Lund University

Bachelor Thesis

Numerical Implementations of the Generalized Minimal Residual Method (GMRES)

Author:
Nils Ivo Dravins

Supervisor:
Philipp Birken

A thesis submitted in partial fulfilment of the requirements for the degree of Bachelor of Science

in the

Division of Numerical Analysis
Centre for Mathematical Sciences

November 2015

"The competent programmer is fully aware of the limited size of his own skull. He therefore approaches his task with full humility, and avoids clever tricks like the plague."

Edsger Dijkstra

LUND UNIVERSITY

Abstract

Division of Numerical Analysis
Centre for Mathematical Sciences

Bachelor of Science

Numerical Implementations of the Generalized Minimal Residual Method (GMRES)

by Nils Ivo Dravins

The generalized minimal residual method (GMRES) is an iterative method used to find numerical solutions to non-symmetric linear systems of equations. The method relies on constructing an orthonormal basis of the Krylov space and is thus vulnerable to an imperfect basis caused by computational errors. There have been attempts to address this issue by devising variations of the method that are less sensitive to poorly conditioned problems. The GMRES algorithm is typically used when the dimensions of the problem are very large, thus it is of interest to investigate ways in which the computational and memory cost of running it can be reduced. One method for doing so involves replacing the matrix-vector multiplications with an approximating function.

This work compares variations of the GMRES algorithm against each other by using it as a solver in simulations mirroring real-world applications.

Contents

Chapter 1

The Generalized Minimal Residual Method

1.1 Background

Through linearization of differential equations, many problems in physics and the numerical sciences can be reduced to linear systems of equations. It is therefore of great interest to have efficient and numerically stable methods for solving them. Challenges to classical methods for solving linear systems appear when looking at large-dimensional cases when memory usage and computational price make them unpractical. Efficient methods can be found if the system possesses certain properties, symmetry being a case example where the symmetric LQ method and minimal residual method algorithm are well suited [4]. Another interesting case is that of large sparse non-symmetric systems, which will be the focus of this thesis.

In 1986 Saad and Schultz first published, through a generalization of the minimal residual method, the so-called GMRES algorithm [2]. This method used Arnoldi's method for generating an orthonormal basis of the Krylov subspace. Since then several ways of implementing the GMRES algorithm have been devised. One of particular interest to this work is one published by Walker in 1988 which replaces the Gram-Schmidt process used in the standard implementation of GMRES, by using Householder transformations to store the basis of the Krylov subspaces [1]. This work aims to examine and compare the numerical properties of these different implementations of the GMRES algorithm.

When dealing with practical problems, the system matrix can often be a Jacobian which

allows for the use of so-called Jacobian-free implementations of the GMRES algorithm (see 1.4). A central theme in this work is their performance when compared to those utilizing full matrix multiplication.

1.2 The Problem

The generalized residual method is an iterative method suited for solving large linear systems of equations

$$Ax = b \tag{1.1}$$

where $A \in \mathbb{R}^{n\times n}$ is a general matrix.

1.2.1 GMRES - The general idea

Firstly let us remind ourselves that the Krylov space of dimension k is defined as [6];

$$K_k(A, r_0) := span\{r_0, Ar_0, ..., A^{k-1}r_0\} \tag{1.2}$$

Now let x_0 denote the initial guess, in each iteration of the GMRES algorithm one has to solve $\min_{x\in x_0+K_k(A,r_0)} \|Ax-b\|_2$, this can be done efficiently if one has an orthonormal basis (ONB) of the Krylov space $K_k(A, r_0)$. Let $\{v_1, v_2, ..., v_k\}$ be such a basis; then one can, by placing each basis vector as a column, construct an orthogonal matrix $V_k \in \mathbb{R}^{n\times k}$. Then each vector $x^{(k)} \in x_0 + K_k(A, r_0)$ can be written as $x^{(k)} = x^{(0)} + V_k y$ for some $y \in \mathbb{R}^k$. Thus we may rewrite the problem as:

$$||b - Ax||_2 = ||b - A(x^{(0)} + V_k y)||_2 = ||b - Ax^{(0)} + AV_k y||_2 \tag{1.3}$$

Now we may introduce $r_0 := b - Ax^{(0)}$, then we get, using the above:

$$\min_{x\in x_0+K_k(A,r_0)} \|Ax - b\|_2 = \min_{y\in\mathbb{R}^k} \|r_0 - AV_k y\|_2 \tag{1.4}$$

As y is now chosen without restrictions from $\mathbb{R}^k$, this is an ordinary least-squares problem. It is clear that this method needs an orthonormal basis of the Krylov space to

construct the matrix V_k. It is the method of how this basis is obtained and saved that marks the main difference between different methods that will be discussed in this work.

1.2.2 The Gram-Schmidt implementation

Let us first consider the Gram-Schmidt method of constructing an ONB commonly known as the Arnoldi process [1,5];

Algorithm 1 Arnoldi's Method. Let $v_1 = \frac{r_0}{||r_0||_2}$

1: **for** m = 1,2,..., **do**
2: a. Set
3: $h_{im} = (Av_m)^T v_i;\ i = 1, 2, ..., m,$
4: $\hat{v}_{m+1} = Av_m - \sum_{i=1}^{m} h_{im} v_i$
5: $h_{m+1,m} = ||\hat{v}_{m+1}||_2$
6: b. If $h_{m+1,m} = 0$, then stop; otherwise, set
7: $v_{m+1} = \hat{v}_{m+1}/h_{m+1,m}$
8: **end for**

With this notation we have the following relation;

$$AV_k = V_{k+1}\bar{H}_k \tag{1.5}$$

Where $\bar{H}_k \in \mathbb{R}^{(k+1)\times k}$ is an upper Hessenberg matrix with one extra row inserted at the bottom which contains only one entry.

$$\bar{H}_k = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,k} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,k} \\ 0 & h_{3,2} & \cdots & h_{3,k} \\ 0 & 0 & \cdots & h_{4,k} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & h_{k+1,k} \end{pmatrix}. \tag{1.6}$$

This can then be used, setting $r_0 = \beta v_1$:

$$||r_0 - AV_k y||_2 = ||r_0 - V_{k+1}\bar{H}_k y||_2 = ||V_{k+1}(\beta e_1 - \bar{H}_k y)||_2 =_{(*)} ||\beta e_1 - \bar{H}_k y||_2 \tag{1.7}$$

(*) Note that this equality only holds true when V_k is an orthogonal transformation, thus an imperfect basis would render this equality false.

Thus we only need to compute the solution of

$$\min_{y \in \mathbb{R}^k} \|\beta e_1 - \bar{H}_k y\|_2 \tag{1.8}$$

which can be done using the QR-decomposition:

$$||Q(\beta e_1 - \bar{H}_k y)||_2 = ||\bar{g}_k - \bar{R}_k y||_2 \tag{1.9}$$

where $\bar{R}_k$ is upper triangular with one aditional zero row at the bottom and $g_k = (\gamma_1, ..., \gamma_{k+1}) \in \mathbb{R}^{k+1}, \bar{g_k} = (g_k, \gamma_{k+1})$.

Then the minimum is obtained when $R_k y = g_k$ (here R_k denotes the upper triangular matrix without the extra zero row at the bottom); thus we have

$$\min_{x \in x_0 + K_k(A, r_0)} \|Ax - b\|_2 = \min_{y \in \mathbb{R}^k} \|r_0 - AV_k y\|_2 = \min_{y \in \mathbb{R}^k} \|r_0 - V_{k+1}\bar{H}_k y\|_2 = |\gamma_{k+1}| \tag{1.10}$$

Further we know that $R_k y = g_k$ has a unique solution, so it is sufficent to check γ_{k+1} for a chosen tolerance. When this is fulfilled, we simply solve $R_k y = g_k$ and since R_k is upper triangular, this can be done with backwards substitution. When this is done, we can compute a new solution $x = x^{(0)} + V_k y$. Note that we only solve for y when the tolerance is satisfied, when this is not the case we simply iterate futher.

Algorithm 2 Gram-Schmidt implementation of the GMRES [2]

1: Start: Choose x_0 and compute $r_0 = b - Ax_0$ and $v_1 = r_0/||r_0||_2$
2: **for** j = 1,2,..., **do**
3: a. Compute and save Av_j as to only compute it once.
4: b. Set $h_{i,j} =< Av_j, v_i >, i = 1, 2, ..., j$
5: c. Set $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$
6: d. Set $h_{j+1,j} = ||\hat{v}_{j+1}||_2$
7: e. Set $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$
8: **end for**
9: e. Form the approximate solution: $x_k = x_0 + V_k y_k$, where y_k minimizes (1.8)

At the heart of Arnoldi's method used in this implementation is the Gram-Schmidt process of orthonormalization. We will therefore refer to this implementation of the GMRES algorithm as the Gram-Schmidt implementation.

1.2.3 The Householder implementation

There is another option when seeking to form an ONB of the Krylov space employing Householder transformations [1]. These can be represented by a matrix of the form $P = I - 2uu^T$, where I is the n by n unit matrix and u is a vector of length n with $||u||_2 = 1$ [4]. We will call u the Householder vector that determines P. A great numerical advantage in working with Householder matrices is that they can be fully characterized by a vector, thus requiring less space for storage compared to saving a full matrix. Another useful property is that the matrix itself must not be explicitly constructed for a vector to be multiplied with it, this can instead be done with inner products, saving computational effort.

