# poisson.h Documentation

Haochen(Langford) Huang

November 8, 2024

version:1.0

## 1 Introduction and Background

**poisson.h** serves as toolbox which provides functions to construct V-cycle iteration solver for implicit equations. A specific one for solving poisson equation is constructed within the headfile as an example. We shall first introduce the constructing toolbox.

Assuming the governing equation can be written as

$$f(x) = y \tag{1}$$

where $y$ is the known variable, $x$ is the desired variable and $f$ represents linear operator that satisfies

$$f(x_a + x_b) = f(x_a) + f(x_b) \tag{2}$$

Now consider the discrete form of operator $\hat{f}$ which takes all desired variable from every cell (suppose the total number of cell is $n$) to express the local known variable $y_i$ then yields the implicit equation group

$$\hat{f}(x_1, x_2, x_3, \cdots, x_n) = y_i \quad i = 1, 2, \cdot, n \tag{3}$$

which can be solved by indirective iterative method such as Jacobi method, G-S method[**moin2010fundamentals**] *etc.* Moreover, constraints **??** provide another perspective to construct equation group. Use $x_1^e, x_2^e, \cdots, x_n^e$ to denote exact solution of equation **??** and $x_1^k, x_2^k, \cdots, x_n^k$ to represent result of $k$th iteration. Following equation **??** we have

$$\hat{f}(\delta x_1^k, \delta x_2^k, \cdots, \delta x_n^k) = RES^k \tag{4}$$

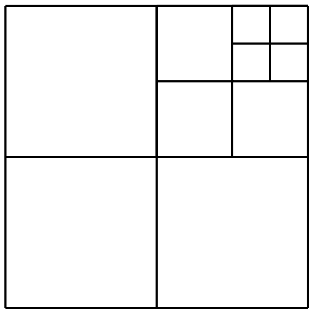
where $\delta x_i^k = x_i^e - x_i^k$, $RES^k = \hat{f}(x_1^e, x_2^e, \cdots, x_n^e) - \hat{f}(x_1^k, x_2^k, \cdots, x_n^k)$. The criterion of solution then becomes $|RES^k|_\infty < \epsilon$ where $\epsilon$ is a setting tolerance.

There are many techniques to accelerate the convergence of iteration, and multigrid method[**wesseling1995introducti** may be one of the most famous which employs iterations on every layer of the mesh to reduce the residual of corresponding wavenumber. A similar methodology is applied by quadtree/octree in Basilisk. Take quadtree as an exapmle. Consider tree architecture in Fig.**??**, the actual calculating rules for this problem is shown in **??** where ● represents leaf cells (the finest cell at this area and is not divided by higher level) and the value it carrying is the the final value shown in the result called active value. ● represents ghost cell served as boundary condition whose value is computed by bilinear interpolation. Finally ● represents value carried by parent cell. The parent cell, indicated by its name, will be divided into 4(8) children cells in finer layer (level in Basilisk).[**van2018towards**]

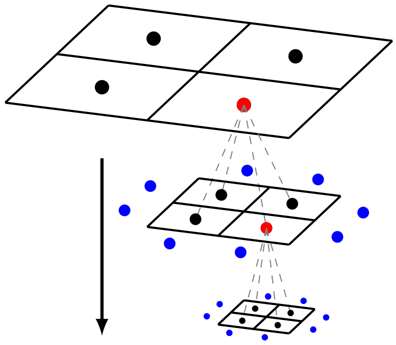
A single round of iteration is accomplished by two procedures. First, from highest level to lowest one, assign residual to each cell of current level which form the *R.H.S.* of equation **??**. Second, starting from lowest level to the highest, obtain the result after few iterating (by Jacobi method or GS method) on current level and use it to compute initial value on next level. We shall first dive into second procedure which is more sophisticated.

Calculations happens at every level shown in Fig.**??**, when it comes to higher level the boundary condition is first set and then undergoes the iteration on cells at same level instead of whole domain. Moreover, the initial value on each level is obtained by prolongation (bilinear mostly) from previous mesh level.

In order to facilitate equation **??** we also need residual, which only exists at leaf cell, of every cell at each level. This procedure is achieve by restricting[**popinet2015quadtree**] (averaging mostly) value on 4(8) children cells, which is much simpler compared to bilinear that use in previous description.



(a) 2D quadtree example.



(b) Calculation for each level.

Figure 1: Quadtree example. Arrow in (b) indicates calculating sequence.

