MASTERARBEIT / MASTER’S THESIS

Titel der Masterarbeit / Title of the Master’s

"Experimental Studies of the Influence of Silent Data Corruption on FT-GMRES"

verfasst von / submitted by

Markus Gruber BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2017 / Vienna 2017

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	066 940
Studienrichtung lt. Studienblatt / / degree programme as it appears on the student record sheet:	Masterstudium Scientific Computing
Betreut von / Supervisor:	Assoz. Prof. Dr. Wilfried Gansterer, Privatdoz.

Contents

List of Tables

List of Figures

1 Abstract

Iterative methods give the possibility to solve the problem $Ax = b$ for the vector x in an efficient way for sparse matrices A. There are different computational steps which have to be done in these methods. Sometimes it can happen that any of those is affected by bit-flips and thus the algorithm yields wrong results. These errors in the main memory occur if a bit changes its value from 0 to 1 or from 1 to 0 with and without any external influence because of the fact that there is poor isolation between two transistors where each single one influences the others. The computer system can potentially detect and correct bit-flips with so called Error Correcting Codes (ECC memory) which is relatively costly in terms of additional power consumption and memory storage of about 12.5 % against non-ECC memory [1][2]. This fact is mainly related for correcting a single bit-flip for a 64 byte memory block (1 byte = 8 bits) with 8 additional bytes (a ratio of 8:1). It causes a performance lose of about 5 % - 15% [1][2] depending on the workload and utilization of the memory.

In some numerical methods there is maybe the possibility to decrease this constraint because only a few computational steps must be done with high reliability. These methods will lead to fault tolerant iterative linear solvers (see section 6.2). These are methods where only a few parts of the used algorithm must be protected against bit-flips to ensure high enough reliability. The Flexible GMRES (F-GMRES) [3] as well the Fault Tolerant GMRES (FT-GMRES) [4] solver are mainly based on the classical GMRES algorithm [5] where these methods use it as an inner/outer approach. In these methods the GMRES solver (see section 5.2.10) is applied for the outer as well for the inner solver whereas bit-flips are only allowed in the inner solver of the FT-GMRES (see section 6.2.4).

One key aspect of this work on the F-GMRES solver (see section 6.1.3) is when does the F-GMRES outperform the standard GMRES solver in terms of operations shown with some synthetic and real problems. It can be shown that using the flexible method also helps to decrease the number of needed operations massively for those used problems in this master work. The F-GMRES performs sometimes better but as well worse than using the standard GMRES solver.

The other part of this master work is to analyze the convergence behavior of the FT-GMRES in the presence of bit-flips during applying the matrix vector operation respective orthogonalization process ($h_{i,j}$). For each position of $h_{i,j}$ (line 5 - 10 of algorithm 6) different perturbations ($E_{perturbed}$) are induced in the inner solver of the FT-GMRES (line 4 of algorithm 14) with different kinds of errors (see section 8) to observe the behavior for single and as well multiple faults. In the case of a fault for positions of $h_{i,j}$ a new $\bar{h}_{i,j}$ is computed such that $h_{i,j}$ is multiplied with $E_{perturbed}$. Perturbations where $E_{perturbed}$ is lower than 1 have less impact on the FT-GMRES whereas perturbations with $E_{perturbed}$ greater than 1 or more are more problematic for all values of $h_{i,j}$.

Another purpose of this work is also to give some recommendations (see section 10.1) for the FT-GMRES based on what is known so far but with further researches to detect and correct more faults in the inner solver of the FT-GMRES because only perturbations larger than the norm ($||.||_2$) [6] of matrix A could be detected and corrected during applying the sparse vector operation [4]. One additional improvement (see section 10.2) which is done in this master work is that with the help of observing the residuum [7] ($||r||_2 = ||Ax - b||_2$) of the inner solver it is possible to protect the outer solver from (some) additional iterations which extends the current option(s) to detect and correct also perturbations smaller than the norm ($||.||_2$). Furthermore this approach can be also used to protect the whole inner solver from faulting which is a novel against previous works for the FT-GMRES so far [4]. One additional possibility would be to check the structure (see section 10.3) of the so called Hessenberg matrix (see figure 48) which is build during the orthogonalization process ($h_{i,j}$) in the inner solver of the FT-GMRES. This approach looks really promising because a lot more faults could be corrected especially in the case of solving symmetric problems, matrix A

is equal to its transpose ($A = A^T$), but if a fault was not detected then there was low overhead.

Flexible preconditioning (different iterations for the inner solver) helps to decrease the number of floating point operations but has less effect on the overhead because if a fault occurs then there must be done additional operations which results in the same costs as an not corrected error.

2 Summary

2.1 Motivation of this work and problem setting

Mainly the motivation of this work is driven by the expectation that through better manufacturing processes the distance between transistors will shrink and the negative influence between them will increase that's why the number of bit-flips will increase in future computer systems [8][9]. Bit-flips in the DRAM can be detected and corrected through the so called ECC (Error Correcting Code) memory this will lead to systems with higher reliability but applying ECC memory will also increase the energy requirements. Furthermore because of this fact the increase of bit-flips will also lead to higher energy requirements and some additional restrictions to build faster and much more complex computer clusters. That's why it is expected that correcting bit-flips in computer systems will be become more and more costly through rising the power consumption for this reason new techniques must be explored to go against this technology trend. This additional problem should be negated as much as possible through new explored techniques.

One way is perhaps to define sections where everything should be computed with high enough reliability and the other parts of the used solver should run with lower reliability this approach is also called selective reliability in section 4.1. Most of the computations should be done with lower reliability to decrease the energy requirements whereas the part of the algorithm with high reliability should ensure correctness of the computed result. In some numerical methods there is the possibility to apply selective reliability because of the fact that two solvers can be combined together as an inner/outer approach which leads to flexible methods (see section 6.1). In selective reliability the inner solver is mainly used to do the main work with low reliability and perhaps to speed up the computations through preconditioning whereas the outer solver has to check for correctness with high reliability. A further generalization of flexible methods will lead to fault tolerant iterative linear solvers through selective reliability (see section 6.2).

2.2 The problem which is considered in this master work

The main problem about this master work is focusing on the question of how to detect stagnating convergence of the FT-GMRES solver where faults lead to a bad behavior in the presence of bit-flips and to poor preconditioning of the inner solver (line 4 of algorithm 14). Especially in the case of the FT-GMRES where bit-flips in the inner solver can lead to a total flattening of the residual curve ($||r_2|| = ||Ax-b||_2$) in the inner solver and in general to a lower reduction of the residual for solving the problem $Ax = b$ in the presence of a fault. Stagnating convergence leads to the problem of a slow down of the used solver to solve the problem $Ax = b$ or in the worst case to a not satisfying solution x. Furthermore every undetected and not corrected fault in the inner solver will perhaps lead to more iterations for the outer solver of the FT-GMRES but it will also rise those energy requirements which must be avoided in every case. The main problem about poor preconditioning from the inner solver is how to detect stagnating convergence caused by bit-flips to ensure higher reliability of the outer solver but as well for the whole FT-GMRES solver. That's why one additional question of this master work is how to protect the whole FT-GMRES solver such that the outer solver is "nearly" unaffected from flipping a bit which means further fault detectors are searched.

2.3 The importance of this problem

Stagnating convergence also occurs in different contexts of iterative linear solvers especially in the FT-GMRES the main problem is just how to protect the inner solver from poor preconditioning such that the outer solver perhaps never converges to a reasonable solution, this question is really important for this master work. The main problem is just how to detect and correct stagnating convergence of the inner solver such that the FT-GMRES solver shows the same behavior for the number of outer iterations in a failure case as the F-GMRES solver without flipping a bit. That's why a further question is just how to achieve the purpose such that in every case the Fault Tolerant GMRES (FT-GMRES) has equal energy requirements or even decreases it against the Flexible GMRES solver (F-GMRES) to save energy in the best case without any additional disadvantage.

In the case of the totally unprotected FT-GMRES solver the overhead (additional iterations) because of a fault in the inner solver is approximately the same or even more like using ECC memory to correct faults. There are different possibilities to protect the outer solver from additional iterations but the inner solver can be always restarted even if the situation is not clear which leads perhaps to less iterations for the outer solver especially when no fault is occurred. A possibility to achieve this aim is observing the residual ($||r||_2 = ||Ax-b||_2$) which can be used to protect the outer solver. There must be done further researches but observing the residual is just one additional recommendation which leads to further problems like how to determine the right parameters for doing a restart of the inner solver and a better preconditioning for the outer solver of the FT-GMRES.

2.4 The current state of the FT-GMRES approach

Fault detection is really important for the FT-GMRES solver as well for any other fault tolerant method. The whole fault methodology for the FT-GMRES is described in section 8 which is the same like used from Sandia Labs in [4]. Faulting is mainly done during the orthogonalization process of $h_{i,j}$ (line 5 - 10 of algorithm 6) in the inner solver about the FT-GMRES (line 4 of algorithm 14). The orthogonalization process and the sparse matrix vector product need the most operations of the GMRES solver but it is also hard to protect these two sections from flipping a bit. In this model faulting is mainly done with $h_{i,j}$ such that a perturbed $\bar{h}_{i,j}$ is computed with $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times E_{perturbed}$ where $E_{perturbed}$ is the real disturbance during a fault. This faulting methodology will affect the whole inner solver except matrix A and the right hand side b and is restricted for floating point values, it is the first (important) computation. Sandia Labs already did some experiments about the FT-GMRES solver in [4] but in this approach it was only possible to detect and correct large perturbations greater than the norm ($||.||_2$) of matrix A during the matrix vector operation respective $h_{i,j}$, just for example the multiplied disturbance has a magnitude of the exponent of about 150 ($E_{perturbed} = 10^{150}$). Small disturbance factors have a magnitude of the exponent of about -300 ($E_{perturbed} = 10^{-300}$) and with the basis of 10 which are also large faults.

Faults lower than the used norm ($||.||_2$) of matrix A during the sparse matrix vector operation respective orthogonalization process ($h_{i,j}$) are undetected and not corrected in [4] and will lead to the problem of stagnating convergence at some specific positions in the inner solver of the FT-GMRES. This leads to the problem that for some specific positions during the orthogonalization process of computing $h_{i,j}$ the solution begins to totally stagnate if a fault occurs. This kind of error detection introduced in [4] can be only used to check faults during the sparse matrix product and orthogonalization process ($h_{i,j}$) but it doesn't protect the inner solver at all especially from stagnating convergence and poor preconditioning. There are already some researches done for detecting stagnating convergence like in [7] which are applied in other contexts but with the aim to detect stagnating convergence in the standard GMRES approach but not in the FT-GMRES solver.

2.5 The scientific gap

The scientific gap is mainly that only some specific kinds of faults can be detected in the FT-GMRES solver where fault detection is restricted on the sparse vector operation ($y = Ax$) at the moment. Those kinds of errors which can be only detected are disturbances greater than the norm

($||.||_2$) of matrix A where $E_{perturbed}$ is greater than 1 in [4]. The sparse matrix vector computation respective orthogonalization process ($h_{i,j}$) needs a lot of operations which gives the possibility to reduce the power consumption a lot but still there must be more possibilities found to protect more parts of the inner solver of the FT-GMRES from flipping a bit. Perturbations lower than the norm ($||.||_2$) of matrix A where $E_{perturbed}$ is lower than 1 cannot be detected and corrected during the sparse matrix vector operation in the inner solver of the FT-GMRES at the moment in [4] which is also a focus of this work in-depended of the used fault detector. There are no possibilities for the FT-GMRES solver to protect the whole (outer) solver from additional iterations at the moment. Another scientific gap is how does the overhead depend on the position ($h_{i,j}$) where the fault occurs for single and multiple faults and does preconditioning help to decrease the overhead.

2.6 The main focus of this master work

The main problem which is considered in this master work is how to protect the inner solver of the FT-GMRES (line 4 of algorithm 14) from flipping some bits. One fault detector based on the norm ($||.||_2$) of the used matrix A gives the possibility to detect large perturbations during the sparse matrix vector operation which is in general not applicable because computing $||.||_2$ is nearly the same as solving $Ax = b$ but it is also possible to use other norms. This fault detector is mainly related to the fact because in the GMRES a matrix vector product is applied ($y = Ax$) so by using the norm ($||.||_2$) it follows that $||y||_2 \leq ||A||_2||x||_2$ by knowing that $||x||_2 = 1$ the vector y can be at least $||y||_2 \leq ||A||_2$ in a failure free case (see section 8). There are no fault detectors for small perturbation factors ($E_{perturbed} < 1$) because in the case of the (FT-)GMRES solver it means that the end residuum ($||r||_2 = ||Ax - b||_2$) is already known which can be achieved with it. In [7] an approach is considered for how to detect stagnating convergence but only for the standard GMRES solver which can be adapted for the inner solver of the FT-GMRES and can be used to detect small and as well large perturbations. This work also tries to answer how does the position of a fault during the orthogonalization process ($h_{i,j}$) affect the overhead. In this work there are also further possibilities searched for fault detection and correction in the inner solver of the FT-GMRES.

2.7 The results of this master work

The main result is that stagnating convergence can be detected in the inner solver of the FT-GMRES, the problem is just to decide if this is the case or not. There are mainly two problems solved in this master work while the relative change between two residuals in the inner solver was observed. For the first case (2D Poisson matrix) it was possible to detect stagnating convergence whereas in the second problem (adder_dcop_63 matrix [10]) it was not the case. The first matrix has a different behavior as the second problem because for this problem the relative change between two residuals is approximately the same which means the residual reduction in the inner solver is nearly the same from one iteration to the next iteration. In the case of the second problem (adder_dcop_63 matrix) the relative change is not always the same between two iterations which makes fault detection really hard because of the convergence behavior. Furthermore it means that the convergence behavior must be known before solving the problem with fault detection.

One far better approach is to check the structure of the Hessenberg matrix (see figure 48) because if a fault occurs in the inner solver of the FT-GMRES a deviation from the real structure can be observed which looks really promising but there must be two different cases considered. Not for each fault a change of the structure could be observed which means not every bit-flip can be detected for computing $h_{i,j}$. Values of $E_{perturbed} = 10^{150}$ and $E_{perturbed} = 10^{-300}$ are mostly used to inject faults. Small perturbation coefficients ($E_{perturbed} < 1$) will mainly lead to stagnating convergence on some specific positions like during computing the value of $h_{1,2}$ and $h_{2,3}$ whereas large perturbation coefficients ($E_{perturbed} > 1$) are more problematic during the whole computation.

3 Synopsis

Related Work: In section 4 a summary about different scientific papers is given mainly about fault tolerant iterative linear solvers and those different techniques which are used in it. Furthermore in section 4.1 topics of concepts about fault tolerance are more explicit explained.

Iterative methods: In section 5.1 the main classification of iterative methods is given whereas in general iterative methods can be classified in two kinds of solvers (stationary and non-stationary).

Krylov subspace methods: In section 5.2 Krylov methods are discussed in general which are considered as iterative (non-stationary) methods. These methods are mainly used if matrix A is sparse to solve $Ax = b$, from 5.2.1 to section 5.2.8 Krylov methods are discussed in detail.

The Conjugate Gradient (CG) and Steepest Descent method (SD): In section 5.2.9 the basics about the Steepest Descent (SD) (see section 5.2.9.1) and the Conjugate Gradient (CG) (see section 5.2.9.2) are introduced, only the CG solver is considered as a Krylov subspace solver.

The GMRES method: In section 5.2.10 the GMRES solver is introduced which can be used to solve an equation system $Ax = b$ for any type of matrix which is an advantage against the CG.

Problem description of exa scale computing: In section 5.4 most of the problems about HPC (High Performance Computing) are discussed and which limitations come from it like the power wall problem problem and fault tolerance which is a problem for huge compute clusters.

Bit-flips in practice: In section 5.7 a summary of different scientific papers about bit-flips in the DRAM is given which shows the effect of different influences on the number of bit-flips.

Flexible solvers (F-solvers): In section 6.1 the idea behind flexible solvers is shown. In flexible methods there are always two iterative solvers one outer and one inner solver. The outer solver has to check for convergence with high reliability whereas the inner solver should do the main work.

Fault Tolerant Iterative Linear solvers (FT-solvers): In section 6.2 the basics about Fault Tolerant Iterative Linear solvers are explained. This kind of solver is mainly based on the flexible solver but the main contrast is that in the inner solver bits are now allowed to flip which causes some overhead. This overhead (additional iterations or floating point operations) can be huge.

Dealing with rank deficiency in F-GMRES/FT-GMRES: In section 6.3 there is a short summary for the F-GMRES and FT-GMRES of how to handle poor preconditioning of the inner solver which may affect the outer solver negatively especially the built Hessenberg matrix (see section 5.2.10) to solve the problem $Ax = b$, unlucky preconditioning can abort the whole computation.

Relaxation strategies for nested Krylov methods: In section 6.4 nested *inexact* Krylov methods are discussed. This topic is important for the initialization of the inner solver for flexible methods but as well important for the preconditioning of nested Krylov methods like F-GMRES.

Experiments related to GMRES and Flexible GMRES (F-GMRES)[1]: In section 7 there are different experiments done but there are mainly two problems solved in section 7.1 with the Flexible GMRES and GMRES solver. These problems are solved where the computing time is compared against the GMRES solver. There is also the relation between the condition number (see formula 72) and the number of outer iterations of the F-GMRES shown for different initializations of the inner solver. In section 7.2 there is a comparison between F-GMRES and GMRES on different diagonal matrices for observing the convergence behavior.

Injecting errors in Fault Tolerant GMRES (FT-GMRES)[1]: Section 8 describes how faults are injected in FT-GMRES and why this specific method is used. It mainly describes the fault methodology how faults are caused in the inner solver of the FT-GMRES during the sparse matrix vector operation and orthogonalization process ($h_{i,j}$).

[1] ..new topics which are not explored

Experiments related to Fault Tolerant GMRES (FT-GMRES)[2]: Section 9 is mainly about the Fault Tolerant GMRES (FT-GMRES) where several experiments are done. This section has different subsections which are explained afterwards. There are different experiments done for single and multiple faults and other parts of the used GMRES algorithm.

Faulting on the first and last Modified Gram Schmidt iteration[2]: There are some experiments done in section 9.1.1 and 9.1.2 to determine the behavior in the case of a single fault. In this case the inner solver of the FT-GMRES is initialized with random values for vector $w_j(w_0)$. The Gram Schmidt method is mainly used in the GMRES solver (see algorithm 3 and 4).

Testing of all positions of the orthogonalization ($h_{i,j}$) with single faults[1]: Section 9.2 shows the influence of single faults during the sparse matrix vector operation and orthogonalization process ($h_{i,j}$) in the inner solver of the FT-GMRES. There is mainly a heat map used which shows the number of outer iterations on all positions for a specific kind of perturbation and for a single fault. These heat maps visualize the number of outer iterations if in the inner solver is a fault.

Testing of all positions of the orthogonalization ($h_{i,j}$) with multiple faults[1]: Section 9.3 shows the influence of multiple faults during the sparse matrix vector operation and orthogonalization process ($h_{i,j}$) in the inner solver of the FT-GMRES. There is mainly a heat map used which visualizes the number of outer iterations if on and after a position ($h_{i,j}$) multiple faults occur.

Comparison between the 2D Poisson and adder_dcop_63 matrix[1]: Section 9.4 does a comparison and short analysis between two different types of matrices namely the 2D Poisson and adder_dcop_63 [10] matrix. The adder_dcop_63 matrix has a different behavior as the 2D Poisson matrix if a fault occurs with a small perturbation coefficient ($E_{perturbed} = 10^{-300}$). This matrix has the ability to tolerate more faults because small perturbation coefficients have less impact on the number of outer iterations for the FT-GMRES whereas a better behavior can be also observed for large perturbations. Faulting during solving the 2D Poisson matrix shows a similar behavior like in [4]. This is an indicator that both implementations (in this master work and in [4]) have the same initialization (w_0) for the inner solver (line 4 of algorithm 14) of the FT-GMRES such that the inner solver should be initialized with zero values for vector $w_j(w_0)$ to ensure fast convergence.

In [4] there was solved the mult_dcop_03 [10] matrix instead of the adder_dcop_63 problem but if a relative residuum ($||r||_2 = ||Ax-b||_2/||b||_2$) of 10^{-8} should be achieved for the first problem then the number of outer iterations is exactly the same (28), also for solving the 2D Poisson problem.

Faulting while solving the 2D Poisson and adder_dcop_63 problem with and without ILU-preconditioning in the inner solver of the FT-GMRES[1]: Section 9.5 shows also the influence of preconditioning with the ILU-factorization [6] in the inner solver of the FT-GMRES during solving the 2D Poisson and adder_dcop_63 [10] matrix in the case of a single fault (from section 9.4.6 to 9.5.10). Depending on the initialization of the inner solver and error of factorization the overhead can be decreased for small perturbations whereas large perturbations are still problematic, faulting in the inner solver will always lead to a worse residuum.

Faulting while solving the 2D Poisson and adder_dcop_63 problem with flexible preconditioning by the inner solver of the FT-GMRES[1]: In section 9.6 are some experiments done with applying flexible preconditioning in the presence of a single fault while solving the 2D Poisson and adder_dcop_63 matrix. Flexible preconditioning mainly helps to decrease the needed number of operations to converge but in the case of a fault in the inner solver the overhead (additional iterations) in floating point operations can be very high. The overhead is estimated hard because the additional work is counted in flops (floating point operations) not in additional iterations.

Results of some experiments (worst overhead) with the corresponding matrix properties[1]:

[2] ..there are already existing works

In section 9.7 there is a short summary about some experiments and properties of all used matrices where the worst overhead is shown for experiments which are discussed and not discussed in detail in this master work. This section is done to get a better overview about the overhead in the presence of a single fault for different matrices [10] and the corresponding properties. The overhead for a single fault can be specified with 0 to 300 % where about 15 - 20 % are mostly seen.

Faulting with single faults while solving further problems (symmetric, un-symmetric)[1]: In section 9.8 there are mainly experiments done for solving symmetric and un-symmetric problems and determining the overhead in the presence of a single fault for different kinds of faults and matrices. This section is done for section 10.3 where the overhead is shown for the same matrices but with fault detection. All used matrices are taken from [10] except the 2D Poisson matrix.

Faulting with single faults while solving further preconditioned problems with ILU-factorization (symmetric, un-symmetric)[1]: In section 9.9 there are further experiments done with four different matrices where ILU-preconditioning is applied to show the effect of preconditioning with other matrices but also to compare these matrices without ILU-preconditioning.

Faulting with single faults while solving further problems with flexible preconditioning by the inner solver of the FT-GMRES (symmetric, un-symmetric)[1]: In section 9.10 there are additional experiments done with seven matrices for applying flexible preconditioning in the presence of a single fault. Flexible preconditioning mainly helps to decrease the needed number of operations but has no influence on the observed overhead.

Faulting with multiple faults while solving further problems (un-symmetric)[1]: In section 9.11 there are experiments done for solving un-symmetric and symmetric problems and determining the overhead in the presence of multiple fault for different kinds of faults and matrices. There are mainly two matrices solved in this section to show the effect multiple faults, if all values of a row or column faults during computing the Hessenberg matrix in the inner solver of the FT-GMRES.

Different disturbed Hessenberg matrices[1]: In section 9.12 there are different Hessenberg matrices shown after applying the Givens rotation (see section 5) and for a single fault. This section shows how the structure changes for the Hessenberg matrix and for different kinds of problems in the presence of a single fault which can be used for fault detection (see section 10.3).

Improvements and recommendations for Fault Tolerant GMRES (FT-GMRES)[1]: In section 10 there is mainly an overview (in section 10.1) to summarize what is known so far for error and fault detection but with also some further improvements for the FT-GMRES which are done in this master work. There is also a comparison between different approaches like using the norm ($||.||_2$) of matrix A or detecting stagnating convergence, some are applicable some aren't.

Fault detection through the relative change between two residuals[1]: In section 10.2 an approach is considered to determine the parameters for detecting stagnating convergence in the inner solver of the FT-GMRES. There are also some experiments done with the 2D Poisson and adder_dcop_63 matrix to show which faults can be detected or not during the sparse matrix vector operation and orthogonalization process ($h_{i,j}$) in the inner solver of the FT-GMRES. The overhead can be decreased depending on the threshold value of the relative change and used problem.

Fault detection through checking the structure of the Hessenberg matrix[1]: In section 10.3 an additional approach is considered to detect faults through the structure of the so called Hessenberg matrix (see figure 48). This kind of fault detection is mainly related to the upper part of the Hessenberg matrix which stores all values of $h_{i,j}$. There are some experiments done with different matrices but also with the 2D Poisson and adder_dcop_63 matrix. If a deviation from the standard case is detected then restarting the inner solver is applied. Overall this approach looks really promising to detect faults in the inner solver of the FT-GMRES but it is not possible to detect every fault. Overall this kind of error detection helps to decrease the overhead in most cases.

4 Related Work

4.1 Topics related to fault tolerant iterative linear solvers (overview)

This section should give a short overview which topics are also related to fault tolerant iterative linear solvers (see section 6.2) but also which are not discussed in detail in this master work.

Self-stabilization [11]: describes the meaning that a system has the property regardless of the initial state to converge to a legitimate state after a finite number of steps. If this final state should be legitimate or not depends on some global rules, if those are checked and all according rules are satisfied then the outcome is that the whole system is in a state which is accepted. Self-stabilization occurs in different ways but there must be a mechanism which brings the whole system from a not accepted state in an accepted state. E. Dijkstra describes self-stabilization in computer system in the year of 1974.

Selective reliability [12]: Selective reliability is the property of a system or application to do something with a certain reliability but most of the computations are done with a lower reliability than the final computed solution. With selective reliability it is still possible to achieve the same result like where the reliability is not changed but still high enough to ensure correctness of the computed result. This can be seen as the application of the principles of self-stabilization. There is still a global checker which has the duty to ensure correctness of the computed result such that everything is computed with enough high reliability. Checking for correctness must be also done in high reliability if not the computed result might be wrong because of a lower reliability.

Algorithm Based Fault Tolerance (ABFT) [13]: ABFT is a summary of different techniques but in contrast to selective reliability everything is done with the same reliability such that faults during the computation may occur. ABFT is mainly related to checksums but with the appliance of these different techniques the correct result must be still computed. For example this can be also a parity bit to know if all bits are even or odd. If this checksum is not found in the computed result then there must be something wrong during the computation. ABFT is not only related to fault tolerant iterative linear solvers, it is a summary of different techniques to improve and ensure a certain reliability of a computer system.

Partial Recomputing [14]: can be also considered as a part of ABFT but it is mostly used in parallel environments or compute clusters and differ in much aspects from the general idea of ABFT. The aim of scientific applications is to scale up a problem such that a lot CPU cores must be used because a scientific problem is often parallelized to decrease the computing time. Because of the fact that some data are multiple times available on different nodes and CPUs it is possible to recompute some faulty parts or in the simplest way to do a data exchange between some notes such that data which are left on a node are restored from other nodes of the compute cluster.

4.2 Selective Reliability

4.2.1 Flexible Conjugate Gradient (F-CG)

In [15] the Flexible CG (F-CG) solver (see section 6.1.2) is introduced similar to the Flexible GMRES (F-GMRES, see section 6.1.3) which is also discussed in section 4.2.2. The F-CG is build with two CG solvers as an inner/outer - approach in [16]. In this method there are always two solvers one inner and outer solver. The duty of the outer solver (line 3 - 10 of algorithm 10) is to check for convergence and correctness of the computed solution x to solve the problem $Ax = b$ whereas the inner solver (line 4 of algorithm 10) should do the main work. Like the CG solver (see section 5.2.9.2) the F-CG can be only used to solve problems where the matrix A is positive definite (positive definiteness means that the product of $x^T Ax$ is always greater than zero for any vector x).

The focus of [15] is mainly on the behavior of the coupling between the inner and outer solver by also considering different problems. The author of [15] figures out that the F-CG gives some benefits in terms of stability for solving the problem $Ax = b$ if and only if the accuracy of the computed solution from the inner solver is "high" enough but there will be no advantage if both solvers have "medium" accuracy. It is not really specified what is considered as "high" or "low" accuracy especially in terms of operations and when to terminate the inner solver if the solution of it cannot be improved anymore and to decrease the unnecessary operations caused by the inner solver. The number of outer iterations depends mainly on the number of iterations from the inner solver and on the used problem in [15]. There are also some problems solved in this study where the standard CG solves these problems faster than the F-CG or both are equivalent in the number of operations which depends mainly on the properties of the used matrix A.

In [12] two different solvers are implemented of the Conjugate Gradient (CG), one with the ability of self-stabilization (see section 4.1) which is similar to the standard CG solver and one with the same inner/outer approach mentioned before (see section 6.1.2). The solver with the ability of self-stabilization is mainly two times the CG solver but there are correction steps done in the second solver which are done with high reliability in comparison to the first one. Both implementations have different pros and cons. One main contrast between both solvers is the ability to cope with different kinds of faults. The CG solver with the property of self - stabilization and the F-CG solver with the inner/outer approach have similar behaviors if a fault occurs in the mantissa or sign of a floating point value from the solution vector x, in the inner solver of the F-CG solver. The solver with the ability of self-stabilization is preferable if a fault occurs in the exponent of the solution vector x which causes larger errors. The author figures out that this property is mainly related to the speed of solving $Ax = b$ for the standard CG solver. In [12] only the fault rate is given for flipping a bit during the sparse matrix vector product but not how high the caused perturbation is is left. All these comparisons are done with different problems but also for different fault rates during the sparse matrix vector product which also affects the solution vector x.

In [17] a study is considered which is mainly related to the standard CG solver and about the computational overhead in iterative methods which is important for designing fault tolerant iterative linear solvers. It mainly shows the overhead if bit-flips happen in different data and sections of the used algorithm for different probabilities, just for example like in the matrix A and the solution vector x. The focus of [17] is also on the overhead which can be caused by bit-flips where the caused overhead depends more on the section which faults of the used iterative method.

4.2.2 Flexible GMRES (F-GMRES)

In [3] Yousef Saad introduces the principle that the GMRES solver (see section 5.2.10) can also be applied as an inner/outer - approach which shows some advantages in the case of the achieved accuracy for the solution vector x. This inner/outer - approach (see section 6.1.3) uses two GMRES solvers with nesting whereas the inner solver does the main work but the outer solver has to check the accuracy of the achieved solution x. There are two different test problems in [3] but both illustrate that this inner/outer - approach also called Flexible GMRES (F-GMRES) has some advantages against the classical GMRES solver especially for the achieved accuracy of the solution but by also decreasing the number of needed operations to converge. This comparison is mainly done with the classical GMRES solver as the outer solver but also in combination with the Conjugate Gradient solver (CG) and with ILU-preconditioning [6] for the inner solver. Preconditioning like using ILU-factorization (matrix A is decomposed in $A \approx LU$, see section 5.2.7) is mainly used for speeding up the computation of iterative methods. The first problem of this comparison shows a faster approximation to the desired solution but with the combination of GMRES and CG as the inner solver rather than using two times the standard GMRES solver as an inner/outer - approach. In the case of the second example using the CG and GMRES solver has some advantages against all other methods especially against applying the standard GMRES solver in terms of needed operations.

Yousef Saad mentioned in the conclusion of [3] that using preconditioning can be "unpredictable". It is not clear why the combination of using GMRES with the CG solver is the best combination of all these solvers and solved problems maybe because of the fact that both matrices are positive definite (positive definiteness means that the product of $x^T Ax$ is always greater than zero for any chosen vector x). Both combinations of GMRES / GMRES and GMRES / CG outperform all others, especially the F-GMRES outperforms the GMRES solver in terms of needed operations.

Sandia Labs introduced the Fault Tolerant GMRES (FT-GMRES) in [4][12] with the help of the Flexible GMRES (F-GMRES). In [4] James Elliot et al. tried to find out the impact of single bit-flips during preconditioning of the inner solver (line 4 of algorithm 14) in the FT-GMRES and with the question of how does the computation slows down in the presence of a single fault. The maximum allowable perturbation which comes from the matrix A depends on the used norm of $||.||_2$ and can be used as an error detector during the orthogonalization process ($h_{i,j}$) respective matrix vector product which indicates if a bit-flip respective fault happens in the inner solver. In all these experiments faulting was only done once on some specific positions during the orthogonalization process of $h_{i,j}$ (see section 8) for two different matrices but only for two positions and cases.

Something which is related to fault tolerance about the (FT-)GMRES can be found in [18] where different techniques are compared like TMR but by also focusing on this inner/outer - approach. In Triple Modular Redundancy (TMR) the same computation is done three times where the most obvious solution is used, this method has a large overhead of about 200 %. The Fault Tolerant GMRES (FT-GMRES) is mainly based on the Flexible GMRES (F-GMRES), if a problem occurs a restart can be applied but this will lead to some problems for the computation speed.

This effect can be negated like in [18]. It introduces a new technique based on multiple check-pointing but by also using this inner/outer approach introduced before. Multiple check-pointing means that different states of the vector w_j (line 4 of algorithm 14) are saved for different iteration indexes of the inner solver whereas the best promising state of w_j is used if a fault occurs. If the accuracy of the new vector w_j is satisfying then the outer solver is allowed to use it computed from the inner solver. This new technique gives some benefits against partial recomputing (see section 4.1) because of a lower overhead also shown in [18]. Partial recomputing allows to recompute faulty parts of the solution vector x those can be detected with the help of the residuum ($||r||_2 = ||Ax - b||_2$). Multiple check-pointing gives some advantages against partial recomputing if the probability of faulting is really high whereas partial recomputing should only be applied for low probabilities of flipping some bits. In this approach [18] there are also two different strategies applied just one for the outer solver and one for the inner solver. The outer solver checks the residuum ($||r||_2 = ||Ax - b||_2$) whereas in the inner solver it is possible to do some rollbacks if a fault occurs. A rollback means that all old data of the inner solver like the vector w_j are restored. Because of the need to apply the Givens rotation (see section 5.2.10.3) for both solvers it is not clear how to store all multiple vectors and how to do a rollback efficiently because a rollback will also influence the built Hessenberg matrix (see formula 47 and formula 48) of the inner solver which is maybe not really efficient for the storage place and energy efficiency.

Preconditioning is mainly used for speeding up the computation time in iterative methods, these methods usually try to decompose the matrix A. In [19] preconditioning for the FT-GMRES is done where opaque preconditioners are used. Opaque means using an existing preconditioner because there is no intent to change million lines of code which preconditioners usually have for iterative methods. Those preconditioners (additive Schwarz domain decomposition [20] and ILU [6]) are used in a parallel environment, this study tries to answer the question how much additional iterations does the used preconditioner need if a fault occurs and how does an error from a node influence the others. In this study bit-flips are not considered as errors they are seen as a bad output from the preconditioner because of the used matrix A. In general bit-flips are now an extension of numerical errors but single bit-flips are not considered in this view because single bit-flips can cause the same error like multiple bit-flips.

This point of view makes a lot of considerations quite easier because it doesn't matter how often bit-flips happen but only which error is caused at the end because of a perturbation from flipping some bits. Something which is not clear in this paper is how faulting is applied for these different preconditioners which makes reconstructing [19] nearly impossible. The main problem is just that preconditioners usally have million lines of code. There is also a similar study in [21] which shows at which bit-position of the exponent with the according floating point representation occurs the greatest overhead because of a single bit-flip this study is also related to [19] where it comes out that the highest overhead occurs between the bit positions of 32 and 64 (bit positions which cause the greatest change of the floating point value).

4.3 ABFT (Algorithm Based Fault Tolerance)

Another topic which is related to fault tolerant iterative linear solvers is ABFT. It stands for Algorithm Based Fault Tolerance (ABFT). The main idea is that with some additional computations mainly with checksums and with some extensions like check-pointing it is possible to detect and correct some faulty data. Check-pointing should only be applied for large matrices because the overhead of additional computations will always be too high for small matrices for paying off its cost. In mostly all approaches of ABFT there is always an underlying assumption mainly about the according fault rate. ABFT techniques are also based on different parameters, just to be effective the right part parameters have to be found.

4.3.1 ABFT for the GMRES solver

The QR-factorization [6] is used for much problems. The QR-process tries to decompose the matrix A iteratively in $A = QR$ where Q is an orthogonal matrix such that $QQ^T = I$ with I as the identity matrix. This kind of process is mainly used for overdetermined and regular equation systems but this method is also used in the GMRES algorithm (see section 6). The matrix R also denotes an upper triangular matrix. In [22] a technique is introduced to detect and correct faults in the so called QR-process. This technique for error detection and correction is mainly based on checksums where additional rows and columns are added to $A = QR$ such that $(A\ Ae\ Aw) = Q(R\ c\ v)$ to make fault detection possible where c and v are vectors and matrix Ae and Aw are additional columns for matrix A. This kind of approach makes fault detection possible but also fault correction based on the QR update. This paper doesn't show how to handle each different fault but the author figures out that this kind of process mentioned in [22] handles any fault during the QR-factorization.

There is also a performance comparison against the MAGMA library [23] which is used in HPC for different problems like the matrix vector operation ($y = Ax$). This kind of approach in [22] shows really less overhead in comparison to the standard QR-factorization without check-pointing. It is also possible to extend this approach mentioned in [22] to the standard LU-factorization [6] which is used for arbitrary dense matrices. ABFT is here applied in the way that fault tolerance comes from some checksums and trough an additional correction step during the QR-factorization process if something is computed wrong or bit-flips happen.

4.3.2 ABFT for the Conjugate Gradient (CG) solver

Something which is related to ABFT for the CG solver (see section 6.1.2) can be found in [24]. In this paper there is a hybrid approach applied but without using the principles of the Flexible Conjugate Gradient (F-CG). In this study the used solver is only based on the standard CG solver but with applying partial recomputing and as well adopting check-pointing. Partial recomputing can be used to recompute faulty parts of the solution vector x from the previous computed residuum ($r = Ax - b$) whereas check-pointing secures the last state of x. This approach shows that partial recomputing can be also combined with other techniques to decrease the overhead caused by some bit-flips. There are still some unreliable sections in this algorithm which are not explicit mentioned. Fault detection is still applied with high reliability but most of the computations are done with lower reliability. In this study check-pointing and fault detection is mainly applied with an additional created fault detector based on the current residuum (r) for the current solution vector x and with the according right hand side b. Additionally the decision to apply recovering from a previous checkpoint or to do partial recomputing is based on two computed residuals, one from the last checkpoint and also on the current faulty solution vector x which belongs to the current residuum. If the residuum for recovering is lower than the residuum of the current solution vector x which should be used for partial recomputing then recovering is applied but if not partial recomputing is applied.

There are also other methods for ABFT which are also called Online-ABFT. Online-ABFT tries to find optimal parameters for recover and check-pointing with low overhead. In [25] most of the recover and check-pointing techniques are based on some error assumptions. Through finding these parameters it is possible to get nearly optimal checkpoints with low overhead which is done in the case of the Conjugate Gradient (CG) solver. The main problem here is because of the according probability distribution about the fault rate it is not possible to find these optimal parameters analytically. In this study there are also some fault detectors used to decide if to do check-pointing or a recover from a previous state. Some of the math comes from [26] which focus is mainly on check-pointing. This paper is a generalization of check-pointing and recovering but without using or focusing on any iterative method.

4.3.3 Fault tolerance through equilibrated matrices

In [27] it comes out that values of matrix A which have the same order of magnitude through scaling the entries the used solvers (in this case the Jacobi method) are less affected through bit-flips. The effect of bit-flips is decreased through an equilibrium in matrix A and vector b. This technique has mainly the aim to decrease the norm ($||.||_\infty$, absolute largest element of matrix A) to approximately 1 and as well the condition number (see formula 72) of matrix A which can be also done with other techniques. It also helps to approximate the norm $||.||_2$ of matrix A better. This method scales all values of matrix A (and b) with the aim that all of them have a specific magnitude such that the absolute greatest value ($||.||_\infty$) is at most 1, it gives also an upper bound for the caused absolute error which also decreases the influence of faults in the presence of bit-flips. This method and the related avoidance technique is linked to the fact that if two vectors u and v are normalized ($||u||_2 = 1$ and $||v||_2 = 1$) then the absolute greatest error can be at most 1 ($||uv - 1||_2 \leq 1$) which is also related to the fact that small magnitudes will give less errors whereas great magnitudes large errors. It also helps to speed up the computation time like in the case of the 2D Poisson matrix of about 50 % or even more less computing time for the Jacobi method (see section 5.2.2) in [28]. For these experiments in [28] there is also low overhead seen in the presence of bit-flips. This study also explores the relation between the position of a single bit-flip and the observed overhead because only some specific positions are problematic namely bit-positions greater than 52 (from 0 to 64) which cause large changes.

4.4 Partial recomputing

In general partial recomputing is used in parallel environments because data are multiple times available on different nodes of the used server, so if one of them crashes some data of the solution vector x can be reconstructed like in the case of the CG with the help of the other nodes. In [14] global and local recomputing techniques are introduced which make correcting faulty parts of the solution vector x possible if those faulty parts are detected. The main contrast is that local recomputing is used for single bit-flips in contrast global computing is used for multiple faults in the solution vector x. In [14] the basics about local and global recomputing are shown but with also some case studies. Partial recomputing can be the way to cope with bit-flips but with a low fault probability in contrast to other techniques if bit-flips happen with high probability. There are also different methods used in [14] for recomputing the faulty parts of the solution vector x, mainly linear and quadratic interpolation both have advantages and disadvantages. Linear interpolation uses two points to compute a new value between them whereas quadratic interpolation tries to minimize the total error. Quadratic interpolation is preferable over the linear interpolation method in most cases but it needs more operations to apply. Furthermore global recomputing is also preferable over local computing if the number of faults rises on the compute nodes.

Some more general topics about partial recomputing can be found in [29]. The focus of [29] is mainly to find out some contrasts between several recover and check pointing strategies. The main outcome is that there is really less influence on the compute overhead if there is a checkpoint at every iteration step which ensures the solution vector x in the main memory but without using the hard disk. Partial recomputing makes some sense in parallel environments because data are available multiple times. As a result of this, in [30] it is introduced parallel recomputing without high overhead. Data which are multiple times available distributed over different nodes and processors can be easily used for recomputing. This technique is simple based on a set for each node and as well on the solution vector x to know which values are left for recomputing. This can cause some additional overhead in parallel environments but should maintain most of the computation speed of the used solver by simple interchanging the unknown parts for the vector x and matrix A.

Recomputing can only be applied if bit-flips are detected, maybe through checksums for locating the faulty parts with hopefully increasing the computing overhead less which is done in [31]. The aim of it is mainly to detect faults through checksums with an additional vector (c) but it also uses partial recomputing. The math behind this approach is really simple but there is no comparison to other techniques, the overhead which is caused by this technique should be really low like mentioned in this paper. This method is mainly restricted to the matrix vector operation ($y = Ax$). So it uses two techniques partial recomputing and error localization with checksums. The main problem of this approach is that it needs two matrix vector operations, one from the left and one from the right of matrix A. The additional left one is needed for locating the faulty parts and if the difference between $c^T y$ and $(c^T A)x$ is unequal to zero a fault is occurred during computing $y = Ax$.

4.5 Self-stabilization

Self-stabilization is another topic of this work. In [11] Edsger W. Dijkstra introduced the first description of what does it mean if a system regardless which kind of system converges to a stable state. This stable state can be just seen as the correct output or solution vector x from any iterative method for solving an equation system ($Ax = b$). He wrote this short paper especially in the context of distributed communication systems but it has still an important impact to other researches. In this view there is always a global checker which tests if the current state of the system is correct or not based on some specific rules. Like in [32] self-stabilizing solvers are an alternative to the selective reliability solvers introduced in [4]. It is possible to do a correction step in reliable mode for self-stabilizing solvers if the residuum stops to minimize. This paper is mainly forced on the Conjugate Gradient (CG, see section 5.2.9.2) and Steepest Descent (SD, see section 5.2.9.1) method where finding an alternative search direction for lowering the residuum is important.

In [32] it is also shown that methods can have different properties for fault tolerant iterative linear solvers where some are more invulnerable against bit-flips. Like in [32] the outcome is that self-stabilizing iterative linear solvers have the ability to cope with faults which cause high changes in the exponent in contrast to the inner/outer - approach which is used for the Flexible CG.

4.6 Questions

In general flexible methods show a good way to decrease the needed number of operations to achieve the same accuracy as the standard solver but there are no criteria when to use the standard solver or the flexible method of it. Especially there are no which can be easily applied in practice for solving large equation systems to decide which kind of solver is preferable for solving this kind of matrix. The question is just when does the Flexible CG (F-CG) outperform the standard CG solver or vice versa in general. In all case studies seen so far the standard CG solver outperforms the Flexible CG whereas the Flexible GMRES outperforms in most cases the GMRES. Furthermore there is also the question about the optimal parametes for the flexible methods (F-GMRES) such that the lowest number of operations is achieved to converge to a specific residuum.

