Short Training Program Report

Implementation of Implicit Solution Techniques for Non-equilibrium Hypersonic Flows

Julian Koellermeier
RWTH Aachen University

Supervisor: Prof. Thierry Magin
von Karman Institute for Fluid Dynamics

Advisor: Michael Kapper PhD
von Karman Institute for Fluid Dynamics

Abstract

This work presents an implicit framework for the solution of partial differential equations of elliptic type. The discretization scheme uses a dual time stepping method to converge to the next physical time level in order to capture non-linearity and make use of steady-state techniques. For the solution of the large linear system of equations we employ a restarted GMRES method combined with a Gauss-Seidel preconditioner and overrelaxation.

We obtain a substantial speedup resulting from the application of the methods mentioned above. We also investigate the effect of different parameter choices on the convergence behavior and runtime of the solution process. The additional improvements lead to reductions in both iteration numbers and runtime.

Contents

List of Figures

List of Tables

List of Symbols

Acronyms

CFD	Computational Fluid Dynamics
DTS	Dual Time Stepping
EFD	Efficient Finite Differences
FD	Finite Differences
GMRES	Generalized Minimal Residual Method
GMRES(m)	restarted GMRES
GS	Gauss-Seidel Method
IC	Initial Condition
JFM	Jacobian Free Method
LLS	Linear Least Squares
LSE	Linear System of Equations
NSE	Navier-Stokes Equations
PDE	Partial Differential Equation
RHS	Right Hand Side
SOR	Successice Over-Relaxation

Sub- and Superscripts

i	face index
k	dual time iteration index
L	value at left cell of the current face
n	physical time iteration index
R	value at right cell of the current face
x	in x direction
y	in y direction

Symbols

T	temperature
t	physical time
q	heat flux
∇	gradient of a function
κ	heat conductivity
ω	parameter in non-linear heat conductivity model
u	x-component of velocity
v	y-component of velocity
Φ	velocity potential
Δ	Laplacian of a function
ρ	density
p	pressure
τ_{ij}	entries of stress tensor
E	total energy
η	viscosity
λ	thermal conductivity
Q	vector of flow variables
$F(Q)$	flux function
$L(Q)$	elliptic or diffusive RHS operator
$a_{i,j}$	coefficients of LLS reconstruction at face i
M, N, C, D	matrices for LLS
τ	dual time
$R(Q)$	discretized RHS operator
τ	dual time
$\Delta\tau$	dual time step
Δt	physical time step
$\overline{R}$	discretized RHS including time derivative term
A	matrix for the LSE
b	RHS for the LSE
P	preconditioning matrix
D, L	diagonal and lower part of matix M
ω	overrelaxation parameter for GS
Kr_k	Krylov subspace of dimension k
r	residual in linear solver
H_k	Hessenberg matrix inside GMRES
Q_k	matrix of orthogonal basis vectors of Kr_k
$n_{max,GMRES}$	maximum number of restarts
m	maximum dimension of the Krylov subspace
$n_{max,GS}$	maximum number of Gauss-Seidel iterations inside
eps	threshold for the residual inside GMRES(m)
f	arbitrary function for FD
ϵ	step size for FD
e_j	j-th unit vector
v	vector for Jacobian free method
$\kappa(A)$	condition number of matrix A
$\|\cdot\|_2$	euclidian 2-norm

Chapter 1

Introduction

1.1 Numerical Simulations

Modern engineering applications are more and more influenced by numerical simulations. During the past decades the development of efficient numerical codes together with major advances in computer architecture made it possible to simulate more complex models with a very high resolution.

The upcoming field of Computational Fluid Dynamics (CFD) has provided many dedicated algorithms to solve very specific problems as well as more general types of equations.

Though computer simulations are very powerful, they usually go hand in hand with experimental research because the calculated results needs to be verified using real experiments. This is crucial because many of the numerical concepts rely on mathematical properties that do normally not describe the entire physical characteristics of the problem. After verification and validation of the simulation, the calculations can be extended to other problems or parameter regions were no experiments are available to obtain completely new results.

In order to do so, researchers all over the world develop general algorithms that can be used in a general framework afterwards to solve many different problems. In this context there is a large interest in the investigation of already existing algorithms under different circumstances. That is also one of the aims of this project.

1.2 Aims of the project

The project builds up on an existing software framework which was developed by Kapper (see [3]). It was designed for simulations of different physical models using various techniques and successfully tested before. The aim is now to implement an implicit solution scheme for the simulation of fluid dynamic equations, especially for diffusion-type equations.

While trying to keep the framework as general as possible to allow user-friendly addition of new physical models, we want to employ different numerical methods to

come up with an efficient solution of the emerging equation system.

The concrete implementation will include an implicit numerical scheme which is time accurate and uses a dual time stepping scheme (DTS). For the solution of the large system of equations we will make use of the preconditioned GMRES method. The convergence will be accelerated using Gauss-Seidel preconditioning and over-relaxation, also called successive over-relaxation (SOR). In a parameter study we will later find out optimal values for the parameters of the linear solver as well as for the preconditioner to ensure a good convergence behavior.

In further work the implicit method implemented will be applied to more complex hydrodynamic models such as the Boltzmann moment systems with a Grad closure following the regularization proposed by Struchtrup and Torrilhon.

Chapter 2

Physical Models

In general, we want to cover diffusion-type equations. But with the help of the operator splitting technique (see next chapter) it is also possible to simulate more arbitrary models. Therefore we will not only describe elliptic equations but also the well known Navier-Stokes equations which contain convective terms. We start with a more simple one-dimensional equation.

2.1 Heat Equation

We consider the two-dimensional heat equation on a non-regular grid. Therefore the problem equation reads

$$\frac{\partial T}{\partial t} + \nabla (q) = 0 \tag{2.1}$$

The heat flux q itself is modelled by *Fourier's law*, saying that the heat flux through a surface is directly proportional to the negative temperature gradient across the surface (for more information see [2]).

$$q = -\kappa(T)\nabla T \tag{2.2}$$

Here κ is the heat conductivity of the material and its value may depend on the value of the temperature T itself.

We consider two possible models

$$\kappa = 1 = const \tag{2.3}$$

$$\kappa = T^{\omega}, \quad \omega = 2.5 \tag{2.4}$$

Model (2.3) will lead to a linear PDE and model (2.4) results in a non-linear equation, which has an influence on the computations.

2.2 Potential Flow

The *potential equation* is a special case of the stationary heat equation. For example, it can describe the inviscid flow past and airfoil. Assuming an irrotational flow, i.e. $\overrightarrow{\omega} = rot\overrightarrow{v} = 0$, the momentum equations are already fulfilled. Furthermore, the *potential* Φ is introduced as

$$u = \frac{\partial \Phi}{\partial x} \quad v = \frac{\partial \Phi}{\partial y} \tag{2.5}$$

Which is consistent with the assumption $\omega = 0$

Continuity equation $div\overrightarrow{v} = 0$ then gives us the *Laplace equation* or potential equation as follows

$$\triangle \Phi = 0 \tag{2.6}$$

Whereas the velocity of the flow field can be later derived using 2.5. More details about potential flows can be found in [8].

As the equation is linear, there should not be any problems concerning stability and therefore this equation is very good for a first test of any solver.

The implementation presented in this project is treating time depending PDEs. This means that the solution of the potential equation is obtained by approaching the steady state of the general time dependent equation.

2.3 Navier-Stokes Equations

The Navier-Stokes equations describe the general motion of fluids. In contrast to the potential flow, viscous terms are included making the system of equations very difficult to solve. The conservation of mass, momentum and energy in two dimensions can be written as follows

$$\frac{\partial}{\partial t}\rho + \frac{\partial}{\partial x}(\rho u) + \frac{\partial}{\partial y}(\rho v) = 0 \tag{2.7}$$

$$\frac{\partial}{\partial t}(\rho u) + \frac{\partial}{\partial x}(\rho u^2 + p + \tau_{xx}) + \frac{\partial}{\partial y}(\rho uv + \tau_{xy}) = 0 \tag{2.8}$$

$$\frac{\partial}{\partial t}(\rho v) + \frac{\partial}{\partial x}(\rho uv + \tau_{xy}) + \frac{\partial}{\partial y}(\rho v^2 + p + \tau_{yy}) = 0 \tag{2.9}$$

$$\frac{\partial}{\partial t}(\rho E) + \frac{\partial}{\partial x}(\rho uE + up + u\tau_{xx} + v\tau_{xy} + q_x) + \frac{\partial}{\partial y}(\rho vE + vp + v\tau_{yy} + u\tau_{xy} + q_y) = 0 \tag{2.10}$$

Here τ is the symmetric stress tensor, which can be modelled for a *Newtonian fluid* as

$$\tau_{xx} = -\eta\left(2\frac{\partial u}{\partial x} - \frac{2}{3}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)\right) \tag{2.11}$$

$$\tau_{xy} = -\eta\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \tag{2.12}$$

$$\tau_{yy} = -\eta\left(2\frac{\partial v}{\partial y} - \frac{2}{3}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)\right) \tag{2.13}$$

And the heat fluxes are given by Fourier's law 2.2 again

$$q_x = -\lambda\frac{\partial T}{\partial x} \tag{2.14}$$

$$q_y = -\lambda\frac{\partial T}{\partial y} \tag{2.15}$$

With temperature T computed from any equation of state, e.g. $T = T(E, p)$. η is called the *viscosity* and λ is again the heat conductivity, which we assume constant for the first applications. Nevertheless non-linear models for the stresses and the parameters can also be easily implemented.