After introducing the mesh architecture, we shall now step a little further to see the solver structure provided by 'poisson.h' and to perceive the overall workflow. FIG.**??** displays whole system as well as its workflow. As can be seen from the sketch, the whole solver consists of four functions, **mg_solve**, **mg_cycle**, **relax** and **residual**. Their nesting relating is shown by corresponding position, *e.g.* **relax** is inside **mg_cycle** while **residual** and **mg_cycle** locate inside **mg_solve** indicates that **relax** is called by **mg_cycle** and **mg_cycle** along with **residual** are directly called by **mg_solve**. Detailed workflow is also presented, after inputting $\mathbf{x}^0, \mathbf{y}$ before the residual actually meet the tolerance $\epsilon$, **mg_solve** plays as a manager to make rest functions coordinate, $\mathbf{x}^k$ is conveyed between **mg_cycle** and **residual** to renew. Number behind each step represents the order within the loop. $\mathbf{x}^k, \mathbf{y}$ is first sent to residual to compute residual $RES^k$ which served as parameter in **mg_cycle**. $\mathbf{x}^k$ and $n$ are also taken into **mg_cycle** where $n$ controls iteration number on each mesh level. $\mathbf{x}^{k+1}$ is obtained by first solving equation **??** for $\delta\mathbf{x}^k$ then execute update

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \delta\mathbf{x}^{k+1} \tag{5}$$

Loop will break out either residual satisfies tolerance constraint or number of round exceed setting threshold. Readers may notice there is no parameters conveyed within **mg_cycle**, this is because relationship between **relax** and **mg_cycle** cannot be simply abstracted as 'linear' as depicted in this figure. Structure inside **mg_cycle** is demonstrate in Fig.**??** as described before residual is assigned to each level then relax is called at each level multiple times updating $\delta\mathbf{x}^k$ in the form (condition varies according to iteration method)

$$\delta x_i^{k+1} = F(\delta x_1^k, \delta x_2^k, \cdots, \delta x_{i-1}^k, \delta x_{i+1}^k, \cdots, \delta x_n^k, RES^k) \tag{6}$$

Back to **mg_solve**, readers may notice from Fig.**??** that all the function within, including **mg_solve** itself, are divided into three layer by dashed line and each layer is named by Roman number from top to bottom. Higher the layer, more irreplaceable the function is. Therefore, functions at III can be changed or altered based on one's purpose. In another word, users can choose their own **relax** and **residual** based on equation they cope with. The governing equation for **poisson.h** is

$$L(a) = \nabla \cdot (\alpha \nabla a) + \lambda a = b \tag{7}$$

where $L$ is a linear operator. Based on above discussion such equation can be solved by multigrid solver only if one constructs appropriate **relax** and **residual** function. Another example is referred to headfile **viscosity.h** where same solver construction is used for totally different linear equation.

