FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
INSTITUT FÜR INFORMATIK (MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG)

Lehrstuhl für Informatik 10 (Systemsimulation)

Runtime-Adaptivity Techniques for Multigrid Methods

Volker Daum

Studienarbeit

Runtime-Adaptivity Techniques for Multigrid Methods

Volker Daum

Studienarbeit

Aufgabensteller:	Prof. Dr. U. Rüde
Betreuer:	Dr. Ing. G. Horton
	Dipl.-Inf. M. Kowarschik
Bearbeitungszeitraum:	01.11.2000 – 31.08.2001

Erklärung:

Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 4. Oktober 2001 ....

Contents

1 Introduction

The numerical treatment of differential equations, which arise in physical problems often involves the necessity to solve large linear systems. One class of methods which was tailored to solve such large linear systems are the multigrid methods. While they are very efficient and can achieve asymptotically optimal convergence they often have to be modified to fit the problem at hand. To further improve the performance of these methods and to at least partially overcome the need to customize the algorithm for a specific problem, we will try to allow it to adapt at runtime to our specific problem. The work done here is largely based on the work of Prof. Ulrich Rüde [4] [3].

1.1 Multigrid Methods

For the work we want to do on this topic it is sufficient to consider a very simple multigrid method, namely a geometric multigrid using a correction scheme and either just plain V-cycles or FMG. We will work in 2-D on uniform meshes and discretize with finite differences.

First I will in short explain how this basic algorithm works (see also [1]). Geometric multigrid was an effort to overcome the limitations of a simple iterative algorithm like the Gauss-Seidel method. The Gauss-Seidel iterates over the grid and successively sets the residual at each gridpoint to zero, which naturally changes again the residuals of the neighbouring points. It is quite obvious that the Gauss-Seidel only has a “very local view” of the grid and therefore is only good at correcting very highly oscillatory errors. Errors which have a very low frequency are locally seen almost non-existent and therefore are only very slowly eliminated. Thus the Gauss-Seidel makes the error “smoother”. The basic idea of multigrid is that when such a low frequent error is represented on a coarser grid it appears much more high frequent to the Gauss-Seidel and is therefore faster eliminated. This then yields the traditional V-cycle (here for a correction scheme): We first apply a few Gauss-Seidel steps (usually 1 or 2) to smooth the error then we calculate the residual and restrict it to a coarser grid (full weighting). On the coarser grid we solve the residual equation

$$Ae = r \tag{1.1.1}$$

either by again recursively applying this scheme or, if we are already on a very coarse grid, by a direct method. The error is then interpolated and used to correct the solution on the fine grid. Finally we again do a few Gauss-Seidel steps to correct errors originating from the interpolation.

Depending on the problem we have to deal with we should see a convergence rate of a bout 0.1 per V-cycle. Instead of just doing several of these V-cycles starting from the finest grid, there is another cycling scheme called the Full MultiGrid (FMG). When we employ FMG then we start with solving the problem on the coarsest grid. This solution is then interpolated to the next finer grid. On this finer grid we use our previous solution as an initial guess to a V-cycle (or even several V-cycles). The result of this is again interpolated to the next finer grid, and so on until we reach the finest grid.

1.2 Adaptivity

It was already mentioned that often multigrid has to be in one way or another be adapted to the specific problem at hand. This is done in many different ways. Some of these like the introduction of certain multipliers of the matrix coefficients of a problem or a certain preconditioning of the matrix we have to deal with are not relevant for this work as they are too problem specific and could not be done automatically by a program. Another possibility is to do a grid refinement only in areas where it is promising that this will yield substantially better results. The problem with an adaptivity of this kind is to decide which areas are "promising" and what "promising" means in this case. A third idea would be to restrict the smoothing to areas where additional smoothing seems to be especially necessary. And in fact this is the first point of the algorithm presented here.

2 Modelproblems

To present and analyze the different techniques I will introduce in this work we will need some model problems. The two model problems I will use for this both are discretizations of a simple Poisson equation and have a singularity of some kind. Singularities are usually quite a problem for numerical methods and should therefore be good to illustrate how the algorithm works.

2.1 two Peaks

The first model problem is not a too realistic one, but in exchange it should show very clearly how the algorithm works. The equation is a Poisson equation on the unit square with two singularities in the right hand side:

$$-\Delta u = f(x, y), \quad 0 < x < 1, \quad 0 < y < 1 \tag{2.1.1}$$

$$f(x, y) = \begin{cases} 100 & \text{if} \quad x = 0.5 \wedge y = 0.5 \\ 200 & \text{if} \quad x = 0.75 \wedge y = 0.25 \\ 0 & \text{else} \end{cases} \tag{2.1.2}$$

This is discretized by a five point stencil:

$$\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix} \tag{2.1.3}$$

Additionally we have dirichlet boundary conditions of the type

$$u = 0 \text{ on } \partial\Omega \tag{2.1.4}$$

The height of the two peaks of a numerical solution to this problem will greatly depend on the discretization. Figure 2.1.1 shows a solution to this problem generated with the above described standard multigrid on a 128×128 grid.

2.2 Four corner problem

The four corner problem is a somewhat more realistic problem. The setup is, that you have four plates. Two of these plates are isolating (with a coefficient of ϵ) and the other two conduct warmth very well (with a coefficient of 1). Then these plates are put together, so

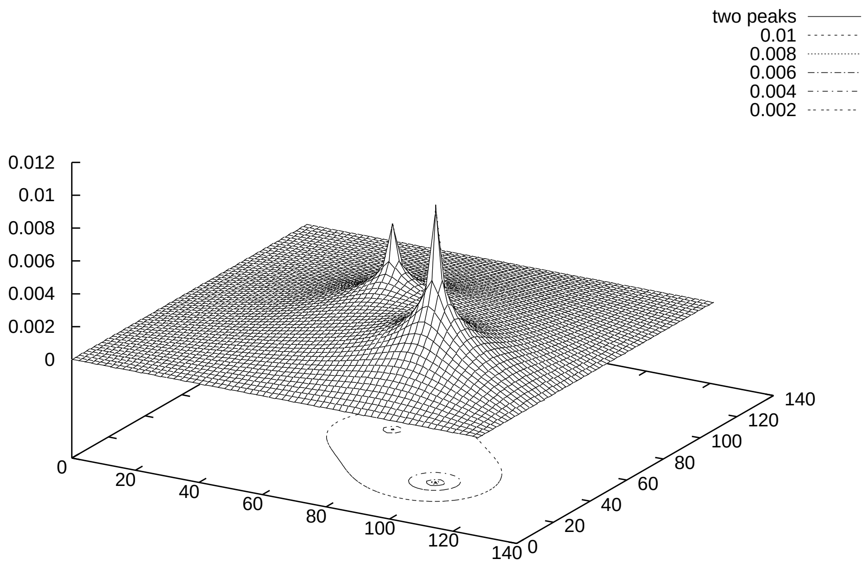

Figure 2.1.1: Modelproblem 1, Poisson equation with two peaks on a 128×128 grid.

