Multigrid Implementation in COMSOL Multiphysics®- Comparison of Theory and Practice

Wolfgang Joppich, University of Applied Sciences Bonn-Rhein-Sieg, Grantham Allee 20, D-53757 Sankt Augustin, wolfgang.joppich@h-brs.de

Abstract

Multigrid methods (MG) belong to the fastest solvers for partial differential equations. The key for this is an appropriate composition of the algorithmic components. The multigrid solver implemented in COMSOL Multiphysics is analyzed with respect to components and with respect to its numerical properties.

Keywords: linear system of equations, solver, multigrid method

1 Introduction

The simulation of many problems in engineering and natural sciences requires the numerical solution of partial differential equations. Usually, the finally occurring linear systems of equations are very large and demand for efficient algorithms, ideally with a linear dependence on the problem size. Multigrid methods in principle possess this property together with a convergence rate which is bounded away from one by a constant which is independent from the mesh size. To show these features the algorithmic components of MG, such as smoothing, creation of grid hierarchy, and grid transfer operators, have to be composed in an appropriate and problem-dependent way. The theory of MG offers concrete recommendations how to choose the algorithmic components in order to obtain optimal results [1, 3, 6, 7]. This implies that MG is not a fixed solver but rather a solution method with specific components adapted to the underlying problem.

COMSOL Multiphysics [2] offers geometric MG for solving linear systems of equations iteratively. To estimate the quality of the implementation the features of COMSOL are used to compose different MG algorithms. They are applied to selected model problems in order to determine empirical convergence rates, which are compared both with theoretically predicted convergence rates [4] and with empirical convergence rates from a MG matlab implementation [5].

2 The Multigrid Method

The neat combination of the two basic principles *smoothing* and *coarse grid correction* leads to the iteratively cycling MG method. An elementary understanding of these principles will be provided by analyzing the Poisson equation on the unit square with Dirichlet boundary conditions.

2.1 Smoothing

The problem is $-\Delta u = f$ on the unit square $\Omega \subset I\!R^2$ with Dirichlet boundary conditions $u = g$ on the boundary $\partial\Omega$. The domain Ω is covered by a mesh with quadratic cells: $h = h_x = h_y = \frac{1}{N}$, where N typically is a power of two. Thus the discrete computational domain including its boundary is $\overline{\Omega}_h = \{(x_i, y_j) \mid x_i = ih, y_j = jh \text{ with } 0 \le i, j \le N\}$. Grid functions defined on $\overline{\Omega}_h$ are denominated by the subscript h: u_h, f_h, g_h. Evaluating the grid function u_h at a grid point $\mathbf{x} = (x_i, y_j) = (ih, jh)$ is written as $u_{i,j}$.
The Laplacian $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ is approximated by central differences. Collecting the

discrete equations results in a linear system of equations $-\Delta_h u_h = f_h$. Due to the discretization formula the matrices are sparse. To solve the systems, the use of iterative schemes is natural, especially for large N. The use of Jacobi- or Gauss-Seidel methods, including their weighted variants, is self-evident but shows, that an initially good convergence becomes bad, very soon.

The functions $\varphi^h_{l,m}(x,y) = \sin l\pi x \cdot \sin m\pi y$ build a basis of the space of grid functions of $\overline{\Omega}_h$ with boundary values equal to zero and they are eigenfunctions of the discrete Laplacian Δ_h. It is easy to show that they are also eigenfunctions of the Jacobi-iteration operator $\mathbf{P}_J$, which allows an easy calculation of the eigenvalues. The spectral radius of the Jacobi method for the discrete Poisson equation is given by $\rho(\mathbf{P}_J) = \cos \pi h$ which explains the convergence behavior: the smaller h the worse the convergence. Such h-dependent convergence rates can be seen for other iterative methods, too.

A straightforward way to construct a coarser $2h = H$-mesh from the previously described $\overline{\Omega}_h$ is to omit every second line and row (standard coarsening). For standard coarsening frequencies with wave numbers $1 \leq l, m < \frac{N}{2}$ can be seen on the coarse grid [5]. They are called low-frequent with respect to the fine grid. On the other hand, those with wave numbers $\max(l,m) \geq \frac{N}{2}$ are called high-frequent.