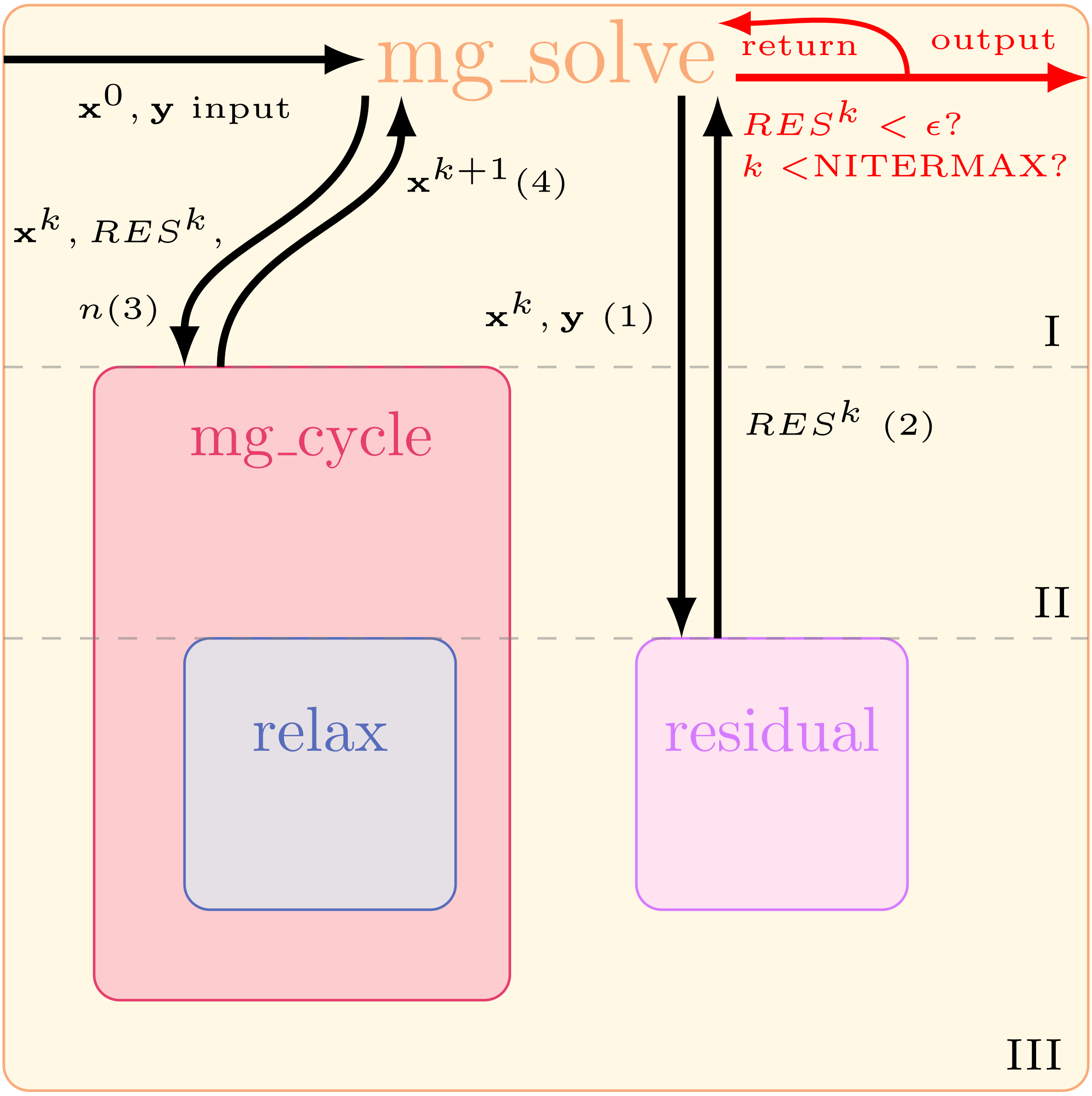



Figure 2: Architecture of the solver. Nested relationship of functions is indicated by box containing relationship *e.g.* **mg_cycle** contains **relax** but not **residual** indicates that **relax** is called in **mg_cycle** while **residual** is called in **mg_solve**.

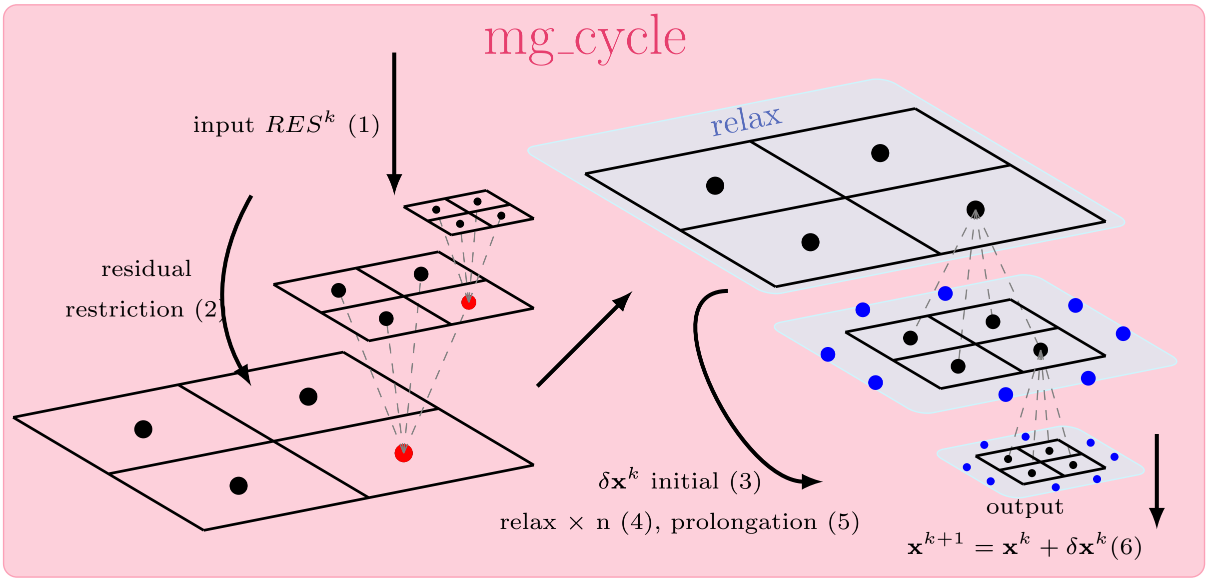



Figure 3: Combination between **mg_cycle** and **relax**. The 'round' of iteration described before is also demonstrated in a detailed way. **relax** herein is embed into every level of the mesh and is executed several times (depends on parameter $n$) on each level to accelerate convergence.

