Multigrid

A highly versatile integration grid class for C++

Tobias Stollenwerk

May 3, 2012

Contents

1. Introduction

The multigrid class is a highly versatile integration grid class for `C++`. It features the integration of multiple grid regions in order to resolve sharp peaks or steps in the integrand function. Hereby one can choose between equidistant, tangential and logarithmically dense grid regions. Although the number of grid regions is not bounded, there is an index function which analytically calculates the inverse of the grid and is useful for fast interpolation purposes. Intersection and overlap of different grid region is possible will be dealt with by favouring the better resolve grid region.

The original purpose of the multigrid class was to resolve a lot of sharp Lorentz-like peaks in a self-consistent calculation on strongly correlated electron systems.

2. Quick Start Guide

The purpose of this section is to give you a brief introduction of how the multigrid is used in practice. It may suffice for the most applications.

2.1. Installation

At the moment the only possiblility to use the multigrid class is to directly incorporate the source file into your program. It is planned to create a library version. At first download the latest version of the multigrid class from

`https://github.com/tstollenw/multigrid`

unpack it and put the files

```
multigrid.cpp
multigrid.h
grid.cpp
grid.h
mesh.cpp
mesh.h
```

into your program directory. Include and link the files in the usual way. There is an example program `main.cpp` and the corresponding makefile `makefile` which will create an excecutable `example.out`.

2.2. First Steps

The numerical calculation of an integral over some function $f(\omega)$

$$I = \int d\omega f(\omega) \tag{2.1}$$

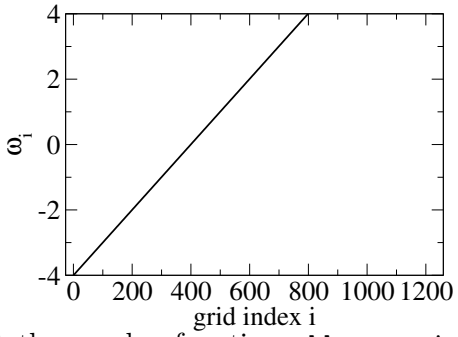
can be written as

$$I = \sum_{i=0}^{M} d\omega_i f(\omega_i) \tag{2.2}$$

where the ω_i is the discrete integration grid and $d\omega_i$ are the corresponding weights. The integration grid is a mapping from a discrete index $i \in \{0, M\}$ to a contineous variable ω_i. The multigrid class provides both the integration grid and its weights (which are calculated by the trapez rule) as well as an inverse mapping (back from any ω to the nearest index i).

In the following we will show the basic functionallity of the multigrid class by some simple examples. All these examples are more or less all part of the example program `main.cpp`. At first we will create a simple equidistant grid with from -4 to 4 with a resolution of 0.01 by

```
multigrid mgrid;
mgrid.add_gr_equi(-4, 4, 0.01);
mgrid.create();
```



After the declaration of the multigrid named `mgrid`, the member function `add_gr_equi` adds an equidistant grid region to the grid. The grid is created by invoking the `create` member function and it is accessed over its member variables (see table 2.1).

Name	Type	Description
`M`	integer	Number of grid points
`omega`	STL-Vector	Grid
`domega`	STL-Vector	Weights of the grid
`omega_min`	double	Minimum grid value (equal to `omega[0]`)
`omega_max`	double	Maximum grid value (equal to `omega[M]`)
`inverse`	returns integer	Inverse mapping: $\omega \to i$

Table 2.1.: Most important members of the multigrid

The calculation of the above integral from -4 to 4 is then done by

```
double I=0;
for (int i=0; i<=mgrid.M; j++)
{
        I += f(mgrid.omega[i]) * mgrid.domega[i];
}
```

2.3. Logarithmically Dense Grid Regions

To resolve steps or very sharp peaks in the integrand function one needs a lot of integration grid points at specific regions. The multigrid class provides a tool to solve such problems: logarithmically dense grid regions (LGR). An LGR is determined by four variables, i.e. the center of the grid region ω_0 which corresponds for example to the

position of a peak in the integrand function, the half width of the grid region ω_1, the maximal resolution $d\omega_{max}$ at the center of the grid region and the minimal resolution $d\omega_{min}$ at the edges of the grid region (see figure 2.1).