Below is a pseudocode that uses Householder transformations to construct an ONB [1];

Algorithm 3 Householder orthonormalization. Suppose v_1 is given with $||v_1||_2 = 1$

1: Choose P_1 such that $P_1 v_1 = e_1$, this can be done by setting $u := ||v_1||_2 e_1 - v_1$, then $P_1 = \mathbb{I} - \frac{2uu^T}{u^T u}$
2: **for** m = 1,2,..., **do**
3: a. Set $v_m = P_1 ... P_m e_m$
4: b. If $(v_1, Av_1, ..., Av_m)$ has rank m, then stop;
5: otherwise, choose P_{m+1} such that $P_{m+1} ... P_1 (v_1, Av_1, ..., Av_m)$ is upper-triangular (here P_{m+1} is chosen analogously to the example in 1.)
6: **end for**

This method is used in the implementation of the GMRES method below. As in the Gram-Schmidt implementation, the least-squares problem is solved by maintaining a QR-factorization and then solving an upper-triangular system with backward substitution.

Algorithm 4 Householder implementation of the GMRES [1]

1: Compute $r_0 = b - Ax_0$ and determine P_1 such that $P_1 r_0 = \pm||r_0||_2 e_1 \equiv w$
2: **for** m = 1,2,..., **do**
3: a. Evaluate $v \equiv P_m ... P_1 A P_1 ... P_m e_m$
4: b. If $v^{(m+1)} = ... = v^{(n)} = 0$, then proceed to step (e); otherwise, continue.
5: c. Determine P_{m+1} with a Householder vector having first m components zero such that $P_{m+1}v$ has zero components after the $(m+1)$st.
6: d. Overwrite $v \leftarrow P_{m+1}v$
7: e. If $m > 1$, overwrite $v \leftarrow J_{m-1}...J_1 v$
8: f. If $v^{m+1} = 0$, proceed to step (i); otherwise, continue.
9: g. Determine a Givens rotation J_m acting on components m and $m+1$ such that $(J_m v)^{(m+1)} = 0$
10: h. Overwrite $v \leftarrow J_m v$ and $w \leftarrow J_m w$
11: i. Set $R_m = \begin{cases} (v), & \text{if } m = 1 \\ (R_{m-1}, v) & \text{if } m > 1 \end{cases}$
12: j. If $|w^{(m+1)}| \leq TOL$ or $m = MAXIT$, then solve for y_m and overwrite x_0 with x_m; otherwise, increment m.
13: **end for**
14: a. Determine y_m which minimizes $||w - R_m y||_2$, by solving a $m \times m$ upper-triangular system with the first m rows of R_m as the coefficent matrix and the first m components of w as the right-hand side.
15: **for** k = 1,2,...,m **do**
16: b. Overwrite $x_0 \leftarrow x_0 + y_m^{(k)} P_1 ... P_k e_k$
17: **end for**

In this implementation, the basis is not explicitly stored but instead the basis vectors of the Krylov space $v_m = P_1 ... P_m e_m$ are generated as part of the calculations when needed. The Householder vectors will be of decreasing length, the first will have a length equaling the system dimension and then they will decrease by one for each iteration. It then makes sense to add another row to the upper triangular matrix R, so that they can be stored in place of the zeroes below the diagonal; in this way memory can be saved.

1.2.4 Expected differences

We will consider two main issues; the first will be sensitivity to problem condition which is determined by the different approaches for generating a basis, the second will be a consideration of the memory use and computation effort demanded by each of the algorithms.

We first consider the Arnoldi iteration using the modified Gram-Schmidt process [4], which is mathematically equivalent to the classical Gram-Schmidt but possesses superior numerical properties. There is a significant risk of error if the vectors on which it operates

are not sufficently independent. Let $S = (s_1, ..., s_m)$ be an n by m matrix whose columns are to be orthogonalized. After applying the modified Gram-Schmidt we obtain a matrix $Q = (q_1, ..., q_m)$ and a unit rounding error eps. Björck [3] has then shown that

$$Q^T Q = I + E;\ where\ E \approx eps\kappa_2(S), \tag{1.11}$$

where $\kappa_2(S)$ is the condition number given by the ratio of the largest singular value to the smallest. This means that if $\kappa_2((v_1, ..., v_m, Av_m))$ is large, the v_{m+1} generated by the m-th step of the Arnoldi iteration may have significant non-zero components in the span of $\{v_1, ..., v_m\}$. It has been suggested [2] that this imperfect orthogonalization can be a significant source of errors in practial uses of the GMRES algorithm.

We now consider the alternative method using Householder transformations, which theoretically should be less sensitive if the orthogonalized vectors are not very independent. A Householder transformation is an orthogonal linear transformation of the form $P = I - 2uu^T$ [4], where I is the n-dimensional unit matrix and u is a vector with n entities. Here we note that we need only n entries to characterize the $n \times n$ matrix P. We further note that given two normalized vectors, it is easy to calculate the vector u that defines a Householder reflection of one vector onto the other.

To orthonormalize the columns of $S = (s_1, ..., s_m)$, we determine a series of Householder transformations $P_1, ..., P_m$ such that $P_m...P_1S = R$ is upper triangular. A natural approach would be to determine $P_1, ..., P_m$ inductively by imposing the requirement that $P_k...P_1(s_1, ...s_k)$ be upper triangular for each $k \in \{1, ..., m\}$. If, further, we have that $P_m...P_1S = R$, then this directly implies that the first $k-1$ components of the k-th Householder vector must be zero. We can use this fact in that we avoid storing the zeros explicitly. Finally, using that the Householder transformation defines an orthogonal transformation, we have that $S = P_1...P_mR$. Thus, the matrix Q constructed by the columns of $P_1...P_m$ gives us the orthonormalized columns of S. If Q is computed using floating-point arithmetic then, from [3], we have the following relation for the error:

$$Q^T Q = I + E;\ where\ E \approx eps, \tag{1.12}$$

where u is the unit rounding error. We thus note that the error term depends only on the rounding error, not on the condition number as we saw in the Gram-Schmidt

method considered above. These superior properties form a theoretical motivation for implemententing a GMRES algorithm using Housholder transformations.

1.3 Memory usage and numerical complexity

The theoretical considerations indicate that the Householder implementation has superior stability. While the Householder method requires sligthy less memory compared to the Gram-Schmidt implementation, it requires more individual computations, however, it can be shown [1] that the increase in arithmatic is always less than a factor of three.

This is chiefly caused by the fact that in the Gram-Schmidt implementation, the basis of the Krylov space is explicitly calculated and stored, and thus can always be called in from storage when needed. In the Householder implemenation, the Householder vectors are stored instead, they are then used to calculate the basis of the Krylov space needed. Except for this difference, both methods are almost the same with regard to complexity.

The Householder implementation (Algorithm 4) requires about $2n$ multiplications and one matrix-vector product in part (1) for each m. Steps (3a) and (6d) require about $4nm$ multiplications and one matrix-vector product for each m. If the for-loop in (2) is carried out a total of m times, this results in about $2mn(m+1)$ multiplications and m matrix-vector products. For each iteration $k, (k = 1, ..., m)$ in step (16b) requires about $2nk - n$ multiplications and thus contributes with approximately nm^2 multiplications in total. The entire algorithm then requires about $n(3m^2 + 2m + 2)$ multiplications and $m + 1$ matrix-vector products.

The Gram-Schmidt implementation requires one matrix-vector product to compute the initial residual and about $2n$ multiplications are needed to normalize the residual. At the m-th iteration, the cost of computing one additional basis of the Krylov space is one matrix-vector multiplication and about $2n(m + 1)$ multiplications. If a total of n iterations are computed, the total cost is then m matrix-vector multiplications and $nm(m + 3)$ scalar multiplications. Computing the approximate solution after m iterations then requires nm additional scalar multiplications. In total, the Gram-Schmidt implementation costs about $n(m^2 + 4m + 2)$ scalar multiplications and $m + 1$ matrix-vector multiplications.

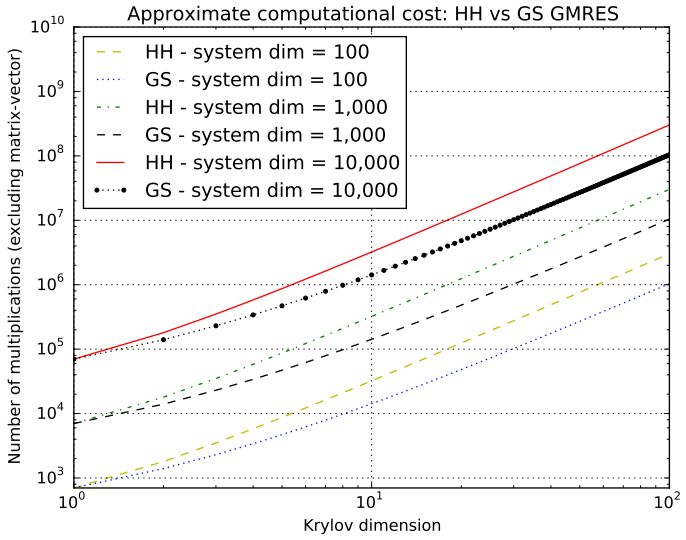

FIGURE 1.1: Approximate number of scalar multiplications, excluding the $m+1$ matrix-vector multiplications, required for the two different implementations. (HH=Householder, GS=Gram-Schmidt)

The additonal arithmetic cost of the Householder implementation may be significant for large-dimensional systems in combination with a large dimension of the Krylov space. In the case where the maximum number of iterations ($maxit$) is small, this should cause no significant increases in computation time. A significant part of the computational effort is devoted to calculating matrix-vector products which are relatively costly. There are methods by which these can instead be approximated, sacrificing accuracy for quicker computations and/or reduced memory cost. In some of the numerical experiments done in the scope of this work, an approximate method for calculating the matrix-vector multiplication was used. Methods that replace matrix multiplication with this function are known as Jacobian-free methods. Because the matrix-vector multiplications can be replaced by approximating functions, they are left out from this consideration. The numerical and memory cost of these approximating functions should thus be considered separately.