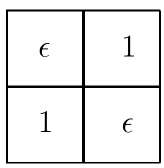

Figure 2.2.1: Schematic picture of the four corner problem.

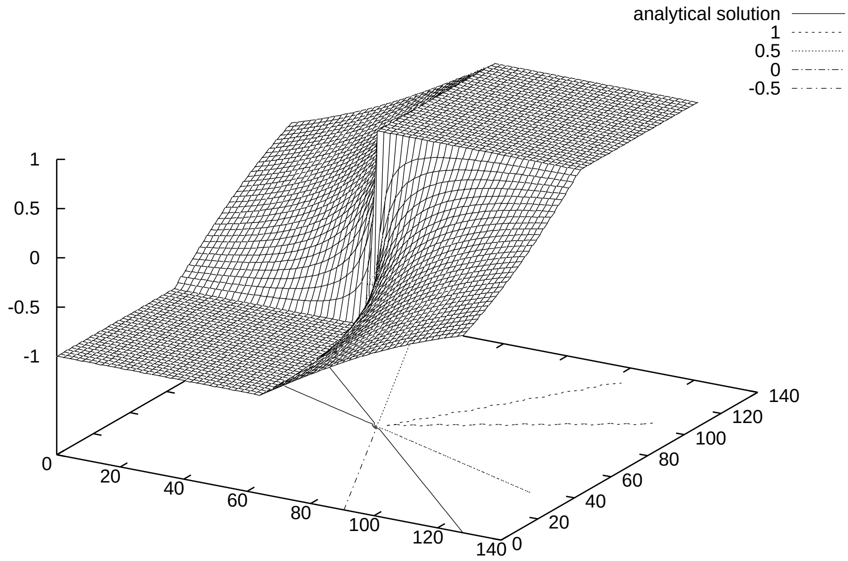

Figure 2.2.2: Modelproblem 2, analytical solution to the four corner problem with $\epsilon = 10^{-6}$

that the isolating plates and the conducting plates only touch in the centre of the domain. (see Figure 2.2.1) In literature problems of this kind are usually called intersecting interfaces [2]

Mathematically this is again described by the Poisson equation for the heat distribution

$$-\nabla D(x,y)\nabla u = f(x,y), \quad -1 < x < 1, \quad -1 < y < 1 \tag{2.2.1}$$

$$D(x,y) = \begin{cases} 1 & \text{if} \quad x < 0 \wedge y < 0 \quad \text{or} \quad x > 0 \wedge y > 0 \\ \epsilon & \text{if} \quad x > 0 \wedge y < 0 \quad \text{or} \quad x > 0 \wedge y < 0 \end{cases} \tag{2.2.2}$$

As right hand side we set

$$f(x,y) = 0 \tag{2.2.3}$$

It can be shown that (in polar coordinates r and Φ)

$$u_\alpha(r,\Phi) = r^\alpha \begin{cases} \cos(\alpha(\Phi - \pi/4)) & \text{in the northeast quadrant} \\ -\beta\sin(\alpha(\Phi - 3\pi/4)) & \text{in the northwest quadrant} \\ -\cos(\alpha(\Phi - 5\pi/4)) & \text{in the southwest quadrant} \\ \beta\sin(\alpha(\Phi - 7\pi/4)) & \text{in the southeast quadrant} \end{cases} \\ \text{where} \\ 0 < \alpha < 1, \quad \beta = \cot(\alpha\pi/4), \quad \epsilon = \tan^2(\alpha\pi/4) \tag{2.2.4}$$

is an analytical solution to this equation if the boundary conditions are set to

$$u = u_\alpha \text{ on } \partial\Omega \tag{2.2.5}$$

Figure 2.2.2 shows a plot of this analytical solution with $\epsilon = 10^{-6}$.

For the numerical solution of this problem we first need a discretization. Because D is piecewise constant the equation reduces to the Poisson equation on the four quadrants and

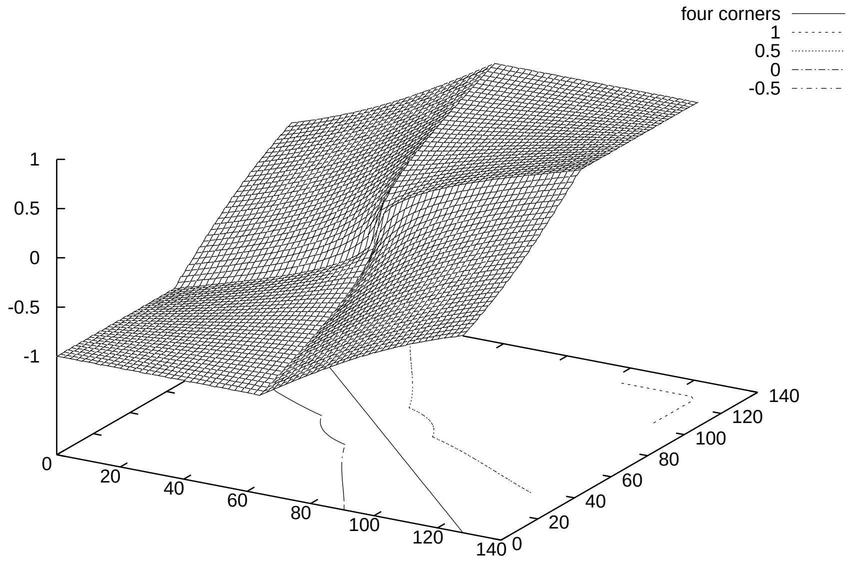

Figure 2.2.3: Modelproblem 2, numerical solution to the four corner problem with $\epsilon = 10^{-6}$ on a 128×128 grid

can be discretized with a five point stencil, which is in the southwest and northeast quadrants just the same stencil as in modelproblem 1. In the other two quadrants this stencil has to be multiplied by ϵ so that we get:

$$\underset{\text{southwest and northeast}}{\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}} \quad \underset{\text{northwest and southeast}}{\epsilon \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}} \tag{2.2.6}$$

On the boundaries between the plates we just average and get

$$\underset{\text{between southwest and northwest}}{\begin{bmatrix} & -\epsilon & \\ -(1+\epsilon)/2 & 2(1+\epsilon) & -(1+\epsilon)/2 \\ & -1 & \end{bmatrix}} \tag{2.2.7}$$

This is analogously done on the other edges. In the centre of the domain where all four plates meet we get

$$\underset{\text{in the centre of the domain}}{\begin{bmatrix} & -(1+\epsilon)/2 & \\ -(1+\epsilon)/2 & 2(1+\epsilon) & -(1+\epsilon)/2 \\ & -(1+\epsilon)/2 & \end{bmatrix}} \tag{2.2.8}$$

In figure 2.2.3 we see the result of the standard multigrid on a 128×128 grid. Just as in the first modelproblem we can observe that the result does not look too good, due to the singularity in the centre of the domain.

