ITERATIVE METHODS

THANH-VAN HUYNH, MICHAEL THIELE, FLORIAN WOLF

Thanks Gabriele and Christian for your great work.

Abstract. This is a report about the findings of Thanh-Van Huynh, Michael Thiele and Florian Wolf in the exercises accompanying Iterative Methods for Linear Systems. The lecture was held by Jun.-Prof. Dr. Gabriele Ciaramella with the assistance of Christian Jäckle in the winter term 2020/21 at the University of Constance.

Contents

1. Introduction

In the exercises we studied optimal control problems governed by the Laplace equation. We want to solve

$$\min_{y,u\in L^2(\Omega)} J(y,u) := \frac{1}{2}\ \|y-y_d\|^2_{L^2(\Omega)} + \frac{\nu}{2}\ \|u\|^2_{L^2(\Omega)} \tag{1.1}$$

$$\text{s.t.}\ \ -\Delta y = f + u \text{ in } \Omega \tag{1.2}$$

$$y = 0 \text{ on } \partial\Omega \tag{1.3}$$

for a regularization parameter $\nu > 0$. We are looking for a control function u. With discretized norm

$$\|\mathbf{x}\|^2_{L^2_h(\Omega)} := h^2 \sum_{j=1}^{n} \mathbf{x}_j^2$$

for $h > 0$ small enough, such that

$$A\mathbf{y} = \mathbf{f} + \mathbf{u}$$

is satisfied. Here $A = -\frac{1}{h^2} \cdot T$ where T is the two-dimensional laplace matrix, as we will see later. Notice that we do not have any boundary conditions. One possible way of finding a solution is the so-called reduced approach: We consider $\mathbf{y}$ as a function of $\mathbf{u}$, therefore obtaining the problem

$$\min_{\mathbf{u}\in\Omega} \hat{J}(\mathbf{u}) := J(\mathbf{y}(\mathbf{u}), \mathbf{u}).$$

2. Stationary Methods

2.1. **Deriving the Optimality System.** We see that

$$\|\mathbf{x}\|^2_{L^2_h(\Omega)} = h^2\ \|\mathbf{x}\|^2_2$$

E-mail address: thanh-van.huynh@uni-konstanz.de, michael.thiele@uni-konstanz.de, florian.2.wolf@uni-konstanz.de.
Date: October 19, 2020.

holds for arbitrary x. In combination with $\mathbf{y} = A^{-1}(\mathbf{f} + \mathbf{u})$, we obtain

$$\begin{aligned}\min_{\mathbf{u}\in\Omega} \hat{J}(\mathbf{u}) &= \frac{1}{2}\, h^2 \left\| A^{-1}(\mathbf{f}+\mathbf{u}) - \mathbf{y}_d \right\|_2^2 + \frac{\nu}{2}\, h^2 \|\mathbf{u}\|_2^2 \\ &= \frac{1}{2}\, h^2 \langle A^{-1}(\mathbf{f}+\mathbf{u}) - \mathbf{y}_d, A^{-1}(\mathbf{f}+\mathbf{u}) - \mathbf{y}_d\rangle + \frac{\nu}{2}\, h^2 \langle \mathbf{u}, \mathbf{u}\rangle \\ &= \frac{1}{2}\, h^2 \left(A^{-1}(\mathbf{f}+\mathbf{u}) - \mathbf{y}_d\right)^\top \left(A^{-1}(\mathbf{f}+\mathbf{u}) - \mathbf{y}_d\right) + \frac{\nu}{2}\, h^2 \mathbf{u}^\top \mathbf{u}.\end{aligned}$$

As we know that a solution $\mathbf{u}$ to the problem above has to satisfy $\nabla \hat{J}(\mathbf{u}) = 0$, this results in

$$\nabla \hat{J}(\mathbf{u}) = h^2 A^{-1}\left(A^{-1}(\mathbf{f}+\mathbf{u}) - \mathbf{y}_d\right) + \nu h^2 \mathbf{u} = 0.$$

We can do the following transformations

$$\begin{aligned} h^2 A^{-1}\left(A^{-1}(\mathbf{f}+\mathbf{u}) - \mathbf{y}_d\right) + \nu h^2 \mathbf{u} &= 0 \\ A^{-1}\left(A^{-1}(\mathbf{f}+\mathbf{u}) - \mathbf{y}_d\right) + \nu \mathbf{u} &= 0 \\ A^{-1}A^{-1}(\mathbf{f}+\mathbf{u}) - A^{-1}\mathbf{y}_d + \nu \mathbf{u} &= 0 \\ A^{-1}A^{-1}\mathbf{f} + A^{-1}A^{-1}\mathbf{u} - A^{-1}\mathbf{y}_d + \nu \mathbf{u} &= 0 \\ A^{-1}A^{-1}\mathbf{u} + \nu \mathbf{u} &= A^{-1}\mathbf{y}_d - A^{-1}A^{-1}\mathbf{f} \\ \left(\nu \mathrm{I} + A^{-1}A^{-1}\right)\mathbf{u} &= A^{-1}\left(\mathbf{y}_d - A^{-1}\mathbf{f}\right)\end{aligned}$$

and derive the optimality system, in the report called initial system,

$$\left(\nu \mathrm{I} + A^{-2}\right)\mathbf{u} = A^{-1}\left(\mathbf{y}_d - A^{-1}\mathbf{f}\right) \tag{2.1}$$