1.4 Jacobian-free implementations

Jacobian-free methods are applicable when the system matrix is a Jacobian, e.g., if the system considered is a linearization of a system of nonlinear differential equations. As the system matrix is used only for matrix-vector multiplication, this step can be replaced by the following quotient approximation [6];

$$\frac{\delta \mathbf{F}(\mathbf{y})}{\delta \mathbf{y}}\mathbf{v} \approx \frac{\mathbf{F}(\mathbf{y}+\epsilon\mathbf{v}) - \mathbf{F}(\mathbf{y})}{\epsilon} \tag{1.13}$$

Thus all system matrix vector-products can be replaced by a difference quotient. If the parameter ϵ is chosen to be very small, the approximation performs better but cancellation errors become a substantial problem. A choice that avoids cancellation problems but still gives a moderately small ϵ is given by;

$$\epsilon = \frac{\sqrt{eps}}{||v||_2} \tag{1.14}$$

Where *eps* is the machine accuracy. One should also note that when using the GMRES, this approximation performs well because the vectors in the matrix multiplications are normalized [6].

Besides the storage advantage, this method also offers ease of implementation in that, instead of computing the Jacobian by analytical means of difference quotients, we need only function evaluations. Furthermore, this allows us to increase the convergence speed of the of the Newton scheme; as we never need to compute the Jacobian but only an approximation thereof. The method used when employing matrices is always of Newton-type and thus has first-order convergence. The matrix-free approximation, however, is for the second-order Jacobian and can thus obtain second-order convergence, provided the proper forcing terms are employed. Regarding convergence, the following theorem [6] describes the convergence behavior of the Jacobian-free Newton-Krylov scheme:

1.4.1. Theorem 1. *Using the standard assumptions [6], there are δ, $\bar{\sigma}$ and C_G such that if x_0 is in a δ-neighborhood of x^* and the sequences $\{\eta_k\}$ and $\{\epsilon_k\}$ satisfy*

$$\sigma_n = \eta_n + C_G\epsilon_n \leq \bar{\sigma},$$

then the Jacobian-free Newton-GMRES iteration converges linearly. Moreover,

- *if $\sigma_k \to 0$, the convergence is superlinear and*
- *if $\sigma_k \leq K_\eta ||\boldsymbol{F}(x_k)||^p$ for some $K_\eta > 0$ and $p \in [0, 1]$, the convergence is superlinear with order $1 + p$*

This theorem says that for superlinear convergence, the ϵ in (1.14) needs to approach zero, but as pointed out, this leads to cancellation errors. Because of this, a behavior like quadratic convergence can only be expected in the initial stages of the iterations, where ϵ is small compared to η_k.

1.5 Method and goal

In the scope of this work we will use different variations of the GMRES algorithm on simulations of practical problems mirroring real-world applications of the code.

There are three main distinctions to be made, one is the version of GMRES implemented, that is the Householder or Gram-Schmidt methods introduced earlier. We will further test how these methods perform when the matrix-vector multiplication is replaced by an approximating function in the so-called Jacobian-free methods (JF). The JF methods themselves allow for further comparison relating to the degree of the function used when approximating the matrix-vector multiplication (for an overview of the structure of the testing please see figure 1.5 on page 15).

The parameters we will look at are the residual $||Ax - b||_2$ and estimates thereof given by the algorithm, as well as the result when Ax is found by an approximating function. The estimates provided by the algorithm are theoretically equivalent to the calculated residual when no approximation function is used, though they may still be subject to computing errors. The interest is to see how well this equality translates when using the Jacobian-free approximation, as this is of vital importance to a reliable solver. Further,

we will compare the performance of the JF methods to those using the full Jacobian for matrix multiplication.

All of the simulations were run on a PC running a 32-bit Linux OS and all were made within the Tau Code library which is written in C++. The author of this work has written the solver implementing the Jacobian-free Householder method, whereas the Jacobian-free Givens implementation was provided to him by the supervisor and the Householder utilizing full matrix multiplication for the Jacobian was obtained from the libraries used. All input parameters used are included in the appendixes A,B and C.

The ultimate aim of these experiments is to assess the performance of the Jacobian free implementations against those utilizing full matrix multiplication.

1.5.1 Expected differences

When using full matrix multiplication we should expect $||Ax - b||_2$ to be equal to the value given by the algorithm (the w-vector), save for rounding errors. The Jacobian-free methods introduce a potential for further error caused by the approximating function, the size of which it is the goal of this work to determine.

When using the Gram-Schmidt implementation we expect a larger error sensitivity to rounding errors caused by an imperfect orthonormal basis than that in the Householder implementation. While this error is dependent on problem condition, it should only be visible in late iterations as the error propagates with the dimension of the Krylov space. In the first couple of iterations no noticeable difference against the Householder implementation is to be expected.

1.5.2 Comparison of practical implementations

We look at three different problems on which we test several variations of the GMRES algorithm. The first distinction we make is the order of discretization used; in the scope of this work we will consider those of the first and second order (see Chapter 2). For each problem and discretization we then examine the residual behavior in dependence of the dimension of the Krylov space, this is done for both the Householder and Gram-Schmidt implementations. In the case of the first-order discretization, the Tau Code library allows for a construction of the matrix, thus in this case we can compare the performance

of the Jacobian-free methods to a Householder implementation using explicit matrix multiplication.

1.5.3 Airfoil (NACA0012), wind turbine & flanged shaft

In the first problem we consider a particular airfoil shape which is one of many designs developed and classified by the National Advisory Committee for Aeronautics (NACA), hence it is known by the code NACA0012. The system has a dimension of 18,420 making it the lowest-dimensional problem considered within the scope of this work.

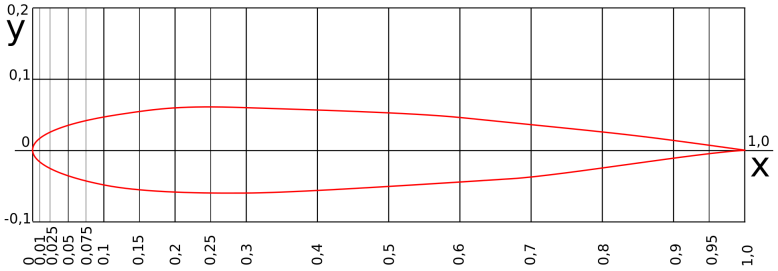

FIGURE 1.2: Shape of the NACA0012 airfoil [F1]

The second problem we consider is a wind turbine where the system dimension is 97,380.

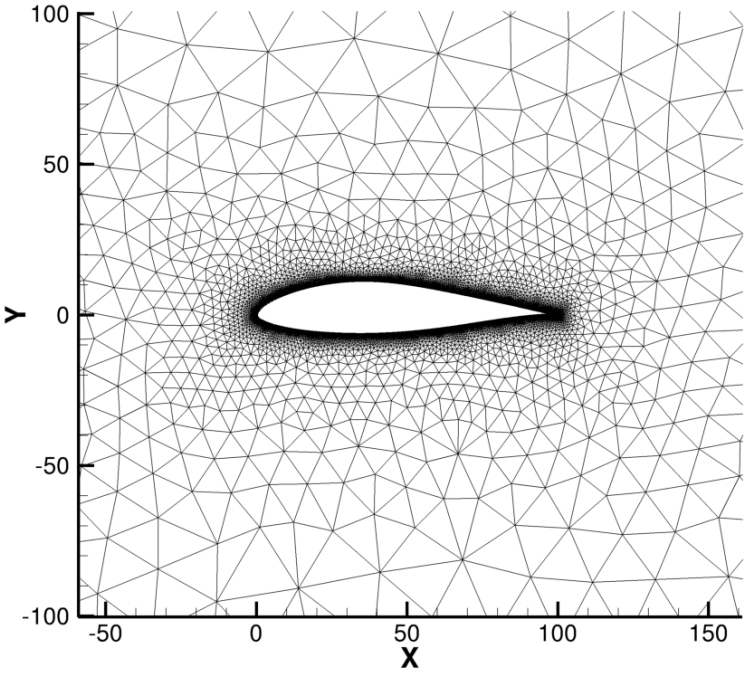

FIGURE 1.3: Detail of the mesh used when simulating the wind turbine.

Finally we look at a simulation of a flanged shaft where the systems dimension is 568,208, making it the highest-dimensional system considered.