Partial recomputing is mainly used in parallel environments where all processors are distributed over different clusters and data transfer is mainly done via the network. One question about this approach is if this kind of fault detection and correction is also applicable on shared memory systems in an efficient way which means all used processors are on the same node and there is no data exchange via the network and how will it influence the compute overhead. The main question is just how will the number of used processors influences the compute overhead because of the data exchange and the implicit synchronization of all compute cores because of partial recomputing, in general there are no performance considerations like that.

ABFT is mainly done with checksums which leads to an additional compute overhead especially during applying the (sparse) matrix vector operation the number of additional operations depends on the number of non-zero elements of matrix A. So the question is just if the number of elements rises how does the compute overhead will increase because the compute overhead will depend on the size of matrix A and the density of the non-zero elements of matrix A.

The self-stabilization approach for the CG solver shows a good convergence behavior against the Flexible CG if a faults occures in the exponent. This approach is also important for other kinds of fault tolerant iterative linear solvers but is it also possible to use this property for the GMRES ?

5 Background

5.1 Iterative methods [33][34]

Most iterative methods are based on the fixed point iteration for searching roots of a function $f(x)$. In the fixed point method a solution is searched with the iterative process of $x_{j+1} = f(x_j)$ where index j is the current computation step. The vector x_{j+1} is the new computed solution for a given function $f(x_j)$ with x_j as the previous computed value(s). If the solution of x_{j+1} doesn't improve anymore after a number of j steps then the computation is terminated and a solution x_{j+1} is found with a given accuracy (i.e. stopped at $x_{j+1} \approx f(x_j)$). Iterative methods can be classified in stationary and non-stationary methods whereas non-stationary methods are related to Krylov methods (see section 5.2). Krylov methods are mainly used if the matrix A to solve $Ax = b$ has sparse density of all non-zero values, it means that most values of A are set zero.

Stationary methods: These are iterative methods where the data in the equation to compute x_{j+1} is fixed, they have the general form of:

$$x_{j+1} = Px_j + c. \tag{1}$$

Matrix P is called the iteration matrix and depends on the used algorithm but matrix P and the vector c do not change their values for each iteration step j.

Non-stationary methods (Krylov methods): These are methods where the data changes at each iteration step, they have the general form of:

$$x_{j+1} = x_j + \alpha_j p_j. \tag{2}$$

In this algorithm the vector p_j is called the search direction and α_j the step length or step size in the right direction to compute x_{j+1} where the residual ($||r||_2 = ||Ax - b||_2$) should be minimized. Note that the vector α_j and p_j change their values for each iteration step j. Non-stationary methods are considered as Krylov methods an introduction can be found in section 5.2.

For all iterative methods there must be always a stopping criterion if the solution x_j doesn't change anymore after a finite number of steps. One simple stopping rule would be the residuum which is defined as $||r_j||_2 = ||Ax_j - b||_2$ for each iteration j. Therefore one termination criterion would be:

$$\frac{||r_j||_2}{||b||_2} \leq \epsilon \text{ whereas } \epsilon \text{ is called the relative residuum.} \tag{3}$$

In extension of this approach there would be:

$$\frac{||x_{j+1} - x_j||_2}{||x_{j+1}||_2} \leq \delta_j \text{ with } \delta_j \text{ as the relative change between } x_{j+1} \text{ and } x_j. \tag{4}$$

To guarantee an improvement δ_j is decreased from the current iteration to the next iteration as long as the solution of x_{j+1} does not change anymore against the previous value x_j. If the condition is not satisfied after j iterations the computation is terminated and a new solution x_{j+1} is found.

It is common to use a combination of both criteria with δ_j and ϵ.

Comparison between iterative and direct methods [35][36]

There are mainly two classifications in direct and iterative methods for solving the linear system of equations $Ax = b$ where A is a matrix and with the right hand side b for the solution vector x. Direct methods are mainly based on the Gaussian elimination technique [6]. These methods are well-known for their robustness within general problems and dense matrices. Direct methods are not preferable for sparse matrices just because of the elimination process these techniques tend always to fill in non-zero elements in the matrix A.

These methods based on the Gaussian elimination for solving systems of linear equations try to compute the exact solution and do always have a certain amount of work with the same number of operations for the according input size of matrix A. Additionally if a lot of elements of matrix A are zeros some work is done which is not needed for solving the linear equation system because of unnecessary floating point operations.

In iterative methods mainly a sequence of matrix vector products is applied that try to better approximate the solution vector x for each new iteration step. Once a solution is near enough to the exact solution which minimizes the residuum ($||r||_2 = ||Ax-b||_2$) the computation is terminated. This stopping criterion leads to the main contrast of direct methods because it is just possible to define arbitrary precision for the solution vector x with iterative methods. Another difference to direct methods is that iterative methods can sometimes take advantage of the structure about the problem which is considered in the way how matrices are stored in the memory. In most cases it leads to the outcome that less computations are done if the used matrix A is sparse where some additional speed up can be gained with iterative methods in comparison to direct methods.

In practice a lot of problems lead to sparse matrices which means that a lot of values of matrix A are equal to zero. In this case iterative methods are always faster because of only doing the relevant work for all non-zero values. One drawback of iterative methods is that in some cases iterative methods don't converge to a sufficient solution.

Nevertheless iterative methods are quite faster than direct methods if matrix A is sparse. Just consider solving a linear system $Ax = b$ for matrix $A \in R^{n \times n}$ with size n the direct solution would take $\mathcal{O}(n^3)$ whereas the according iterative method would only take $\mathcal{O}(n^2)$ operations for the same result. The big $\mathcal{O}$ notation in this case is mainly related to the most relevant term which causes the highest costs (highest number of operations).

Direct methods should also not be applied if the entries of the used matrix A change because the factorization process of matrix A is really expensive. Therefore direct methods are only applicable for multiple right hand sides. In table 1 a comparison is done between the most important classes of solving very large equation systems. This comparison is an estimate over all classes of solvers. The decision when to use which solver is mainly based on the matrix structure and density of non-zero elements for matrix A, both processes (direct or iterative) are the main classes respective methods for solving a liner equation system of $Ax = b$.

Method	Computational Costs	Memory Storage
Direct	$\mathcal{O}(n^{2.3})$	$\mathcal{O}(n^{1.7})$
Iterative	$\mathcal{O}(n^{1.2})$	$\mathcal{O}(n)$

Table 1 Comparison between iterative and direct methods by means of computational and storage costs. [35][36]

5.2 Krylov subspace methods [5][16][37][38][39]

5.2.1 Introduction

Iterative methods which should solve the equation system $Ax = b$ are mainly used when direct methods are not efficient enough or only approximate solutions are needed. The most common used method for direct methods is the Gaussian LU-factorization [6] which is mainly used for non-sparse matrices, that's why direct methods are applied for dense matrices. The LU-factorization decomposes matrix A with $A \approx LU$ to a lower (L) and upper triangular matrix (U).

In contrast iterative methods are mainly used if equation systems ($Ax = b$) with sparse matrices have to be solved. Iterative methods for solving the equation system $Ax = b$ are based on Krylov methods which have the aim to find in each step a better approximation for the solution vector x than in the previous step. The aim is to find after a finite number of steps a good solution which should satisfy a given accuracy. This solution x computed with iterative and Krylov methods should be as close as possible to the real solution $x_\star$, the accuracy in comparison to direct methods can be arbitrary chosen depending on some input parameters like the number of iterations. This fact can be an advantage but also a disadvantage in comparison to direct methods. It is an advantage because the precision of the solution depends on the number of iterations and with a higher number of iterations a better solution can be found. In contrast if the solution cannot be improved anymore and the accuracy stagnates the chosen iterative method does not terminate. It also takes some effort to find some good parameters which solves the equation system $Ax = b$ efficiently.

These Krylov methods can be defined on different constraints, one of the most popular constraint is the computed residuum ($||r||_2 = ||Ax - b||_2$) or the property that it has to be orthogonal to each other residuum such that $r_j^T r_i = 0$ with $j \neq i$ (, orthogonality means that the product of the current residuum r_j and a different chosen r_i is zero) like in the case of the Conjugate Gradient (CG) method. These constraints are important to define the properties of the affine space. The affine space is nothing else as a projection from one Krylov subspace to the next computed one. It defines how to approximate from a given solution x to a better solution from one iteration to the next iteration which could have an impact on the computation speed and robustness of the used iterative method. One way how to define Krylov methods and the according affine space is shown in the next section 5.2.2 with the Jacobi method. Because of the slow computation speed a Krylov method can be speeded up with preconditioners.

Most important Krylov subspace methods:

- In the Conjugate Gradient method (CG, see section 5.2.9) all different chosen residuals r_j and r_i are orthogonal such that the product of $r_j^T r_i$ is zero with $j \neq i$. This method is derived from the Steepest Descent method (SD) but it is much more faster through searching a direction vector to minimize the residuum. The CG method is preferable for positive definite matrices (, it means that for any chosen vector x not equal to zero the constraint of $x^T Ax > 0$ is satisfied).
- In the Generalized Minimal Residual method (GMRES, see section 5.2.10), the norm ($||.||_2$) of the residual $r = Ax - b$ should be minimized. The GMRES solver is really robust because for each iteration a better solution for vector x can be found. This method can be applied to arbitrary matrices. GMRES is quite popular to use like the CG method but it is a stateful solver which means that this solver needs more memory storage to solve the problem $Ax = b$.

5.2.2 The Jacobi method [6][40]

In this section the Jacobi method is introduced which helps to define a Krylov subspace method. This is the easiest way to define a Krylov method so there can be also other ways to define Krylov subspace methods on way is just by the Jacobi method.

So let b the vector of the ride hand side from a given linear equation system $Ax = b$ with a square matrix $A \in \mathbb{R}^{n \times n}$ and size n then this equation system $Ax = b$ is a short form of:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{5}$$

, where the vector x should be the desired solution at the end of the computation. The matrix A just stores all elements of the equation system which can be a sparse as well non-sparse matrix and with the given right hand side b.

In the Jacobi method the matrix A is decomposed in a diagonal matrix D and the remainder of A with R. The computation speed of the Jacobi method mainly depends on the eigenvalues λ of matrix A with $Ax = \lambda x$ such that $(A - I\lambda)x = 0$ for any chosen vector x then $(A - I\lambda) = 0$ which must be solved for λ and with I as the according identity matrix and size of n. The Jacobi method like other Krylov methods converges really fast to the solution of x if the spectral radius ρ is lower than 1 with $\rho(D^{-1}R) < 1$ (or if $\rho(A) < 1$) where in the best case it should be almost zero.

The spectral radius ρ [6] of a matrix A is defined through $\rho(A) = \max\limits_{1 \leq k \leq m} |\lambda_k(A)|$ where the largest eigenvalue λ of $\rho(A)$ for a matrix A is computed from the origin of all eigenvalues with m as the number of independent eigenvalues. In practice the norm $||.||_2$ is used to compute the spectral radius of a matrix A with $\rho(A) = ||A||_2$ [6]. Let matrix A be decomposed with:

$$A = D + R, D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix}, R = \begin{bmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & 0 \end{bmatrix}. \tag{6}$$

So from $Ax = b$ it is obtained $(D + R)x = b$. In the Jacobi process a fix point method is applied where a solution is found with $x_{j+1} = f(x_j)$ until $x_{j+1} \approx x_j$. This formula can be rewritten to $x = D^{-1}(b - Rx)$. It follows that $x_j = D^{-1}(b - Rx_j)$ to solve the system $Ax = b$. A better solution of x_{j+1} can be now obtained with applying the Jacobi method respective fix point iteration:

$$x_{j+1} = f(x_j) = D^{-1}(b - Rx_j) \tag{7}$$

, with x_j which is the approximation in the j-th iteration of x where x_{j+1} is the next approximation for the final solution vector x if j goes to infinity ($j \to \infty$). In general the computation can be stopped after a certain number of iterations if the desired accuracy of the solution vector x is reached like for example the residual ($||r||_2$) is below a certain value (ϵ) such that $\frac{||r||_2}{||b||_2} \leq \epsilon$.

5.2.3 From Jacobi method to Krylov subspace methods [38]

The easiest way to introduce Krylov methods is to use the Jacobi method as introduced before. This method is the easiest and simplest method but has only a low convergence rate μ. This rate μ can be computed with $\lim_{j\to\infty} \frac{|x_{j+1}-x_*|}{|x_j-x_*|^q} = \mu$ [41] which depends on the used value of q $(= 1, 2, 3, \dots)$ and x_* as the exact solution where x_j is the current solution in step j. If q is set to 1 and the value of μ is between 0 and 1 then this method converges linearly for quadratic convergence q is 2.

The linear equation system $Ax = b$ with the matrix A can be rewritten which can be transformed to $A = D + (A - D)$ where D is a diagonal matrix, then it leads to the equation system of:

$$((D + (A - D))x = b \;\longleftrightarrow\; Dx = (D - A)x + b. \tag{8}$$

By applying the fixed point method like before this also leads to the Jacobi method:

$$x_{j+1} = \hat{B}x_j + \hat{b}$$

$$\text{, with } \hat{B} = I - D^{-1}A \text{ and } \hat{b} = D^{-1}b.$$

Then the approximate solution x_j should converge to the real solution x_* with:

$$x_j \to x_* \text{ if } j \to \infty$$

, where $x_* = A^{-1}b$ is the exact solution of the used problem by solving the problem directly. Then the defect vector d_j can be defined as:

$$d_j = x_j - x_*. \tag{9}$$

This vector d_j gives the distance from the exact solution x_* to best solution x_j found so far. Because of the fact that the exact solution x_* is not known the defect vector d_j cannot be computed in real this can be often the case. Therefore to check for convergence the residuum r_j can be used instead:

$$r_j = b - Ax_j \tag{10}$$

, it follows that:

$$r_j = -A(x_j - x_*) = -Ad_j \tag{11}$$

, because x_* solves $Ax = b$ with $x = x_*$. The residuum r_j is just used to indicate how far away the vector x_j is to satisfy $Ax_* = b$. Both the residuum r_j and the defect vector d_j can be used as an error correction method by iteratively applying this method the residuum would be minimized from one iteration j to the next iteration $j + 1$. This is the main goal of iterative linear solvers.

From the definition of the residuum r_j and by assuming that $D = I$ with $B = I - A$ it follows:

$$r_j = Bx_j + b - x_j = x_{j+1} - x_j, \text{ with } r_j = b - Ax_j.$$

Such that a new solution of x can be computed with x_{j+1}:

$$x_{j+1} = x_j + r_j. \tag{12}$$

By multiplying this formula with matrix $-A$ a recursion is obtained in the case of the residuum r_{j+1} and r_j with using the fact that $r_{j+1} - b = -Ax_{j+1}$ and $r_j - b = -Ax_j$ it follows:

$$r_{j+1} = r_j - Ar_j = Br_j. \tag{13}$$

This recursion defines different Krylov subspaces with the help of the Jacobi method. These subspaces are always spanned by computing the next residuum r_{j+1} from the current residuum r_j.

5.2.4 From recursion of the residuum ($r_j = Ax_j - b$) to Krylov subspace methods [38]

So with computing r_j and applying the recursion mentioned before and with also doing some inductions it follows that:

$$r_j = p_j(A)r_0 \ \in \ span \ \left\{r_0, Ar_0, \dots, A^j r_0\right\} \tag{14}$$

, with $p_j(\xi) = (1-\xi)^j$ which is a polynomial of exact degree j. From the definition of computing the residuum and by summing up the partial sums of all the previous computed residuals the approximate solution x_j can be computed:

$$x_j = x_0 + r_0 + \dots + r_{j-1} = x_0 + q_{j-1}(A)r_0 \tag{15}$$

, with the polynomial q_{j-1} and the degree of $j-1$. It follows that x_j lies in the affine space of $x_0 + \left\{r_0, Ar_0, \dots, A^{j-1}r_0\right\}$. This affine space of x_j can be computed by always shifting the subspace of r_{j-1} from one iteration to the next iteration. It takes $j+1$ matrix vector operations for q_{j-1} and $p_j(A)r_0$ because of the fact that the span over all subspaces $\left\{r_0, Ar_0, \dots, A^{j-1}r_0\right\}$ has to be computed. So the Krylov subspace method can be defined through the definition of x_j or r_j both possibilities will lead to a recursion to solve the linear equation system $Ax = b$ where a termination criterion must be defined.

In general the most expensive computation is the (sparse) matrix vector product $y = Ax$ for each iteration j, this is the most expensive task of all iterative methods but if matrix A to solve $Ax = b$ is sparse this task can be done really efficient. The speed up of Krylov subspace methods comes mainly from the fact that for iterative methods only matrices with a lot of zeros are used where only a sparse matrix vector operation should be applied for iterative methods.

5.2.5 Definition of the Krylov subspace [38]

From a nonsingular matrix $A \in C^{n \times n}$ with $y \neq o \in C^n$ and size of n, the m-th Krylov (sub)space $K_m(A, y)$ generated by A from y is:

$$K_m = K_m(A, y) = span(y, Ay, \dots, A^{m-1}, y). \tag{16}$$

It is obvious that $K_1 \subseteq K_2 \subseteq K_3 \dots \subseteq K_m$ and for each further iteration the dimension always increases by exactly one for each new computation step. One simple question is just when does the equal sign holds for $x_m \in x_0 + K_m(A, r_0)$. This is a termination criterion for iterative methods.

Let $x_\star$ be the solution of $Ax = b$ and with x_0 as any initial starting vector of it and $r_0 = b - Ax_0$ as the corresponding initial residual, so $x_\star$ is computed with:

$$x_\star \in x_0 + K_v(A, r_0) \tag{17}$$

, after a finite number of steps and $K_v(A, y)$ as the smallest A-*invariant* subspace that contains y. $K_v(A, y)$ has still the property that this subspace is in the affine space of K_m with the attribute of $K_1 \subseteq K_2 \subseteq K_3 \subseteq K_v \dots \subseteq K_m$. v is also called the grade of matrix A which gives the smallest $A - invariant$ subspace of $K_v(A, y)$ that contains y where all subspaces are independent. Independent means that there is nothing that equals two different Krylov subspaces.

If the vectors $r_0, Ar_0, \dots, A^{m-1}r_0$ are spanning the Krylov subspace K_m with m independent vectors then a dimension of $dim[K_m(A, r_0)] = m$ can be computed. Furthermore if there is an

$A^m r_0 \in K_m(A, r_0)$ for any index j which equals two Krylov subspaces then following equation can be obtained:

$$K_{m+j}(A, r_0) = K_m(A, r_0). \tag{18}$$

This is indeed the case when the residuum r_m is zero for exact precision and two Krylov subspaces are not independent anymore after index m then this iterative method terminates. The value of m is just the number of iterations when to stop this computing process.

Then it follows from the linear combination of:

$$0 = c_0 r_0 + c_1 A r_0 + \cdots + c_{m-1} A^{m-1} r_0 + c_m A^m r_0 \tag{19}$$

, and with the fact that at least the variables c_m and c_0 are unequal to zero ($c_m \neq 0$ and $c_0 \neq 0$) such that the linear combination is independent like the vectors $r_0, Ar_0, \ldots, A^{m-1} r_0$. If this is not the case no Krylov subspace can be found. If $r_0 \neq 0$ and $c_0 \neq 0$ then it is possible to compute the left side from this linear combination:

$$A^{-1} r_0 = \frac{-1}{c_0} \sum_{j=0}^{m} c_j A^{j-1} r_0 \in K_m(A, r_0). \tag{20}$$

The smallest index m with:

$$m = dim[K_m(A, r_0)] = dim[K_{m+1}(A, r_0)] \tag{21}$$

is called the grade of matrix A with respect to r_0 which is the smallest number of iterations when to stop this recursion or iterative process because no further independent Krylov subspace can be found.

Mainly the idea of iterative methods is to generate a sequence such that $x_m = x_\star$ with $x_m \in x_0 + K_m(A, r_0)$ of $Ax = b$ where the residuum r_m converges to zero. This true solution x_m is found if all residuals are linear independent and r_m is zero. This condition of the residuum r_m only holds for exact arithmetic not for machine precision in computer systems, therefore there must be a threshold value.

5.2.6 Definition of Krylov subspace methods [38]

An iterative method is called a Krylov subspace method for solving a linear equation system $Ax = b$ with starting from an arbitrary initial vector x_0 and the related residuum $r_0 = b - Ax_0$ if after m iterations exact m subspaces are created such that the desired solution x_m is found with the property that:

$$x_m - x_0 = q_{m-1}(A) r_0 \in K_m(A, r_0) \tag{22}$$

, where $x_m - x_0$ is in the affine space of K_m and with a polynomial q_{m-1} of order $m - 1$.

For the computed residual r_m another statement can be done if r_m exits:

$$r_m = p_m(A) r_0 \in r_0 + AK_m(A, r_0) \subseteq K_{m+1}(A, r_0). \tag{23}$$

There is always a new affine space which can be computed at each iteration step for the residual r_m and p_m as a polynomial of degree m which is related to the polynomial q_{m-1} with the exact degree $m - 1$. q_m is transformed to p_m with the relation of:

$$p_m(\xi) = 1 - \xi q_{m-1}(\xi) \tag{24}$$

, this relation also satisfies $p_m(0) = 1$.

Sometimes the residual of the m-th iteration cannot be computed therefore the condition if it exists is important. All residuals must be independent such that the product of $r_j^T r_i$ is zero for any chosen index j not equal to i ($j \neq i$) and with the computed residuals r_j and r_i where both should be independent perhaps this is not the case through rounding errors. If this problem occurs then there also exists no Krylov subspace and also no approximate solution x for the problem $Ax = b$. It is also worth to ask if an approximate solution can be found in a finite number of m steps within n iterations as size of matrix A that's why $x_m - x_0$ should be in the affine space of $q_{m-1}(A)r_0$.

5.2.7 Preconditioning of Krylov subspace solvers [33][38][42]

In most cases iterative solvers converge really slow because of the spectrum [6] if those eigenvalues of matrix A have too much distance between them. That's why preconditioning is the way to go because if the spectrum (see section 5.2.2) is lower than one the same used solver will be much faster than a solver without preconditioning. There are also a lot of cases where preconditioning is the only way to solve the linear equation system $Ax = b$. It is possible to decide between left, right and split preconditioning and afterwards the equation system is solved for $\hat{A}x = \hat{b}$.

Left Preconditioning:

$$CA(x) = Cb \tag{25}$$
$$(\hat{A}x = \hat{b}).$$

Right Preconditioning:

$$AC(C^{-1}x) = b \tag{26}$$
$$(\hat{A}\hat{x} = \hat{b}).$$

Split Preconditioning:

$$C_L A C_R (C_R^{-1} x) = C_L b \tag{27}$$
$$(\hat{A}\hat{x} = \hat{b}).$$

There are different ways how preconditioning can be done, the first two are left and right preconditioning whereas the last one is split preconditioning. Matrix C and the split preconditioner with $C_L C_R$ are the approximate inverse of matrix A, so matrix C should have the property that $CA = I$ in the case of left preconditioning. Sometimes C is replaced with matrix M and with the property that $M \approx A$ which means a matrix M is searched such that $I \approx M^{-1}A$ with I as the identity matrix. Then in the case of right preconditioning $x = M^{-1}u$ solves the problem in an efficient way for $AM^{-1}u = b$. From the above formulas matrix C is replaced with M^{-1} whereas matrix C_L and C_R are replaced with M_L^{-1}, M_R^{-1} if matrix M is used instead of matrix C. In preconditioned Krylov subspace methods the system $\hat{A}\hat{x} = b$ is solved first and afterwards the solution is back substituted.

The LU-factorization [6] for full matrices is some way of preconditioning because it scales all values of around 1 and beneath and bounds the error of the solution vector x to a lower value. In the case of the LU-factorization the first step is to factorize A with $A = PLU$ (P ... pivoting matrix, L ... lower matrix, U ... upper matrix). Then solve $Pz = b$ ($PLUx = b$) with $z = P^T b$ and afterwards apply $Ly = z$ and $Ux = y$.

For sparse matrices often the incomplete LU (ILU [6]) decomposition is applied where matrix L (lower matrix) and U (upper matrix) of matrix A are chosen such that there are less fill ins in comparison to the standard Gaussian factorization. The incomplete LU (ILU [6]) factorization does similar the same like the standard LU-factorization but tries to avoid to fill matrix L and U with unnecessary elements.

Often there is a parameter called threshold value or factorization error to determine when to stop the factorization of matrix A. This parameter is used to determine if this new computed value with a non-zero fill in should be deleted in relation to the other values. There are different algorithms of how to apply incomplete decomposition for matrix A that's why it is hard to determine the number of operations for incomplete factorization generally.

5.2.8 Inexact Krylov methods [43][44][45]

In every Krylov method a matrix vector product is applied such that $y = Ax$. In contrast to exact Krylov methods a different matrix $\mathcal{A}$ is considered to build all these subspaces for each iteration. Matrix A is now replaced with $\mathcal{A}$ whereas the matrix E represents the difference between the exact matrix A and the perturbed matrix $\mathcal{A}$ such that $\mathcal{A} = A+E$. Matrix E stores the perturbations mainly because of rounding errors from the floating point representation and the input itself. Matrix E will lead to a different Krylov subspace K_m which is now disturbed:

$$K_m = K_m(\mathcal{A}, y) = span(y, \mathcal{A}y, \ldots, \mathcal{A}^{m-1}, y). \tag{28}$$

This Krylov subspace gives also a bound for the vector y because $||y||_2 \leq ||\mathcal{A}||_2||x||_2$. In general there can be only computed upper bounds for exact and inexact Krylov methods. Inexact Krylov is a special term which comes from numeric aspects with mainly focusing on the error which can be caused from the used matrix A. Because of the fact that this matrix A is also error prone through the representation of floating point values some further considerations about the Krylov subspace have to be done.

Like in [43] its focus is mainly on the GMRES solver and the built Krylov subspace but this paper also shows the possibility of how to extend this approach to other iterative methods. The error matrix E has to be considered for further operations but it also allows to bound the maximum allowable error which can be done with the GMRES solver. This given bound can be really useful for finding a termination criterion for solving the equation system $Ax = b$.

In this model it is still allowed that matrix E changes from one iteration to the next iteration. In [44] the author tries to determine the convergence behavior of the GMRES but as well for the CG solver with the point of view for exact and inexact Krylov methods. This paper is more dedicated to the convergence behavior of those two mentioned solvers. Something which is left is how flexible methods (see section 9) will change the built inexact Krylov subspace in terms of the error bound and the speed of solving the problem $Ax = b$.

Something more can be found in [45] about the perturbation theory of the GMRES solver, the focus is still mainly on the perturbed Krylov subspace of matrix $\mathcal{A}$.

5.2.9 The Conjugate Gradient (CG) and Steepest Descent (SD) method[16][37]

The Conjugate Gradient (CG) is mainly based on the Steepest Descent (SD) method both are discussed in this master work. The CG method is one of the most popular methods for positive definite matrices (see formula 29) which converges really fast to the solution x for solving the problem of $Ax = b$. The CG like the SD method can be only applied for symmetric positive definite matrices.

5.2.9.1 The Steepest Descent method (SD) [16][37]

For a given matrix A this matrix is positive definite if and only if the following equation holds:

$$x^T Ax > 0, \text{ for any chosen vector } x \text{ except the zero vector.} \tag{29}$$

This leads to a quadratic form for a function $f(x)$ which is similar to the expression of the residuum but with respect to positive definiteness:

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c, \text{ with } c \text{ as any constant scalar and } b \text{ as the right hand side.} \tag{30}$$

If the derive $f(x)'$ with respect to x of this function $f(x)$ is computed it leads to following form:

$$f(x)' = \frac{1}{2}A^T x + \frac{1}{2}Ax - b. \tag{31}$$

The function $f(x)$ should be minimized but with the property that matrix A should be positive definite as well symmetric ($A^T = A$) then the function $f(x)$ is solved for:

$$f(x)' = Ax - b, \text{ note that } \frac{1}{2}(A^T + A)x = b \text{ is symmetric.} \tag{32}$$

This property comes from the fact that if a look at following relationship is done:

$$f(p) = f(x_\star) + \frac{1}{2}(p - x_\star)^T A(p - x_\star). \tag{33}$$

, with p as an arbitrary chosen point. The right term will be always positive if a different point p is chosen from $x_\star$ ($p \neq x_\star$) but with $x_\star$ as the minimum of $f(x)$ if matrix A is positive definite.

The SD method is started with an arbitrary starting vector x_0 and after computing a series of points $x_1, x_2, x_3, \ldots$ the function $f(x)$ should be minimized. In the case of the SD and CG method the function $f(x)$ should be minimized so it must be go in a specific direction of $f(x_j)'$ with $-f(x_j)' = b - Ax_j$, the so called gradient (direction) of this function $f(x)$ which will always point upwards but it must be go downwards to minimize the residuum ($||r||_2$). In this method also computing a step size α is needed to know how far to go to minimize the residuum ($||r||_2 = ||Ax - b||_2$). Let d_j be the error from the solution $x_\star$ with $d_j = x_j - x_\star$. Then d_j indicates how far x_j is from the real solution $x_\star$ for each iteration j and with the according residual $r_j = b - Ax_j$ which indicates the distance of Ax_j to the right hand side b. It is easy to see that $r_j = -Ad_j$ but also $r_j = f(x_j)'$.

In the SD method a step size α is searched along the residuum r such that:

$$x_1 = x_0 + \alpha r_0 \tag{34}$$

, for the initial residuum r_0 and starting vector x_0 with x_1 as a new computed solution.

Let $\frac{d}{d\alpha}f(x_1)$ be the directional derivative which minimizes the function $f(x_1)$ with respect to α such that $\frac{d}{d\alpha}f(x_1)$ is zero for the point x_1. Then by the chain rule $\frac{d}{d\alpha}f(x_1) = f'(x_1)^T \frac{d}{d\alpha}x_1 = f'(x_1)^T r_0$, it follows that $\frac{d}{d\alpha}f(x_1)$ must be orthogonal such that $f'(x_1)^T \frac{d}{d\alpha}x_1 = 0$ for the related step size α. To compute the step size α note that $f'(x_1) = -r_1$, so let $r_1^T r_0 = 0$ then it follows that for α:

$$\begin{aligned}
(b - Ax_1)^T r_0 &= 0 \\
(b - A(x_0 + \alpha r_0))^T r_0 &= 0 \\
(b - Ax_0)^T r_0 - \alpha (Ar_0)^T r_0 &= 0 \\
(b - Ax_0)^T r_0 &= \alpha (Ar_0)^T r_0 \\
r_0^T r_0 &= \alpha (Ar_0)^T r_0 \\
\alpha &= \frac{r_0^T r_0}{r_0^T A r_0}. \qquad (35)
\end{aligned}$$

So by putting it all together the algorithm of the Steepest Descent (SD) method is as follows:

$$\begin{aligned}
r_j &= b - Ax_j \\
\alpha_j &= \tfrac{r_j^T r_j}{r_j^T A r_j} \\
x_{j+1} &= x_i + \alpha_j r_j. \qquad (36)
\end{aligned}$$

The next residuum is obtained by also multiplying the last equation with $-A$ and $-b$:

$$r_{j+1} = r_j + \alpha_j A r_j. \qquad (37)$$

Algorithm 1 The Steepest Descent (SD) algorithm for solving a symmetric positive definite system (SPD) [32]:

Input: Matrix A, right hand side b with the initial starting vector x_0 and ϵ as the accuracy for solving $Ax = b$ where m is the number of maximum allowable iterations.

Output: Approximate solution of vector x_j for $j > 0$.

```
$j = 0$, $r_0 = b - Ax_0$, $x_j = x_0$            ▷ Compute starting residuum $r_0$.
$||r_j||_2^2 = ||r_0||_2^2 = r_j^T r_j$
while $||r_j||_2/||r_0||_2 > \epsilon$ and $j < m$ do    ▷ Check for convergence.
    $q_j = Ar_j$                                  ▷ Perform matrix vector product.
    $\alpha_j = ||r_j||_2^2/(r_j^T q_j)$              ▷ Compute new step size $\alpha_j$.
    $x_{j+1} = x_j + \alpha_j r_j$                    ▷ Do the according step for current solution $x_j$.
    $r_{j+1} = r_j - \alpha_j q_j$                    ▷ Compute a new residuum for $x_j$.
    $||r_{j+1}||_2^2 = r_{j+1}^T r_{j+1}$             ▷ Compute a new normalized residuum for $||r_{j+1}||_2^2$.
    $j = j + 1$                                   ▷ Compute next iteration index $j$.
end while
return $x_j$
```

In algorithm 1 the SD method is shown which minimizes the residuum $r = b - Ax$ for symmetric positive definite matrices (SPD). In general the SD method converges really slow in contrast to the CG method so the CG method is always preferable for solving symmetric positive matrices (SPD).

5.2.9.2 The Conjugate Gradient method (CG) [16][37]

In contrast to the SD method the CG algorithm computes some orthogonal search directions for the vectors $p_0, p_1, \dots, p_{m-1}$ furthermore for each search direction p also a step size α is computed to minimize the residuum $r = Ax - b$. The following sequence should solve the problem $Ax = b$ in a finite number of steps with the iteration index j:

$$x_{j+1} = x_j + \alpha_j p_j. \tag{38}$$

So let a set of vectors $\{p_1, \dots, p_j\}$ be orthogonal with the respect to the positive symmetric matrix A then such following constraint can be obtained:

$p_j^T A p_i = 0$, whenever $j \neq i$ which means that vector p_j and p_i are orthogonal to A. (39)

There is a sequence of different x_js which converges to the desired solution $x_\star$ as long as j is less than m within at most m steps such that $j \to m$. This sequence should be linear independent with the respect to the search direction p, so following equation can be obtained by also considering α:

$$x_\star - x_0 = \sum_{i=0}^{m-1} \alpha_i p_i. \tag{40}$$

Equation 40 is based on 38 where a sequence is searched such that the residuum r is minimized, at the end all computation steps are summarized to compute $x_\star$ with the initial value x_0. So if this equation 40 is multiplied with $p_j^T A$ from the left and by considering $Ax^\star = b$ but also using the constraint $p_j^T A p_i = 0$ whenever $j \neq i$ because all search directions must be A-orthogonal it follows:

$$p_j^T A(x_* - x_0) = p_j^T (b - Ax_0) = p_j^T r_0. \tag{41}$$

On the right side following equation can be obtained:

$$p_j^T A \sum_{i=0}^{m-1} \alpha_0 p_0 = \alpha_j p_j^T A p_j.$$

Therefore:

$$p_j^T r_0 = \alpha_j p_j^T A p_j. \tag{42}$$

It follows that:

$$\alpha_j = \frac{p_j^T r_0}{p_j^T A p_j} = \frac{p_j^T r_j}{p_j^T A p_j} \tag{43}$$

, because of the obvious fact that $p_j^T r_0 = p_j^T r_j$.

Thus a search direction α_j is found. Also let r_{j+1} be the next computed residuum:

$$r_{j+1} = b - Ax_{j+1} = b - A(x_j + \alpha_j p_j) = r_j - \alpha_j A p_j. \tag{44}$$

Alternatively a new direction p_{j+1} is computed from the current p_j with the following sequence such that p_j and p_{j+1} are linear independent ($p_j p_{j+1} = 0 \; \forall j$) for all indexes j:

$$p_{j+1} = r_{j+1} + \beta_j p_j. \tag{45}$$

The restriction of independence for p_j and p_{j+1} also leads to the fact that the product of $p_j^T p_i$ is zero for all indexes j unequal i ($\forall j \neq i$). From the expression of r_{j+1} and p_{j+1} the step size β_j can be computed for the next orthogonal search direction p_{j+1}, so it follows that:

$$\beta_j = -\frac{r_{j+1}^T A p_j}{p_j^T A p_j} \text{ by equation 45 and } p_{j+1}^T A p_j = 0.$$

$$\beta_j = -\frac{r_{j+1}^T A(r_j - r_{j+1})}{\alpha_j A p_j^T A p_j} \text{ by equation 44.}$$

$$\beta_j = -\frac{r_{j+1}^T (r_j - r_{j+1})}{r_j^T p_j} \text{ by equation 43.}$$

$$\beta_j = +\frac{r_{j+1}^T r_{j+1}}{r_j^T r_j} \text{with the help of } r_{j+1}^T r_j = 0. \tag{46}$$

The last equation is obtained from the condition that all computed residuals must be independent such that the product of $r_j^T r_i$ is zero for all indexes j unequal i ($j \neq i$) and for any chosen residuum r_j not equal r_i ($r_j \neq r_i$). This orthogonality property is also obtained by the SD method. In algorithm 2 the CG method is shown which is one of the most popular algorithms to solve problems of $Ax = b$ for symmetric positive definite (SPD) matrices. It has the input of matrix A, the right hand side b and the number of maximum allowable iterations m to converge to the solution x with accuracy ϵ.

Algorithm 2 The Conjugate Gradient (CG) algorithm for solving a symmetric positive definite system (SPD) [32]:

```
Input: Matrix $A$, right hand side $b$ with the initial starting vector $x_0$ and $\epsilon$ as the accuracy for
    solving $Ax = b$ where $m$ is the number of maximum allowable iterations.
Output: Approximate solution of vector $x_j$ for $j > 0$.
1:  $j = 0$, $r_0 = b - Ax_0$, $x_j = x_0$                ▷ Compute starting residuum $r_0$.
2:  $p_0 = r_0$, $||r_j||_2^2 = ||r_0||_2^2 = r_0^T r_0$
3:  while $||r_j||_2 > \epsilon \times ||r_0||_2$ and $j < m$ do   ▷ Check for convergence.
4:      $q_j = Ar_j$                                  ▷ Perform matrix vector product.
5:      $\alpha_j = ||r_j||_2^2 / (p_j^T q_j)$             ▷ Compute new step size $\alpha_j$.
6:      $x_{j+1} = x_j + \alpha_j r_j$                    ▷ Do the according step for current solution $x_j$.
7:      $r_{j+1} = r_j - \alpha_j q_j$                    ▷ Compute a new residuum for $r_{j+1}$.
8:      $\beta = ||r_{j+1}||_2^2 / ||r_j||_2^2$             ▷ Compute a new step size $\beta$ for search direction $p_j$.
9:      $p_{j+1} = r_{j+1} + \beta p_j$                   ▷ New search direction for $p_{j+1}$.
10:     $j = j + 1$                                   ▷ Compute next iteration index $j$.
11: end while
12: return $x_j$
```

One main advantage of the CG method is that there is always a constant work for each iteration step because there is only a sparse matrix vector product (line 4) and some vector operations (line 5 - 9). The number of operations in the main loop of the CG solver can be obtained in section 6.1.4.

5.2.10 GMRES and orthogonalizing with classical Gram Schmidt (GS) and modified Gram Schmidt (MGS) methods [5][39][46][47]

5.2.10.1 Introduction and the orthogonalization process of the GMRES solver

The most important part of the GMRES solver is the Arnoldi process. This method mainly tries to solve the problem $Ax = b$ through orthogonalization of the residuum r for matrix A and the right hand side b. This process also builds a Krylov subspace with $K_m = K_m(A, r) = [r, Ar, A^2r, \dots, A^{m-1}r]$ like in any other iterative method. The GMRES solver should minimize the problem of $\frac{1}{2}||b - Ax_m||_2^2$ for vector $x_m \in x_0 + K_m$, x_m is the computed solution after m iterations where x_0 is the initial solution for the Krylov subspace (matrix) K_m. In the Arnoldi process a basis matrix Q_m is computed with the vectors of $q_1, q_2, \dots, q_m$ such that $Q_m = [q_1, q_2, \dots, q_m]$ where the main property of each vector q_j should be orthogonality to each other vector q_i such that the product of $q_j^T q_i$ is zero for all indexes j unequal to i ($\forall j \neq i$).

Matrix Q_m stores all computed basis vectors $q_1, q_2, \dots, q_m$, this matrix Q_m has size $n \times m$ where n is the length of vector x and with m for the number of iterations. This orthogonalization process also leads to the so called Hessenberg matrix H_m with $H_m = Q_m^{-1} A Q_m$ which comes mainly from the similarity transformation of $Q_m H_m = A Q_m$. The so called Hessenberg matrix H_m is computed with the orthogonal matrix Q_m which obtains those eigenvalues λ of matrix A ($Ax = \lambda x$). Orthogonal matrices have the property that the product of $QQ^T (Q^T Q)$ is I with I as the identity matrix and size n. It leads to the fact that matrix A multiplied from the left and right ($Q^{-1}AQ$) with the orthogonal matrix Q^{-1} and Q will not be scaled but only the values of matrix A are projected such that the main properties of A are unchanged. In this iterative process the size of H_m increases for each new iteration of m. This iterative process of the GMRES solver can be now obtained with $AQ_m = Q_m H_m$ and using a basis matrix Q_m, now take $j < m$ then the equation of $AQ_m = Q_m H_m$ with the form:

$$AQ_m = Q_m \times \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \cdots & & h_{1,j} & \cdots & & h_{1,m} \\ h_{2,1} & h_{2,2} & h_{2,3} & \cdots & & h_{2,j} & \cdots & & h_{2,m} \\ 0 & h_{3,2} & h_{3,3} & \cdots & & h_{3,j} & \cdots & & h_{3,m} \\ \vdots & 0 & h_{4,3} & \ddots & & \vdots & & & \\ & & 0 & \ddots & h_{j-1,j-2} & & & & \vdots \\ & & & 0 & h_{j,j-1} & h_{j,j} & & & \\ \vdots & \ddots & \ddots & & 0 & h_{j+1,j} & & & \vdots \\ \vdots & \ddots & \ddots & & & 0 & \ddots & \ddots & \vdots \\ 0 & \cdots & & & & & 0 & h_{m,m-1} & h_{m,m} \end{bmatrix} \tag{47}$$

, can be reformulated. Afterwards consider only parts of the equation namely Q_j and Q_{j+1} then following equation is obtained with $AQ_j = Q_{j+1}\hat{H}_j$ and a different matrix $\hat{H}_j$ such that:

$$\hat{H}_j = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & & \cdots & h_{1,j} \\ h_{2,1} & h_{2,2} & h_{2,3} & & \cdots & h_{2,j} \\ 0 & h_{3,2} & h_{3,3} & & \cdots & h_{3,j} \\ & 0 & h_{4,3} & \ddots & \vdots & \\ & & 0 & \ddots & h_{j-1,j-2} & \vdots \\ \vdots & & & & h_{j,j-1} & h_{j,j} \\ 0 & & & & 0 & h_{j+1,j} \end{bmatrix} \tag{48}$$

, with matrix $A \in \mathbb{C}^{n \times n}$, $Q_j \in \mathbb{C}^{n \times j}$, $Q_{j+1} \in \mathbb{C}^{n \times (j+1)}$ and the Hessenberg matrix $\hat{H}_j \in \mathbb{C}^{(j+1) \times j}$ such that both sides of the equation system $AQ_j = Q_{j+1}\hat{H}_j$ hold a matrix with size $n \times j$. Matrix $\mathbb{C}$ can also have real and complex values. If both sides from the previous equation system ($AQ_j = Q_{j+1}\hat{H}_j$) are compared then following equation can be obtained for each single basis vector q:

$$Aq_j = h_{1,j}q_1 + h_{2,j}q_2 + \cdots + h_{j,j}q_j + h_{j+1,j}q_{j+1}. \tag{49}$$

An iterative process is revealed for computing the next basis vector q_{j+1} from the previous vectors:

$$q_{j+1} = \frac{Aq_j - \sum_{i=0}^{j} h_{i,j}q_i}{h_{j+1,j}}. \tag{50}$$

This recursive process for the unitary (orthogonal) matrix Q_j ($Q_jQ_j^H = Q_j^HQ_j = I$) which computes the orthogonal basis vectors $q_1, \ldots, q_j$ is formally known as the Arnoldi process. Unitary matrices have complex values whereas orthogonal matrices have only real values such that $Q^H = Q^{-1}$, Q^H is the conjugate transpose for matrix Q (unitary matrix).