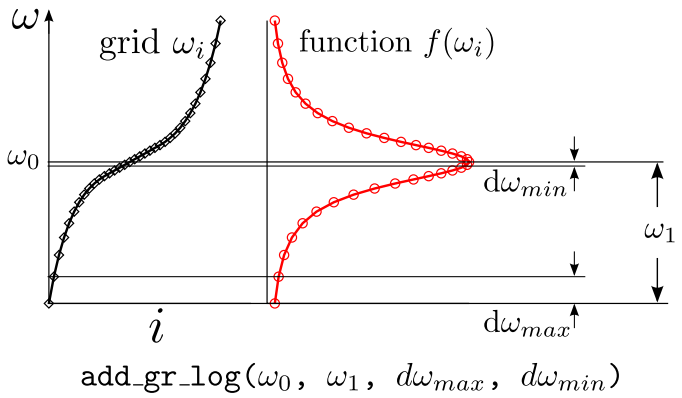
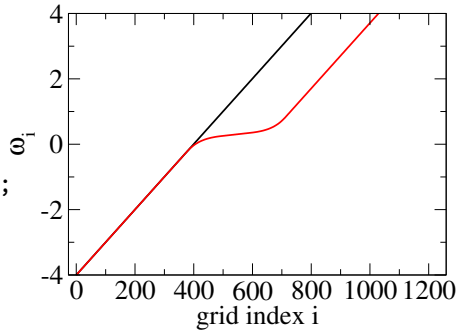


Figure 2.1.: Logarithmically dense grid region to resolve a peaked integrand function

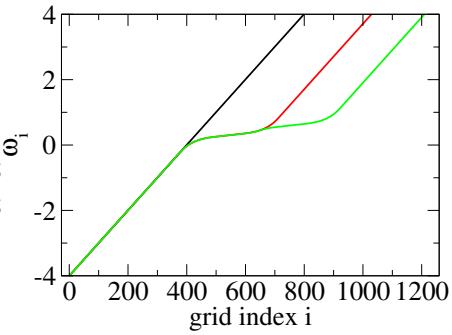
It is created by the function `add_gr_log`(ω_0, ω_1, $d\omega_{max}$, $d\omega_{min}$). For example the following code adds a LGR on top of the equidistant gridregion from above. Note that since the equidistant grid region was added first, it determines the outer boundaries of the whole grid (here from -4 to 4). The first added grid region is therefore a special one and is called the basic grid region.

```
multigrid mgrid;
mgrid.add_gr_equi(-4, 4, 0.01);
mgrid.add_gr_log(0.3,0.5,0.001,0.01);
mgrid.create();
```



The strength of the multigrid is that one can add now more and more grid regions on top of each other. The `create` function will take care of calculating intersection points between the grid regions by favouring the better resolved grid region. In the following example the there are two intersecting LGR on top of an equidistant grid region.

```
multigrid mgrid;
mgrid.add_gr_equi(-4, 4, 0.01);
mgrid.add_gr_log(0.3,0.5,0.001,0.01);
mgrid.add_gr_log(0.6,0.5,0.001,0.01);
mgrid.create();
```



These are only the basic features of the multigrid class. There is an algorithm which decides where to cut grid regions if there is intersection or even skip a particular grid region in special cases. The decisive element is the grid resolution exactly at the center of a given grid region ω_0. This is called the peak point. Hereby it is possible to add hundreds of grid regions on top of each other without losing the resolution at every single peak point. In figure 2.2 there is an example for the necessity for multiple LGR in the integration grid. The integrand function has several sharp peaks which has to be resolved. Each peak is resolved by a LGR.