1 or alternatively, in the following called factored system,

$$\left(A^2 \nu + \mathrm{I}\right)\mathbf{u} = A\mathbf{y}_d - \mathbf{f}. \tag{2.2}$$

2.2. **Laplace Matrices.** First, we implement a function `FD_Laplacian` to create the Finite Difference matrix representation of the Laplace operator in sparse. Depending on whether we choose $d = 1$ or $d = 2$, we obtain the 1D Laplace Matrix

$$T_1 = \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \end{pmatrix} \in \mathbb{R}^{m\times m}$$

or the 2D Laplace Matrix (by using the Kronecker product $\otimes$)

$$T_2 = \mathrm{I}_m \otimes T_1 + T_1 \otimes \mathrm{I}_m = \begin{pmatrix} T_1 & \mathrm{I}_m & & \\ \mathrm{I}_m & T_1 & \mathrm{I}_m & \\ & \ddots & \ddots & \ddots \end{pmatrix} \in \mathbb{R}^{m^2\times m^2}.$$

. FD_Laplacian

```
Enter: matrix dimension $m \in \mathbb{N}$, $d = \{1,2\}$ choosing 1D or 2D
Return: Laplace matrix $T \in \mathbb{R}^{m\times m}$
diag = -2*sparse.identity($m$)
onesUpper = sparse.eye($m, k=1$)
onesLower = sparse.eye($m, k=-1$)
$T$ = diag + onesUpper + onesLower
if $d$ == 2:
    eye = sparse.eye($m$)
    $T$ = sparse.kron(eye, $T$) + sparse.kron($T$, eye)
return($T$)
```

2.3. **Fast Poisson Solver.** Now, we implement the Fast Poisson Solver `Fast_Poisson` to efficiently solve a linear system $Au = b$. As we do not have any boundary conditions, we have a simpler implementation:

. Fast_Poisson

```
Enter: matrix $V$ of eigenvectors of the matrix $A \in \mathbb{R}^{m\times m}$,
vector $\lambda$ of eigenvalues of the matrix $A$,
right-hand side $b$ of the linear system
Return: solution $u$ of the linear system $Au = b$
$B$ = reshape($b, (m, m)$)
$\tilde{B} = (V^\top (V^\top B)^\top)^\top$
for i in 0:m
    for j in 0:m
        $\tilde{u}_{ij} = (\tilde{B}_{i,j})/(\lambda_i + \lambda_j)$
$U = (V(V\tilde{U})^\top)^\top$
$u$ = reshape($U, m^2$)
return($u$)
```

2.4. **Convergence Analysis.** Now, we solve the system (2.1) by the stationary iteration

$$\mathbf{u}^{n+1} = -\frac{1}{\nu} A^{-2}\mathbf{u}^n + \frac{1}{\nu} A^{-1}(\mathbf{y}_d - A^{-1}\mathbf{f}) \tag{2.3}$$

and analyze its convergence behavior for different m and ν. First, we fix $\nu = 0.01$, then $m = 15$.