In the first step of this process the iterative method is started from the normalized vector q_1 such that $||q_1||_2 = 1$ for an arbitrary matrix A, it follows for the next computed basis vector q_2 that:

$$q_2 = \frac{Aq_1 - h_{11}q_1}{h_{21}}.$$

Hence the most compute intensive part of the GMRES algorithm is the sparse matrix vector product of Aq_i which can be done by an efficient subroutine. Now because of the fact that the product of q_1 and q_2 should be orthogonal such that $q_1^Tq_2 = 0$ it holds that:

$$0 = q_1^TAq_1 - h_{11}q_1^Tq_1.$$

So by taking the advantage of the normalization for vector q_1 ($q_1^Tq_1 = 1$) it follows that:

$$q_1^TAq_1 = h_{11}q_1^Tq_1$$

$$h_{11} = \frac{q_1^TAq_1}{q_1^Tq_1} = \frac{q_1^TAq_1}{1}.$$

Finally let the vector $v = Aq_1 - h_{11}q_1$ and compute $h_{21} = ||v||_2$ for normalization of the next basis vector q_2 then following equation can be obtained:

$$q_2 = \frac{v}{h_{21}}.$$

This will lead to the so called Arnoldi process or formally known Gram Schmidt process for finding m independent basis vectors stored in matrix Q_m:

$$q_j = \frac{Aq_j - \sum_{i=1}^{j-1} h_{i,j}q_i}{||v_{j,j}||_2} \tag{51}$$

, this process is presented in algorithm 3 which is also called classical Gram Schmidt method [48]. The aim of this process is to find m independent basis vectors for the matrix A in m iterations which are stored in the matrix Q_m but also the values of the Hessenberg matrix $\hat{H}$ must be saved in $\hat{H}_m$. The Classical Gram Schmidt (GS) and Modifiied Gram Schmidt (MGS) process are similar which is shown in algorithm 3 and 4 because only a single line has changed (line 5). The classical GS and MGS methods are mathematically the same, the main contrast is that Aq_j is replaced with v_j in the algorithm of MGS. This operation (Aq_j) is only done once which leads to higher stability but also to a lower number of operations. In the MGS algorithm orthogonalization is done against the previous computed basis vectors which also leads to a higher stability.

Algorithm 3 Classical GS algorithm [47][48]:

```
1:  Let $x_0 \in R^n$
2:  Choose $r^0 = b - Ax^0$, $v^1 = \frac{r^0}{||r^0||}$
3:  for $j = 1, \ldots, m$ do
4:      $v_j = Aq_j$
5:      for $i = 1, \ldots, j$ do
6:          $h_{i,j} = (q_i)^T (Aq_j)$
7:          $v_j = v_j - h_{i,j} v_i$
8:      end for
9:      $h_{j+1,j} = ||v_j||$
10:     $q_{j+1} = v_j / ||h_{j+1,j}||$
11: end for
```

Algorithm 4 Modified GS algorithm [47][48]:

```
1:  Let $x_0 \in R^n$
2:  Choose $r^0 = b - Ax^0$, $v^1 = \frac{r^0}{||r^0||}$
3:  for $j = 1, \ldots, m$ do
4:      $v_j = Aq_j$
5:      for $i = 1, \ldots, j$ do
6:          $h_{i,j} = (q_i)^T (v_j)$
7:          $v_j = v_j - h_{i,j} v_i$
8:      end for
9:      $h_{j+1,j} = ||v_j||$
10:     $q_{j+1} = v_j / ||h_{j+1,j}||$
11: end for
```

Since stability takes a big concern the invariant $||1 - q_j^T q_j||_2 \leq \epsilon_{Ortho}$ [48] with ϵ_{Ortho} as a threshold value can be used to give a good prospect about the orthogonality but as well stability of the vector q_j which can be computed for each iteration j. This property of orthogonality for the GMRES solver might be crucial for solving $Ax = b$ because it mainly affects the stability of solving $Ax = b$.

5.2.10.2 Arnoldi Iteration as Projection onto Krylov subspaces [39]:

An alternative for introducing the Arnoldi process is to start with the Krylov matrix K_m:

$$K_m = [r, Ar, A^2 r, \ldots, A^{m-1} r]. \tag{52}$$

Then by multiplying from the left side with matrix A following Krylov subspace is obtained:

$$AK_m = [Ar, A^2 r, A^3 r, \ldots, A^m r]. \tag{53}$$

For both equations before the last equation can be rewritten to:

$$AK_m = K_m [e_2, e_3, \ldots, e_m, -c] \tag{54}$$

, where

$$c = -K_m^{-1} A^m r. \tag{55}$$

Then by assuming K_m is invertible it follows equivalently:

$$AK_m = K_m C_m \tag{56}$$

, with the upper Hessenberg matrix C_m:

$$C_m = [e_2, e_3, \ldots, e_m, -c]. \tag{57}$$

It can be shown that matrix A and C_m are similar through the formula $K_m^{-1} A K_m = C_m$. The main problem is that matrix K_m is usually ill-conditioned because of the product $A^m r$, it converges to the most dominating eigenvalue of A, ill-conditioned problems can be only solved hard. It also means that inverting matrix K_m is not a good approach and should be avoided as much as possible because small perturbations would make solving the problem $AK_m = K_mC_m$ inaccurate. Let K_m be decomposed with matrix Q_m and R_m such that $K_m = Q_mR_m$ also called QR-factorization [6] with an orthogonal matrix Q_m and an upper tridiagonal matrix R_m it follows for matrix K_m:

$$\begin{aligned} K_m &= Q_mR_m \\ K_m^{-1}AK_m &= C_m \\ R_m^{-1}Q_m^{-1}AQ_mR_m &= C_m \\ Q_m^{-1}AQ_m &= R_mC_mR_m^{-1} \\ Q_m^{-1}AQ_m &= H_m. \end{aligned} \tag{58}$$

This equation 58 is also a similarity transformation from matrix A to H_m with $Q_m^{-1}AQ_m = H_m$. It should be pointed out that this is not a good approach because it is computationally intensive and unstable. Computing the vector c involves solving the (ill-conditioned) linear system $K_mc = A^mb$, it would also need to find the inverse of R_m. However the formula which is observed above can be interpreted as an orthogonal projection of A on K_m or H_m because:

$$Q_m^{-1}AQ_m = H_m. \tag{59}$$

, whereas matrix Q_m represents the basis vectors of all columns about the Krylov matrix K_m.

In the GMRES method the residuum r_m should be minimized after some finite number of steps where the real solution $x_\star$ is found with $x_\star = A^{-1}b$, so the GRMES should minimize $||r_m||_2$ with:

$$||r_m||_2 = ||b - Ax_m||_2 \rightarrow min. \tag{60}$$

Now start with the Krylov matrix K_m, thus the column space of K_m is AK_m.
The desired vector $x_m \in K_m$ can be found with:

$$x_m = x_0 + K_mc, \text{ to solve } Ax = b. \tag{61}$$

Then the problem becomes:

$$||r_m||_2 = ||b - A(x_0 + K_mc)||_2 \rightarrow min. \tag{62}$$

Instead to use the Krylov subspace K_m the basis vectors of Q_m are used instead with the factorization of $K_m = Q_mR_m$ (y is still the solution of equation 55) then the same problem is solved with:

$$x_m = x_0 + Q_my. \tag{63}$$

, afterwards a solution x_m is computed.

Let $x_m = x_0 + Q_my$ with $y \in \mathbb{R}^m$ and $e_1 = (1, 0, \ldots, 0)^T \in R^{m+1}$ it follows for the residuum r_m:

$$\begin{aligned} r_m = ||b - Ax||_2 &= ||b - A(x_0 + Q_my)||_2 = ||r_0 - AQ_my||_2 \\ &= ||||r_0||_2v_1 - AQ_my||_2 = ||||r_0||_2v_1 - Q_{m+1}\hat{H}_my||_2 \\ &= ||Q_{m+1}(||r_0||_2e_1 - \hat{H}_my))||_2 = ||||r_0||_2e_1 - \hat{H}_my||_2. \end{aligned} \tag{64}$$

The last steps follows from $Q_{m+1}Q_{m+1}^T = I$ and with the help of $v_1 = Q_{m+1}e_1$. Now the problem $r_m = ||b - Ax||_2 \rightarrow min$ can be rewritten to $r_m = ||\beta e_1 - \hat{H}_my||_2 \rightarrow min$ with $\beta = ||r_0||_2$.

5.2.10.3 Applying Givens rotation in GMRES [5][47]:

Because the Hessenberg matrix $\hat{H}_m$ is not an upper triangular matrix like in equation 66, it makes it hard to solve the problem $\hat{H}_m y_m = \beta e_1$ for vector y_m therefore another projection method is used which is called the Givens rotation such that all values below the main diagonal are set to zero. This method reduces the number of non-zero values for the matrix $\hat{H}_m$ with the appearance of:

$$\hat{H}_m = \begin{bmatrix} * & * & * & \dots & * \\ * & * & * & \dots & * \\ & * & \ddots & \ddots & \vdots \\ & & * & \ddots & * \\ 0 & & & * & * \end{bmatrix} \tag{65}$$

(Values marked with "*" are non-zero values.)

, which can be more efficiently solved afterwards for the right hand side βe_1 and vector y_m but also with a lower number of operations. The Givens rotation reduces the Hessenberg matrix $\hat{H}_m$ with a rotation plane G at each iteration of m. It follows that there are only m Givens rotations needed at all for m iterations whereas matrix $\hat{R}_m$ and R_m are in general an upper triangular matrix. Then the new computed Hessenberg matrix $\hat{H}_m$ after applying m Givens rotation has the appearance of:

$$G_{m+1,m} \dots G_{32} G_{21} \hat{H}_m = \begin{bmatrix} * & * & * & \dots & * \\ & * & * & \dots & * \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & * \\ 0 & & & & * \end{bmatrix} = \hat{R}_m = \begin{bmatrix} R_m \\ 0 \end{bmatrix} \tag{66}$$

, with matrices $G_{21} \dots G_{32} \dots G_{m+1,m} \in \mathbb{R}^{(m+1)\times(m+1)}$ as the according rotate plane for the Givens rotation but also only the last matrix column of $\hat{H}_{m+1}$ has to be updated from the last iteration because all other non-zero values are already set to zero such that:

$$\hat{H}_{m+1} = \left[\begin{array}{cccc|c} & & & & * \\ & \hat{H}_m & & & \vdots \\ & & & & \vdots \\ & & & & * \\ \hline 0 & \dots & \dots & 0 & * \end{array}\right]. \tag{67}$$

In the end there should be m Givens rotations done for m iterations for reducing the number of non-zero elements in matrix $\hat{H}_m$, now problems for matrix $\hat{H}_m$ can be solved more efficiently.

With applying Givens rotations and the substituting of $\beta e_1 = \hat{z}_m$ it follows for the residuum:

$$||r||_2 = ||b - Ax||_2 = ||\beta e_1 - \hat{H}_m y||_2 = ||\hat{z}_m - \hat{R}_m y||_2 \tag{68}$$

, with

$$\hat{z}_m = G_{m+1,m} \begin{bmatrix} \hat{z}_{m-1} \\ 0 \end{bmatrix}. \tag{69}$$

The vector $\hat{z}_m$ can be presented like: $\hat{z}_m = \begin{bmatrix} \hat{z}_m \\ \sigma_m \end{bmatrix}$ whereas σ_m is some error term.

From this notation the residuum can be expressed as:

$$||r_m||_2 = ||b - Ax_m||_2 = ||\hat{z}_m - R_m y_m||_2^2 = ||z_m - R_m y_m||_2^2 + ||\sigma_m||_2^2 = ||\sigma_m||_2^2. \tag{70}$$

Using $||\sigma_m||_2^2$ can be a good indicator for the residuum ($||r_m||_2 = ||Ax_m - b||_2$) and will save a lot of floating point operations because computing the residuum r_m always needs a sparse matrix operation (Ax) and a subtraction between two vectors. In this work $||\sigma_m||_2^2$ is also called the approximation error and can be also seen as a pseudo residuum because $||r_m||_2$ is not computed directly.

During applying the Givens rotation a matrix G must be computed to attain an upper triangular matrix, this matrix G must be orthogonal and unitary which means that the determinate of G must be 1 such that $det(G) = 1$. From the fact that the determinate must be 1 it follows that matrix G is just a rotation plane and will not scale the values of the Hessenberg matrix $\hat{H}$ in any direction. The matrix $G_j = \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix}$ is only used to reduce the number of values for non-zero elements of the Hessenberg matrix $\hat{H}_j$ for each iteration j. Now the matrix G_j can be computed with the equations of $det(G_j) = 1 = c_j^2 + s_j^2$ and $0 = \hat{h}_{i+1,j} = -s_i h_{i,j} + c_i h_{i+1,j}$. The last condition follows from the fact that $\hat{h}_{i+1,j}$ should be set to zero after applying matrix G_j for the current computed Hessenberg matrix $\hat{H}_j$. Then these two equations are solved with $c_j = h_{i,j}/\eta$ and $s_j = h_{i+1,j}/\eta$ with using $\eta = \sqrt{h_{i,j}^2 + h_{i+1,j}^2}$ such that a matrix G_j is obtained which is applied for the next matrix $\hat{H}_{j+1}$.

Algorithm 5 Givens algorithm for attaining an upper Hessenberg matrix [39][46][47]:

```
Input: Hessenberg matrix $\hat{H}_j$ with size $j$.
Apply old Givens rotation to $j$-th column of $\hat{H}_j$ and vector $c$, $s$ with size of $m$.
for $i = 1, 2, .., (j-1)$ do
    $\begin{bmatrix} \hat{h_{i,j}} \\ \hat{h_{i+1,j}} \end{bmatrix} = \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix} \begin{bmatrix} h_{i,j} \\ h_{i+1,j} \end{bmatrix}$
end for
▷ Compute new Givens rotation for elimination of $\hat{H}_j(j+1, j)$.
$\tau = |h_{j,j}| + |h_{j+1,j}|$
$\nu = \tau\sqrt{(h_{j,j}/\tau)^2 + (h_{j+1,j}/\tau)^2)}$
$c_j = h_{j,j}/\nu$
$s_j = h_{j+1,j}/\nu$
▷ Apply Givens rotation to $\hat{H}_j$.
$h_{j,j} = \nu$
$h_{j+1,j} = 0$
▷ Apply Givens rotation to right hand side of $z_j$.
$z_{j+1} = -s_j z_j$
$z_j = -c_j z_j$
if $||z_{j+1}||_2/||b||_2 \leq \epsilon$ then
    Converged to the desired tolerance $\epsilon$.
    return
end if
```

Algorithm 5 shows the Givens rotation which is mostly taken from the book [47]. It makes solving the equation system $||r_m||_2 = ||\hat{z}_m - \hat{R}_m y_m||_2 \approx ||Ax - b||_2$ really efficient.

5.2.10.4 The Generalized Minimal Residual (GMRES) algorithm [4][5][12]

The GMRES solver which is presented in algorithm 6 consists mainly of the orthogonalization process (line 3 - 14), a sparse matrix vector product (line 4) and the optional rank revealing decomposition with the Givens rotation (line 16) for each iteration j. The Givens rotation is shown in algorithm 5. The most compute intensive parts of the GMRES solver (see section 6.1.4) are the sparse matrix vector product and the orthogonalization process with the Gram Schmidt or Modified Gram Schmidt method shown in algorithm 3 and 4. Like in any other iterative method the initialization is really important for the solution vector x because the used solver converges faster if the initialization for vector x_0 is near to the real solution $x_\star$. Using the GMRES solver has some advantages but also some disadvantages, the most obvious problem is that because of storing the Hessenberg matrix $\hat{H}$ for solving the equation system $Ax = b$ this solver needs some additional storage place for $\hat{H}$ in the main memory in contrast to the Conjugate Gradient (CG) method which is only able to solve positive definite matrices (SPD). The GMRES solver is able to solve any kind of problem for a sparse matrix A. This solver like any other solver can be extended to a flexible solver shown in algorithm 11 which is just an application of iterative refinement explained in section 9.

Algorithm 6 GMRES without restarts [4][5][12]:

Input: Matrix A, right hand side b and initial starting vector x_0 and m for the number of iterations.
Output: Approximate solution of vector x_m for $m > 0$.

```
 1: $r_0 = b - Ax_0$                                    ...▷ Compute initial residuum vector.
 2: $\beta = ||r_0||_2$, $q_1 = r_0/\beta$                ...▷ Compute first basis vector $q_1$.
 3: for $j = 1, 2, ...$until convergence and $j < m$ do
 4:   $v_{j+1} = Aq_j$                                  ...▷ Perform matrix vector product.
 5:   ▷ Orthogonalize basis vector $q_j$ (line 6-10).
 6:   for $i = 1, 2, ...j$ do
 7:     $h_{i,j} = q_i^T v_{j+1}$
 8:     $v_{j+1} = v_{j+1} - h_{i,j} q_i$
 9:   end for
10:   $h_{j+1,j} = ||v_{j+1}||_2$
11:   if $h_{j+1,j} \approx 0$ then
12:     Solution found $x_{j-1}$.
13:     return
14:   end if
15:   $q_{j+1} = v_{j+1}/h_{j+1,j}$                     ...▷ New basis vector $q_{j+1}$.
16:   ..▷ Apply Givens rotation to $\hat{H}(1{:}j,1{:}j)$) (, optional do a rank revealing decomposition [49]).
17:   $y_j = argmin_y||\hat{H}(i : j+1, 1 : j)y - \beta e_1||_2$   ...▷ Solve least square problem.
18:   $x_j = x_0 + [q_1, q_2, \dots, q_j]y_j$             ...▷ Compute solution $x_j$.
19: end for
```

The GMRES solver can be restarted it is just possible to reuse the old solution x_m from the previous computation and to do the whole process again. It gives the possibility to restart the process from a better solution x_m computed so far. This Krylov subspace method does no contrast where to start for the solution vector x_0 which leads to the fact that the storage place for building the Hessenberg matrix is less restrictive. For speeding up this method in algorithm 6 of the standard GMRES solver line $v_{j+1} = Aq_j$ has to be replaced for preconditioning with $v_{j+1} = L(\backslash U\backslash(Aq_j))$ if ILU-preconditioning is applied [6] but also the residuum r has to be replaced with $r = U\backslash(L\backslash(b - Ax))$.

5.3 The condition number ($\kappa(A)_2$) of a matrix A [50]

The condition number ($\kappa(A)_2$) of a matrix A is the relation between the input and output error of a matrix A. High condition numbers indicate that the input error is amplified because of the entries of matrix A and the related eigenvalues λ. If matrix A has a high condition number then matrix A will amplify small disturbances like during the matrix vector product (Ax) and will be sensitive to small disturbances. In general the condition number ($\kappa(A)_2$) of a matrix A is computed with its inverse A^{-1} such that $AA^{-1} = I$ where I is the identity matrix and with the according norm ($||.||_2$):

$$\kappa(A)_2 = \|A\|_2 \|A^{-1}\|_2 \text{ [50].} \tag{71}$$

From this equation 72, it follows that this property $\kappa(A)_2$ of a matrix A with the used norm $||.||_2$ is determined if the lowest ($\lambda_1(A)$) and greatest ($\lambda_n(A)$) eigenvalue is known and can be computed with:

$$\kappa(A)_2 = \frac{||\lambda_n(A)||_2}{||\lambda_1(A)||_2} \text{ [50]} \tag{72}$$

, like in the case of a diagonal matrix (see formula 102 and formula 103). The condition number $\kappa(A)_2$ also depends on the used norm ($||.||_2$) but if another norm is used the magnitude of the computed condition number differs less against another computed condition number with another norm because all norms have nearly the same explanatory power. If a matrix A has condition number 1 then this matrix A is well conditioned and the error is not amplified. The condition number of a matrix A is also used to determine how fast a solver could converge to the solution vector x or if the problem to solve ($Ax = b$) is poor or well conditioned, a well conditioned problem converges always faster than an ill conditioned problem. Well conditioned problems always have a condition number 1 whereas ill conditioned problems have a greater condition number.

5.4 Problem description of exa scale computing [1][2][51][52][53][54]

In recent years the number of components and transistors has increased in computer systems through massive scaling the number of cores of a CPU (Central Processor Unit). The main reason to speed up programs with increasing parallelism is that at some specific point it is not possible to rise the frequency (f) and voltage (U) because of the fast increase of energy consumption (P). Another point is that the IPC (Instructions Per Cycle) of a single core has not increased in the last years. IPC is mainly the number of instructions which can be done during each compute cycle of a single compute core of a CPU. So the most speed up was achieved in the past with scaling the number of cores and transistors of a CPU. This energy requirement P for each computing core is proportional to the square of the voltage (power consumption: $P \approx CU^2 f$ [55] with C the capacity as the ability to store electrical charge, U the voltage and f as the frequency). Therefore the limit of the maximum allowable usage of energy (P) restricts the frequency (f) and as well the voltage (U) because of the possibility of damaging the used components. This problem is mainly known as the *power wall* problem [52][53] which is illustrated in figure 1. It is impossible to pass over this point and to increase the power consumption more because of damaging the used components of a CPU.

For computer systems it is impossible to increase the number of components beyond any limit without changing the reliable mode. A system which runs in higher reliability will also need a higher power consumption. Lowering the reliable mode means changing to an unreliable mode where more components are allowed to fail. A higher power consumption (P) than recommended will lead to a faster aging of the used components and will perhaps also destroy them. The *power wall* problem [52][53] mainly limits the density of waste heat per surface of each CPU core because of cooling restrictions from the used components.

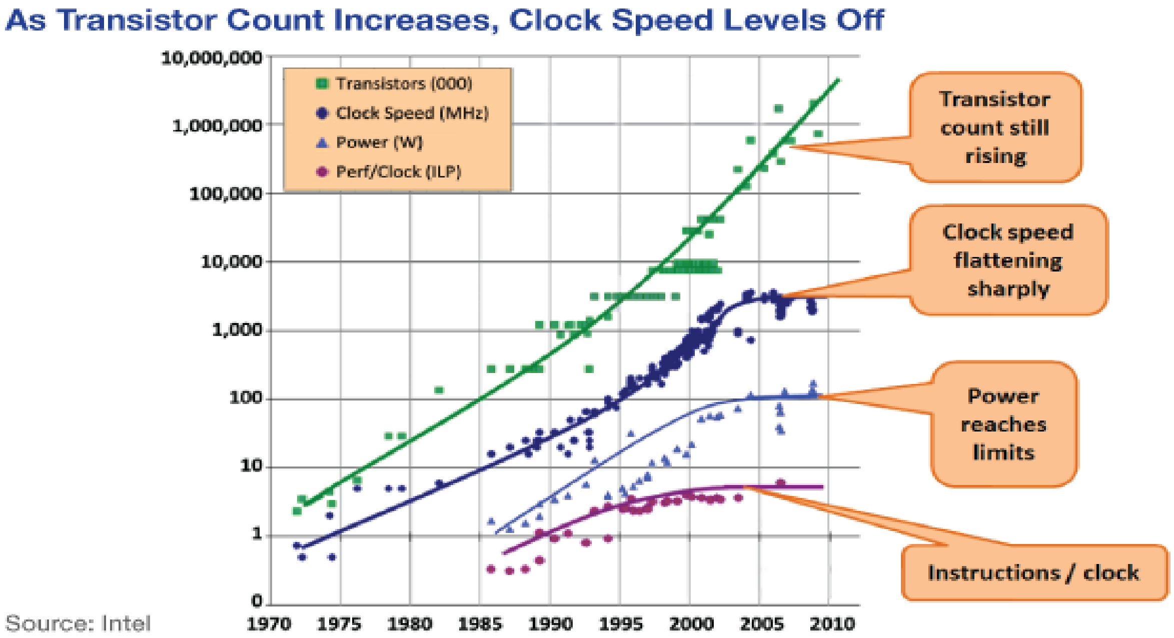


Figure 1 Illustration of the *power wall* problem [52][53] based on Intel slides. [56]

There are also other restrictions like the *memory wall* [57] besides the *power wall* [52] problem. At some point of an algorithm the most limitation for speeding up the program comes from the bandwidth limitation (e.g. bandwidth between cache and global memory, bandwidth between two network cards, . . .). This problem is mainly important for scientific applications as well as for other applications.

With increasing the level of parallelism through the number of cores and used components the chance that some of those computations fail rise. This is mainly a problem on huge computer clusters with a massive number of CPU cores beyond one million and especially more. Each single transistor has a chance to fail and as more transistors are used the chance that a fault happens rises. This probability may rise in the best case linearly with the number of the used components.

A description used to determine the reliability of a system is called Mean Time Between Failures (MTBF). This fraction describes the average time after a failure to the next occurrence of a failure. This value has more explanatory power than only taking the probability of all failures. For example a modern hard disk has a MTBF of 300000 to 1200000 hours [54] but this time refers to aggregate analysis of a large number of hard drives because it would take really too long to test only a single unit. A modern DRAM (Dynamic Random Access Memory [58]) chip for the main memory has a MTBF of about 14 to 142 years [51]. So mean MTBF refers to a large number of components rather than to a single one and can be computed in arbitrary units. MTBF is also referred to the average time between two failures if the error is correctable.

In the main memory of each computer system there is often used ECC (Error Correcting Code) memory which consumes a lot of electricity power of about 12.5 % against non-ECC memory [2] and leads to a performance lose of about 5 % up to 15 % [1][2]. ECC memory mainly detects and corrects bit-flips if a failure occurs in the memory. Lowering the energy requirements means more bit-flips will happen, these bit-flips can be corrected as well not corrected by the ECC memory. The chance to detect and correct a bit-flip is mainly related to the voltage level (U) of the used memory modules. Lowering the voltage (U) leads to a lower reliable mode of the used ECC memory.

Some applications show that only in special sections high reliability is needed where most of the work can be done in unreliability. In numeric applications the result must be checked with high reliability if the computation should be correct or not. This leads to a layered approach where data can be still exchanged between reliable and unreliable sections (see section 5.5). At the end of the computation the right result should be still achieved even with some faulty data during the computation.

This is the point where self-stabilizing systems (see section 4.1) or selective reliability (see section 6.2) play a big role just because it is not possible to detect and correct each fault of each component if the reliable mode is decreased. The property of self-stabilization describes that a system reaches a valid state or satisfies some specific rules in a finite number of steps no matter of the initial state or the invalid states between the computation. Designing new algorithms would help here to decrease these energy requirements with the property of self-stabilization such that it is still possible to achieve the same results with some faults like in the case if everything is computed with the same reliable mode. There are some nice side effects of self-stabilizing systems because it is possible to decrease the energy requirements but as well to increase the level of parallelism. There are also some other aspects of self-stabilizing systems which make designing new algorithms very interesting. It is possible to relax some components as well as the used algorithm itself but also the programming model. This relaxation must still satisfy specific constraints like the upper bound of the computed error (e.g. difference between reliable and unreliable computations, result of the computation in unreliable mode against the exact solution, . . .). The driving force using self-stabilizing algorithms is just to increase the number of components as well as to decrease the vulnerability but these algorithms also change the perspective to design new algorithms and how to exploit more parallelism.

Self-stabilization is not only important in the context of High Performance Computing (HPC) because there are also other applications where these kind of algorithms take place (e.g. sensory, communication, . . .). In figure 2 the history of flops (floating point operations per second) is shown of the fastest super computer per year taken from the source [59], to achieve further progress of more flops new techniques must be explored like the property of self-stabilization or selective reliability (see section 6.2) to overcome different problems about HPC.

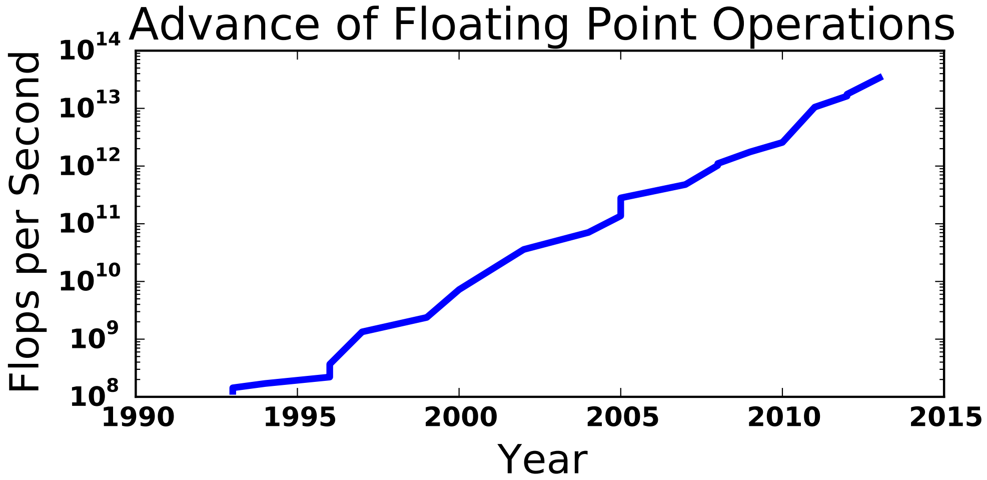


Figure 2 The history for change in flops (floating point operations per second) from 1990-2014 for the fastest super computer(s) per year. [59]

5.5 The statistical and sandbox reliability model [4][12]

There are two different models which are applied in fault tolerant iterative linear solvers, the statistical and the sandbox reliability model. Both methods differ in the number of computations and operations which leads to different aspects but also to various advantages and disadvantages.

***The statistical model* [12]:**
The statistical model is mainly based on some assumptions for faults and errors (see section 5.6). It is common in this model to do different trials for solving the equation system $Ax = b$ with an existing iterative method in unreliability when bit-flips are allowed to happen and with isolating the most obvious outliers. It also means if one of these assumptions is wrong no errors and faults can be detected. Just for example in the statistical methods finding some outliers is based on the predefined median. If those data and conditions for computing it are imprecise it is hard to exclude some wrong data. Another problem is if a single outlier is the real solution, no solution can be found because it is excluded. It also takes additional computations for each further execution of the used solver to determine the most obvious solution.

The sandbox reliability model [4][12]:
In contrast to the statistical model, the sandbox reliability model is related to IT-security systems where some parts of a computer program are isolated to run in a secured area from the rest of the system. In this section the isolated part is allowed to do anything but it is not possible to harm the host system anymore. Sand-boxing gives a layered approach where parts are isolated from the rest of the system but with still the property to exchange information between these two sections just for example from reliable to unreliable mode and vice versa. This model is not totally new but it can be extended to iterative solvers. Most of the workload and computations in this model should be done in unreliability. In figure 3 the sandbox reliability model in the context of (fault tolerant) iterative linear solvers is shown.

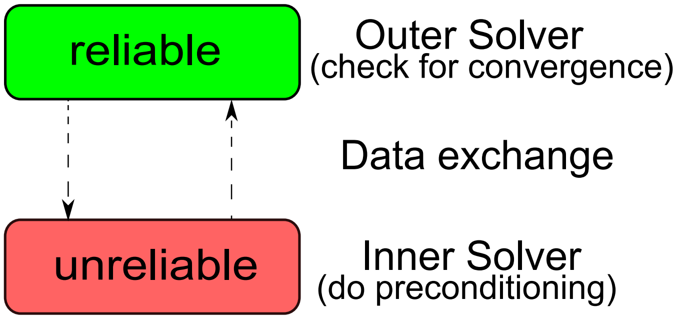


Figure 3 The sandbox reliability model.

Some properties about the sandbox reliability model:

- In the sandbox reliability model for iterative methods there are two solvers one outer but as well one inner solver.
- There are some numerical methods where only a few computations have to be done with high reliability to ensure correctness of the final computed result.
- It leads to a layered approach where it is allowed that information is exchanged, if most of the time is spend in the inner solver then the total costs can be reduced.

5.6 Classification of faults and failures [4][12][60]

There can be hard and soft faults during some computations of a program, it mainly depends on the abstraction level, the classification can be found below in figure 4 which gives a general overview.

- **Hard faults**: *"Cause program interruption and are outside the scope of what the executable program can directly detect. These faults can result from hardware failure or from data integrity faults that lead to an incorrect execution path."* [12]
- **Soft faults**: *"Do not cause immediate program interruption and are detectable via introspection by user code. Soft faults occur as "bit-flips" such as incorrect floating point or integer data, or perhaps incorrect address values that still point to valid user data space. Although it is difficult to detect all soft faults, some modest amount of introspection can be very effective at dramatically reducing their impact."* [12]

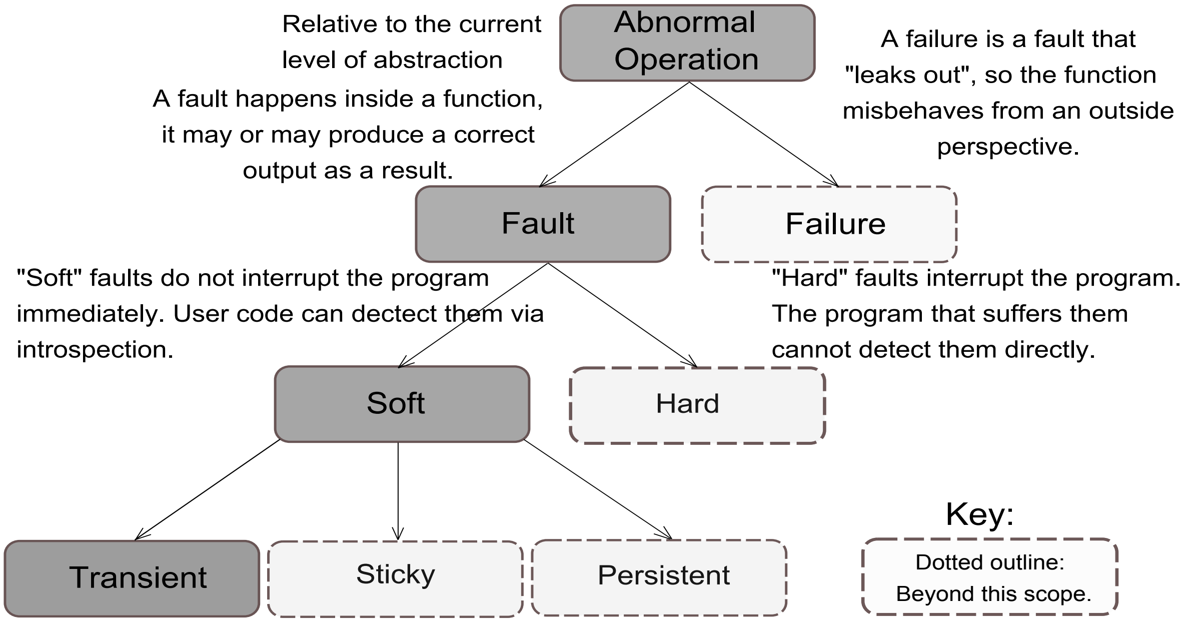

Relative to the current
level of abstraction
A fault happens inside a function,
it may or may produce a correct
output as a result.
Abnormal Operation
A failure is a fault that
"leaks out", so the function
misbehaves from an outside
perspective.
Fault
Failure
"Soft" faults do not interrupt the program
immediately. User code can dectect them via
introspection.
"Hard" faults interrupt the program.
The program that suffers them
cannot detect them directly.
Soft
Hard
Transient
Sticky
Persistent
Key:
Dotted outline:
Beyond this scope.

Figure 4 Taxonomy of faults and failures. [4][12]

There are different kinds of faults which can happen during processing a program. How this classification is done mainly depends on the level of abstraction. The main distinction between faults and failures is that failures lead to incorrect outputs outside a function whereas faults only misbehave inside a function and may produce incorrect data outside a function.

A fault can be also classified in a soft and hard fault. Soft faults are sometimes also called Silent Data Corruptions (SDCs), these kinds of faults will not interrupt the program but will perhaps lead to incorrect output results. If one of these faults influences the output of a program or function then this fault becomes a failure, just for example the whole program crashes. Hard faults will totally crash the entire program but it is possible to turn hard faults in soft faults like with checkpointing such that the program is allowed to proceed. A further discrimination between faults and failures can be found in figure 4 where also soft faults are classified in different kinds of faults.

A software program may totally crashes but if it is possible to continue with faulty data the failure becomes a soft fault. If it is possible to proceed with those faulty data the outcome of the solution will be perhaps valid or invalid. In the case of an invalid solution the soft fault becomes a failure since the perturbation is not observed and corrected. Soft faults can be also classified in three subgroups (persistent, sticky and transient). The occurrence of Silent Data Corruptions (SDCs) must be decreased and in the best case totally negated.

The whole description of these subgroups can be found in the table below. The term persistent means that those faulty data are not observed and corrected whereas a sticky fault indicates that some data are only faulty for some restricted time. Transient faults occur in a really short time but it is possible to become persistent because of exchanging some data in the main memory with copying data from the cache (fast memory for the CPU) back to the main memory. If this is the case without detecting the change it becomes impossible to correct this fault.

Sub-classification of soft faults [12]:

- **Persistent fault:** *"The incorrect bit pattern will not change as execution proceeds. Example: The primary source of a data value (and any subsequent copies) are incorrect, so there is no ability to restore correct state."* [12]
- **Sticky fault:** *"The incorrect bit pattern can be corrected by direct action. Example: A backup source for the data exists and can be used to restore correct state."* [12]
- **Transient fault:** *"The incorrect pattern occurs temporarily. Example: Data in a cache is incorrect, but the correct value is still present in main memory and the cache value is flushed."* [12]

In figure 5 a further classification can be found according to the impact of reading a faulty bit. It is mainly related to the ability of detecting and correcting a fault in a microprocessor system. A true DUE is mainly a fault where a system has error detection but this fault is not correctable in comparison to false DUE. A soft error (SDC) mainly affects the outcome if the system has no error detection. Bit-flips in the main memory can be also undetected with no effect on the result.

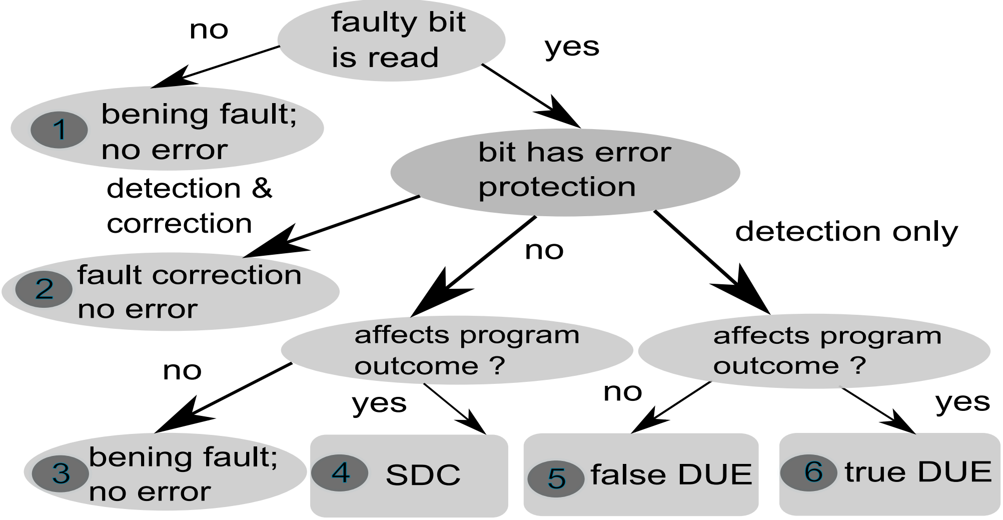


Figure 5 Classification of the possible outcomes of reading a faulty bit in a microprocessor system. SDC = silent data corruption. DUE = detected unrecoverable error. [60]

5.7 Bitflips in practice [8][9][58][61][62][63][64]

Soft errors are mainly caused by the environment from cosmic rays or electrical noise but in general through the huge energy spikes between two transistors which change the entire state of a transistor and lead to the problem of flipping some bits because of the poor isolation between two transistors. The purpose must be in computer systems to decrease the effect of bit-flips without too much effort of checking and correcting all soft errors (see figure 4) in the DRAM (Dynamic Random Access Memory [58]) because of increasing the energy requirements. DRAM is often specified as RAM or main memory of a computer system, the content of this memory type must be refreshed constantly.

The property of detecting and correcting all soft errors in the DRAM leads to high reliable sections, this additional security costs a lot of money because of the extra energy usage even in desktop computers reliable becomes an important factor. The aim must be to decrease the energy requirements with lowering the voltage level of the DRAM [58] which is only a part of the whole energy consumption of a computer system.

Nevertheless each contribution for lowering the energy requirements is needed to achieve exa scale computing which are 10^{18} floating point operations per second (on a compute cluster). It is also impossible to detect and correct each single bit-flip if the number of processors grows. There are more and more transistors on less space mainly through better manufacturing processes that's why the number of relative bit-flips increases because each single transistor affects the other ones surrounding it. In figure 6 a possible development of bit-flips in the DRAM is shown in the future but also for the past and present, the expectation is that the number of bit-flips will increase.

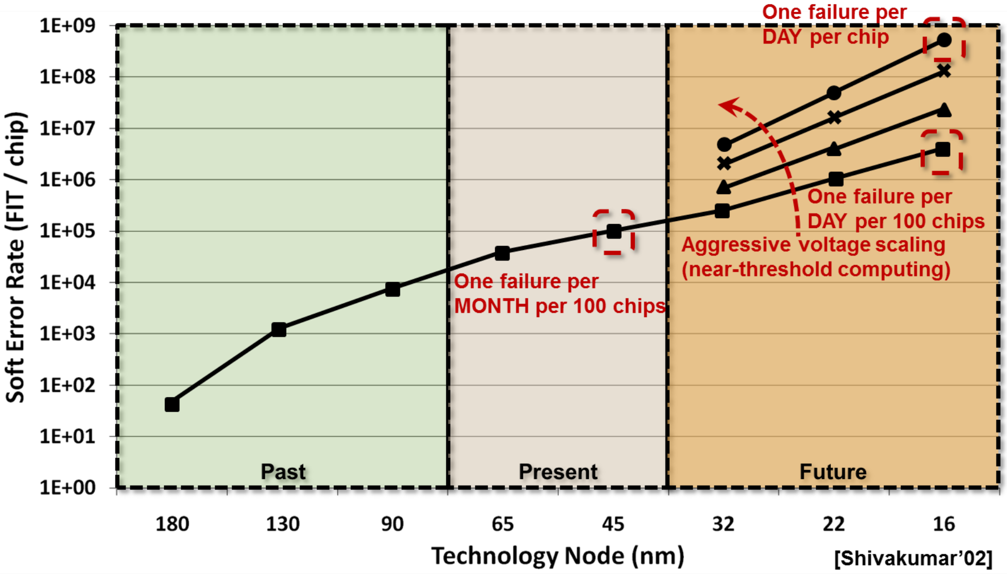
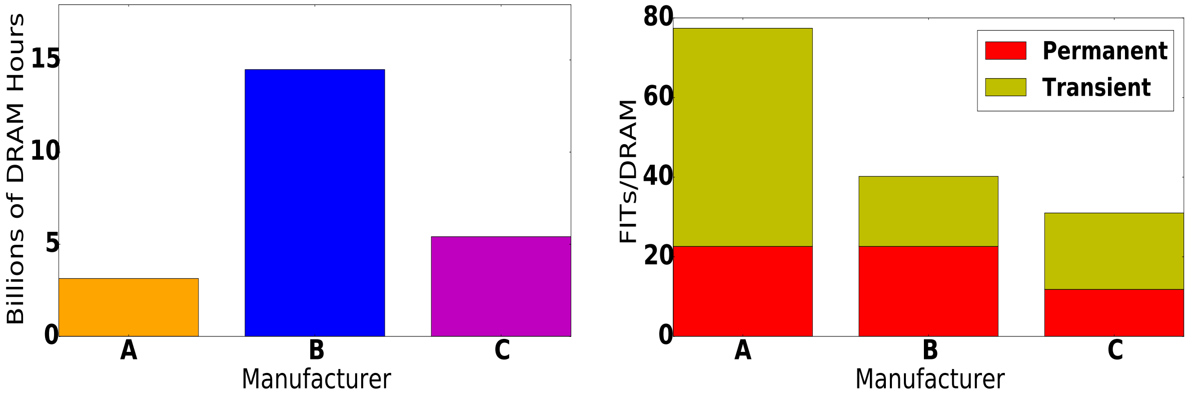


Figure 6 Transistor size in the past, present and future with fault rates in FIT. [8]

The number of bit-flips has increased by a better manufacturing process from one generation to the next generation of the DRAM in the past but as well in the present shown in figure 6. It is expected that the soft error rate FIT (Failure In Time [65]) will also increase in the future through better manufacturing processes because there are more and more transistors on less space where each single transistor influences the others. This rate is usually given in 10^9 hours (FITh) which means that it needs 10^9 hours from one single upset to the next one for flipping a bit but can be also defined for seconds and other time units. If this rate is used for seconds then this fault rate (FITs) is computed for 10^9 seconds but in the case of hours (FITh) are 10^9 hours used. Flipping a bit is mainly caused by the fact that more and more transistors have less space on the same die (space of the CPU - Central Processor Unit, integrated-circuit, . . .) because the density of transistors increases.

