Multigrid

A highly versatile integration grid class for C++

Tobias Stollenwerk

May 7, 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 favoring the better resolved 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 possibility 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 executable `example.out`.

2.2. First Steps

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

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

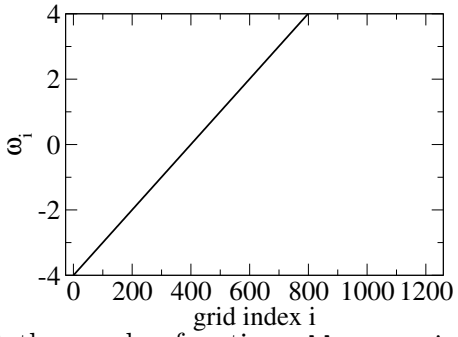
can be written as

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

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 continuous 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 functionality 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 (mapping $i \to \omega$)
`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).

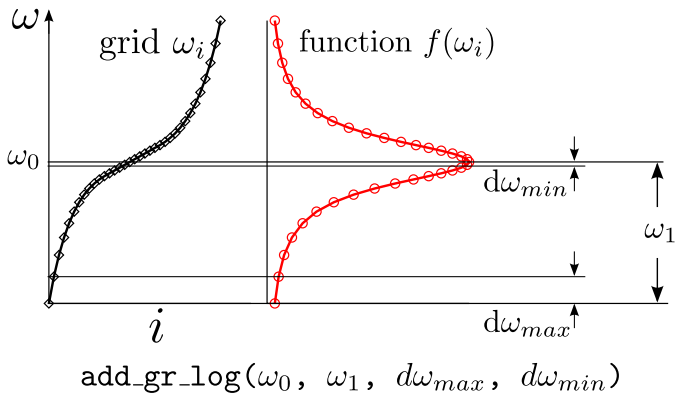
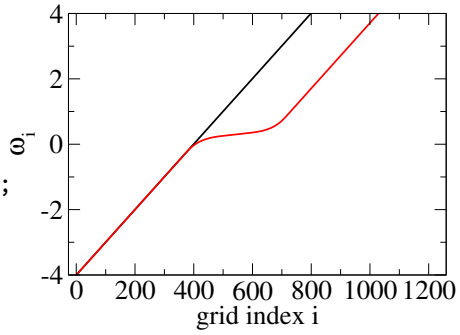


Figure 2.1.: Logarithmically dense grid region to resolve a peaked integrand function

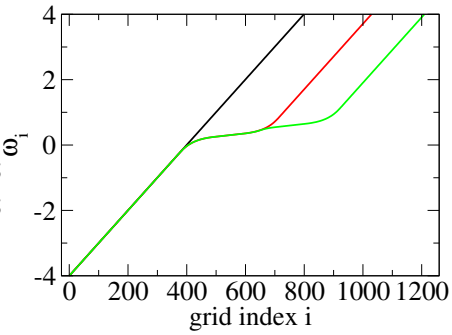
It is created by the function `add_gr_log(`ω_0`,` ω_1`,` $d\omega_{max}$`,` $d\omega_{min}$`)`. For example the following code adds a LGR on top of the equidistant grid region from above. Note that since the equidistant grid region was added first, it determines the outer boundaries of the whole grid (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 favoring 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.

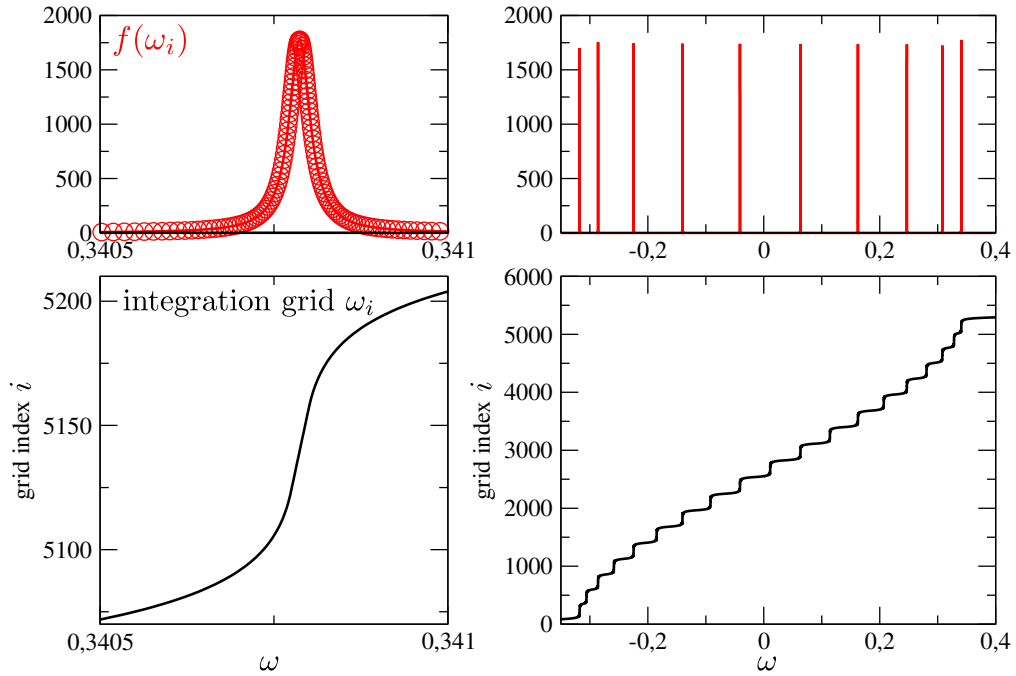


Figure 2.2.: Multigrid with various logarithmically dense grid regions to resolve a multiple peaked integrand function

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 restrict ourselves to the trapezoidal rule which approximates the integrand function by linear regions connecting the sample points (see figure 3.1).