For a derivation and a more detailed explanation about the Navier-Stokes equations, we suggest [8] again.

Chapter 3

Numerics

3.1 Operator Splitting

Many physical models can be written in the following form

$$\frac{\partial}{\partial t}Q + divF(Q) = L(Q) \tag{3.1}$$

Where Q is the vector of the unknown variables (e.g. $Q = T$ in case of the heat equation), $F(Q)$ is the corresponding convective flux vector (e.g. $F(Q) = 0$ for the heat equation as there is no convection) and $L(Q)$ is a diffusive term or source term on the right hand side of the equation (e.g. $L(Q) = \triangle T$ for the heat equation with constant $\kappa = 1$).

Also the fluxes for the Navier-Stokes equations (2.7) can be split into diffusive and convective terms. The diffusive terms include all the terms of the stress tensor and the heat fluxes, whereas the convective terms are given by combinations of velocity and pressure.

This distinction is made because the physical processes of convection and diffusion are very different and therefore require special numerical treatments. A convective term for example is modelling transport in a certain direction so that an appropriate upwinding scheme is the method of choice for the numerical solution. In contrast to that, transport phenomena modelled by diffusive terms describe the spreading of particles, mass or energy without any distinct direction. So a symmetric stencil should be used for the discretisation of the operators or the reconstruction of the diffusive fluxes.

In order to seperate the treatment of those two different effects, one can split the original PDE into two systems as follows

$$\begin{aligned} \frac{\partial}{\partial t}Q_1 + divF(Q_1) &= 0 \\ \frac{\partial}{\partial t}Q_2 &= L(Q_2) \end{aligned}$$

Now the single processes can be treated seperately, but in order to get the solution of the original system (3.1) one has to somehow put the solutions of the single systems together.

This can be carried out by using the computed solution of the first equation as an initial condition for the next step of the second equation and vice versa. The scheme then consists of two steps on each time level.

First the convection equation is solved for one step using the old solution from the diffusion equation as initial condution

$$\begin{aligned} PDE &: \frac{\partial}{\partial t}\overline{Q} + \frac{\partial}{\partial x}F(\overline{Q}) = 0, \\ IC &: \overline{Q}(x, t^n) = Q^n \end{aligned}$$

Then the diffusion equation is solved for one step using the old solution from the convection equation as initial condution

$$\begin{aligned} PDE &: \frac{\partial}{\partial t}Q = L(Q), \\ IC &: Q(x, t^n) = \overline{Q}^{n+1} \end{aligned}$$

The fashion can be extended to a second order scheme, where a full step of the diffusive solution is preceded and followed by two half steps of the convective solution. For more details, the reader is referred to [7])

3.2 Spatial Discretization

The spatial discretization is very important for the implementation of the method, as it will be necessary to evaluate the Jacobian of the discretized fluxes. So we first have to explain the spatial discretization.

We employ an approach by Kapper, who uses a six point stencil for the reconstruction of the fluxes over the cell boundaries including a least squares solution. We will here only describe the procedure for the heat equation. The application of the same ideas can then easily be adopted for the other models.

The differential operators are first reformulated into fluxes over the four cell boundaries

$$\frac{\partial}{\partial x}F_x(Q) + \frac{\partial}{\partial y}F_y(Q) = \sum_{i=0}^{4} F_i(Q) \cdot n_i = \sum_{i=0}^{4} F_{i,x}(Q) \cdot n_{i,x} + F_{i,y}(Q) \cdot n_{i,y} \tag{3.2}$$

Where $F_{i,x}$ is the flux over cell boundary i in x direction, which is in case of the heat equation

$$F_{i,x}(T) = \kappa_i(T)\frac{\partial T}{\partial x}_i \quad F_{i,y} = \kappa_i(T)\frac{\partial T}{\partial y}_i \tag{3.3}$$

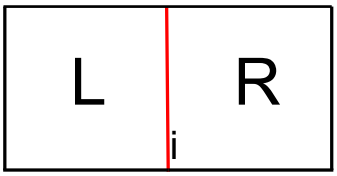


Figure 3.1: Stencil for the computation of κ using left and right neighbour of face i

The heat conductivity is averaged using the values of κ from the neighbour cells like in figure 3.1

$$\kappa_i(T) = \frac{1}{2}\left(\kappa(T_L) + \kappa(T_R)\right) \tag{3.4}$$

Note that this choice is different from an approach, where the temperature is first averaged as follows

$$\kappa_i(T) = \kappa(\frac{1}{2}\left(T_L + T_R\right)) \tag{3.5}$$

For the evaluation of the derivatives of T or Q in general, we consider a linear reconstruction of the values as follows

$$Q(x,y)_i = a_{i,0} + a_{i,1} \cdot x + a_{i,2} \cdot y \tag{3.6}$$

The coefficients $a_{i,j}$ are calculated using the stencil from 3.4.
Now the derivatives are easy to obtain

$$\frac{\partial Q}{\partial x}_i = a_{i,1} \quad \frac{\partial Q}{\partial y}_i = a_{i,2} \tag{3.7}$$

Is is important to know, that the coefficients $a_{i,j}$ are calculated using a least-squares approach which includes the six surrounding cells. Considering M as the matrix of the least-squares problem we get

$$\begin{pmatrix} a_{i,0} \\ a_{i,1} \\ a_{i,2} \end{pmatrix} = M^{-1} \cdot \begin{pmatrix} T_{i,0} \\ \vdots \\ T_{i,5} \end{pmatrix} \tag{3.8}$$

For the boundary cells, the fluxes are the same, but the coefficients are calculated using

$$\begin{pmatrix} \widetilde{a_{i,0}} \\ a_{i,1} \\ a_{i,2} \end{pmatrix} = \begin{pmatrix} a_{i,0} \\ a_{i,1} \\ a_{i,2} \end{pmatrix} - N_i \cdot \left[C_i \cdot \begin{pmatrix} a_{i,0} \\ a_{i,1} \\ a_{i,2} \end{pmatrix} - D_i \right] \tag{3.9}$$

The additional matrices N_i, C_i, D_i contain information about the boundary types for example. For additional details, we recommend [3]).

3.3 The Dual Time Stepping Method

In order to speed up convergence of unsteady flows, the dual time stepping method (first described by MERKLE in [5]) can be used. It introduces a new artificial time derivative into the partial differential equation. For example, we may consider

$$\frac{\partial Q}{\partial \tau} + \frac{\partial Q}{\partial t} + L(Q) = 0 \tag{3.10}$$

Where we can think of $L(Q)$ being an elliptic operator or anything modelling diffusive processes, like $L(T) = -\frac{\partial^2 T}{\partial^2 x} - \frac{\partial^2 T}{\partial^2 y}$ in (2.1) in two dimensions for constant heat conductivity $\kappa = 1$.

The time t is called the *physical time*, whereas the pseudo or artificial time τ is called the *dual time*.

In general, the elliptic operator may be discretized using any appropriate finite volume or finite difference method. Here we consider the discretization from above (3.2). This results in the semidiscrete formulation of equation (3.10)

$$\frac{\partial Q}{\partial \tau} + \frac{\partial Q}{\partial t} + R(Q) = 0 \tag{3.11}$$

It gives us a formulation which is general with respect to the fluxes and spatial discretization so that different approaches can be easily tested.

Now there are several possibilities to discretize the time derivatives. For applications it is very useful to choose a fully implicit approach in order to avoid time step constraints. We will now describe possible discretizations.

3.3.1 Semi-implicit Scheme

In most applications, one would use backwards differences for the physical time in order to avoid time step constraints due to problems with numerical stability. So we first choose an implicit discretization for t but an explicit for the dual time τ.

After the backwards difference in physical time, the semidiscrete system looks as follows

$$\frac{\partial Q}{\partial \tau} \,|_{n+1} = -\left(\frac{Q^{n+1} - Q^n}{\Delta t} + R(Q^{n+1})\right) =: -\overline{R}(Q^{n+1}) \tag{3.12}$$

The important difference to an explicit scheme is, that the discretized elliptic term on the right hand side has to be evaluated at the new time level.

Using an explicit difference formula for the dual time τ we work around this and get

$$\frac{Q_{k+1}^{n+1} - Q_k^{n+1}}{\Delta \tau} = -\overline{R}(Q_k^{n+1}) \tag{3.13}$$

Which can be easily solved for the update vector

$$\Delta Q_{k+1}^{n+1} = -\Delta\tau \overline{R}(Q_k^{n+1}) \tag{3.14}$$
$$Q_{k+1}^{n+1} = Q_k^{n+1} + \Delta Q_{k+1}^{n+1} \tag{3.15}$$

Though this was a implicit approach with respect to the physical time, there is still a time step constraint namely for the dual time step $\Delta\tau$ because the corresponding derivative was only discretized explicitly. On the other hand, the update is directly obtained without solving a coupled system of equations.

