Eric Näser
ID: 202300343

Report Assignment 1

Task 1 - Problem

Which task was given?

GitHub Repository:

https://github.com/eric-official/advanced-numerical-methods

Solve 2D Heat Conduction Equation

In the first task, the 2D heat conduction equation (Laplace equation) has to be solved. The following details were provided:

- **Length and width** were given with L = W = 1
- The **general equation** of the heat conduction equation was stated as:

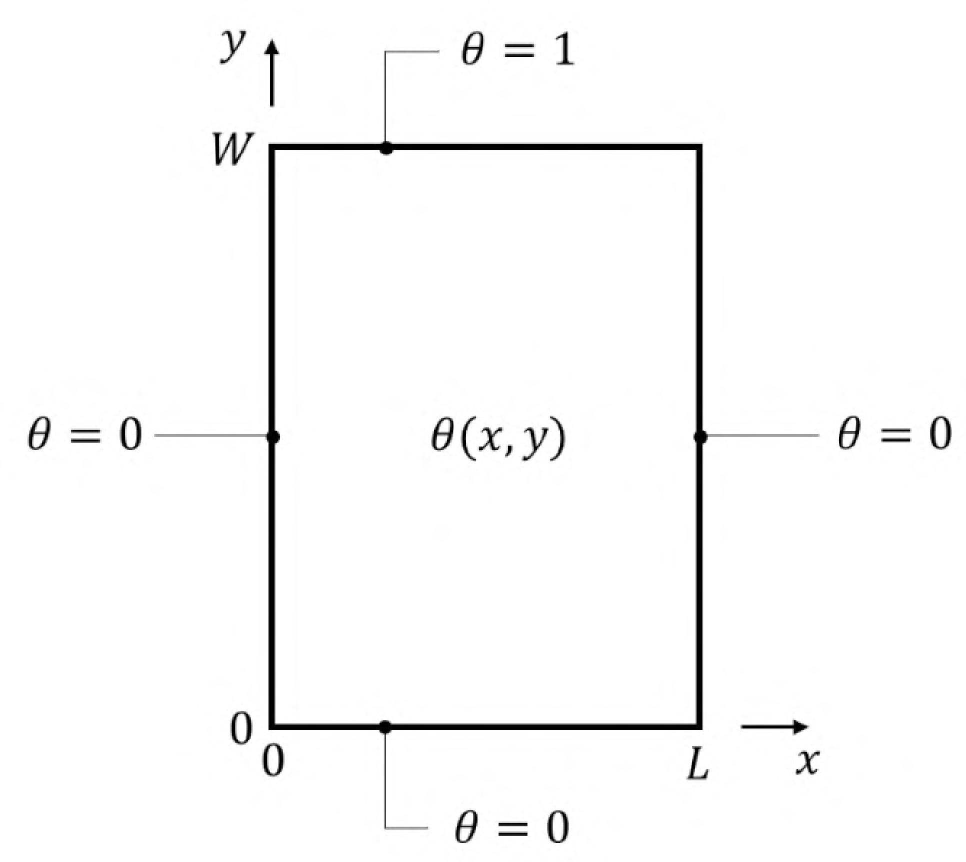
$$\frac{\partial^2\theta}{\partial x^2}+\frac{\partial^2\theta}{\partial y^2}=0$$

- The **boundary conditions** were specified with the following values:

$$\begin{cases}\theta(0,y)=0\\ \theta(L,y)=0\\ \theta(x,0)=0\\ \theta(x,W)=1\end{cases}$$

Illustration of Problem

The given problem can illustrated in the following diagram:



Key Deliverables

To analyse the developed solution, the following deliverables should be created:

- For each solution of the matrix solver, a **contour plot** has to be drawn
- Furthermore, for each solution of the matrix, a **centerline diagram** has to be plotted
- The results of different **matrix solvers and different grid densities** had to be compared
- For the following exact solution, the **error for different grid densities** has to be analysed

$$\theta(x,y)=\frac{2}{\pi}\sum_{n=1}^{\infty}\frac{(-1)^{n+1}+1}{n}\sin\left(\frac{n\pi x}{L}\right)\frac{\sinh(n\pi y/L)}{\sinh(n\pi W/L)}$$

Task 1 - Numerical Methods

Which numerical methods did I utilize to achieve the first task?

GitHub Repository:

https://github.com/eric-official/advanced-numerical-methods

Discretization Scheme

Discretization is done using the **finite difference method**, a numerical method for solving differential equations by approximating derivatives with finite differences. The finite difference method results in numerous ordinary differential equations where one equation represents the dynamical behaviour of a single quantity at a specific location in the space domain.

For approximation, the **central difference method** is used. This method utilizes the slope of a line passing through two points lying on opposite sides of the point at which the derivative is approximated.