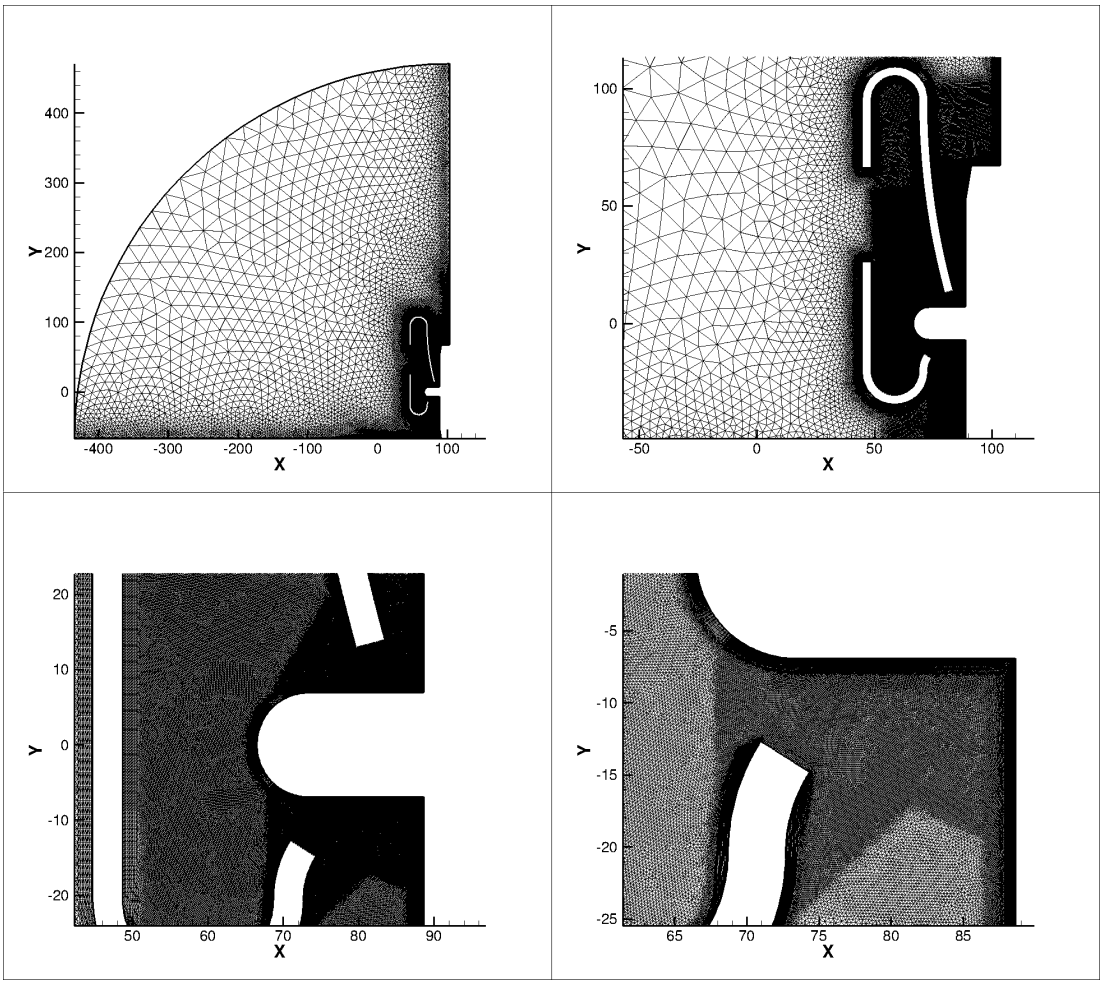

FIGURE 1.4: Mesh used when simulating the flanged shaft [7]

In the interest of giving the reader a clear overview of the comparisons made, below is a flow-chart showing the structure of the experiments and the groupings of the results.

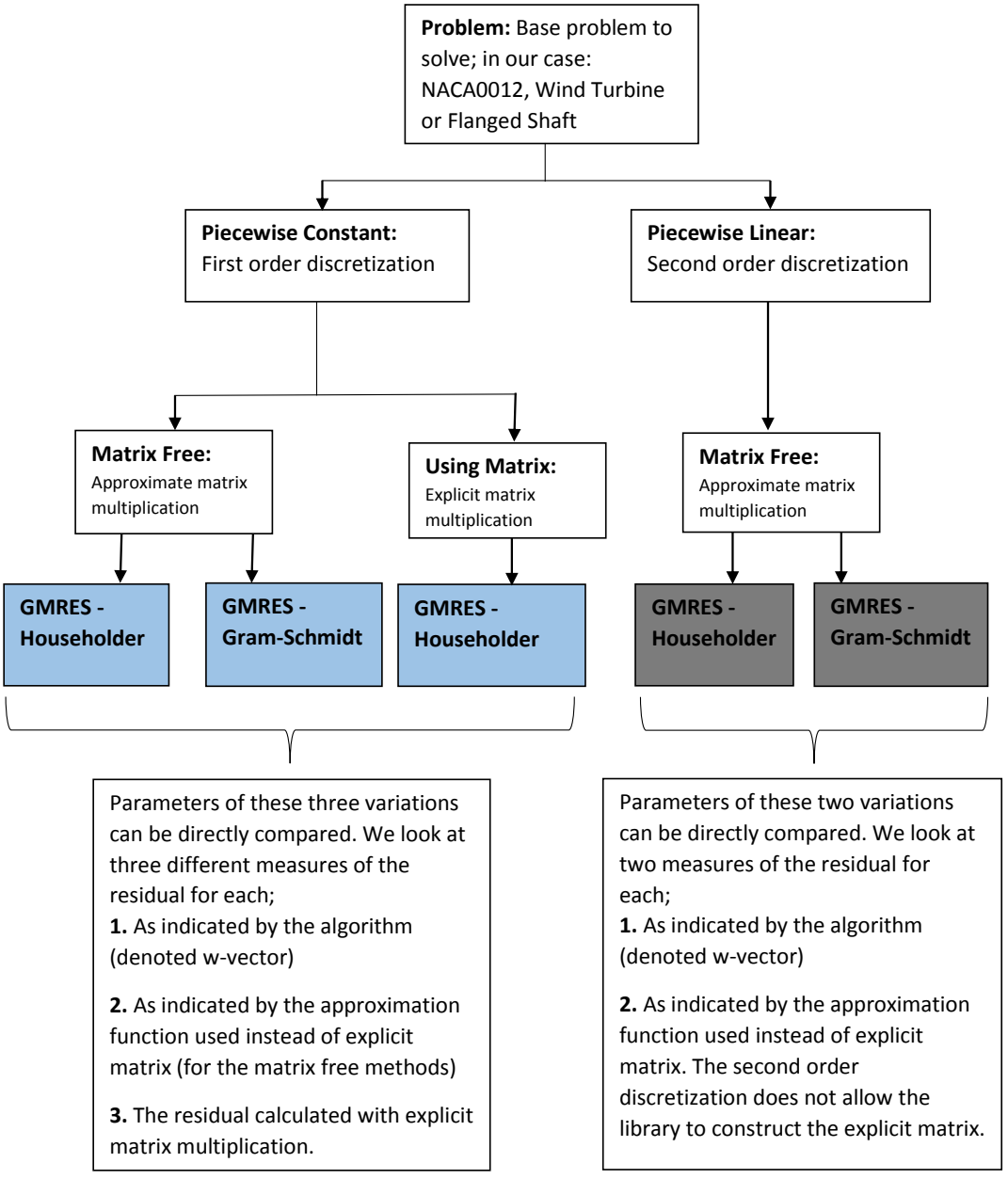

FIGURE 1.5: Flowchart: an overview of the structure of the testing

Chapter 2

Numerical Experiments and Analysis

2.1 First-order discretization

We first look at the data obtained when running the simulations (described in 1.5.3) using first-order discretization, that is when individual cells in the mesh contain piecewise constant functions. The initial guess for the solution of Eq. (1.1) in all simulations is the zero vector of system dimension. Using discretization of the first order we will compare two variations of matrix-free GMRES solvers, one being the Householder implementation described in Algorithm 4, the other being the Gram-Schmidt implementation described in Algorithm 2. The results of the two matrix-free methods will then be compared to a Householder implementation using explicit matrix multiplication. This will be done for all three problems introduced in the previous chapter.

Each plot will thus contain three approximations of the residual for the two matrix-free methods and two for that of the matrix method. In the interest of keeping cluttering of the plots to a minimum, the following abbreviations are used in the legends: HH refers to Householder, GS to Gram-Schmidt, JF to Jacobian-free and M to Matrix. The residuals calculated using the approximation function replacing matrix multiplication are denoted as approximation, w-vector refers to the residual as indicated by the algorithms. $||Ax - b||_2$ is the residual explicitly calculated using proper matrix multiplication. This notation will be used henceforth in all plots of the residuals.

2.1.1 NACA0012

Below are the data obtained when running the NACA0012 simulation using the Jacobian-free Gram-Schmidt implementation:

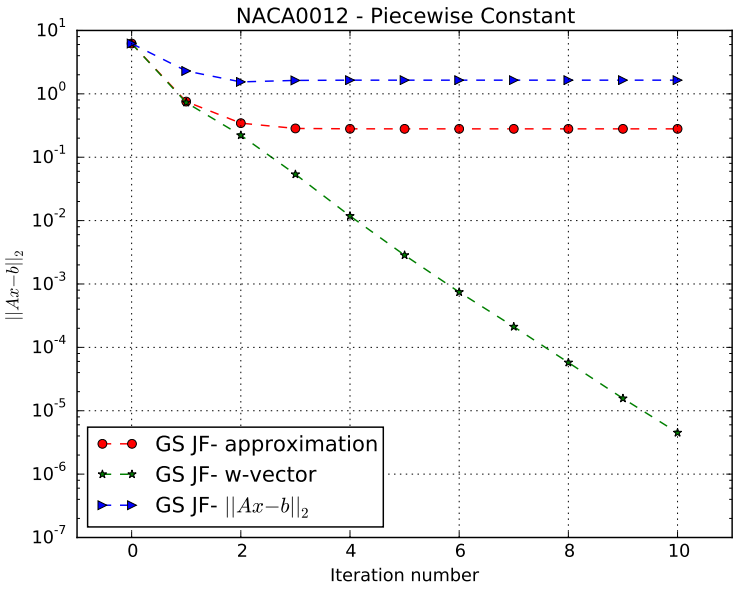

FIGURE 2.1: NACA0012 - Residuals for GS-JF, using first-order discretization.