The analysis corresponding to the above classification of high and low frequent shows: high frequent components are damped fast, low frequent components are damped slowly.

Conclusion 2.1 *This behavior, which is characteristic for many iterative methods, is the smoothing property, the first basic principle of MG methods.*

2.2 Coarse Grid Correction

Still having in mind $-\Delta_h u_h = f_h$ including Dirichlet boundary conditions, we switch to the notation $L_h u_h = f_h$. The unknown solution u_h is decomposed into the current approximation $w_h^{(n)}$ after n iterations and into the unknown corresponding error $u_h = w_h^{(n)} + \tilde{e}_h^{(n)}$ or $\tilde{e}_h^{(n)} = u_h - w_h^{(n)}$. Knowing the error to an approximation, the solution could be determined. To obtain an equation for the error, the operator L_h is applied to $\tilde{e}_h^{(n)}$, and we get the residual equation $L_h \tilde{e}_h^{(n)} = r_h^{(n)}$.

Adding $\tilde{e}_h^{(n)} = L_h^{-1} r_h^{(n)}$ to $w_h^{(n)}$ would deliver the solution u_h. When not solving the residual equation exactly, but approximately by a residual correction $\hat{e}_h^{(n)}$, then $w_h^{(n)} + \hat{e}_h^{(n)}$ will not be equal to u_h, but it should improve $w_h^{(n)}$ such that $w_h^{(n+1)} = w_h^{(n)} + \hat{e}_h^{(n)}$ is an improved approximation to u_h. This is the well-known idea of defect correction methods. The essential idea is to calculate the approximate solution to the residual equation not on the h-mesh, but on a $H = 2h$-mesh. Therefore an appropriate restriction operator I_h^H transforms the fine grid function $r_h^{(n)}$ to the coarse grid function $r_H^{(n)} = I_h^H r_h^{(n)}$. L_H is the same operator like L_h but h replaced by H. Then $L_H e_H^{(n)} = r_H^{(n)}$ is solved: $e_H^{(n)} = L_H^{-1} r_H^{(n)} = L_H^{-1}(I_h^H r_h^{(n)}) = L_H^{-1}(I_h^H f_h - I_h^H L_h w_h^{(n)})$. An interpolation operator I_H^h from $\overline{\Omega}_H$ to $\overline{\Omega}_h$ transfers the coarse grid solution to the fine grid $\hat{e}_h^{(n)} = I_H^h e_H^{(n)}$. The interpolated quantity improves $w_h^{(n)}$ to $w_h^{(n+1)} = w_h^{(n)} + \hat{e}_h^{(n)}$.

Conclusion 2.2 *These steps are called coarse grid correction. The coarse grid correction is the second basic principle of MG methods.*

2.3 The Correction Scheme

Coarse grid correction makes only sense if the fine grid function to be transferred no longer contains high frequent components. The final idea now is to use the smoothing property for eliminating high frequent components of the error in combination with coarse grid correction for a fast computation of the residual correction. As a result the two grid correction scheme to determine $w_h^{(n+1)}$ from $w_h^{(n)}$ can be formulated. $\mathcal{S}_h^{\nu} w_h^{(n-1)}$ denotes the application of ν smoothing steps to $w_h^{(n-1)}$.

Algorithm 2.1 *Correction Scheme (CS)*

$$
\begin{cases}
(1) & \textit{pre smoothing} & \bar{w}_h^{(n)} := \mathcal{S}_h^{\nu_1} w_h^{(n-1)} \\
(2) & \textit{residual calculation} & r_h^{(n)} := f_h - L_h \bar{w}_h^{(n)} \\
(3) & \textit{residual restriction} & r_H^{(n)} := I_h^H r_h^{(n)} \\
(4) & \textit{exactly solving the} & \\
 & \textit{coarse grid problem} & L_H \tilde{e}_H^{(n)} = r_H^{(n)} \\
(5) & \textit{correction transfer} & \tilde{e}_h^{(n)} := I_H^h \tilde{e}_H^{(n)} \\
(6) & \textit{correction} & \tilde{w}_h^{(n)} := \bar{w}_h^{(n)} + \tilde{e}_h^{(n)} \\
(7) & \textit{post smoothing} & w_h^{(n+1)} = \mathcal{S}_h^{\nu_2} \tilde{w}_h^{(n)}
\end{cases}
$$