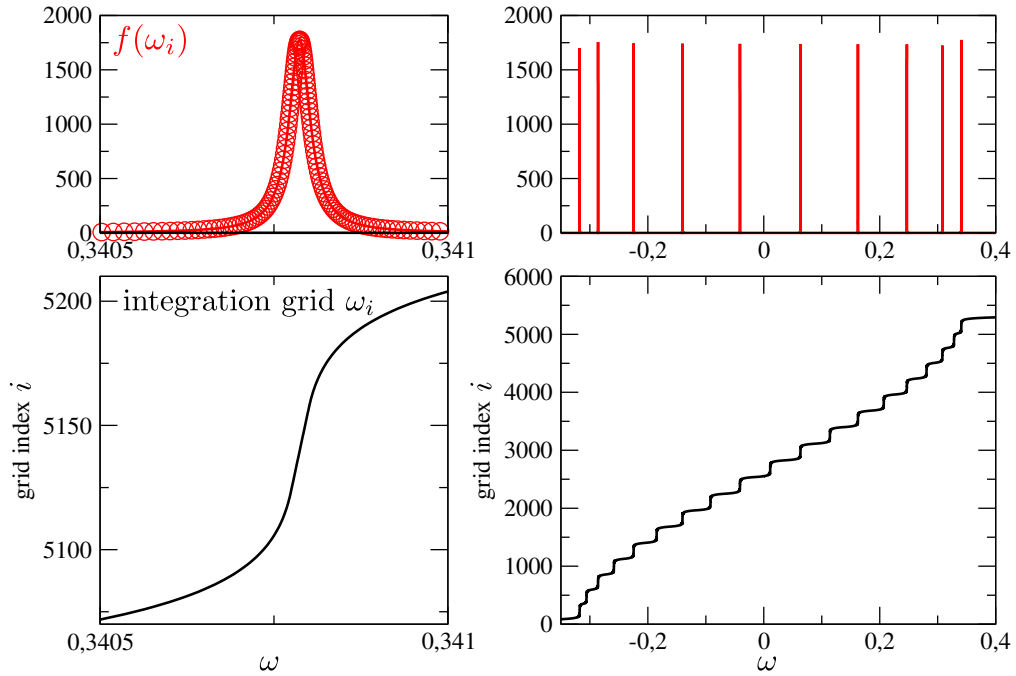


Figure 2.2.: Multigrid with various logarithmically dense grid regions to resolve a multiple peaked integrand function

3. Numerical Integration

3.1. Numerical Integration - Trapezoidal rule

In this chapter, we will briefly review the concepts of numerical integration in one dimension. Consider integral over some function $f(\omega)$

$$I = \int_a^b d\omega f(\omega) \tag{3.1}$$

For numerical calculation one has to discretize the interval of integration $[a, b]$:

$$\omega \to \omega(i) \quad ; \quad i \in \{0, \ldots, M\} \quad \text{and} \quad \omega(0) = a, \quad \omega(M) = b.$$

Where the strictly increasing mapping $\omega : \mathbb{N} \to \mathbb{R} : i \mapsto \omega(i)$ is called a grid. The numerical approximation for the integral now reads

$$I = \sum_{i=0}^{M} d\omega(i) f(\omega(i)) \tag{3.2}$$

where the $d\omega(i)$ are the so called weights. There are various methods to calculate the weights out of the grid itself. We will restict ourselfes to the trapezoidal rule which approximates the integrand function by linear regions connecting the sample points (see figure 3.1).

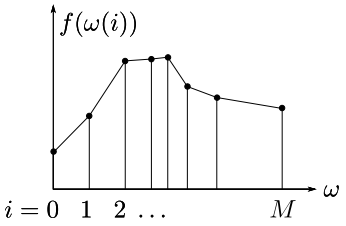


Figure 3.1.: Trapezoidal rule for a non-equidistant grid.

The numerical approximation for the integral is then

$$
\begin{aligned}
I &= \sum_{i=0}^{M-1} \frac{f(\omega(i+1)) + f(\omega(i))}{2} (\omega(i+1) - \omega(i)) \\
&= f_0 \Bigg(\underbrace{\frac{\omega(1) - \omega(0)}{2}}_{d\omega(0)} \Bigg) + f_1 \Bigg(\underbrace{\frac{\omega(N) - \omega(N-1)}{2}}_{d\omega(N)} \Bigg) + \sum_{i=1}^{M-1} f(\omega(i)) \Bigg(\underbrace{\frac{\omega(i+1) - \omega(i-1)}{2}}_{d\omega(i)} \Bigg) \\
&= \sum_{i=0}^{M} d\omega(i)\, f(\omega(i))
\end{aligned}
$$

therefore the weights are given by

$$
d\omega(i) = \begin{cases} \frac{\omega(1)-\omega(0)}{2} & i = 0 \\ \frac{\omega(i+1)-\omega(i-1)}{2} & i \in \{1, \ldots, M-1\} \\ \frac{\omega(N)-\omega(N-1)}{2} & i = M \end{cases}
$$

3.2. Interpolation - Inverse mapping

In order to calculate a function f which is known on a given grid $\omega(i)$, on another grid $\epsilon(i)$ we need to interpolate. We restict ourselfes to linear interpolation and we further assume, that the inverse mapping of the first grid

$$
i(\omega) : \ \mathbb{R} \to \mathbb{N} : \quad \omega \mapsto i \qquad ; \ \omega \in [\omega(0), \omega(M)]
$$

is known.

The interpolation is then done in the following way. For each point of the new grid $\epsilon(j) \equiv \epsilon_j$ one calculates the inverse of the old grid $I = i(\epsilon_j)$. The corresponding old grid point $\omega(I) \equiv \omega_i$ is then the nearest grid point to the interpolation point ϵ_j (Figure 3.2).