We see that the indicated residual in the w-vector follows the approximated residual only in the first iteration, then the approximated residual stabilized while the indicated residual continues falling. The residual that is found by explicitly calculating $||Ax - b||_2$ falls in the first two iterations, then rebounds slightly and lies constant. It should be noted that the calculated residual is significantly larger than the approximated residual.

The fact that the approximation diverges from the calculated residual at the first iteration raises questions about the accuracy of the approximation function used instead of the explicit matrix multiplication. This can be stated as: at iteration one we are considering a one-dimensional Krylov space and errors of the kind arising from an imperfect basis are not expected to have an impact.

The observation that the w-vector starts to show significant divergence from the approximated residual at iteration two further adds to the suspicion that an error is present in the matrix-vector approximation. As the indicated (w-vector) and the approximated residual are mathematically equivalent, we should expect only computational rounding errors, with a two-dimensional basis these would be expected to be negligible.

It should however be noted that other issues cannot be excluded.

We now look at the results obtained using the Jacobian-free Householder implementation:

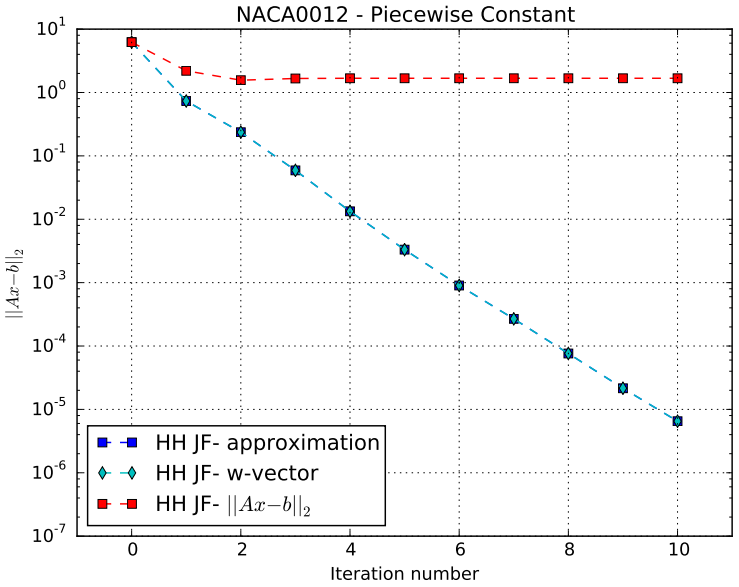

FIGURE 2.2: NACA0012 - Residuals for HH-JF, using first-order discretization.

As we saw when using the Jacobian-free Gram-Schmidt implementation, we see a divergence between the calculated residual and the other two metrics at the first iteration. Again this suggests that the accuracy of the approximating function may not be satisfactory for obtaining a useful result.

A major difference compared to the GS-implementation is that the approximated residual follows the indicated residual which is consistent with the theory outlined previously. This could mean that in the case of this problem, the Householder implementation is more stable than the Gram-Schmidt one or indeed, that there is a bug in the code of the Gram-Schmidt implementation.

In any case the indicated residual seems to be an accurate estimate of the approximated residual but none of them can be trusted to represent the calculated residual $||Ax - b||_2$.

Finally we look at the case where we use explicit matrix multiplication with the Householder implementation:

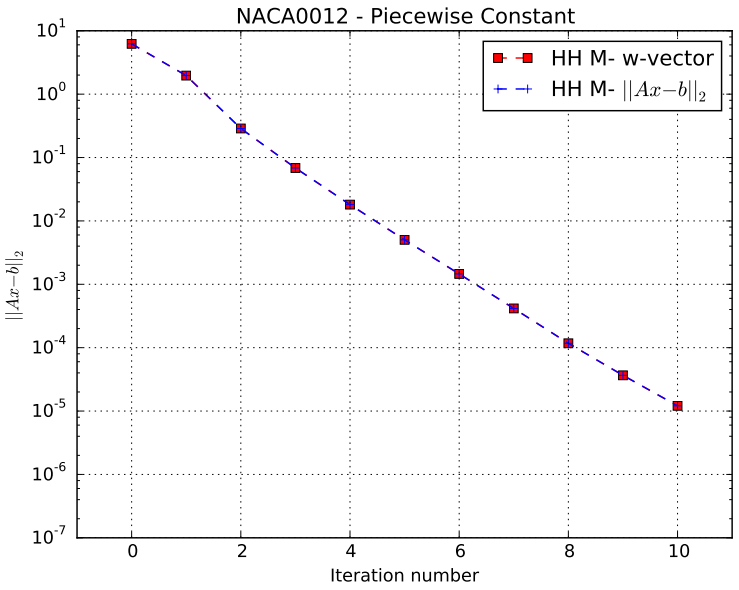

FIGURE 2.3: NACA0012 - Residuals for HH-M, using first-order discretization.

In this case, we are using the full matrix which means that we no longer have an approximating function to which the results can be compared. We can however see how the w-vector compares to the calculated residual $||Ax - b||_2$.

What we see is that for all iterations the indicated residual matches, up to rounding errors, the calculated residual. This is consistent with the theory of them being mathematically equal and serves to reinforce earlier suspicions that the approximating function in the Jacobian free methods is lacking in accuracy, as the only change in the algorithm is its replacement with full matrix multiplication.

2.1.2 Wind turbine

We now consider the data obtained from the wind turbine simulation using the Jacobian-free Gram-Schmidt implementation:

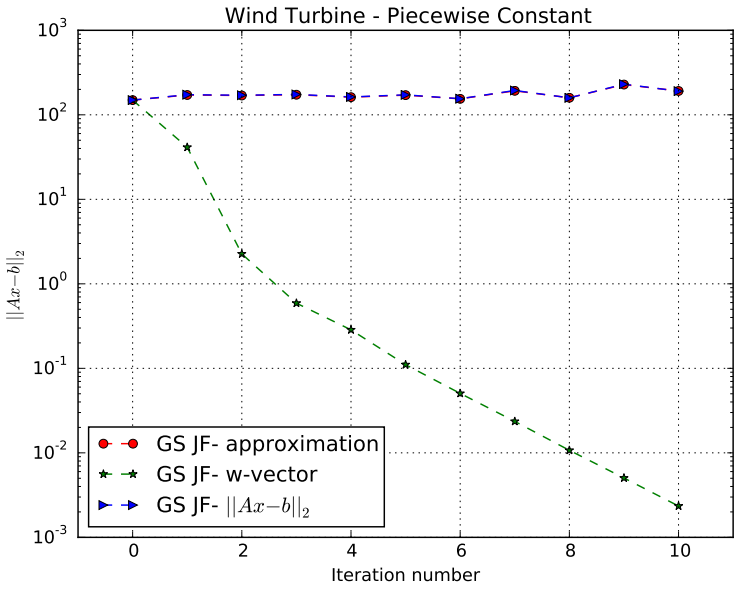

FIGURE 2.4: Wind turbine - Residuals for GS-JF, using first-order discretization.

We observe that the indicated residual of the w-vector diverges sharply from the other two metrics at the first iteration and then keeps falling. In this case the approximated and the calculated residual both indicate a residual slightly larger than the initial guess and seem to follow each other with regard to their values. We would expect the indicated residual to match the approximated residual in the case of a stable implementation, regardless of the accuracy of the approximation, the fact that we do not see this suggests to the author that the Gram-Schmidt implementation is faulty or contains a bug.

Taken as a whole, this behavior was not observed in the NACA0012 simulation and it is not consistent with the theoretical predictions made in the scope of this work.

Next, we examine the data obtained from the wind turbine using the Jacobian-free Householder implementation:

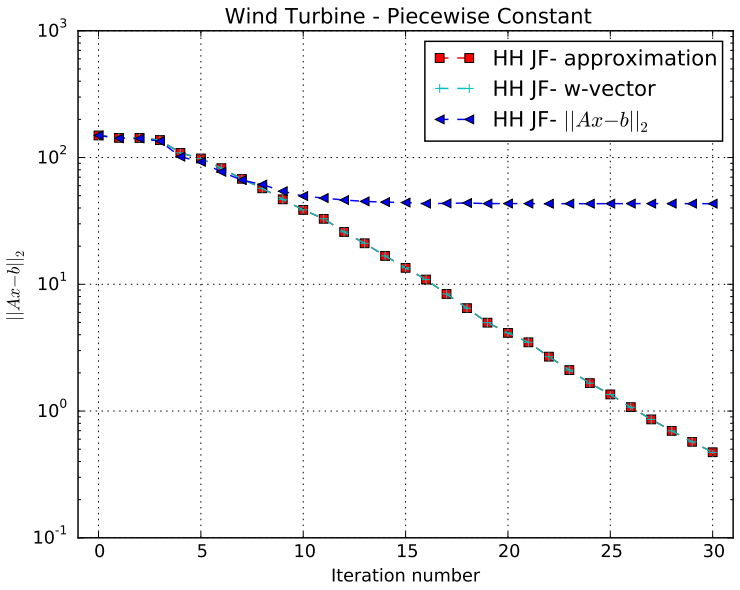

FIGURE 2.5: Wind turbine - Residuals for HH-JF, using first-order discretization.

In this case we see that for the first four iterations all metrics follow each other, then the calculated residual $||Ax - b||_2$ begins to diverge from the other two. The residual as indicated by the w-vector follows the approximated residual in all iterations considered.