The pre smoothing step eliminates high frequent error components. The smooth error can be approximated well on the coarse grid and the coarse grid problem is well defined. Its solution is interpolated to the fine grid. Interpolation introduces high frequent modes, which are eliminated again by smoothing - post smoothing. Rekursively applying the two grid correction scheme to step (4) of Algorithm 2.1 creates the multigrid method, marching through a hierarchy of grids from the finest grid to the coarsest grid where the corresponding discrete problem has to be solved exactly. The number of γ executions of the two grid scheme to solve step (4) creates the different types of cycles: $\gamma = 1$ produces the V-cycle and $\gamma = 2$ delivers the W-cycle. The so-called F-Cycle is a compromise between V- and W-cycle.

3 From Theory to Practice

Due to the diversity of components it is legitimate to ask whether MG components influence MG convergence. The next question concerns the search for optimal components: can MG convergence using specified components be predicted? The most powerful method for such predictions is the local Fourier analysis (LFA) [4]. The LFA provides theoretical convergence rates, which are denoted in the following by ρ_{lfa}. In [5] a matlab environment (Grafical User Interface and MG solver) is described which includes a collection of components for model problems. The corresponding empirical convergence rates are denominated by ρ_{gui}.

3.1 The FEM Package COMSOL Multiphysics

COMSOL Multiphysics [2] offers techniques for modelling and simulating a large set of problems from engineering and science. COMSOL includes state of the art direct and iterative solver methods. The geometric MG method is among the iterative and pre-conditioning algorithms. Facing the discussed variety of MG components one should be aware that a commercial software product can not offer all desired or possible components a MG developer would like to have. Robustness, maintainability, and compatibility with the general design of data structure and software architecture are only a few reasons which require pragmatic decisions with respect to selection and implementation of MG components. The effect of such decisions is of interest. Can COMSOL-MG-solver reach the convergence behavior which is expected for typical model problems or have concessions to be made?

COMSOL Version 4.3b is used in the sense of a black box solver. All given data (convergence rate, time) are from information of the software (log information). To determine the time per cycle, the solution time for one cycle is subtracted from the solution time using $n+1$ cycles and this time difference is divided by n, thus eliminating the setup phase for the problem. The linear residual norm reached after $n+1$ cycles is divided by the linear residual norm using only one cycle. The n-th root of this quotient is the empirical asymptotic convergence rate ρ_{comsol}.

3.2 Practical Experiments

All given times refer to an Intel Core i7 Extreme 980X (Gulftown) processor at 3.33 GHz with 6 cores using turboboosting and hyperthreading, and 24 Gigabyte main memory. The experiments are designed in a way to analyse the general behavior of the geometric MG solver. The cycle types V, F, and W are compared with respect to convergence and time requirements. Because cycles with two pre-smoothing steps and one post-smoothing step turned out to be very efficient in practice, exclusively this cycle type is used. The tolerance for the accuracy of the iterative solver is always set to 10^{-10}.

Experiment 3.1 *The first experiment solves the Poisson equation on the unit square with Dirichlet boundary conditions. The first grid hierarchy is generated from a mapped mesh (quadratic cells) with 4.000.000 elements on the finest level l_9 by standard coarsening to a coarsest level l_1 with 256 elements. The problem to be solved on l_9 has 16.008.001 degrees of freedom.*