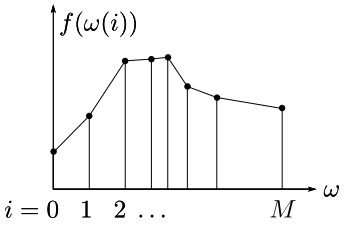


Figure 3.1.: Trapezoidal rule for a non-equidistant grid.

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 restrict ourselves 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).

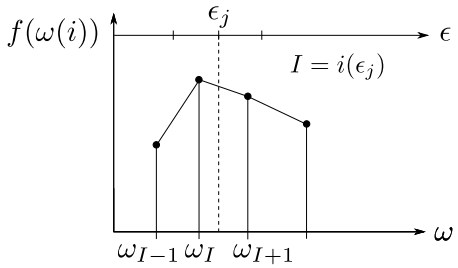


Figure 3.2.: Linear interpolation of $f(\{\omega_i\})$ on a new grid $\{\epsilon_j\}$

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 point density reads

$$\frac{di}{d\omega}(\omega) = \frac{1}{d\omega} \tag{4.1}$$

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`	1^{st}	Number of grid points
ω_{min}	`double omega_min`	2^{nd}	Lower boundary
ω_{max}	`double omega_max`	3^{rd}	Upper boundary

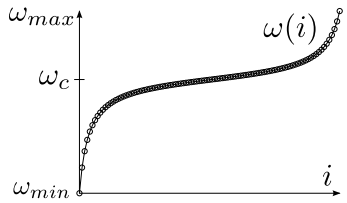
Table 4.1.: Defining properties of the `equigrid` class

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

$$\frac{di}{d\omega}(\omega) = \frac{1}{c\Delta u\left(1 + \left(\frac{\omega-\omega_c}{c}\right)^2\right)} \tag{4.2}$$

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

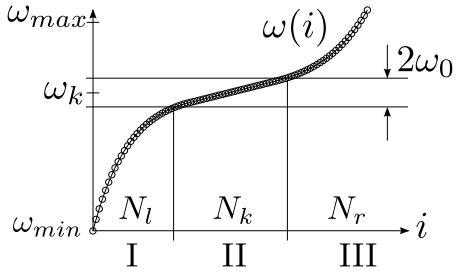
Table 4.2.: Defining properties of the `tangrid` class

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

$$\omega(i) = \begin{cases} -\exp(-c_1(i - i_1)) + \omega_k & i \in \{0, \ldots, N_l - 1\} \\ \omega_k - \omega_0 + d\omega_k(i - N_l) & i \in \{N_l, \ldots, N_l + N_k - 1\} \\ \exp(c_2(i - i_2 - N_l - N_k)) + \omega_k & i \in \{N_l + N_k, \ldots, N_l + N_k + N_r\} \end{cases} \tag{4.3}$$

The four boundary conditions:

$$\begin{aligned} \omega(0) &= \omega_{min} & \omega(N_l - 1) &= \omega_k - \omega_0 \\ \omega(N_l + N_k) &= \omega_k + \omega_0 & \omega(N_l + N_k + N_r) &= \omega_{max} \end{aligned} \tag{4.4}$$

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

$$\frac{di(\omega)}{d\omega} = \begin{cases} \frac{1}{c_1(\omega_k - \omega)} & \text{for } \omega \in [\omega_{min}, \omega_k - \omega_0) \\ \frac{1}{d\omega_k} & \text{for } \omega \in [\omega_k - \omega_0, \omega_k + \omega_0) \\ \frac{1}{c_2(\omega - \omega_k)} & \text{for } \omega \in [\omega_k + \omega_0, \omega_{max}] \end{cases} \tag{4.5}$$

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 specifying 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

In this chapter will introduce all the knowledge which is required for the usage of the multigrid. In order to understand the selection and cutting procedure which is the very heart of the multigrid class we refer to chapter 6.

5.1. Overview

A multigrid consist of a single basic grid region which defines the outer boundaries of the multigrid and multiple grid regions. 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. How this is done in detail will be part of chapter 6.

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 6.1 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 $d\omega_l$ and $d\omega_r$ 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
$d\omega_l$	Maximal resolution left to ω_c
$d\omega_r$	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_-	$\omega_c - 3d\omega$	$\omega_c - 3c\Delta u$	$\omega(i = N_l - 3)$
$d\omega_l$	$d\omega$	$c\Delta u$	see Appendix A.2
$d\omega_r$	$d\omega$	$c\Delta u$	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 member 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 member 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 adding 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, "gr");
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 "gr" and has 100 points in both of the exponential 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}. The resulting multigrid is shown in figure 5.1.

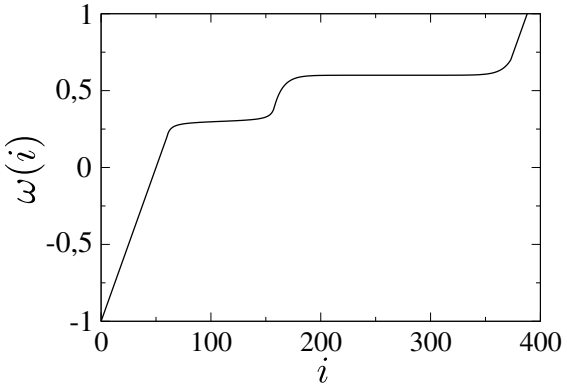


Figure 5.1.: Multigrid corresponding to listing 5.2

Listing 5.3: Example for replacing a grid region

```
mgrid.replace_gr_equi(100, -0.5, 0.0, "gr");
mgrid.create();
```

In listing 5.3 the logarithmic grid region named "gr" from the multigrid instance "mgrid" is replace by an equidistant grid region with the same name. In order to apply this change the create function has to be invoked once again. Note, that a if one tries to

replace a non-existing grid region, this will result in a simple adding of the desired grid region. The resulting multigrid is shown in figure 5.2.