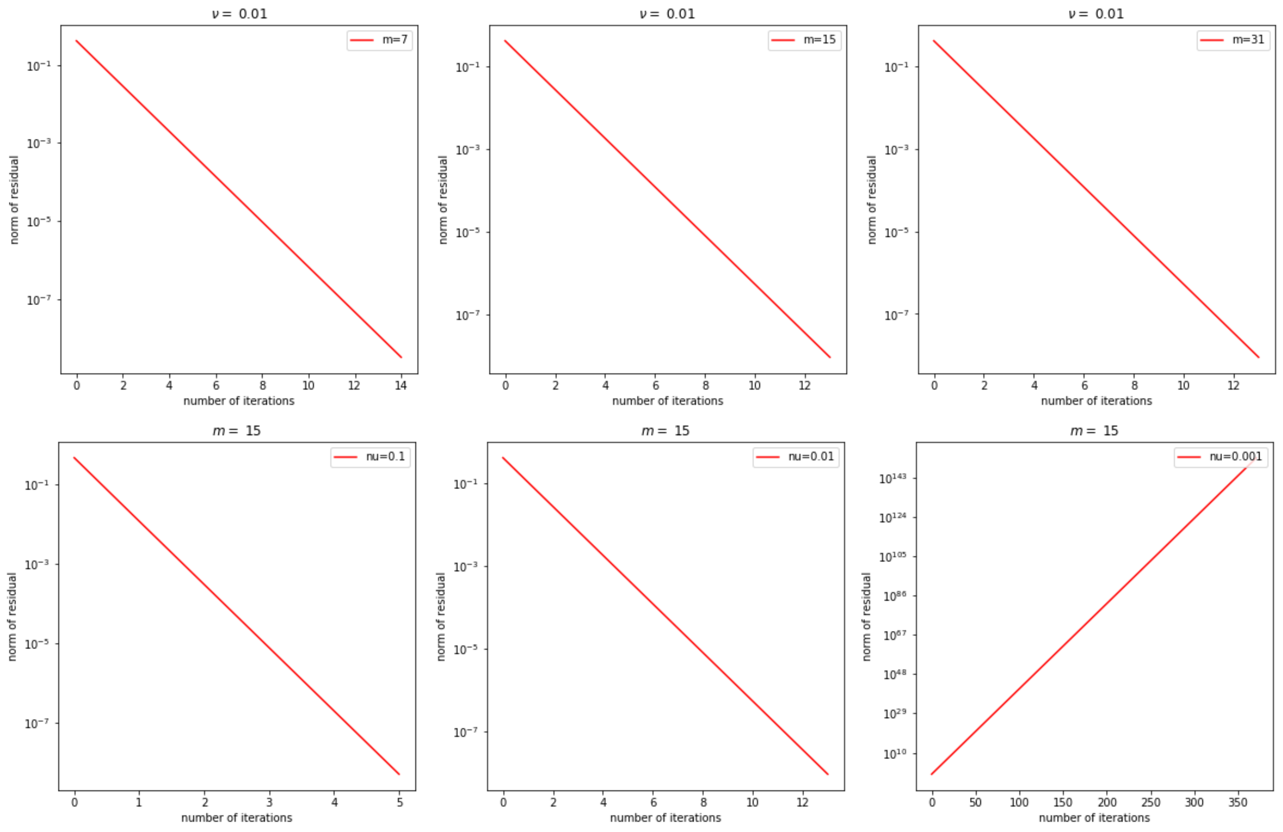

FIGURE 1. Convergence behaviour of the stationary iteration for different values of m and ν

As the figure shows, large m will lead to fast convergence. The same goes for ν: bigger values are the key to convergence, while choosing small ν will result in divergence.

2.4.1. *Eigenvectors and eigenvalues of the laplace matrices.* As we have seen in the section regarding the stationary solver and we will see this again, if we consider solving (2.1) using the conjugated gradient method, the poisson solver is the key, to get a fast solution to the A^{-1} problem. The solver requires a decomposition of the matrix V in diagonalized form. In the beginning we achieved this task using `scipy.sparse.eigs`. Luckily the eigenvectors and eigenvalues of the one dimensional laplace matrix are easy to calculate by hand, exploiting its Toeplitz-structure. For the $m \times m$ one-dimensional laplace, we get the following eigenvalues

$$\lambda_k = -2 + 2\cos\left(\frac{\pi k}{n+1}\right), \quad k = 1, \dots, m$$

and the corresponding eigenvectors with entries

$$(\mathbf{v}_k)_i = \sin\left(\frac{i \cdot \pi k}{m+1}\right), \quad i, k = 1, \dots, m$$

Now we can include these manually to get a more exact and especially quicker solver.

To get the eigenvectors and eigenvalues of the two-dimensional laplace matrix, we use problem 10 of the stationary iterative methods chapter of the script and proof its third bullet point. Therefore we use the formula $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$. For $j, k \in \{1, \dots, m\}$ arbitrary we get for $\lambda := \lambda_j + \lambda_k$ and $\mathbf{v} := \mathbf{v}_j \otimes \mathbf{v}_k$ that

$$\begin{aligned} A\mathbf{v} &= (\mathrm{I}_m \otimes T_1 + T_1 \otimes \mathrm{I}_m)\, \mathbf{v} \\ &= (\mathrm{I}_m \otimes T_1)\,(\mathbf{v}_j \otimes \mathbf{v}_k) + (T_1 \otimes \mathrm{I}_m)\,(\mathbf{v}_j \otimes \mathbf{v}_k) \\ &= (\mathbf{v}_j \otimes \lambda_k \mathbf{v}_k) + (\lambda_j \mathbf{v}_j \otimes \mathbf{v}_k) \\ &= \lambda_k\,(\mathbf{v}_j \otimes \mathbf{v}_k) + \lambda_j\,(\mathbf{v}_j \otimes \mathbf{v}_k) \\ &= \lambda \mathbf{v} \end{aligned}$$

Therefore the eigenvectors and eigenvalues of the two-dimensional laplace are obtained by using the kronecker product of the one-dimensional eigenvectors and summing up the one-dimensional eigenvalues. We will use the eigenvalues, to calculate the condition number of the matrix in the section about Krylov methods.

2.5. **Damped Jacobi.**

Now we solve the system

$$(\nu A^2 + I)\mathbf{u} = A(\mathbf{y}_d - A^{-1}\mathbf{f}) = A\mathbf{y}_d - \mathbf{f} \tag{2.4}$$

using the damped-Jacobi iteration

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \omega D^{-1}\left(A\mathbf{y}_d - \mathbf{f} - (\nu A^2 + I)\mathbf{u}^k\right) \tag{2.5}$$

with a damping parameter $\omega \in (0, 1]$ and $D = \operatorname{diag}(\nu A^2 + I)$. Out of the three parameters $\omega = 0.6, \nu = 10^{-5}$ and $m = 15$ we fix two and modify one in order to analyse the convergence behavior of th damped-Jacobi iteration.

As the figure at the bottom shows, and after some trial-and-error, we observe that $\omega \approx 0.5$ is the optimal choice to reach efficient convergence. For smaller ω damped-Jacobi still converges, but slower, and for $\omega \geq 0.7$ we do not obtain convergence. The dimension of ν also has a great impact: as ν gets smaller, we start to converge. As for the value of m, choosing bigger m will lead to divergence of the damped-Jacobi method.