Table 1: *Solving the Poisson problem on a mapped mesh for 16.008.001 degrees of freedom on totally 9 Levels (9L).*

solver	time [s]	no. of cycles	time per cycle	ϱ_{comsol}	mem. (GB)
V-9L SOR	175	10	2.8	0.046	17.0
F-9L SOR	181	9	3.9	0.040	15.6
W-9L SOR	184	9	4.3	0.040	15.7
V-9L SSOR	195	10	3.7	0.053	15.9
F-9L SSOR	197	9	5.6	0.048	15.0
W-9L SSOR	203	9	6.3	0.048	15.0
V-9L Vanka	196	10	4.8	0.053	18.0
F-9L Vanka	205	9	6.1	0.048	18.0
W-9L Vanka	210	9	6.9	0.048	18.0
MUMPS	756			tol. n. r.	34.1
Pardiso	909			tol. n. r.	40.0
Spooles	909			–	

V-, W-, and F-cycles with two pre-smoothing and one post-smoothing steps in combination with the most natural smoother in this problem class, SOR ($\omega = 1.0$) are applied first. The desired accuracy is reached within 9 to 10 iterations. The total solution time ranges from 175 to 184 seconds. The time per cycle reflects the numerical complexity of the cycle types: the V-cycle is the cheapest one, the W-cycle is the most expensive one. Well known is the slightly worse convergence rate of the V-cycle compared to that of the other cycle types. F- and W-cycle show the same convergence behavior, although the F-cycle is cheaper than the W-cycle. The average empirical convergence rate is $\rho_{comsol} = 0.046$ for V- and $\rho_{comsol} = 0.040$ for both F- and W-cycle. Compared to the usually conservative prediction of the Fourier analysis with $\rho_{lfa} = 0.119$ for W- and $\rho_{lfa} = 0.131$ for V-cycle this is an excellent result, especially when comparing with the matlab implementation applying a V-cycle with Gauss-Seidel smoothing, FW on a 1025×1025-mesh, and standard coarsening for totally ten MG level: $\rho_{gui} = 0.084$.

Due to the offered collection of point smoothers SSOR is the next natural choice. SSOR realizes a forward scanning during the first step and a backward scanning during the second step. The invested numerical work per SSOR step therefore is twice that for a single SOR step. The increased smoothing effort does not pay out, although the time per cycle is increased by almost fifty

per cent.
Another successfully used smoother in the MG context is the Vanka smoother. This method simultaneously solves a small local system for all degrees of freedom belonging to a particular element. This type of block relaxation can be recommended especially for Navier Stokes equations. The selection of this smoother for the Poisson problem reproduces the convergence speed of the SSOR smoother. Solving the small local systems increases both the time per cycle and the memory requirement for this type of smoothing.
Just to be mentioned: the direct solver of COMSOL (MUMPS and Pardiso multithreaded) required 756 and 909 seconds, respectively. The relative tolerance was not reached. The direct solvers requested more than twice the memory compared to the most efficient multigrid variant. □

Conclusion 3.1 *The quality of the smoother influences convergence speed. The V-cycle using Gauss-Seidel (SOR) creates the fastest MG method for the Poisson model problem.*

The Jacobi method plays an important role for the theoretical analysis and development of the MG method. COMSOL also offers this type of smoother. The question is, which properties this solver shows in practice. Because the previous experiment has been performed on a mapped mesh, which is not typical for FEM applications, the following experiment uses a hierarchy of free triangular meshes.

Experiment 3.2 *The grid hierarchy starts from the coarsest mesh with 268 elements and totally seven, eight or nine levels with* $l_8 = 4.390.912$ *and* $l_9 = 17.563.648$ *elements are used. These meshes lead to 8.786.945 and 35.137.537 degrees of freedom, on* l_8 *and* l_9*, respectively.*
Selecting the smoothing scheme to Jacobi with default parameters, especially relaxation factor $\omega = 1.0$*, the MG method for the Poisson problem on 7 levels comes to an astonishing result: divergence.*

Table 2: *Solving the Poisson equation with MG using the Jacobi smoother on a free triangular mesh*

solver	time [s]	no. of cycles	time per cycle	ϱ_{comsol}	mem. (GB)
W-7L Jac $\omega = 1.0$				div.	
V-8L Jac $\omega = 1.0$	84	29	1.1	0.402	8.7
V-8L Jac $\omega = 0.8$	71	17	1.1	0.194	8.6
F-8L Jac $\omega = 0.8$	77	17	1.4	0.192	8.6
W-8L Jac $\omega = 0.8$	79	17	1.6	0.192	8.7
V-9L Jac $\omega = 0.8$	596	17	8.4	0.193	30.7
V-9L SOR	450	10	3.0	0.051	29.3