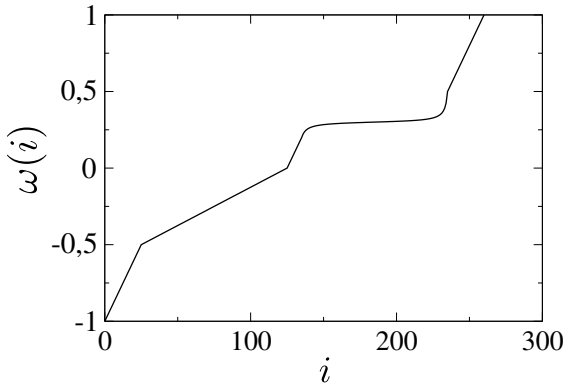


Figure 5.2.: Multigrid corresponding to listing 5.3

5.2.4. More convenient ways of adding and replacing

There are more convenient ways of adding or replacing equidistant and logarithmic grid regions. For the former it might be desirable to specify the grid resolution $d\omega$ instead of the number of grid points M. And for the latter it would be convenient just to give the maximal resolution at the center point $d\omega_{min}$ and the minimal resolution at the boundaries $d\omega_{min}$ together with the center point ω_k and the half width of grid region $\omega_1 = (\omega_r - \omega_l)/2$ instead of specifying the grid point numbers N_l and N_r, ω_0 and the boundaries ω_l and ω_r. Therefore there are wrapper function for the adding and replacing of equidistant and logarithmic grid regions which fulfill possess these properties. They are shown in the tables 5.7 and 5.8. For examples see the quick start (Chapter 2), where these wrapper functions are uses throughout the entire chapter. Note that the center point of the equidistant grid region is naturally chosen to be $\omega_c = (\omega_r - \omega_l)/2$.

Argument	Type	Description
1^{st}	`double`	Lower boundary (ω_l)
2^{nd}	`double`	Upper boundary (ω_r)
3^{rd}	`double`	Resolution ($d\omega$)
(4^{th})	`string`	Name of the grid region (`id`)(optional)

Table 5.7.: Arguments of the wrapper functions `add_gr_equi` and `replace_gr_equi`

For the logarithmic grid region the mapping from maximal and minimal resolutions $d\omega_{max}$ and $d\omega_{min}$ to the loggrid variables ω_0, N_l and N_r is determined by the following

Argument	Type	Description
1^{st}	`double`	Center point (ω_k)
2^{nd}	`double`	Half width (ω_1)
3^{rd}	`double`	Maximal resolution at the center point ($d\omega_{min}$)
4^{th}	`double`	Minimal resolution at the boundaries ($d\omega_{max}$)
(5^{th})	`string`	Name of the grid region (`id`)(optional)

Table 5.8.: Arguments of the wrapper functions `add_gr_log` and `replace_gr_log`

equations

$$\omega_0 = \frac{\omega_1 d\omega_{min}}{d\omega_{max}}$$
$$N_l = N_r = log\left(\frac{\omega_1}{\omega_0}\right)\frac{\omega_1}{d\omega_{max}}$$

Note that hereby we made the logarithmic grid region symmetric around the center point.

5.3. Fundamental and special grid regions

As it was explained in the beginning of this chapter, the base grid region defines the outer boundaries of the multigrid. A grid region which is added to the multigrid after the base grid region is called fundamental grid region (FGR). The selection and cutting procedure will remove all overlaps between the FGR (see chapter 6. By adding such a FGR, all the grid points of the base grid which lie inside the boundaries of the FGR will be removed and replaced by the grid points of the FGR.

Up to now, the base grid has to be one of the grid regions based on the simple grids: equidistant, tangential or logarithmic. But in principle nothing speaks against making the base grid a multigrid itself. This is achieved by letting the basic grid region and the FGR go through the selection and cutting procedure and building a "multigrid"-like grid which serves as a "base" grid for some other grid regions which will be called special grid regions (SGR). The will be specified by replacing the '`gr`' in the member functions of section 5.2 by a '`sgr`'.