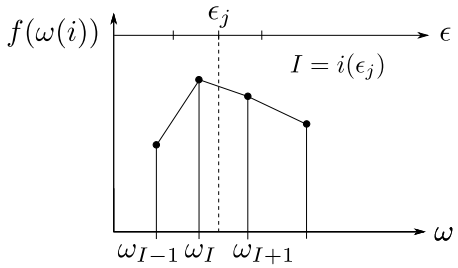


Figure 3.2.: Linear interpolation of $f(\{\omega_i\})$ on a new grid $\{\epsilon_j\}$

The value of the function on the new grid $f(\epsilon_j)$ is then obtained by linear interpolation between the old grid point ω_I and its nearest neighbor (right neighbor for $\omega_I \leq \epsilon_j$, left

neighbor for $\omega_I > \epsilon_j$).

$$f(\epsilon_j) = \begin{cases} f(\omega_I) + \frac{f(\omega_{I+1}) - f(\omega_I)}{\omega_{I+1} - \omega_I}(\epsilon_j - \omega_I) & \text{for } \omega_I < \epsilon_j \\ f(\omega_I) + \frac{f(\omega_{I-1}) - f(\omega_I)}{\omega_{I-1} - \omega_I}(\epsilon_j - \omega_I) & \text{for } \omega_I > \epsilon_j \end{cases}$$

4. Simple grids

In the following we will introduce all kinds of available grids together with their numerical structure. These grids will be the building blocks of the multigrid.

4.1. The base class: grid

As it was shown in chapter 3 all one needs for numerical integration is the integration grid itself and the corresponding weights. All of this is comprised in the class `grid` whose members are listed in table 2.1. The grid class is defined in the `grid.h` file as follows

```
class grid
{
        public:
        int M;
        double omega_min;
        double omega_max;
        vector <double> omega;
        vector <double> domega;
        virtual unsigned int inverse (double omega)=0;
};
```

The grid itself and its weights are `STL-vectors` [1]. The inverse mapping is virtual, and so is the grid class itself. Only the classes which are derived from the grid class are non-virtual. If one is not familiar with the concept of inheritance one can think of it in the following way: Since all integration grids should possess the above attributes (table 2.1), it is reasonable to define a skeleton for an integration grid. This is the virtual grid class. It is not possible to create an instance of the grid class, but its possible to give it as an argument in a function. For example:

```
void saveGrid(grid & mgrid, string filename)
{
        ofstream out;
        out.open(filename.c_str());
        for (int j=0; j!=mgrid.M+1; j++)
        {
                out << j << "\t" << mgrid.omega[j] << endl;
        }
        out.close();
}
```

And this is the whole purpose of using a virtual base grid class. In the following we will introduce all kinds of available grids.

4.2. Equidistant grid: equigrid

The simplest grid class is the equidistant grid class. It is called `equigrid` and the grid is defined in the following way:

$$\omega(i) = \omega_{min} + d\omega \cdot i$$

with $i \in 0, \ldots, M$ and $\omega(M) = \omega_{max}$. Beside the grid class attributes (table 2.1), the equigrid has only one additional variable: the grid resolution $d\omega$. The index function, which is the inverse of $\omega(i)$ is given by

$$i(\omega) = \frac{\omega - \omega_{min}}{d\omega}$$

The class is defined in the files `grid.h` and `grid.cpp`. Its defining member variable are listed in table 4.1.

Name	Member	Constructor Argument	Description
M	`int M`	1st	Number of grid points
ω_{min}	`double omega_min`	2nd	Lower boundary
ω_{max}	`double omega_max`	3rd	Upper boundary

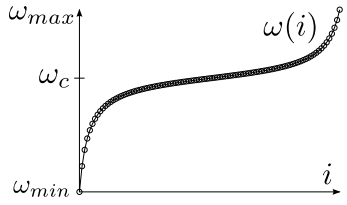
Table 4.1.: Defining properties of the `equigrid` class

An equigrid is created by calling its constructor (see table 4.1):

```
equigrid egrid(M, omega_min, omega_max);
```

4.3. Tangential grid: tangrid

The tangential grid is called `tangrid`. Beside the grid class attributes (table 2.1), it has the a cluster point at ω_c



The "sharpness" of this curve is controlled by the parameter c. The grid is defined by

$$\omega(i) = c\tan(u_1 + \Delta u \cdot i) + \omega_c$$

where $i \in 0, \ldots, M$, and the parameters u_1^0 and Δu are calculated such, that boundary conditions $\omega(0) = \omega_{min}$ and $\omega(M) = \omega_{max}$ are fulfilled. Hence

$$u_1 = \arctan\left(\frac{\omega_{min} - \omega_c}{c}\right)$$
$$u_2 = \arctan\left(\frac{\omega_{max} - \omega_c}{c}\right)$$
$$\Delta u = \frac{u_2 - u_1}{M}$$

The index function, i.e. the inverse of the grid is given by

$$i(\omega) = \frac{1}{\Delta u}\left(\arctan\left(\frac{\omega - \omega_c}{c}\right) - u_1\right)$$

The point density reads

$$\frac{di}{d\omega}(\omega) = \frac{1}{c\Delta u\left(1 + \left(\frac{\omega-\omega_c}{c}\right)^2\right)}$$

The class is defined in the files `grid.h` and `grid.cpp`. Its defining member variable are listed in table 4.2.