3 Smoother

Now that we have introduced the basics and the problems on which we want to test our methods, it is about time that we really get started. Multigrid algorithms spend the most computational time in their smoothing routine (e.g. the Gauss-Seidel), so it is quite natural that we will start there to introduce some adaptivity. For this we will consider the Gauss-Seidel isolated from the multigrid for the moment.

3.1 Idea

As we have already stated in the introduction, the Gauss-Seidel only smoothes errors of a certain frequency on one grid. But in most cases these frequencies are not evenly distributed on the domain. Thus in a few places you do not need to do much relaxation and in other places you would be happy if you could apply a few extra iterations to optimize the convergence rate. This is exactly what we want to achieve here.

To decide at run time where we want to do these extra relaxation steps the algorithm will keep an “active set” of points. As we are viewing the Gauss-Seidel isolated from the multigrid we will have all the points on the grid in this active set at the beginning. When relaxing a point it is first calculated how much this point would change. If the change is below a certain tolerance then this change is just discarded and the point is removed from the active set. If the change is above the tolerance then the change is applied. This means that the residual at this point is set to zero, so that we can again safely remove that point from the active set. But changing the value of a point also means that we change the residual of all points coupled to this one. Therefore we have to add all the neighbouring points to the active set. Like this the algorithm keeps iterating until the active set is emptied. When we are finished this means that at each point the residual is

$$
\begin{aligned}
|u_{ij}^{(new)} - u_{ij}| &< \tau \\
\left|\frac{-a_{i+1,j}u_{i+1,j} - a_{i-1,j}u_{i-1,j} - a_{i,j+1}u_{i,j+1} - a_{i,j-1}u_{i,j-1} + h^2 f_{i,j}}{a_{i,j}}\right| &< \tau \\
\left|\frac{-a_{i+1,j}u_{i+1,j} - a_{i-1,j}u_{i-1,j} - a_{i,j+1}u_{i,j+1} - a_{i,j-1}u_{i,j-1}}{h^2} + f_{i,j}\right| &< \frac{|a_{i,j}|}{h^2}\tau \\
|r_{i,j}| &< \frac{|a_{i,j}|}{h^2}\tau
\end{aligned}
\tag{3.1.1}
$$

with τ being the tolerance, $a_{i,j}$ the matrix coefficients and $r_{i,j}$ the residual in the point (i,j). This means that after a run of our modified Gauss-Seidel we can guarantee a certain “smoothness” in every point and that we can steer the quality of the smoothness by the tolerance.

3.2 Implementation of the active set

How well this adaptive algorithm performs depends quite a bit on the order in which the points are fetched from the active set. It would be ideal if we could always get the point from the active set, which has the highest residual, but to do this would mean that we really get a huge overhead, so we settle with simpler implementations of the active set.

For our algorithm the active set has to be able to perform these basic operations:

- insert a point into the active set (has to check whether the point is already in the set)
- remove and return a more or less ”random” member of the set

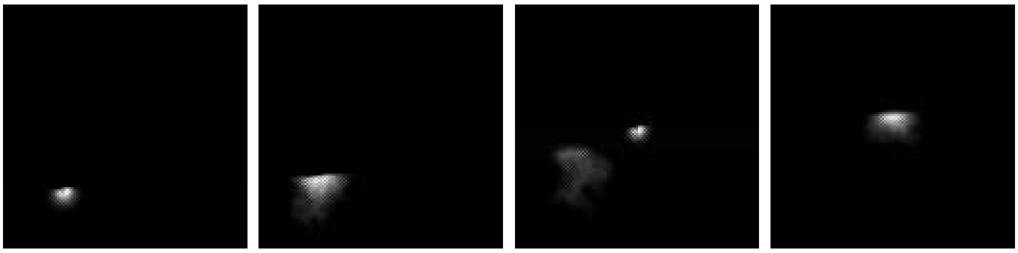

Figure 3.3.1: Activity of the Gauss-Seidel applied to peaks-problem (section 2.1) with a tolerance of $0.5E^{-6}$ and the active set implemented as a stack; 20000 point-iterations per picture; from left to right: Iterations 1-20000, iterations 200001-220000, iterations 420001-440000 and iterations 500001-520000; All in all 548532 point-iterations were needed.

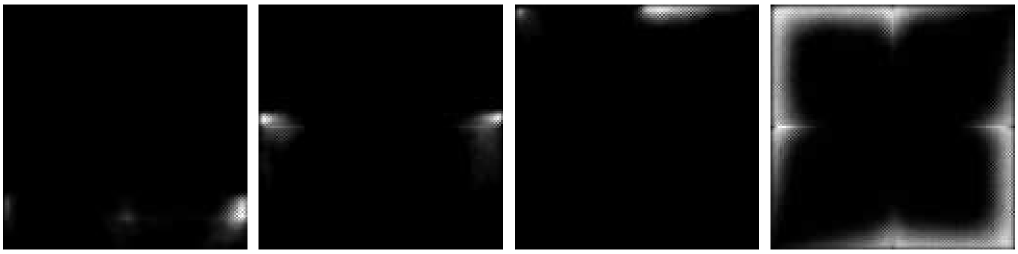

Figure 3.3.2: Activity of the Gauss-Seidel applied to four-corner problem (section 2.2) with a tolerance of $0.5E^{-3}$ and the active set implemented as a stack; 40000 point-iterations per picture; from left to right: Iterations 100001-140000, iterations 2000001-240000, iterations 400001-440000 and a summary over the complete run; all in all 837492 point-iterations were needed.

- check whether the set is empty

In all of the implementations I have done I used a "double strategy": To check whether an element is already in the set a bit list is best suited, as this allows to do this check with a complexity of $\mathcal{O}(1)$. But to determine which point should be returned on access, different strategies were used. To be able to analyze the behaviour of these strategies I have generated pictures which show the activity of the algorithm. The white areas of the figures 3.3.1 through 3.5.2 denote a high activity, dark areas a low activity.