Listing 5.4: Example for adding special 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, "gr");
mgrid.add_sgr_equi(0.5, 0.8, 0.001);
mgrid.create();
```

Listing 5.4 is exactly the same as listing 5.2 but with an additional adding of a special equidistant grid region from 0.5 to 0.8 with resolution 0.001. (Note, that we used one of the wrapper functions from section 5.2.4 here.) This SGR will just "bump away" most parts of the logarithmic grid region, even thought its maximal resolution (10^{-6}) is much higher than the one of the SGR (10^{-4}). Therefore one can use SGR for "tricking" the selection and cutting procedure to some extent. It is needless to say, that if you add multiple SGR they will also run through a selection and cutting procedure. The resulting multigrid of listing 5.4 is shown in figure 5.3.

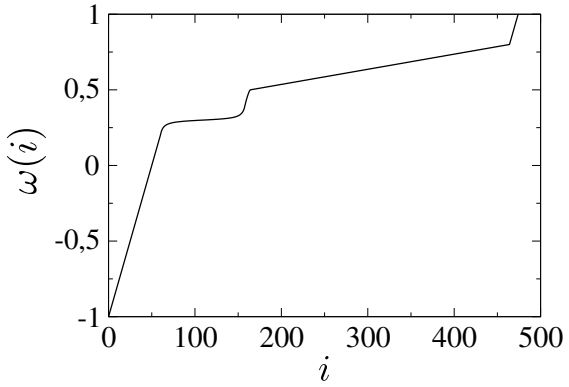


Figure 5.3.: Multigrid corresponding to listing 5.4

6. Selection and Cutting in the Multigrid

In this chapter we the details of the selection and cutting procedure will be explained. Moreover, we will show how to keep track of inverse mapping while adding fundamental and special grid regions to the multigrid. For the sake of clarity, we will ignore the special grid regions in most parts of this chapter and come back to them in the last part of this chapter.

6.1. Cutting procedure

As it was shown in the last chapter, each grid region has a center point ω_c, a maximal resolution below and above the center point $d\omega_l$ and $d\omega_r$, and a upper an lower boundary ω_l and ω_r. If there is overlap between two grid regions, the boundaries and grid parameters have to be adapted such, that the resolution remains constant by crossing over the cutting point. From now on, we will indicate the grid regions with the smaller center point by a superscript l for left, and the other one by a superscript r for right (figure 6.1).

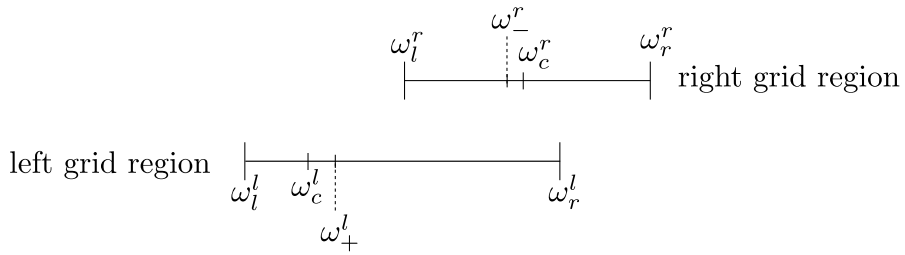


Figure 6.1.: Nomenclature for the cutting of two intersecting grid regions

The cutting point must be right from the center point of the left grid region plus a small grid dependent buffer, this point is called ω_+^l (see 5.2). It also has to be left from the center point of the right grid region minus a small grid dependent buffer, this point is called ω_-^r (see 5.2). Moreover the cutting point must not exceed the right boundary of the left grid region ω_r^l while at the same time it must not fall below the left boundary of the right grid region ω_l^r. Therefore the cutting point ω_s must lie inside the interval

$$\omega_s \in [\, max(\omega_+^l, \omega_l^r)\,, \;\; min(\omega_-^r, \omega_r^l)\,] \;:= I \tag{6.1}$$

In this interval, the grid point density of the left grid region $\frac{di}{d\omega}|_l$ is monotonically decreasing and the grid point density of the right grid region $\frac{di}{d\omega}|_r$ is a monotonically

increasing. The cutting point should be that point, where the grid point densities of both grid regions become equal

$$\left.\frac{di}{d\omega}\right|_l (\omega_s) \stackrel{!}{=} \left.\frac{di}{d\omega}\right|_r (\omega_s) \tag{6.2}$$

This equation, can be solved analytically for all combinations of equidistant, tangential and logarithmic grid regions. This is done in appendix B.1. The calculated cutting point does not always lie inside the interval I. In such a case one has to put more thought in what to do. This is done in the following where we list all possibilities which can occur if a solution to equation 6.2 exists (see also figure 6.2):

- **Case A**: $\omega_s \in I$: If the calculated cutting point lies inside the interval I, both grids are cut at this particular point so that there is no more intersection between the grids.
- **Case B**: $\omega_s \notin I$ and $\omega_s \in [\omega_l^r, \omega_+^l)$ and $\omega_s \notin (\omega_-^r, \omega_r^l]$: This means, that the maximal resolution of the left grid is smaller than the resolution of the right grid at ω_+^l. Therefore the purpose of the left grid region is assumed to be fulfilled by the right grid region, and it is skipped.
- **Case C**: $\omega_s \notin I$ and $\omega_s \notin [\omega_l^r, \omega_+^l)$ and $\omega_s \in (\omega_-^r, \omega_r^l]$: This means, that the maximal resolution of the right grid is smaller than the resolution of the left grid at ω_-^r. Therefore the purpose of the right grid region is assumed to be fulfilled by the left grid region, and it is skipped.
- **Case D**: $\omega_s \notin I$ and $\omega_s \in [\omega_l^r, \omega_+^l)$ and $\omega_s \in (\omega_-^r, \omega_r^l]$: The distance between both center points is so small, that the calculated cutting point lies inside both buffer regions. Here, the grid region with the smaller maximal resolution is skipped.
- **Case E**: $\omega_s \notin I$ and $\omega_s < \omega_l^r$. The resolution of the right grid region is greater than the resolution of the left grid region over the whole range of the right grid region, even at ω_l^r. Therefore the right grid region will not be cut at all. If there is still enough space for the left grid region, i.e. if $\omega_+^l \leq \omega_l^r$, the left grid region will be cut at ω_l^r. If not, the left grid region will be skipped.