3.3.2 Fully implicit Scheme

In general, an explicit method has a certain time step constraint in order to be numerically stable. To overcome those problems, a fully implicit scheme can be used, which we describe in the following.

Again, we start from the PDE (3.10) and introduce the artificial time derivative to obtain

$$\frac{\partial Q}{\partial \tau} + \frac{\partial Q}{\partial t} + L(Q) = 0 \tag{3.16}$$

For the disretization of the different time derivatives, we use backwards difference formulas. This leads to

$$\frac{Q_{k+1}^{n+1} - Q_k^{n+1}}{\Delta\tau} + \frac{Q_{k+1}^{n+1} - Q^n}{\Delta t} + R(Q_{k+1}^{n+1}) = 0 \tag{3.17}$$

Note that we can also derive methods for higher order in time using special time discretizations.

Similar to the semi-implicit scheme, we define the operator $\overline{R}(Q_{k+1}^{n+1})$ as follows

$$\overline{R}(Q_{k+1}^{n+1}) = \left(\frac{Q_{k+1}^{n+1} - Q^n}{\Delta t} + R(Q_{k+1}^{n+1}) \right) \tag{3.18}$$

In order to be able to solve for the unknown Q_{k+1}^{n+1}, we linearize the right hand side $-\overline{R}(Q_{k+1}^{n+1})$ to get the update of the dual time step:

$$\overline{R}(Q_{k+1}^{n+1}) = \overline{R}(Q_k^{n+1}) + \frac{\partial \overline{R}}{\partial Q} \cdot \Delta Q_{k+1}^{n+1} \tag{3.19}$$

$$\Delta Q_{k+1}^{n+1} = Q_{k+1}^{n+1} - Q_k^{n+1} \tag{3.20}$$

Note that the linearisation does not introduce an error in the case of a linear equation (e.g. heat equation with constant heat conductivity) because the reconstruction of the fluxes is also linear.

Now we can write down the linear system of equations

$$\frac{\Delta Q_{k+1}^{n+1}}{\Delta \tau} = -\left(\overline{R}(Q_k^{n+1}) + \frac{\partial \overline{R}}{\partial Q}\Delta Q_{k+1}^{n+1}\right) \tag{3.21}$$

After rearranging terms, we get the equation that we have to solve for the update ΔQ_{k+1}^{n+1}

$$\left(I + \Delta\tau \cdot \frac{\partial \overline{R}}{\partial Q}\right)\Delta Q_{k+1}^{n+1} = -\Delta\tau \cdot \overline{R}(Q_k^{n+1}) \tag{3.22}$$

The value for the dependent variable Q at the new physical time step $n+1$ is now obtained by convergence of the inner dual time loop: $Q_k^{n+1} \to Q^{n+1}$ as $k \to \infty$

3.4 Solution of the large linear System of Equations

Because of the spatial discretization, neighbouring values couple and we get a large and sparse linear system of equations which has to be solved for the variable T

$$AQ = b \tag{3.23}$$

In order to solve the emerging system of equations, one should apply iterative methods, as it is computationally too expensive to solve the whole system directly by means of inversion.

For this purpose, we use a preconditioned restarted GMRES solver, which is applicable for every non-singular system matrix. In the case of a singular matrix there are possibilities end up in a so called *lucky breakdown*, which means that one still gets a solution.

3.4.1 Preconditioning

As a possibility to speed up the convergence of the iterative solver, preconditioning should be applied, which can be done by a simple Gauss-Seidel method for example. For further preconditioning techniques and a mathematical derivation see [4].

Applying left-preconditioning to the system, the equation reads

$$P^{-1}(AQ - b) = 0 \tag{3.24}$$

Where P is called the *preconditioning matrix*. Compared to that, the right-preconditioning approach yields

$$AP^{-1}\widetilde{Q} = 0 \qquad P^{-1}\widetilde{Q} = Q \tag{3.25}$$

Note that both possibilities are equivalent to the original system of equations $AQ = b$.

As we want to use a Gauss-Seidel solver, we introduce the preconditioning matrix P as

$$P = (D + L) \tag{3.26}$$

Here D is the diagonal part of the system matrix A in equation (3.23) and L is the lower part respectively.

The application of P^{-1} to a vector x is easy to perform, as the computation of every entry x_k^{m+1} only requires values from the last iteration and already calculated values of the new iteration

$$x_k^{m+1} = \frac{1}{a_{k,k}} \left(b_k - \sum_{i=1}^{k-1} a_{k,i} \cdot x_i^{m+1} - \sum_{i=k+1}^{n} a_{k,i} \cdot x_i^m \right) \tag{3.27}$$

As this method of preconditioning is usually very slow, we make use of the so called *over-relaxation*. Therefore we introduce a new parameter ω. The next value inside the Gauss-Seidel algorithm is then computed using a combination of the old value x_k^m and the new value x_k^* which was obtained using equation 3.25 as follows

$$x_k^{m+1} = (1 - \omega)\, x_k^m + \omega x_k^* \tag{3.28}$$

The stability theorie (for example VON NEUMANN analysis [1]) tells us, that ω has to be chosen between 0 and 2 to ensure convergence, where $\omega \in (0, 1)$ is called *under-relaxation* and $\omega \in (1, 2)$ is called *over-relaxation* respectively. For some highly non-linear equations under-relaxation might be necessary to obtain convergence at all, but we are interested in over-relaxation to accelerate the convergence of the Gauss-Seidel method. As will be shown later, the choice of ω has a large influence on the computation time of the linear solver (see 4.2).

Because of its simplicity, the Gauss-Seidel preconditioner needs to be applied more than one time to ensure convergence. It is possible to do many iterations of the Gauss-Seidel in order to approach the solution further. We will later take a closer look at this topic in the results chapter.

Especially for other models, we can use other preconditioners like the incomplete LU decomposition (ILU), because the Gauss-Seidel method might not work for all of the models we want to use during the future work on the project.

3.4.2 restarted GMRES

In general, GMRES is an iterative method for the numerical solution of large linear systems of equations like

$$Ax = b \tag{3.29}$$

One important advantage is, that there are no restrictions for the system matrix A. This is especially useful, as the matrix in our case is non-symmetric due to the transformation of the curvilinear coordinate system.

The GMRES method was developed by SAAD and SCHULZ in 1986 [6] and has been thoroughly used in many applications since then.

The method iteratively approaches the solution by finding a vector in the so called *Krylov subspace* with minimal residual. This *Krylov subspace* is denoted as

$$Kr_k = span\{r, Ar, A^2r, ..., A^{k-1}r\} \tag{3.30}$$

With the residual $r = Ax_n - b$. Instead of using the almost linear dependent vectors $r, Ar, ...$ for the computation, an orthogonal basis of the Krylov subspace is constructed. This is done using the *Arnoldi iteration*. For the update of the solution a least squares problem has to be solved

$$min|| \; ||r_0||_2 e_1 - H_k y_k|| \tag{3.31}$$

where H_k is an upper *Hessenberg* matrix resulting from the Arnoldi process.

Afterwards, the update can be computed as

$$x_k = x_0 + Q_k y_k \tag{3.32}$$

where $Q_k = (v_1, ..., v_k)$ is the matrix composed of the orthogonal basis of Kr_k.

A possible problem for this particular method is, that the memory requirements are very high, as every iteration needs to store one additional vector of the size of the solution vector for the basis of the Krylov subspace.

In order to overcome this problem, one often uses a *restarted GMRES* method. This means, that only a few iterations are performed and then the calculation starts again with the last solution vector as start value. Due to that, the memory storage is limited. Considering m as the number of maximum inner iterations before a restart, this method is often called the *GMRES(m)* method.

In pseudocode, the method including left preconditioning with the matrix P^{-1} then looks like this

for $n = 0, 1, 2, ..., n_{max}$ **do**
  $r_n = P^{-1}(b - Ax_n)$ (includes Gauss-Seidel solve)
  $\gamma_0 := \|r_n\|_2, v_1 := \frac{r_n}{\gamma_n}$
  for $j = 1, ..., m$ **do**
    $w := P^{-1}Av_j$ (includes Gauss-Seidel solve)
    for $i = 1, ..., k$ **do**
      $h_{ij} := v_i^T w$
      $w := w - h_{ij} v_i$
    end for
    $h_{j+1,j} := \|w\|_2$
    if $h_{j+1,j} = 0$ **then**
      j:=m
    else
      $v_{j+1} := \frac{w}{h_{j+1,j}}$
    end if
    solve LLS: $min\| \; \|r_0\|_2 \; e_1 - H_k y_k\|$
  end for

update $x_{n+1} := x_n + Q_k y_k$
end for

In contrast to that, using right preconditioning leads to slightly changed algorithm

for $n = 0, 1, 2, ..., n_{max}$ **do**
  $r_n = b - Ax_n$
  $\gamma_0 := \|r_n\|_2, v_1 := \frac{r_n}{\gamma_n}$
  for $j = 1, ..., m$ **do**
    $w := AP^{-1}v_j$ (includes Gauss-Seidel solve)
    for $i = 1, ..., k$ **do**
      $h_{ij} := v_i^T w$
      $w := w - h_{ij} v_i$
    end for
    $h_{j+1,j} := \|w\|_2$
    if $h_{j+1,j} = 0$ **then**
      j:=m
    else
      $v_{j+1} := \frac{w}{h_{j+1,j}}$
    end if
    solve LLS: $min\| \|r_0\|_2 \, e_1 - H_k y_k\|$
  end for
  update $x_{n+1} := x_n + Q_k P^{-1} y_k$ (includes Gauss-Seidel solve)
end for

Both algorithms rely on the calculation of an initial residual, which is here denoted by γ_0. An important difference is that this residual is only in the preconditioned space in the case of left-preconditioning. As a consequence, a very small value of γ_0 does not necessarily imply a small residual of the equation that one wants to solve. On the other hand, right preconditioning calculates the exact residual. Of course, both variants converge to the same unique solution but the size of the residuals inside of the methods is not comparable.