3.3 Stack

One of the simplest ideas to implement this is to organize the points in a stack. In the implementation I have done all the needed memory is allocated in the setup phase of the algorithm, so that we do not have to deal with memory allocation and deallocation during the Gauss-Seidel anymore, thus reducing overhead.

Let us now take a look at the pictures in figure 3.3.1 that show the activity of the algorithm on the first model problem, with the two peaks (Section 2.1). The first thing one can observe is that the algorithm seems to "spread" a lot and that it first sticks to iterate around the first peak before, after about 42000 iterations, it starts to work on the second peak. This spreading does not look that bad in the pictures 3.3.2 of the second model problem (Section 2.2), but it is still visible. The spreading effect is not too surprising if one considers, how the way the

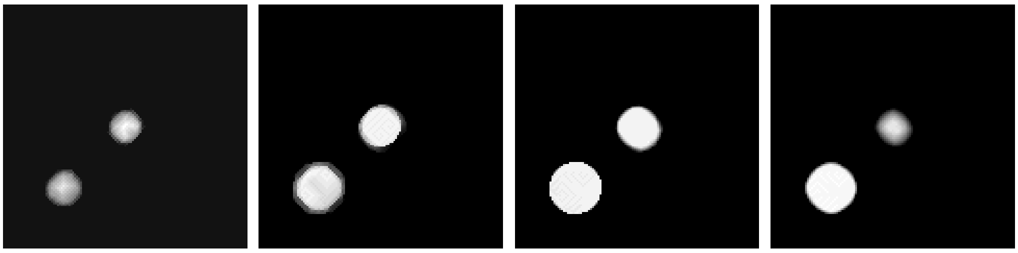

Figure 3.4.1: Activity of the Gauss-Seidel applied to model problem 1 (section 2.1) with a tolerance of $0.5E^{-6}$ and the active set implemented as a FIFO; 20000 point-iterations per picture; pictures show iterations 1-80000 from left to right. All in all 90931 point-iterations were needed.

stack works will affect the Gauss-Seidel. In a stack the last value that was inserted into it, is also the first one that will be removed. Therefore when we relax a point all its neighbours are pushed onto the stack. In the next step of the iteration one of these neighbours will be removed and relaxed. This means that the algorithm will keep correcting the errors which were introduced be the previous correction. For example if the ordering in which the points are inserted to the active set was clockwise beginning with the northern point. This means that the last point pushed onto the stack after a relaxation is always the western point. If a point is relaxed then all its neighbours are pushed onto the stack and the last of these, the western point is fetched for the next relaxation. This point is again relaxed, its neighbours pushed on the stack and the next western point fetched. Like this the algorithm keeps marching to the west until the residual drops below the tolerance. If this happens the algorithm will get as next point the southern point of the previous relaxation. One could think of this as the algorithm going a step back taking the southern point and then again march to the west, and so on. This behaviour is typical for stack based algorithms. In graph searches this is usually described as a depth first search. Still, when we look at the last picture of figure 3.3.2 which is a summary of all the point-relaxations needed in this example we see that the relaxation stays pretty local.

3.4 FIFO

Another possibility to implement the set is to use a FIFO. The FIFO is about as simple as a stack. In this case I implemented it as a ringbuffer of a fixed size so that no dynamic memory allocation during the run of the Gauss-Seidel is necessary. The pictures in figure 3.4.1 and 3.4.2 show that this approach looks much more like what we expect an adaptive Gauss-Seidel to do. Additionally we see that the FIFO needs only 90931 point-iterations for the peaks-problem and 391219 point-iterations for the four-corner problem, whereas the stack needed 548532 point-iterations for the peaks-problem and 837492 point-iterations for the four-corner problem. This is an improvement by a factor of 5 or 2 respectively. When the Gauss-Seidel works with the active set implemented as a FIFO it always relaxes the "oldest" point first. When the stack is used then the next point to relax is always the "newest". But why does this ordering of the point relaxations work more efficiently in our examples? If you reason intuitively, then on average the residual of a point and its neighbours which have been relaxed several times, is lower than that of a point which has been relaxed only a few times. If this assumption would not hold then the algorithm would diverge. Furthermore it is very probable

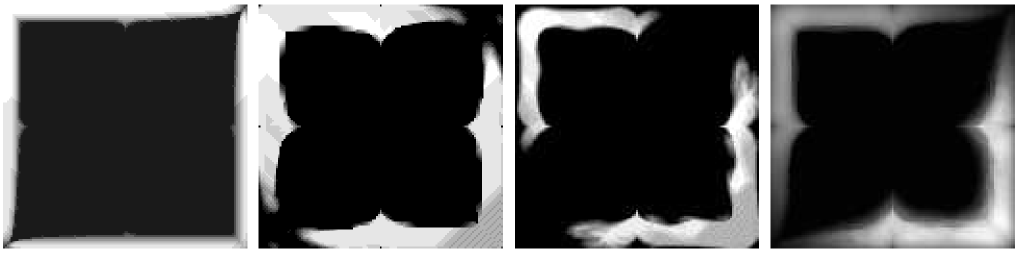

Figure 3.4.2: Activity of the Gauss-Seidel applied to model problem 1 (section 2.1) with a tolerance of $0.5E^{-3}$ and the active set implemented as a FIFO; 40000 point-iterations per picture; Iterations 1-40000, iterations 120001-160000, iterations 240001-280000 and a summary over the complete run; All in all 391219 point-iterations were needed.

that a point which has just been added to the set, has been relaxed more often than a point which has been in the set for a long time. Therefore the FIFO is in most cases a small step in the direction of the ideal algorithm that was mentioned in the beginning of this section, which always relaxes the point with the biggest residual first, and obviously this seems to pay off.