Matrix Solvers

For solving the 2D Heat Conduction Equation, three matrix solvers have been implemented. Those are namely **Jacobi, Gauss-Seidel and Successive over-relaxation**. For reasons of space and clarity, the analysis of Jacobi has been omitted. The relaxation factor omega for Successive over-relaxation got defined with a value of 1.5. The discretized equation with the central differencing scheme for Gauss-Seidel and Successive over-relaxation is implemented in Python.

Grid Generation

For representing the space domain and calculating temperature changes respectively, a 2D uniform grid has been generated. To analyse the results with respect to the grid density, the number of points per dimension is set to 20 and 50. The Python code for grid generation with the given boundary values looks as follows:

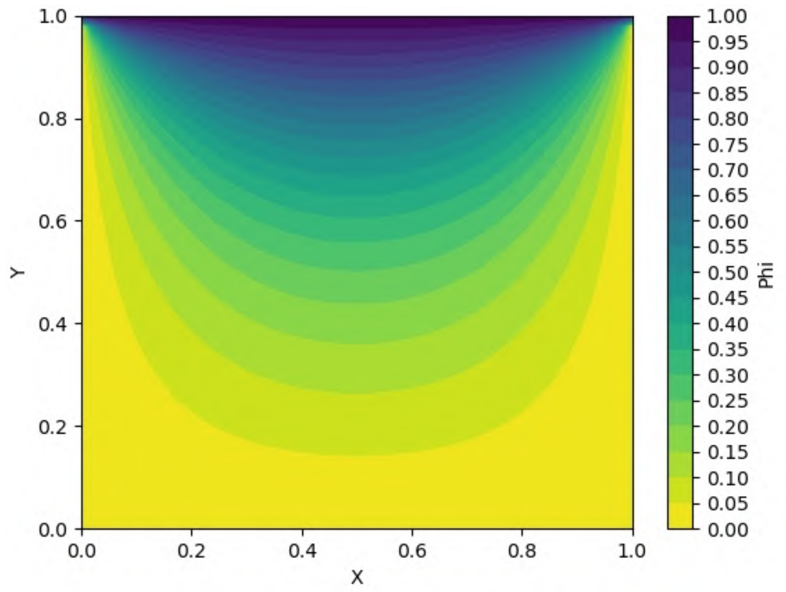
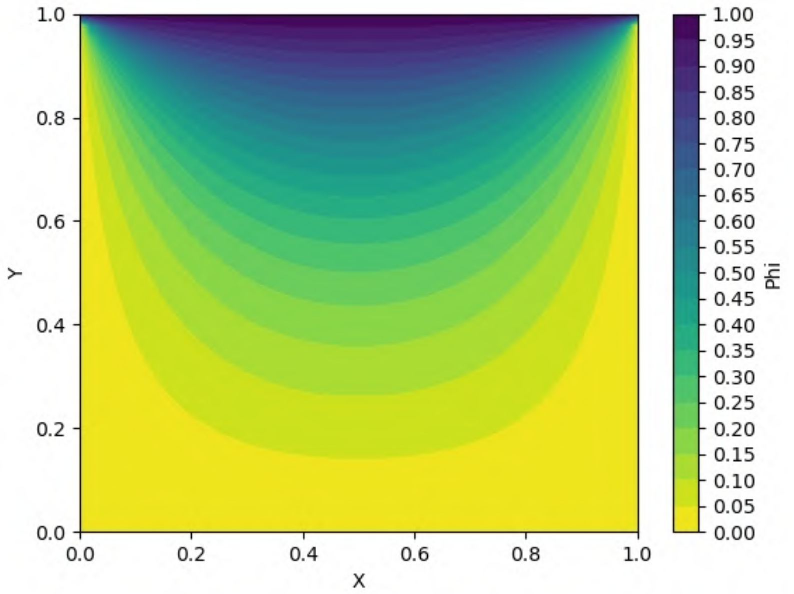
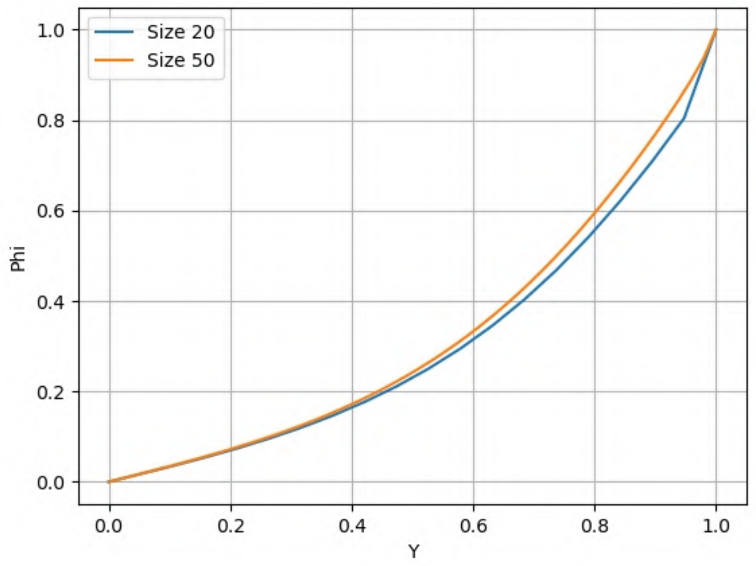
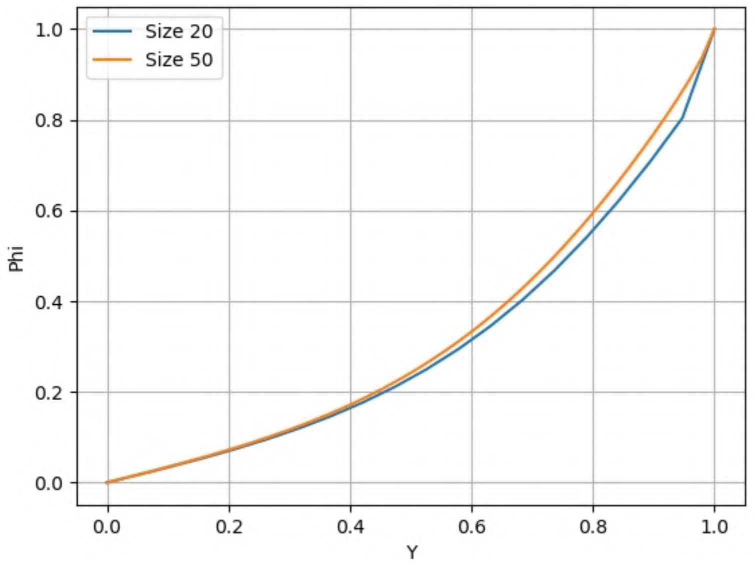
```
T = np.zeros((nx, ny))

# Boundary conditions set-up
T[:, 0] = left_t
T[:, ny - 1] = right_t

T[0, :] = lower_t
T[nx - 1, :] = upper_t
```