In both cases there are several parameters, that ensure or accelerate the convergence of the GMRES(m):

$$m \;:\; \text{maximum dimension of the Krylov subspace} \tag{3.33}$$
$$n_{max,GS} \;:\; \text{maximum number of Gauss-Seidel iterations inside} \tag{3.34}$$
$$eps \;:\; \text{threshold for the residual inside GMRES(m)} \tag{3.35}$$

Note that we do not specify a threshold for the residual inside the Gauss-Seidel solver. The reason for this is, that we do neither expect nor need convergence inside this preconditioner. The Gauss-Seidel could even fail to converge, because the system matrix might not be diagonally dominant (depending on size of the timesteps), which is a requirement for the convergence of the Gauss-Seidel method. Therefore, we only iterate a few times to come closer to the solution and then apply GMRES(m) to solve the problem completely.

In our calculations, we used for example $m = 20, n_{max,GS} = 100$ and $eps = 10^{-4}$ and observed a good convergence behaviour of the method. Though, these values might need to be changed in order to obtimize the convergence speed for different physical or dual step sizes.

A value of 10^{-4} might seem quite high for a residual threshold of the GMRES(m) method, but it is not necessary to fully converge with the iterative solver as convergence is only mandatory for the dual iterations. So in order to save computational time, one should better do more dual iterations than resolve each single dual iteration more accurately. We observed that a value of $eps = 10^{-4}$ is sufficient for the convergence of the dual iterations. Larger values can spoil the quality of the solution at every dual time step too much and therefore lead to a larger amout of dual iterations until convergence. Again, this also depends on the physical and dual time step.

3.5 Analytical Computation of the Jacobian

As an example of the computation of the Jacobian, we explain the calculation for the heat equation assuming nonlinear heat conductivity.

If we consider a linear equation and all the steps of the flux reconstruction are also linear, the term $\frac{\partial \overline{R}}{\partial T}$ will not depend on the values of the solution, when we choose constant heat conductivity (2.3). Because of that the calculation of the corresponding term can be carried out before the iteration procedure starts in this case. In the case of variable heat conductivity (2.4) the equation becomes non-linear and the Jacobian has to be evaluated after every change in the variable T.

Obviously, a certain value only couples with other values inside the stencil of the discretization. As the fluxes are reconstructed using a six point stencil, we get a nine point stencil for the cell values. The numbering of the cells is chosen as follows

Now one has to get the derivative of the right hand side $\overline{R}(T)$ which is basically the derivative of the spatial discretization using the fluxes over the boundary plus one extra term for the physical time discretization.

The most difficult part here is to consider the values inside the flux stencil with the right indices for the horizontal and vertical faces.

Here also the derivative of the heat conductivity κ has to be taken into account. Using the ansatz (2.4), the derivative looks like

$$\frac{\partial \kappa}{\partial T} = \omega \cdot T^{\omega - 1} \tag{3.36}$$

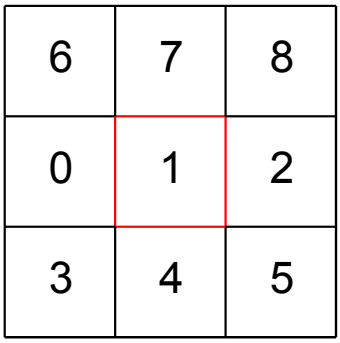


Figure 3.2: Stencil for the computation of the value at the cell center

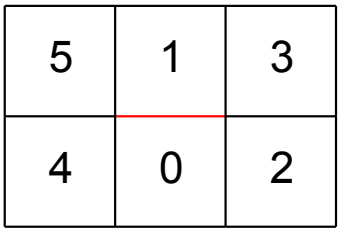


Figure 3.3: Stencil for the computation of the fluxes over a horizontal face

The final computation of the coefficients for the system matrix in the notation $AT = b$ can be seen in the appendix A

For the application it is important to know that the matrices M, N, C used in the computation of the coefficients are constant throughout the whole flow simulation for each cell. It is possible to calculate all the coefficients before the simulation starts and there is no recalculation necessary, if one uses the constant heat conductivity because then it is $\partial\kappa_{i,j} = 0$ and so the Jacobian does no longer depend on the temperature itself.

As one can easily see in A, the analytical derivation of the Jacobian is very elaborate task. Not only the structure of the equations themselves, but also the discretization has to be taken into account. This leads to an error-prone exercise where small mistakes can cause totally wrong solutions afterwards. Furthermore, some complex reconstructions and discretizations might not even have an analytical derivative or the derivative is at least not easy to obtain. This is one of the main disadvantages of the analytical calculation of the Jacobian.

The obvious advantage is, that the solution is (except for numerical round-off errors) exact to machine precision. Due to the manual calculation, it is also easier to exploit the sparsity of the matrix efficiently.

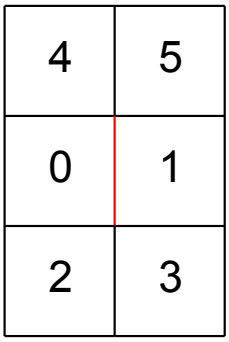


Figure 3.4: Stencil for the computation of the fluxes over a vertical face

3.6 Conditioning Number of System Matrix

While solving the equation

$$AQ = b \tag{3.37}$$

The properties of the system matrix A are very important. We have already pointed out that we cannot assume symmetry, because of the coordinate transformation from curvilinear to cartesian coordinates. Additionally, the conditioning number of the matrix A is important.

In general, the conditioning number of a non-singular matrix is defined as

$$\kappa(A) = \|A\|_2 \|A^{-1}\|_2 \tag{3.38}$$

κ is a measure for the inaccuracy of the solution of the problem. A large conditioning number may lead to large errors in the solution, whereas a small conditioning number indicates a small amplification of errors.

Hence one is interested in obtaining a well-conditioned matrix, represented by a small conditioning number.

As it is known from linear algebra, diagonally dominant matrices often have a small conditioning number. A matrix A is said to be diagonally dominant, if

$$\sum_{j=1, j\neq i}^{n} |a_{i,j}| < |a_{i,i}| \tag{3.39}$$

For different ratios of the step sizes $\frac{\Delta\tau}{\Delta t}$ this property of the matrix is more likely to be achieved. The reason for this is, that the entries $a_{i,i}$ on the diagonal of the matrix A are

$$a_{i,i} = 1 + \frac{\Delta\tau}{\Delta t} + \Delta\tau \cdot (\dots) \tag{3.40}$$

In contrast to that, the off-diagonal entries can be computed as

$$a_{i,j} = \Delta\tau \cdot (\dots) \tag{3.41}$$

The term $\frac{\Delta\tau}{\Delta t}$ resulting from the physical time discretization therefore increases the diagonal entries and helps to improve the conditioning number of the system matrix A.

We computed the matrix A explicitly for several values of $\Delta\tau$ and Δt. The conditioning numbers for the choice of $\kappa \equiv 1$ are presented in the following table.

	$\Delta t = 0.1$	$\Delta t = 1$
$\Delta\tau = 0.05$	3.6134	4.7330
$\Delta\tau = 0.1$	4.9196	8.1242
$\Delta\tau = 0.2$	6.2254	14.0531

Table 3.1: Conditioning number for different time step sizes

On the first view one can see that the conditioning number is relatively small for every presented combination of the time steps. Furthermore, the conditioning number is increasing with both the physical and the dual time step. But the values for the conditioning numbers in this table are still very small compared to real ill-conditioned problems.

Additional calculations have shown, that the system matrix A is diagonally dominant for all the cases tested above.