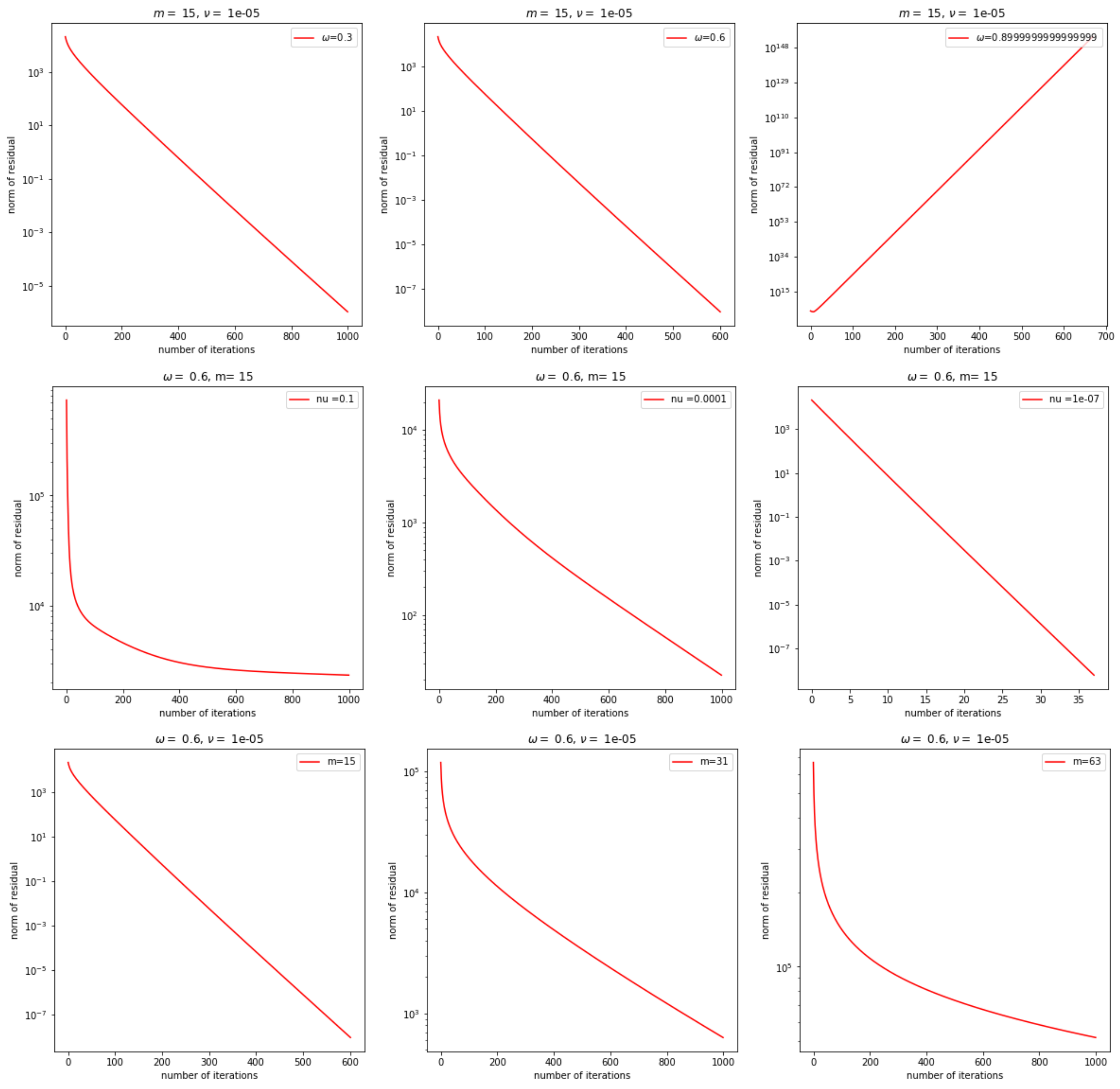

FIGURE 2. Convergence behaviour of the damped-Jacobi iteration for different values of ω, ν and m

3. Krylov Methods

3.1. **Convergence Analysis.** In this part of the project we want to use the fact, that the matrix arising in our disretization is symmetric and positive definite. For this reason we want to apply the conjugated gradient (CG) method, to solve our systems (2.1) and (2.2). For the first system we use the already implemented fast poisson solver, to get a matrix free version of the left-hand side. In the second case we construct our sparse matrices and use the `scipy.sparse.linalg.LinearOperator` method, to create a linear operator of our left-hand side of the equation. So in both cases the solution process is completely matrix-free.

Looking at figure 3 and 4, we can see the convergence behaviour of the two systems. In the plots ν ist getting bigger going from top to bottom and m is getting bigger going from left to right. As you can directly see in the plots, the solution process of the factored system is getting worse, as the size of the matrix increases. The initial system seems to be quite stable regarding finer grids. Meanwhile, the behaviour regarding different values of ν is

