Computational Physics - Problem Sheet 4

Christian Gorjaew (354259)
Julius Meyer-Ohlendorf (355733)

1 Relaxation methods for the Laplace equation

The code for this task can be found in the file "cp_ex04_p1.py". The relaxation methods are implemented as functions which return the relaxed grids. Each function takes an optional third parameter, which specifies the desired number of iterations N_{iter}.
The theoretical number of iterations $N_{\text{iter}}^{\text{theo}}$ required to decrease the error by a factor of 10^{-p} was derived in lecture 5, slide 48 and 53 for the Jacobi and Gauss-Seidel method respectively and in lecture 6, slide 10 for the successive overrelaxation method (SOR). A 2D square lattice with $N_{\text{S}} = N_{\text{l}} \times N_{\text{l}}$ and $N_{\text{l}} \to \infty$ was assumed. Resulting in:

$$N_{\text{iterJac}}^{\text{theo}} = \frac{1}{2} p N_{\text{S}} \tag{1}$$

$$N_{\text{iterGauS}}^{\text{theo}} = \frac{1}{4} p N_{\text{S}} \tag{2}$$

$$N_{\text{iterSOR}}^{\text{theo}} = \frac{1}{3} p \sqrt{N_{\text{S}}} \tag{3}$$

The following table shows our obtained number of iterations N_{iter} and the corresponding theoretical values $N_{\text{iter}}^{\text{theo}}$, required to reach an accuracy of $\delta\Phi = 10^{-5}$, with $N_{\text{l}} = 81$.

Method	Jacobi	Gauss-Seidel	SOR
$N_{\textbf{iter}}$	5948	3425	154
$N_{\textbf{iter}}^{\textbf{theo}}$	16402	8201	135

Table 1: Required number of iterations N_{iter} and corresponding $N_{\text{iter}}^{\text{theo}}$ for an accuracy of 10^{-5} with $N_{\text{l}} = 81$.

Our obtained values are not consistent with the theoretical values, especially for the Jacobi and Gauss-Seidel method the obtained values are roughly smaller by a factor of $\frac{1}{2}$. The fact that the theoretical values were derived assuming $N_{\text{l}} \to \infty$ could be a possible explanation for the large deviations.

All relaxation schemes were started with the initial discretized grid Φ_0 which can be seen in Figure 1.

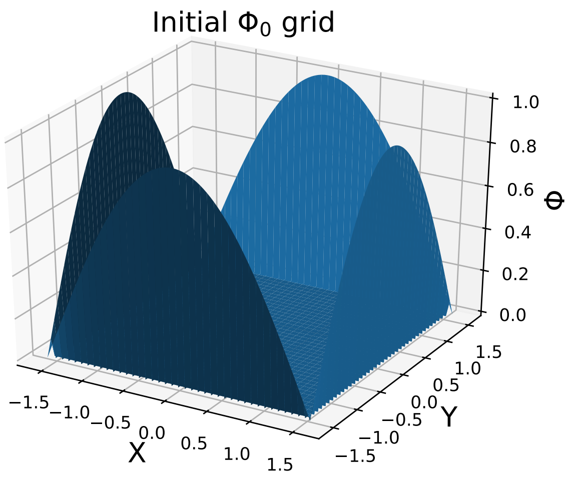
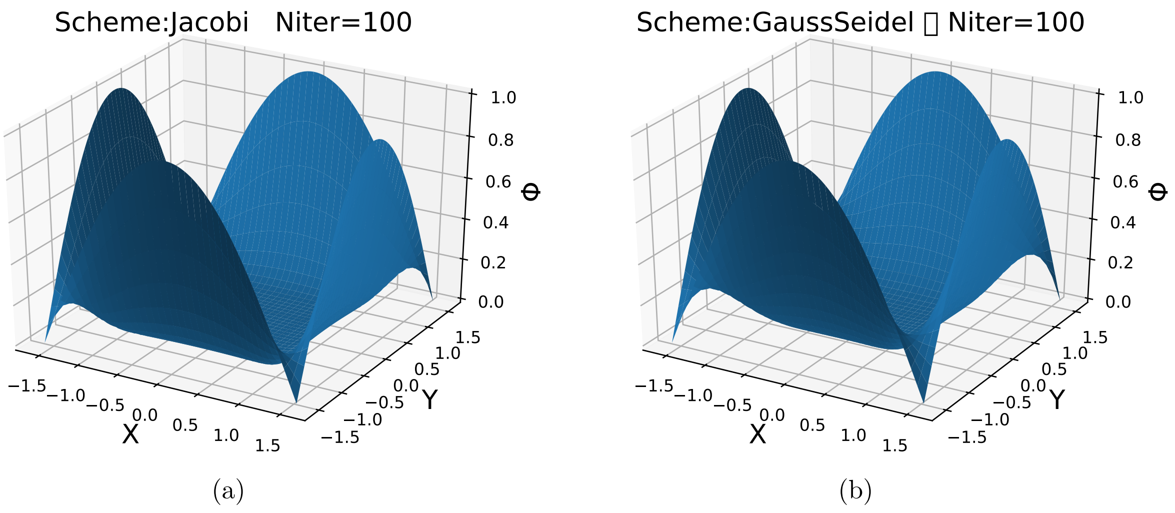