Chapter 4

Results

4.1 Simulation Results

4.1.1 Domain

Before we see the first simulation results using the framework explained in detail in chapter 3, we need to define a domain on which the PDEs are to be solved. In the case of potential equation, an interesting test case is the inviscid flow past a cylinder. In order to model this, we consider a domain according to the following figure

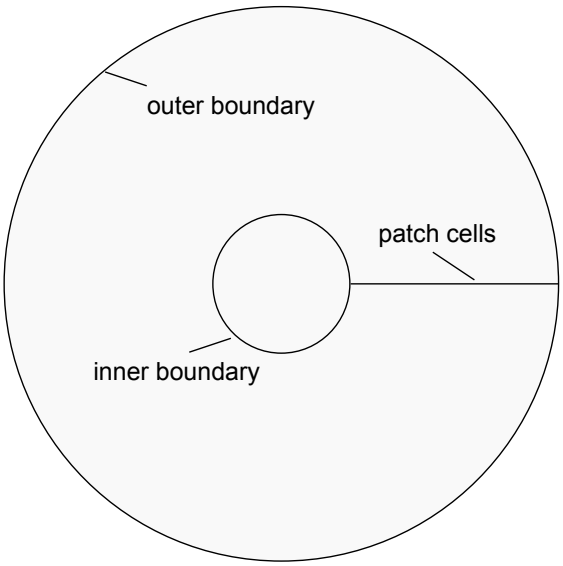


Figure 4.1: Domain for simulations

The domain of interest has an inner boundary which is the small cylinder and an outer boundary which is considered to be far away from the circle.

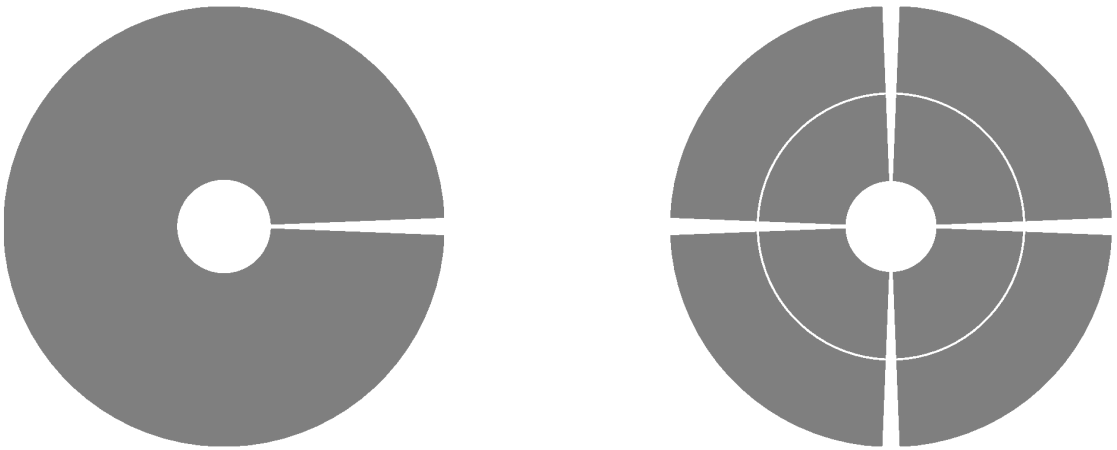
4.1.2 Grid

For the first simulations, we consider a non-cartesian grid according to figure 4.2. The grid consists of 3200 points, where the spatial discretization is more accurate near the inner boundary and coarser next to the outer boundary

Figure 4.2: Computational grid for the simulations

Furthermore, there is a layer of patch cells in the middle of the right part of the domain. In this way, the information exchange over the boundaries of the domains can also be tested in this simple test scenario.

We also made computations with a decomposed domain, that is shown in figure 4.3. The results are the same in the stationary case, so we will only focus on the simple domain now.



(a) single domain (b) eight domains

Figure 4.3: Domain decomposition using one (left) or eight domains (right)

4.1.3 Boundary and Initial Conditions

For the simulation of the potential flow past a cylinder, we use slip conditions at the wall which is the inner boundary in figure 4.1. In order for the inner boundary to become a streamline, we have to enforce the potential lines to have a gradient in normal direction to the wall.

At the outer boundary, we consider a free stream velocity $u = 0.1, v = 0$ which corresponds to the related derivatives according to the definition of the potential 2.5.

As initial conditions we choose $\Phi = 1.5$ in the whole domain. Referring to equation 2.5 only the derivative of the potential Φ is defined, so we can basically add another value to the potential and still satisfy the definition. This makes the choice of initial conditions somewhat arbitrary.

4.1.4 Solution of Potential Equation

The solution of the potential equation for the flow around a cylinder is well known and can even be calculated using analytical methods. Nevertheless we want to show the solution computed with our new framework. Figure 4.4 is calculated using the whole software including preconditioning and the restarted GMRES method. By comparison with the analytical solution we see that the solution is the same and the test case is therefore solved successfully.

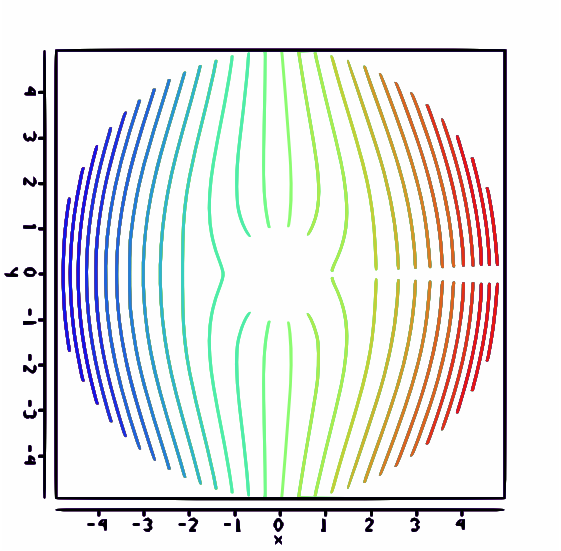


Figure 4.4: Simulation result for potential flow past a cylinder

4.2 Linear Solver

The large linear system of equations emerging from the discretization has to be solved numerically. In section 3.4 we explained the techniques used to approach the solution iteratively. In order to get the solver running efficiently, we have to find the right parameters for the specific methods. We will now start discussing the effects of the different parameters on the quality of the solution and the associated runtime needed for the simulation.

4.2.1 Gauss Seidel Preconditioner

Before working on the GMRES method we have to consider the preconditioning because it is highly relevant for the runtime of the whole solution process as we will see later. So we start with examining the effect of the over-relaxation parameter ω. Figure 4.5 shows the results of a first test, using the Gauss-Seidel solver only to solve the linear system of equations emerging from a discretization using the domain above and $\Delta\tau = 1000$, $\Delta t = 10$. The residual after 100 Gauss-Seidel iterations is plotted against the parameter ω.

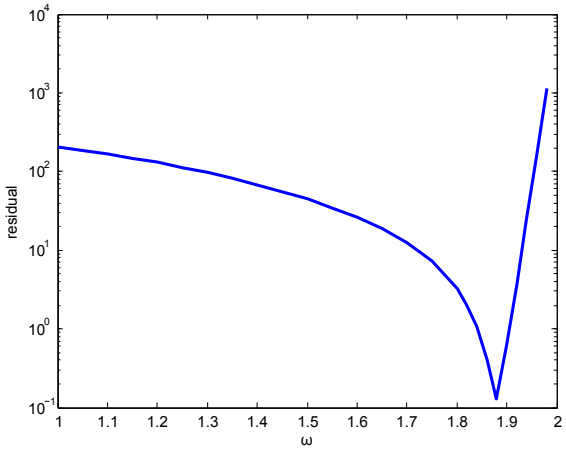


Figure 4.5: Residual of the GS solver with respect to the over-relaxation parameter ω after 100 iterations

First of all, we see that the accuracy of the solution is not good enough, because the residual is in every case above 10^{-1}. That shows that the GS solver should in general not be used alone to solve the system because it needs too many iterations to solve the system with the desired accuracy.

Furthermore, the effect of ω on the residual is quite impressive. The residual constantly decreases with increasing ω until the optimal value $\omega_{opt} = 1.88$ in this case. After that, the residual is increasing again until the method is diverging for values of ω larger than 2. This behavior is typical for the Gauss-Seidel method with over-relaxation, but the optimal value for ω depends on the problem. In practical application one should

better use a value a bit smaller than the optimal ω to be on the safe side.

With the help of this test we can use the Gauss-Seidel solver for preconditioning the LSE.

4.2.2 GMRES Solver

After the investigation of the preconditioning we will now take a closer look at the GMRES solver.

GMRES(m) with and without Preconditioning

For the same setup as mentioned above, we test the performance of the GMRES method with respect to preconditioning and the dimension of the Krylov subspace. Figure 4.6 shows the results in separate plots for the different dimensions. The number of iterations of the GMRES(m) refers to the number of restarts that are done during the calculation. It is important to say, that we did not use over-relaxation in this test.

We can already obtain a good convergence rate for the non preconditioned method, indicated by the linear slope in the log-plot. But similar as the GS solver itself the GMRES solver cannot reach a very small residual after a few iterations without preconditioning. When using Gauss-Seidel for preconditioning with 100 subiterations, the residual decreases much faster and reaches the threshold of 10^{-13} after a small number of iterations.

Furthermore, the number of iterations needed until convergence decreases with the dimension of the Krylov subspace. This is not surprising because a higher dimension means more subiterations in every GMRES(m) circle before a restart.