# 2 Multigrid Solver

As indicated in section **??**, we here first introduce general structure of multigrid solver which consists of **mg_solve** and **mg_cycle**.

## 2.1 mg_cycle

### 2.1.1 Parameters

| Name | Data type | Status | Option/Default | Representation (before/after) |
|---|---|---|---|---|
| $a$ | scalar* | update | compulsory | $\mathbf{x}^k/\mathbf{x}^{k+1}$ |
| $res$ | scalar* | unchanged | compulsory | $\delta\mathbf{x}^k$ |
| $da$ | scalar* | unchanged | compulsory | $\rho^{n+\frac{1}{2}}$ |
| $relax$ | void* | unchanged | compulsory | **relax** |
| $data$ | void* | unchanged | compulsory | **Poisson** (struct defined below) |
| $nrelax$ | int | unchanged | compulsory | $n$ |
| $minlevel$ | int | unchanged | compulsory | $minlevel$ |
| $maxlevel$ | int | unchanged | compulsory | $maxlevel$ |

### 2.1.2 Worth Mentioning Details

As described in Sec.**??** and Fig.**??** function **mg_cycle** serves as a subcomponent to update the result. Details of such function have been explored before and shall not be repeated here.

### 2.1.3 Program Workflow

**Starting Point**
**input**:
$a = x^k$ $res = RES^k$ $da$=$\delta x^k$
$relax$=relax function
$data$ = Poisson structure
$nrelax$=times relax function applied to each level
$minlevel$=minimum of mesh level
$maxlevel$=maximum of mesh level

**RES restriction**
To restrict residual from finer mesh to coarser mesh.
**Entering of iterative**
Iterate starting form lowest (coarsest mesh) level.

**Initial for Iteration**
Set initial guess for each level. If is on the coarsest level, the initial guess is set to 0 otherwise the initial guess comes from bilinear interpolation of coarser level.

**Relax on each level**
Boundary condition (value of blue point in figure**??**) is first calculated and assigned at certain level. After setting the initial value and boundary condition, relaxation is then conduct on each level by employing **relax**. At the same time, $\delta\mathbf{x}$ is computed and restored in *da*.

**Final update**
$\mathbf{x}^{k+1} = \mathbf{x}^k + \delta\mathbf{x}$

## 2.2 mg_solve

### 2.2.1 Parameters

| Name | Data type | Status | Option/Default | Representation (before/after) |
|---|---|---|---|---|
| *a* | scalar* | update | compulsory | $\mathbf{x}^0/\mathbf{x}^{final}$ |
| *b* | scalar* | unchanged | compulsory | $\mathbf{y}$ |
| *residual* | scalar* | unchanged | compulsory | **residual** |
| *relax* | void* | unchanged | compulsory | **relax** |
| *data* | void* | unchanged | compulsory | **Poisson** (struct defined below) |
| *nrelax* | int | unchanged | optional/4 | $n$ |
| *res* | scalar* | unchanged | compulsory | $RES$ |
| *minlevel* | int | unchanged | optional/0 | $minlevel$ |
| *tolerance* | double | unchanged | optional/$10^{-3}$ | $\epsilon$ |

### 2.2.2 Worth Mentioning Details

Two optional components **residual** and **relax** are input as void pointer in this function. The return of this function is of self-defined data structure 'mgstats' which indeed contains information of the whole multigrid circle. The information includes *i*: the total number **mg_cycle** is employed inside **mg_solve**, *resb* and *resa*: maximum residual before/after the cycle.

### 2.2.3 Program Workflow

**Initial Settings**
*NITERMAX*=100 and *NITERMIN*=1 is the maximum and minimum times **mg_cycle** employed by **mg_solve** Self-defined structure 'mgstats' serves as return value for all multigrid solver. It contains basic information about this circle.