Task 1 - Results and Discussion

Which results have I achieved with my solution?

GitHub Repository:

https://github.com/eric-official/advanced-numerical-methods

	Gauss-Seidel	Successive over-relaxation	Discussion
Contour plot with 50 mesh points per dimension			For both matrix solvers, the highest temperatures are at the top of the domain while the lowest temperatures are at the bottom. This suggests that there is a heat source on the top of the domain, as expected by the given boundary conditions. The temperature gradients in the diagram are also non-uniform. The largest temperature gradients are found near the heat source while the gradients become less steep as the distance from the heat source increases.
Centerline			The previously made interpretation of the gradient is also visible in the centerline plots. Here it can be easily seen that for lower y-values at the bottom side of the domain, the gradient is smaller than at the top side. Furthermore, the temperature distribution on the grid with 50 points per dimension is smoother than the temperature distribution on the grid with 20 points per dimension due to the finer solution of the grid.

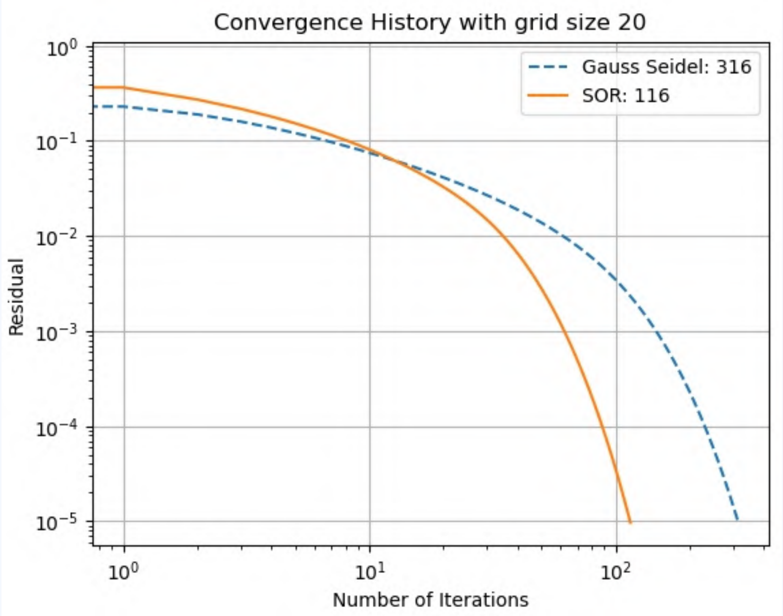
Task 1 - Results and Discussion

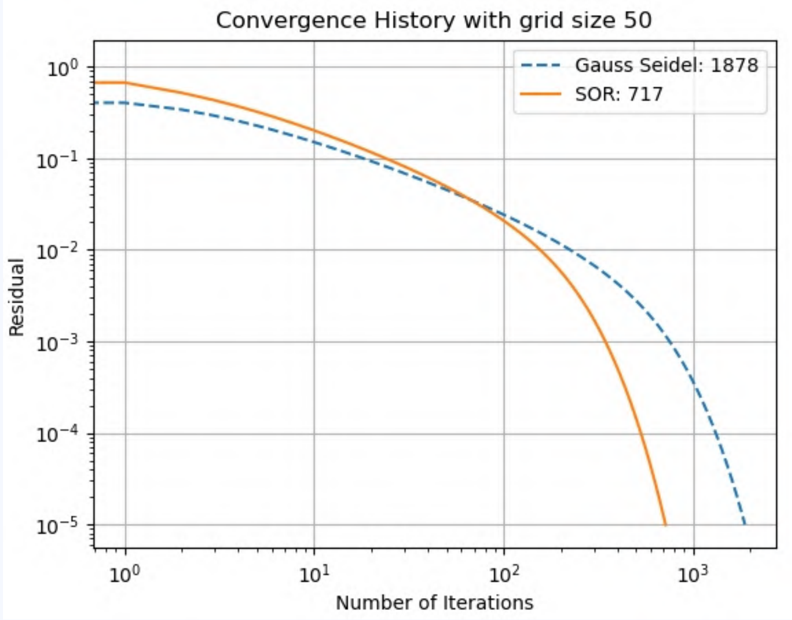
Which results have I achieved with my solution?

GitHub Repository:

https://github.com/eric-official/advanced-numerical-methods

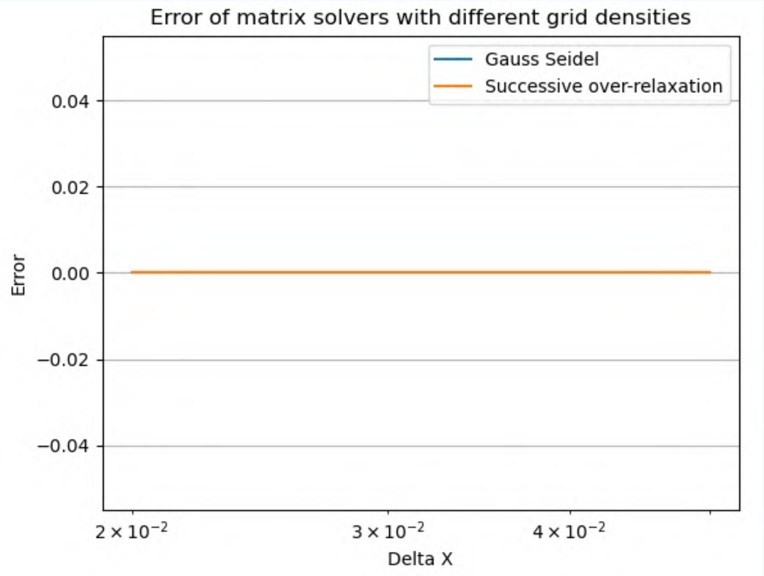
Error Convergence History