- **Case F**: $\omega_s \notin I$ and $\omega_s > \omega_r^l$. The resolution of the left grid region is greater than the resolution of the right grid region over the whole range of the left grid region, even at ω_r^l. Therefore the left grid region will not be cut at all. If there is still enough space for the right grid region, i.e. if $\omega_-^r \geq \omega_r^l$, the right grid region will be cut at ω_r^l. If not, the right grid region will be skipped.

Of course there are cases, in which a solution to equation 6.2 does not exist. This happens, if the lowest resolution of one grid region is higher than the highest resolution of the other. This is basically the same situation as in the cases E and F in the above listing. The only difference is, that one has to find out which of the both grid regions should be favored. This is done by comparing the maximal resolution of both grid

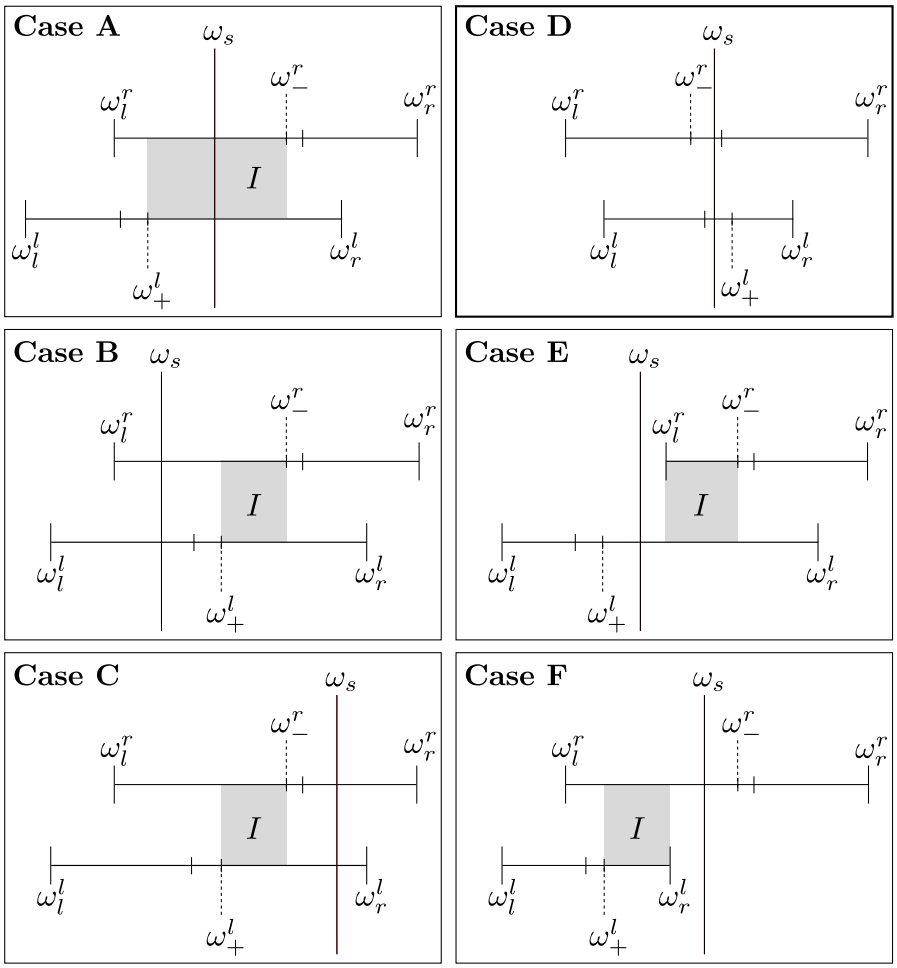


Figure 6.2.: Different cases for the cutting of two overlapping grid regions

regions. Afterwards, one checks if there is still enough space for the non-favored grid region. If this is the case it will be cut, otherwise it will be skipped (see cases E and F).

By cutting a grid region at a given cutting point, one has to adjust the grid parameter of this grid region such, that the maximal resolution remains the same. This is of course different for the equidistant, tangential and logarithmic grid regions and will be shown in appendix B.2.

Grid regions which have overlap with the boundaries of the basic grid region are selected and cut in a similar way. If the center point is outside of these boundaries, the grid region is skipped. It is also skipped if there is not enough space between the buffer region and the boundaries (like in the cases E and F). Otherwise it is cut at the boundaries.

So far, we have only considered two overlapping grid regions. We have shown how the selection an cutting procedure works in this case. Treating multiple overlapping grid regions is a bit more complicated, but can be boiled down to the two grid region

case in the end. We will show how this is done in the following. At first, all grid regions are sorted with respect to their center points. Afterwards one iterates through all neighboring pairs of grid regions, starting from the left, i.e. the grid region with the lowest center point, and applies the selection and cutting procedure to each of this pairs. If none of both grid regions is decided to be skipped, the algorithm proceeds to the next pair of neighboring grid regions. Otherwise, if one of the grid regions has to be skipped, it is necessary to re-evaluate all the the pairings to the left of this grid regions. Therefore the algorithm erases this particular grid region and restarts the iterating pair selection and cutting procedure from the left. This is continued until the most right grid region pair is reached.

6.2. Subgrids and inserting grid regions

If all fundamental grid regions have gone through the cutting procedure explained in 6.1, 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. 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. For the same reason, we know the inverse of the basic grid region. In order to calculate the inverse of the resulting multigrid, one has to keep track of the indices at which the non-overlapping grid regions are inserted. We will show how this is done in the next two section. First of all it will turn out to be useful to introduce the notion of a subgrid, and the corresponding class `subgrid`, here.

Name	Description
ω_l	Lower boundary
ω_r	Upper boundary
i_l	Lower index cut region in basic grid
i_r	Upper index cut region in basic grid
N_R	Difference in overall grid points before and after insertion
`type`	Type of subgrid
...	Grid region parameters (e.g. ω_k and ω_0 for the loggrid)

Table 6.1.: Properties of the subgrid

In table 6.1 the properties of the subgrid class are shown. Like a grid region, a subgrid contains the information about the boundaries and the type of the corresponding simple grid. It does not contain any information about the center point or the resolution since the cutting and selection procedure has already been performed. But it stores information about the inserting of the subgrid into the basic grid region, i.e. the lower and upper index of the region which will be cut out of the basic grid i_l and i_r, and the difference in the overall number of grid points before and after the insertion N_R.

The most decisive property of a subgrid is, that there is no overlap between different

subgrids (in contrast to grid regions). But after applying the selection and cutting procedure to the grid regions they will not have any overlap, too. Therefore all grid regions can be mapped to subgrids. After this is done, the subgrids can be inserted into the basic grid. In the following, we will show how a subgrid $\epsilon(i)$ of length N with boundaries ϵ_l and ϵ_r is inserted into a basic grid $\omega(i)$ of length M (see figure 6.3).