The first thing to note is that this behavior is consistent with the behavior previously seen when using the Jacobian-free Householder implementation in the NACA0012 simulation. That is to say, the algorithm seems to be performing as expected in that w-vector and the approximation match and the most likely cause of the divergence seen in the calculated residual once again is thought to be an imprecise approximating function.

Finally we look at the results when using full matrix-multiplication in the Householder implementation:

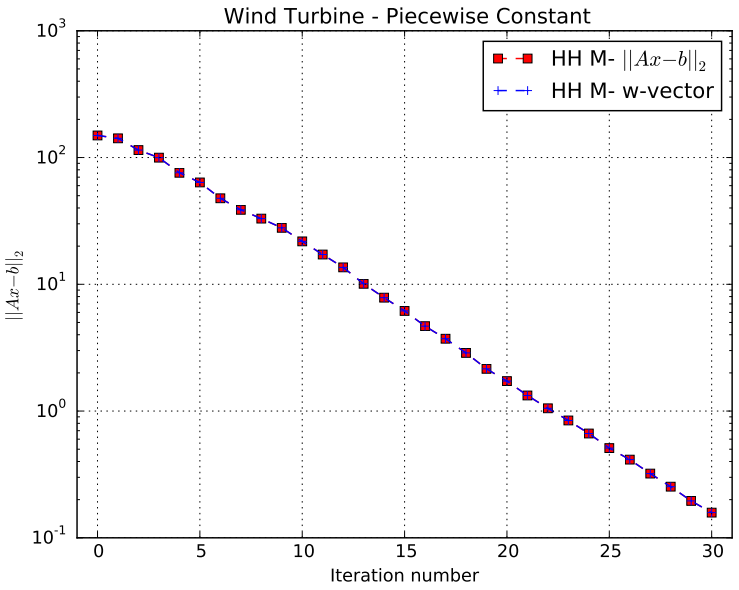

FIGURE 2.6: Wind turbine - Residuals for HH-M, using first-order discretization.

We see a familiar behavior to that observed in the NACA0012 experiment in that the indicated residual closely follows the calculated $||Ax - b||_2$. This is wholly consistent with the theoretical predictions in that both measures are mathematically equivalent and should differ only by rounding errors caused when calculating the basis of the Kyrlov space.

This result serves to further strengthen the argument that the approximating function is lacking in accuracy.

2.1.3 Flanged shaft

Finally we consider the flanged shaft. We begin by looking at the Jacobian-free Gram-Schmidt implementation:

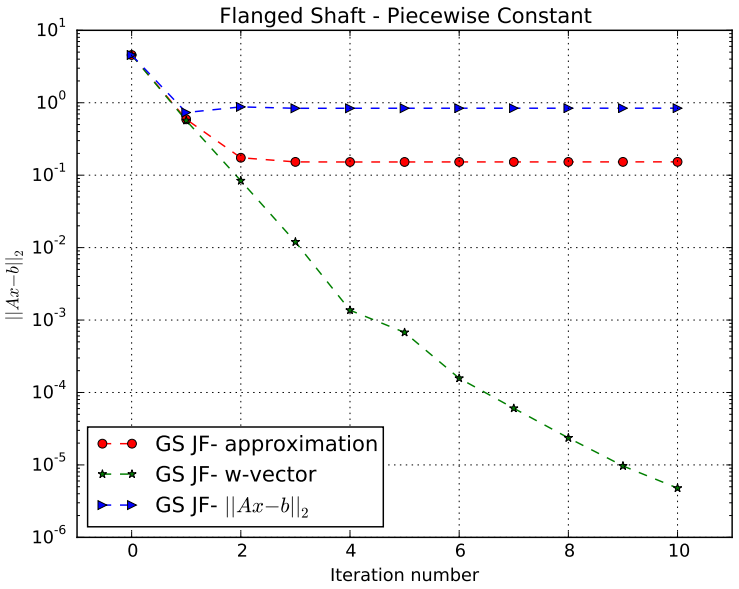

FIGURE 2.7: Flanged Shaft - Residuals for GS-JF, using first-order discretization.

We see a behavior similar to the two previous simulations in that the indicated residual falls consistently for each iteration while the approximated and the calculated residuals diverge in the first iterations to then stabilize at different levels. In this case, the calculated residual falls in the first iteration to then slightly rebound and stabilize.

The approximated residual seems to largely follow the indicated residual in the first iteration but shows significant divergence in the second. Again, the discrepancy between approximated and calculated residual indicates a faulty approximation function while the difference between the indicated and approximated residuals could be the result of a bug in the GS-implementation.

Next we examine the results when using the Jacobian-free Householder implementation:

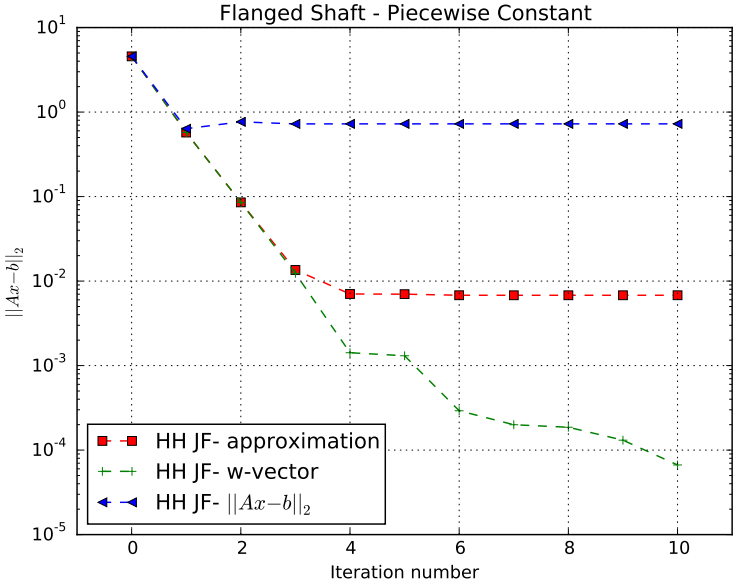

FIGURE 2.8: Flanged Shaft - Residuals for HH-JF, using first-order discretization.

This is an interesting result as it differs from the behavior previously seen when using the Jacobian-free Householder in that after iteration four, the indicated and approximated residuals begin to diverge, in the previous two experiments this was not observed. The reason for this could be that the approximation function presents an upper limit to how small the residual can be made. This suspicion is strengthened by the result presented below where the approximating function has been replaced with explicit matrix multiplication.

As in all previous experiments the calculated residual falls for the first iteration to then slightly rebound and stabilize. The solution found does not correspond the indicated residual which is used as the termination criterion.

Finally we look at the Householder implementation using full matrix multiplication:

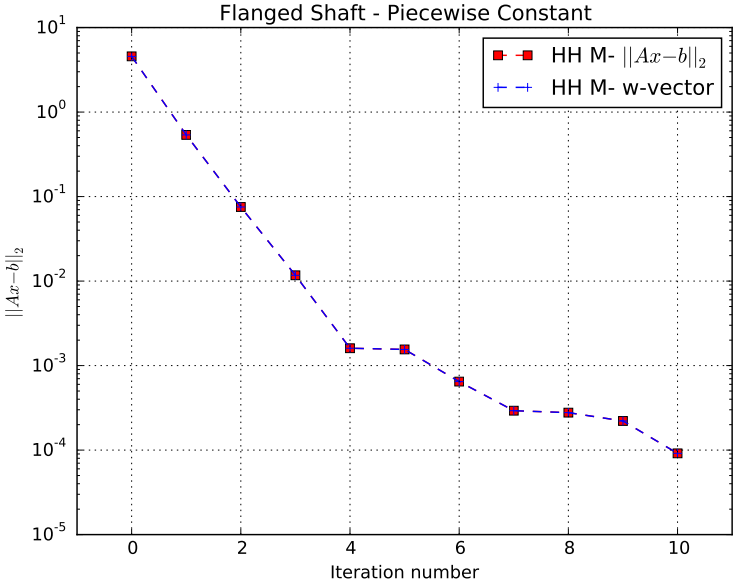

FIGURE 2.9: Flanged Shaft - Residuals for HH-M, using first-order discretization.

We see a behavior consistent with all previous simulations in that the calculated and indicated residuals match for all iterations considered. This is in line with what the theory predicts and strengthens the suspicion presented in the JF HH-implementation above, that is that the approximating function can be a limiting factor for the approximated residual as well as the calculated residual.

2.2 Second-order discretization

We now examine the performance of the two matrix-free methods, Householder and Gram-Schmidt, when using the second-order discretization. Here the individual cells in the mesh will contain piecewise-linear functions instead of the piecewise-constant as was the case when using first-order discretization. A important difference to note is that the Tau Code library does not permit to generate a full matrix when using the second-order discretization, thus we will only compare the residual as indicated by the algorithm (as before denoted w-vector) and the approximated residual calculated with the function replacing matrix-vector multiplication.

2.2.1 NACA0012

We begin by examining the results from the NACA0012 example.