Name	Member	Constructor Argument	Description
M	`int M`	1st	Number of grid points
ω_{min}	`double omega_min`	2nd	Lower boundary
ω_{max}	`double omega_max`	3rd	Upper boundary
ω_c	`double omega_c`	4th	Center point
c	`double c`	5th	Controls sharpness

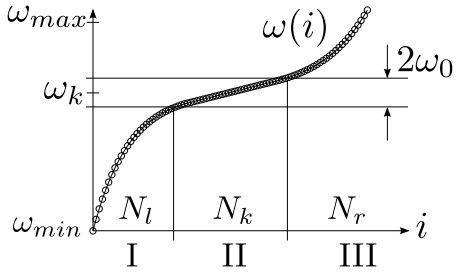
Table 4.2.: Defining properties of the `tangrid` class

A tangrid is initialized by its constructor (see table 4.2):

```
tangrid tgrid(M, omega_min, omega_max, omega_c, c);
```

4.4. Logarithmic grid: loggrid

To resolve very sharp peaks in the integrand function the logarithmic grid is the right tool. It is named `loggrid` and it constructed in the following way. There are three grid regions: one below I, one around II and one above III the center point ω_k with the highest grid point density.



Region I and III are exponential and region II is linear. The half width of the linear region ω_0 also defines the "sharpness" of the exponential grid regions. The number of grid points in region I, N_l and in region III, N_r can be chosen different from each other. Both will also influence the "sharpness" of the exponential grid regions. The resolution of the linear grid region $d\omega_k$ is the same as the maximal resolution in the exponential grid regions and it determines the number of grid points in region II, N_k (see Appendix A.2). The grid is defined as

$$\omega(i) = \begin{cases} -\exp(-c_1(i - i_1)) + \omega_k & i \in \{0, \ldots, N_l - 1\} \\ \omega_k - \omega_0 + d\omega_k(i - N_l) & i \in \{N_l, \ldots, N_l + N_k - 1\} \\ \exp(c_2(i - i_2 - N_l - N_k)) + \omega_k & i \in \{N_l + N_k, \ldots, N_l + N_k + N_r\} \end{cases} \tag{4.1}$$

The four boundary conditions:

$$\begin{aligned} \omega(0) &= \omega_{min} & \omega(N_l - 1) &= \omega_k - \omega_0 \\ \omega(N_l + N_k) &= \omega_k + \omega_0 & \omega(N_l + N_k + N_r) &= \omega_{max} \end{aligned} \tag{4.2}$$

determine the four parameters i_1, i_2, c_1 and c_2 (see Appendix A.1). The inverse is given by

$$i(\omega) = \begin{cases} \frac{\log(\omega_k - \omega)}{-c_1} + i_1 & \text{for } \omega \in [\omega_{min}, \omega_k - \omega_0) \\ \frac{\omega - \omega_k + \omega_0)}{d\omega_k} + N_l & \text{for } \omega \in [\omega_k - \omega_0, \omega_k + \omega_0) \\ \frac{\log(\omega - \omega_k)}{c_2} + i_2 + N_l + N_k & \text{for } \omega \in [\omega_k + \omega_0, \omega_{max}] \end{cases}$$

and the grid point density reads

$$\frac{di(\omega)}{d\omega} = \begin{cases} \frac{1}{c_1(\omega_k - \omega)} & \text{for } \omega \in [\omega_{min}, \omega_k - \omega_0) \\ \frac{1}{d\omega_k} & \text{for } \omega \in [\omega_k - \omega_0, \omega_k + \omega_0) \\ \frac{1}{c_2(\omega - \omega_k)} & \text{for } \omega \in [\omega_k + \omega_0, \omega_{max}] \end{cases}$$

The class is defined in the files `grid.h` and `grid.cpp`. Its defining member variable are listed in table 4.3.

