating the thresholds.
REAL ul	Upper limit threshold for the operation.
REAL ll	Lower limit threshold for the operation.
REAL uval	Value to set when the upper limit was exceeded.
REAL lval	Value to set when the lower limit was exceeded.

This function does not return any value.

5.2 Complex grid routines

5.3 Mixed real - complex grid routines

5.3.1 grid_real_to_complex_re() – Copy real grid to real part of complex grid

Copy a real grid to real part of a complex grid (imaginary part is set to zero). The following arguments apply:

Argument	Description
cgrid *dest	Destination grid.
rgrid *source	Source grid.

This function does not return any value.

5.3.2 grid_real_to_complex_im() – Copy real grid to imaginary part of complex grid

Copy a real grid to imaginary part of a complex grid (real part is set to zero). The following arguments apply:

Argument	Description
cgrid *dest	Destination grid.
rgrid *source	Source grid.

This function does not return any value.

5.3.3 grid_add_real_to_complex_re() – add real grid to real part of complex grid

Add a real grid to real part of a complex grid (imaginary part unchanged). The following arguments apply:

Argument	Description
cgrid *dest	Destination grid.
rgrid *source	Source grid.

This function does not return any value.

5.3.4 grid_add_real_to_complex_im() – add real grid to imaginary part of complex grid

Add a real grid to imaginary part of a complex grid (real part unchanged). The following arguments apply:

Argument	Description
cgrid *dest	Destination grid.
rgrid *source	Source grid.

This function does not return any value.

5.3.5 grid_product_complex_with_real() – Product of real grid with complex grid

Calculates product of a real grid with a complex grid: dest(complex) = dest(complex) * source(real). The following arguments apply:

Argument	Description
cgrid *dest	Destination grid.
rgrid *source	Source grid.

This function does not return any value.

5.4 Wave function routines

5.5 Linear algebra routines

6 External functions

7 Utilities

8 Examples

9 References