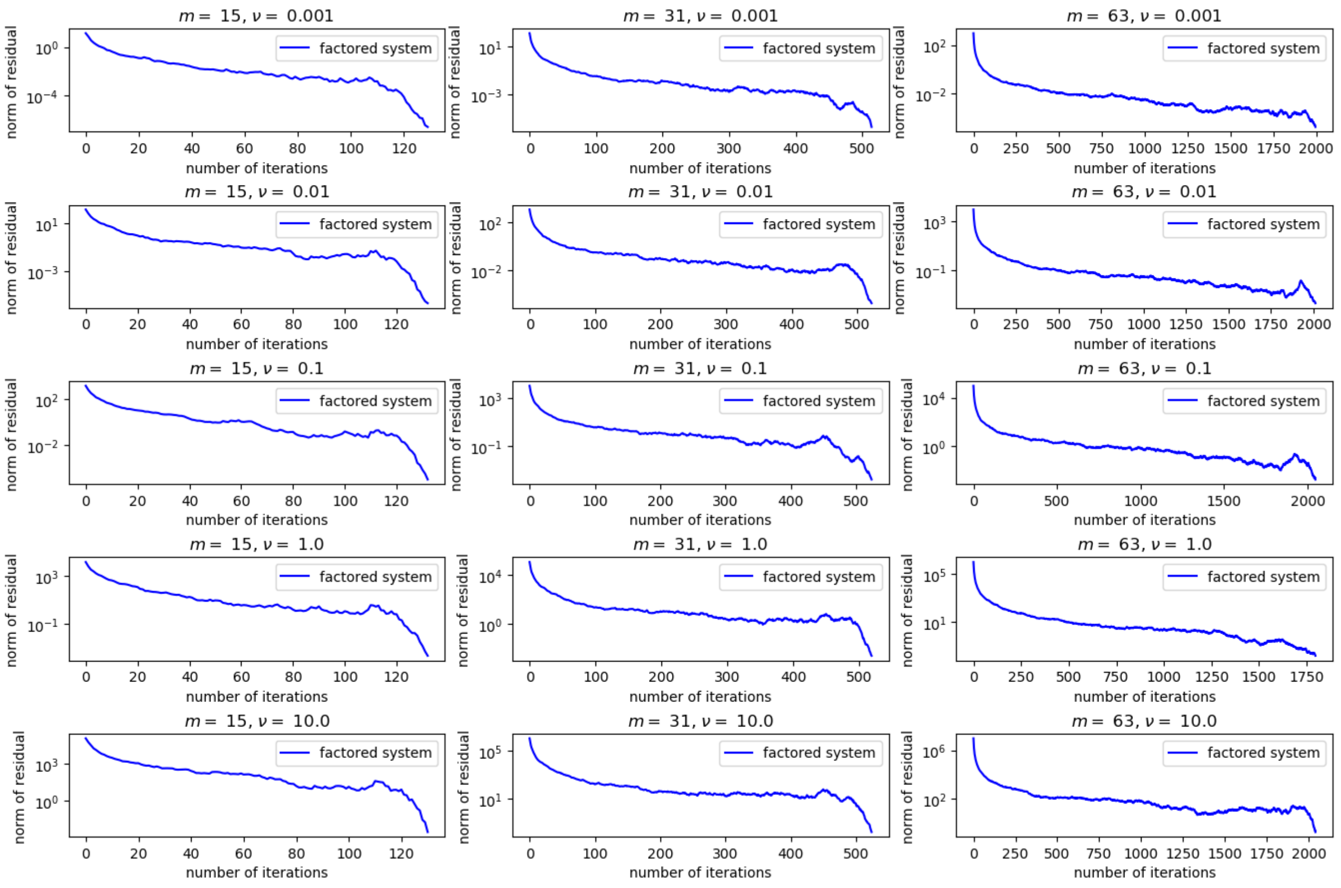

FIGURE 3. Convergence behaviour for different values of m and ν using CG to solve the factored systems. The x-axis is representing the number of iterations and the y-axis shows the normalized norm of the residuals in the discretized norm.

more interesting. While the convergence solving the factored system is getting worse if ν increases, the convergence rate of the initial system is getting faster and faster. So the systems behave in an opposite way, regarding the size of ν.

3.2. **Condition Number.** One can explain this, if we take a closer look at the conditon number of the systems for the upper combinations of ν and m. We can see this in figure 5 and 6. The three plots of the condition number explain really good, why we get the upper type of convergence behaviour. While both condition numbers get overall bigger from left to right (so with increasing matrix size due to bigger m), one can clearly see in each of the plots that the systems have opposed curves.

While the factored system (2.2) is worse conditioned for bigger ν, the condition number of the initial system gets better the bigger ν is. Looking at the formulas (2.1) and (2.2) this should now be really obvious. In (2.2) we factor our matrix A^2 by ν, so the bigger ν is, the worse the condition number gets. In the initial system (2.1) the identity matrix with factor ν dominates the diagonal of the left-hand side for bigger ν. So the condition number is getting better as ν increases, as the system is getting more and more diagonal dominant.

4. Multigrid

In our last section we use multigrid methods to solve our problems (2.1) and 2.2. These problems can be derived by discretization, as shown in the introduction. So we are able to apply multigrid methods. We have already seen two stationary methods:

$$\mathbf{u}^{n+1} = -\frac{1}{\nu} A^{-2} \mathbf{u}^n + \mathbf{f} \tag{4.1}$$

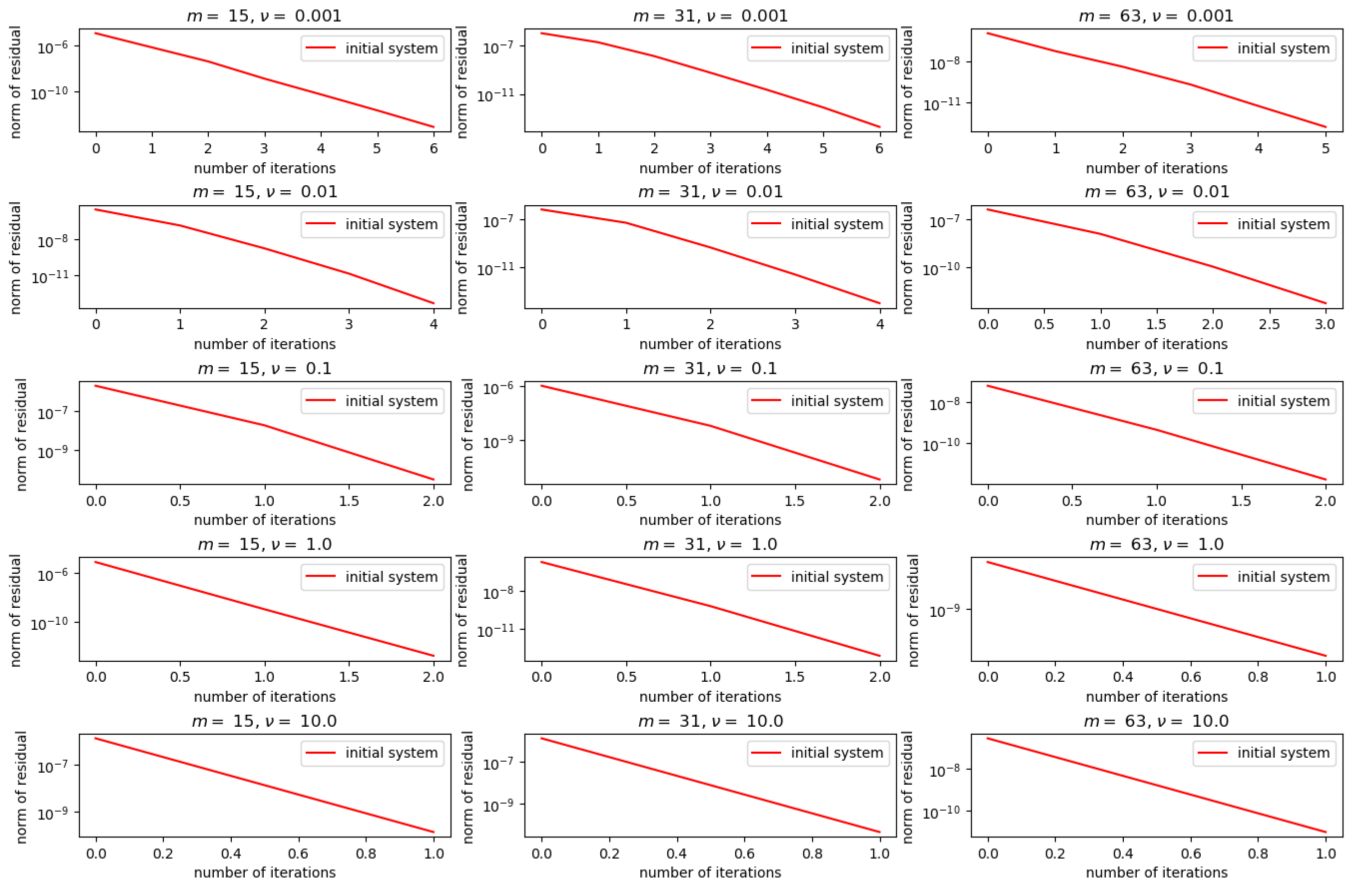

FIGURE 4. Convergence behaviour for different values of m and ν using CG to solve the initial systems. The x-axis is representing the number of iterations and the y-axis shows the normalized norm of the residuals in the discretized norm.

and

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \omega D^{-1}(\mathbf{f} - (\nu \mathrm{I} + A^2)\mathbf{u}^n) \tag{4.2}$$

In the following we will refer to (4.1) as the stationary method, in order to distinguish this method from (4.2), which is called damped Jacobi method with damping parameter $\omega \in (0, 1]$. We used (4.1) to solve (2.1) and (4.2) to solve (2.2). Now we want to apply (4.1) and (4.2) as smoothers in a multigrid method. In both multigrid methods we use the interpolation matrix P and the full weighting restriction matrix $R := \frac{1}{2}P^T$ in our coarse correction step. Furthermore we do three Pre-Smoothing and three Post-Smoothing steps.

4.1. **Stationary Methods as Smoother.** First we analyse (4.1) and it's corresponding multigrid method. We expect that the convergence of our multigrid methods gets better for rising ν. Regarding the notation of the script $M^{-1}N = -\frac{1}{\nu}A^{-2}$ is the iteration matrix of the stationary method. Therefore our stationary method converges faster for ν rising, which is shown in 2.5. For smaller ν we can expect slower or not even convergence at all, if the stationary method does not converge.

Now let us check, if our predictions were right. From figure 7 we can observe, that our multigrid method converges only for huge ν. This can be explained by the fact, that our sationary method converges only for huge ν. In this case the term $\nu\mathrm{I}$ dominates $\nu\mathrm{I} + A^{-2}$ and we get a iteration which would be similar to a jacobi one. As bigger m gets, ν needs to grow faster to achieve this effect. Therefore our multigrid method is not independent of the size m, because our stationary method can stop converging for m rising. Though provided that our multigrid method converges, it needs only a few iterations. In this case our method is extremly quick

4.2. **Damped Jacobi as Smoother.** Now we want to analyse the multigrid method with damped jacobi as smoother. We set $\omega = 0.5$, so we go half the way of every step. Let's make some predictions what happens, if we run our multigrid method. First we expect, that the multigrid method will be faster for shrinking ν and slower for