Reliability of a computer system is also an important factor because each single bit-flip which is not observed and not corrected in the used memory could lead to an increase of additional costs. This is especially the case for large server clusters like for Google and Microsoft where each loss of a system might cause bigger problems because of poor reliability. Therefore each failure of a server will rise the costs. Some case studies about bit-flips in the DRAM can be found in [64]. This study was done for three different vendors because of the need of anonymization those names are not revealed. In this statistic the number of working hours is observed and as well the number of bit-flips. All three vendors A, B and C are on the same server namely the Cielo cluster a supercomputer located in Los Alamo. This cluster contains approximately 8500 compute nodes where each node has about 32GB DDR3 RAM of memory capacity.

On this supercomputer each vendor has about the same size of DRAM capacity, just about a third of the total DRAM capacity. During the measurement interval this DRAM was heavily utilized which is shown in figure 7a whereas the fault rate of each vendor is shown in figure 7b with FITs (1 failure/billion seconds). The number of operation hours of the used DRAM is high enough for observing the failure rates with enough accuracy on this compute cluster (Cielo supercomputer).



(a) Operational hours per DRAM vendor

(b) Fault rates per DRAM vendor

Figure 7 Operational hours and fault rates per DRAM vendor. [64]

The definition of permanent and transient faults can be found in section 5.6 where most of the error types are explained. Transient faults are bit-flips in the main memory which occur for a short period of time whereas permanent faults are undetected and not correct in the main memory. In figure 7b two interesting observations can be done. The first one is that more than 50 percent of all faults are transient for all three vendors.

The next interesting observation which follows from the first one is that the vendor which has the highest rate of transient faults also has the highest number of permanent faults which indicates that transient faults also lead to persistent faults. It is not clear why in figure 7b manufacture A and B differ maybe because of the usage of different protection methods.

Some other statistics can be found in [64] which are done on the Jaguar supercomputer [66] this server has 18688 compute nodes and for each node there are 16 GByte DDR2 memory. A fault rate of 0.057 - 0.071 FITh/Mbit (Mbit = 10^6 bits) can be found on this system. This fault rate (FITh) per one million bits (Mbit) is really less. The fraction of 1 FITh/Mbit can be transformed from one upset (change of a single bit) every 10^9 hours per 10^6 bits to 10^{-15} upsets/bit-hour (1 FITh/Mbit = 1 upset/10^9 hours/ 10^6 bits = 10^{-15} upsets/bit-hour) which are 5.7 - 7.1 $\times 10^{-17}$ upsets/bit-hour in the case of 0.057 - 0.071 FITh/Mbit. This rate also shows that the number of bit-flips is really less even on systems with a lot of memory capacity shown in [64]. This case study was done from the year of 2009 until the late of 2010.

The next statistics about bit-flips in the DRAM can be found in [61]. This study was done on several Google servers and with six different platforms. In this work a platform is defined by the motherboard and memory generation. It tries to answer the questions how do CPU utilization, temperature and memory allocation influence the number of bit-flips in the DRAM. This study was done between January 2006 to June 2008 over 2.5 half years. In this paper the meaning of correctable (CE) and uncorrectable errors (UE) are used, both are related to the number of bit-flips. Uncorrectable errors cannot be detected and corrected with the ECC (Error Correcting Code) memory.

In figure 8 the effect of aging of the DRAM is shown there is a strong(er) relation between the age and the number of bit-flips in the DRAM. In [61] there is also shown that there is a slight correlation between temperature, CPU and memory utilization, and for the number of bit-flips in the DRAM. Furthermore the error rate is more influenced by the utilization and memory allocation of the DRAM than on other properties. The temperature has only some slight effect on the error rate of the DRAM but the CPU and memory utilization have more influence on the number of bit-flips.

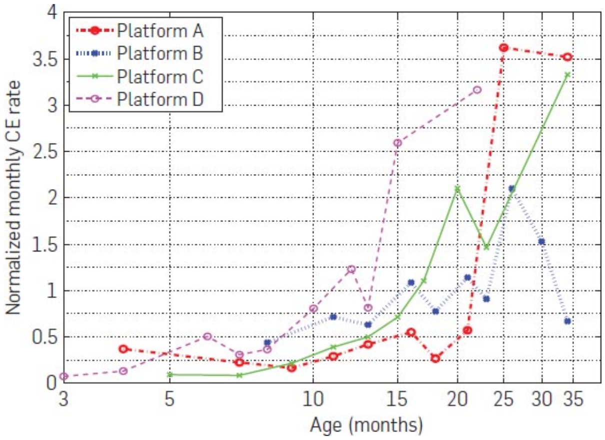


Figure 8 Effect of aging of the DRAM. [61]

The next case study about bit-flips in the main memory can be found in [9]. This study is mainly about how does the technology trend help to decrease or will perhaps increase the number of bit-flips in the main memory. Another question is where most bit-flips come from and what is about the average error rate (FITh) for different types of memory devices. Table 2 shows that the number of bit-flips is quite high with SRAM (Static Random Access Memory) and it is significant higher than using DRAM (Dynamic Random Access Memory). SRAM exhibits data remanence in contrast to DRAM where the content has to be dynamic refreshed. DRAM is mainly used if huge capacities of memory are needed like for the main memory of a computer system or for the memory of a graphic card (GPU) whereas SRAM is used if memory with higher frequencies (higher access speed and lower latencies) but with lower capacity is needed which is mainly applied for the caches (fastest memory of a CPU and GPU).

In the case of the DRAM the number of bit-flips will decrease if a better manufacturing process is used but through increasing the number of cell arrays (number of bits for storing the content) the number of bit-flips will increase because there are more and more transistors on less space as in the case of the MLC Flash (Multi Level Cell) which is used for example in USB devices. In contrast to DRAM the FITh rate of SRAM will be always constant. This is mainly a pitfall between SRAM and DRAM but the FITh/bit rate of SRAM is quite higher than using DRAM.

Device Type	Soft Error Rate	Comments
SRAM	10^{-4} to 10^{-2} FITh/bit	"Flat trend with design rule. Single bit ECC protection and interleave." [9]
DRAM - Cell	10^{-10} to 10^{-5} FITh/bit	"Fixed cell capacitance causes downward trend as design rule shrinks. ECC protection and interleave." [9]
DRAM - Logic	0.1 to 10 FITh/chip	"Dominates cell upset in newer technologies on a bit error rate basis. Requires multi-bit ECC." [9]
MLC Flash	10^{-8} to $10-5$ FITh/bit	"Trending up as process technology shrinks. Embedded ECC protection." [9]
LOGIC	$\approx$ 10x less than SRAM	"Will probably trend flat as process technology shrinks. Difficult to protect." [9]

Table 2 Summary of high energy neutron soft error rates. [9]

The author of [9] also mentioned that most of the bit-flips will come in the future from the logic for accessing and updating the content of the memory device rather than from the cell arrays themselves which store the real content. This content can be easily secured with ECC (Error Correcting Code) memory from flipping a bit. This protection method will also have a lower overhead if the space of the content (number of bits for storing the real data) will increase in the future, therefore only a few bits for securing a lot of bits are used with ECC memory.

The main focus of [67] is how reliability of a computer system can be defined because there is a misinterpretation until now. The reliability of a computer system does not entirely depend on the number of bit-flips but on the affect which is caused by them. There are also some comparisons between different protection methods for the memory, it comes out that nearly all are insufficient for future computer systems like SEC-DEC which can only correct one bit-flip but can detect two of it. Furthermore DRAM is also more problematic than SRAM because SRAM causes less uncorrectable errors, therefore DRAM faults will be a major concern in the future.

Sometimes security is also related to bit-flips there is a common test of how stable a system against bit-flips in the DRAM is, this test is nothing else as a security lack. This security invasion is often called rowhammer attack which was first figured out by Yoongu Kim and his colleges in [63]. They observed that if bits from the same address in the DRAM are overwritten multiple times within a short period of time than also bits which are near the same address can be affected too. Yoongu Kim also figured out that the number of caused bit-flips mainly depends on the vendor system for this security breach also called rowhammer attack. He discovered that the effect of changing the state of a transistor rises with lowering the distance between two transistors because of a higher electric disturbance. The according assembler code for the rowhammer attack (see algorithm 8) is shown in table 3, where also a dummy code is used which has no influence on the number of bit-flips (see algorithm 8).

Algorithm 7 Rowhammer Attack:

```
1: code1a:
2: mov (X), % Read from address X
3: mov (Y), % Read from address Y
4: clflush (X) %Flush cache for address X
5: clflush (Y) %Flush cache for address Y
6: mfence %Ensure global emory integrity
7: jmp code1a
```

Algorithm 8 Dummy Code:

```
1: code1b:
2: mov (X),
3: clflush (X)
4: mfence
5: jmp code1b
```

Table 3 Assembler code for testing of bit-flips. [63]

In this assembler code for testing of bit-flips with the rowhammer attack in algorithm 7 there is no external influence of the content for two different accessed addresses in the DRAM but it is really fast exchanged within the while loop (from line 1 to 7). The aim is just to affect other bits and to change the states which are near those two addresses (X and Y). In line 2 and 3 of algorithm 7 (code1a) two different contents are read from two different addresses (X and Y) after reading the contents they are exchanged in line 4 and 5 of algorithm 7, there is no external influence only the contents of two different accessed addresses are exchanged.

The rowhammer attack (algorithm 7) can also be used as a benchmark because it shows in a good way how often in a computer system bit-flips occur in comparison to other vendor systems. In algorithm 8 there is also no external influence but it is mainly used to show that reading (line 2) and writing (line 3) of the same address X has no influence on the number of bit-flips.

Bit-Flip	Sandy Bridge	Ivy Bridge	Haswell	Piledriver
’0’ → ’1’	7992	10273	11404	47
’1’ → ’0’	8125	10449	11467	12

Table 4 Number of bit-flips induced by disturbance on a 2GB module. [63]

Two vendors are compared in table 4 where the company Intel (with CPUs: Sandy Bridge, Ivy Bridge, Haswell) has at most bit-flips in comparison to the company AMD (with CPU: Piledriver). During this experiment the DRAM was initialized with zeros (’0’) and ones (’1’) and afterwards both codes of table 3 were executed. The code1b (algorithm 8) doesn’t produce so much disturbances as code1a but the rowhammer attack (algorithm 7) leads to a certain number of bit-flips shown in table 4. If a bit changes its value from ’0’ $\rightarrow' 1'$ or ’1’ → ’0’ during the execution of code1a and code1b then this is a bit-flip. The while loop in code1a and code1b was applied one million times for both initializations (’0’, ’1’).

This attack or test with code1a can be eliminated with ECC memory (Error Correcting Code) which costs more energy and money than using standard DRAM. There was a factor of 250 for the number of bit-flips between these two mentioned vendors where the company Intel has at most bit-flips. In this study [63] it comes out that bit-flips are related to the memory access speed of the CPU. So in all bit-flips don’t only depend on the used technology and age but also on the used vendor and memory access speed.

6 Flexible and Fault Tolerant GMRES (F and FT-solvers)

6.1 Flexible solvers (F-solvers)

6.1.1 Newton's fixed point method and iterative refinement [68]

Flexible methods are mainly related to the Newton's fixed point method because solving a linear system $Ax = b$ can be reformulated to search for a vector x where the function $f(x) = b - Ax$ should be minimized. This function $f(x)$ is often called the residuum r. Newton's fixed point method gives the possibility to search for this vector x such that function $f(x)$ should be minimized where a solution is found for $Ax = b$ iteratively. This method is a special case of the fixed point iteration. In arbitrary functions or polynomials a root can be found, the main restriction for solving $Ax = b$ is there must be only one root for $f(x)$. This method is now applied iteratively with an arbitrary chosen start vector x_0 for the solution vector x which should minimize the function $f(x)$:

$$x_{j+1} = x_j - (\nabla f(x))^{-1} f(x_j). \tag{73}$$

So in the case of linear equation systems the function $f(x)$ is set to $f(x) = b - Ax$ and with the according derivation of $\nabla f(x) = A$. Therefore the residual r_j is computed with $r_j = b - Ax_j$ for each iteration j, so from the fixed point iteration the following relationship is observed:

$$x_{j+1} = x_j - (\nabla f(x))^{-1} f(x_j) \tag{74}$$

$$x_{j+1} = x_j - A^{-1}(b - Ax_j) \tag{75}$$

$$x_{j+1} = x_j + A^{-1} r_j. \tag{76}$$

From the last equation 76 by setting $Ad_j = r_j$ or $d_j = A^{-1} r_j$ for the update vector d_j of each iteration j and using the initial starting vector x_0 an iterative algorithm can be applied. This algorithm is often called defect error correction because the residuum r should be minimized for each iteration step j. The defect error correction method is shown in algorithm 9 for the given matrix A, right hand side b and ϵ as the desired accuracy for solving $Ax = b$.

Algorithm 9 Iterative refinement with defect error correction [68]:

```
Input: Matrix $A$, right hand side $b$ and tolerance $\epsilon$ for the relative residuum.
1: $x_0 = zeros()$                       ▷ Initial guess for vector $x$ (double precision).
2: $r_0 = b - Ax_0$                       ▷ Compute starting residuum (double precision).
3: while $||r_j||_2/||r_0||_2 > \epsilon$ do       ▷ Check for convergence (double precision).
4:     $r_j = b - Ax_j$                   ▷ Compute new residuum (double precision).
5:     Solve $Md_j = r_j$ for $d_j$         ▷ Apply preconditioning for matrix $M$ (single precision).
6:     $x_{j+1} = x_j + d_j$               ▷ Do the update for vector $x$ (double precision).
7: end while
```

Defect error correction which is shown in algorithm 9 allows different precision levels like single or double precision. The update in line 6 but as well checking for convergence in line 3 for the solution vector x_{j+1} must be done in higher precision. All other computations can be applied with lower precision like single precision. In the standard case preconditioning or solving $Md_j = r_j$ is done in line 5 for a matrix M such that $M \approx A$ to ensure fast convergence. Matrix A can be decomposed with the ILU-factorization [6] which can be also done for matrix M. The main contrast between right preconditioning and flexible preconditioning is that in the case of right preconditioning the problem $Md_j = r_j$ is solved always with the same number of iterations for the inner solver but in flexible methods there are various iterations to achieve a specific accuracy.

6.1.2 The Flexible Conjugate Gradient (F-CG) method [15][32]

The Flexible Conjugate Gradient method (F-CG) is mainly based on the CG solver (see section 5.2.9.2). In this method the main aim of the inner solver is to improve the accuracy of the residuum (search direction) for vector r_j in line 4 of algorithm 10 whereas the outer solver has to check for convergence in line 3. The inner solver has to solve the problem $q_j = M_j r_j$ for vector r_j (r_j is the current residuum) where matrix M can be a different matrix from A which should be approximately the same like matrix A such that $M \approx A$. In this case ϵ_2 is the accuracy of the inner solver for solving the problem $q_j = M_j r_j$. The inner solver does always a constant work in line 4 of algorithm 10, preconditioning for the outer solver is mainly done in this line. Improving the accuracy for vector r_j gives some pros and cons. One obvious advantage is if the inner solver achieves a high enough accuracy for the solution vector r_j than most of the work can be done in the inner solver rather than in the outer solver.

In contrast if the accuracy for vector r_j (search direction) is too low of the inner solver then the F-CG will converge slowly to the desired solution of x and the outer solver will need more iterations. Furthermore in some scenarios applying any decomposition for matrix A to solve instead the problem for matrix M in the inner solver is preferable with the property that $A \approx M$. This could speed up to solve the problem $Ax = b$ for the whole solver. The achieved accuracy of the inner solver is just crucial for the outer solver of the F-CG but also in general for other flexible methods. The inner solver of the Flexible CG method should mainly improve the accuracy of the current residuum r_j which should also lead to a higher stability against the standard CG solver. The F-CG is shown in algorithm 10 where in line 4 the inner solver is applied.

Algorithm 10 The Flexible Conjugate Gradient (F-CG) algorithm for solving a symmetric positive definite system (SPD) [15][32]:

Input: Matrix A, right hand side b and initial starting vector x_0 and ϵ as the accuracy for solving $Ax = b$ and m_1 for the number of outer iterations and m_2 for the iterations of the inner solver.

Output: Approximate solution of vector x_j for $j > 0$.

```
1:  $j = 0$, $r_0 = b - Ax_0$, $x_j = x_0$                ▷ Compute starting residuum $r_0$.
2:  $p_0 = r_0$, $||r_j||_2^2 = ||r_0||_2^2 = r_0^T r_0$
3:  while $||r_j||_2/||r_0||_2 > \epsilon$ and $j < m_1$ do    ▷ Check for convergence.
4:      Solve $q_j = M_j r_j$ for $r_j$        ▷ Do preconditioning like $r_j = CG(M_j, q_j, r_{j-1}, m_2, \epsilon_2)$.
5:      $q_j = Ar_j$                             ▷ Perform matrix vector product.
6:      $\alpha_j = ||r_j||_2^2/(p_j^T q_j)$     ▷ Compute new step size $\alpha_j$.
7:      $x_{j+1} = x_j + \alpha_j r_j$           ▷ Do the according step for current solution $x_j$.
8:      $r_{j+1} = r_j - \alpha_j q_j$           ▷ Compute a new residuum for $x_j$.
9:      $\beta = ||r_{j+1}||_2^2/||r_j||_2^2$    ▷ Compute a new step size for search direction $p_j$.
10:     $p_{j+1} = r_{j+1} + \beta p_j$          ▷ New search direction for $p_{j+1}$.
11:     $j = j + 1$
12: end while
13: return $x_j$
```

This algorithm 10 for the F-CG is mainly based on the CG solver (see section 5.2.9.2) which is shown in algorithm 2. It does always the same work for each iteration in line 4. Solving different matrices will lead to different behaviors so in every case different parameters have to be used which will be an additional effort. Finding some good parameters for this flexible method is not just easy as expected because there is restricted stuff known about flexible methods. Hence the F-CG can solve the problem in the inner solver if $M = A$ because it minimizes the residuum directly.

6.1.3 The Flexible Generalized Minimal Residual (F-GMRES) method [4][12]

The Flexible GMRES (F-GMRES) is based on the standard GMRES algorithm (see section 5.2.10). In algorithm 11 there are two GMRES solvers used, one which does the flexible (right) preconditioning in line 4 which can be also found in the defect error correction method (see section 9). Solving $w_j = GMRES(M_j, q_j, w_0)$ for any vector w_j (search direction) is the same like solving $x = GMRES(A, b, x_0)$ for any vector x and matrix A with the right hand side b in the case of the standard GMRES solver where the vector x solves the problem $Ax = b$ with accuracy ϵ. The vector w_0 is just the initial solution of w_j for solving $M_j w_j = q_j$. In this case ϵ_2 is the accuracy of the inner solver for solving the problem $q_j = M_j w_j$. The aim of the inner solver in algorithm 11 and line 4 is just to do the main work with preconditioning for the outer solver such that the outer solver does less iterations to solve the problem $Ax = b$ in comparison to the standard GMRES solver in algorithm 6. So mainly this approach uses two times the GMRES solver from algorithm 6 as one for the inner solver (line 4) and one for the outer solver in line 1 to 24. Like in the case of the GMRES

Algorithm 11 The Flexible GMRES (F-GMRES) algorithm without restarts [4][12]:

Input: Matrix A, right hand side b and initial starting vector x_0 and ϵ as the accuracy for solving $Ax = b$ and m_1 for the number of outer iterations and m_2 for the iterations of the inner solver.

Output: Approximate solution of vector x_{m_1} for $m_1 > 0$.

```
 1: $r_0 = b - Ax_0$                                   ...▷ Compute initial residuum vector.
 2: $\beta = ||r_0||_2$, $q_1 = r_0/\beta$             ...▷ Compute first basis vector $q_1$.
 3: for $j = 1, 2, ...$until convergence and $j < m_1$ do
 4:   Solve $q_j = M_j w_j$ for $w_j$                  ...▷ Do preconditioning like $w_j =$
      $GMRES(M_j, q_j, w_0, m_2, \epsilon_2, \dots)$.       (In this master work $M_j \approx M \approx A$.)
 5:   $v_{j+1} = Aw_j$ ($v_{j+1} = AM_j^{-1}q_j$)      ...▷ Perform matrix vector product.
 6:   ▷ Orthogonalize basis vector $q_j$ (line 7-11).
 7:   for $i = 1, 2, ...j$ do
 8:     $h_{i,j} = q_i^T v_{j+1}$
 9:     $v_{j+1} = v_{j+1} - h_{i,j} q_i$
10:   end for
11:   $h_{j+1,j} = ||v_{j+1}||_2$
12:   ..▷ Do rank revealing decomposition [49] (, apply Givens rotation to $\hat{H}(1{:}j,1{:}j)$).
13:   if $\hat{H}(j+1, j) < \epsilon_{Machine}$ then   ...▷ Stopped if machine precision is
      achieved because $\hat{H}(j+1, j) \approx 0$ so no further residual reduction.
14:     if $\hat{H}(1 : j, 1 : j)$ no full rank then
15:       Did not converged. Try recovery strategies.
16:     else
17:       Solution is $x_{j-1}$.
18:     end if
19:   end if
20:   $q_{j+1} = v_{j+1}/h_{j+1,j}$                    ...▷ New basis vector $q_{j+1}$.
21:   $y_j = argmin_y ||\hat{H}(i : j+1, 1 : j)y - \beta e_1||_2$   ...▷ Solve least square problem.
22:   $x_j = x_0 + [q_1, q_2, \dots, q_j]y_j$          ...▷ Compute solution $x_j$.
23: end for
24: Solution found $x_{j-1}$.
```

where the vector x can be reused for the next computation, the current computed vector w_j gives the possibility to restart the inner solver if the desired accuracy ϵ_2 is not reached. Therefore the F-GMRES overcomes some memory restrictions in comparison to the standard GMRES solver.

6.1.4 Determining the number of operations for (F-)GMRES and (F-)CG

In this section the workload of the CG and GMRES solver is determined but also for the flexible methods. In general the number of operations in floating point operations or also called flops is estimated for the main computations and not especially for the initialization. A flop is counted for example if for two values an addition ($a + b$) is done then it is a flop but if a and b are vectors with length n then there are n flops. Most operations are done in the main loop like for the GMRES solver in algorithm 6 between line 3 and 19 and for the CG in algorithm 2 between line 3 and 11.

Flops of the CG solver (see algorithm 2***) at iteration*** m ***and with size*** n ***of matrix*** A:
Sparse matrix vector product (SpMV, line 4): $2mnnz(A)$ flops for all non-zero elements ($nnz(A)$) of A at iteration m so there are about $2nnz(A)$ multiplications and additions for each iteration of m.
Orthogonalization (line 6 to line 10):
Line 5: n multiplications and $n - 1$ additions with 1 division.
Line 6: n multiplications and n additions.
Line 7: n multiplications and n additions.
Line 8: n multiplications and $n - 1$ additions with 1 division.
Line 9: n multiplications and n additions.
So in total there are $8n + 2(n - 1) + 2 \approx 10n$ vector operations at all for a matrix A with size n.

Total flops ($Flops_{CG}$***) of the CG method at iteration*** m***:*** $Flops_{CG}(m{,}n{,}A) = (10n$ and $2nnz(A))m$ operations for m iterations, additionally for the initialization some operations have to be done like $2nnz(A)$ and $3n$ vector operations for computing the residuum $||r||_2$ with n as the size of matrix A.

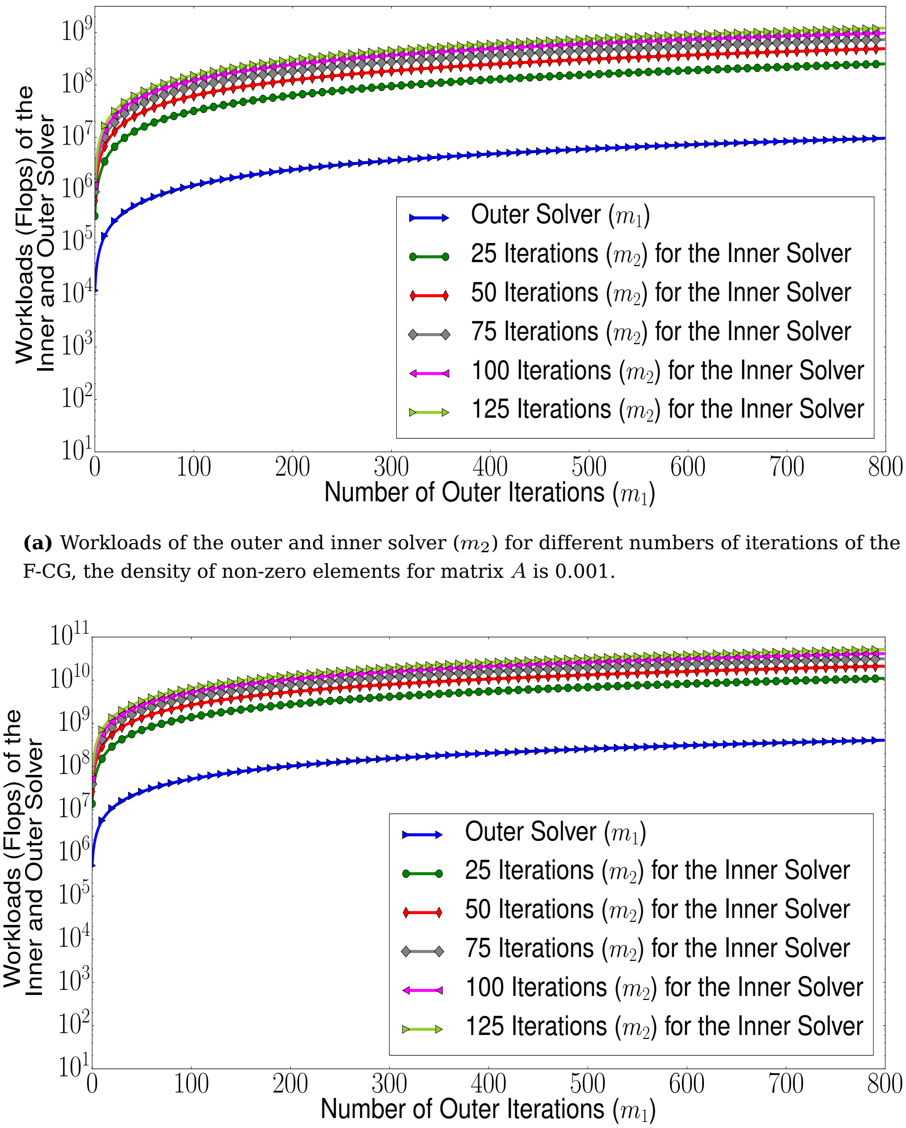
Total flops ($Flops_{F-CG}$***) of the F-CG solver:***
$Flops_{F-CG}(m_1, m_2, n, A) = m_1\ Flops_{CG}(m_2, n, A) + Flops_{CG}(m_1, n, A)$ where m_1 is the number of iterations for the outer solver and m_2 as the number of iterations for the inner solver of the F-CG.

Flops of the GMRES solver (see algorithm 6) at iteration m ***and with size*** n ***of matrix*** A***:***
Sparse matrix vector product (SpMV, line 4): In general there is for each non-zero element ($nnz(A)$) of matrix A a multiplication and addition therefore there are $2mnnz(A)$ operations for m iterations. A value of two (2) is for general sparse matrices, for diagonal matrices a value of one (1) is used.
Orthogonalizing with MGS (line 5 - 10): $\approx 2m^2n$ flops [48][69].
Givens rotation for a upper Hessenberg matrix (line 16): $3m^2$ flops (general case) [6][69][46].
Backward substitution of the decomposition for matrix $\hat{H}$ *(line 17):* m^2 flops [5][47].
Update for vector x *(line 18):* $2mn$ flops in total [5][47].
In the case of a symmetric problem (see figure 105) there are $3m$ flops for the Givens rotations [46].

Total flops ($Flops_{GMRES}$***) of the GMRES method at iteration*** m ***and for size*** n ***of matrix*** A***:***
$Flops_{GMRES}(m, n, A) = 2mnnz(A) + 2m^2n + 3m^2 + m^2 + 2mn$ flops in total at iteration m and with size n of matrix $A \in R^{n\times n}$. The number of flops is estimated for a two stage method (MGS+Givens rotation) those are counted for the most efficient version (algorithmic) of the GMRES algorithm.

Total flops ($Flops_{F-GMRES}$***) of the Flexible GMRES method at iteration*** m_1***:***
$Flops_{F-GMRES}(m_1, m_2, n, A) = m_1\ Flops_{GMRES}(m_2, n, A) + Flops_{GMRES}(m_1, n, A)$ where m_1 is the number of iterations for the outer solver and with m_2 for the number of inner iterations of the F-GMRES method. This is the case without initialization for the inner solver of the F-GMRES but there would be additionally (m_1+1) SpMV products and $3mn$ vector operations for computing $||r||_2$.

6.1.5 Workloads of the outer and inner solver of F-CG for various matrix densities



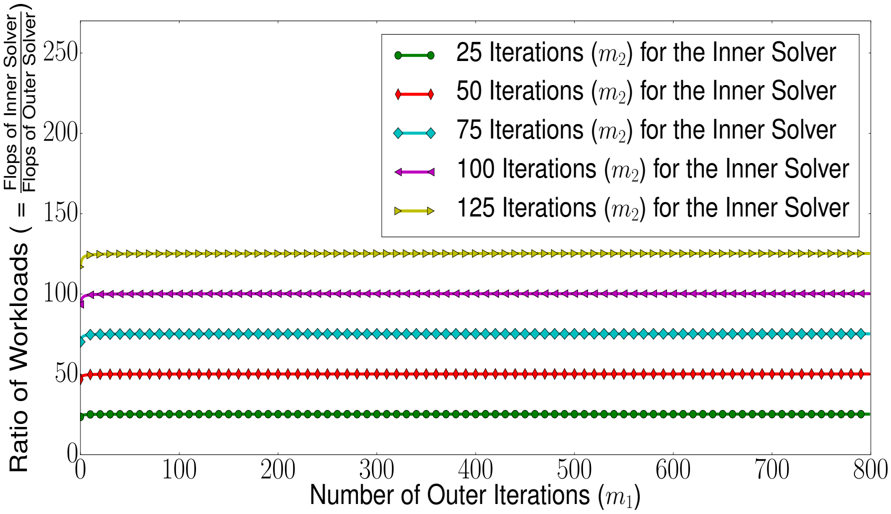
(a) Workloads of the outer and inner solver (m_2) for different numbers of iterations of the F-CG, the density of non-zero elements for matrix A is 0.001.

(b) Workloads of the outer and inner solver (m_2) for different numbers of iterations of the F-CG, the density of non-zero elements for matrix A is 0.25.

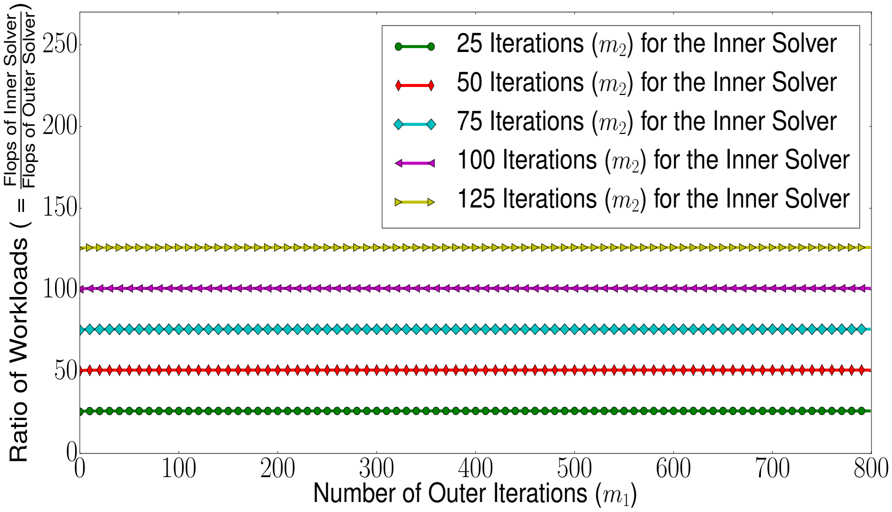
Figure 9 Workloads of the outer and inner solver for different numbers of iterations of the F-CG and various densities of non-zero elements for matrix A.

In figure 9 the workloads of the inner and as well outer solver of the F-CG are shown in number of floating point operations or also called flops. The workloads are computed for a matrix with size 1000×1000 where a density of 1.00 means that all elements are non-zero elements in contrast a density of 0.001 denotes a diagonal matrix with 1000 elements. The workload in number of floating point operations for the CG (see section 6.1.4) can be computed with $Flops_{CG}(m, n, A)$. Then the workload for F-CG is given by $Flops_{F-CG}(m_1, m_2, k, A) = m_1\ Flops_{CG}(m_2, n, A) + Flops_{CG}(m_1, n, A)$ with m_1 as the number of outer iterations. The increase of flops for the inner solver mainly depends on the non-zero elements as in the case for the outer solver of the F-CG shown in figure 9a.

6.1.6 Workload distribution of F-CG for various densities of non-zero elements



(a) Ratio of workloads between the outer and inner solver (m_2) for different numbers of iterations, the density of non-zero elements for matrix A is 0.001.

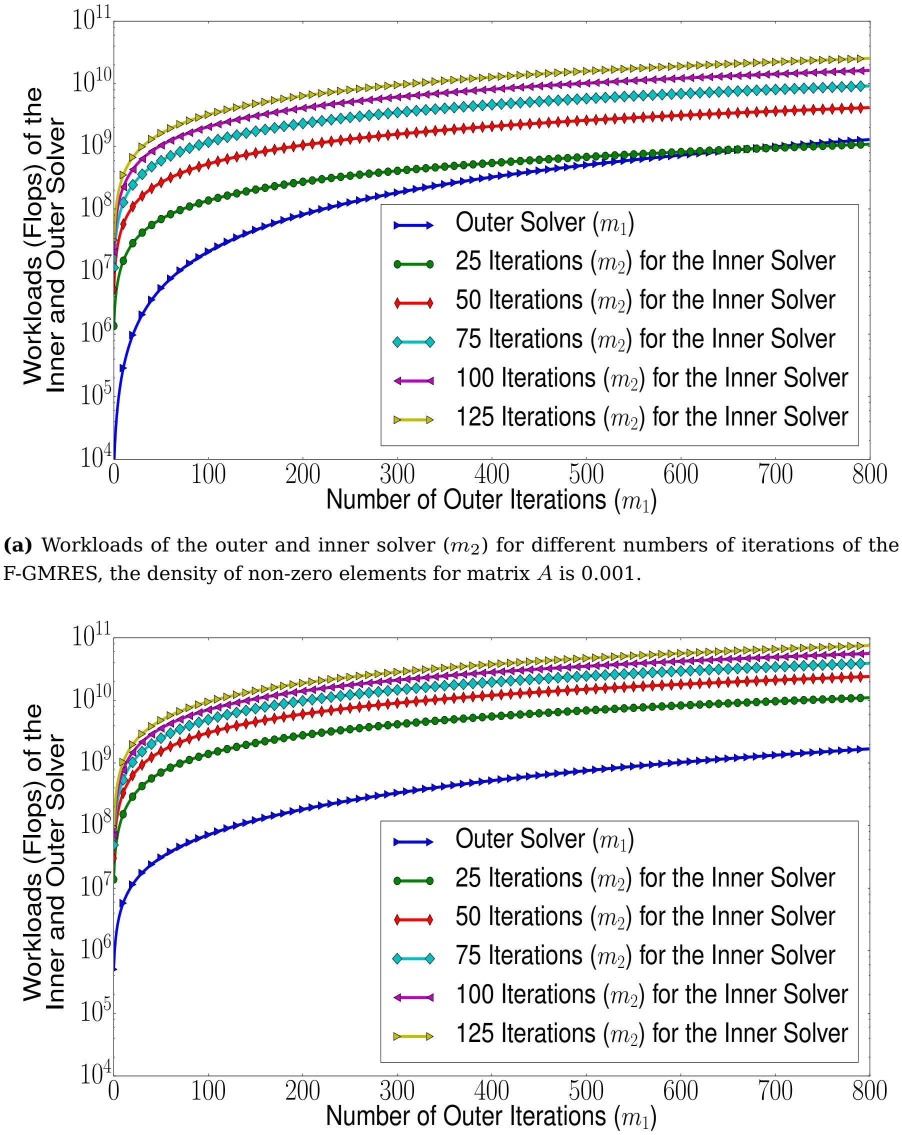


(b) Ratio of workloads between the outer and inner solver (m_2) for different numbers of iterations, the density of non-zero elements for matrix A is 0.25.

Figure 10 Ratio of workloads between the outer and inner solver for different numbers of iterations of the F-CG and various densities of non-zero elements for matrix A.

In figure 10 the ratio of workloads between the outer and inner solver of the F-CG is shown for a matrix with size 1000×1000 where a density of 1.00 means that all elements are non-zero elements in contrast a density of 0.001 denotes a diagonal matrix with 1000 elements. The total workload in number of floating point operations or also called flops for the CG (see section 6.1.4) can be computed by $Flops_{CG}(m, n, A)$ with m as the number of iterations, $nnz(A)$ as the number of non-zero elements and n for the size of matrix A. For the F-CG the total workload is given by $Flops_{F-CG}(m_1, m_2, k, A) = m_1\ Flops_{CG}(m_2, n, A) + Flops_{CG}(m_1, n, A)$ with m_1 for the outer iterations. Therefore the ratio of flops for the inner and outer solver is $\frac{m_1\ Flops_{CG}(m_2,n,A)}{Flops_{CG}(m_1,n,A)} = m_2$.

6.1.7 Workloads of the outer and inner solver of F-GMRES for various matrix densities



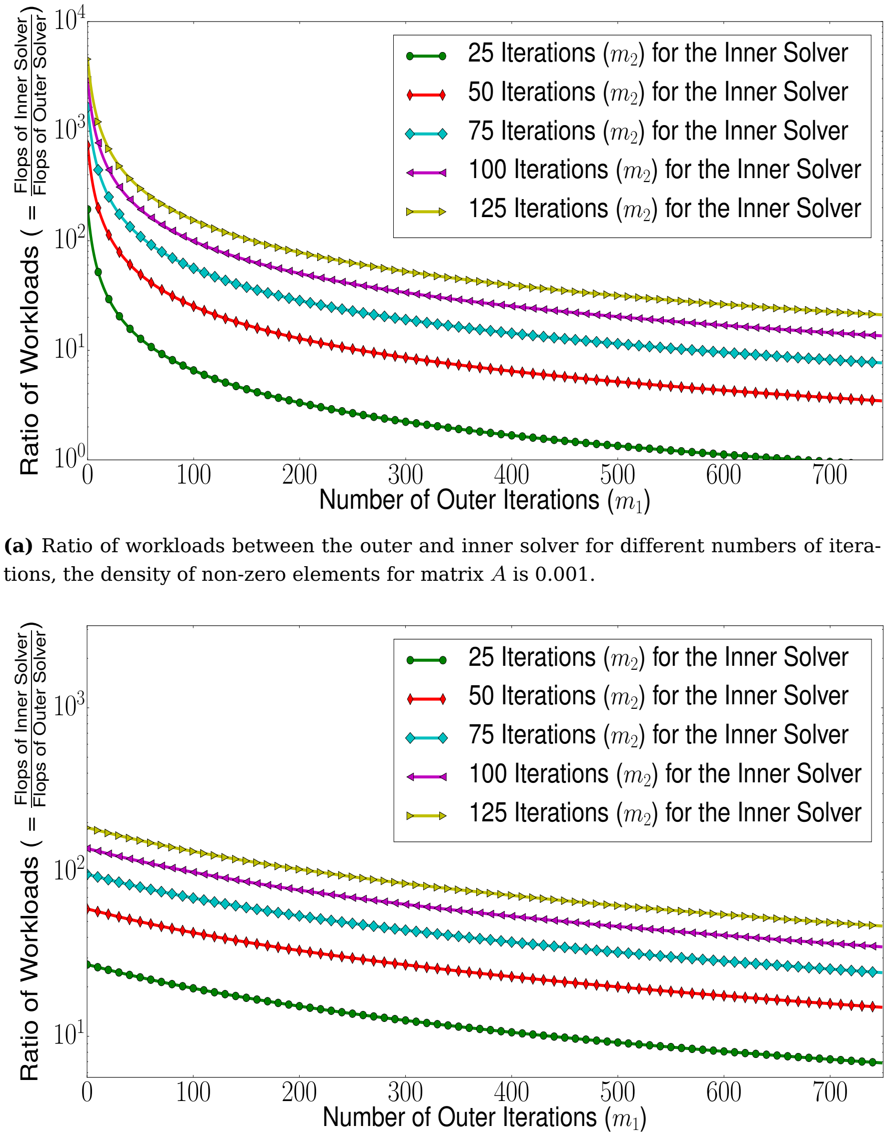
(a) Workloads of the outer and inner solver (m_2) for different numbers of iterations of the F-GMRES, the density of non-zero elements for matrix A is 0.001.

(b) Workloads of the outer and inner solver (m_2) for different numbers of iterations of the F-GMRES, the density of non-zero elements for matrix A is 0.25.

Figure 11 Workloads of the outer and inner solver for different numbers of iterations of the F-GMRES and various densities of non-zero elements for matrix A.

In figure 11 the workloads for the inner and as well outer solver of the F-GMRES are shown in number of floating point operations. The workloads are computed for a matrix with size 1000×1000 where a density of 1.00 means that all elements are non-zero elements in contrast a density of 0.001 denotes a diagonal matrix with 1000 elements. The workload in number of floating point operations for the GMRES (see section 6.1.4) can be computed with $Flops_{GMRES}(m, n, A)$. For the F-GMRES the workload is given by $Flops_{F-GMRES}(m_1, m_2, k, A) = m_1\ Flops_{GMRES}(m_2, n, A) + Flops_{GMRES}(m_1, n, A)$ with m_1 as the number of outer iterations. The increase of flops for the inner solver mainly depends on the non-zero elements in contrast to the outer solver shown in figure 11a.

6.1.8 Workload distribution of F-GMRES for various densities of non-zero elements



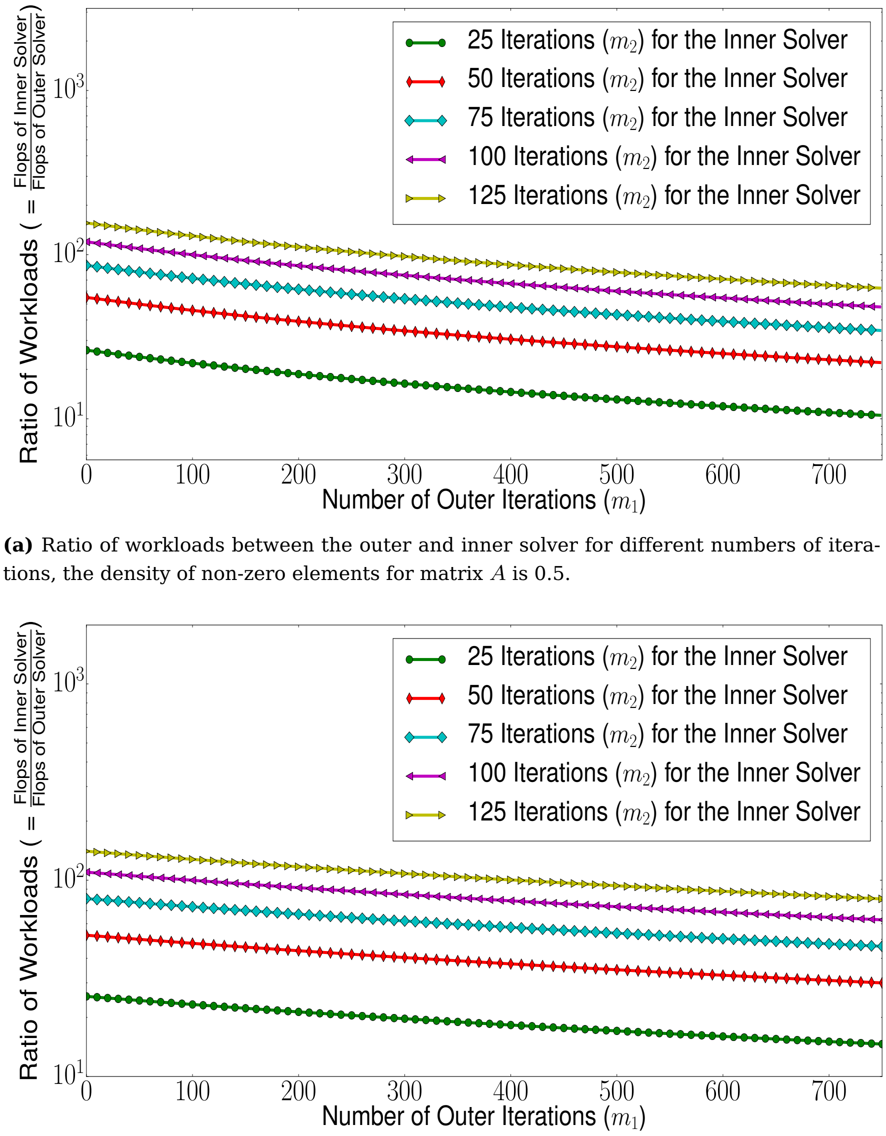
(a) Ratio of workloads between the outer and inner solver for different numbers of iterations, the density of non-zero elements for matrix A is 0.001.