To compare the performance of the used matrix solvers Gauss-Seidel and Successive over-relaxation, the error convergence history has been plotted. During the calculation of the matrix solvers, the absolute error between the value of the previous and the current iteration has been calculated. When the error reaches a value of 1e-5, the matrix solver stops. The diagrams above show the error convergence of both matrix solvers with 20 and 50 points per grid dimension. It can be seen that Successive over-relaxation converges faster than Gauss-Seidel. While Successive over-relaxation only needs 116 and 717 iterations, Gauss-Seidel needs 316 and 1878 iterations respectively.

Comparison with exact solution



For comparing the results with respect to different grid densities the mean absolute error between the exact and solved solution has been calculated. The exact solution and the solved solution with the matrix solver are equal across all analyzed grid densities. For the grids with 20 points per dimension as well as 50 points per dimension, the error between the exact and solved solution is always zero. To show that the error has not been determined zero incorrectly because of the y-axis scale in the diagram above, the numbers from the error calculation are printed. The following picture shows values from error calculation with the exact value, the solved value and the error between the values from left to right.

```
0.799255946804796 0.799255946804796 0.0
0.7897042634698358 0.7897042634698358 0.0
0.7800819575578041 0.7800819575578041 0.0
```

Task 2 - Problem

Which task was given?

GitHub Repository:

https://github.com/eric-official/advanced-numerical-methods

Solve Convection-Diffusion Problem

In the second task, the convection-diffusion problem has to be solved.