This is in complete agreement with theory, which states for the Poisson problem that Jacobi with relaxation factor $\omega = 1.0$ *has bad smoothing properties. The smoothing analysis for MG proves, that Jacobi with* $\omega = 0.8$ *has optimal smoothing properties [1, 3, 6, 7]. Nevertheless, the smooting is worse than that for Gauss-Seidel (SOR), which is theoretically expected and practically observed.*
The problem on l_8 *is solved up to the required accuracy within 17 cycles (all types). Again, F- and W-cycle reach the same* $\rho_{comsol} = 0.192$*, whereas the V-cycle converges slightly worse. The time per cycle is smaller than 2 seconds, with a total time below 80 seconds in general. The V-cycle with Jacobi smoothing (*$\omega = 0.8$*) on* l_9 *requires 17 iterations, solves within 596 seconds, shows* $\rho_{comsol} = 0.193$*, and needs 8.4 seconds per cycle. This problem on* l_9 *could not be solved by a direct solver. The convergence rates on* l_8 *and* l_9 *are almost identical.* □

Conclusion 3.2 *The previous experiments show that smoothing is a key component for MG. The use of default parameters can be counter-productive, because smoothing properties can be destroyed. The results concerning V-, F-, and W-cycle show that the COMSOL MG implementation behaves as one would expect from theory.*

The following experiment will show that knowledge about MG theory helps finding the limits of MG even for a model problem.

Experiment 3.3 *The differential equation* $Lu = \varepsilon \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ *with* $\varepsilon = 0.01$ *is considered now. The similarity to the Poisson equation implies a similar choice of components. A hierarchy of totally eight free triangular meshes is created by regular refinement from* $l_1 = 268$ *elements to* $l_8 = 4.390.912$ *elements with 8.786.945 degrees of freedom (see Experiment 3.2).*

Table 3: *The anisotropic model problem on a free triangular mesh of 8.786.945 degrees of freedom*

solver	time [s]	no. of cycles	time per cycle	ϱ_{comsol}	mem. (GB)
V-8L SOR	391	298	1.14	0.915	8.4
F-8L SOR	502	297	1.51	0.915	8.4
W-8L SOR	544	297	1.65	0.915	8.4
V-8L SSOR	407	198	1.79	0.877	8.3
V-8L Vanka	450	198	2.00	0.877	9.7
V-8L SORline	441	68	5.08	0.692	10.3
MUMPS	192			tol. n. r.	16.5
Pardiso	101			tol. n. r.	20.0
Spooles	487			tol. n. r.	24.2

The results in Table 3 show a dramatically increased time to solve. The large number of necessary cycles is caused by the bad convergence rate of only $\varrho = 0.915$ *for SOR-smoother. Cycles with SSOR- and Vanka-smoother are not significantly faster.*
The matlab program with Gauss-Seidel smoothing comes to a $\rho_{gui} = 0.939$ *and local Fourier analysis predicts* $\rho_{lfa} = 0.942$*, confirming the observations and indicating a principal difficulty. Nevertheless, using an appropriate smoother for this anisotropic problem, the Matlab program reaches* $\rho_{gui} = 0.025$.
The explanation requires insight into MG theory. Due to the $\varepsilon = 0.01$ *the coupling of unknowns into y-direction is stronger than the coupling into x-direction. Therefore the error into the direction of weak coupling is smoothed less than the error into y-direction. To compensate this, there are essentially two strategies. The first one tries to capture the anisotropy by a modified coarsening (semi-coarsening into the direction of strong coupling). The second modifies the smoother such that all strongly coupled variables (y-direction) are solved together (block relaxation). On a standard FD-mesh this is a column. If the columns are scanned zebra-like the smoothing is even improved and MG convergence reaches the above mentioned* $\rho_{gui} = 0.025$*. Either approach is not easy to realize in a finite element context. COMSOL offers a so-called SOR-line smoother which automatically searches for anisotropies either in the mesh or in the system matrix. After some tuning of parameters of the SOR-line smoother a* $\rho_{comsol} = 0.692$ *could be reached, and only 68 cycles were necessary to reach the standard accuracy. The big disadvantage: the time per cycle is 5.08 seconds and the time to solve is 441 seconds. This is not really convincing, but has to be expected for MG with strong anisotropies.* □