(b) Ratio of workloads between the outer and inner solver for different numbers of iterations, the density of non-zero elements for matrix A is 0.25.

Figure 12 Ratio of workloads between the outer and inner solver for different numbers of iterations of the F-GMRES and various densities of non-zero elements for matrix A.

In figure 12 the ratio of workloads between the outer and inner solver of the F-GMRES is shown for a matrix with size 1000×1000 where a density of 0.25 means that a fourth are non-zero elements in contrast a density of 0.001 denotes a diagonal matrix with 1000 elements. The total workload in number of floating point operations for the GMRES (see section 6.1.4) can be computed by $Flops_{GMRES}(m, n, A)$ with m as the number of iterations, $nnz(A)$ as the number of non-zero elements and n for the size of matrix A. For the F-GMRES the total workload is given by $Flops_{F-GMRES}(m_1, m_2, k, A) = m_1\ Flops_{GMRES}(m_2, n, A) + Flops_{GMRES}(m_1, n, A)$ with m_1 for the outer iterations. Therefore the ratio of the inner and outer solver in figure 12 is $\frac{m_1\ Flops_{GMRES}(m_2,n,A)}{Flops_{GMRES}(m_1,n,A)} \approx \frac{m_2^2}{m_1}$ which tends "fast" to zero for 25 inner iterations.

6.1.9 Workload distribution of F-GMRES for various densities of non-zero elements



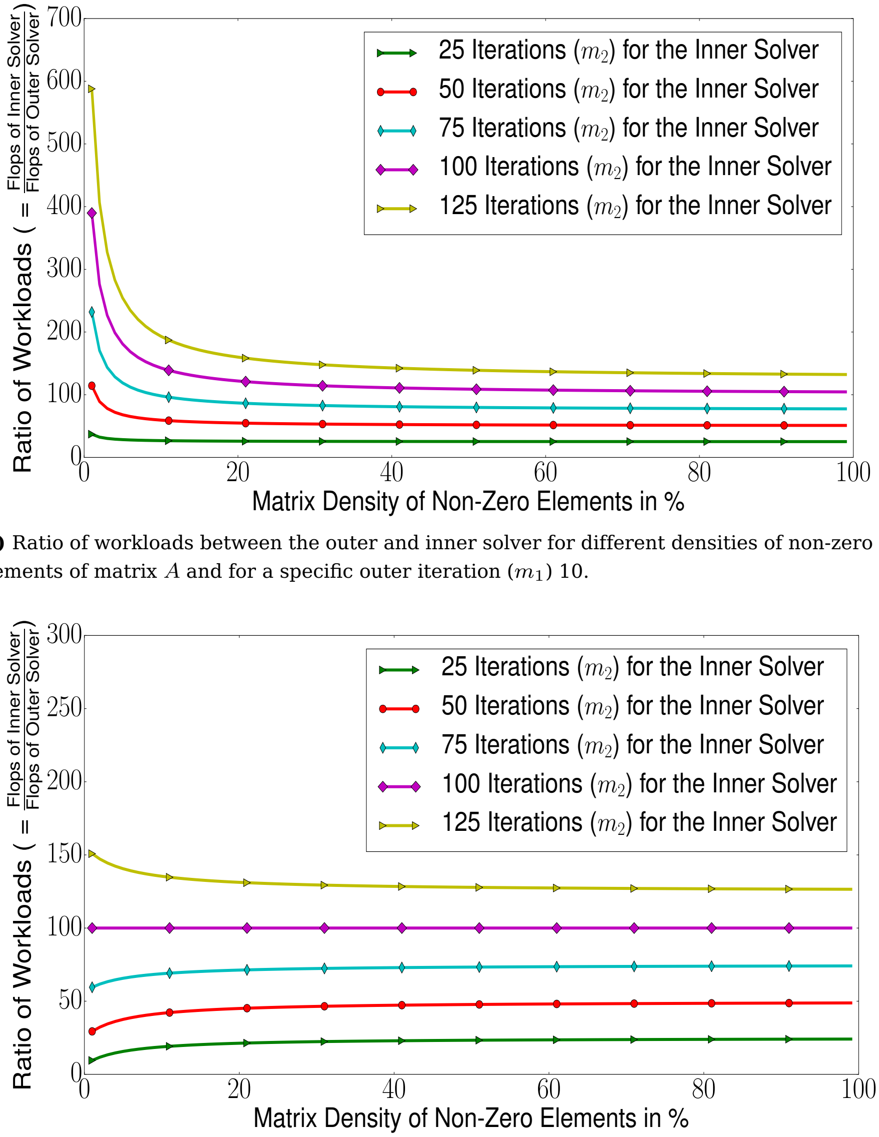
(a) Ratio of workloads between the outer and inner solver for different numbers of iterations, the density of non-zero elements for matrix A is 0.5.

(b) Ratio of workloads between the outer and inner solver for different numbers of iterations, the density of non-zero elements for matrix A is 1.0.

Figure 13 Ratio of workloads between the outer and inner solver for different numbers of iterations of the F-GMRES and various densities of non-zero elements for matrix A.

In figure 13 the ratio of workloads between the outer and inner solver of the F-GMRES is shown for a matrix with size 1000×1000 where a density of 1.00 means that all elements are non-zero elements in contrast a density of 0.5 denotes a matrix where the half are non-zero elements. The total workload in number of floating point operations for the GMRES (see section 6.1.4) can be computed by $Flops_{GMRES}(m, n, A)$ with m as the number of iterations, $nnz(A)$ as the number of non-zero elements and n for the size of matrix A. For the F-GMRES the total workload is given by $Flops_{F-GMRES}(m_1, m_2, k, A) = m_1\ Flops_{GMRES}(m_2, n, A) + Flops_{GMRES}(m_1, n, A)$ with m_1 for the outer iterations. Therefore the ratio of the inner and outer solver is $\frac{m_1\ Flops_{GMRES}(m_2,n,A)}{Flops_{GMRES}(m_1,n,A)} \approx m_2$ in figure 13 which tends slowly to zero because in matrix A are a lot of non-zero elements.

6.1.10 Workload distribution of F-GMRES for various densities of non-zero elements and for a fixed outer iteration



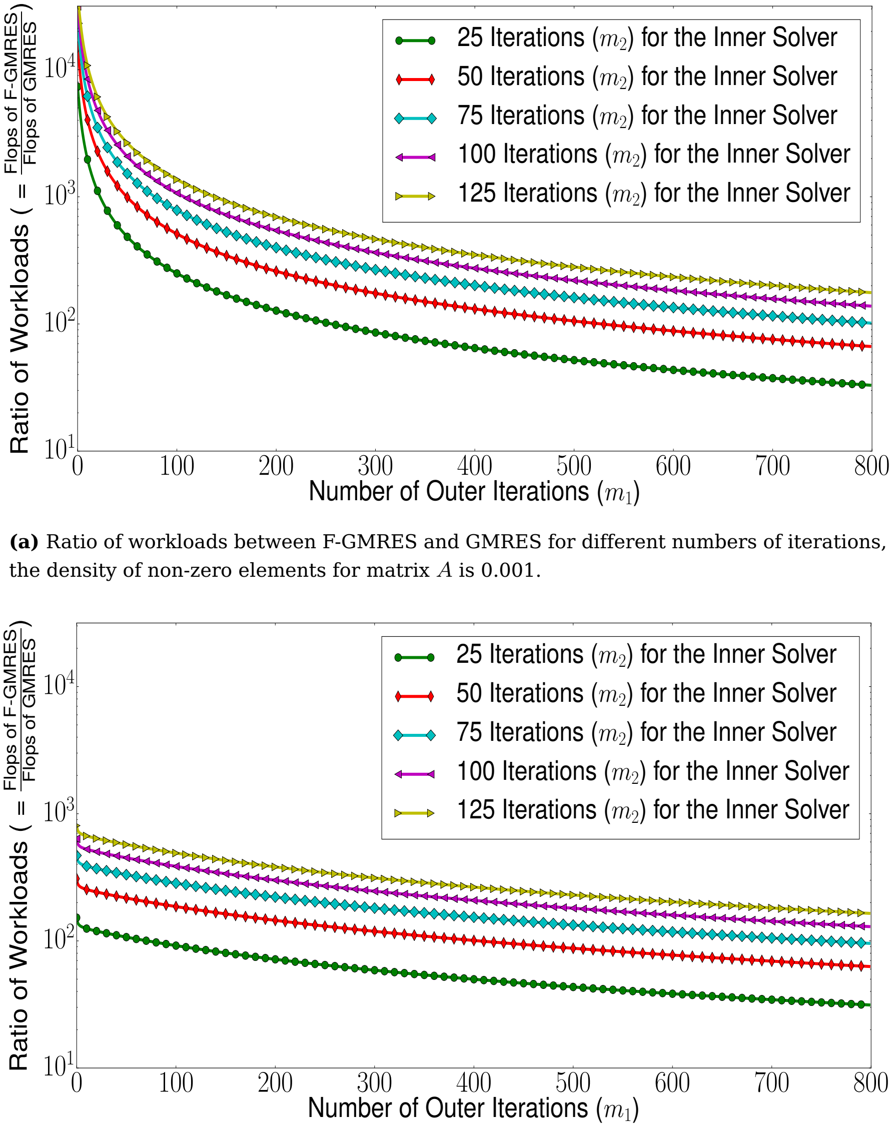
(a) Ratio of workloads between the outer and inner solver for different densities of non-zero elements of matrix A and for a specific outer iteration (m_1) 10.

(b) Ratio of workloads between the outer and inner solver for different densities of non-zero elements of matrix A and for a specific iteration (m_1) 100.

Figure 14 Ratio of workloads between the outer and inner solver of the F-GMRES for various densities of non-zero elements of matrix A and for a fixed outer iteration (m_1).

Like in section 6.1.9 the ratio of workloads between the outer and inner solver of the F-GMRES for different densities of non-zero elements is shown in figure 14. The main contrast is that the ratio of workloads is only computed for some fixed outer iterations (m_1) but for different densities of non-zero elements of matrix A. As in section 6.1.9 the ratio of workloads between the outer and inner solver is computed with $\frac{m_1\ Flops_{GMRES}(m_2,n,A)}{Flops_{GMRES}(m_1,n,A)}$ which tends to about m_2 for very high numbers of non-zero elements. It also means in almost all cases most of the workload is done in the inner solver but the ratio of workloads tends to zero for very large numbers of outer iterations (m_1).

6.1.11 Workload distribution between F-GMRES and GMRES for various densities of non-zero elements



(a) Ratio of workloads between F-GMRES and GMRES for different numbers of iterations, the density of non-zero elements for matrix A is 0.001.

(b) Ratio of workloads between F-GMRES and GMRES for different numbers of iterations, the density of non-zero elements for matrix A is 0.25.

Figure 15 Ratio of workloads between F-GMRES and GMRES for various densities of non-zero elements of matrix A and different iterations for the inner solver (m_2) of the F-GMRES.

In figure 15 the ratio of workloads between F-GMRES and GMRES for different numbers of iterations of the inner solver (m_2) and various densities of non-zero element for matrix A is shown. Like in section 6.1.7 a matrix with size 1000 × 1000 is used where a density of 0.001 denotes a diagonal matrix with 1000 non-zero elements. In contrast a density of 1.00 denotes a matrix where all its values are non-zero elements. The number of operations is computed as in section 6.1.4. In figure 15a the ratio tends to zero for very large outer iterations (m_1) which means the GMRES solver needs more operations than F-GMRES. In figure 15a and 15b this ratio tends to zero for very large outer iterations and slowly because of the orthogonalization process from the GMRES solver.

6.2 Fault Tolerant Iterative Linear solvers (FT-solvers) [4][11][12]

6.2.1 Fault tolerance and selective reliability [11]

Fault tolerance is mainly based on the concepts of Dijkstra (1973) [11] these principles can be also applied in different contexts. The first applying of the Dijkstra algorithm was in the relation of communication systems. In this model there are some nodes or PCs where all of them are connected through a communication network. For each node there can be also a different state which can be at least valid or invalid which depends on some rules. Each state of each node also relies on the neighbor states.

Furthermore in this model there is a global checker which observes if the whole system is in a correct or invalid state which is also based on some rules similar to the case of local rules. The whole system is in a valid state if these global rules are satisfied if one of them is hurt the whole system is in an invalid state. There is also a global clock which brings each node into the next state from the current state based on some predefined rules. The step size of this clock can be chosen arbitrary which means it can be continuous or discrete. In the model of Dijkstra the step size is chosen discrete.

The next computed state for each node also depends on the states of the neighbors. If any node is in an invalid state a local mechanism based on some rules brings this node back in a valid state. After some time steps the whole system should be in a valid state. This correction step for each node can be done after a finite number of time steps periodically to guarantee that the whole system converges to a valid state. Dijkstra allows to define different rules, it mainly depends on the opinion what is considered as a valid or invalid state. There may be also some pitfalls because some of these assumptions may be wrong.

In iterative methods these global rules are often based on some variables like the residuum ($||r||_2 = ||Ax - b||_2$). In the case of the defect error correction or iterative refinement method in section 9 the whole solver consists of an outer solver and inner solver. The outer solver has to check for convergence (predefined rule) as long as the solution vector x doesn't satisfy a specific accuracy whereas the inner solver should do the main work. In the Newton's fixed point method but from the point of view of Dijkstra the outer solver is the global checker which has to check these variables or rules like if the residuum ($||r||_2 = ||Ax - b||_2$) is minimized or not.

If one of these rules is not satisfied (e.g. the residuum is not minimized) the computation has to be proceed as long as a solution is found for $Ax = b$. Then if a solution x is found or the whole solver is in a valid state the residuum r is minimized. There can be also other rules defined for this approach. In conclusion fault tolerant iterative linear solvers are nothing else as a generalization of iterative refinement like in section 9 with the extension of the Dijkstra approach [11].

In figure 16 the main concept about fault tolerant iterative linear solvers is shown. There is an initial solution x_0 whereas the outer solver checks if the solution x for solving $Ax = b$ computed so far minimizes the residuum ($||r||_2 = ||Ax - b||_2$). If not the solution is improved in the inner solver which is done in unreliability. After a finite number of iterations for the inner solver the outer solver checks for convergence. This must be done in reliability because it must be guaranteed that the decision based on some rules like if the residuum ($||r||_2 = ||Ax - b||_2$) is minimized or the solution vector x is valid is reliable. In contrast to this approach it would be too chaotic if this would be not the case such that there are no sections in the used algorithm where nothing is trustworthy so nothing can be predicted of the computed solution x for solving $Ax = b$ because everything would be unreliable.

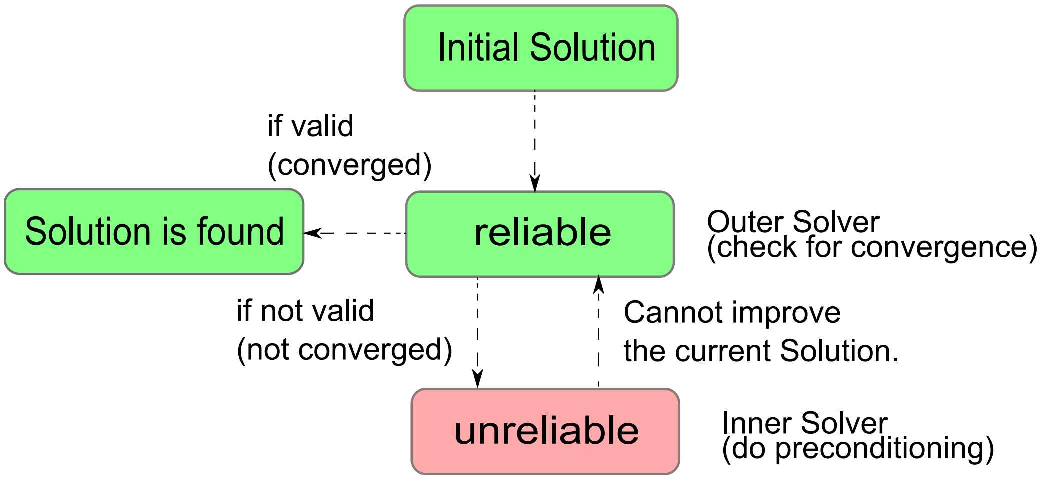


Figure 16 Illustration of the conceptual framework of Fault Tolerant Iterative Linear solvers.

6.2.2 Relation between iterative refinement and sand boxing

The main contrast between applying different precision levels and sandboxing with reliable and unreliable sections is that reliability is related to higher precision but lower precision levels are referred to unreliable computations. There must be always a section above all computations which checks for correctness of $Ax = b$, this is always done with higher accuracy or with reliability.

It doesn't matter how this part of the algorithm is designed which checks for convergence and estimates the error of the solution vector x but this section must guarantee correctness of the finally computed solution. In algorithm 12 and 13 two perspectives are shown, the first one from the point of view with iterative refinement and two predefined precision levels (single and double) in algorithm 12 and in contrast to sand-boxing like in algorithm 13. The outer solver checks for convergence and correctness of the computed solution which is done in reliability, this is mainly done during testing the condition of the while loop (line 4) in algorithm 12 and 13.

Algorithm 12 Iterative refinement with different precision levels (single, double):

```
Input: Matrix $A$, right hand side $b$ and
    tolerance $\epsilon$ for the relative residuum.
1: $x_0 = zeros()$                        single or
2:                                        double precision
3: $j = 0, r_0 = b - Ax_0$                double precision
4: while $||r_j||_2/||r_0||_2 > \epsilon$ do   double precision
5:     $r_j = b - Ax_j$                   double precision
6:     Solve $M_j d_j = r_j$ for $d_j$    single precision
7:     $x_{j+1} = x_j + d_j$              double precision
8:     $j = j + 1$
9: end while
```

Algorithm 13 Iterative refinement with sand boxing (reliable, unreliable):

```
Input: Matrix $A$, right hand side $b$
    and tolerance $\epsilon$ for the relative residuum.
1: $x_0 = zeros()$                        in reliable mode
2:                                        or unreliable mode
3: $j = 0, r_0 = b - Ax_0$                in reliable mode
4: while $||r_j||_2/||r_0||_2 > \epsilon$ do   in reliable mode
5:     $r_j = b - Ax_j$                   in reliable mode
6:     Solve $M_j d_j = r_j$ for $d_j$    in unreliable mode
7:     $x_{j+1} = x_j + d_j$              in reliable mode
8:     $j = j + 1$
9: end while
```

In algorithm 12 it is shown defect error correction with different precision levels. It is possible to achieve the accuracy of double precision where most of the computations are done in single precision during line 6 of algorithm 12. Algorithm 13 shows the same method but with sand boxing where most of the computations should be done in unreliable mode during the same line 6. Both algorithms are really similar the only thing which has changed is that computing with single precision is now done with unreliability. Bad preconditioning comes now from bit-flips in the inner solver (line 6) of algorithm 13 which leads to more iterations for the outer solver.

6.2.3 Some assumptions on the unreliable model

The aim of fault tolerant iterative linear solvers is that most of the operations should be done in unreliable mode (in the inner solver) nevertheless if everything is done in unreliability it would be not possible to get any deterministic behavior because every operation is allowed to fail. If this would be the case the whole solver might never converge to a satisfying solution x in every computation, so some assumptions must be done based on the according knowledge so far.

One good discrimination can be done by separating the used data in static and dynamic data. Dynamic data are data where the values change during the computation whereas static data are not allowed to do this like for the used matrix A and the right hand side b. So changing the state is mainly allowed to the vector x for example. It would never be possible to check for convergence with $||r||_2 = ||Ax - b||_2$ because all data are perhaps faulty especially the matrix A and vector b in this case a different problem would be solved for the matrix A and b.

Maybe matrix A and vector b can be reloaded from the hard disk. All other data are allowed to change its state during the computation and can be faulty or not as long as no further assumptions are done for the unreliable mode. Also control structures like for-loops, if-clauses and index variables for doing the memory access of matrix A and b should be error free and also accessing to other variables for static data.

Therefore a further differentiation can be done because flipping a bit should be only possible with floating point operations and not allowed for integer variables. Floating point values need much more storage place than integer variables but allowing integer variables to fail will complicate the whole model and will contribute really less for reducing the energy requirements.

Because of the assumption that matrix A and vector b should not be not effected from bit-flips in the DRAM, faulty data from the cache (fast memory for the CPU) should also not disturb those data in the main memory otherwise wrong data in the cache would change these data in the main memory. It also means that data from the DRAM can be reloaded. Based on these assumptions given above further adoptions can be done but those two main assumptions should not be hurt because it will lead to any nondeterministic behavior and it wouldn't be able to solve the problem $Ax = b$.

Properties of the unreliable model:

- Integer variables are not allowed to fail because there will be less contribution for lowering the energy requirements.
- Also for-loops, if-conditions and so on are done with high reliability because it will contribute nothing if this would be done with low reliability.
- Faulting is mainly restricted to floating point values.
- Static data like for matrix A and b are not allowed to be faulty especially in the main memory.
- Dynamic data like the vector x are allowed to produce wrong results.
- Checking for convergence like the residuum $||r||_2 = ||Ax - b||_2$ must be done in reliability.

6.2.4 The Fault Tolerant GMRES (FT-GMRES) method [4][12]

The Fault Tolerant GMRES (FT-GMRES) is similar to the Flexible GMRES (F-GMRES) as in section 6.1.3 the only thing which has changed is that solving $w_j = GMRES(M_j, q_j, w_0)$ for vector w_j will be done in unreliable mode where faulting is now allowed. The outer solver has still the duty to check for correctnesses and convergence because most of the computations should be done in unreliable mode. The Fault Tolerant GMRES (FT-GMRES) solver is shown in algorithm 14 only a single line has changed. Line 4 in algorithm 14 has changed from reliable to unreliable mode in contrast where everything is done with the same reliability as in the case of the F-GMRES solver in section 6.1.3. This inner/outer - approach also allows to restart the inner solver (line 4) with a solution vector w_j if not a satisfying solution w_j is computed so far. The vector w_0 is just the initial vector for the solution w_j in algorithm 14 and line 4 for the inner solver. Bit-flips in the inner solver (line 4) are not seen anymore as single events but as an error which disturbs vector w_j and comes mainly from the poor preconditioning and leads to the problem that the outer solver (line 3-23) has to do more iterations if the inner solver does no satisfying preconditioning for the outer solver. In algorithm 14 the value ϵ_2 is the accuracy of the inner solver for solving the problem $q_j = M_j w_j$.

Algorithm 14 The Fault Tolerant GMRES (FT-GMRES) algorithm without restarts [4][12]:

Input: Matrix A, right hand side b and initial starting vector x_0 and ϵ as the accuracy for solving $Ax = b$ and m_1 for the number of outer iterations and m_2 for the iterations of the inner solver.

Output: Approximate solution of vector x_{m_1} for $m_1 > 0$.

```
1:  $r_0 = b - Ax_0$                                   ...▷ Compute initial residuum vector.
2:  $\beta = ||r_0||_2$, $q_1 = r_0/\beta$             ...▷ Compute first basis vector $q_1$.
3:  for $j = 1, 2, ...$until convergence and $j < m_1$ do
4:    Solve $q_j = M_j w_j$ for $w_j$                  ...▷ Do preconditioning in unreliable
    mode like $w_j = GMRES(M_j, q_j, w_0, m_2, \epsilon_2, \dots)$.   (In this master work $M_j \approx M \approx A$.)
5:    $v_{j+1} = Aw_j$ ($v_{j+1} = AM_j^{-1}q_j$)      ...▷ Perform matrix vector product.
6:    ▷ Orthogonalize basis vector $q_j$ (line 7-10).
7:    for $i = 1, 2, ...j$ do
8:      $h_{i,j} = q_i^T v_{j+1}$
9:      $v_{j+1} = v_{j+1} - h_{i,j} q_i$
10:   end for
11:   $h_{j+1,j} = ||v_{j+1}||_2$
12:   ..▷ Do rank revealing decomposition [49] (, apply Givens rotation to $\hat{H}(1{:}j,1{:}j)$).
13:   if $\hat{H}(j+1, j) < \epsilon_{Machine}$ then   ...▷ Stopped if machine precision is
    achieved because $\hat{H}(j+1, j) \approx 0$ so no further residual reduction.
14:     if $\hat{H}(1 : j, 1 : j)$ no full rank then
15:       Did not converged. Try recovery strategies.
16:     else
17:       Solution is $x_{j-1}$.
18:     end if
19:   end if
20:   $q_{j+1} = v_{j+1}/h_{j+1,j}$                   ...▷ New basis vector $q_{j+1}$.
21:   $y_j = argmin_y||\hat{H}(i : j+1, 1 : j)y - \beta e_1||_2$   ...▷ Solve least square problem.
22:   $x_j = x_0 + [q_1, q_2, \dots, q_j]y_j$          ...▷ Compute solution $x_j$.
23: end for
24: Solution found $x_{j-1}$.
```

6.3 Dealing with rank deficiency in F-GMRES/FT-GMRES [70]

This is a short section which summarizes different strategies to handle unlucky preconditioning of the inner solver in F-GMRES and FT-GMRES. These possibilities are not applied in this work.

6.3.1 Additional failure modes for the F-GMRES

In the case of the F-GMRES there are additional failure modes. For the F-GMRES the expression of $\hat{H}(j+1, j) = 0$ does not necessary mean convergence because for the standard GMRES solver $\hat{H}(1:j, 1:j)$ is always nonsingular if j is the smallest iteration index with $\hat{H}(j+1, j) = 0$ in that case [70]. In contrast for the Flexible GMRES $\hat{H}(1:j, 1:j)$ may not be nonsingular for the outer solver in line 13 of algorithm 11. This is Saad's proposition (2.2) in [3] which can also happen in exact arithmetic for the F-GMRES. It is a really seldom event which may occur through an unlucky choice of the preconditioners for the F-GMRES like for example if "$M_j^{-1} = A$ and $M_{j+1}^{-1} = A^{-1}$" [70]. This rank deficiency because of preconditioning can be inexpensive detect through a rank revealing decomposition like in [49]. This approach does not need more operations than the standard QR - factorization and can be done with $\mathcal{O}(m^2)$ operations in m iterations for the upper Hessenberg matrix. This ability to detect rank deficiency fits in the general approach of the F-GMRES it either:

1. "converges to the desired tolerance," [70]
2. "correctly detects an invariant subspace, with a clear indication $(H(j+1, j) = 0$ and $H(1:j, 1:j)$ is nonsingular), or" [70]
3. "gives a clear indication of failure $(H(j+1, j) \neq 0$ and detected rank deficiency of $H(1:j, 1:j))$." [70]

These detector methods can be applied for the outer solver of the F-GMRES in line 13 of algorithm 11.

6.3.2 Recover strategies for the FT-GMRES

If rank deficiency is detected in the outer solver of the FT-GMRES in line 15 of algorithm 14 then different possibilities of fault correction can be applied. This line covers mainly the cases where unlucky preconditioning is applied of the inner solver. So there are three different recover/restarting strategies to deal with unlucky preconditioning of the inner solver in the FT-GMRES:

1. "retry the current iteration starting from Line 5 inclusive;" [70]
2. "retry the current iteration after Line 5, but replace w_j with a random" vector (scaled appropriately according to best estimates of $||A^{-1}||_2$); or" [70]
3. "stop and return x_{j-1}, the last good approximate solution." [70]

This table also covers different possibilities for the F-GMRES not only for the FT-GMRES because unlucky preconditioning can happen through a fault but as well without a fault.

6.4 Relaxation strategies for nested Krylov methods [47][71][72]

This section is mainly related to flexible methods of (*inexact*) Krylov methods but as well linked to the initialization of the inner solver for F-GMRES and FT-GMRES. It gives also some short prospects and strategies for preconditioning of the inner solver. This section sums up most of the important topics and papers such that not in every case full proofs are given so mainly just see [71] and [72].

6.4.1 Relaxation strategies for inexact Krylov subspace methods [71]

The central problem of iterative methods is to find a vector x that approximately solves:

$$Ax_\star = b \text{ such that } ||b - Ax||_2 < \theta \tag{77}$$

, for a given accuracy θ which measures the distance to the right hand side b. In Krylov subspace methods there is in each step a matrix vector product applied whereas in *inexact* Krylov subspace methods not the the exact matrix vector product is performed such that there are some deviations. So let $\mathcal{M}_\eta(v)$ be the approximate matrix vector operation of Av and with the relative precision η such that:

$$\mathcal{M}_\eta(v) = Av + g \text{ with } ||g||_2 \leq \eta||A||_2||v||_2. \tag{78}$$

In this case it is assumed that the matrix vector product of $\mathcal{M}_\eta(v)$ becomes more costly if the relative precision η is chosen smaller. So let η be chosen dynamically for each matrix vector operation and with the according residuum $||r_j|| = |b - Ax_j||_2$ for each iteration j:

$$\eta_j = \frac{\theta}{||r_j||_2}. \tag{79}$$

In this equation it becomes more obvious that for the very first matrix vector operations the precision η_j for each iteration j has to be more precise because of the large residuum $||r_j||_2$. Afterwards the precision η_j becomes more relaxed as soon as the used solver starts converging, this is the point when the residuals become increasingly smaller. In *inexact* Krylov methods a perturbation of $||g_{j-1}||_2 \leq \eta_{j-1}||A||_2||v||_2$ is added in step j for each matrix vector operation. In the case of the GMRES following relation can be identified which links together the parts of interests:

$$AQ_m = Q_{m+1}\underline{\hat{H}}_m \text{ and } x_m = Q_m\hat{H}_m^{-1}e_1 \text{ with } Q_me_1 = b \tag{80}$$

, with $\underline{\hat{H}}_m$ being a $(m+1) \times m$ upper Hessenberg matrix and $\hat{H}_m$ as the $m \times m$ upper block of $\underline{\hat{H}}_m$. The equation of $x_m = Q_m\hat{H}_m^{-1}e_1$ comes from equation 63 and by solving $||||r_0||_2e_1 - \hat{H}_my||_2$ for vector y where $||r_0||_2 = ||b||_2 = ||1||_2$ and $x_0 = 0$. In the case of *inexact* Krylov methods then this relation with the perturbation matrix F_m becomes:

$$AQ_m + F_m = Q_{m+1}\underline{\hat{H}}_m \text{ and } x_m = Q_m\hat{H}_m^{-1}e_1 \text{ with } Q_me_1 = b. \tag{81}$$

The vector f_j is the $j+1$-th column of matrix F_m which contains the deviation from each exact matrix vector product in the step $j+1$ and therefore it follows that $||f_j||_2 \leq \eta_j||A||_2||q_j||_2$ for all iterations and matrix vector operations which can be easily checked.

As a consequence of the perturbation term F_m the residual r_m is not the residual anymore for vector x_m therefore r_m is the computed residual instead of the true residual which is computed with $b - Ax_m$. Then in *inexact* Krylov methods following equation becomes the main focus of interest:

$$||b - Ax_m||_2 \leq ||r_m||_2 + ||r_m - (b - Ax_m)||_2 \tag{82}$$

, which expresses the residual gap between the *true* $b - Ax_m$ and the *computed* residual r_m. So if the residual gap becomes small then there is also less distance between the true and computed residual which means both are nearly the same. So from the above equation it follows that:

$$r_m - (b - Ax_m) = r_m - r_0 + AQ_m \hat{H}_m^{-1} e_1 = -F_m \hat{H}_m^{-1} e_1 = -\sum_{j=1}^{m} f_{j-1} e_j^T \hat{H}_j^{-1} e_1. \tag{83}$$

From this equation also following expression can be obtained with using the norm $||.||_2$:

$$-\sum_{j=1}^{m} f_{j-1} e_j^T \hat{H}_j^{-1} e_1 \leq \sum_{j=1}^{m} \eta_{j-1} ||A||_2 ||q_{j-1}||_2 ||e_j^T \hat{H}_j^{-1} e_1||_2 \tag{84}$$

, which gives the upper bound of the perturbation and let $||A||_2||q_j||_2||e_j^T \hat{H}_j^{-1} e_1||_2 \leq ||A||_2||A_j^{-1}||_2||r_j||_2$ with $r_j = b - Ax_j$ and x_j as the approximate solution vector of the Krylov subspace $\mathcal{K}_j$. Formula 84 shows that the reachable precision is determined by the residual gap which can be also used for *inexact* Krylov methods because if this iterative process is stopped at $||r_j||_2 \leq \theta$ then the residual gap specifies the precision of the *inexact* Krylov method, if θ is small then also the product with some constants. Finally this bound holds mainly for exact matrix vector products not for the perturbed matrix vector product of Av.

6.4.2 Nested inexact Krylov methods [71]

In this section nested Krylov methods are discussed which are shown in section 6.1 but also nested *inexact* Krylov methods. In the case of *inexact* Krylov methods when solving the linear equation system of $Ad = r$ with at most the relative residual precision of ξ following notation is used:

$$d = \mathcal{M}_\xi(r), \text{ where } d \text{ such that} ||r - Ad||_2 \leq \xi ||r||_2. \tag{85}$$

Notice this is indeed in the case of flexible preconditioning like in algorithm 9 that may change for each iteration step which depends on ξ but also on r.

6.4.2.1 The outer iteration: Richardson iteration

Then the *inexact* Krylov method is defined with the Richardson iteration as the outer iteration like in algorithm 9 for iteration $j = 1, \ldots, m$ by the following recurrence:

$$\begin{aligned} d_{j-1} &= \mathcal{M}_{\xi_{j-1}}(r_{j-1}) \\ x_j &= x_{j-1} + d_{j-1} \\ r_j &= r_{j-1} + A_{\eta_{j-1}} d_{j-1}. \end{aligned} \tag{86}$$

It can be easily verified that this also fits in the general idea of:

$$AQ_m + F_m = Q_{m+1} \underline{\hat{H}}_m \text{ and } x_m = Q_m \hat{H}_m^{-1} e_1 \text{ with } Q_m e_1 = b. \tag{87}$$

Furthermore with the help of the estimate for $||d_j||_2 \leq ||A^{-1}||_2 ||r_j||_2 (1 + \xi_j)$ following bound on the norm can be found for the residual gap:

$$||r_j - (b - Ax_j)||_2 \leq ||A||_2 \sum_{j=0}^{m-1} \eta_j ||d_j||_2 \leq ||A||_2 ||A^{-1}||_2 \sum_{j=0}^{m-1} \eta_j ||r_j||_2 (1 + \xi_j). \tag{88}$$

This inequality also comes from $||r_j - r_{j-1}||_2 \leq ||A_{\eta_{j-1}} d_{j-1}||_2$. This result in formula 88 suggests to pick up a tolerance η_j equal to $\theta / ||r_j||_2$ in step $j + 1$ which is then consistent with the unconditioned case. From the Richardson iteration following equation can be obtained:

$$
\begin{aligned}
||b - Ax_j||_2 &= ||b - A(x_{j-1} + d_{j-1})||_2 \\
&\leq ||r_{j-1} - (b - Ax_{j-1})||_2 + ||r_{j-1} - Ad_{j-1}||_2 \\
&\leq \eta_{j-1}||A||_2||x_{j-1}||_2 + \xi_{j-1}||r_{j-1}||_2.
\end{aligned}
$$

This suggests to use $\eta_j = \xi_j||r||_2$ then roughly following equation can be obtained:

$$||b - Ax_j||_2 \leq ||A||_2||A^{-1}||_2\xi_{j-1}||r_{j-1}||_2. \tag{89}$$

In formula 89 the main outcome is that the precision of the matrix vector product is chosen like the current residual precision times the expected residual reduction. This version of the Richardson iteration can be seen as periodic restart of a Krylov subspace method used for the inner iteration.

6.4.2.2 The outer iteration: Flexible GMRES (F-GMRES)

The Flexible GMRES which is based on Saad [5] is a way to deal with flexible preconditioning. This method constructs an orthogonal basis matrix Q_{m+1} for AQ_m where $w_j = \mathcal{P}_\xi(q_j)$ (preconditioned basis vector) with the inexact matrix vector product then the most important relations can be summarized with (see algorithm 11):

$$AQ_m + F_m = Q_{m+1}\underline{\hat{H}}_m \text{ and } x_m = Q_m\hat{H}_m^\dagger e_1 \text{ with } r_m = Q_{m+1}(I - \underline{\hat{H}}_m\hat{H}_m^\dagger)e_1, \tag{90}$$

with $\underline{\hat{H}}_m$ as an upper Hessenberg matrix. From this equation following relation is obtained:

$$||r_m - (b - A_m)||_2 = ||F_m\hat{H}_m^\dagger e_1||_2 \leq ||A||_2||A^{-1}||_2||\hat{H}_m^\dagger||_2 \sum_{j=0}^{m-1} \eta_j||r_j||_2(1 + \xi_j). \tag{91}$$

In general the norm of $||\hat{H}_m^\dagger||_2$ ($\dagger$ is called the pseudo inverse according to Moore Penrose [69] such that $H^{-1} \approx H^\dagger$) is hard to estimate therefore it is assumed that $||\hat{H}_m^\dagger||_2$ is bounded by a modest constant k then following equation can be obtained for the residual gap:

$$||r_m - (b - A_m)||_2 = k\theta||A||_2||A^{-1}||_2||\hat{H}_m^\dagger||_2(1 + \max_j \xi_j). \tag{92}$$

If α is defined as $(r_j =)\alpha = e_{j+1}^T(I - \underline{\hat{H}}_j\hat{H}_j^{-1})e_1$ and by using the optimality condition of the Flexible GMRES solver (the residual is decreased in the inner solver of the flexible method but in the case of the GMRES solver there are basis vectors used) then following solution is found with $y_{j+1} = [(\hat{H}_j^{-1}e_1)^T, \alpha]^T$ such that (for the full prove see [71] and [72]):

$$
\begin{aligned}
||r_{j+1}||_2 \leq ||b - Ax_{j+1}||_2 = ||b - AQ_{j+1}y_{j+1}||_2 = ||b - (AQ_jy_{j+1} + Aw_{j+1}y_{j+1})||_2 = \\
||b - AQ_j\hat{H}_j^{-1}e_1 - Aw_{j+1}\alpha||_2 = ||(q_{j+1} - Aw_{j+1})\alpha||_2 \leq \xi_j\alpha \leq \ ||\xi_jr_j||_2.
\end{aligned} \tag{93}
$$

The product of $\xi_j\alpha$ gives the bound for the residual reduction of the outer solver. This result shows that in the case of the associated Arnoldi process this method suffers from reasonable breakdowns. For example if the residuals are very large than the Flexible GMRES might be not a good choice because the bound which is given by $\xi_j\alpha$ is very rough such that a near break down cannot be found. However if the value of ξ_j is small enough but also for all iterations $\xi = \xi_j$ then it is not such a serious problem and the Flexible GMRES shows a good converge behavior. Equation 93 also means that the residual reduction is bounded by the preconditioning of the inner solver.

It is also worth to say that this relation is always satisfied for $w_{j+1} = 0$ as initialization because it is ensured that the new residuum ($||r_{j+1}||_2$) is at least smaller than $||\alpha q_{j+1}||_2$. In the case of flexible preconditioning (right preconditioning) often a reduction for the inner solver is chosen such that $||(q_{j+1} - Aw_{j+1})||_2/||q_{j+1}||_2 \leq \epsilon_2$ which is the main contrast between flexible (always the same precision in the inner solver) and right preconditioning (always the same number of iterations).

6.4.3 Some notes on the preconditioned GMRES solver [42][47]

In the standard case of the GMRES solver a Krylov polynomial is searched which minimizes the 2-norm of the residual $r_j = b - Ax_j$ with the Krylov subspace $K_j = [r, Ar, A^2r, \dots, A^{j-1}r]$ such that:

$$||r_j||_2 = \min_{x_j \in K_j(A,r)} ||b - Ax_j||_2. \tag{94}$$

Let $x_j \in K_j(A,r) = [r, Ar, A^2r, \dots, A^{j-1}r]$ in other words $x_j = \sum_{i=1}^{j-1} c_i A^i r$ where $x_0 = 0$ ($||r||_2 = ||b||_2$) which should minimize the residual ($||r_j||_2$). Then the residual ($||r_j||_2$) can be rewritten to $||r_j||_2 = ||b - A(\sum_{i=1}^{j-1} c_i A^i r)||_2$. Furthermore define following polynomial p_j as:

$$p_j(A) = 1 - \sum_{i=1}^{j-1} c_j A^j r \tag{95}$$

, which is the same like for formula 14. Then the 2-norm of the residual ($||r_j||_2$) reduces to:

$$||r_j||_2 = ||b - Ax_j||_2 = \min_{p_j \in P^j, p_j(0)=1} ||p_j(A)r||_2 \tag{96}$$

, with P^j as a set of polynomials with degree j or less. In equation 96 a polynomial with the lowest degree is searched which minimizes the residual of $||r||_2$. Just by dividing through $||r||_2$ the residual reduction of the standard GMRES solver is shown and following equation can be obtained:

$$\frac{||r_j||_2}{||r||_2} = \min_{p_j \in P^j, p_j(0)=1} ||p_j(A)||_2. \tag{97}$$

Now decompose matrix A with $A = V\Lambda V^{-1}$ if it is diagonalizable where matrix Λ stores all eigenvalues (λ) of A which is a diagonal matrix and let $I = VV^{-1}$ then this equation becomes:

$$\frac{||r_j||_2}{||r||_2} = \min_{p_j \in P^j, p_j(0)=1} ||V||_2 ||p_j(\Lambda)||_2 ||V^{-1}||_2. \tag{98}$$

Let $\kappa(V)_2$ be the condition number (see section 72) of V then this equation can be reformulated:

$$\frac{||r_j||_2}{||r||_2} = \min_{p_j \in P^j, p_j(0)=1} \kappa(V)_2 ||p_j(\Lambda)||_2. \tag{99}$$

Afterwards this equation becomes by only considering the largest eigenvalues (λ):

$$\frac{||r_j||_2}{||r||_2} \le \min_{p_j \in P^j, p_j(0)=1} \kappa(Q)_2 \max_{\lambda \in \rho(A)} |p_j(\lambda)| \tag{100}$$

, where for the polynomial p_j only the largest eigenvalues (λ) of the spectrum ($\rho(A)$) from matrix A are important. Hence for unitary matrices as in the case of V the condition number of V is 1. Equation 100 means that if the polynomial p_j is small then the there is a fast residual reduction. That's why preconditioning should be used to decrease the spectrum ($\rho(A)$) of matrix A. Let $M^{-1}Ax = M^{-1}b$ be the preconditioned system in the case of left preconditioning and $AM^{-1}x = bM^{-1}$ in the case of right preconditioning and decompose matrix A in $A = M - N$ then the system which should be solved is $(I - B)x = b$ with $B = NM^{-1}$ for right preconditioning. Now formula 97 becomes with the decomposition of $B = XDX^{-1}$, similar like for matrix A:

$$\frac{||r_j||_2}{||r||_2} \le \min_{p_j \in P^j, p_j(1)=1} \kappa(X)_2 \max_{\lambda \in \rho(B)} |p_j(\lambda)|. \tag{101}$$

If the spectral radius of $\rho(I - B)$ is lower than 1 then the problem of $(I - B)x = b$ can be solved otherwise not, for fast convergence the value of $\rho(I - B)$ should be almost zero. Hence preconditioning is not only applied in the way that always the same matrix M is used there can be instead a second solver like in the case of flexible preconditioning (see section 6.1).

7 Experiments related to GMRES and F-GMRES

7.1 Relation between condition number and number of outer iterations of GMRES and Flexible GMRES (F-GMRES)

In this section two different problems and matrices (see formula 102 and 103) are constructed to get control over the condition number ($||.||_2$) (see section 5.3) and to observe the impact of changing this value on the number of outer iterations (m_1) for the GMRES (see algorithm 6) and F-GMRES (see algorithm 11) solver. This diagonal matrix D is mainly built to determine the number of outer iterations (m_1) and the impact of changing the condition number ($||.||_2$) but as well the number of iterations for the inner solver (m_2) of the F-GMRES. The Flexible GMRES (F-GMRES) and the Fault Tolerant GMRES (FT-GMRES) always consist of two solvers just one inner and one outer solver. The outer solver of the Flexible GMRES (F-GMRES) is from line 3 to 24 in algorithm 11, it uses the algorithm 6 of the standard GMRES solver as the inner solver in line 4 for preconditioning.