A loggrid is initialized by its constructor (see table 4.3):

```
loggrid loggrid(N_l, N_r, omega_min, omega_max, omega_k, omega_0);
```

Name	Member	Constructor Argument	Description
N_l	`int N_l`	1^{st}	Number of grid points in region I
N_r	`int N_r`	2^{nd}	Number of grid points in region III
ω_{min}	`double omega_min`	3^{rd}	Lower boundary
ω_{max}	`double omega_max`	4^{th}	Upper boundary
ω_k	`double omega_k`	5^{th}	Center point
ω_0	`double omega_0`	6^{th}	Half width of the region II

Table 4.3.: Defining properties of the `loggrid` class

4.5. Using the grid classes

All the above grids are derived from the base class `grid`. Therefore it is possible to write an function which performs the numerical integration without specifing the type of grid one wants to use:

```
double integrate(double (&f)(double), grid & g)
{
        double I=0
        for (int i=0; i<=mgrid.M; i++)
        {
                I+= f(mgrid.omega[i]) * domega[i];
        }
        return I;
}
```

which can then be uses for example with an equigrid and a Gauss function

```
double gauss(double x)
{
        return exp(-x*x);
}
int main()
{
        equigrid egrid(100, -4, 4)
        double I;
        I=integrate(gauss, egrid);
}
```

5. Multigrid

5.1. Overview

A multigrid consist of a single basic grid which defines the outer boundaries of the multigrid and multiple grid regions (GR). Each of these grid regions can be equidistant, tangential or logarithmic. If the grid regions overlap each other, they will be cut in such a way, that the grid resolution remains constant while crossing over the cutting point. It could also happen that a particular grid region is forced out by another grid region because the purpose of the former one is already served by the latter one. The first added grid region defines the outer boundaries of the multigrid and it is excluded from the cutting procedure. It is called basic grid region (BGR).

A multigrid is created in the following way (see listing 5.1). First it has to be initialized. Afterwards one can add, replace and remove multiple grid regions on the multigrid. This will be explained in detail in section 5.2. Finally the member function `create` in invoked. Hereby the cutting and selection procedure will be performed and the multigrid will be created.

Listing 5.1: Creating a multigrid

```
multigrid mgrid;
...
... //add, replace and remove grid regions
...
mgrid.create()
```

The multigrid is also derived from the grid class and can therefore be used exactly in the same way then the other grids (see chapter 4.5)

5.2. Grid regions

In this chapter we will introduce the notion of a grid region. We will assume, that the purpose of a grid region is to resolve some feature of the integrand function which is located a a single point. This will be the center point of the grid region ω_c. Together with the left and right boundaries ω_l and ω_r, the type of grid region (equidistant, tangential or logarithmic) and the grid region parameters (e.g. ω_k and ω_0 for the loggrid) these are the defining properties of a grid region. The grid region is implemented as a class named `gridregion` and is defined in the file `multigrid.h`. Its defining member variables are shown in table 5.1.

Name	Description
ω_c	Center point
ω_l	Lower boundary
ω_r	Upper boundary
`type`	Type of grid region
`id`	Name of the grid region (optional)
	Grid region parameters (e.g. ω_k and ω_0 for the loggrid)

Table 5.1.: Defining properties of the grid region

It obvious, that a possible cutting point can not be arbitrary narrow to the center point ω_c. For example in the loggrid case, for numerical reasons at least three point of the exponential grid regions must survive in order to have a starting, an intermediate and an endpoint of the grid. Also the corresponding weights should not be smaller than the machine precision. Therefore one introduces a buffer region around ω_c which is defined by its upper and lower boundaries ω_+ and ω_-. In section 5.3 we will see, that in order to calculate the cutting point according to the grid point density (grid resolution) the maximal grid resolution on both sides of the center point $d\omega^l_{min}$ and $d\omega^r_{min}$ is necessary. Hereby we assume that the grid point density is monotonically increasing from the boundaries towards the center of the grid region. (This is true for all three kinds of grids from chapter 4). Table 5.2 shows the relevant derived properties of the grid region class, i.e. they are calculated directly out of the defining properties.

Name	Description
ω_+	Upper boundary of the buffer region around ω_c
ω_-	Lower boundary of the buffer region around ω_c
$d\omega^l_{min}$	Maximal resolution left to ω_c
$d\omega^r_{min}$	Maximal resolution right to ω_c

Table 5.2.: Relevant derived properties of the grid region

Note that there are actually more member variables. But since they we will never appear to the user will not discuss them here. In the following we will introduce the different kinds of grid regions. There are equidistant, tangential and logarithmic grid regions. Naturally, the boundaries of a particular grid region ω_l and ω_r will correspond to the boundary ω_{min} and ω_{max} of the simple grids from chapter 4. In table 5.3 the mapping from the grid class member variables to the grid region member variables is shown. As mentioned earlier, the boundaries of the buffer region around the center point are chosen such, that there will be 3 points left. In the case of the logarithmic grid region, the buffer region always includes the linear region II of the loggrid.