- **Length and width** were given with L = W = 1 as well as the **velocity components** u = x and v = -y
- The **general equation** of the convection diffusion equation was stated as:

$$\frac{\partial(\rho\phi)}{\partial t}+\frac{\partial(\rho u\phi)}{\partial x}+\frac{\partial(\rho v\phi)}{\partial y}=\frac{\partial}{\partial x}\left(\Gamma\frac{\partial\phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(\Gamma\frac{\partial\phi}{\partial y}\right)$$

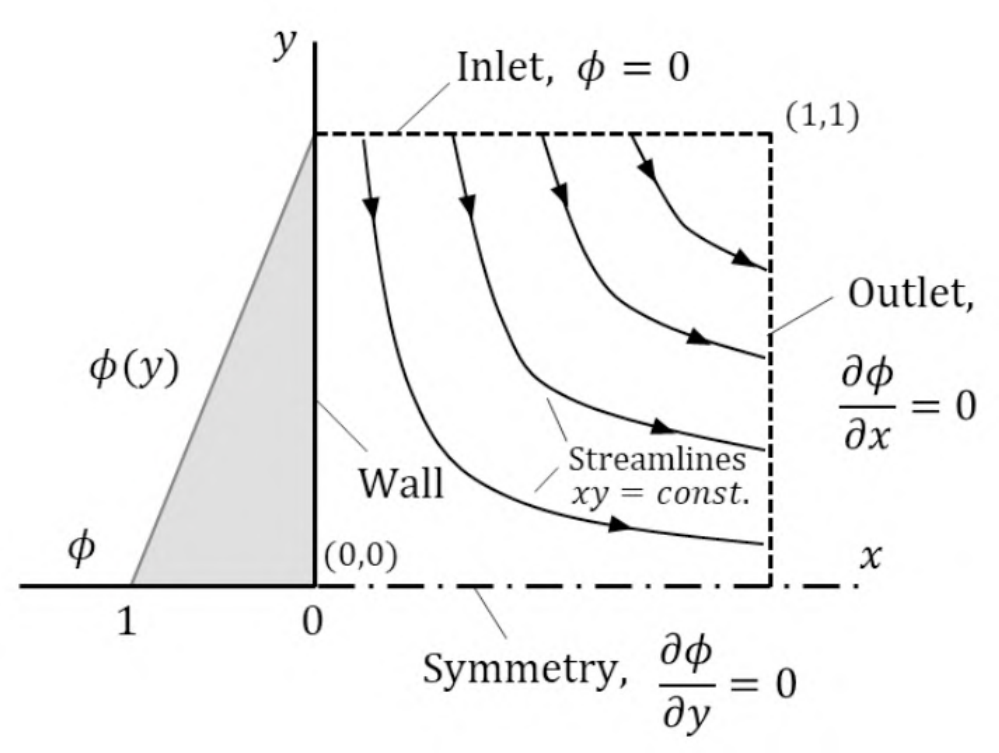
- For the initial condition Phi = 0 and t = 0, **boundary conditions** were specified with:

$$\begin{cases} \phi\big|_{x=0}=1-y & \frac{\partial\phi}{\partial x}\Big|_{x=1}=0 \\ \frac{\partial\phi}{\partial y}\Big|_{y=0}=0 & \phi\big|_{y=1}=0 \end{cases}$$

- Simulations for the steady case with Rho = 1.0 and Gamma = 0.01 or 0..001 as well as transient case with Rho = 1.2 and Gamma = 0.1

Illustration of Problem

The given problem can illustrated in the following diagram:



Key Deliverables

To analyse the developed solution, the following deliverables should be created:

- For each solution of the matrix solver, a **contour plot** has to be drawn
- Furthermore, at least **two unsteady and two unsteady solvers** have to be used
- The results of different **matrix solvers** have to be compared
- The **error has to be estimated** by using the finest grid

Task 2 - Numerical Methods

Which numerical methods did I utilize to achieve the second task?

GitHub Repository:

https://github.com/eric-official/advanced-numerical-methods

Discretization Scheme

Discretization is done using the **finite volume method**, a numerical approach for solving differential equations by quantifying the flow of quantities through control volumes. In the finite volume method, the system yields multiple conservation equations, with each equation characterizing the accumulation and transport of a specific quantity within an individual control volume.

For approximation, the **upwind scheme** is used. This method approximates the transport of quantities in a specific direction.

Matrix Solvers

For solving the 2D Heat Conduction Equation, two matrix solvers for the steady equation and the transient equation have been implemented. Those are namely **Gauss-Seidel, Successive over-relaxation, Explicit Euler and Implicit Euler**. For Successive over-relaxation, the relaxation factor omega got defined with a value of 1.1. The discretized equation with the upwind differencing scheme for the matrix solvers is implemented in Python.