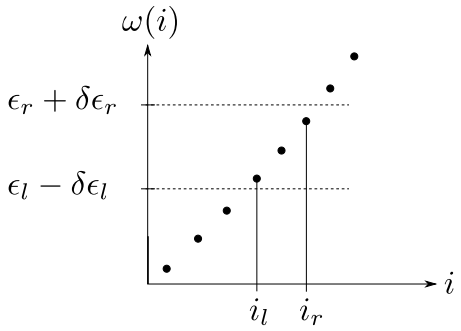


Figure 6.3.: Inserting of a subgrid

The indices i_l and i_r are obtained by

$$\begin{aligned} \omega(i_l) &> \epsilon_l - \delta\epsilon_l \quad \text{with } i_l \text{ minimal} \\ \omega(i_r) &< \epsilon_r + \delta\epsilon_r \quad \text{with } i_r \text{ maximal} \end{aligned}$$

where the machine precision buffer δ ensures that the resulting grid point difference ($\sim$ weights) does not become too small. The difference in the overall grid point number is then given by

$$N_R = N + 1 - (i_r - i_l + 1)$$

The subgrid is inserted into the base grid at the index i_l. The resulting grid $\tilde{\omega}(i)$ is then given by

$$\tilde{\omega}(i) = \begin{cases} \omega(i) & i = 0, \ldots, i_l - 1 \\ \epsilon(i - i_l) & i = i_l, \ldots, i_l + N \\ \omega(i - N_R) & i = i_l + N + 1, \ldots, M + N_R \end{cases}$$

The grid $\tilde{\omega}(i)$ then serves as a basic grid for the insertion of the next subgrid. Nevertheless if there is more than one subgrid, there is an additional subtlety which has to be taken into account: If one inserts a subgrid left from a subgrid which was inserted earlier, one has to shift the indices i_l and i_r for the latter one by the number of replaced grid points N_R of the former one. This means, that if one adds a subgrid with N_R^{new}, one has to shift the indices i_l and i_r of all subgrids which lie right of the former one by

$$\begin{aligned} i_l &\to i_l + N_R^{\text{new}} \\ i_r &\to i_r + N_R^{\text{new}} \end{aligned}$$

. By this procedure the multigrid is build up out of several subgrids and the basic grid.

6.3. Inverse of the multigrid

In this section we will show how to calculate the inverse $i(\omega)$ of a multigrid which consists of a basic grid and several subgrids. As an example we take the multigrid from listing 5.3, see figure 6.4.

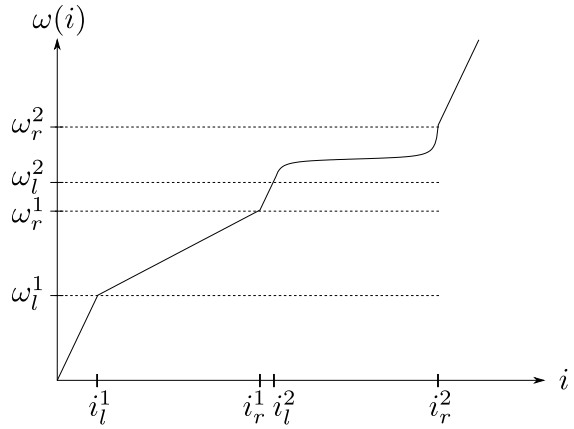


Figure 6.4.: Calculating the inverse mapping of a multigrid

There is a equidistant basic grid and two subgrids, an equidistant one and a tangential one. We will denote the basic grid by $\omega^0(i)$, the equidistant subgrid by $\omega^1(i)$ and the tangential subgrid by $\omega^2(i)$. The corresponding inverse mappings are denoted by $i^0(\omega)$ for the basic grid, $i^1(\omega)$ for the equidistant subgrid, $i^2(\omega)$ for the tangential subgrid. In order to calculate the inverse of a given value ω one has to find out if ω lies in one of the subgrid regions. And if this is the case, in which one of the subgrids is it. Then the inverse is calculated in the following way

$$i(\omega) = \begin{cases} i^1(\omega) + i_l^1 & \omega \in [\omega_l^1, \omega_r^1] \\ i^2(\omega) + i_l^2 & \omega \in [\omega_l^2, \omega_r^2] \\ i^0(\omega) + N_S & \omega \notin [\omega_l^1, \omega_r^1] \cup [\omega_l^2, \omega_r^2] \end{cases} \tag{6.3}$$

Where N_S is the sum of all number of replaced points N_R^k which correspond to subgrids k, which are left of ω, i.e. which fulfill $\omega_r^k < \omega$:

$$N_S = \sum_{\substack{k \\ \omega_r^k < \omega}} N_R^k$$

The generalization to an arbitrary number of subgrids it straight forward. Now it is clear, why the subgrids contain i_l and N_R: They are directly used in the calculation of the inverse mapping.

6.4. Fundamental and special grid regions

The above multigrid was comprised of a basic grid and several subgrids. Up to now these subgrids were build out of fundamental grid regions. Therefore they are so called fundamental subgrids and we will call the resulting grid a fundamental grid. In the last section we have shown how to calculate the inverse of such a fundamental grid. Nothing prevents us from taking this fundamental grid as a basic grid for inserting more subgrids. In accordance to the special grid regions introduced in the last chapter, we will call them special subgrids. After the selection and cutting procedure the special grid regions are mapped onto special subgrids. In the same way the fundamental subgrids were inserted into the basic grid to form the fundamental grid, the special subgrids are inserted into the fundamental grid to form the resulting multigrid. While calling the inverse of a value ω of a multigrid with special subgrids inside, the algorithm will begin by checking weather ω is inside one of the special subgrids (just like for the fundamental subgrids in equation 6.3). If this is the case it will calculate the inverse mapping of the subgrids directly (again, just like in equation 6.3). Otherwise it will call the inverse mapping of the fundamental grid (just like the inverse of the basic grid $i^0(\omega)$ is called in 6.3).

7. Mesh - Multigrid without Inverse Mapping

There might be occasions where the inverse mapping of the multigrid is not needed. The mesh grid class is exactly the same as multigrid class, but without the inverse mapping. All multigrid commands also work for the mesh class. For example