The aim of this two solver level nesting is that the inner solver should do the main work in lower accuracy or in unreliable mode for the FT-GMRES and the outer solver only has to check for convergence such that a satisfying solution is found. So most of the work should be done by the inner solver whereas the outer solver should only do a few iterations. The problem here is just that the inner solver can be stuck in too much work without converging to a desired solution so there must be always a limit for the number of iterations of the inner solver. This limit (m_2) for the number of iterations for the inner solver can be also controlled this is the second parameter beside for the accuracy of the solution vector w_j. The used matrix A is now a diagonal matrix D to get control over the condition number ($||.||_2$) which has the appearance for two different kinds of problems:

Problem 1 (positive definite):
Matrix with linear separated eigenvalues

$$A = D = \begin{pmatrix} d_1 & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & d_n \end{pmatrix}, \quad (102)$$

Problem 2 (positive definite):
Matrix with linear separated eigenvalues

$$A = D = \begin{pmatrix} \frac{1}{d_1} & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{d_n} \end{pmatrix}. \quad (103)$$

This diagonal matrix D (for problem 1) is mainly built for easier controlling the condition number ($||.||_2$) of a matrix A ($= D$). There are also other possibilities to control this number. This diagonal matrix D for problem 1 is built with a lowest (d_1) and largest value (d_n) where the lowest value is always set to 1 for variable d_1 and for each trial but the greatest value d_n is changed to control the condition number ($||.||_2$). All values in this matrix D are sorted in increasing order with $d_1 < d_2 < d_3 < \cdots < d_n$. The function to compute all other entries between the greatest (λ_n) and lowest value (λ_1) is a linear function with $D(i) = y(i) = k(i-1) + d_1$ for $i = 1 \dots n-1$ but k is computed with $k = (d_n - d_1)/(n-1)$. Thus the eigenvalues λ of the used diagonal matrix D are precisely the $D(i)$s because of the relation $Ax(= Dx) = \lambda x$. So the last eigenvalue (λ_n) is also the greatest value (d_n) of this matrix $A(= D)$ which gives the possibility to control the condition number $\kappa(A)_2$ ($||.||_2$) for each trial. The main contrast between problem 1 and problem 2 is that in the case of problem 2 all values of matrix A are the inverse of matrix A for problem 1 where the condition number is the same. In all experiments for F-GMRES and FT-GMRES the right hand side b is always computed with $b = Ax_{solution}$ where $x_{solution}$ is a vector with only ones, in contrast the outer solver is always initialized with x_{start} a vector with only zeros (see section 12).

7.1.1 Problem 1, relation between condition number and preconditioning by the inner solver of the F-GMRES with random values as initial vector for $w_j(w_0)$

In this section the number of outer iterations (m_1) is examined through changing the number of iterations for the inner solver (m_2) of the F-GMRES as well the condition number ($||.||_2$) of the used matrix D introduced in the section before (see section 7.1). The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of the solution x to solve $Ax = b$ is always lower than 10^{-9} for each trial.

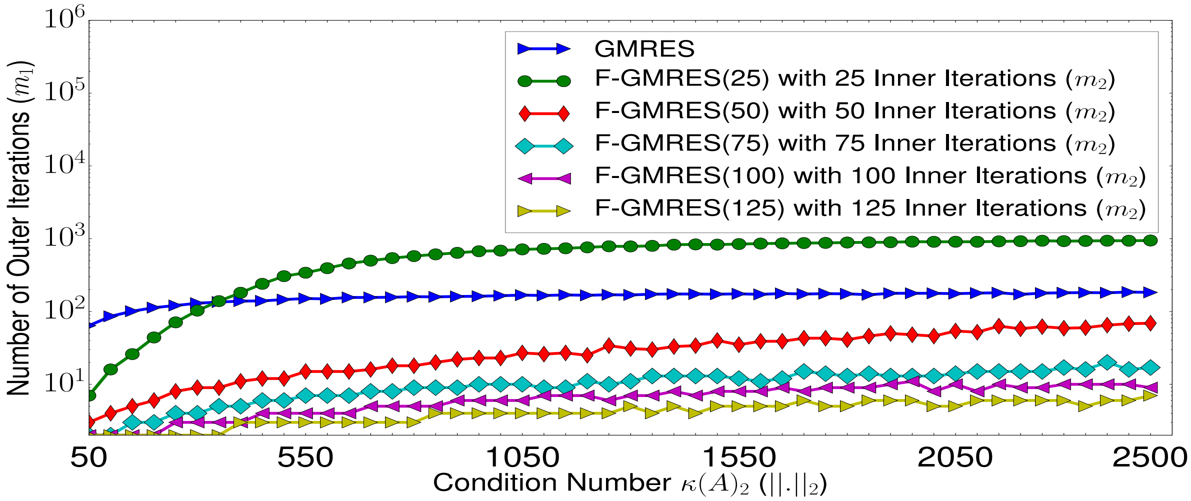


Figure 17 Problem 1, relation between condition number and preconditioning by the inner solver of the F-GMRES. The inner solver is initialized with random values for vector $w_j(w_0)$.

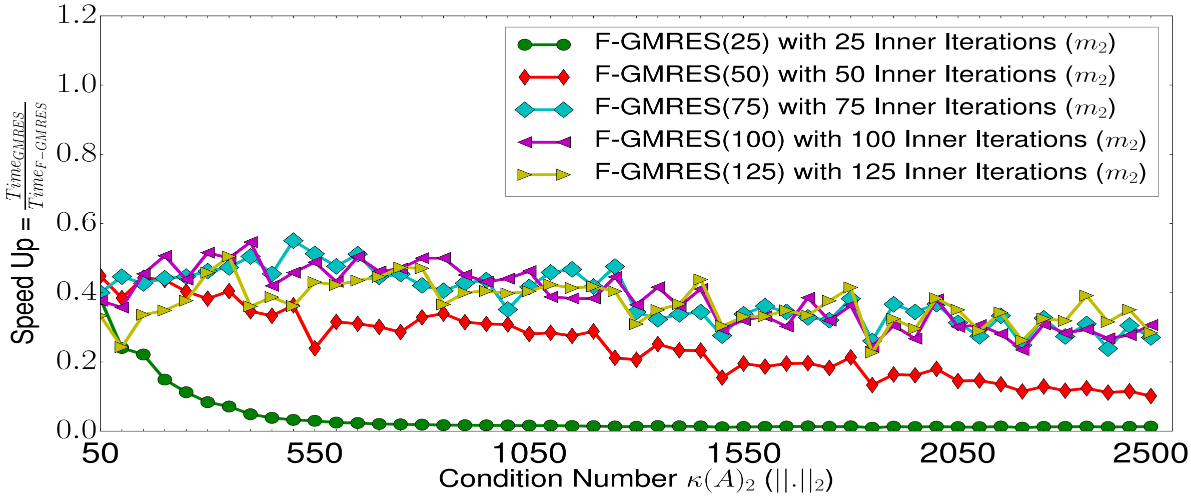


Figure 18 Problem 1, speed up for solution times of F-GMRES against the standard GMRES solver with random values as the initial vector of $w_j(w_0)$, for the inner solver of the F-GMRES.

In figure 17 the number of outer iterations is shown for problem 1 with different condition numbers ($||.||_2$), the inner solver of the F-GMRES is initialized with random values for vector $w_j(w_0)$. In figure 18 the speed ups of the F-GMRES against the GMRES solver are shown for different numbers of iterations of the inner solver (m_2), the F-GMRES is always slower with this initialization. In this case the 2-norm of matrix A is the largest eigenvalue d_n so preconditioning has to be done.

7.1.2 Problem 2, relation between condition number and preconditioning by the inner solver of the F-GMRES with random values as initial vector for $w_j(w_0)$

In this section the number of outer iterations (m_1) is examined through changing the number of iterations for the inner solver (m_2) of the F-GMRES as well the condition number ($||.||_2$) of the used matrix D introduced in the section before (see section 7.1). The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of the solution x to solve $Ax = b$ is always lower than 10^{-9} for each trial.

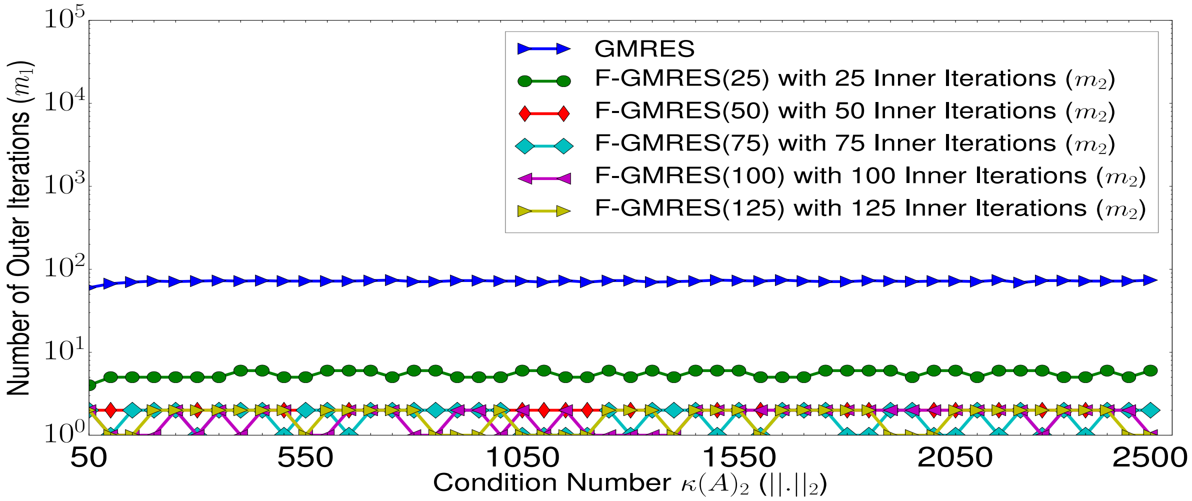


Figure 19 Problem 2, relation between condition number and preconditioning by the inner solver of the F-GMRES. The inner solver is initialized with random values for vector $w_j(w_0)$.

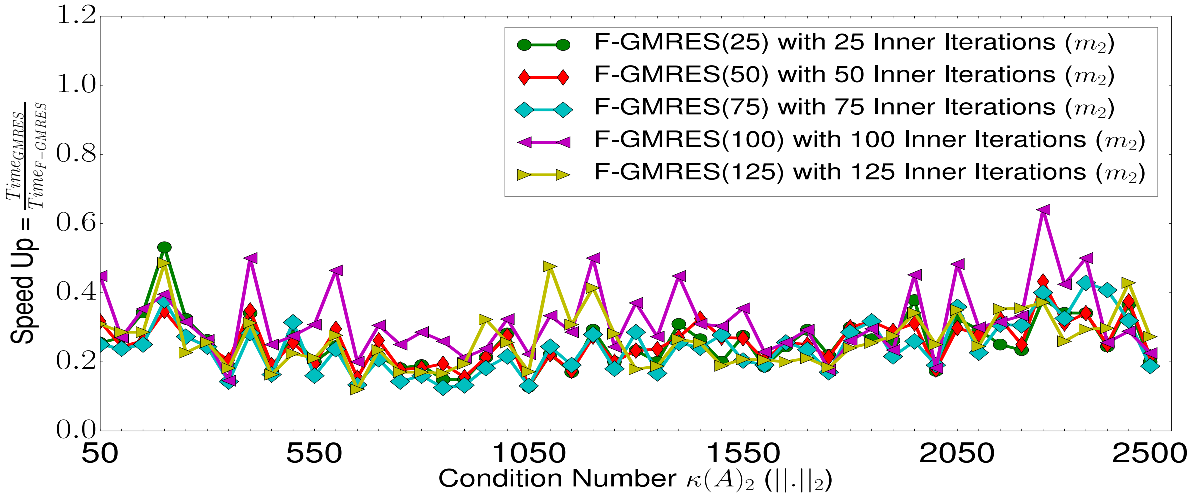


Figure 20 Problem 2, speed up for solution times of F-GMRES against the standard GMRES solver with random values as the initial vector of $w_j(w_0)$, for the inner solver of the F-GMRES.

In figure 19 the number of outer iterations is shown for problem 2 with different condition numbers ($||.||_2$), the inner solver of the F-GMRES is initialized with random values for vector $w_j(w_0)$. In figure 20 the speed ups of the F-GMRES against the GMRES solver are shown for different numbers of iterations of the inner solver (m_2), the F-GMRES is always slower with this initialization. In this case the 2-norm of matrix A is always 1 so preconditioning of F-GMRES is unnecessary.

7.1.3 Problem 1, relation between condition number and preconditioning by the inner solver of the F-GMRES with zero values as initial vector for $w_j(w_0)$

In this section the number of outer iterations (m_1) is examined through changing the number of iterations for the inner solver (m_2) of the F-GMRES as well the condition number ($||.||_2$) of the used matrix D introduced in the section before (see section 7.1). The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of the solution x to solve $Ax = b$ is always lower than 10^{-9} for each trial.

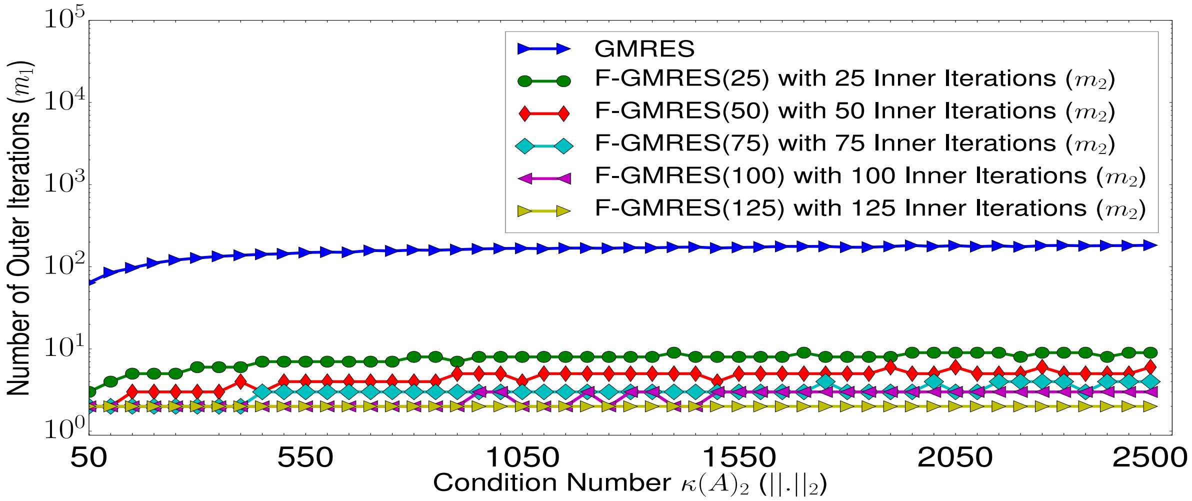


Figure 21 Problem 1, relation between condition number ($\kappa(A)_2$) and preconditioning by the inner solver of the F-GMRES. The inner solver is initialized with zero values for vector $w_j(w_0)$.

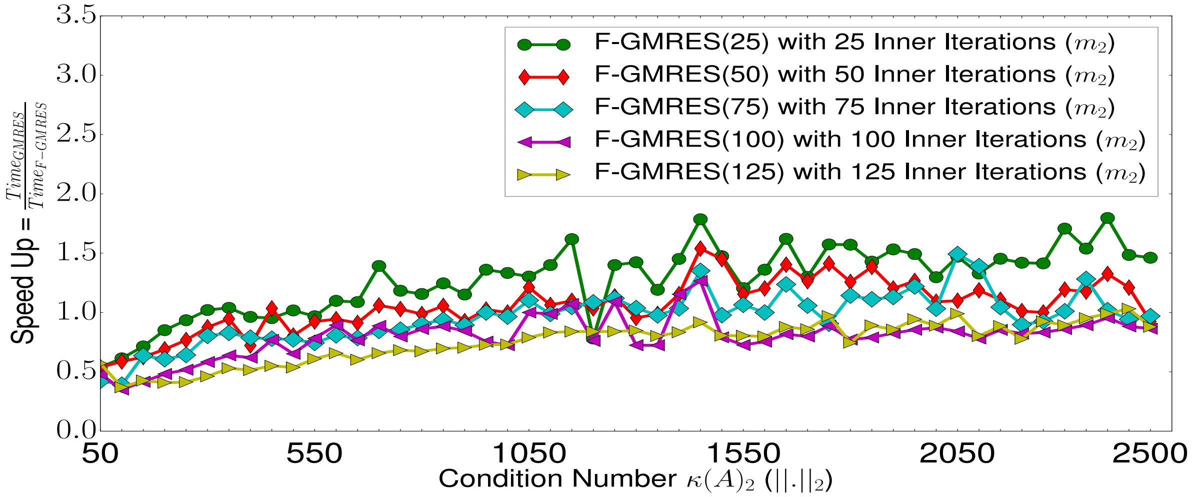


Figure 22 Problem 1, speed up for solution times of F-GMRES against the standard GMRES solver with zero values as the initial vector of $w_j(w_0)$, for the inner solver of the F-GMRES.

In figure 21 the number of outer iterations is shown for problem 1 with different condition numbers ($||.||_2$), the inner solver of the F-GMRES is initialized with zero values for vector $w_j(w_0)$. In figure 22 the speed ups of the F-GMRES against the GMRES solver are shown for different numbers of iterations of the inner solver (m_2), the F-GMRES is faster with this initialization. In this case the 2-norm of matrix A is the largest eigenvalue d_n so preconditioning has to be done.

7.1.4 Problem 2, relation between condition number and preconditioning by the inner solver of the F-GMRES, zero values as initial vector for $w_j(w_0)$

In this section the number of outer iterations (m_1) is examined through changing the number of iterations for the inner solver (m_2) of the F-GMRES as well the condition number ($||.||_2$) of the used matrix D introduced in the section before (see section 7.1). The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of the solution x to solve $Ax = b$ is always lower than 10^{-9} for each trial.

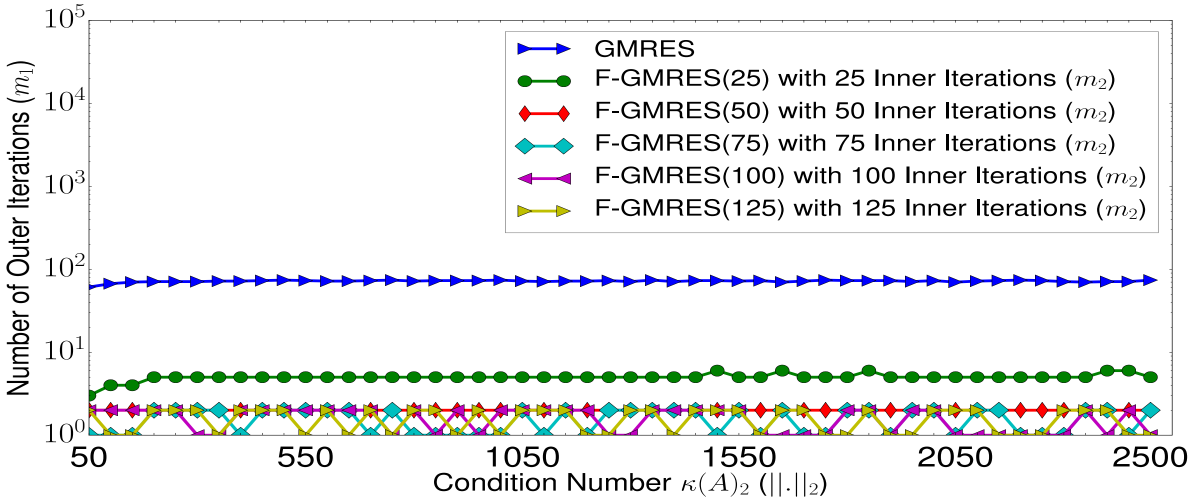


Figure 23 Problem 2, relation between condition number ($\kappa(A)_2$) and preconditioning by the inner solver of the F-GMRES. The inner solver is initialized with zero values for vector $w_j(w_0)$.

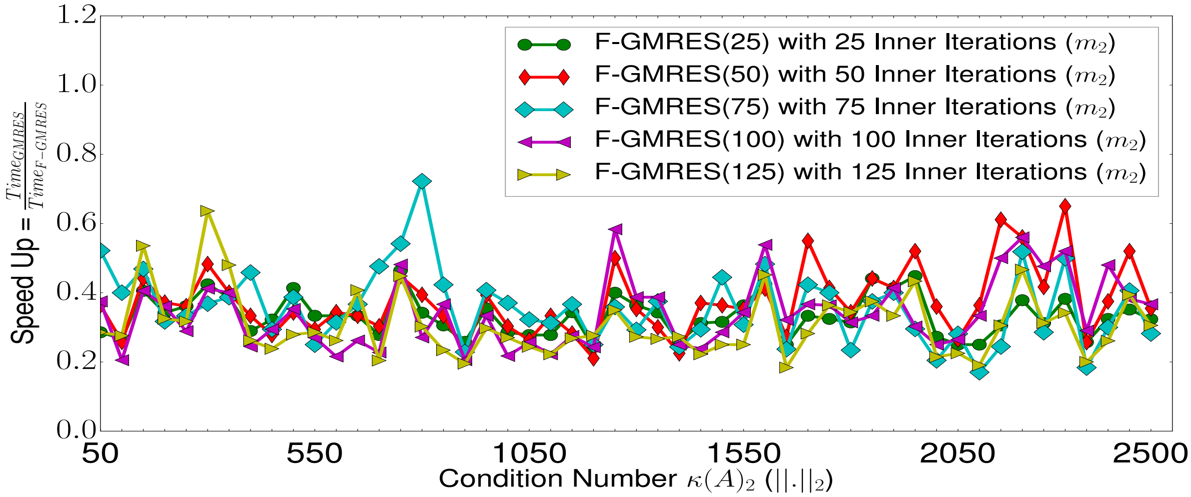
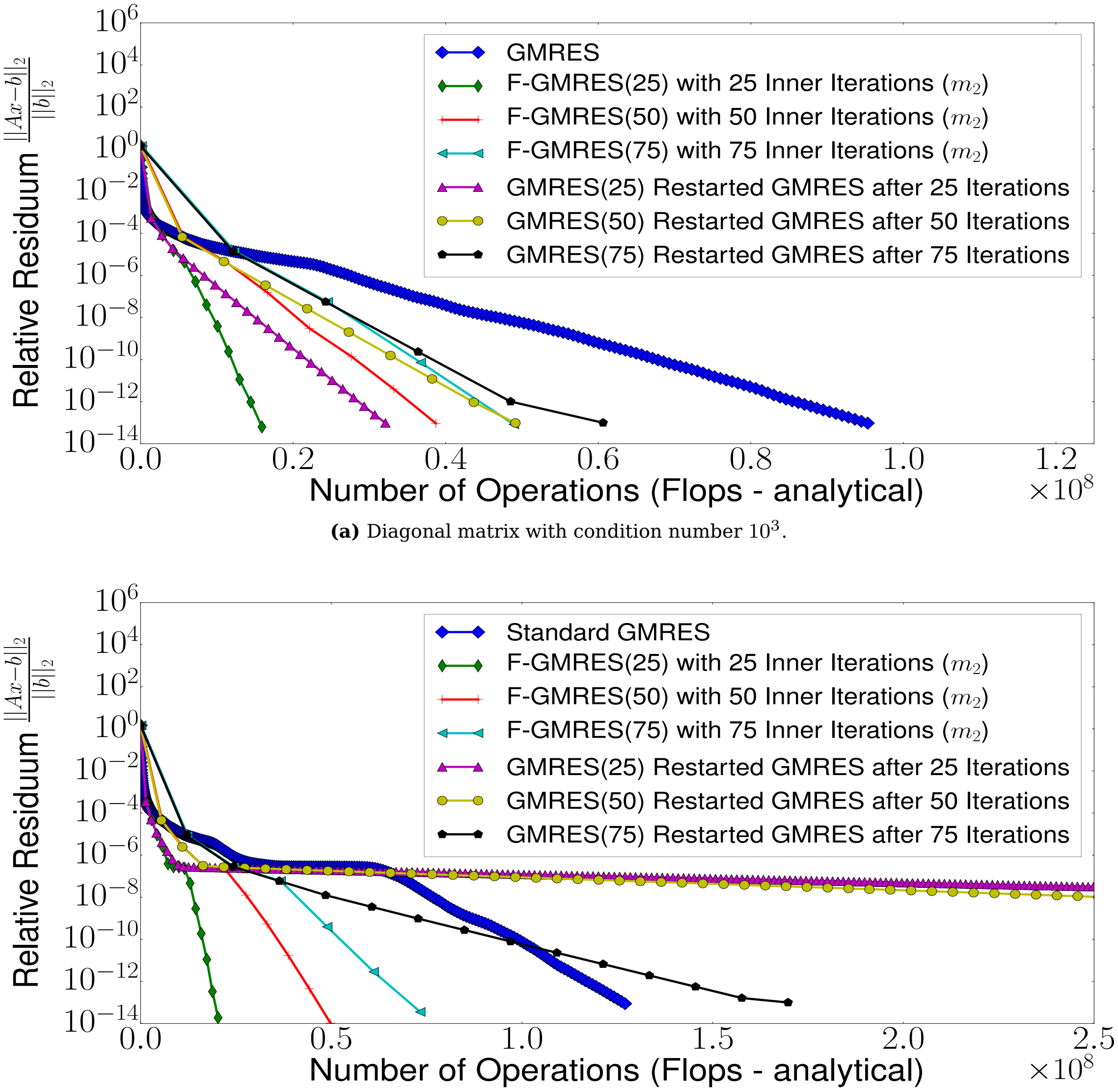


Figure 24 Problem 2, speed up for solution times of F-GMRES against the standard GMRES solver with zero values as the initial vector of $w_j(w_0)$, for the inner solver of the F-GMRES.

In figure 23 the number of outer iterations is shown for problem 2 with different condition numbers ($||.||_2$), the inner solver of the F-GMRES is initialized with zero values for vector $w_j(w_0)$. In figure 24 the speed ups of the F-GMRES against the GMRES solver are shown for different numbers of iterations of the inner solver (m_2), the F-GMRES is always slower with this initialization. In this case the 2-norm of matrix A is always 1 so preconditioning of F-GMRES is unnecessary.

7.2 Different test cases for GMRES and Flexible GMRES (F-GMRES)

7.2.1 GMRES and F-GMRES on different diagonal matrices - Inner solver initialized with zero values



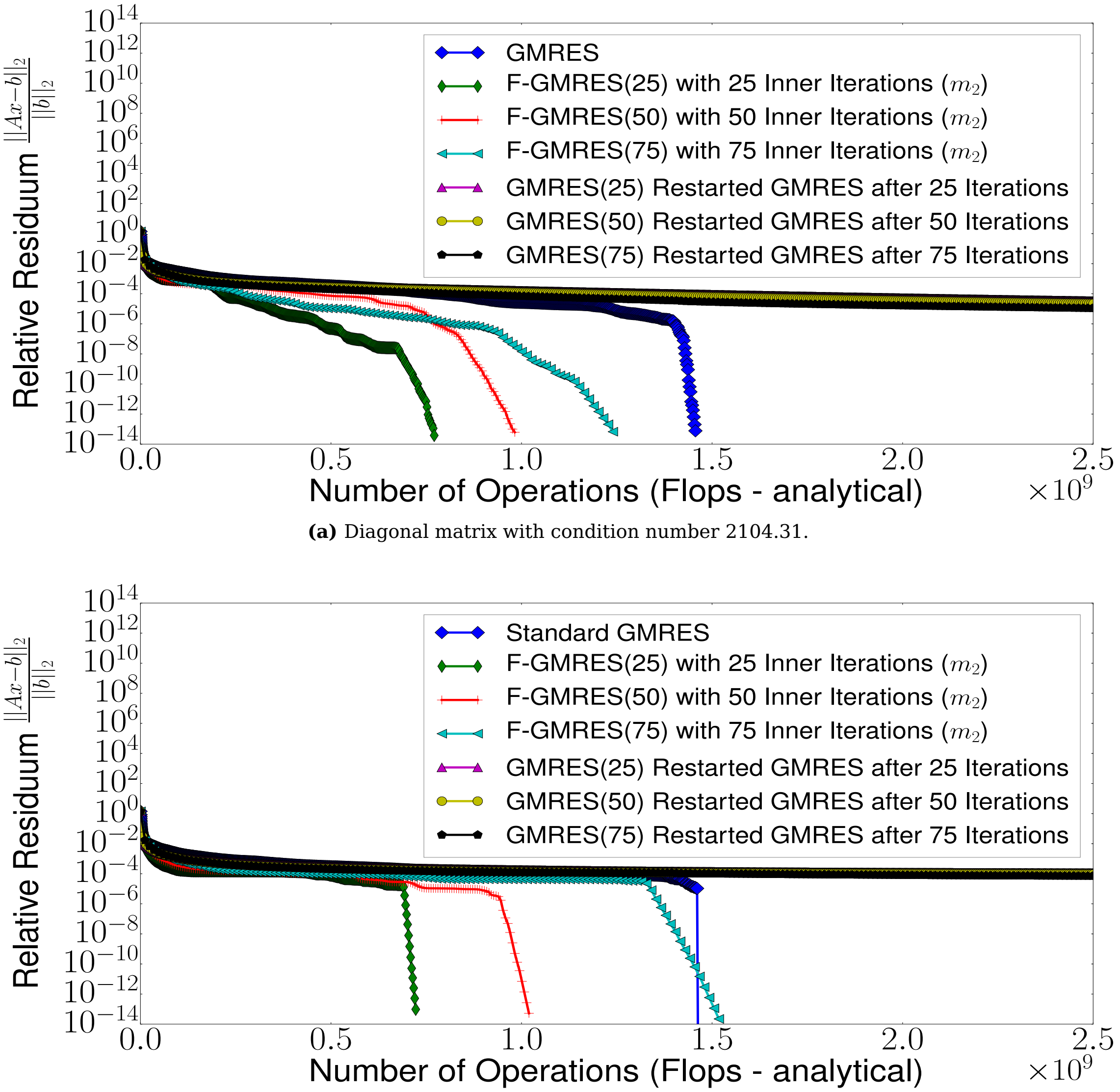
(a) Diagonal matrix with condition number 10^3.

(b) Diagonal matrix with condition number 10^5.

Figure 25 Residuum curves of (F-)GMRES for different matrices with linear separated values.

In figure 25 the condition number ($||.||_2$) is changed from 10^3 to 10^5 for a diagonal matrix D. All used diagonal matrices D are built like in section 7.1 for problem 1 (see formula 102) which means that all values are linear separated from each other one. In the case of a condition number ($||.||_2$) 1000 the first value (d_1) is set to 1 whereas the last value (d_n) is set to 1000. Figure 25 shows that the Flexible GMRES (F-GMRES) outperforms the standard GMRES solver on this kind of problem for a positive definite matrix (see formula 29) but by also considering the needed number of operations to converge to a specific residuum ($||r||_2 = ||Ax - b||_2/||b||_2$). The inner solver of the F-GMRES is initialized with zero values for vector $w_j (w_0)$ (line 4 of algorithm 11). Restarting the GMRES and reusing the old solution x from the previous computation is not a good approach.

7.2.2 GMRES and F-GMRES on different diagonal matrices with random values - Inner solver initialized with zero values



(a) Diagonal matrix with condition number 2104.31.

(b) Diagonal matrix with condition number 5870.31.

Figure 26 Residuum curves of (F-)GMRES for different matrices with random values.

In figure 26 a diagonal matrix D is used where for each trial all its values are randomized this will lead to different condition numbers ($||.||_2$). All values of this diagonal matrix D are randomly computed such that matrix D is an arbitrary matrix. Figure 26 shows that the Flexible GMRES (F-GMRES) outperforms the standard GMRES solver on this kind of problem for a non-positive definite matrix (see formula 29) but by also considering the needed number of operations to converge to a specific residuum ($||r||_2 = ||Ax - b||_2/||b||_2$). The inner solver of the F-GMRES is initialized with zero values for vector $w_j(w_0)$ (line 4 of algorithm 11). Restarting the GMRES and reusing the old solution x from the previous computation is not a good approach because of stagnation for the accuracy of vector x. Figure 26 also shows that the F-GMRES has good properties to solve $Ax = b$.

8 Injecting errors in Fault Tolerant GMRES (FT-GMRES)

8.1 Introduction

The purpose of the FT-GMRES solver is that the main work should be done in unreliable mode (line 4 of algorithm 14) whereas the outer solver (line 14 - 19 of algorithm 14) should only check for convergence within a few iterations. For testing the behavior of the FT-GMRES in the presence of a single fault two types of perturbations are injected during the orthogonalization process for computing $h_{i,j}$ (line 5 - 10 of algorithm 6) in the inner solver of the FT-GMRES (line 4 of algorithm 14). Both possibilities are based on the floating point representation where both types of faults should give a baseline for other types of faults as well for further experiments.

The total number of iterations for the outer solver (m_1) of the F(T)-GMRES is observed which is needed to compute a solution x for solving $Ax = b$ with a specific accuracy and without a fault, afterwards the same problem is solved but during these computations values of $h_{i,j}$ from the orthogonalization process are disturbed like in the case of a single fault for a single position of $h_{i,j}$ such that some perturbations are induced (line 5 - 10 of algorithm 6). The following two types of perturbations ($E_{perturbed}$) are injected during the orthogonalization process (line 5 - 10 of algorithm 6) which can be caused by a single fault, both values are based on the floating point representation:

Type	Value
Error Type 1	$E_{perturbed} = 10^{150}$
Error Type 2	$E_{perturbed} = 10^{-300}$

Figure 27 Different error types which are used for observing the behavior of the FT-GMRES.

The total perturbation depends on the used norm ($||.||_2$) of matrix A because the maximum propagation which can be caused is based on the entries of matrix A, values of matrix A with lower magnitude will give less change whereas values with a large magnitude of the used matrix A will give a higher error propagation. Faulting in this master work is mainly done during the orthogonalization process for computing $h_{i,j}$ (line 5-10 of algorithm 6) in the inner solver (line 4 of algorithm 14) of the FT-GMRES. For each new iteration j of the inner solver a matrix vector product is performed ($v_{j+1} = Aq_j$) which is also done in algorithm 6 and line 4. This operation gives the upper bound for injecting a perturbation in vector v_{j+1} but as well $h_{i,j}$.

For each sparse matrix vector product following operation is performed $v_{j+1} = Aq_j$ and by using the norm ($||.||_2$) it follows that $||v_{j+1}||_2 \leq ||A||_2||q_j||_2$. So if the vector q_j is disturbed because of a fault then the maximum perturbation which can be injected in vector v_{j+1} is $||v_{j+1}||_2 = ||E_{injected}||_2 \leq ||A||_2 \times ||E_{perturbed}||_2$. Therefore the product of $||A||_2|| \times ||E_{perturbed}||_2$ gives the upper bound for a single fault in vector v_{j+1}. Injecting NaNs or Infs will reveal nothing therefore it is recommended to use floating point values as perturbations for variable $E_{disturbed}$. Hence in general the vector $||q_j||_2$ is set to 1 ($||q||_2 = 1$) because of the normalized residuum ($\frac{||r_0||_2}{\beta} = ||1||_2 = ||q_1||_2$) which is computed in the standard GMRES solver, this fact can be also used as an error detector.

$\hat{H}(i,j)$ is mainly related to the positions of the built Hessenberg matrix whereas $h_{i,j}$ is related to the positions of the orthogonalization process. For the very first positions of $h_{i,j}$ (line 5-10 of algorithm 6) faulting during the matrix vector product is the same as faulting during the orthogonalization process. For each position of $h_{i,j}$ with $i = 1$ a matrix vector product is applied for every j such that a fault can be also injected with $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times E_{perturbed}$ over the whole orthogonalization process with a disturbed $h_{i,j}$ ($\bar{h}_{i,j}$) including the sparse matrix vector product.

8.2 Error Injection Methology

The aim of GMRES as well F(T)-GMRES is to build a Hessenberg matrix $\hat{H}$ to solve the equation system $Ax = b$. The Hessenberg matrix $\hat{H}$ is build during the orthogonalization process through computing $h_{i,j}$ (in line 5-10 of algorithm 6) with the Gram Schmidt (GS) or the Modified Gram Schmidt process (MGS) which ensures more stability. In the case of a symmetric problem this orthogonalization process leads to a tridiagonal matrix (see formula 105) which is a special case of an upper Hessenberg matrix $\hat{H}$ whereas for a non-symmetric system an upper Hessenberg matrix will be built (see formula 104). Formular 104 and 105 show these matrices before applying the Givens rotation (see section 5) which eliminates values. Symmetric problems are problems if rows and columns are exchanged such that matrix A is unchanged ($A = A^T$). In the case of a diagonal matrix (symmetric), matrix $\hat{H}$ will be as well a tridiagonal matrix [38]. If some of those entries are disturbed a penalty in solving time should be observed for the FT-GMRES solver (see section 6.2.4).

Upper Hessenberg matrix

$$\hat{H}(i,j) = \begin{array}{cc} & \rightarrow \text{Iteration (Index) } j \\ \downarrow \text{Iteration (Index) } i & \begin{bmatrix} x & x & x & x \\ x & x & x & x \\ 0 & x & x & x \\ 0 & 0 & x & x \end{bmatrix} \end{array}, \tag{104}$$

Tridiagonal (Hessenberg) matrix

$$\hat{H}(i,j) = \begin{array}{cc} & \rightarrow \text{Iteration (Index) } j \\ \downarrow \text{Iteration (Index) } i & \begin{bmatrix} x & x & 0 & 0 \\ x & x & x & 0 \\ 0 & x & x & x \\ 0 & 0 & x & x \end{bmatrix} \end{array}. \tag{105}$$

A worst case scenario should be observed if one of the first entries of the Hesserberg matrix ($\hat{H}_{1,j}$) is disturbed since all other values afterwards are still affected. In further computations faulting is also done on the last positions of the Hessenberg matrix ($\hat{H}_{i=j,j}$). There are mainly two test cases like faulting on the first Modified Gram Schmidt iteration ($h_{1,j}$) and faulting on the last Modified Gram Schmidt iteration ($h_{i=j,j}$) which are observed in this work for a single perturbation during the orthogonalization process for computing $h_{i,j}$ in the inner solver of the FT-GMRES:

1. Faulting on the first MGS-iteration ($\overline{h}_{i=1,j}$): Faulting is done for positions of $h_{i=1,j}$ during the orthogonalization process ($h_{i,j}$) whereas the position j is changed for each single trial.
2. Faulting on the last MGS-iteration ($\overline{h}_{i=j,j=i}$): Faulting is done on the last positions of $h_{i=j,j=i}$ during orthogonalization ($h_{i,j}$), it is totally different from faulting on the first iteration.

Figure 28 Two possibilities of faulting during the orthogonalization process in the FT-GMRES.

In the following experiments the number of outer iterations (m_1) is determined in a *failure-free* run. After the first run the computation is repeated (for all outer iterations) like in section 9.1 for some position of $h_{i,j}$ in the inner solver of the FT-GMRES where the Hessenberg matrix $\hat{H}$ should be disturbed. Single values of the Hessenberg matrix $\hat{H}(i,j)$ are set to be faulty because of a single disturbance during the orthogonalization process ($h_{i,j}$). For these experiments there are also two possible cases like faulting on the first MGS ($\overline{h}_{i=1,j}$) and on the last MGS ($\overline{h}_{i=j,j}$) iteration shown in table 28. On these positions where single values of $\hat{H}$ are disturbed there are also two possibilities of faulting with two types of errors shown in table 27. For each trial with a single fault where one of these values of the Hessenberg matrix $\hat{H}$ is disturbed the number of outer iterations (m_1) is observed to converge to the same relative residuum ($||r||_2 = ||Ax - b||_2 / \ ||b||_2$) for example 10^{-9} as without faulting, it means that always the same accuracy with/without a fault must be achieved.

9 Experiments related to Fault Tolerant GMRES (FT-GMRES)

9.1 Faulting on the first and last Modified Gram Schmidt (MGS) iteration

9.1.1 Faulting on the first MGS-iteration - Inner solver initialized with random values

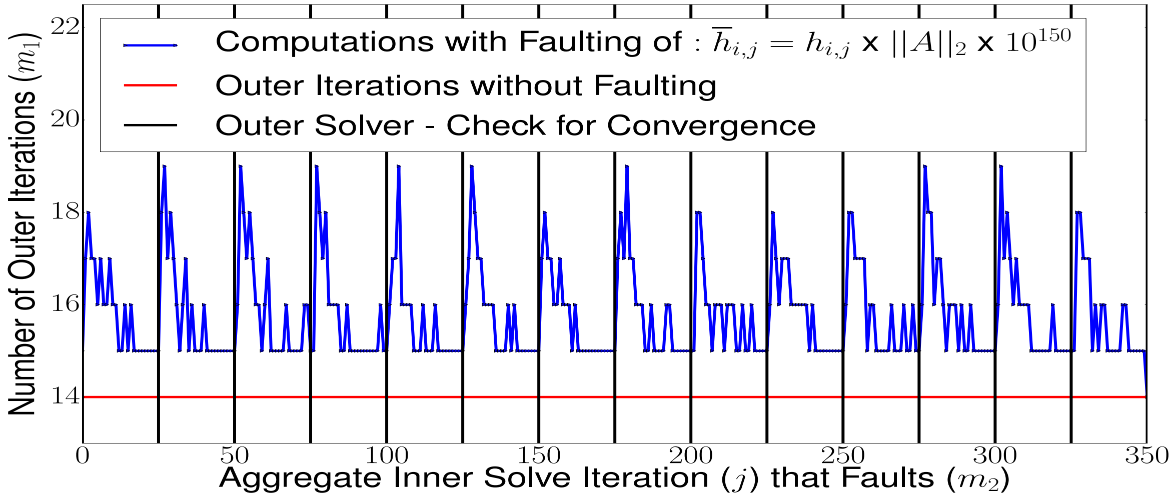


Figure 29 Faulting with a single disturbance and error type 1 on the first MGS-iteration ($h_{i=1,j}$) during solving a diagonal matrix with condition number 142.5 for $i = 1$ and $j = 1, \ldots, 25$.

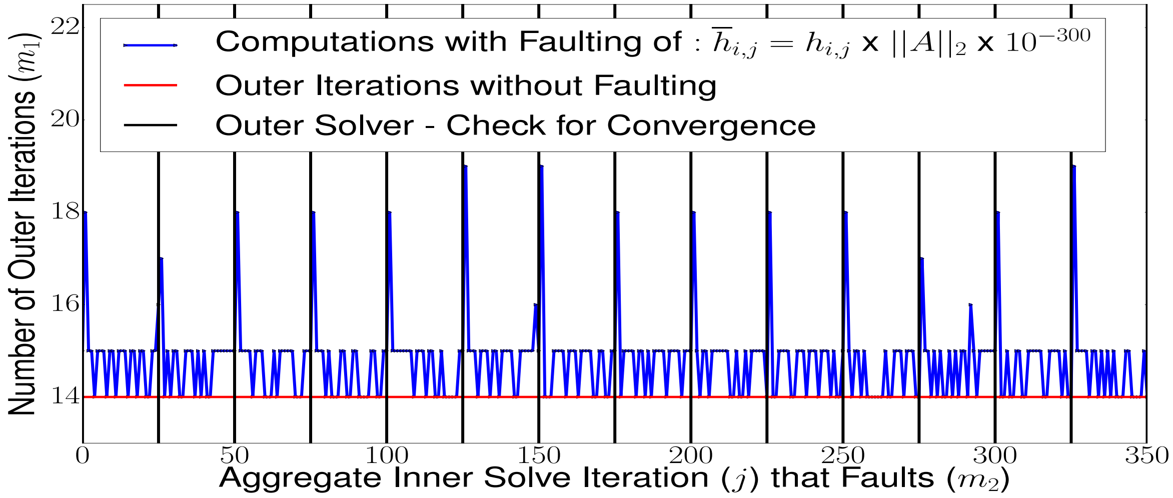


Figure 30 Faulting with a single disturbance and error type 2 on the first MGS-iteration ($h_{i=1,j}$) during solving a diagonal matrix with condition number 142.5 for $i = 1$ and $j = 1, \ldots, 25$.