3.5 Priority queue

The priority-queue, as we chose to call it, was an attempt to do a further step in the direction of the ideal algorithm. Simplified it just uses two FIFOs and two bitlists. Accordingly we also need two tolerances. A low tolerance which corresponds to the tolerance used in the other two cases (stack and FIFO) which is the one that shall be fulfilled at the end of the run of the Gauss-Seidel, and a high tolerance, which as the name suggests, is higher than the low tolerance by a certain factor. One of the two FIFOs that we use is the really active queue. Whenever a point is relaxed and the resulting change is above the high tolerance its neighbours are inserted into this queue, the high queue. If it is only above the low tolerance than its neighbours are inserted into the other queue, the low queue. If the change is below the low tolerance then, just as before the change is discarded. The next point that should be relaxed is always taken from the high queue. When the high queue is empty then the high tolerance is reduced and all the points from the low queue are inserted into the high queue. This is done until the high tolerance is equal to the low tolerance (or alternatively until the high and the low queue are both empty). All this, inserting the points from the low queue into the high queue, removing points from the high queue and inserting points into the queue is still done in $\mathcal{O}(1)$, but all these actions certainly carry a certain overhead. Simply put all this is more or less the same as starting the Gauss-Seidel with the FIFO as active set, several times with a step by step reduced tolerance. The only difference is that we do not have to initialize each run with all points, but only with the points that remained in the low queue during the previous run. But this also means that the positive effect of the low queue will get smaller when the low tolerance is low. When we take a look at the pictures in figure 3.5.1 and 3.5.2 we see very nicely that the algorithm does more or less what is expected: During the first passes where the high tolerance is still pretty high it stays close to the areas where high activity is expected (for example the peaks) and when the high priority is step by step reduced to the low priority the areas of activity grow broader and broader. On the other

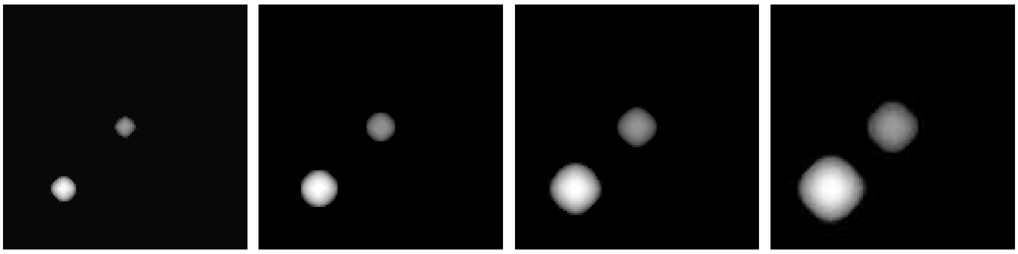

Figure 3.5.1: Activity of the Gauss-Seidel applied to model problem 1 (section 2.1) with a tolerance of $0.5E^{-6}$ and the active set implemented as priority-queue; pictures generated after each reduction of the tolerance e.g. from left to right: iterations 1-19978, iterations 19979-29419, iterations 29420-54183 and iterations 54184-117992. All in all 117992 point-iterations were needed.

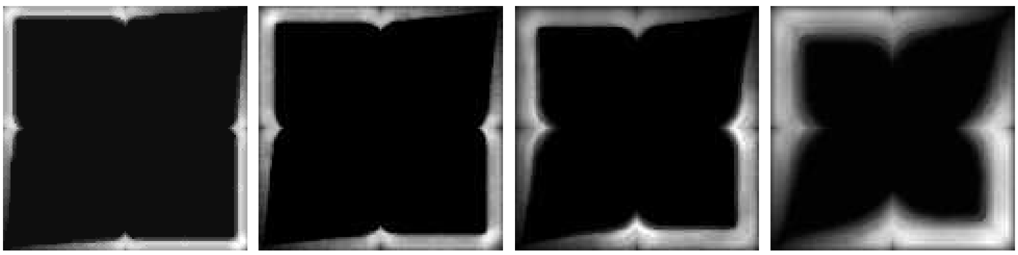

Figure 3.5.2: Activity of the Gauss-Seidel applied to model problem 1 (section 2.1) with a tolerance of $0.5E^{-3}$ and the active set implemented as priority-queue; pictures generated after each reduction of the tolerance e.g. from left to right: iterations 1-40629, iterations 40630-72933, iterations 72934-166614 and and a summary over the complete run; all in all 408935 point-iterations were needed.

hand we also have to note that the priority queue needed a few more point-iterations than the FIFO. For the generation of the pictures in figure 3.5.1 and 3.5.2 the high tolerance was halved in each step. With a little experimentation one can get better results from the priority queue by choosing the high tolerances differently. But this also shows one of the problems of this approach: How should the high tolerance be chosen and how should it be reduced. Another problem is that we do not really know anything about the residuals. We just suppose that the residuals behave according to the changes of a points neighbours. As an improvement you could for example think about taking the matrix coefficients into account for calculating whether a point should go into the high or the low queue. There are probably endless possibilities for further extensions of this idea, like using more queues, and not just two or smart algorithms for choosing the tolerances. The question is just: Is it worth it?

3.6 Summary

One last interesting point is to compare the adaptive algorithm with its different implementations of the active set to the original non-adaptive Gauss-Seidel. For this comparison I chose the two peaks problem with exactly the same tolerance ($\tau = 0.5E-6$). To compare the results the adaptive algorithm was run and when it was finished the global residual was measured. Then the standard Gauss-Seidel was applied to the problem and run until it reached the same global residual. In the following small table you can see the results:

set type	global residual	point iterations adaptive GS	standard GS
Stack	$6.199285E-03$	548532	661289
Fifo	$1.366947E-02$	90931	306451
priority queue	$8.321130E-03$	117992	499999

We see that the adaptive algorithm really performs (depending on the implementation of the active set) a lot better than the standard algorithm. But in this case we have chosen a problem which is tailored to show this result. I have run the same test on another problem: a Poisson equation with a right hand side of $f = 0$ and an initial guess of a sin-curve in the whole domain. This problem offers absolutely no chance for adaptivity. The results were even worse than expected. I expected the adaptive algorithm to need about as much point iterations as the standard algorithm, but in fact it needed (again depending on the type of active set used) 2 to 6 times as much iterations. And even more interesting was, that in this case the stack outperformed the fifo and the priority queue. So if we do not have the chance to really work adaptive then the standard algorithm actually works a lot better. But one also has to see that when all this is applied in the context of the multigrid we have the additional advantage that in most cases the adaptive Gauss-Seidel does not start with a fully initialized active set but instead already with an adaptively initialized set (see section 4.3).

Inside the multigrid additional things influence the adaptive Gauss-Seidel: The better the problem is already solved and the lower the tolerance gets the more homogeneous the residuals will look and the less chance for adaptivity in the relaxation will be left. On the other hand in the multigrid the Gauss-Seidel is run only for a few iterations and especially these first few iterations on a grid are the most promising for adaptivity. In general we probably will not (if we are just looking at the Gauss-Seidel) see as much improvement just by the adaptive relaxation as the above table shows and the fewer iterations we are doing the more the differences between the different set implementations will diminish.

To sum it up we have a stack that did not perform too well with our model problems, a fifo that performed quite well and the priority queue which could perform even a little bit better than the FIFO if it got the necessary fine tuning, but which also comes with even more overhead than the other two solutions. For the moment the FIFO seems to be the best choice. It is simple and performs well. Therefore the FIFO is the one which will be used throughout the rest of this thesis.