↓

**Input**
*a* = $\mathbf{x}^0$, *b*=$\mathbf{y}$, *nrelax* is the one used in **mg_cycle** *minlevel* is the coarsest mesh level user would like to iterate. *tolerance*=$\epsilon$, *relax* and *residual* are two pointers points to related function **residual** and **relax**.

↓

**Pointer Preparation**
Note that 'list_clone' directly copy the data to the pointer address instead of point to the original address. Therefore *da* and *a* are two pointers with same data but different address while *pre* point to *res*. If *res* points NULL, *pres* will directly points to the same address. Note the change of *res* hereinafter shall not influence the NULL address of *pres*.

**Boundary and initial information settings**
Homogeneous boundary conditions are applied to $\delta \mathbf{x}$. Return value *s* is initialized. *s.sum* is the summation of R.H.S. of equation **??**.

**Residual before mg_cycle**
Optional function **residual** is called to calculate the residual before **mg_cycle**.

**Entering iteration**
**mg_cycle** is called iteratively until the residual meets tolerance or iteration exceeds ***NITERMAX*** (default by 100)

**Alteration of iteration inside mg_cycle**
Based on the speed of convergence, times of **relax** applied on each level will inside **mg_cycle** be altered by changing *s.nrelax* (*nrelax* in **mg_cycle**).
**End of iteration**

**Warning and others**
Once fail to converge, warning will be output through standard output. Extra allocated memory (*da* and *res* if not by default) then is freed. Since *res* point to a specific address no matter the input, *pres* serves as condition.

## 2.3 Poisson-Helmholtz solver

Now we shall introduce the application for Poisson-Helmholtz equation of multigrid solver.

## 2.4 Poisson structure

### 2.4.1 Parameters

| Name | Data type | Status | Option/Default | Representation (before/after) |
|---|---|---|---|---|
| *a* | scalar | unchanged | compulsory | $\mathbf{a}^0$ |
| *b* | scalar | unchanged | compulsory | $\mathbf{b}$ |
| *alpha* | face vector | unchanged | optional/1 | $\alpha$ |
| *lambda* | scalar | unchanged | optional/0 | $\lambda$ |
| *tolerance* | double | unchanged | optional/$10^{-4}$ | $\epsilon$ |
| *nrelax* | int | unchanged | optional/4 | $n$ |
| *minlevel* | int | unchanged | optional/1 | $n$ |
| *res* | scalar* | unchanged | optional/NULL | $RES$ |

### 2.4.2 Worth Mentioning Details

Such structure is built for conveying $\alpha$ and $\lambda$ to solve equation **??**. *res* serves as convergence monitor. anything concerned with embed is under construction.

### 2.4.3 Workflow

## 2.5 relax

### 2.5.1 Parameters

| Name | Data type | Status | Option/Default | Representation (before/after) |
|---|---|---|---|---|
| *al* | scalar* | unchanged | compulsory | $\mathbf{a}^0$ |
| *bl* | scalar* | unchanged | compulsory | $\mathbf{b}$ |
| *l* | int | unchanged | compulsory | current level |
| *data* | void* | unchanged | compulsory | Poisson data structure |

### 2.5.2 Worth Mentioning Details

Consider the discrete form of equation **??** using 5-points Laplacian operator in 2D

$$\frac{\alpha_{i+\frac{1}{2},j}\frac{a_{i+1,j}-a_{i,j}}{\Delta} - \alpha_{i-\frac{1}{2},j}\frac{a_{i,j}-a_{i-1,j}}{\Delta}}{\Delta} + \frac{\alpha_{i,j+\frac{1}{2}}\frac{a_{i,j+1}-a_{i,j}}{\Delta} - \alpha_{i,j-\frac{1}{2}}\frac{a_{i,j}-a_{i,j-1}}{\Delta}}{\Delta} + \lambda a_{i,j} = b \tag{8}$$

and arrange it into forms of equation **??** hence the expression of relaxation

$$a_{i,j} = \frac{\alpha_{i+\frac{1}{2},j}a_{i+1,j} + \alpha_{i-\frac{1}{2},j}a_{i-1,j} + \alpha_{i+\frac{1}{2},j+\frac{1}{2}}a_{i,j+1} + \alpha_{i,j-\frac{1}{2}}a_{i,j-1} - b\Delta^2}{\alpha_{i+\frac{1}{2},j} + \alpha_{i-\frac{1}{2},j} + \alpha_{i,j+\frac{1}{2}} + \alpha_{i,j-\frac{1}{2}} - \lambda\Delta^2} \tag{9}$$