In figure 29 faulting with error type 1 for a single perturbation of 10^{150} on $h_{i=1,j}$ is more critical than faulting with 10^{-300} and error type 2 in figure 30 for single faults. In figure 30 there are only some specific positions namely $\hat{H}(1,2)$ where some overhead is caused. The inner solver of the FT-GMRES is initialized with random values for vector $w_j(w_0)$, it does 25 iterations ($j = 1, \ldots, 25$) for each execution. The used matrix A is built like in section 7.1 for problem 1 (see formula 102). The relative residuum ($||r||_2 = ||Ax - b||_2 / \, ||b||_2$) is below 10^{-9} for all trials and solutions of x. Vertical bars in figure 29 and 30 indicate the start of a new inner solve of the FT-GMRES.

9.1.2 Faulting on the last MGS-iteration - Inner solver initialized with random values

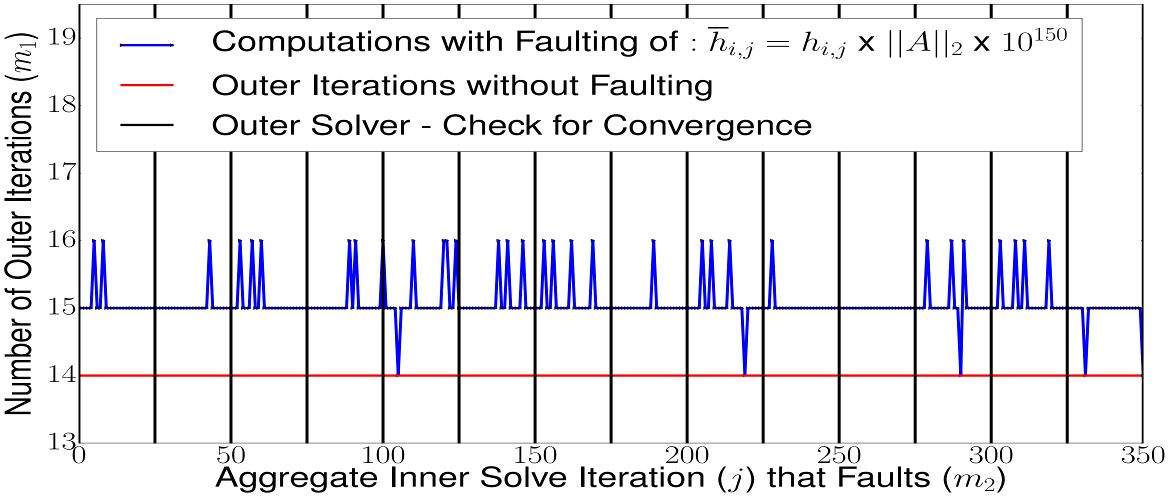


Figure 31 Faulting with a single disturbance and error type 1 on the last MGS-iteration ($h_{i=j,j}$) during solving a diagonal matrix with condition number 142.5 for $i = 1$ and $j = 1, \ldots, 25$.

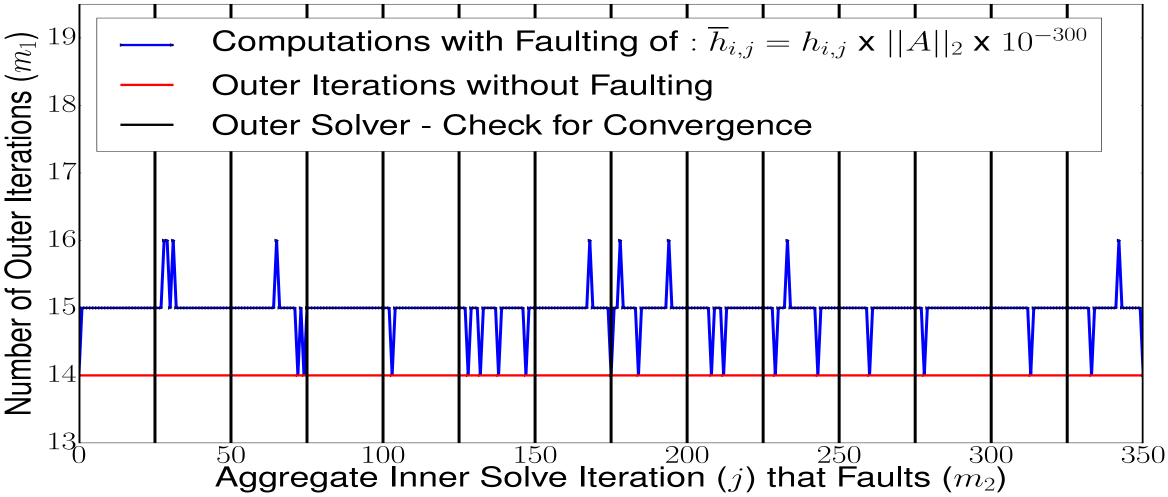


Figure 32 Faulting with a single disturbance and error type 2 on the last MGS-iteration ($h_{i=j,j}$) during solving a diagonal matrix with condition number 142.5 for $i = 1$ and $j = 1, \ldots, 25$.

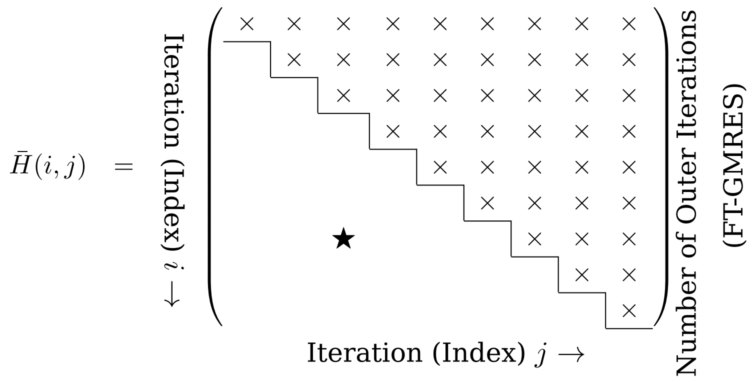
Single faults during the last positions of $h_{i,j}$ ($h_{i=j,j}$) have nearly no affect on the number of outer iterations presented in figure 31 and 32 regardless of the size of the perturbation. It means that faulting on the last positions of $h_{i,j}$ is almost uncritical for faults on $h_{i=j,j}$. The inner solver of the FT-GMRES is initialized with random values for vector $w_j(w_0)$, it does 25 iterations ($j = 1, \ldots, 25$) for each execution. The used matrix A is built like in section 7.1 for problem 1 (see formula 102). The relative residuum ($||r||_2 = ||Ax - b||_2 / ||b||_2$) is below 10^{-9} for all trials and solutions of x. Vertical bars in figure 31 and 32 indicate the start of a new inner solve of the FT-GMRES.

9.2 Testing of all positions of the orthogonalization with single faults

In order to get a point of view for faults on all positions during the orthogonalization process ($h_{i,j}$) a matrix $\bar{H}(i,j)$ can be build instead which shows the number of outer iterations (m_1) for a single fault on a specific position during building the Hessenberg matrix $\hat{H}(i,j)$ in the inner solver of the FT-GMRES (line 4 of algorithm 14). This matrix $\bar{H}(i,j)$ can be seen as a pseudo Hessenberg matrix where all values of $\bar{H}(i,j)$ are not those values of the orthogonalization process for computing $h_{i,j}$ (from line 5-10 of algorithm 6) but the achieved number of outer iterations (m_1) for the FT-GMRES.

This matrix $\bar{H}(i,j)$ is shown in formula 106, faulting can be only applied on values of the upper part of the pseudo Hessenberg matrix ($\bar{H}(i,j)$) respecitive $\bar{h}_{i,j}$ marked with "$\times$" whereas the lower part shows the number of outer iterations (m_1) without faulting marked with "$*$". At positions where the index i is greater than j ($i > j$) no faulting can be applied in algorithm 6 for the given loop variables i and j. On positions where $i \leq j$ faulting with any kind of perturbation ($E_{perturbed}$) can be applied like $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times E_{perturbed}$ (from table 27). Matrix $\bar{H}(i,j)$ shows the maximum occurrence for the number of outer iterations (m_1) in the presence of a single fault on a specific position of $h_{i,j}$ in the inner solver of the FT-GMRES (see algorithm 14 and line 4).

Matrix $\bar{H}(i,j)$ which shows the number of outer iterations (m_1) in the presence of a single fault for each position of $h_{i,j}$ in the inner solver of the FT-GMRES:



(106)

Figure 33 Matrix $\bar{H}(i,j)$ which visualizes where faulting is applied during the orthogonalization process ($h_{i,j}$) in the inner solver of the FT-GMRES with the according outer iterations (m_1).

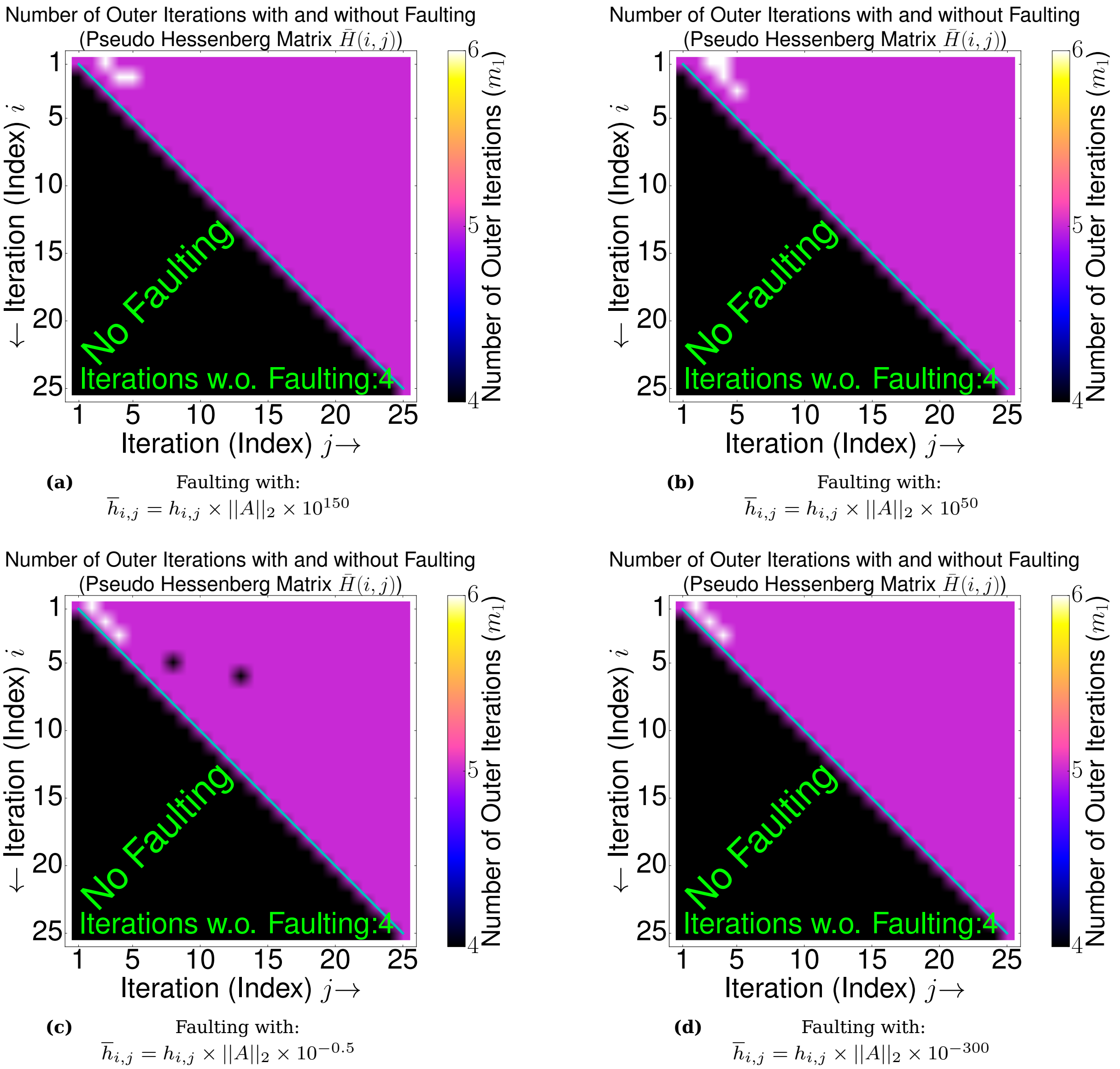
In formula 106 each position marked with "$\times$" shows at which position it is possible to apply a fault with a perturbation ($E_{perturbed}$) such that $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times E_{perturbed}$. This comes mainly from the fact that the loops in algorithm 6 for the index variables i and j describe an upper Hessenberg matrix $\hat{H}(i,j)$. Faulting can be only applied if $i \leq j$, these positions are marked with "$\times$".

Formula 106 shows the maximum number of outer iterations in the presence of a single fault in matrix $\bar{H}(i,j)$. All other positions of the orthogonalization process ($h_{i,j}$) where faulting is not applied are marked with "*". In figure 106 a matrix is shown with size of 9×9, so there are 9 iterations at all for the inner solver of the FT-GMRES (line 4 of algorithm 14). The related overhead is defined as:

$$Overhead = \frac{\text{Maximum number of outer iterations (in the presence of a fault)}}{\text{Number of outer iterations without a fault } +1}. \qquad (107)$$

This additional iteration ($+1$) is mainly for the next trials with a single fault because the used solver (FT-GMRES) has to achieve the same relative residuum as for the first trial without a fault (F-GMRES). For each trial a relative residuum ($||r||_2 = ||Ax - b||_2)/||b||_2$) of 10^{-9} must be achieved.

9.2.1 Faulting on a matrix with low condition number - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

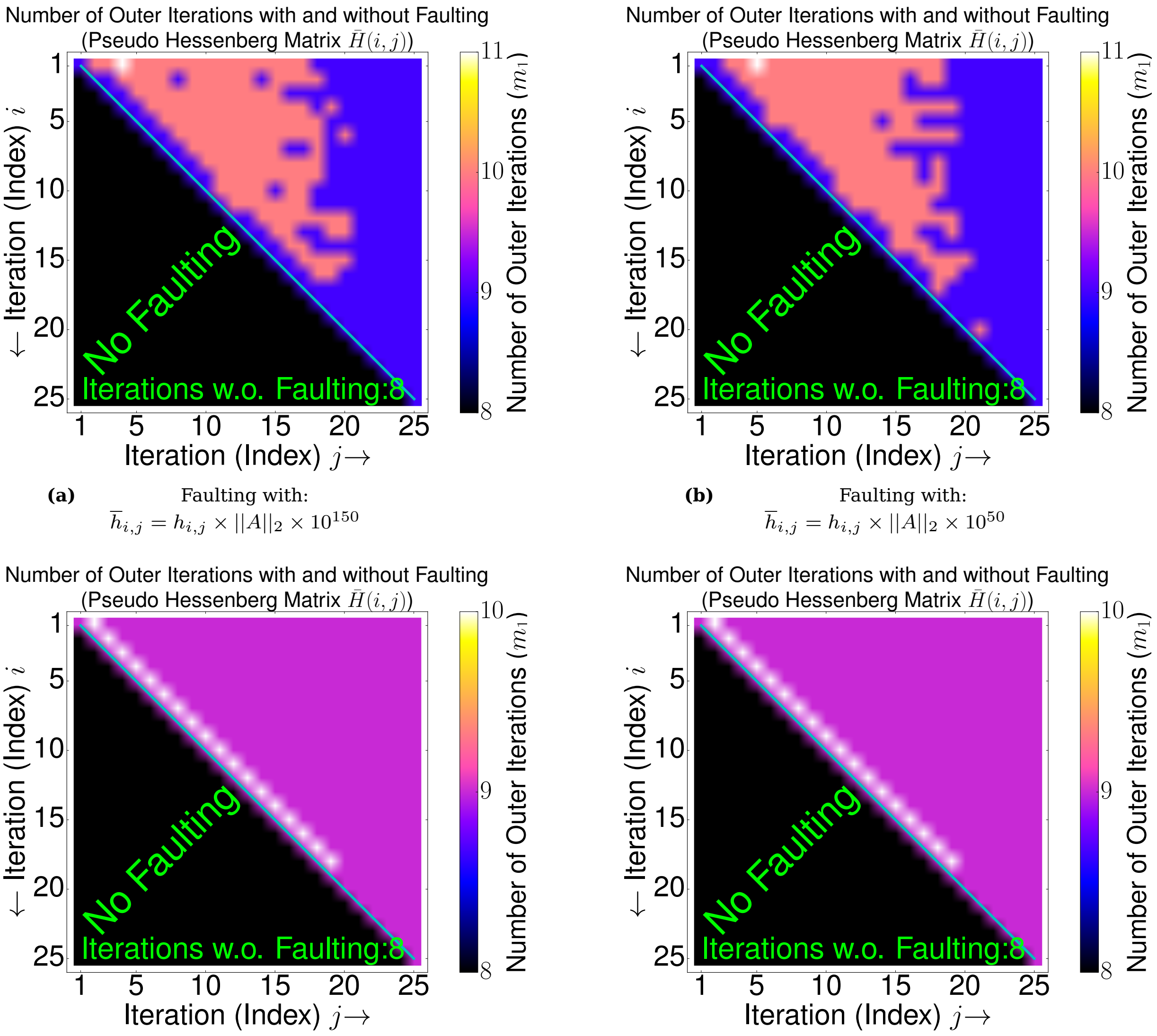
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

Figure 34 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving a diagonal matrix with condition number 142.5.

In figure 34 faulting with different types of errors is presented, it shows that faulting with large perturbations (10^{150} and 10^{50}) is critical on some positions of $\bar{H}(i,j)$. Furthermore faulting with perturbations of $10^{-0.5}$ and 10^{-300} like in figure 34c and 34d is only critical on some specific positions ($\bar{H}(1,2)$, $\bar{H}(2,3)$, $\bar{H}(3,4)$, $\ldots$). The worst overhead is 20 % ($= \frac{6}{5}$) in the presence of a single fault. In figure 34 are 4 outer iterations (m_1) needed for the F(T)-GMRES without a fault. The inner solver is initialized with zero values for vector $w_j(w_0)$. The used matrix A for figure 34 is built like in section 7.1 for problem 1, the relative residuum is below 10^{-9} for all computations.

9.2.2 Faulting on a matrix with high condition number - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

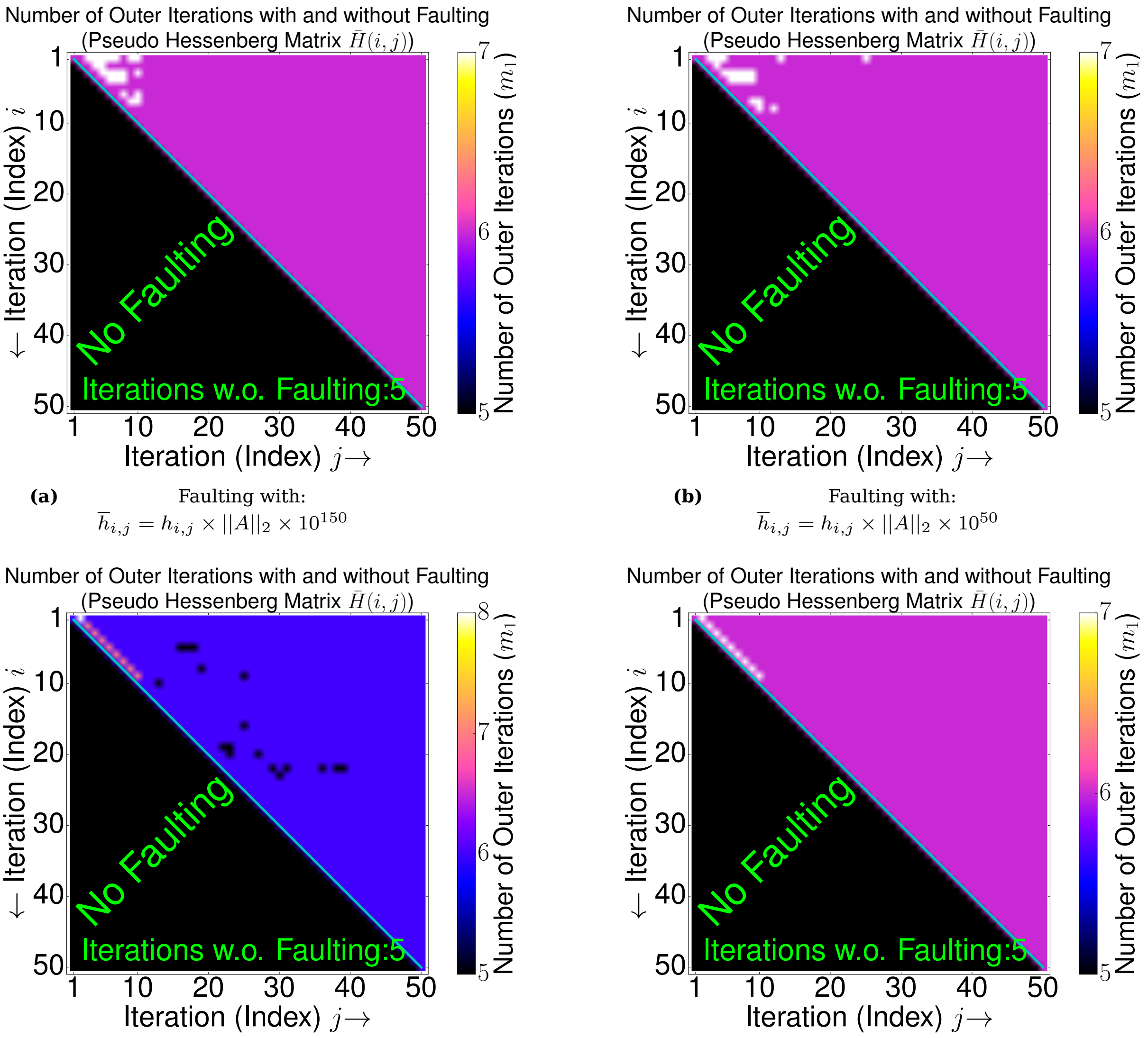
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

Figure 35 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving a diagonal matrix with condition number 1425.

In figure 35 faulting with different types of errors is presented, it shows that faulting with large perturbations (10^{150} and 10^{50}) is critical on some positions of $\bar{H}(i,j)$. Furthermore faulting with perturbations of $10^{-0.5}$ and 10^{-300} like in figure 35c and 35d is only critical on some specific positions ($\bar{H}(1,2)$, $\bar{H}(2,3)$, $\bar{H}(3,4)$, $\ldots$). The worst overhead is 22 % ($= \frac{11}{9}$) in the presence of a single fault. In figure 35 are 8 outer iterations (m_1) needed for the F(T)-GMRES without a fault. The inner solver is initialized with zero values for vector $w_j(w_0)$. The used matrix A for figure 35 is built like in section 7.1 for problem 1, the relative residuum is below 10^{-9} for all computations.

9.2.3 Faulting on a matrix with high condition number - Inner solver initialized with zero values (and with 50 iterations for the inner solver)



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

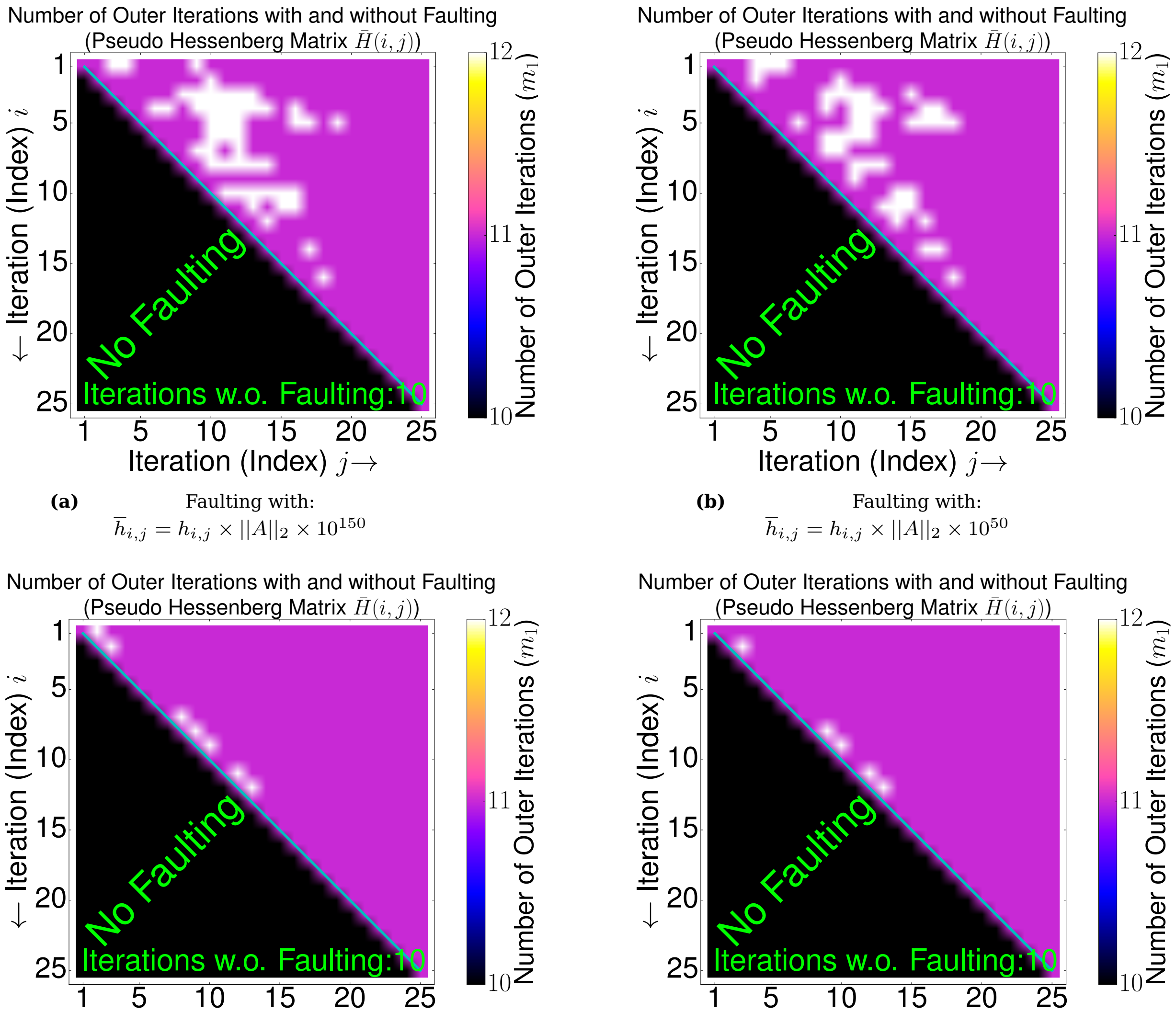
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

Figure 36 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving a diagonal matrix with condition number 1425.

In figure 36 faulting with different types of errors is presented, it shows that faulting with large perturbations (10^{150} and 10^{50}) is critical on some positions of $\bar{H}(i,j)$. Furthermore faulting with perturbations of $10^{-0.5}$ and 10^{-300} like in figure 36c and 36d is only critical on some specific positions ($\bar{H}(1,2)$, $\bar{H}(2,3)$, $\bar{H}(3,4)$, $\ldots$). The worst overhead is 33 % ($= \frac{8}{6}$) in the presence of a single fault. In figure 36 are 5 outer iterations (m_1) needed for the F(T)-GMRES without a fault. The inner solver is initialized with zero values for vector $w_j(w_0)$. The used matrix A for figure 36 is built like in section 7.1 for problem 1, the relative residuum is below 10^{-9} for all computations.

9.2.4 Faulting on a matrix with very high condition number - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

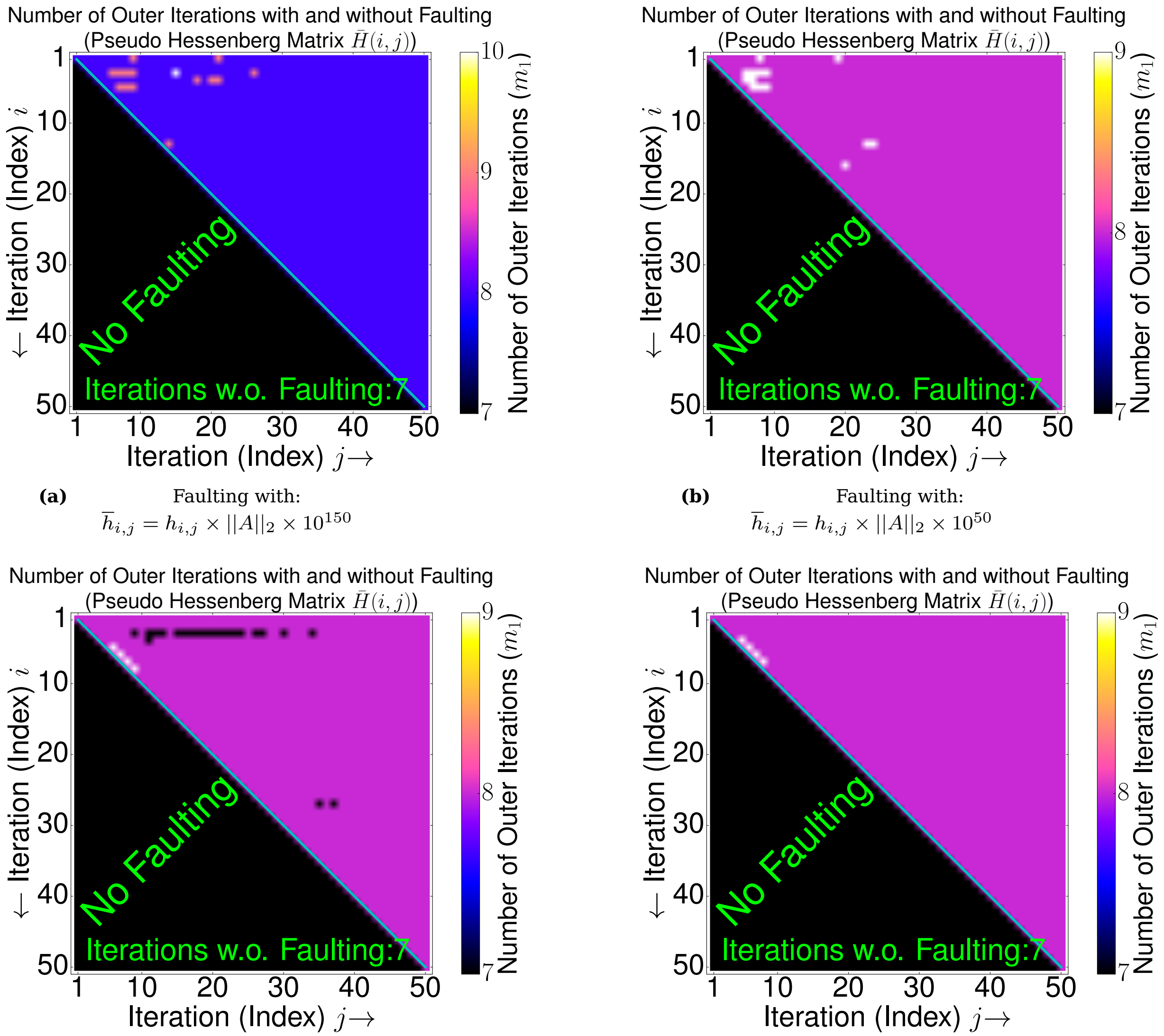
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

Figure 37 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving a diagonal matrix with condition number 14250.

In figure 37 faulting with different types of errors is presented, it shows that faulting with large perturbations (10^{150} and 10^{50}) is critical on some positions of $\bar{H}(i,j)$. Furthermore faulting with perturbations of $10^{-0.5}$ and 10^{-300} like in figure 37c and 37d is only critical on some specific positions ($\bar{H}(1,2)$, $\bar{H}(2,3)$, $\bar{H}(3,4)$, $\ldots$). The worst overhead is 9 % ($= \frac{12}{11}$) in the presence of a single fault. In figure 37 are 10 outer iterations (m_1) needed for the F(T)-GMRES without a fault. The inner solver is initialized with zero values for vector $w_j(w_0)$. The used matrix A for figure 37 is built like in section 7.1 for problem 1, the relative residuum is below 10^{-9} for all computations.

9.2.5 Faulting on a matrix with very high condition number - Inner solver initialized with zero values (and with 50 iterations for the inner solver)



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

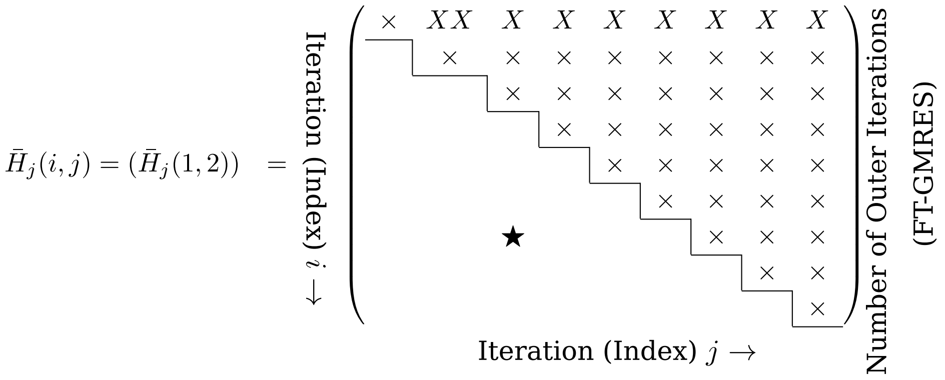
(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

Figure 38 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving a diagonal matrix with condition number 14250.

In figure 38 faulting with different types of errors is presented, it shows that faulting with large perturbations (10^{150} and 10^{50}) is critical on some positions of $\bar{H}(i,j)$. Furthermore faulting with perturbations of $10^{-0.5}$ and 10^{-300} like in figure 38c and 38d is only critical on some specific positions ($\bar{H}(7,8)$, $\bar{H}(8,9)$, $\bar{H}(10,11)$, $\ldots$). The worst overhead is 25 % ($= \frac{10}{8}$) in the presence of a single fault. In figure 38 are 7 outer iterations (m_1) needed for the F(T)-GMRES without a fault. The inner solver is initialized with zero values for vector $w_j(w_0)$. The used matrix A for figure 38 is built like in section 7.1 for problem 1, the relative residuum is below 10^{-9} for all computations.

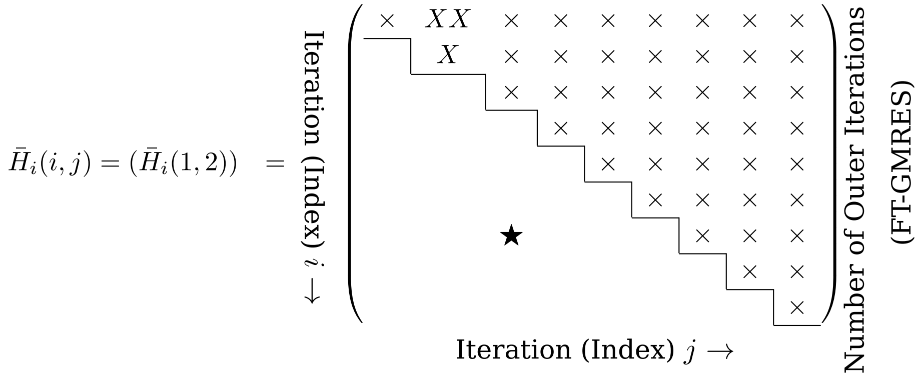
9.3 Testing of all positions of the orthogonalization with multiple faults

In the case of multiple faults there are different possibilities but faulting can be still done on positions like in formula 106 for matrix $\bar{H}(i,j)$. The main contrast of single and multiple faults is that multiple faults occur on different positions of the orthogonalization process for computing $h_{i,j}$ (line 5 - 10 of algorithm 6) in the inner solver of the FT-GMRES (line 4 of algorithm 14) during one execution. Only the number of outer iterations for single and multiple faults can be shown at one position of matrix $\bar{H}(i,j)$. Therefore there must be some restrictions to this approach, like faulting can be only applied along index i or index j which means if position $\bar{H}(i,j)$ is shown with the number of outer iterations it also means that indexes along i or j are affected with the same perturbation. Just for example if the number of outer iterations is shown on $\bar{H}(1,2)$ like faulting on $\bar{h}_{1,2}$ also all positions after $i \geq 1$ are affected if faulting is applied with multiple faults along index i ($\bar{H}_i(i,j)$). In contrast if faulting is applied along index j ($\bar{H}_j(i,j)$) all position after $j \geq 2$ are affected, inclusive the position $\bar{H}(i,j)$ or $\bar{H}_j(i,j)$. In formula 108 and 109 where faulting is applied along index i or index j, the first position always shows the number of outer iterations for multiple faults. The number of outer iterations is shown on positions marked with "XX" (= $\bar{H}_j(i,j)$ or $\bar{H}_i(i,j)$) where positions with "X" are also affected in the case of one execution of the inner solver.



(108)

Figure 39 Matrix $\bar{H}_j(i,j)$ which visualizes where faulting is applied with multiple faults along index j during the orthogonalization process and with the according number of outer iterations.

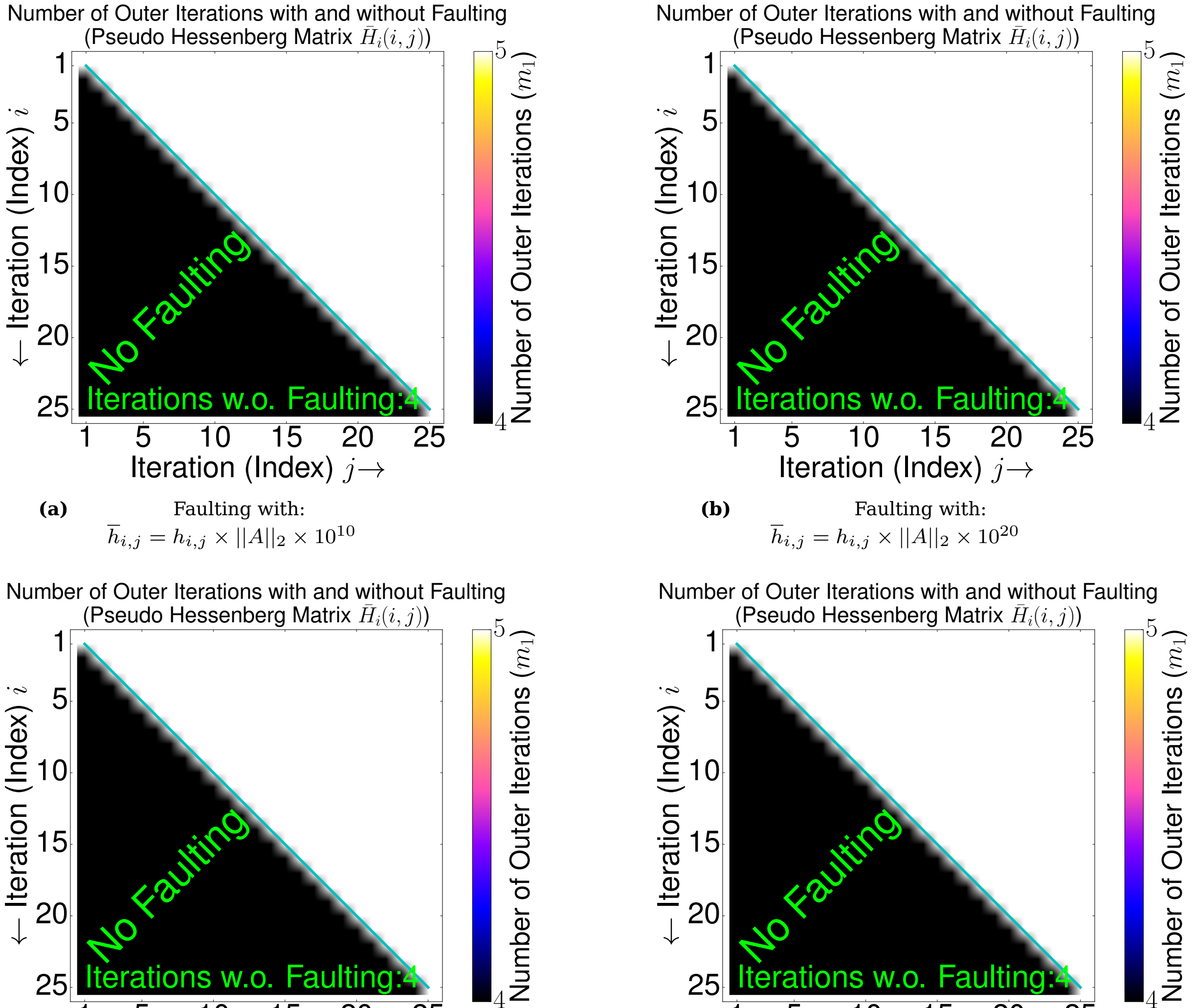


(109)

Figure 40 Matrix $\bar{H}_i(i,j)$ which visualizes where faulting is applied with multiple faults along index i during the orthogonalization process and with the according number of outer iterations.

For more experiments and informations about multiple faults in the FT-GMRES just see [70] but for more experiments with un-symmetric problems in this work just go to section 9.11.

9.3.1 Faulting on a matrix with low condition number along index i - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{20}$

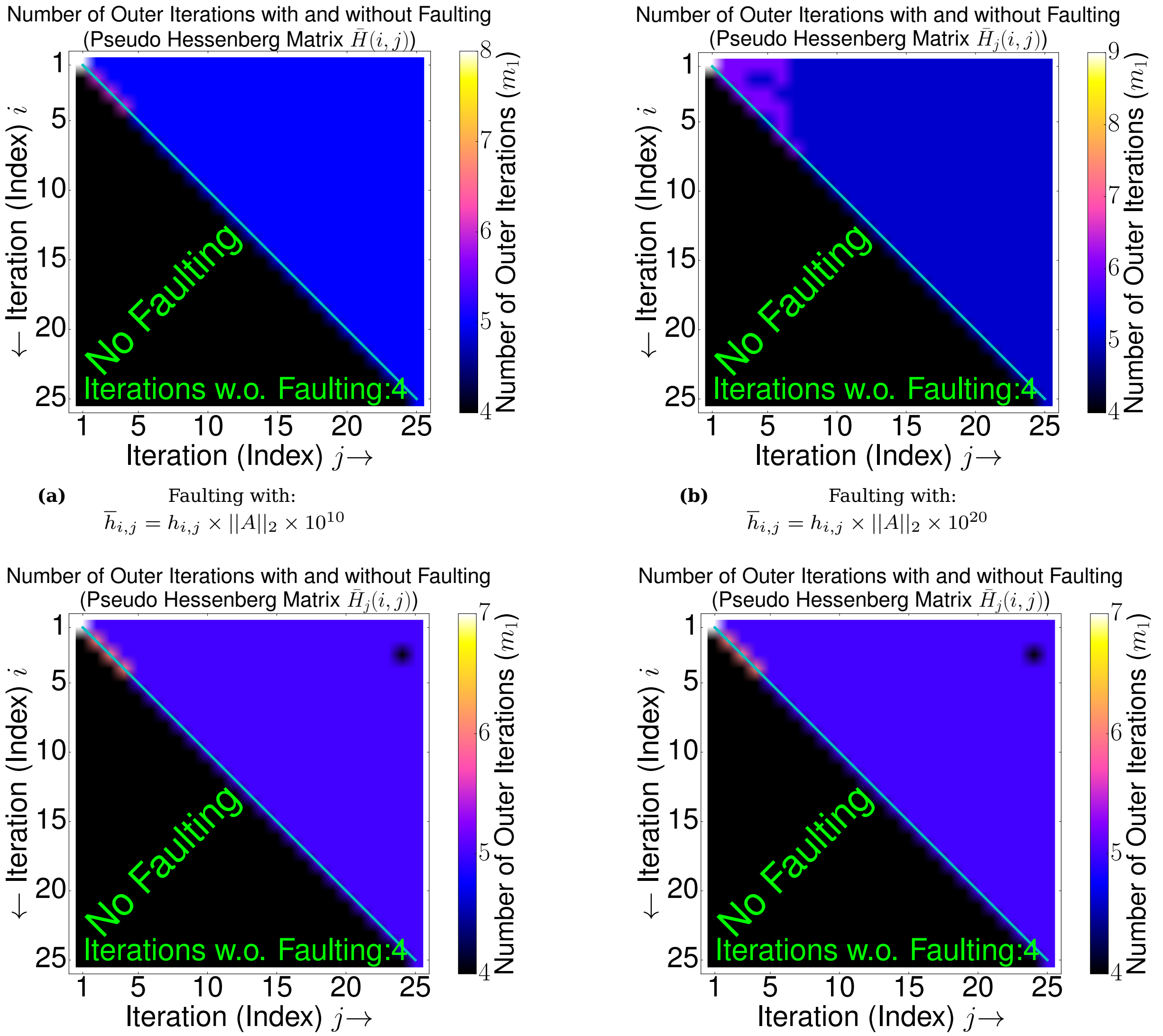
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 2$

Figure 41 Number of outer iterations for multiple faults during orthogonalization ($h_{i,j}$) along index i and for various disturbances while solving a diagonal matrix with condition number 142.5.