Grid Generation

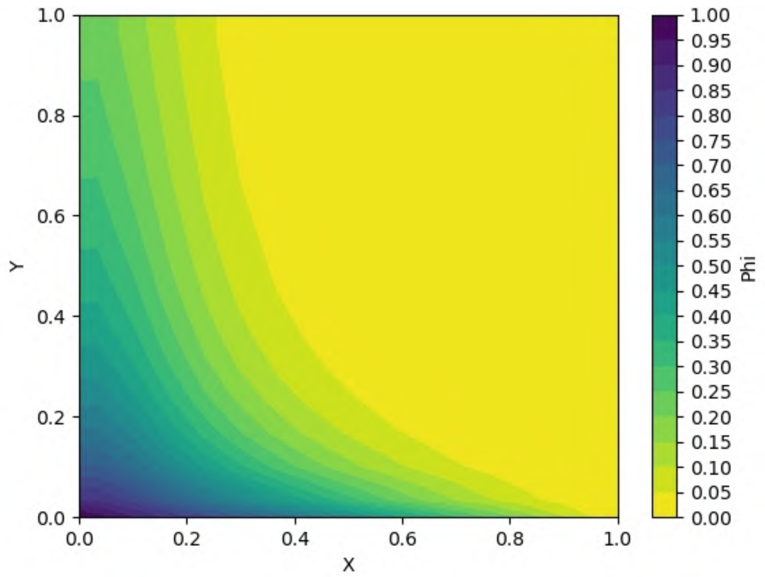
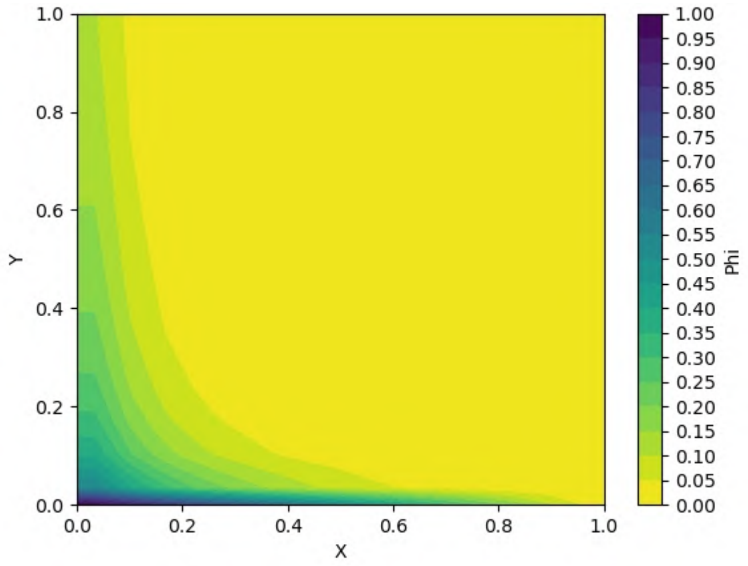
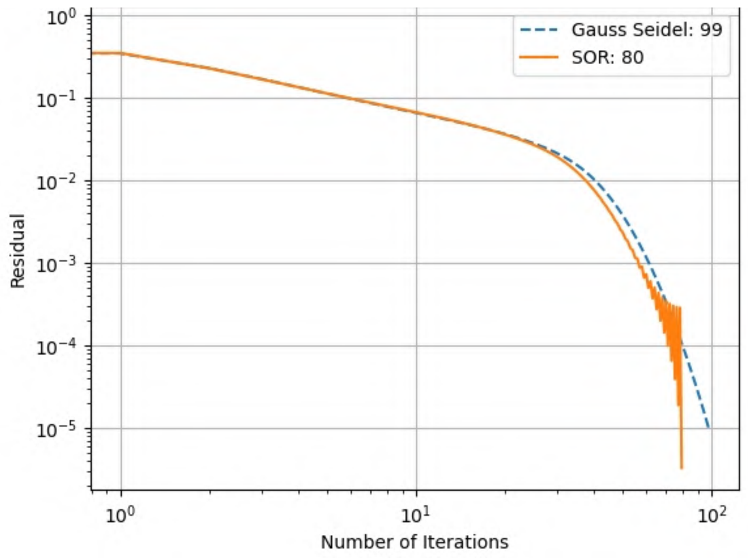
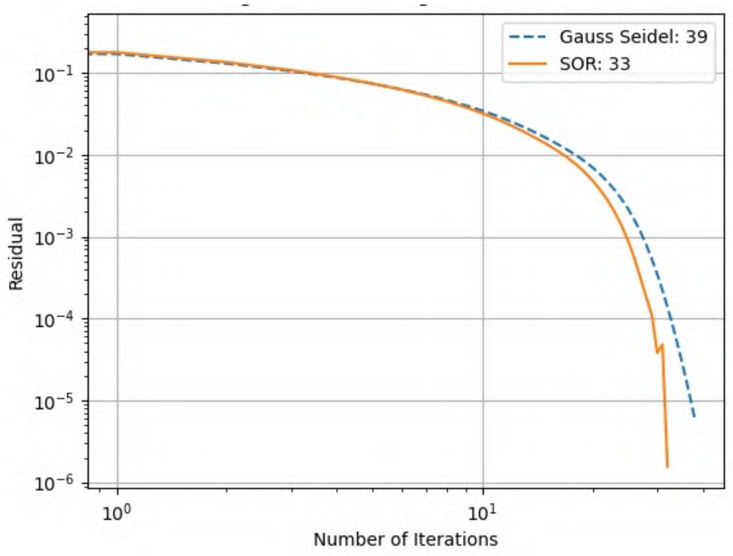
For representing the space domain and calculating temperature changes respectively, a 2D uniform grid has been generated. To analyse the results with respect to the grid density, the number of points per dimension is set to 5, 10 and 15. For solving the transient convection-diffusion equation, the unsteady matrix solvers iterated until time step 1000.