Grid region variable	equidistant	tangential	logarithmic
ω_c	-	ω_c	ω_k
ω_l	ω_{min}	ω_{min}	ω_{min}
ω_r	ω_{max}	ω_{max}	ω_{max}
`type`	`'equi'`	`'tan'`	`'log'`
ω_+	$\omega_c + 3d\omega$	$\omega_c + 3c\Delta u$	$\omega(i = N_l + N_k + 3)$
ω_-	$\omega_c - 3d\omega$	$\omega_c - 3c\Delta u$	$\omega(i = N_l - 3)$
$d\omega^l_{min}$	$d\omega$	$c\Delta u$	see Appendix A.2
$d\omega^r_{min}$	$d\omega$	$c\Delta u$	see Appendix A.2

Table 5.3.: Relation of grid class and grid region member variables

5.2.1. Adding grid regions

In this section we introduce the syntax for adding grid regions to the multigrid. (See section 5.2.3 for examples). An equidistant grid region consists of an instance of an equigrid (see 4.2). It is added to the multigrid by the member function

```
void add_gr_equi
```

Its arguments are shown in table 5.4.

Argument	Type	Description
1^{st}	`int`	Number of grid points (M)
2^{nd}	`double`	Lower boundary (ω_l)
3^{rd}	`double`	Upper boundary (ω_r)
4^{th}	`double`	Center point (ω_c)
(5^{th})	`string`	Name of the grid region (`id`)(optional)

Table 5.4.: Arguments of the member function `add_gr_equi`

A tangential grid region consists of an instance of an tangrid (see 4.3). It is added to the multigrid by the member function

```
void add_gr_tan
```

Its arguments are shown in table 5.5.

A logarithmic grid region consists of an instance of an loggrid (see 4.4). It is added to the multigrid by the member function

```
void add_gr_log
```

Its arguments are shown in table 5.6.

Argument	Type	Description
1^{st}	`int`	Number of grid points (M)
2^{nd}	`double`	Lower boundary (ω_l)
3^{rd}	`double`	Upper boundary (ω_r)
4^{th}	`double`	Center point (ω_c)
5^{th}	`double`	Controls sharpness (c)
(6^{th})	`string`	Name of the grid region (`id`)(optional)

Table 5.5.: Arguments of the member function `add_gr_tan`

Argument	Type	Description
1^{st}	`int`	Number of grid points in region I (N_l)
2^{nd}	`int`	Number of grid points in region III (N_r)
3^{rd}	`double`	Lower boundary (ω_l)
4^{th}	`double`	Upper boundary (ω_r)
5^{th}	`double`	Center point (ω_k)
6^{th}	`double`	Half width of region II (ω_0)
(7^{th})	`string`	Name of the grid region (`id`)(optional)

Table 5.6.: Arguments of the member function `add_gr_log`

5.2.2. Replacing and removing grid regions

We have seen, that if a grid region is added to the multigrid, it is optional give a name to this grid region. In contrast to that, you can only replace or remove grid regions which have a name. One can check if a grid region with the name `id` exists, by calling the function

```
bool gr_exists(string id)
```

If this is the case one can remove it by

```
void rem_gr(string id)
```

In order to replace a grid region by one with the same name but different attributes one calls one of the replace functions

```
void replace_gr_equi(...)
void replace_gr_tan(...)
void replace_gr_log(...)
```

where the arguments are exactly the same as the one for the add functions for the corresponding type of grid (see tables 5.4, 5.5 and 5.6). The only difference is that for the replace functions, you definitely have to give the name of the grid region which is to be replaced, as the `id` argument.

5.2.3. Examples

In this section we will show in some examples how the above grid regions can be added, replaced or removed from the multigrid.

Listing 5.2: Example for the adding of equidistant grid regions

```
multigrid mgrid;
mgrid.add_gr_equi(100, -1, 1, 0);
mgrid.add_gr_tan(100, 0.2, 0.5, 0.3, 0.01);
mgrid.add_gr_log(100, 100, 0.4, 0.7, 0.6, 1E-6, "lgrid");
mgrid.create()
```

In listing 5.2 we show an example for the adding of grid regions. At first an equidistant grid with 100 points from -4 to 4 is added to the multigrid. This defines the basic grid region and therefore the boundaries of the multigrid. The center point of a basic grid region (here at 0) is obsolete because there is no cutting. Afterwards a tangential grid region with 200 points from 0.2 to 0.5 with a center point at 0.3 is added. The parameter c is 0.01 here. Finally a third, logarithmic grid region is added. It overlaps with the tangential grid region and an appropriate cutting point will be calculated such that the grid resolution remains constant across the cutting point. The logarithmic grid region is named "lgrid" and has 100 points in both of the exponetial grid regions. Its boundaries are from 0.4 to 0.7, its center point ω_k is at 0.6 and the half width of the linear region ω_0 is 10^{-6}.