4 Adaptivity in the context of the Multigrid

Up to now we have just looked at the Gauss-Seidel, which we employ as smoother. But our objective is to add adaptivity in all parts of the multigrid. The two important other parts of a multigrid algorithm beside the smoother are the interpolation and the restriction. They do not consume too much runtime themselves (compared to the smoother routine) but they can be used to determine which "parts" of the problem domain are still interesting at all and should be further treated.

4.1 Interpolation

To introduce adaptivity into the interpolation we just need to look at which points have to be interpolated and which can be safely ignored. In our correction scheme every point that has a value different from zero has to be interpolated if we do not want to discard any changes. To determine which points this would be we could just take the straightforward way and test the value at every gridpoint. At a first glance this would not make too much sense as it is about the same amount of work as just doing the full interpolation, but as we will see in section 4.3 this is not quite the case. Another approach is to mark and interpolate every point that was changed during the preceding pre-smoothing, coarse grid correction and post-smoothing steps. The effect, that means the points which are interpolated, of both approaches is the same. The difference is that in the second approach we have to do a lot of the operations that insert a point into an active set. The first approach does not need these, but in exchange has to look at all residuals. In a setting with a low amount of activity in the overall domain the second approach is probably the better idea.

In the program I wrote the adaptivity that was achieved through the interpolation was meager, to say the least. In the examples I did the algorithm almost always did a full interpolation. The reason why this happens is that if a point is changed by the Gauss-Seidel on a very coarse grid, then it has to be interpolated and therefore the values of the nine points on the next finer grid, depending on this one grid point on the coarse grid, have to be interpolated in turn. If the direct solver on the coarsest grid consists of a simple Gauss-Seidel step on a on a single gridpoint then a change of this one point means that a full interpolation on the whole grid has to be done.

4.2 Restriction

The treatment of the restriction is a bit more complicated than that of the interpolation. Again we first have to determine which points have to be restricted and which can be ignored. Theory states that

$$e \sim \sum_{i \, \epsilon \, levels} \|r_i\| \tag{4.2.1}$$

or in words: The error is directly proportional to the sum of the residuals on all levels used in the multigrid. Thus if we can guarantee that the residual on all levels converges towards zero than the error will do this, too.

If we suppose for a moment that in the correction step we could solve the residual equation 1.1.1 exactly, then after the interpolation and correction of the fine grid another restriction of the fine grid residual would yield a right hand side equal to zero as can be seen in equation 4.2.2 and 4.2.3.

$$\begin{array}{rcl} f^l_{new} & = & I^l_{l+1} r^{l+1}_{new} \\ & = & I^l_{l+1}(f^{l+1} - A^{l+1} v^{l+1}_{new}) \\ & = & I^l_{l+1}(f^{l+1} - A^{l+1}(v^{l+1}_{old} + I^{l+1}_l v^l)) \\ & = & I^l_{l+1}(f^{l+1} - A^{l+1} v^{l+1}_{old} - A^{l+1} I^{l+1}_l v^l) \\ & = & I^l_{l+1}(r^{l+1}_{old} - A^{l+1} I^{l+1}_l v^l) \\ & = & I^l_{l+1} r^{l+1}_{old} - A^l v^l \end{array} . \tag{4.2.2}$$

witl+1 l denoting the level, I^l_{l+1} the restriction operator and I^{l+1}_l the interpolation operator. As v^l is the exact solution of

$$A^l v^l = I^H_h r^{l+1}_{old} \tag{4.2.3}$$

this yields that f^l_{new} is zero after interpolation and restriction and this in turn means that the residual on the level l will be zero. If the residual on one grid is zero it follows that the right hand sides and therefore also the residuals, on all coarser grids are zero, too.

Normally we will not be able to exactly solve the residual equations. Nevertheless if we want to be sure that we do not have to recalculate a restriction in a point, we have to be sure that the residuals on all coarser grids are bounded by a certain value. If the residuals in each point on all coarser grids are bounded then the global residuals on all coarser grids will be bounded, too, and this in turn will guarantee the convergence of the algorithm according to equation 4.2.1.

As before we consider a level on which we just did a correction and no post-smoothing, but instead an immediate restriction to the next coarser grid again. The equation for this is exactly the same as equation 4.2.2 with the only difference that v^l is not the exact solution to 4.2.3. Therefore r^l will not be zero, but just be bounded again by the tolerance used for the Gauss-Seidel on the coarse grid. This is sufficient to conclude that the Gauss-Seidel would have nothing to do on this coarse grid as all the residuals are already below the tolerance.

From this we can conclude that, if a point on a grid is changed, then all the restrictions of points on coarser grids, which depend on the changed point will have to be recalculated. For the rest of the points we can omit a new restriction as their values for the right hand side on the coarse grid would yield a residual at this coarse grid point that would already be below the used tolerance. You could also think about just checking the points on the next coarser grid, and not on all grids. Although this would probably work in most cases, too, we would loose the mathematical guarantee that the algorithm converges.

The result of all this can be seen in the pictures in figure 4.2.1 and figure 4.2.2. Those are some examples where the restriction worked adaptive, but in most cases the restriction works either on the full grid, or nothing at all is restricted. Furthermore we can observe that the points where restriction is done has only in some cases something to do with the problem. The reason for this is that after some V-cycles the residuals all have an absolute value of up to the tolerance. The lower the tolerance gets the less the values can differ from each other and the more random the distribution of the values of the residuals will look. If we would

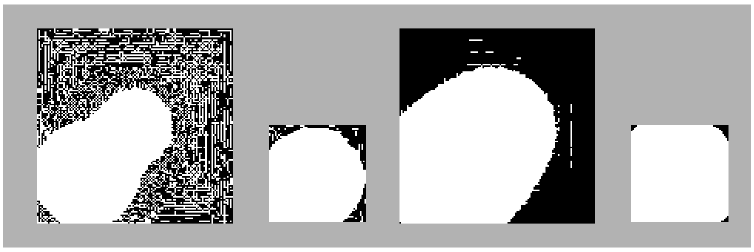

Figure 4.2.1: Pictures of adaptive restrictions in the course of an adaptive FMG run on the peaks problem with a tolerance of $\tau = 1E^{-8}$.