Figure 1: Initial Φ_0 grid which is taken as the initial value for the relaxation schemes.

The relaxed Φ grids after $N_{\text{iter}} = 100$ are shown in Figure 2.



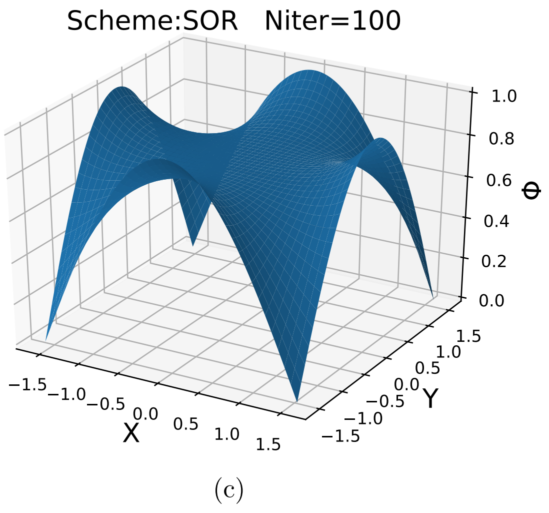
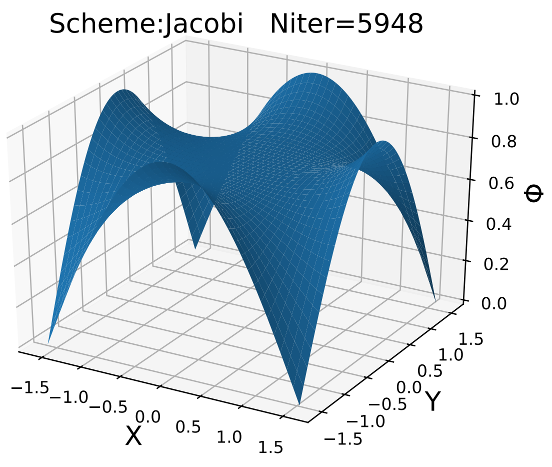
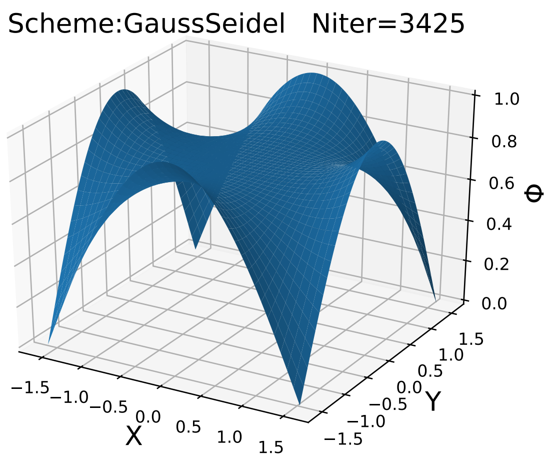


Figure 2: Φ grid after $N_{\text{iter}} = 100$. Jacobi method (a), Gauss-Seidel method (b), SOR method with Chebyshev acceleration (c).

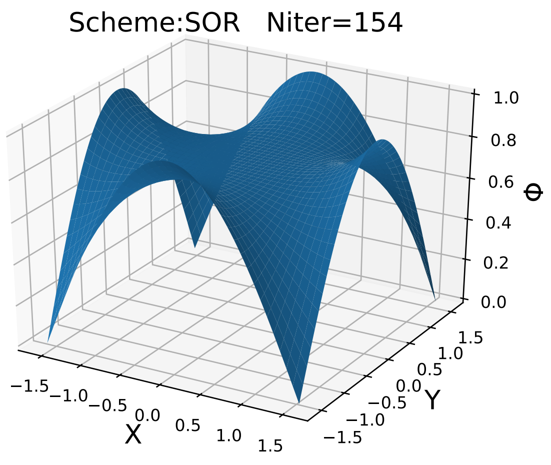
The relaxed grids show the expected behaviour, i.e. the SOR method converges faster than the Gauss-Seidel method, which again converges faster than the Jacobi method.
Furthermore, to get a better idea of the relaxed grids, Figure 3 shows the relaxed grids for all three methods when an accuracy of $\delta\Phi = 10^{-5}$ is reached.