```
mesh 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, "gr");
mgrid.add_sgr_equi(0.5, 0.8, 0.001);
mgrid.create();
```

is exactly the same as listing 5.4, except for the first line. Note that the mesh class is not optimized for fast initialization since internally, the mesh `create` function will call the multigrid `create` function.

In comparison with the multigrid, there is only two additional features, which at the same time destroys the possibility of calculating the inverse of a mesh. The first one is the adding of single points inside grid by `add_spoint`. The second one is the adding of terminating points which serve as the total upper and lower boundary of the mesh by `add_lendpoint` for the lower endpoint and `add_rendpoint` for the upper endpoint. For example see listing 7.1.

Listing 7.1: Example of a mesh

```
mesh amesh;
amesh.add_gr_equi(10, 0.0, 1, 0.5);
amesh.add_gr_log(0.75, 0.1, 1E-1, 1E-2);
amesh.add_spoint(0.76);
for (int j=0; j!=100; j++)
{
        amesh.add_spoint(0.6+j/100.0);
}
amesh.add_lendpoint(0.15);
amesh.add_rendpoint(0.85);
amesh.create();
```

Note, that a inserted points will always take care of not letting the grid point difference ($\sim$ weights) getting to small (beyond machine precision). This is achieved by skipping all existing points which come to narrow to the inserted point.

A. Loggrid parameter

A.1. Boundary conditions

In the following we will show how the four boundary conditions (4.4)

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

determine the four parameters i_1, i_2, c_1 and c_2 in the definition of the logarithmic grid (4.3):

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

Determine i_1 and c_1:

Insert 4.3 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 4.3 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), $d\omega^l_{min}$ and to the right (region III), $d\omega^r_{min}$ of the center point in a logarithmic grid reads

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

analogously we have

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

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

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

B. Cutting grid regions

B.1. Calculation of cutting points

In this section we will calculate the points of equal grid point density ω_s between two grid of equidistant, tangential and logarithmic type. We will use the corresponding equations for the grid point density of these grid types, i.e. equations 4.1, 4.2 and 4.5 in order to solve equation 6.2. We will go through all possible combinations for the left and right grid region, where we denote the left grid region parameters by a superscript l and the right grid region parameter by a superscript r.

1. **loggrid/loggrid**:

$$\frac{1}{c_2^l(\omega_s - \omega_k^l)} \stackrel{!}{=} \frac{1}{c_1^r(\omega_k^r - \omega_s)}$$

$$\Rightarrow \boxed{\omega_s = \frac{c_2^l\omega_k^l + c_1^r\omega_k^r}{c_2^l + c_1^r}}$$

2. **loggrid/tangrid**:

$$\frac{1}{c_2^l(\omega_s - \omega_k^l)} \stackrel{!}{=} \frac{1}{c^r\Delta u^r\left(1 + \left(\frac{\omega_s - \omega_c^r}{c^r}\right)^2\right)}$$

$$\Leftrightarrow c_2^l(\omega_s - \omega_k^l) = c^r\Delta u^r + \frac{\Delta u^r}{c^r}(\omega_s - \omega_c^r)^2$$

$$\Leftrightarrow \omega_s^2 - \left(2\omega_c^r + \frac{c^r c_2^l}{\Delta u^r}\right)\omega_s + \left((\omega_c^r)^2 + (c^r)^2 + \frac{c^r c_2^l}{\Delta u^r}\omega_k^l\right) = 0$$

$$\Rightarrow \boxed{\omega_s = \omega_c^r + \frac{c^r c_2^l}{2\Delta u^r} - \sqrt{\underbrace{\left(\omega_c^r + \frac{c^r c_2^l}{2\Delta u^r}\right)^2 - (\omega_c^r)^2 - (c^r)^2 - \frac{c^r c_2^l}{\Delta u^r}\omega_k^l}_{<0 \text{ if no solution to 6.2 exists}}}}$$

3. **tangrid/loggrid**:

$$\frac{1}{c^l\Delta u^l\left(1 + \left(\frac{\omega_s - \omega_c^l}{c^l}\right)^2\right)} \stackrel{!}{=} \frac{1}{c_1^r(\omega_k^r - \omega_s)}$$

$$\Rightarrow \boxed{\omega_s = \omega_c^r - \frac{c^l c_1^r}{2\Delta u^l} + \sqrt{\underbrace{\left(\omega_c^l + \frac{c^l c_1^r}{2\Delta u^l}\right)^2 - (\omega_c^l)^2 - (c^l)^2 + \frac{c^l c_1^r}{\Delta u^l}\omega_k^r}_{<0 \text{ if no solution to 6.2 exists}}}}$$

4. **tangrid/tangrid**:

$$\frac{1}{c^l \Delta u^l \left(1 + \left(\frac{\omega_s - \omega_c^l}{c^l}\right)^2\right)} \stackrel{!}{=} \frac{1}{c^r \Delta u^r \left(1 + \left(\frac{\omega_s - \omega_c^r}{c^r}\right)^2\right)}$$

$$\Leftrightarrow c^l \Delta u^l + \frac{\Delta u^l}{c^l}(\omega_s - \omega_c^l)^2 - c^r \Delta u^r - \frac{\Delta u^r}{c^r}(\omega_s - \omega_c^r)^2 = 0$$

$$\Leftrightarrow \left(\frac{\Delta u^l}{c^l} - \frac{\Delta u^r}{c^r}\right)\omega_s^2 - \left(\frac{2\Delta u^l \omega_c^l}{c^l} - \frac{2\Delta u^r \omega_c^r}{c^r}\right)\omega_s + \frac{\Delta u^l}{c^l}(\omega_c^l)^2 - \frac{\Delta u^r}{c^r}(\omega_c^r)^2 + c^l \Delta u^l c^r \Delta u^r = 0$$

$$\Leftrightarrow \omega_s^2 - \underbrace{\left(\frac{2\Delta u^l \omega_c^l c^r}{\Delta u^l c^r - \Delta u^r c^l} - \frac{2\Delta u^r \omega_c^r c^l}{\Delta u^l c^r - \Delta u^r c^l}\right)}_{:=p}\omega_s + \underbrace{\left(\frac{c^r \Delta u^l (\omega_c^l)^2 - c^l \Delta u^r (\omega_c^r)^2 + \Delta u^l c^r (c^l)^2 - \Delta u^r c^l (c\right.}_{:=q}$$

$$\Rightarrow \boxed{\omega_s = \frac{p}{2} \pm \sqrt{\underbrace{\frac{p^2}{4} - q}_{\substack{<0 \text{ if no solution} \\ \text{to 6.2 exists}}}}}$$

Where the plus or minus sign is chosen such, that ω_s lies inside $[\omega_c^l, \omega_c^r]$.

5. **loggrid/equigrid**:

$$\frac{1}{c_2^l(\omega_s - \omega_k^l)} \stackrel{!}{=} \frac{1}{d\omega^r}$$

$$\Rightarrow \boxed{\omega_s = \omega_k^l + \frac{d\omega^r}{c_2^l}}$$

6. **equigrid/loggrid**:

$$\frac{1}{d\omega^l} \stackrel{!}{=} \frac{1}{c_1^r(\omega_k^r - \omega_s)}$$

$$\Rightarrow \boxed{\omega_s = \omega_k^r - \frac{d\omega^l}{c_1^r}}$$

7. **tangrid/equigrid**:

$$\frac{1}{c^l \Delta u^l \left(1 + \left(\frac{\omega_s - \omega_c^l}{c^l}\right)^2\right)} \stackrel{!}{=} \frac{1}{d\omega^r}$$

$$\Rightarrow \boxed{\omega_s = \omega_c^l + c^l \sqrt{\underbrace{\frac{d\omega^r}{c^l \Delta u^l} - 1}_{\substack{<0 \text{ if no solution} \\ \text{to 6.2 exists}}}}}$$

8. **equigrid/tangrid**:

$$\frac{1}{d\omega^l} \stackrel{!}{=} \frac{1}{c^r \Delta u^r \left(1 + \left(\frac{\omega_s - \omega_c^r}{c^r}\right)^2\right)}$$

$$\Rightarrow \boxed{\omega_s = \omega_c^r - c^r \sqrt{\underbrace{\frac{d\omega^l}{c^r \Delta u^r} - 1}_{\substack{<0 \text{ if no solution} \\ \text{to 6.2 exists}}}}}$$

9. **equigrid/equigrid**:

$$\frac{1}{d\omega^l} \stackrel{!}{=} \frac{1}{d\omega^r}$$

$$\Rightarrow \boxed{\omega_s = \begin{cases} \omega_l^r & \text{if } d\omega^r \leq d\omega^l \\ \omega_r^l & \text{if } d\omega^r > d\omega^l \end{cases}}$$

B.2. Adjust grid region parameters

After a cutting point according to section 6.1 is found. The left or right part of a grid region will be cut. The cutting point ω_s must be in the interval $[\omega_l, \omega_-]$ for cutting the left part of a grid region an in the interval $[\omega_+, \omega_r]$ for cutting the right part of a grid region. In the following we will show how the corresponding grid region parameters must be altered in order to preserve the maximal grid resolution.

1. **loggrid**, cut left

$$N_l \to \frac{1}{c_1} \log\left(\frac{\omega_k - \omega_s}{\omega_0}\right)$$
$$\omega_l \to \omega_s$$

2. **loggrid**, cut right

$$N_r \to \frac{1}{c_2} \log\left(\frac{\omega_s - \omega_k}{\omega_0}\right)$$
$$\omega_r \to \omega_s$$

3. **tangrid**, cut left

$$M \to \frac{1}{\Delta u} \arctan\left(\frac{\omega_r - \omega_c}{c}\right) - \arctan\left(\frac{\omega_s - \omega_c}{c}\right)$$
$$\omega_l \to \omega_s$$

4. **tangrid**, cut right

$$M \to \frac{1}{\Delta u} \arctan\left(\frac{\omega_s - \omega_c}{c}\right) - \arctan\left(\frac{\omega_l - \omega_c}{c}\right)$$
$$\omega_r \to \omega_s$$

5. **equigrid**, cut left

$$M \to \frac{\omega_r - \omega_s}{\omega_r - \omega_l} M$$
$$\omega_l \to \omega_s$$

6. **equigrid**, cut right

$$M \to \frac{\omega_s - \omega_l}{\omega_r - \omega_l} M$$
$$\omega_l \to \omega_s$$

Note, that all the grid point numbers N_l, N_r, M, ... are chosen to be at least 3 for numerical reasons. This is in accordance this the choice of the buffer regions for the grid regions (see 5.3).

Bibliography

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

Index