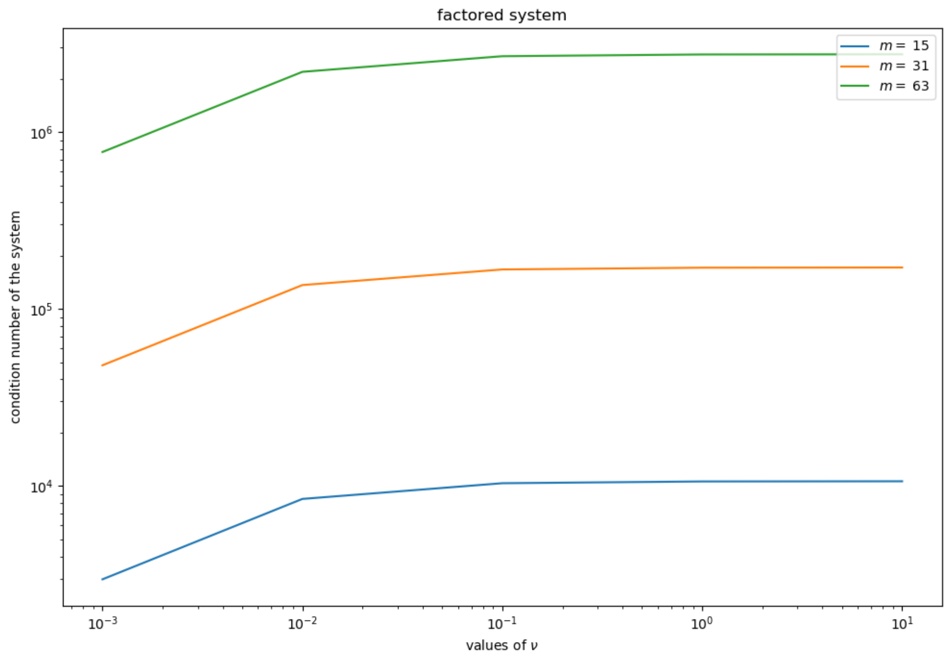

FIGURE 5. Condition numbers of the factored system for different values of m and ν represented in a double-logarithmic plot.

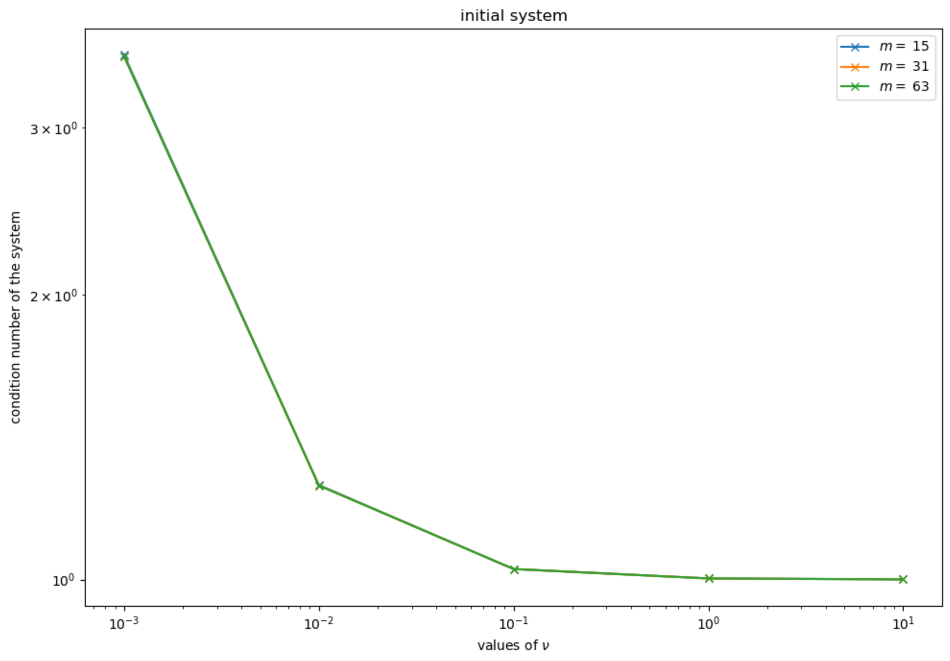

FIGURE 6. Condition numbers of the initial system for different values of m and ν represented in a double-logarithmic plot. (We used small little crosses to indicate that the three plots overlay.

bigger ν. Because of $I + \nu A^2$ is diagonal dominant for small ν, and Jacobi converges for such systems. For rising ν, we'll lose this effect. Nevertheless, A^2 is block tridiagonal:

$$\begin{aligned} A^2 &= (I \otimes T + T \otimes I)^2 \\ &= 2 \cdot T \otimes T + T^2 \otimes \mathrm{I} + \mathrm{I} \otimes T^2 \end{aligned}$$