(a)



(b)



(c)

Figure 3: Relaxed grids for all three methods when an accuracy of $\delta\Phi = 10^{-5}$ is reached. Jacobi with $N_{\text{iter}} = 5948$ (a), Gauss-Seidel with $N_{\text{iter}} = 3425$ (b), SOR with Chebyshev acceleration with $N_{\text{iter}} = 154$ (c).

2 Conjugate-gradient method

The code for this task can be found in the file "cp_ex04_p2.py".

Part a) The code for part a) is given in ll. 1-129. The steepest descent (SD) and conjugate-gradient (CG) methods are implemented according to the schemes presented in lecture 6. Apart from the function arguments that were required by the task, the implemented CG function takes an argument "multA" and "max_iter". Former allows to overwrite the matrix multiplication routine with foresight on part b) of this task. In case none is given, conventional matrix multiplication using the "@"-operator in Python is employed which performs multiplication between arrays following the rules of matrix multiplication (instead of an element-wise multiplication). The second additional argument, "max_iter" sets the maximum number of iterations.

The testing results for both methods employing the given matrix A and $\vec{x}_{init} = \vec{b} = (1, ..., 1)^T$ can be seen in Table 2 where $\vec{x}_{init}$ is the starting point of the minimization.

Method	x_0	$\lvert\vec{x}\rvert$	Iterations
SD	43.883667667	157.798913107	95072
CG	43.883667671	157.798913122	12

Table 2: Test results of the implemented steepest descent (SD) and conjugate-gradient (CG) methods for the given matrix ("CG_Matrix_10x10.dat") and $\vec{b} = (1, ...1)^T$. The initial vector $\vec{x}_{init}$ was set to $\vec{b}$.

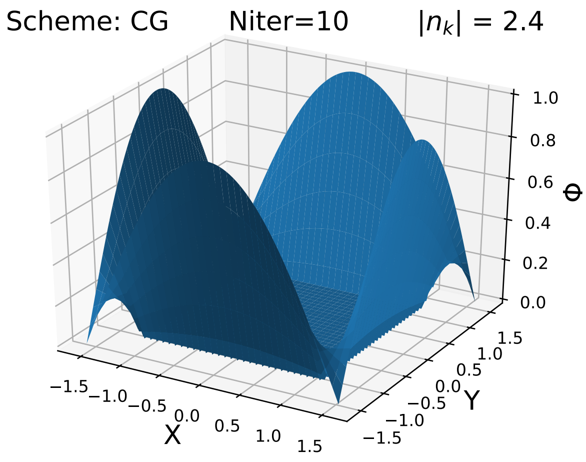
The deviation of x_0 and $\lvert\vec{x}\rvert$ between the two methods occurs at order 10^{-8}. One also sees that number of iterations, at which convergence with respect to the desired threshold is reached, is significantly smaller for the CG method.

Part b) The code for this part is given in ll. 130-270. In order to exploit the sparsity of the matrix that describes the Laplace equation, the matrix multiplication routine is overwritten by a function "multA" as instructed. Here, only those components of the input vector are considered that would be multiplied with non-zero matrix elements in the matrix multiplication. The if-statements inside the loop take care of the correct calculation of components that correspond to lattice sites adjacent to the grid boundary and thus, deviate from the regular addition scheme (c.f. Mazzarello, lecture notes 5, page 32). In particular, the method implements multiplication of the input vector with the following matrix[1]