Summarizing, this simple test shows the advantages of preconditioning very clearly. We see that only 100 Gauss-Seidel iterations for preconditioning can decrease the number of needed restarts essentially. Every single GMRES(m) iteration needs more time as the GS iterations have to be performed inside, of course. We will nevertheless see that also the computational time decreases with the use of preconditioning.

GMRES(m) with varying GS iterations

As the use of preconditioning seems to be beneficial for the iteration count of the GMRES solver, we want to investigate the behavior of the solver while varying different parameters of the preconditioner.

First of all, we change the number of subiterations of the Gauss-Seidel preconditioner in order to find useful values for the simulations.

Figure 4.7 was done using a Krylov subspace dimension of 10 and no overrelaxation. When using no Gauss-Seidel subiterations (case 0), the residual is decreasing very slowly like in figure 4.6. For 20 or 40 iterations, there is no convergence at all, the residual is basically the same during the whole simulation. This is due to the fact, that the small number of Gauss-Seidel iterations is not sufficient to come closer to the solution. In this case, wrong choice of parameters for the preconditioner destroys the properties of the

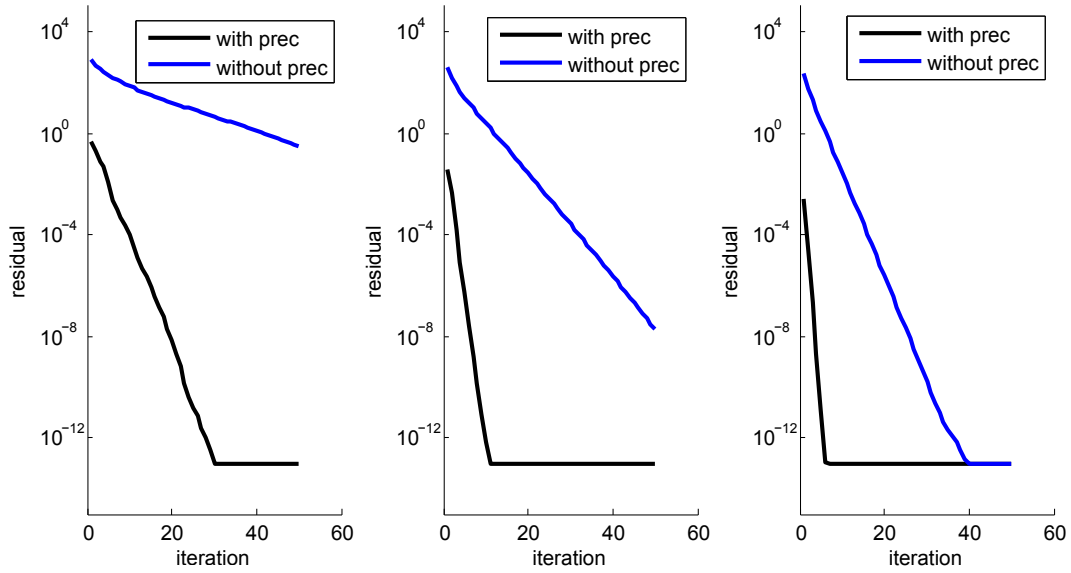


Figure 4.6: Residual of the GMRES solver with respect to the iteration restarts with and without preconditioning. Dimension of Krylov subspace: left 10, middle 20, right 30

GMRES solver. A higher subiteration number restores the convergence of the solver and in fact accelerates the simulation.

The figure should not lead to the conclusion, that a very high number of Gauss-Seidel iterations is the best choice because then the number of GMRES iterations is very small. In that case, the problem would be mainly solved by the Gauss-Seidel solver itself what is not desirable. Instead we want to use the preconditioner only to speed up the GMRES solver because it is well known that it needs less operations than the Gauss-Seidel solver in general.

Nevertheless we have to use a number of subiterations that is large enough to ensure convergence of the solver. In most of the cases we will choose something like 100 Gauss-Seidel iterations, which should be sufficient as the test has shown.

With further tests we obtained that the minimal number of subiterations needed decreases with increasing Krylov subspace dimension. So for a dimension of 20, we could achieve convergence with only 40 subiterations.

GMRES(m) with varying Overrelaxation Parameter ω

We have seen before that the choice of ω is very important for the Gauss-Seidel algorithm. That is why we also expect it to be useful for the reduction of iterations of the GMRES method.

The following figure 4.8 visualizes the effect of the parameter ω on the number of iterations of the GMRES solver. The test has been done using a Krylov subspace dimension of 10 and 100 Gauss-Seidel subiterations. The computation was stopped

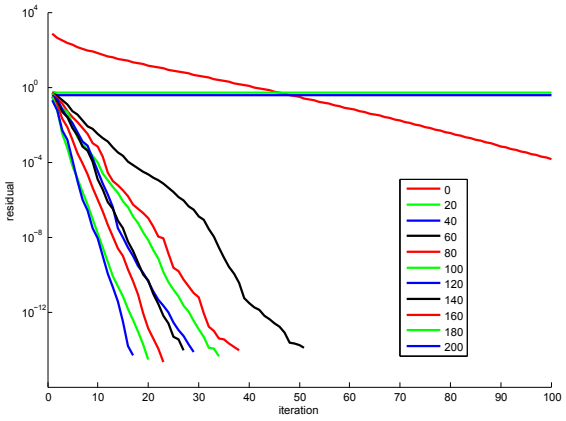


Figure 4.7: Residual of the GMRES solver with respect to the iteration restarts with different numbers of GS subiterations from 0 to 200

when the residual reached 10^{-14}.

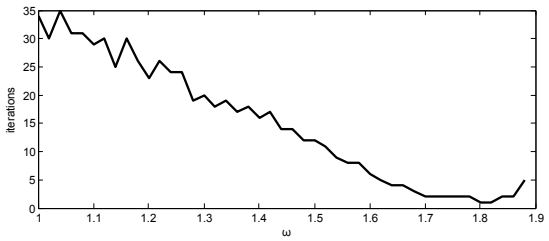


Figure 4.8: Number of iteration restarts of the GMRES solver with respect to the overrelaxation parameter ω

The figure is comparable to the Gauss-Seidel test in figure 4.5. We can see a significant decrease in the iteration count for values of ω around 1.8 which is pretty much the optimal value of ω for this particular test case. For larger values the number of iterations is increasing until we have no convergence at all beginning with $\omega = 1.90$. This reminds us again to better choose a smaller value than a larger one due to the stability problems.

We also visualized the decrease of the residual during the simulation for different values of the relaxation parameter ω in another test. This time, the test setup uses only 50 instead of 100 Gauss-Seidel subiterations but the result is still comparable. The results are shown in figure 4.9.

For the choice of $\omega = 0.5$ which corresponds to under-relaxation the residual decreases very slow, while the other parameters come up with a much better performance. For a value of 1.75 we could reach the desired residual of 10^{-13} within only one GMRES iteration. So there was no restart necessary. Compared to the early attempts without preconditioning and other parameter choices this is a significant improvement.

The results of the two tests are even more important considering that changes in the parameter ω have no influence on the computational time needed for one single GMRES

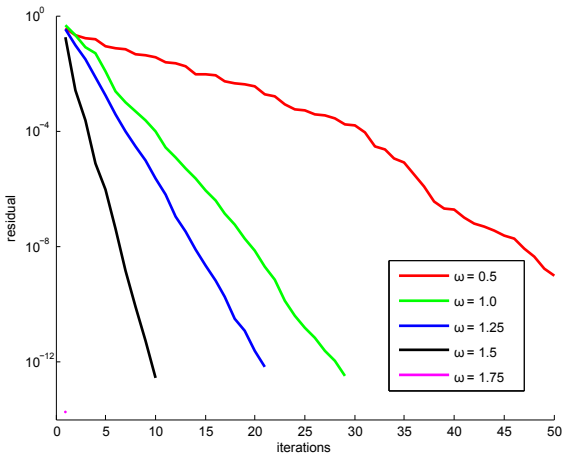


Figure 4.9: Number of iteration restarts of the GMRES solver with respect to the over-relaxation parameter ω

iteration. That means that only by the right choice of one parameter we can reduce the total runtime by a factor of about 30 according to figure 4.5 where the iteration count is reduced from 34 to less than 1 with a change of ω from 1.0 to 1.8.

It turns out that over-relaxation is, besides preconditioning itself, the most effective way to accelerate the computations.

4.2.3 Runtime Measurements

With respect to an *acceleration* of the simulation it is not only important to reduce the iteration count but also to reduce the overall runtime of the simulation. Therefore we made some tests in addition to the results from the previous section that show the advantages of the changes that we applied during the various tests before with respect to the runtime.

First of all, we measure the time for the test in figure 4.8. As the runtime of one GMRES iteration is basically constant, the total runtime of the solution of the LSE is proportional to the number of GMRES iterations. So the figures are almost exactly the same except of the scale of the y-axis. As mentioned before, the runtime decreases rapidly for the optimal choice of $\omega = 1.8$ compared to 1.0.

In another test we refer to the change in the number of Gauss-Seidel subiterations. Using a Krylov dimension of 10, the optimal overrelaxation parameter $\omega = 1.8$ and calculating until the residual drops below 10^{-14} we obtain the following figure 4.11.