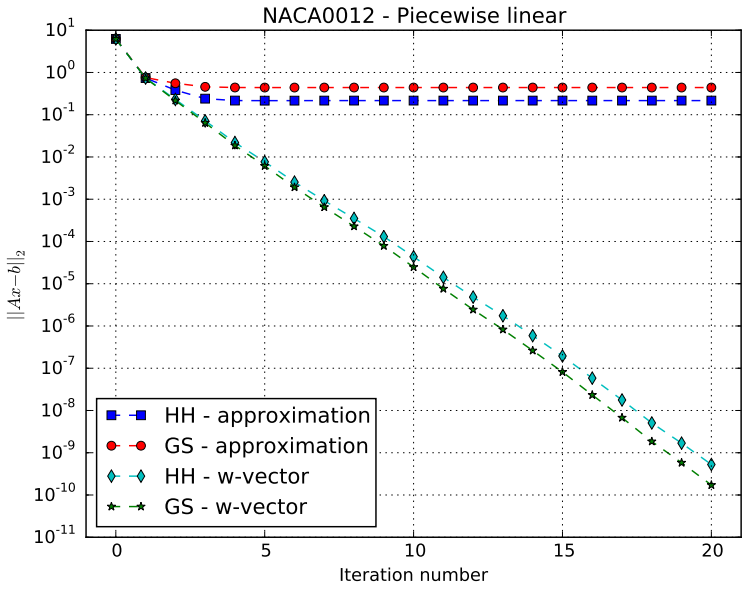

FIGURE 2.10: NACA0012 - Residuals using second-order discretization.

Again we see that, for both implementations, the indicated residual from the w-vector falls consistently with each iteration, yet the approximated residual only noticeably falls in the first few iterations. All measures indicate the residual to be smaller than that produced by the initial guess. A difference from the first-order discretization of NACA0012 is that the w-vector and the approximated residual of the Householder implementation no longer match, as they begin to exhibit significant divergence at the second iteration.

2.2.2 Wind turbine

We now consider the wind turbine example.

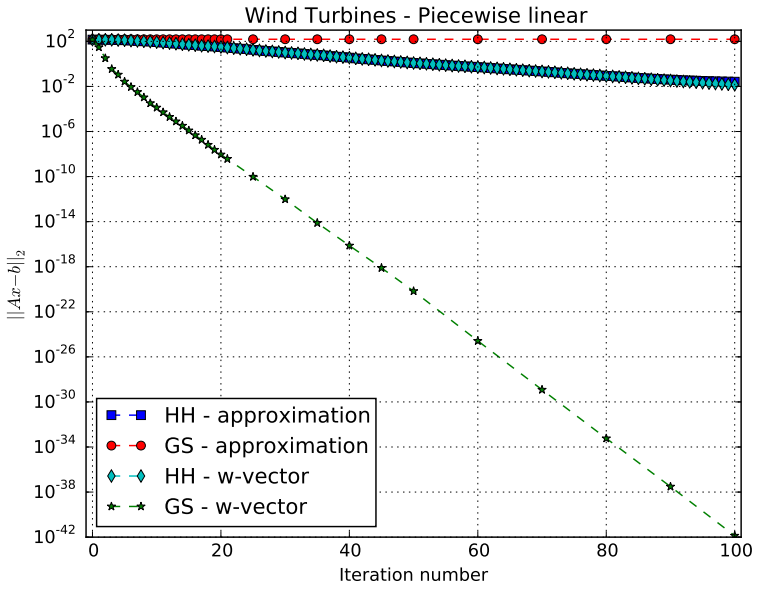

FIGURE 2.11: Wind turbine - Residuals using second-order discretization.

We see that the matrix-free Gram-Schmidt produces a quickly falling w-vector-residual while the calculated residual does not noticeably change from that produced by the initial guess. In this case the rapid fall of the w-vector of the Gram-Schmidt results in a large area of interest making individual changes in the Householder difficult to see, we therefore plot this separately, see figure below.

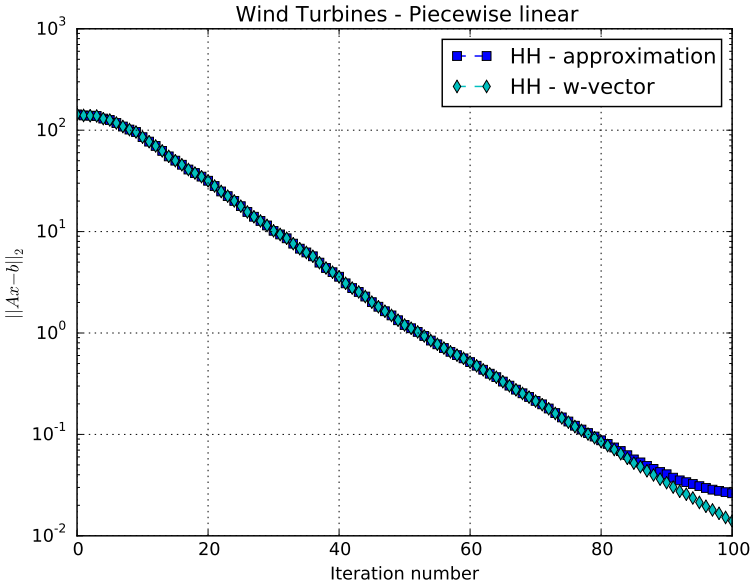

FIGURE 2.12: Wind turbine - Residuals using second-order discretization - Householder only.

In this case we see that for all but the last iterations, the approximated residual follows the indicated ones, but around iteration 90 the two estimates begin to diverge. This is similar to the results seen in the first-order iteration where the Householder w-vector initially followed the approximated residual.

2.2.3 Flanged shaft

Finally we consider the flanged shaft example.

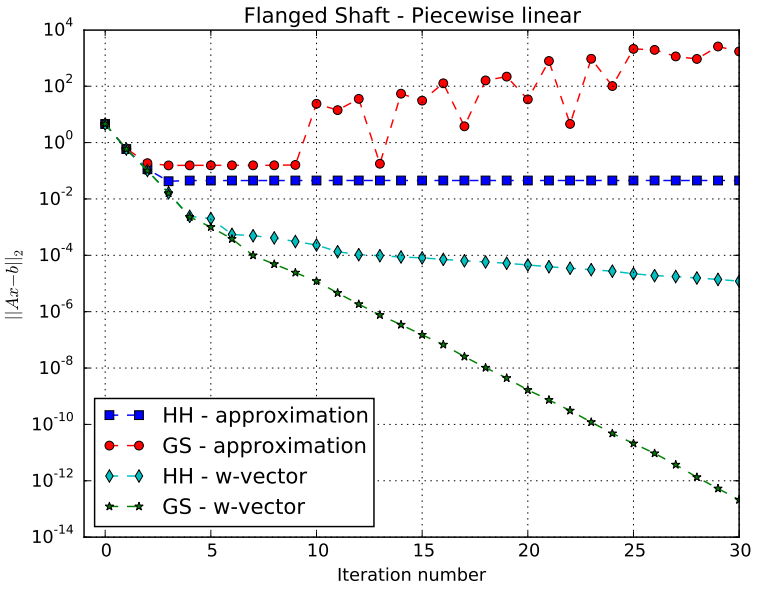

FIGURE 2.13: Flanged shaft - Residuals using second-order discretization.

Here it seems that the matrix-free Gram-Schmidt implementation displays unstable behavior in that the indicated residual falls consistently with each iteration while the approximated residual initially seems to converge to a level slightly above that seen in the Householder implementation, and then exhibits an increasing saw-tooth pattern leading to an estimated residual significantly larger than that produced by the initial guess.

The behavior of the Householder is consistent with previous results in that the indicated residual follows the approximated residual in the initial iterations to then lie dormant at a specified level, below that of the initial guess. The indicated residual of the w-vector decreases consistently with each iteration.

Chapter 3

Conclusions and Comments

3.1 Conclusions

As mentioned in Chapter 1, a fundamental problem when working with the Krylov spaces is the risk of getting, because of computational rounding errors, a flawed orthonormal basis. This is indeed the basic motivation for creating an approach using Householder reflections to store the basis. While we can expect this problem to be present in our simulation, we would not expect it to have a significant impact for a small number of basis vectors. We do, however, see significant divergence in the results early on in the iterations, and the author does not consider this to be consistent with the effects of an imperfect basis as they are orders of magnitude larger than the theory predicts.

Another point to make is that the GMRES algorithm profits from preconditioning of the problems [5] and within the scope of this work no preconditioning was used. This could disadvantage the numerically more sensitive Givens implementation when tested against the more stable Householder.

As stated, the dimensions of the problems are quite large (e.g. 568,208 in the case of the flanged shaft), and there is a wish to avoid explicit matrix multiplications by replacing them with an approximating function. This introduces an error factor that will propagate with each matrix-vector multiplication, which is each iteration. In the case of the first-order discretization, the library allows us to calculate the size and behavior of this error. When using second-order discretization, this is not possible and thus the exact behavior remains unknown. One could, however, try to extrapolate the results from those obtained when using the first order to the second, but this should be done

with caution and should be viewed as speculation until it can be properly confirmed in tests.

A very important observation is that, when using matrix-free methods, the w-vector-residual seems to be a poor indicator for the actual residual $||Ax_m - b||_2$. It is important because this is usually the only estimator for the residual that is used when testing the algorithms for convergence. This means that, unless the solution is explicitly tested, a matrix-free implementation could well return a solution stating convergence for a certain tolerance that the solution does not fulfill. It presents a double danger in that it may give false results and present them as true. Because of this, the author would recommend that every solution obtained when using matrix-free methods be checked separately to make sure that it indeed satisfies the tolerance required.

The matrix-free implementations are outperformed, with regard to consistence of the residual estimate, in every test by the Householder utilizing full matrix multiplication. The latter performs as theory predicts, and there were no observations made that would suggest that the w-vector is not an accurate estimate for the residual when using the full matrix multiplication.