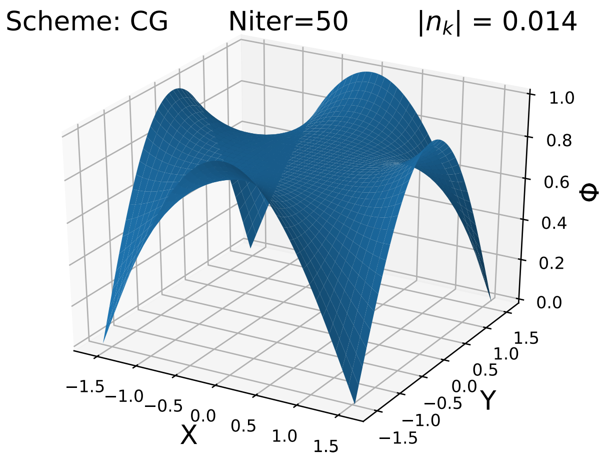
$$A = \begin{bmatrix} D & -I & 0 & 0 & 0 & \dots & 0 \\ -I & D & -I & 0 & 0 & \dots & 0 \\ 0 & -I & D & -I & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -I & D & -I & 0 \\ 0 & \dots & \dots & 0 & -I & D & -I \\ 0 & \dots & \dots & \dots & 0 & -I & D \end{bmatrix} \quad \text{where} \quad D = \begin{bmatrix} 4 & -1 & 0 & 0 & 0 & \dots & 0 \\ -1 & 4 & -1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 4 & -1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 4 & -1 & 0 \\ 0 & \dots & \dots & 0 & -1 & 4 & -1 \\ 0 & \dots & \dots & \dots & 0 & -1 & 4 \end{bmatrix}$$

Here D is a $(N-2) \times (N-2)$ matrix, where $N = N_x = N_y$ is the number of grid points in x and y direction. I is the identity matrix of same dimension as D. Furthermore, in case of the Laplace equation the vector $\vec{b}$ contains the boundary points only.

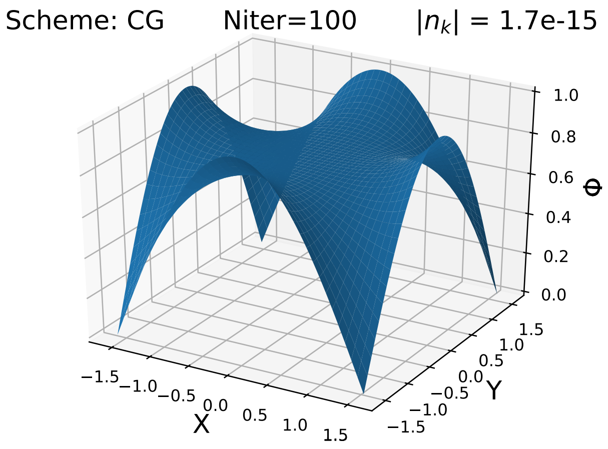
[1]c.f., https://en.wikipedia.org/wiki/Discrete_Poisson_equationOn_a_two-dimensional_rectangular_grid



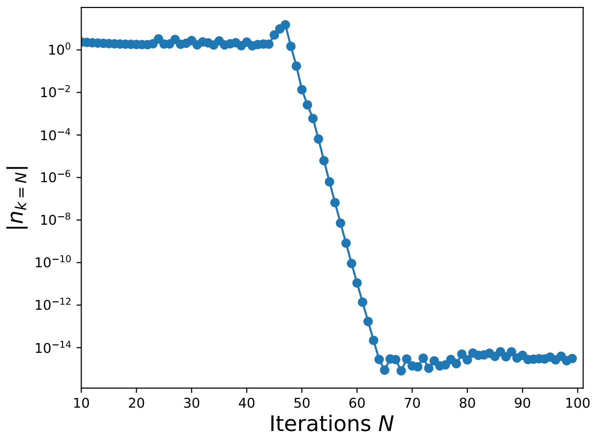
(a)



(b)



(c)



(d)

Figure 4: Grid after 10 (a), 50 (b) and 100 (c) CG iterations. In (d), the accuracy in terms of $|n_k|$ versus number of iterations is shown. n_k is the Euclidean norm of the direction vector $\vec{n_k}$ employed in the last iteration.

With this method, the desired accuracy of $|n_k| = \varepsilon = 10^{-5}$ was reached after 54 CG iteration. The intermediate grid at the requested values can be seen in Fig. 4. After 10 iterations, the grid has not changed significantly when compared with the initial grid (c.f., Fig. 1). A visual distinction between the grid after 50 CG iterations and the relaxed grids in Fig. 3 is not possible. Same holds for the grid after 100 CG iterations. As can be seen in Fig. 4(d), the magnitude of the directional vector n_k decreases exponentially after approximately 50 iterations. After approximately 65 iteration it saturates at around 10^{-15} which indicates that float precision is limiting the maximum achievable accuracy.