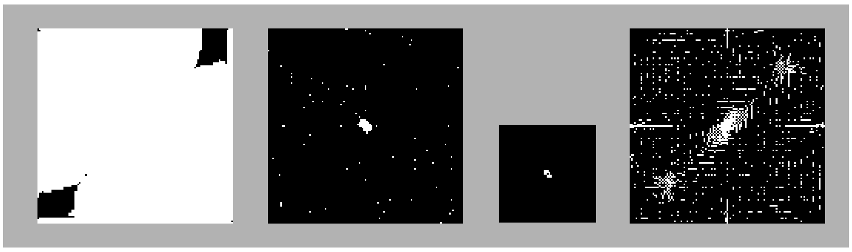

Figure 4.2.2: Pictures of adaptive restrictions in the course of an adaptive FMG run on the four corner problem with a tolerance of $\tau = 1E^{-8}$.

```
ℛ is the active set for the restriction
𝒢 is the active set for the Gauss-Seidel
ℐ is a the active set for the interpolation
l denotes the level in the multigrid

while {𝒢^l is not empty} do
      remove point p_{i,j} from 𝒢^l
      calculate p^{new}_{i,j}
      if |p^{new}_{i,j} - p_{i,j}| > τ then
            insert p_{i+1,j} into 𝒢^l
            insert p_{i-1,j} into 𝒢^l
            insert p_{i,j+1} into 𝒢^l
            insert p_{i,j-1} into 𝒢^l
            insert p_{i,j} into ℐ^l
            insert p_{i,j} into ℛ^l
            set p_{i,j} = p^{new}_{i,j}
      else
            discard p^{new}_{i,j}
      end if
done
```

Figure 4.3.1: Pseudo code for the adaptive Gauss-Seidel

work with a bigger tolerance, so that some parts of the domain already are "better" than the tolerance demands, we would probably see a more problem dependent effect of the restriction.

Naturally all this is, just as it was the case with the interpolation, very dependent on the tolerances used for the Gauss-Seidel as will be discussed in section 4.4.

4.3 How it all fits together

So far we still have viewed all the parts of the algorithm very isolated from each other. In this section I will sketch how the different parts and their adaptivity fit together.

First we have to do one full V-cycle so that in the next V-cycles we can start to do the adaptive restriction as described in section 4.2. The "full" means that we start this first V-cycle with full active sets for the Gauss-Seidel and the restriction on all levels. Nevertheless we can already do adaptive relaxation and interpolation just in the same way as in the other V-cycles. In all later V-cycles the algorithm will start with an active set for the Gauss-Seidel that is initialized by the last V-cycle. Every point that is changed in the Gauss-Seidel is added to the active set for the restriction and in case we are not on the finest grid we also add it to the active set for the interpolation (see also pseudo code 4.3.1). When the Gauss-Seidel is finished we do the restriction. During the restriction we add every point on the coarse grid that is affected by the restriction (that means which's right hand side does not stay zero) to the active set for the Gauss-Seidel on the coarse grid (see also pseudo-code for the restriction 4.3.2). Then the residual equation 1.1.1 is solved on the coarse grid (either

```
𝓡 is the active set for the restriction
𝓖 is the active set for the Gauss-Seidel
𝓣 is a temporary active set
l denotes the level in the multigrid

while {𝓡^l is not empty} do
     remove point p^l_{i,j} from 𝓡^l
     if i mod 2 = 0
          if j mod 2 = 0
               insert point p^{l-1}_{i div 2,j div 2} into 𝓣
          else
               insert point p^{l-1}_{i div 2,j div 2} into 𝓣
               insert point p^{l-1}_{i div 2,j+1 div 2} into 𝓣
          end if
     else
          if j mod 2 = 0
               insert point p^{l-1}_{i div 2,j div 2} into 𝓣
               insert point p^{l-1}_{i+1 div 2,j div 2} into 𝓣
          else
               insert point p^{l-1}_{i div 2,j div 2} into 𝓣
               insert point p^{l-1}_{i div 2,j+1 div 2} into 𝓣
               insert point p^{l-1}_{i+1 div 2,j div 2} into 𝓣
               insert point p^{l-1}_{i+1 div 2,j+1 div 2} into 𝓣
          end if
     end if
done
while {𝓣 is not empty} do
     remove point p^{l-1}_{i,j} from 𝓣
     calculate right hand side r at point p^{l-1}_{i,j}
     insert p^{l-1}_{i,j} into 𝓖^{l-1}
     insert p^{l-1}_{i,j} into 𝓡^{l-1}
done
```

Figure 4.3.2: Pseudo code for the adaptive restriction.

```
𝓘 is the active set for the interpolation
𝓖 is the active set for the Gauss-Seidel
l denotes the level in the multigrid
and I_l^l + 1 is the interpolation operator

while {𝓘^l is not empty} do
     remove point p^l_{i,j} from 𝓘^l
     for m = -1 to 1 do
          for n = -1 to 1 do
               p^{l+1}_{2i+m,2j+n} = p^{l+1}_{2i+m,2j+n} + I_l^{l+1} p^l_{i,j}
               insert p^{l+1}_{2i+m,2j+n} into 𝓘^{l+1} and 𝓖^{l+1}
          done
     done
done
```

Figure 4.3.3: Pseudo code for the adaptive interpolation.

directly or again by smoothing a coarse grid correction) and the solution is interpolated back to the fine grid. Each point on the fine grid that is changed in the course of the interpolation is added to the active set for the Gauss-Seidel and to the active set for the interpolation on the next finer grid (see also pseudo-code for the interpolation 4.3.3). After we have corrected our solution on the fine grid, we do the post-smoothing step and, again, have to add each point that is changed to the active set for the interpolation and the restriction. On arriving on the coarsest grid we solve the equation directly and every point that is different from zero is added to the active set for the interpolation. You could also think about using another tolerance here instead of interpolating every point that is not zero. When we arrive back on the finest grid, after having run through the whole V-cycle we have to keep in mind that we only need to do one smoothing step, which is post- and pre-smoothing in one. If we would do both, then the pre-smoothing for the next V-cycle would naturally do nothing anymore as all the work has already been done in the past post-smoothing step. Thus we can (and in fact have to) omit it right away.

So much for the basic V-cycle strategy. But there is one big problem with this. We cannot reduce the tolerances we are working with in the V-cycle, without having to do another full V-cycle (see section 4.2). But if we already start our first V-cycle with a very low tolerance then most of the work will be done in the first pre-smoothing step and this is definitely not what we want. One method to overcome this limitation is to use full multigrid (FMG) as described in section 1.1. When we use FMG we can choose to use a tolerance that will yield a result close to the truncation error right away. As FMG starts on the coarsest and therefore smallest grid a low initial tolerance does not matter too much in terms of work because there are only a few points an the grid anyway. On the higher levels we always start with an already pretty good approximation that we got from the previous step in the FMG so that we do not run into the problem, that the pre-smoothing on the currently finest grid does all the work.