Task 2 - Results and Discussion

Which results have I achieved with my solution?

GitHub Repository:

https://github.com/eric-official/advanced-numerical-methods

	Gamma = 0.01	Gamma = 0.001	Discussion
Contour plot of Gauss-Seidel with 15 mesh points per dimension			For both matrix solvers and diffusion coefficients gamma, the highest temperatures are at the bottom left corner of the domain. This suggests that there is a heat source in the left corner of the domain, as expected by the given boundary conditions. With decreasing values for gamma, the temperature diffuses less through the domain. The temperature gradients in the diagram are also non-uniform, similar to task 1.
Error Convergence History for Steady Equation			To compare the performance of the used matrix solvers Gauss-Seidel and Successive over-relaxation (SOR), the error convergence history has been plotted. The diagrams to the left show the error convergence of both matrix solvers with 15 points per dimension. Again, it can be seen that SOR converges faster than Gauss-Seidel. Hereby it is important to mention, that SOR behaved instable and had no ability to converge for larger values of omega than 1.1.

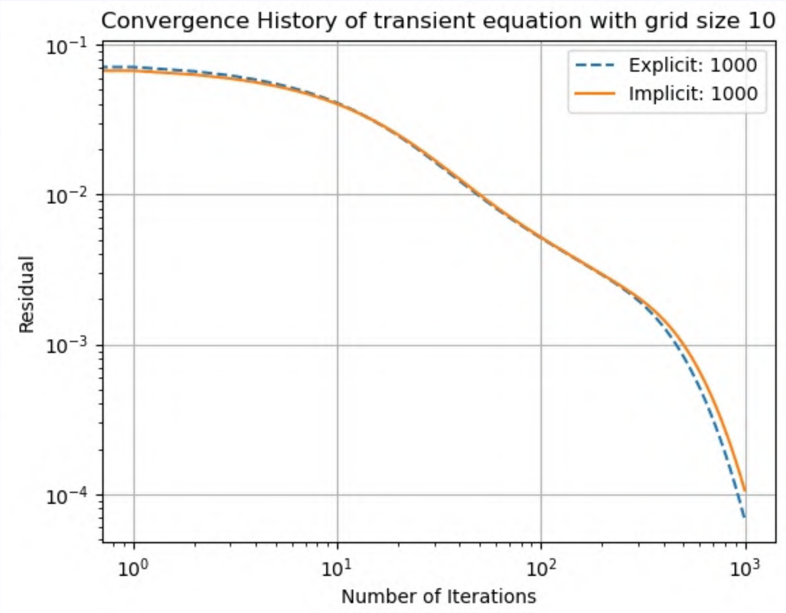
Task 2 - Results and Discussion

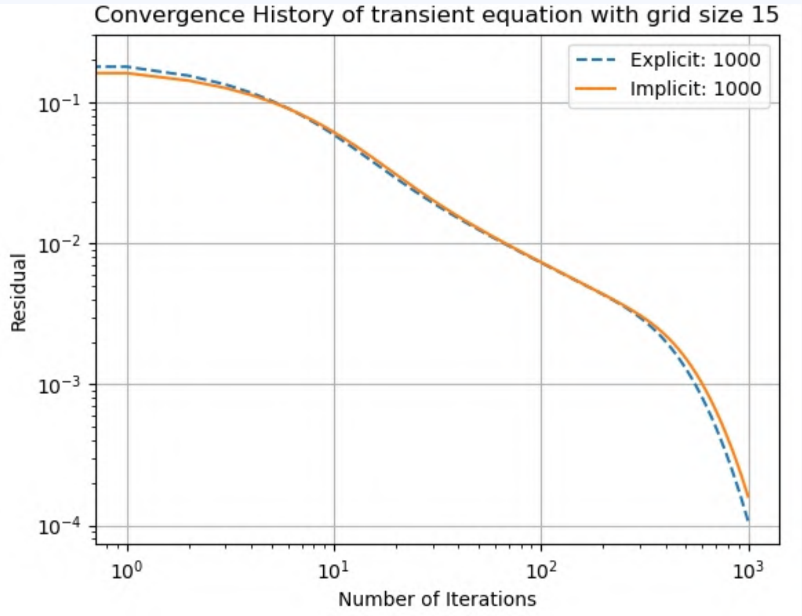
Which results have I achieved with my solution?