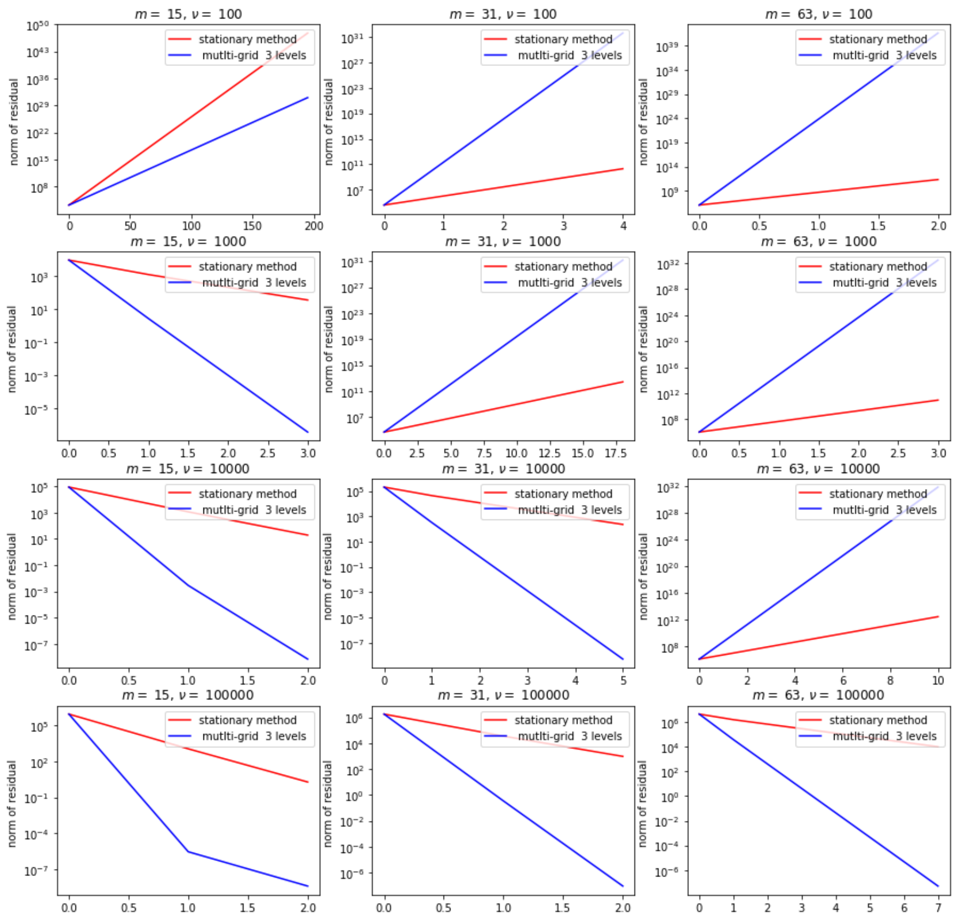

FIGURE 7. Number of iterations of multigrid in comparison with the stationary method for different ν and m

That's why Jacobi should still produces reasonable results.

Again let's verify our predictions. In figure 8 we can see exactly, what we expected. Our method is great for solving problems with small ν. Though for very small ν we come close to:

$$\mathbf{I}\mathbf{v} = \mathbf{f}$$

and the solution isn't so meaningful. Moreover we can see, that the multigrid method converges slower for ν rising. So thats again, what we predicted. From figure 8 we really see, that our multigrid method is independent of the size m. We are able to observe this phenomenon in nearly every row.

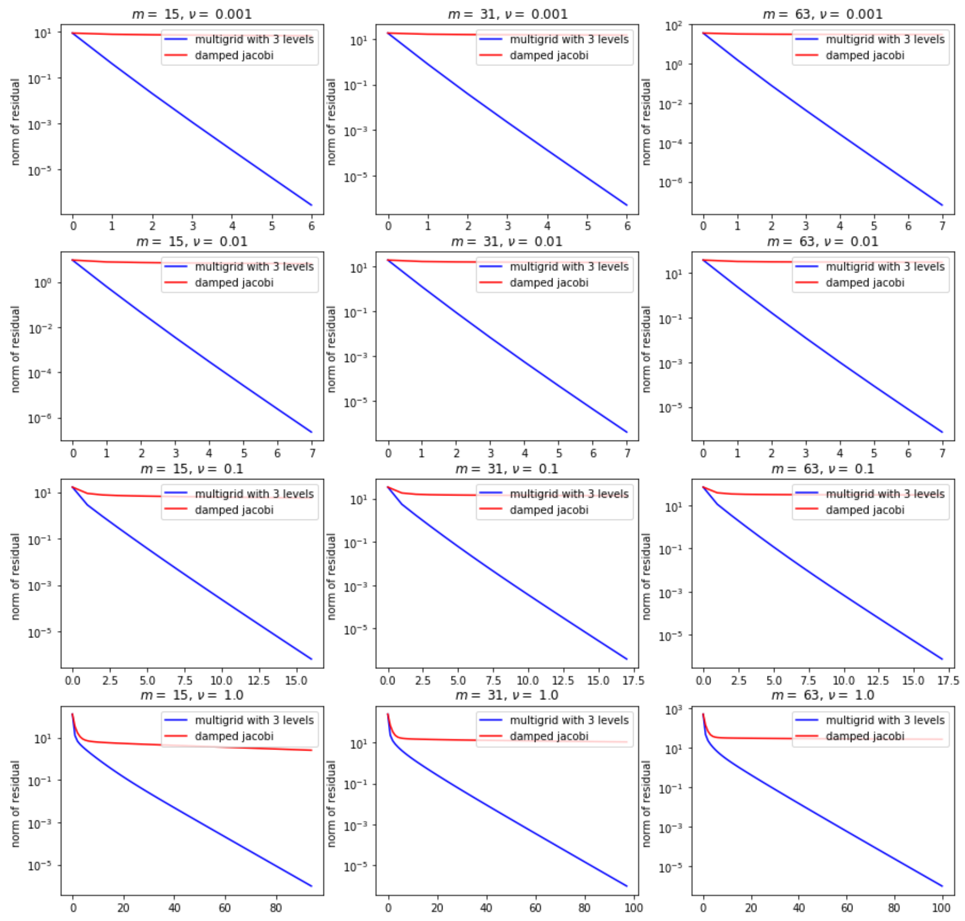

FIGURE 8. Number of iterations of multigrid in comparison with damped jacobi for different ν and m