On the left y-axis, we can see the number of iterations, whereas the right y-axis shows the total time needed for the solution of the LSE. Like in the tests before, we see a decrease in the number of GMRES iteration counts with increasing GS iterations. But once we reduced the number of GMRES iterations to one it is not useful to increase the number of GS iterations more. The runtime will stay almost constant and even increases with higher numbers. So it is a good choice to pick a moderate number of Gauss-Seidel

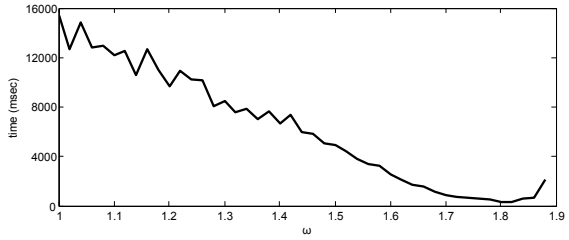


Figure 4.10: Runtime for the solution of the LSE with respect to the overrelaxation parameter ω

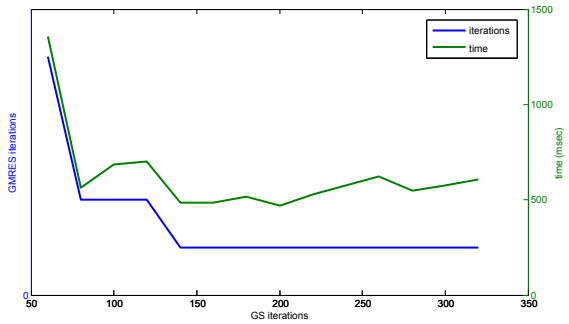


Figure 4.11: Runtime and iteration count for the solution of the LSE with respect to the number of GS subiterations

subiterations in order not to overdo the preconditioning.

The last runtime test for the GMRES solver includes the choice of the Krylov subspace dimension. We are using no preconditioning in order to investigate the behavior of the GMRES solver only. Furthermore we are computing until a residual of 10^{-6}.

The results are shown in figure 4.12

As expected the number of iterations decreases with the Krylov dimension. At the beginning also the runtime decreases but after 30 there is no significant change in the runtime and it even increases a bit. Despite the figure there are two reasons why we should not choose too large values for the Krylov dimension. The first reason is, that the number of iterations and the runtime will be decreased using preconditioning afterwards in any way, so it might be enough to choose a much smaller number. The second reason is the increasing memory consumption of the computation. The method has to store a vector of the size of the whole linear system for each Krylov dimension. This memory demand is usually a bottleneck during the whole flow simulation, because you normally only use a few vectors in the rest of the program to save memory. We can also see, that the runtime is not decreasing that much after around 30. For these reasons we only choose values up to 20 for the dimension of the Krylov subspace.

It is important to mention that this results highly depend on the problem to be solved. A larger domain with a more accurate discretization will definitely need a larger Krylov dimension and probably different preconditioning settings. Nevertheless the val-

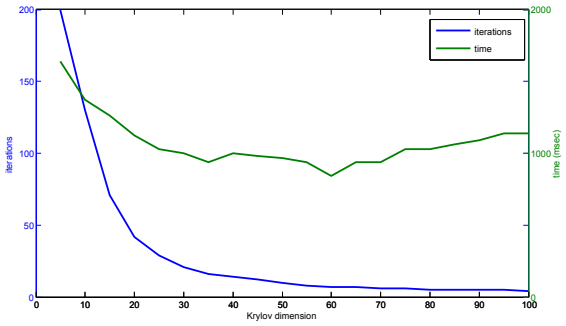


Figure 4.12: Runtime and iteration count for the solution of the LSE with respect to the dimension of the Krylov subspace m

ues presented here can be seen as a good orientation for different simulation cases.

4.3 Comparison with explicit solver

The aim of the project was the acceleration of the calculation using the techniques described in the previous chapter. The results so far only treat improvements for the new methods themselves, so we still have to compare the new implementation with the existing one. The original method was explicit in time and the CFL condition restricted the time step to $\Delta t \leq 0.8 \cdot 10^{-2}$ in the non-linear case, which is in fact very small compared with the time needed for convergence to the steady state.

In order to compare the implicit solver with the old explicit solver, we did time measurements for the following setup. We measured the time needed to calculate the solution at $t = 100(sec)$ starting from $t = 0$ with the initial conditions. Then we compare three test cases

(1) original explicit method: time step size $\Delta t = 0.008$, 12500 timesteps in total

(2) implicit method using GMRES without precontitioning: $\Delta t = 10$, 10 timesteps in total, $\Delta \tau = 10000$, threshold for dual residual $\leq 10^{-6}$, threshold for GMRES residual $eps = 10^{-4}$, Krylov subspace dimension 20

(3) implicit method using GMRES with precontitioning: $\Delta t = 10$, 10 timesteps in total, $\Delta \tau = 10000$, threshold for dual residual $\leq 10^{-6}$, threshold for GMRES residual $eps = 10^{-4}$, Krylov subspace dimension 20, number of GS iterations 100, overrelaxation parameter $\omega = 1.8$

The results of the runtime measurements are shown in figure 4.13 for the linear and non-linear heat equation.

First of all, there is a difference in the runtime for the linear and non-linear model. The runtime for the linear case is much smaller. This is due to the fact, that the non-linear model includes the calculation of the heat conductivity according to equation 2.4.

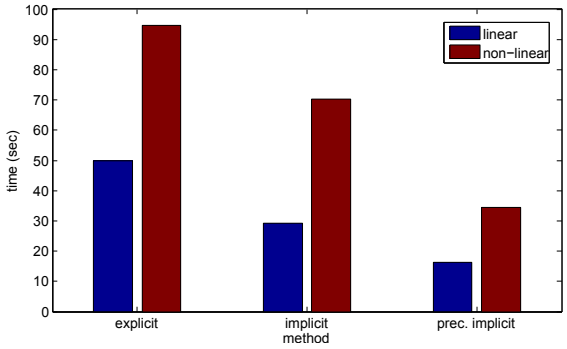


Figure 4.13: Total runtime measurement for different test setups showing speedup for improved implicit method

The calculation is much more complex in terms of computational operations than the linear model. As this calculation has to be done for every face, the non-linearity plays a major role for the runtime. Furthermore, there are more subiterations for the non-linear model in the implicit cases.

Concerning the runtime, the implicit solver is already faster than the original explicit solver. Though the first few physical iterations are very slow, the calculation is speeded up in the vicinity of the steady state because the changes in the flow variable become smaller leading to a decrease in the number of dual time iterations needed to converge on every physical time level. In total, the runtime is reduced by 40% in the linear case with the help of the implicit solver.

When adding preconditioning with the optimal parameters found in the sections above, the results are even better. Here we come up with a decrease in runtime of almost 67% in the linear case. We observed that also the reduction of GMRES iterations inside each dual iteration leads to the large additional speed up.

Chapter 5

Conclusions and Future Work

We have developed an implicit framework that can in general be applied to any PDE of elliptic type. The Dual Time Stepping scheme has been implemented and tested. After the implementation of the restarted GMRES method, we were able to accelerate the convergence of this iterative solver using preconditioning.

Further improvements have been made due to the choice of optimal parameters for the relaxation parameter ω or the dimension of the Krylov subspace, for example. With the help of this we achieved a large speedup compared to the original setting. Mainly the preconditioning itself and the over-relaxation reduced the iteration count as well as the runtime of the solver.

The whole framework has been successfully tested for different equations and the results have shown the benefits of the various numerical methods that were used. With respect to the original explicit solver, we could get a good speedup and the possibility to use arbitrary physical time steps as the implicit method does not need to fulfill any CFL constraint.

Furthermore the linear solver was investigated in detail with respect to the different parameters. We examined the effects of the subiterations number, the over-relaxation parameter ω, the Krylov subspace dimension and the preconditioning itself. Due to this we were able to make the parameter choices mentioned above to obtain better convergence behavior.

The work of this project already provides an operating framework for the solution of PDEs but there are some possible additions to reduce runtime or increase ease of use.

First of all, a finite difference approximation of the Jacobian would be useful to integrate new models more easily. Therefore one has to work on efficient ways to implement the numerical calculation of the derivative of the right hand side term. This would also include an investigation of the optimal step size for the finite differences to balance the discretization errors and round-off errors. One could think of Jacobian free methods for the evaluation of the Jacobian as well.

An addition of new models will be required in order to simulate different physical

processes. Using a finite difference approach, this would only require the flux calculation, because the Jacobian can be evaluated automatically by the numerical derivation.

The concept of adaptivity can be applied to both step sizes Δt and $\Delta \tau$ as well as to the number of preconditioning subiterations. This could give rise to further runtime improvements especially in the vicinity of the steady state of the PDE. In this region, the time steps could be increased to reach the steady state solution earlier and save runtime in return.

Besides that, there is still the possibility to solve the large linear system using a direct solver like the LU-decomposition or any other dedicated method and compare the runtime to the iterative methods.

Due to the general structure of the code, it is also very easy to integrate new solvers, preconditioners or even discretization schemes.