By turning on/off macro 'JACOBI', user can choose type of iteration which employed in the solver. Once turning on, solver will implement Jacobi iteration with relaxation factor of $\frac{2}{3}$ i.e. $\mathbf{a}^{n+1} = (\mathbf{a}^n + 2\mathbf{a}')$. Where $\mathbf{a}'$ is L.H.S. of equation **??** and will be stored in independent address named *c*. If such macro is turned off, *c* will be defined as additional pointer pointed to *a* and Gauss-Seidel method will be employed. Which indicates equation **??** is altered as

$$\delta x_i^{k+1} = F(\delta x_1^{k+1}, \delta x_2^{k+1}, \cdots, \delta x_{i-1}^{k+1}, \delta x_{i+1}^{k}, \cdots, \delta x_n^{k}, RES^k) \tag{10}$$

and there is no use of relaxation factor.

### 2.5.3 Workflow

**Initial settings and Input**
*a*=$\mathbf{a}^0$, *b*=$\mathbf{b}$,*alpha*=$\alpha$,*lambda*=$\lambda$
Note $\alpha$ and $\lambda$ is conveyed through data structure 'Poisson' *p*

$\downarrow$

**Macro switch of Jacobi**
If macro switch 'JACOBI' is active new independent scalar *c* is defined to store result of equation **??**. Otherwise *c* will directly point to same address as *a*.

$\downarrow$

**Relaxation Function**
Implementation of equation **??**.
$n = \alpha_{i+\frac{1}{2},j} a_{i+1,j} + \alpha_{i-\frac{1}{2},j} a_{i-1,j} + \alpha_{i+\frac{1}{2},j+\frac{1}{2}} a_{i,j+1} + \alpha_{i,j-\frac{1}{2}} a_{i,j-1} - b\Delta^2$
$d = \alpha_{i+\frac{1}{2},j} + \alpha_{i-\frac{1}{2},j} + \alpha_{i,j+\frac{1}{2}} + \alpha_{i,j-\frac{1}{2}} - \lambda\Delta^2$

$\downarrow$

**Embed Flux**
TBD

$\downarrow$

**Jacobi Implementation**
L.H.S. of equation **??** is computed and stored in *c*. Once macro JACOBI is active, relaxation factor of $\frac{2}{3}$ is employed.

## 2.6 residual

### 2.6.1 Parameters

| Name | Data type | Status | Option/Default | Representation (before/after) |
|---|---|---|---|---|
| *al* | scalar* | unchanged | compulsory | $\mathbf{a}^k$ |
| *bl* | scalar* | unchanged | compulsory | $\mathbf{b}$ |
| *resl* | scalar* | unchanged | compulsory | $RES^k$ |
| *l* | int | unchanged | compulsory | current level |
| *data* | void* | unchanged | compulsory | Poisson data structure |

### 2.6.2 Worth Mentioning Details

The function aim to solve

$$RES^k = b_{i,j} - \frac{\alpha_{i+\frac{1}{2},j}\frac{a^k_{i+1,j}-a^k_{i,j}}{\Delta} - \alpha_{i-\frac{1}{2},j}\frac{a^k_{i,j}-a^k_{i-1,j}}{\Delta}}{\Delta} - \frac{\alpha_{i,j+\frac{1}{2}}\frac{a^k_{i,j+1}-a^k_{i,j}}{\Delta} - \alpha_{i,j-\frac{1}{2}}\frac{a^k_{i,j}-a^k_{i,j-1}}{\Delta}}{\Delta} - \lambda a^k_{i,j} \quad (11)$$

however careful consideration is needed when it comes to tree grid. Same problem has been fully discussed in section 2.3.2 of 'viscosity.h Documentation' and will not be repeated here.

The maximum residual is returned by **residual** and serves as criterion in **mg_solve** as described in section **??**.

### 2.6.3 Workflow

**Initial settings and Input**
*a*=$\mathbf{a}^k$, *b*=**b**,*alpha*=$\alpha$,*lambda*=$\lambda$
Same as **relax** $\alpha$ and $\lambda$ is conveyed through data structure 'Poisson' *p*. *res* is currently empty and *maxres* is the returned value that contains maximum residual.

**Conservative discretisation**
Implementation of equation **??**. Please check 'viscousity.h Documentation' for more details.

**Embed Flux**
TBD

**Maximum Residual**
*maxres*=$(RES)_{max}$ and will be returned eventually.

**Implementation for Nontree Grid**
Implementation for nontree grid, same scheme has only 1st order on tree grid.

**Final Return**
Return maximum of residual.