Summing up the main points, based on the results obtained in the tests presented above;

- The Jacobian-free Gram-Schmidt implementation exhibits significant instability, the behavior observed is not consistent with the theoretical predictions outlined in Chapter 1.
- When using Jacobian-free methods, the w-vector does not provide a reliable estimate of the residual $||Ax - b||_2$.
- When using the first order-discretization, Jacobian-free implementations significantly decrease the residual $||Ax - b||_2$ only in the first iterations, after this using higher dimensions for the Krylov space does not seem to translate into finding a better solution for the system. This behavior was observed in all three experiments.
- The Householder implementation using full matrix-multiplications displays stable and consistent behavior in that it performs as predicted in each experiment considered.

3.2 Recommendations for further study

The results presented in this work are exclusively obtained with the use of the Tau Code library. If one wishes to study fundamental differences between the two different implementations of the GMRES method, the author would recommend running both methods on a series of known systems of equations where the condition number can easily be manipulated. In order to keep the issues of interests clear, the tests should be made with full matrix multiplication, without using approximations for numerical efficiency and without third-party code. The consequences of this approach would be difficulties of studying real-world applications of the code, which seldom can be reduced to a standard matrix. The avoidance of approximations could result in a significantly higher computational cost, so considerations of higher-dimensional problems could become very costly.

For the reasons mentioned above, further study of the use of GMRES within specialized libraries is also of interest. The use of approximating functions quickly becomes necessary as dimensions increase but their use significantly harms convergence theory, unless the error behavior of the approximating function is well known. In the results presented

in this work, the act of replacing the matrix multiplication with an approximating function significantly damages the accuracy of the solution. It is therefore of interest that the method for approximating the matrix-vector multiplication is tested for accuracy individually outside of the GMRES algorithm. The author thinks the quality of the approximation is central to usefulness of all matrix-free methods.

Summing up main areas for further study;

- Examine differences in performance of the Gram-Schmidt and Householder implementations of GMRES in a controlled environment, in particular their dependence on condition number.
- Search the Gram-Schmidt implementation for bugs.
- Individually test the matrix-vector-multiplication approximation against known matrices and vectors, examine if better methods are available. Examine if preconditioning increases the accuracy of the matrix-free methods.
- Test and document the performance of repeating Jacobian-free implementations of the GMRES algorithm. That is, see how the residual behaves if the function is terminated after a certain number of iterations and then is called again using the output of the previously called function as the new initial guess. This is then repeated until a tolerance is satisfied or a maximum number of restarts is reached.

[1] H. F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM J. Sci. Stat. Comput. Vol 9, No. 1, 1988, pp. 152-163.

[2] Y. Saad and M. H. Schultz, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput. Vol 7, No. 3, 1986, pp. 856-869.

[3] Å. Björck, *Solving linear least-squares problems by Gram-Schmidt orthogonalization*, BIT Num. Math. Vol. 7, No. 1, 1967, pp. 1-21.

[4] G. H. Golub and C. F. Van Loan, *Matrix Computations, 4th Edition*, The Johns Hopkins University Press, Baltimore, MD, 2013.

[5] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations.* SIAM, Philadelphia, PA, 1995.

[6] P. Birken, *Numerical methods for stiff problems*, Lecture Notes, 2015.

[7] The Tau Code collection of libraries used for this work includes TEMPO (Jacobian-free Newton Krylov methods) and FV-TAU (Finite volume methods). It is taken from https://www.mathematik.uni-kassel.de/svn_tau/Tau_Code (retrieved 29-04-2015).

Figures:

[F1] https://commons.wikimedia.org/wiki/File:NACA_0012_Demo.svg (retrieved 17-08-2015).

Appendix A

NACA0012 input parameters

Parameters which are varied throughout the tests are followed by a number (#) and explained below;

```
restart file =
grid name = NACA_LOW_MACH60.tri

Euler/NS = NAVIER_STOKES
Problem type = STEADY
Explicit/Implicit = IMPLICIT
Time integration = 0

Error controlled time adaptivity = 0
fsafety = 0.9
Minimal time step decrease = 0.5
Maximal time step increase = 2.0
Absolute tolerance in time step control = 1e-3
Relative tolerance in time step control = 1e-3

reconstruction_type = TVD
reconstruction = PIECEWISE_LINEAR (1)
limiter = BARTH_JESPERSEN

flux_function = AUSMDV
flux_function_jacobi = AUSMDV

ausmpup_ref_mach = 0.3
ausmpup_a_half =
ausmpup_kpfa_fix =

CFL-number = 4.0
max timesteps = 1
max time  = 999e10
```

```
CFL max number = 100.0
CFL update period = 10
CFL update factor = 1.2

Max Newton steps = 1
Newtontolerance = 1e-4
Eisenstat-Walker forcing terms = 0
Extrapolation = 0

krylov dim = 40 (2)
ilu update period = 30
epsilon = 1e-8
max restarts = 2

output period  = 2000
output prefix = NACA_LOW_MACH60

adaption period = 50000
adapt refine limit = 1.0e-1
adapt coarse limit = 1.0e-3
least triangle area = 0

Mach number = 0.85
angle = 1.25

preconditioner = 0
preconupdates = 0
physical renumbering = 1
matrix free solver = 1 (3)


Reynolds number = 1000
Prandtl number = 1.2
temperature = 300
isotherm or adiabat = ADIABAT
temperature on wall = 273
```

(1) is set to PIECEWISE_LINEAR when measuring second-order discretization, when measuring first-order discretization this is changed to PIECEWISE_CONSTANT

(2) is varied to measure dependence on the dimension of the Krylov space.

(3) is set to 1 when testing matrix-free methods, when examining the Householder using full matrix multiplication it is set to 0.

Appendix B

Wind turbine input parameters

Parameters which are varied throughout the tests are followed by a number (#) and explained below;

```
restart file = wind_96_boundarylayer_steadyEuler_t200.pval
grid name = wind_96_boundarylayer.tri

Euler/NS = NAVIER_STOKES
Problem type = STEADY
Explicit/Implicit = IMPLICIT
Time integration = 1

Error controlled time adaptivity = 0
Constant Delta t = 0.01
Timestep controller = 0
theta = 0.9
Minimal time step decrease = 0.5
Maximal time step increase = 2.0
Absolute tolerance in time step control = 1e-1
Relative tolerance in time step control = 1e-1

reconstruction_type = TVD
reconstruction = PIECEWISE_LINEAR (1)
limiter = BARTH_JESPERSEN

flux_function = AUSMDV
flux_function_jacobi = AUSMDV

ausmpup_ref_mach = 0.3
ausmpup_a_half =
ausmpup_kpfa_fix =
```

```
CFL-number = 100.0
max timesteps = 1
max time  = 202
CFL max number = 30.0
CFL update period = 10
CFL update factor = 1.2

Max Newton steps = 50
Newtontolerance = 1e-13
Eisenstat-Walker forcing terms = 0
Extrapolation = 0

krylov dim = 10 (2)
ilu update period = 30
epsilon = 1e-3
max restarts = 0

output period  = 200000
output prefix = Wind_96_SDIRK2

adaption period = 50000
adapt refine limit = 1.0e-1
adapt coarse limit = 1.0e-3
least triangle area = 0

Mach number = 0.12
angle = 40.0

preconditioner = 0
preconupdates = 0
physical renumbering = 1
matrix free solver = 1 (3)


Reynolds number = 1000
Prandtl number = 1.2
temperature = 300
isotherm or adiabat = ADIABAT
temperature on wall = 273
```

(1) is set to PIECEWISE_LINEAR when measuring second-order discretization, when measuring first-order discretization this is changed to PIECEWISE_CONSTANT

(2) is varied to measure dependence on the dimension of the Krylov space.

(3) is set to 1 when testing matrix-free methods, when examining the Householder using full matrix multiplication it is set to 0.

Appendix C

Flanged shaft input parameters

Parameters which are varied throughout the tests are followed by a number (#) and explained below;

```
restart file =
grid name = flanschmit2duesen.tri

Euler/NS = NAVIER_STOKES
Problem type = FLANSCHWELLE
Explicit/Implicit = IMPLICIT
Time integration = 0

Error controlled time adaptivity = 0
Constant Delta t = 0.01
Timestep controller = 3
theta = 0.9
Minimal time step decrease = 0.5
Maximal time step increase = 2.0
Absolute tolerance in time step control = 1e-3
Relative tolerance in time step control = 1e-3

reconstruction_type = TVD
reconstruction = PIECEWISE_LINEAR (1)
limiter = BARTH_JESPERSEN

flux_function = AUSMDV
flux_function_jacobi = VAN_LEER

ausmpup_ref_mach = 0.3
ausmpup_a_half =
ausmpup_kpfa_fix =
```

```
CFL-number = 2.0
max timesteps = 500
max time  = 10
CFL max number = 0.9
CFL update period = 10
CFL update factor = 1.2

Max Newton steps = 1
Newtontolerance = 1e-4
Eisenstat-Walker forcing terms = 1
Extrapolation = 0

krylov dim = 30 (2)
ilu update period = 30
epsilon = 1e-4
max restarts = 0

output period  = 200000000
output prefix = flanschwelle_2

adaption period = 5000000000
adapt refine limit = 1.0e-1
adapt coarse limit = 1.0e-3
least triangle area = 0

Mach number = 1.0
angle = 1.25

preconditioner = 0
preconupdates = 0
physical renumbering = 1
matrix free solver = 1 (3)



Reynolds number = 1000
Prandtl number = 1.2
temperature = 300
isotherm or adiabat = ADIABAT
temperature on wall = 273
```

(1) is set to PIECEWISE_LINEAR when measuring second-order discretization, when measuring first-order discretization this is changed to PIECEWISE_CONSTANT

(2) is varied to measure dependence on the dimension of the Krylov space.

(3) is set to 1 when testing matrix-free methods, when examining the Householder using full matrix multiplication it is set to 0.