In the near future, the work on this project will lead to a bachelor thesis where we want to use the framework implemented to solve the Navier-Stokes equations or the Boltzmann moment system which is an extension to the Navier-Stokes system.

Appendix A

Analytical Computation of the Jacobian

$$
\begin{aligned}
A_{i,0} =& \Delta\tau \cdot (\\
&+ (n_{0,x} \cdot (\kappa_0 \cdot M_{0,1,0} + \partial\kappa_{0,0} \cdot a_{0,1}) + n_{0,y} \cdot (\kappa_0 \cdot M_{0,2,0} + \partial\kappa_{0,0} \cdot a_{0,2})) \\
&- (0) \\
&+ (n_{2,x} \cdot (\kappa_2 \cdot M_{2,1,5}) + n_{2,y} \cdot (\kappa_2 \cdot M_{2,2,5})) \\
&- (n_{3,x} \cdot (\kappa_3 \cdot M_{3,1,4}) + n_{3,y} \cdot (\kappa_3 \cdot M_{3,2,4})))
\end{aligned} \tag{A.1}
$$

$$
\begin{aligned}
A_{i,1} =& 1 + \frac{\Delta\tau}{\Delta t} + \Delta\tau \cdot (\\
&+ (n_{0,x} \cdot (\kappa_0 \cdot M_{0,1,1} + \partial\kappa_{0,1} \cdot a_{0,1}) + n_{0,y} \cdot (\kappa_0 \cdot M_{0,2,1} + \partial\kappa_{0,1} \cdot a_{0,2})) \\
&- (n_{1,x} \cdot (\kappa_1 \cdot M_{1,1,0} + \partial\kappa_{1,0} \cdot a_{1,1}) + n_{1,y} \cdot (\kappa_1 \cdot M_{1,2,0} + \partial\kappa_{1,0} \cdot a_{1,2})) \\
&+ (n_{2,x} \cdot (\kappa_2 \cdot M_{2,1,1} + \partial\kappa_{2,1} \cdot a_{2,1}) + n_{2,y} \cdot (\kappa_2 \cdot M_{2,2,1} + \partial\kappa_{2,1} \cdot a_{2,2})) \\
&- (n_{3,x} \cdot (\kappa_3 \cdot M_{3,1,0} + \partial\kappa_{3,0} \cdot a_{3,1}) + n_{3,y} \cdot (\kappa_3 \cdot M_{3,2,0} + \partial\kappa_{3,0} \cdot a_{3,2})))
\end{aligned} \tag{A.2}
$$

$$
\begin{aligned}
A_{i,2} =& \Delta\tau \cdot (\\
&+ (0) \\
&- (n_{1,x} \cdot (\kappa_1 \cdot M_{1,1,1} + \partial\kappa_{1,1} \cdot a_{1,1}) + n_{1,y} \cdot (\kappa_1 \cdot M_{1,2,1} + \partial\kappa_{1,1} \cdot a_{1,2})) \\
&+ (n_{2,x} \cdot (\kappa_2 \cdot M_{2,1,3}) + n_{2,y} \cdot (\kappa_2 \cdot M_{2,2,3})) \\
&- (n_{3,x} \cdot (\kappa_3 \cdot M_{3,1,2}) + n_{3,y} \cdot (\kappa_3 \cdot M_{3,2,2})))
\end{aligned} \tag{A.3}
$$

$$
\begin{aligned}
A_{i,3} =& \Delta\tau \cdot (\\
&+ (n_{0,x} \cdot (\kappa_1 \cdot M_{0,1,2}) + n_{0,y} \cdot (\kappa_1 \cdot M_{0,2,2})) \\
&- (0) \\
&+ (n_{2,x} \cdot (\kappa_2 \cdot M_{2,1,4}) + n_{2,y} \cdot (\kappa_2 \cdot M_{2,2,4})) \\
&- (0))
\end{aligned} \tag{A.4}
$$

$$\begin{aligned} A_{i,4} =& \Delta\tau \cdot (\\ &+ (n_{0,x} \cdot (\kappa_0 \cdot M_{0,1,3} + \partial\kappa_{2,0} \cdot a_{2,1}) + n_{0,y} \cdot (\kappa_0 \cdot M_{0,2,3} + \partial\kappa_{2,0} \cdot a_{2,2})) \\ &- (n_{1,x} \cdot (\kappa_1 \cdot M_{1,1,2}) + n_{1,y} \cdot (\kappa_1 \cdot M_{1,2,2})) \\ &+ (n_{2,x} \cdot (\kappa_2 \cdot M_{2,1,0}) + n_{2,y} \cdot (\kappa_2 \cdot M_{2,2,0})) \\ &- (0)) \end{aligned} \tag{A.5}$$

$$\begin{aligned} A_{i,5} =& \Delta\tau \cdot (\\ &+ (0) \\ &- (n_{1,x} \cdot (\kappa_1 \cdot M_{1,1,3}) + n_{1,y} \cdot (\kappa_1 \cdot M_{1,2,3})) \\ &+ (n_{2,x} \cdot (\kappa_2 \cdot M_{2,1,2}) + n_{2,y} \cdot (\kappa_2 \cdot M_{2,2,2})) \\ &- (0)) \end{aligned} \tag{A.6}$$

$$\begin{aligned} A_{i,6} =& \Delta\tau \cdot (\\ &+ (n_{0,x} \cdot (\kappa_0 \cdot M_{0,1,4}) + n_{0,y} \cdot (\kappa_0 \cdot M_{0,2,4})) \\ &- (0) \\ &+ (0) \\ &- (n_{3,x} \cdot (\kappa_3 \cdot M_{3,1,5}) + n_{3,y} \cdot (\kappa_3 \cdot M_{3,2,5}))) \end{aligned} \tag{A.7}$$

$$\begin{aligned} A_{i,7} =& \Delta\tau \cdot (\\ &+ (n_{0,x} \cdot (\kappa_0 \cdot M_{0,1,5}) + n_{0,y} \cdot (\kappa_0 \cdot M_{0,2,5})) \\ &- (n_{1,x} \cdot (\kappa_1 \cdot M_{1,1,4}) + n_{1,y} \cdot (\kappa_1 \cdot M_{1,2,4})) \\ &+ (0) \\ &- (n_{3,x} \cdot (\kappa_3 \cdot M_{3,1,1} + \partial\kappa_{3,1} \cdot a_{3,1}) + n_{3,y} \cdot (\kappa_3 \cdot M_{3,2,1} + \partial\kappa_{3,1} \cdot a_{3,2}))) \end{aligned} \tag{A.8}$$

$$\begin{aligned} A_{i,8} =& \Delta\tau \cdot (\\ &- (0) \\ &+ (n_{1,x} \cdot (\kappa_1 \cdot M_{1,1,5}) + n_{1,y} \cdot (\kappa_1 \cdot M_{1,2,5})) \\ &- (0)) \\ &+ (n_{3,x} \cdot (\kappa_3 \cdot M_{3,1,3}) + n_{3,y} \cdot (\kappa_3 \cdot M_{3,2,3})) \end{aligned} \tag{A.9}$$

Here, $n_{i,x}$ denotes the face normal of the i-th face of the center cell in x direction, $M_{i,j,k}$ is the entry of the least squares matrix for the cell i in the j-th row and k-th column and $A_{i,j}$ is the coefficient for the i-th equation of the system in front of the value that has the index number j inside the stencil of this cell. Furthermore κ_i is the heat conductivity of the face i averaged using formula (3.4) and $\partial\kappa_{i,j}$ is the derivative of κ of the i-th face with respect to the cell with the index j inside the flux calculation stencil.

Notice there is a zero, if the cell does not contribute to the flux over a particular cell face. Furthermore, the physical time discretization adds the second term to the coefficient of the center cell $A_{i,1}$.

If one of the fluxes for a certain cell is a flux over a boundary face, the calculations are a bit more difficult. This is due to the face that the solution from the least squares is different. In this case, one can just substitute the matrix M by another expression as follows

$$\tilde{M} = M - N \cdot C \cdot M \tag{A.10}$$

In this case, also the entries $a_{i,j}$ have to be replaced by the boundary values $\widetilde{a_{i,j}}$

References

[1] Charles Hirsch. *Numerical Computation of Internal and External Flows.* Elsevier, 2007.

[2] Frank P. Incropera and David P. DeWitt. Fundamentals of heat and mass transfer, 2001.

[3] Michael Gino Kapper. *A High-Order Transport Scheme For Collisional-Radiative And Nonequilibrium Plasma.* PhD thesis, The Ohio State University, 2009.

[4] Andreas Meister. *Numerik linearer Gleichungssysteme.* vieweg, 2008.

[5] Charles L. Merkle and Mahesh Athavale. Time-accurate unsteady incompressible flow algorithms based on artificial compressibility. In *Computational Fluid Dynamics Conference, 8th, Honolulu*, pages 397–407, 1987.

[6] Yousef Saad and Martin H. Schulz. A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientifitc and Statistical Computing*, 7:856–869, 1986.

[7] Eleuterio F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics.* Springer, 1999.

[8] Fank M. White. *Fluid Mechanics.* WCB/McGraw-Hill, 1999.