GitHub Repository:

https://github.com/eric-official/advanced-numerical-methods

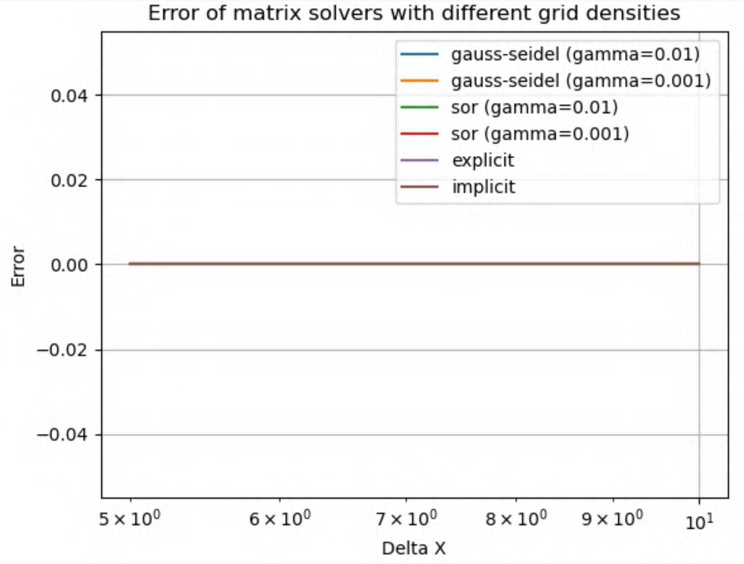
Error Convergence History for Transient Equation





To compare the performance of the used matrix solvers Explicit Euler and Implicit Euler , the error convergence history has been plotted. During the calculation of the matrix solvers, the absolute error between the value of the previous and the current iteration has been calculated. When the time step reaches a value of 1000, the matrix solver stops. The diagrams above show the error convergence of both matrix solvers with 10 and 15 points per grid dimension. In the error convergence plots of the two grid sizes it can be seen that there is no significant advantage of one matrix solver visible.

Comparison with Finest Grid



For comparing the results with respect to different grid densities, the mean absolute error between the finest grid with 15 points per dimension and the other grids has been calculated. To achieve this, all grid indices that yield the same x or y value were determined for the finest and the compared grid. This procedure was done for every used matrix solver. For the grids with 5 points per dimension as well as 10 points per dimension, the error to the finest is always zero. For the steady matrix solvers, the error of zero can be explained by the error threshold of 1e-5 which always stops the calculation once it is reached. For the unsteady matrix solvers, one reason might be the high number of 1000 as a maximum time step.

References

Which books, documents and websites have I used to solve the homework?

[1]J. H. Ferziger and M. Perić, *Computational Methods for Fluid Dynamics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. doi: 10.1007/978-3-642-56026-2., p. 31-78

[2]T. A. Beu, J. Adler, K. Roos, and J. Driscoll, 'INTRODUCTION TO NUMERICAL PROGRAMMING: A Practical Guide for Scientists and Engineers Using Python and C/C++'., p. 209-231

[3]'Iterative Methods for Solving Linear Systems of Equations'. Accessed: Oct. 31, 2023. [Online]. Available: https://johnfoster.pge.utexas.edu/numerical-methods-book/LinearAlgebra_IterativeSolvers.html

[4]T. Steinbacher, K. Förner, and Polifke Wolfgang, 'Numercial Thermo-Fluids'. 2022., p. 36-70

[5]J. Kiusalaas, 'Numerical Methods in Engineering with Python 3'., p.87-104

[6]'Python Programming And Numerical Methods: A Guide For Engineers And Scientists — Python Numerical Methods'. Accessed: Oct. 31, 2023. [Online]. Available: https://pythonnumericalmethods.berkeley.edu/notebooks/Index.html

[7]'Solving 2D Heat Equation Numerically using Python | Level Up Coding'. Accessed: Oct. 31, 2023. [Online]. Available: https://levelup.gitconnected.com/solving-2d-heat-equation-numerically-using-python-3334004aa01a

[8]Skill-Lync, 'Solving the steady and unsteady 2D heat conduction equations.', Skill-Lync. Accessed: Oct. 31, 2023. [Online]. Available: https://skill-lync.com/student-projects/week-5-mid-term-project-solving-the-steady-and-unsteady-2d-heat-conduction-problem-26