5.3. Cutting procedure

If two grid regions intersect, they have to be cut or one of the grid regions has to be erased. This is the content of this chapter.

5.4. Subgrid and inserting grid regions

If all grid regions have gone through the cutting procedure explained in 5.3, we are left with a bunch of non-intersection grid regions. These grid regions are now ready to be inserted into the basic grid region. But there is one difficulty we have not mentioned so far which is the fact that one has to take care of the inverse. Since all grid regions consist of one of the simple grid classes from chapter 4, their inverse mapping is know.

5.5. Inverse of the multigrid

5.6. Fundamental and special grid regions

6. Mesh - Multigrid without inverse mapping

A. Loggrid parameter

A.1. Boundary conditions

In the following we will show how the four boundary conditions (4.2)

$$\omega(0) = \omega_{min} \tag{A.1}$$
$$\omega(N_l - 1) = \omega_k - \omega_0 \tag{A.2}$$
$$\omega(N_l + N_k) = \omega_k + \omega_0 \tag{A.3}$$
$$\omega(N_l + N_k + N_r) = \omega_{max} \tag{A.4}$$

determine the four parameters i_1, i_2, c_1 and c_2 in the definition of the logarithmic grid (4.1):

$$\omega(i) = \begin{cases} -\exp(-c_1(i - i_1)) + \omega_k & i \in \{0, \dots, N_l - 1\} \\ \omega_k - \omega_0 + d\omega_k(i - N_l) & i \in \{N_l, \dots, N_l + N_k - 1\} \\ \exp(c_2(i - i_2 - N_l - N_k)) + \omega_k & i \in \{N_l + N_k, \dots, N_l + N_k + N_r\} \end{cases}$$

Determine i_1 and c_1:

Insert 4.1 into (A.1) and (A.2):

$$-\exp(-c_1(0 - i_1)) + \omega_k = \omega_{min} \quad \Leftrightarrow \quad \log(\omega_k - \omega_{min}) = c_1 i_1 \tag{A.5}$$
$$-\exp(-c_1(N_l - 1 - i_1)) + \omega_k = \omega_k - \omega_0 \quad \Leftrightarrow \quad \log(\omega_0) = -c_1(N_l - 1 - i_1) \tag{A.6}$$

Difference (A.5) − (A.6):

$$\log\left(\frac{\omega_k - \omega_{min}}{\omega_0}\right) = c_1(N_l - 1) \quad \boxed{\Rightarrow c_1 = \frac{\log\left(\frac{\omega_k - \omega_{min}}{\omega_0}\right)}{N_l - 1}}$$

Insert into (A.6):

$$\boxed{i_1 = \frac{\log(\omega_k - \omega_{min})}{c_1}}$$

Determine i_2 and c_2:

Insert 4.1 into (A.1) and (A.2):

$$\exp(c_2(0-i_2)) + \omega_k = \omega_k + \omega_0 \quad \Leftrightarrow \quad \log(\omega_0) = c_2(0-i_2) \tag{A.7}$$

$$\exp(c_2(N_r-i_2)) + \omega_k = \omega_{\max} \quad \Leftrightarrow \quad \log(\omega_{max}-\omega_k) = c_2(N_r-i_2) \tag{A.8}$$

Difference (A.8) − (A.7):

$$\log\left(\frac{\omega_{max}-\omega_k}{\omega_0}\right) = c_2(N_r) \Rightarrow \boxed{c_2 = \frac{\log\left(\frac{\omega_{max}-\omega_k}{\omega_0}\right)}{N_r}}$$

Insert into (A.8):

$$\boxed{i_2 = -\frac{\log(\omega_{max}-\omega_k)}{c_2} + N_r}$$

A.2. Maximal resolution

The maximal resolution of the exponential grid regions to the left (region I), $d\omega^l_{max}$ and to the right (region III), $d\omega^r_{max}$ of the center point in a logarithmic grid reads

$$\begin{aligned} d\omega^l_{max} &= \omega(N_l-1) - \omega(N_l-2) \\ &= -\exp(-c_1(N_l-1-i_1)) + exp(-c_1(N_l-2-1_1)) \\ &= \exp(-c_1(N_l-1-i_1))(e^{c_1}-1) \end{aligned}$$

analogously we have

$$\begin{aligned} d\omega^r_{max} &= \omega(N_l+N_k+1) - \omega(N_l+N_k) \\ &= \exp(-c_2 i_2)(e^{c_2}-1) \end{aligned}$$

The resolution of the linear grid (region II) will be the maximum of both

$$d\omega_k = max(d\omega^l_{max},\, d\omega^r_{max})$$

Bibliography

[1] "Standard template library programmer's guide." `http://www.sgi.com/tech/stl/`.