But FMG introduces other problems. It allows us to say that we can always do adaptive restriction because “we have already been on the coarser grids” and therefore need only recalculate the restriction in case we changed something on any of the finer grids. But this statement that argues with the theoretical considerations from section 4.2 in mind has a weakness: What is "the same tolerance" in FMG on one level over the runs of the V-cycles that have their finest grid on step by step increasing starting levels? I was unable to find a mathematical solution to this problem. I think that the choice of tolerances I describe in section 4.4.2 seemed to work reasonably well for the Poisson equation, but to find a mathematically well funded rule that describes how you have to set the tolerances so that you still have the mathematical guarantee that the adaptive FMG scheme converges was out of the scope of this work.

4.4 Tolerances

Throughout this thesis I have been talking about the tolerances used as stopping criteria for the Gauss-Seidel and their importance. They have a great impact on how the adaptivity really performs, they can be used to steer how much work is done where in the algorithm and they decide how good the results that the algorithm delivers will be. The only problem is to set them so that the algorithm does a good job. But before I make some suggestions on how one could set them I will shortly discuss once more how they influence the algorithm.

4.4.1 Effects on the algorithm

The most obvious influence the tolerances have is that on the Gauss-Seidel. The tolerance steers how smooth the result is that the Gauss-Seidel delivers. Therefore if we supply a too small tolerance to the pre-smoothing step on the finest level, the algorithm would more or less try to solve the problem with the Gauss-Seidel on the finest level instead of just smoothing it a bit. But as the activity of the Gauss-Seidel also decides where restriction and interpolation have to be done, these are also indirectly influenced by the tolerances. There are multiple problems in choosing the tolerances:

- If we set the the tolerances on the coarse grids too high, then the coarse grid correction will be largely ineffective, and we will just be doing something like a normal Gauss-Seidel.
- If we set the the tolerances on the fine grids too low then the first pre-smoothing steps will do most of the work.
- The absolute values on the different grids differ greatly, because we just calculate the error for the next finer grid. Usually the error on a grid is a lot smaller than the absolute values.
- If we are using FMG then which tolerances should be used if the FMG advances (e.g. interpolates the approximation of the solution) to the next finer grid (see also section 4.3).

4.4.2 Some suggestions

Considerations about the tolerances could probably fill a whole book on their own. Nevertheless I will make a few suggestions and will present how I chose the tolerances for my algorithm. The first thing we observed in the preceding section was that we should not use too big tolerances on the coarse grids. The opposite of this, using too small tolerances on the coarse grids is not so dangerous because even if we have a too small tolerance the solution on the coarse and therefore small grid, would not take too much work. It even might be desirable to set the tolerances on the coarse levels a bit smaller compared to the tolerances on the fine grids. The second thing we noted is that in our correction scheme the values on the coarse grids are usually smaller than that of the according fine grids. We can even specify this a bit more exactly: In the case of the Poisson equation when using $u_{i,j} = 0$ as initial guess for the correction step, we have (from equation 3.1.1)

$$\begin{aligned} |r_{i,j}^{(l+1)}| &< \tfrac{4}{h^{(l+1)2}}\tau^{(l+1)} \\ |r_{i,j}^{(l)}| &< \tfrac{4}{h^{(l+1)2}}\tau^{(l+1)} \\ |r_{i,j}^{(l)}| &< 4\tfrac{4}{h^{(l)2}}\tau^{(l+1)} \end{aligned} \tag{4.4.1}$$

with l denoting the level, τ the tolerance and r the point residual. Therefore if we are already on a coarser grid than the finest grid and if we want to get a qualitatively equivalent smoothness on the next coarser grid, then we set the tolerance to,

$$\tau^{(l)} = 4(\tau^{(l+1)}) * \delta \tag{4.4.2}$$

where $1 > \delta > 0$. The motivation for this is the following: If we already are on a level where we are calculating a correction and therefore already have done a restriction, we know from equation 4.4.1 that all absolute values on the right hand side and thus all absolute values of the residuals are between 0 and $4\frac{4}{h^{(l)2}}\tau^{(l+1)}$. If we want to reduce these residuals by the same percentage δ we reduced the residuals on the previous level, we just multiply our previous tolerance with δ and thus get a new tolerance. As we do not know the range of the residuals on the finest grid in advance, we will usually specify an initial tolerance there, which is then scaled according to equation 4.4.2 for the coarser grids. The δ decides how much work is done in a single V-cycle and especially how much work is done on the coarser grids. Equation 4.4.2 just tries to keep τ on the same scale as the residuals we are working with. One should also try to choose the initial tolerance τ according to the choice of δ and the expected absolute size of the residuals.

Additionally we know that when we advance to the next finer level in the FMG scheme the truncation error goes down by a factor of something between 4 (theoretically) and 2 (a more practical value). Therefore when we advance to a new finer level in the FMG scheme we halve the tolerance τ and then during the V-cycles we treat the tolerances according to the relation 4.4.2. Although we have the problem that we cannot give any mathematically guarantees for the algorithm anymore at the moment it seemed to work reasonably well.

5 Conclusion

There are still a lot of open questions, like how to set the tolerances so that the algorithm performs best and how we could perhaps make any mathematical estimates about the performance and behaviour of the algorithm. The real life performance of the algorithm described in this thesis naturally is not too good. The overhead for doing all the bookkeeping operations for the active sets on a point per point basis is just too much. A simple gprof run of the program shows that it spends more than 90% of its time in the set-insert routine. On the other hand this work was mainly intended as a research project on how an algorithm like this would work. In practice one would not do all this on a point per point basis but rather divide the grid into patches and then do all of the above per patch and not per point. If these patches were tailored to the size of the machine caches then the overhead from the set operations probably will not make a too big effect anymore compared to the time needed to load another patch into the caches. Although there are good hopes that an algorithm like this would work well, this still has to be proven and before this can be done a lot of the practical problems like how to choose the tolerances will have to be solved. Thus, there is more than enough work left to do.

References

[1] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A Multigrid Tutorial.* SIAM, second edition, 2000.

[2] Hubbard. *Numerical Solution of Partial Differential Equations - II, Synspade 1970.* Academic Press, 1971.

[3] Prof. Ulrich Rüde. Fully adaptive multigrid methods. *SIAM, Journal of Numerical Analysis*, 30(1):230–248, February 1993.

[4] Prof. Ulrich Rüde. *Mathematical and Computational Techniques for Multilevel Adaptive Methods.* SIAM, 1993.