The experiment for the anisotropic Poisson equation reveals the difficulty of MG for such types of problems. While MG does not solve faster than 391 seconds, the direct solver behave completely different. MUMPS finishes within 192 seconds, but the tolerance of 10^{-10} was not reached. Pardiso with nested dissection multithreaded, forward and backward substitution also multithreaded finished within 101 seconds. The memory requirement for the direct solvers again is about twice that of the MG solver.

Experiment 3.4 *COMSOL provides a model file to simulate a free-convection problem where a thermos holding hot coffee dissipates thermal energy. The default solver is a direct one. In a first step the number of degrees of freedom is increased to 1.975.146. Refining this mesh twice, leads to problems*

of 5.084.465 and 20.323.281 degrees of freedom, respectively. This series of problems is solved with MG, Pardiso, and MUMPS. The corresponding results are shown in Table 4.

Table 4: *Solving a sequence of problems derived from the V3.5a COMSOL model library (heat transfer, thermos laminar flow)*

d.o.f.	solver characteristics	total time (seconds)	mem. (GB)
1.975.146	MG-V(2,1)-6L SOR	50	7.4
	direct Pardiso	83	11.1
	direct MUMPS	157	9.9
	direct Spooles	254	8.6
5.084.465	MG-V(2,1)-7L SOR	128	8.3
	direct Pardiso	220	19.2
	direct MUMPS	410	14.8
	direct Spooles	857	20.6
20.323.281	MG-V(2,1)-8L SOR	695	25.5
	direct Pardiso	cancelled	-
	direct MUMPS	cancelled	43.8
	direct Spooles	omitted	-

Already in case of the moderate size problems a standard MG solver with SOR smoother beats the direct solver both with respect to computing time and with respect to memory requirement. The large problem could not be solved using the direct solver. MUMPS has been cancelled after 27 minutes within the first Newton step (MG required 4 to solve) and indicating the request for 43.8 Gbyte of virtual memory. Basically, the direct solver failed due to their memory requirements. □

4 Conclusion

The essential component for an efficient MG algorithm is the smoothing operator. In COMSOL, the standard methods are available. Variants with different types of scanning the equations are realized, too. Collective methods of Vanka type and block-relaxation methods (SOR-line) are also available. Additional iterative methods can be used in the context of MG (CG with ILU) for smoothing. The grid hierarchy can be created by either coarsening from fine to coarse or by refining from coarse to fine. This is possible on triangular and on mapped meshes. The most relevant cycle types are implemented.
Exploiting all these possibilities, COMSOL allows the composition of time-, convergence-, and memory-efficient MG algorithms both for model problems and concrete aplications which work in the range predicted by theory and known by experience.

References

[1] Brandt, A.: Multigrid Techniques: 1984 Guide With Applications to Fluid Dynamics, GMD-Studie No. 85, GMD, Sankt Augustin, 1984.

[2] COMSOL Multiphysics Reference Manual, COMSOL 4.3b, May 2013, ©COPYRIGHT 1998-2013 by COMSOL AB.

[3] Hackbusch, W.: Multi-Grid Methods and Applications, Springer-Verlag, Berlin, 1985.

[4] Wienands, R. and Joppich, W.: Numerical Insights into Local Fourier Analysis for Multigrid Methods, CRC Press, Boca Raton, 2004.

[5] Joppich, W.: Grundlagen der Mehrgittermethode, Shaker Verlag, Aachen, 2011.

[6] Stüben, K., Trottenberg, U.: Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications, Lecture Notes in Mathematics, 960, pp. 1-176, Springer-Verlag, Heidelberg, 1982.

[7] Trottenberg, U., Oosterlee, C., and Schüller, A.: Multigrid, Academic Press, San Diego, 2001.