In the case of multiple faults along index (iteration) i ($\bar{H}_i(i,j)$) in figure 41 there is no overhead caused but only one additional iteration is needed which is mainly because of achieving the same relative residuum ($||r||_2 = ||Ax - b||_2/\ ||b||_2$) as the F-GMRES after computation (10^{-9}). At each position along index i the same perturbation is induced ($\bar{H}_i(i,j)$). The inner solver is initialized with zero values for vector $w_j(w_0)$. The used matrix A is built like in section 7.1 for problem 1 (see formula 102) with a condition number of 142.5 for all problems and trials. Faulting with multiple faults along index i seems to be almost uncritical. The F(T)-GMRES takes 4 outer iterations (m_1) without a fault for the first computation.

9.3.2 Faulting on a matrix with low condition number along index j - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{20}$

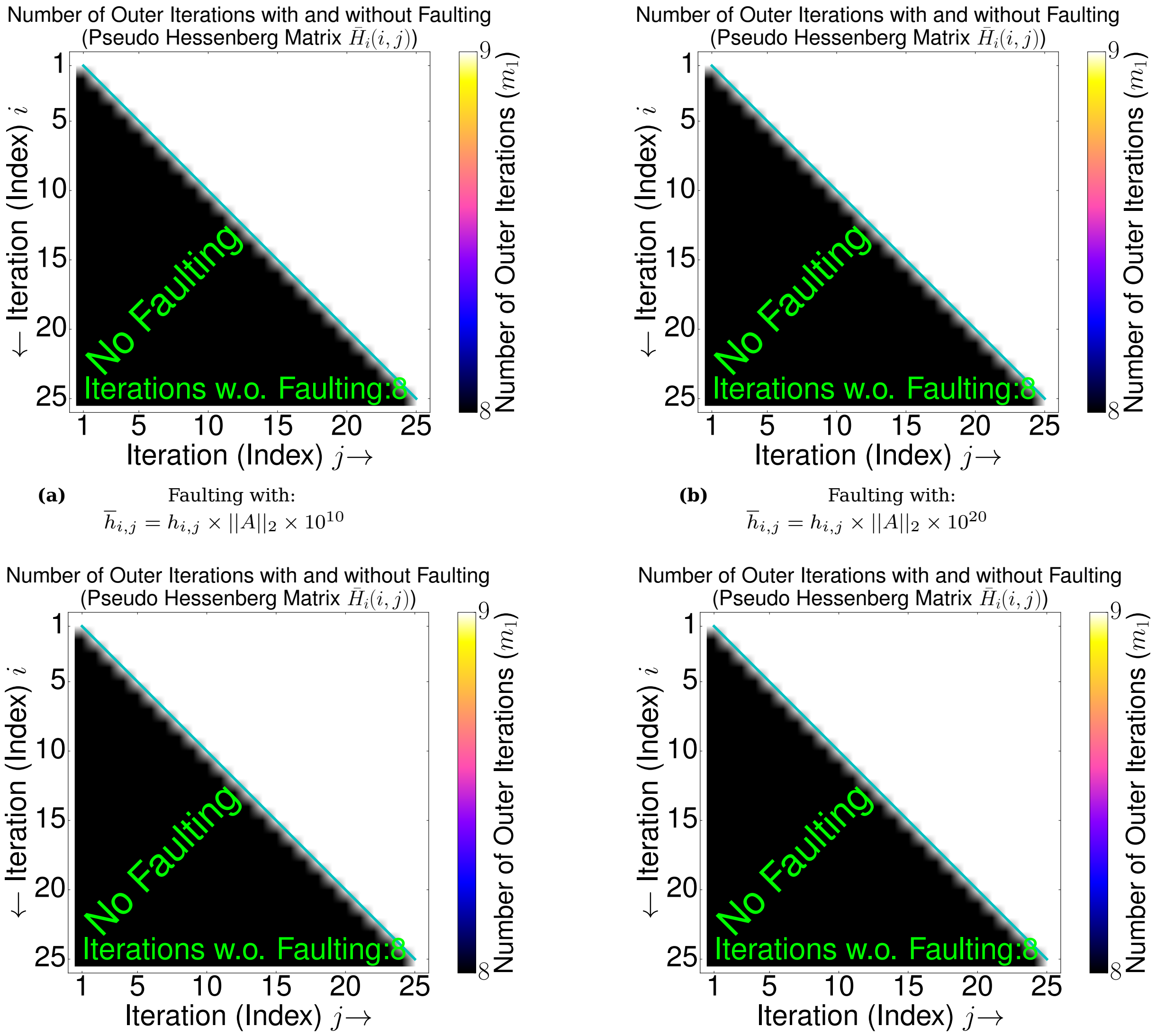
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 2$

Figure 42 Number of outer iterations for multiple faults during orthogonalization ($h_{i,j}$) along index j and for various disturbances while solving a diagonal matrix with condition number 142.5.

In the case of multiple faults along index j ($\bar{H}_j(i,j)$) in figure 42 there is some overhead caused but mainly at positions of $\bar{H}_j(i=j, i=j)$ except in the case of large perturbations but there is no contrast for perturbations of $10^{-0.5}$ and 2. The number of outer iterations is shown for each position ($h_{i,j}$) with multiple faults along index j ($\bar{H}_j(i,j)$). Figure 42b also shows that faulting along index j with large perturbations in the inner solver of the FT-GMRES is critical. The worst overhead is 80 % ($=\frac{9}{5}$) for multiple faults. The inner solver is initialized with zero values for vector $w_j(w_0)$, the used matrix A is built like in section 7.1 for problem 1 with a condition number ($||.||_2$) of 142.5. The F(T)-GMRES takes 4 outer iterations (m_1) without a fault for the first computation.

9.3.3 Faulting on a matrix with high condition number along index i - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with:
$\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(b) Faulting with:
$\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{20}$

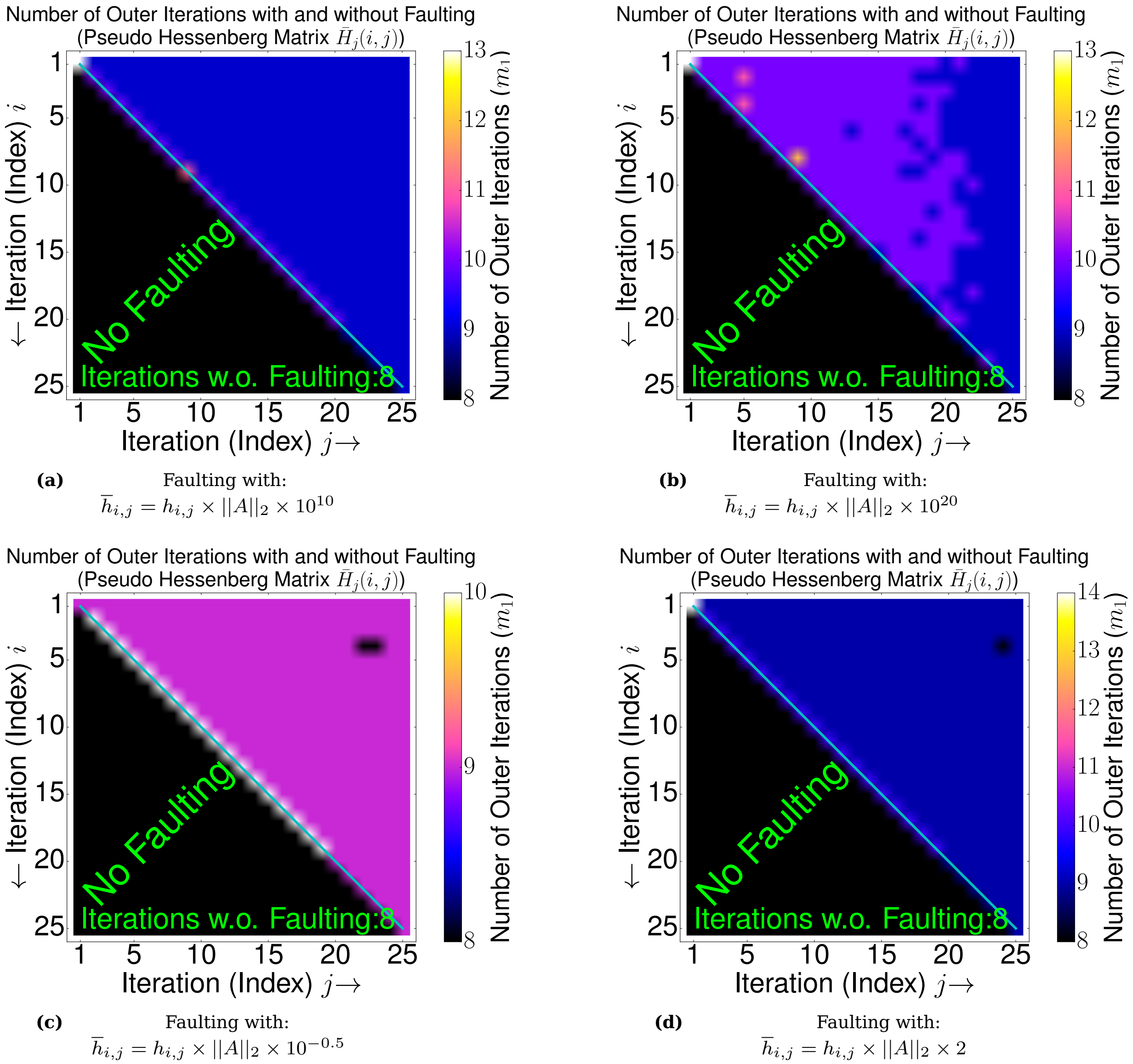
(c) Faulting with:
$\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with:
$\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 2$

Figure 43 Number of outer iterations for multiple faults during orthogonalization ($h_{i,j}$) along index i and for various disturbances while solving a diagonal matrix with condition number 1425.

In the case of multiple faults along index (iteration) i ($\bar{H}_i(i,j)$) in figure 43 there is no overhead caused but only one additional iteration is needed which is mainly because of achieving the same relative residuum ($||r||_2 = ||Ax - b||_2/\ ||b||_2$) as the F-GMRES after computation (10^{-9}). At each position along index i the same perturbation is induced ($\bar{H}_i(i,j)$). The inner solver is initialized with zero values for vector $w_j(w_0)$. The used matrix A is built like in section 7.1 for problem 1 (see formula 102) with a condition number of 1425 for all problems and trials. Faulting with multiple faults along index i seems to be almost uncritical. The F(T)-GMRES takes 8 outer iterations (m_1) without a fault for the first computation.

9.3.4 Faulting on a matrix with high condition number along index j - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{20}$

(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 2$

Figure 44 Number of outer iterations for multiple faults during orthogonalization ($h_{i,j}$) along index j and for various disturbances while solving a diagonal matrix with condition number 1425.

In the case of multiple faults along index j ($\bar{H}_j(i,j)$) in figure 44 there is some overhead caused but mainly at positions of $\bar{H}_j(i=j, i=j)$ except in the case of large perturbations but there is no contrast for perturbations of $10^{-0.5}$ and 2. The number of outer iterations is shown for each position with multiple faults along index j ($\bar{H}_j(i,j)$). Figure 44b also shows that faulting along index j with large perturbations in the inner solver of the FT-GMRES is critical. The worst overhead is 50 % ($= \frac{14}{9}$) for multiple faults. The inner solver is initialized with zero values for vector $w_j(w_0)$, the used matrix A is built like in section 7.1 for problem 1 with a condition number ($||.||_2$) of 1425. The F(T)-GMRES takes 8 outer iterations (m_1) without a fault for the first computation.

9.4 Comparison between the 2D Poisson and adder_dcop_63 problem

In this section two different matrices are compared, the first one is the 2D Poisson and second one is the adder_dcop_63 matrix [10]. Both matrices have unequal properties like the condition number ($||.||_2$) shown in table 45 and both differ in the eigenvalue distribution (λ) presented in table 5. There is also a complete comparison in figure 60 and 61 for the eigenvalue (λ) distribution of the 2D Poisson and adder_dcop_63 matrix which is the main contrast between both matrices.

Properties	2D Poisson matrix
Number of rows	10000
Number of columns	10000
Nonzeros	49600
Structural full rank?	yes
Structure	symmetric
Type	real
Positive definite?	yes
Condition number ($||.||_2$)	6.0107×10^3

Properties	adder_dcop_63 matrix
Number of rows	1813
Number of columns	1813
Nonzeros	11246
Structural full rank?	yes
Structure	un-symmetric
Type	real
Positive definite?	no
Condition number ($||.||_2$)	5.6107×10^{11}

Figure 45 Sample matrices for comparison (2D Poisson and adder_dcop_63 matrix).

The aim of this section is to find out which property of matrix A makes faulting during the orthogonalization process for computing $h_{i,j}$ in the inner solver of the FT-GMRES less harmful but perhaps also in other parts. There are still two different scenarios applied in this section like faulting with error type 1 or error type 2 similar as in section 8.1 and table 27. For both cases the residuals ($||r||_2$) of the outer solver are plotted but as well the approximation errors ($||\sigma_j||_2^2$) of the Hessenberg matrix $\hat{H}(i,j)$ in the inner solver of FT-GMRES which are nearly the same as the residuals. The value $||\sigma_j||_2^2$ can be computed with the help of the Givens rotation (shown in algorithm 5). In section 9.12.2 (2D Poisson) and 9.12.5 (adder_dcop_63 matrix) there is shown which structure for the Hessenberg matrix is mainly build for these two problems in this section.

The 2D Poisson matrix:	**The adder_dcop_63 matrix [10]:**
Condition number ($||.||_2$): 6.0107×10^3	Condition number ($||.||_2$): 5.6107×10^{11}
Greatest eigenvalue of matrix A: 7.9553	Greatest eigenvalue of matrix A: 1.1743
2^{nd} greatest eigenvalue of Matrix A: 7.8888	2^{nd} greatest eigenvalue of matrix A: 1.0014
Smallest eigenvalue of matrix A: 0.1112	Smallest eigenvalue of matrix A: 1.0014 $\times 10^{-12}$

Table 5 Some properties of the 2D Poisson and adder_dcop_63 matrix.

This approximation error $||\sigma_j||_2^2$ is almost identical to the residuum $||r_j||_2$ for each iteration of j but it reduces the number of operations a lot because computing the residuum $||r_j||_2$ will always need a sparse multiplication ($y = Ax$). The value of $||\sigma_j||_2^2$ is computed in line 14 of algorithm 5. The approximation error ($||\sigma_j||_2^2$) and the residuum ($||r_j||_2$) differ really less in terms of the magnitude and convergence history in an error free run where the value of $||\sigma_j||_2^2$ stagnates more in the case of a fault which is then different from $||r_j||_2$. The value of $||\sigma_j||_2^2$ can be also used to build a detector to check for stagnating convergence and if there is a fault in the inner solver.

The outcome of these experiments in this section is that faulting during solving the adder_dcop_63 matrix has less influence on the number of outer iterations in comparison to the 2D Poisson matrix. Especially faulting with perturbations of 10^{-300} in figure 55 has less effect in contrast to figure 53 where faulting is more crucial for more iterations of j except the very first iteration ($h_{1,2}$). The residual curves are also shown in figure 48 for the 2D Poisson and in figure 50 in the case of solving the adder_dcop_63 matrix, this matrix has a larger condition number than the 2D Poisson problem.

9.4.1 Residuum curves of GMRES and F-GMRES for solving the 2D Poisson and adder_dcop_63 problem - Inner solver initialized with zero values

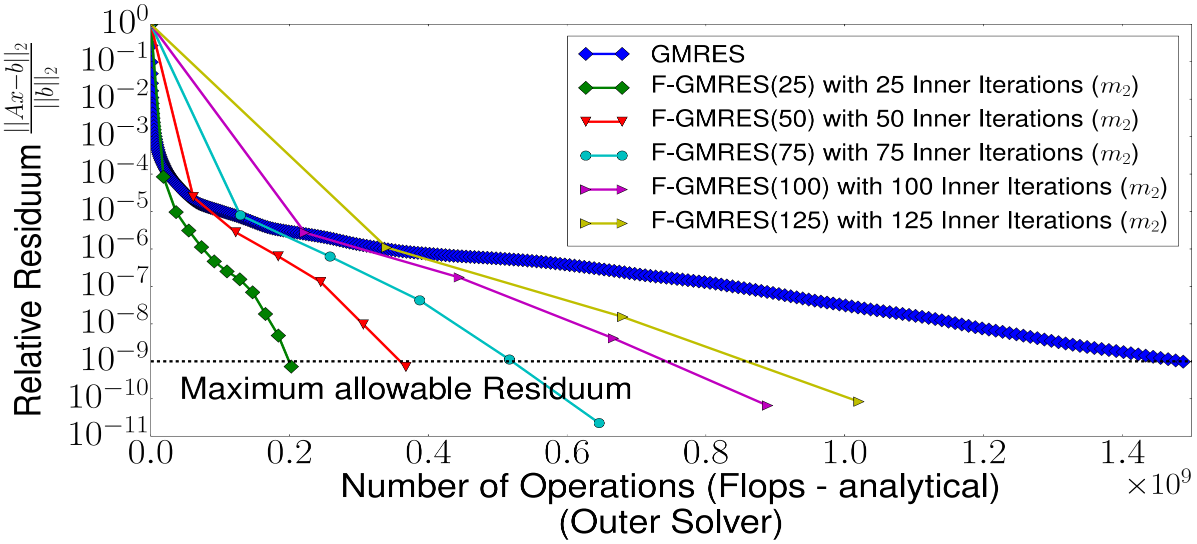


Figure 46 Residuum curves for solving the 2D Poisson problem with the GMRES and F-GMRES solver. The final relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of all computations is below 10^{-9}.

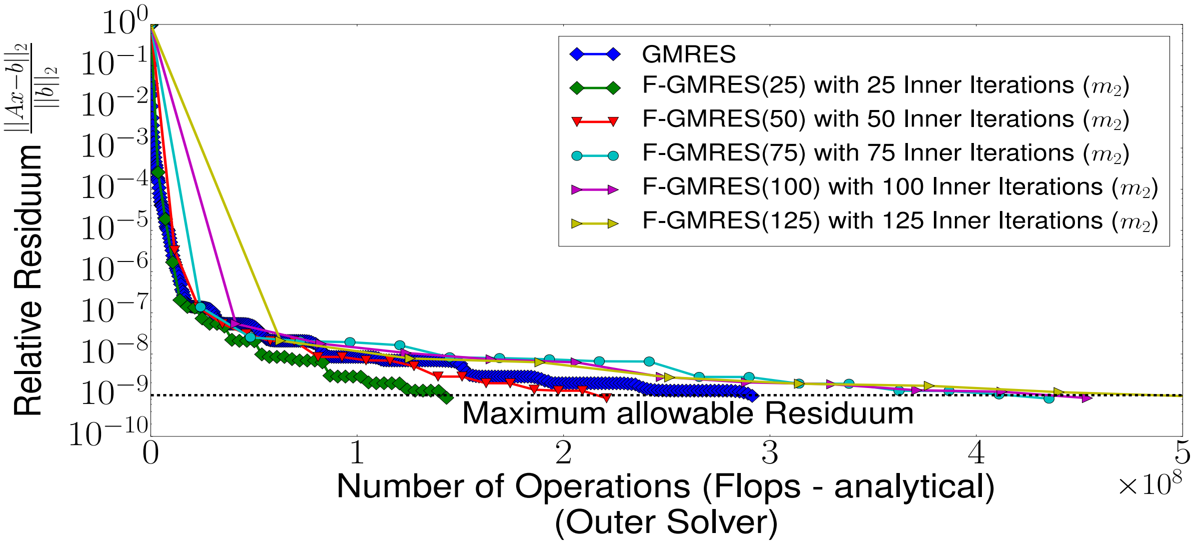
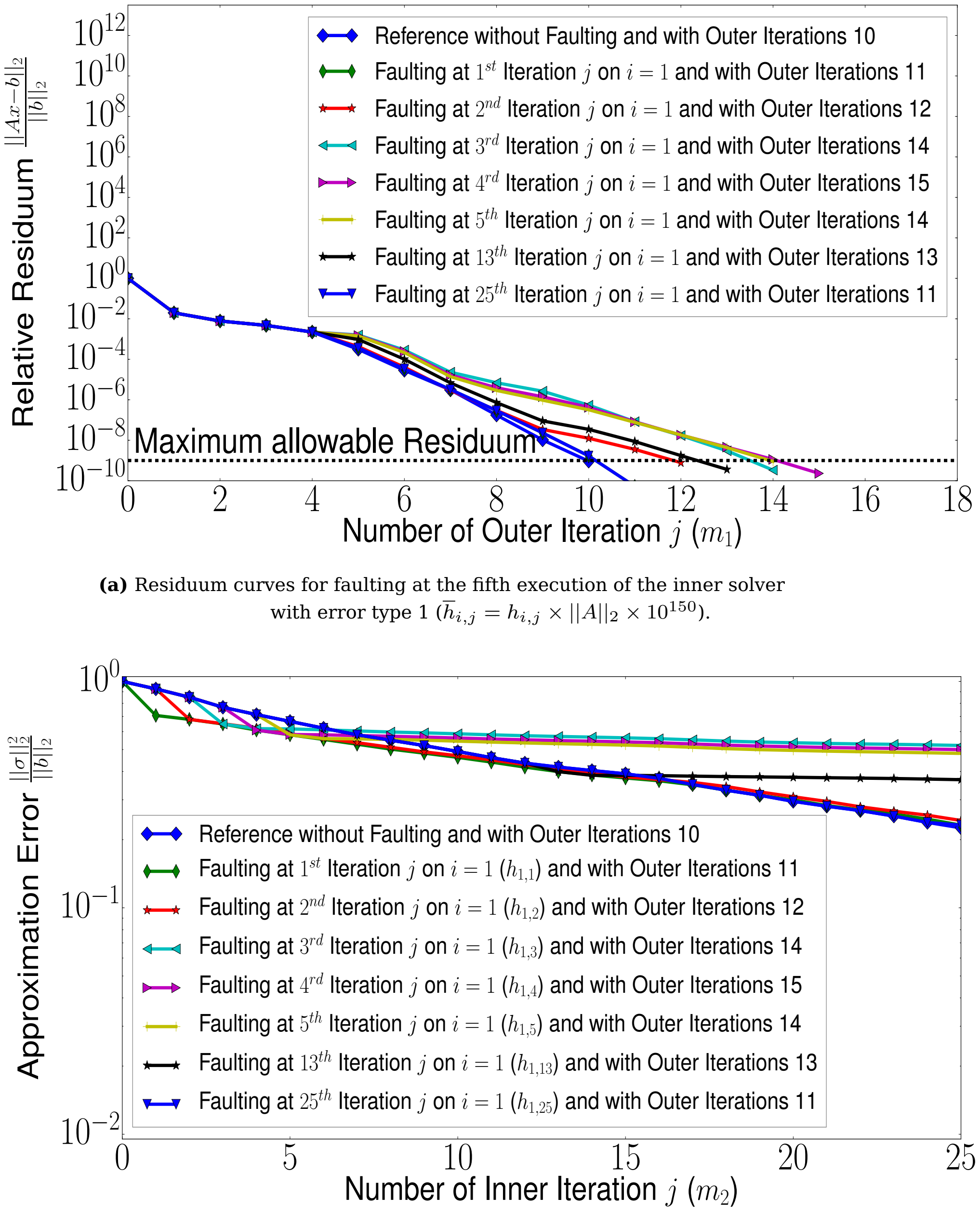


Figure 47 Residuum curves for solving the adder_dcop_63 problem with the GMRES and F-GMRES solver. The final relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of all computations is below 10^{-9}.

In figure 46 and 47 the 2D Poisson and adder_dcop_63 matrix are solved with the GMRES and F-GMRES solver. In both figures different numbers of iterations are used for the inner solver (m_2) of the F-GMRES where the F-GMRES with 25 inner iterations is the fastest possibility in both figures. Overall the F-GMRES solver is quite faster than the GMRES solver in both figures if the number of operations is considered which means the F-GMRES helps to decrease the needed number of operations. The number of operations for GMRES and F-GMRES are computed as in section 6.1.4. The inner solver of the F-GMRES is initialized with zero values for vector $w_j(w_0)$ in both figures.

9.4.2 Residuals and approximation errors for solving the 2D Poisson problem and faulting with error type 1 - Inner solver initialized with zero values



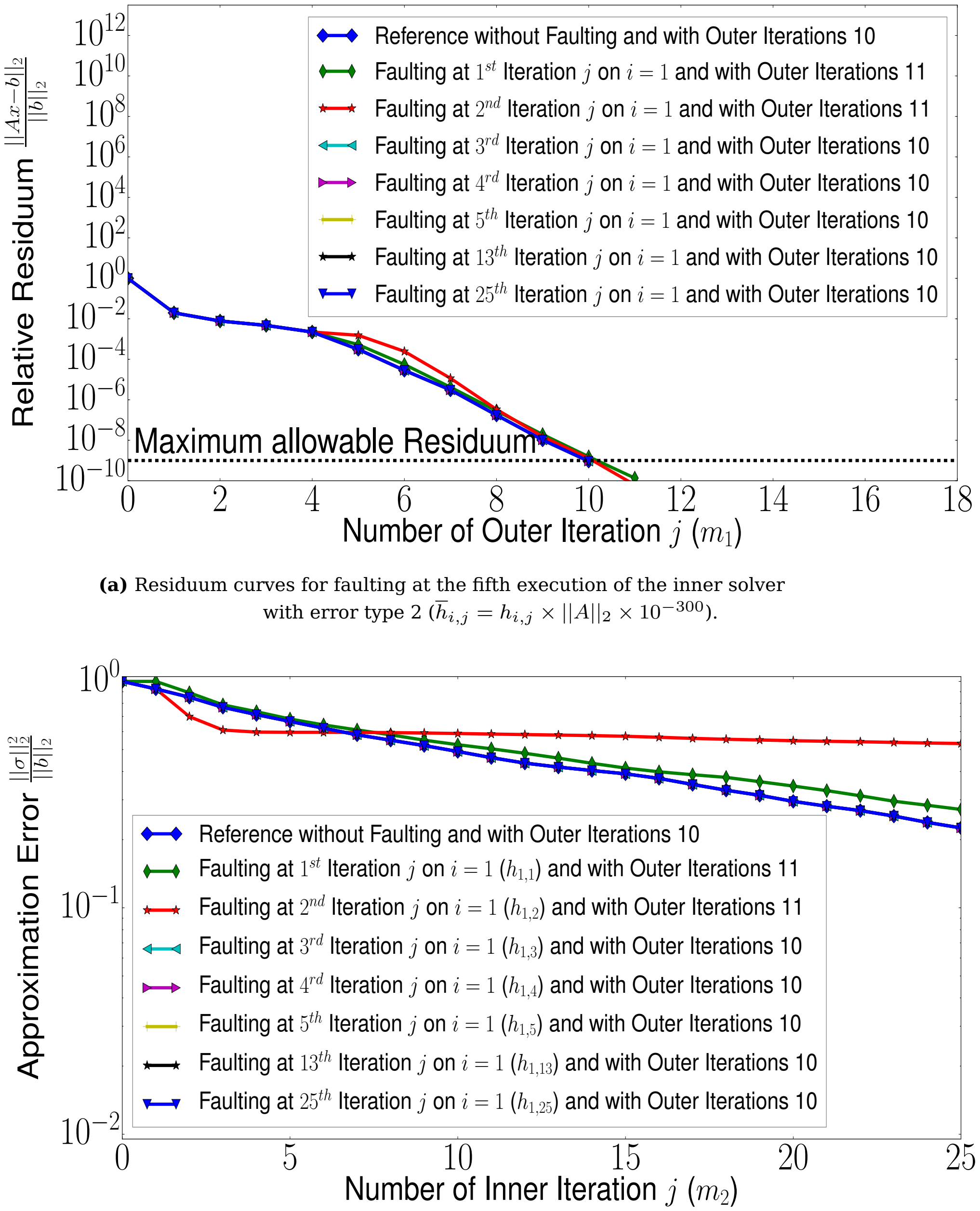
(a) Residuum curves for faulting at the fifth execution of the inner solver with error type 1 ($\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$).

(b) Approximation errors for faulting at the fifth execution of the inner solver with error type 1 ($\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$).

Figure 48 Residuum (outer solver) and the corresponding approximation (inner solver) curves for solving the 2D Poisson matrix.

In figure 48 the residuals ($||r_j||_2$) and approximation errors ($||\sigma_j||_2^2$) are plotted for solving the 2D Poisson matrix and faulting with error type 1. Figure 48b shows that the approximation error respective residual stagnates for some positions of $h_{i,j}$ ($h_{1,3}, h_{1,5}$). The inner solver is initialized with zeros for vector $w_j(w_0)$ and the achieved residuum ($||r||_2 = ||Ax-b||_2/||b||_2$) is below 10^{-9}.

9.4.3 Residuals and approximation errors for solving the 2D Poisson problem and faulting with error type 2 - Inner solver initialized with zero values



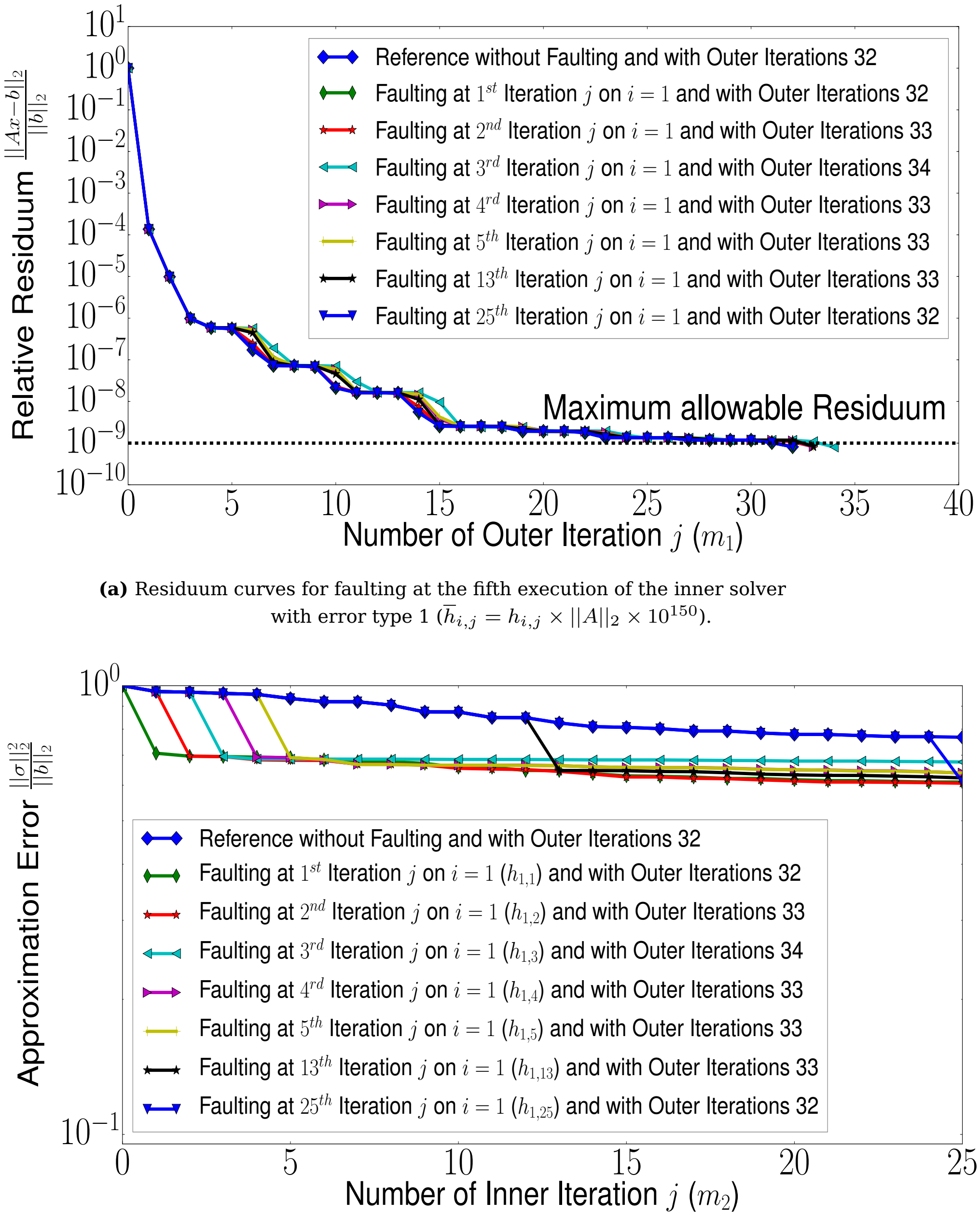
(a) Residuum curves for faulting at the fifth execution of the inner solver with error type 2 ($\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$).

(b) Approximation errors for faulting at the fifth execution of the inner solver with error type 2 ($\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$).

Figure 49 Residuum (outer solver) and the corresponding approximation (inner solver) curves for solving the 2D Poisson matrix.

In figure 49 the residuals ($||r_j||_2$) and approximation errors ($||\sigma_j||_2^2$) are plotted for solving the 2D Poisson matrix and faulting with error type 2. Figure 49b shows that the approximation error respective residual stagnates for a specific position of $h_{i,j}$ ($h_{1,2}$). The inner solver is initialized with zeros for vector $w_j(w_0)$ and the achieved residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9}.

9.4.4 Residuals and approximation errors for solving the adder_dcop_63 problem and faulting with error type 1 - Inner solver initialized with zero values



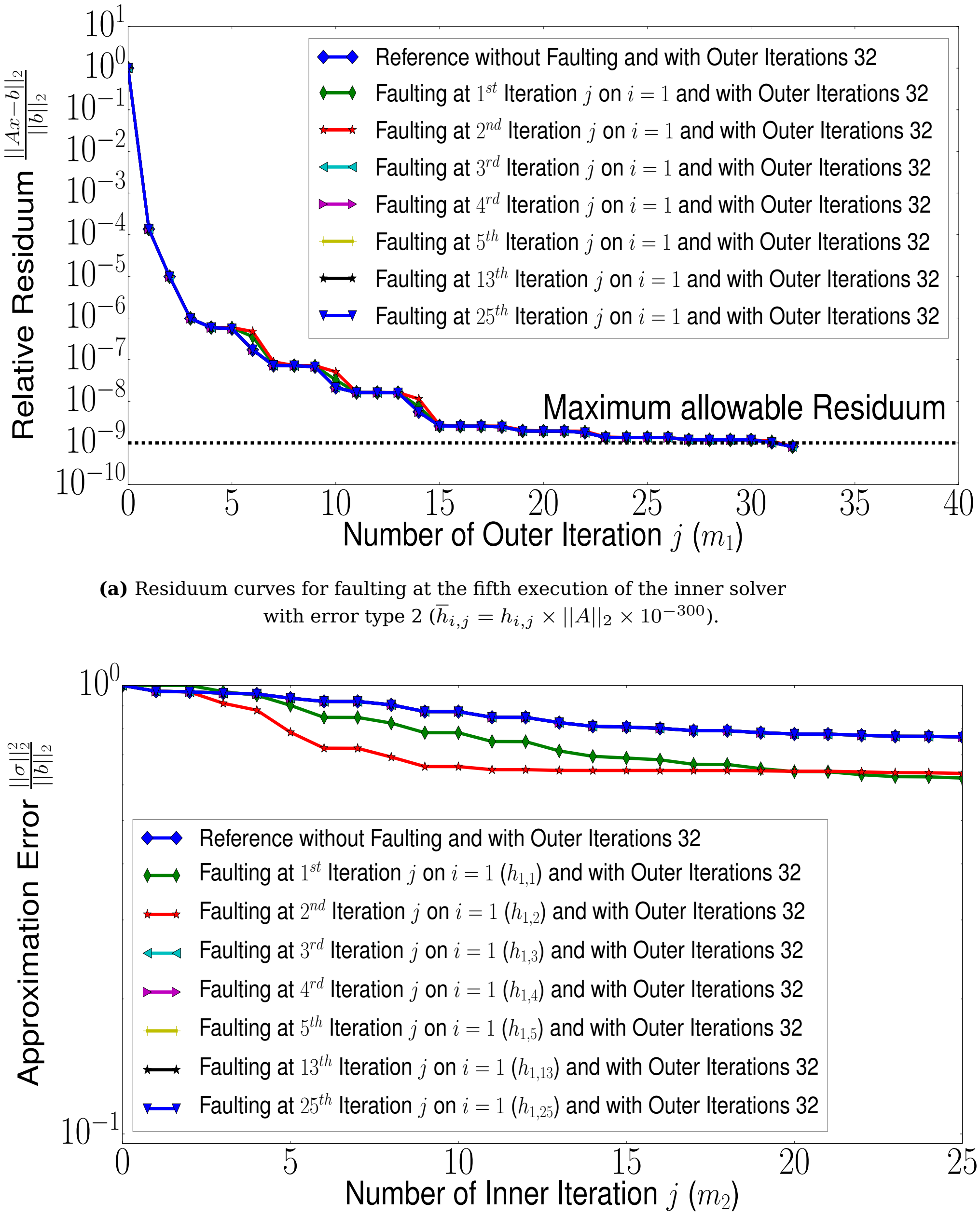
(a) Residuum curves for faulting at the fifth execution of the inner solver with error type 1 ($\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$).

(b) Approximation errors for faulting at the fifth execution of the inner solver with error type 1 ($\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$).

Figure 50 Residuum (outer solver) and the corresponding approximation (inner solver) curves for solving the adder_dcop_63 matrix.

In figure 50 the residuals ($||r_j||_2$) and approximation errors ($||\sigma_j||_2^2$) are plotted for solving the adder_dcop_63 matrix and faulting with error type 1. Figure 50b shows that the approximation error respective residual does not stagnate so much anymore for $h_{i,j}$. The inner solver is initialized with zeros for vector $w_j(w_0)$ and the achieved residuum ($||r||_2 = ||Ax-b||_2/||b||_2$) is below 10^{-9}.

9.4.5 Residuals and approximation errors for solving the adder_dcop_63 problem and faulting with error type 2 - Inner solver initialized with zero values



(a) Residuum curves for faulting at the fifth execution of the inner solver with error type 2 ($\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$).

(b) Approximation errors for faulting at the fifth execution of the inner solver with error type 2 ($\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$).

Figure 51 Residuum (outer solver) and the corresponding approximation (inner solver) curves for solving the adder_dcop_63 matrix.

In figure 51 the residuals ($||r_j||_2$) and approximation errors ($||\sigma_j||_2^2$) are plotted for solving the adder_dcop_63 matrix and faulting with error type 2. Figure 51b shows that the approximation error respective residual does not stagnate so much anymore for $h_{i,j}$. The inner solver is initialized with zeros for vector $w_j(w_0)$ and the achieved residuum ($||r||_2 = ||Ax-b||_2/||b||_2$) is below 10^{-9}.

9.4.6 Faulting on the first MGS-iteration during solving the 2D Poisson problem - Inner solver initialized with zero values

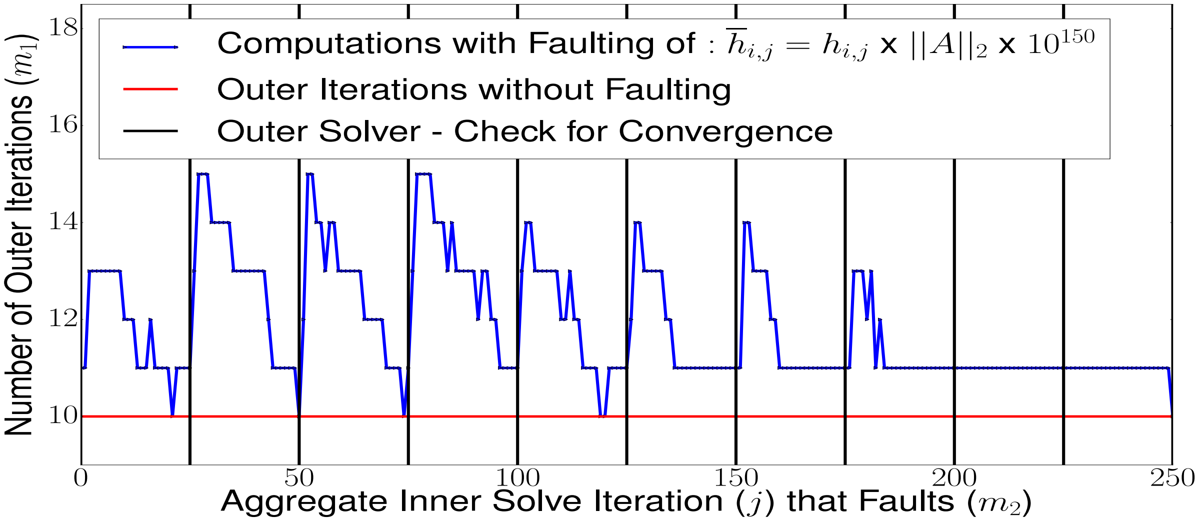


Figure 52 Faulting with single faults and error type 1 on the first MGS-iteration ($h_{i=1,j}$) during solving the 2D Poisson matrix for $i = 1$ and $j = 1, \ldots, 25$. The relative residuum is below 10^{-9}.

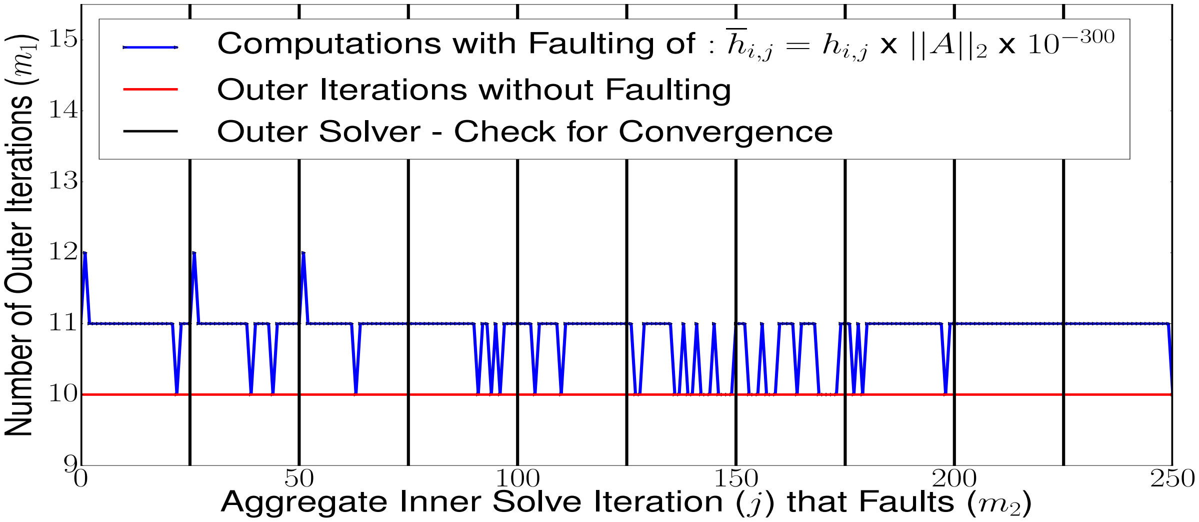


Figure 53 Faulting with single faults and error type 2 on the first MGS-iteration ($h_{i=1,j}$) during solving the 2D Poisson matrix for $i = 1$ and $j = 1, \ldots, 25$. The relative residuum is below 10^{-9}.

In figure 52 and figure 53 the 2D Poisson matrix is solved, both figures show a similar behavior like in [4]. In this case the FT-GMRES takes 10 outer iterations whereas in the worst case 15 outer iterations are observed for a single fault. The inner solver of the FT-GMRES is initialized with zero values for vector $w_j(w_0)$ and it does 25 iterations ($j = 1, \ldots, 25$) for each execution. The relative residuum ($||r||_2 = ||Ax - b||_2 / \, ||b||_2$) is below 10^{-9} for all trials and computed solutions of x. Vertical bars in figure 52 and 53 indicate the start of a new inner solve of the FT-GMRES.

9.4.7 Faulting on the first MGS-iteration during solving the adder_dcop_63 problem - Inner solver initialized with zero values

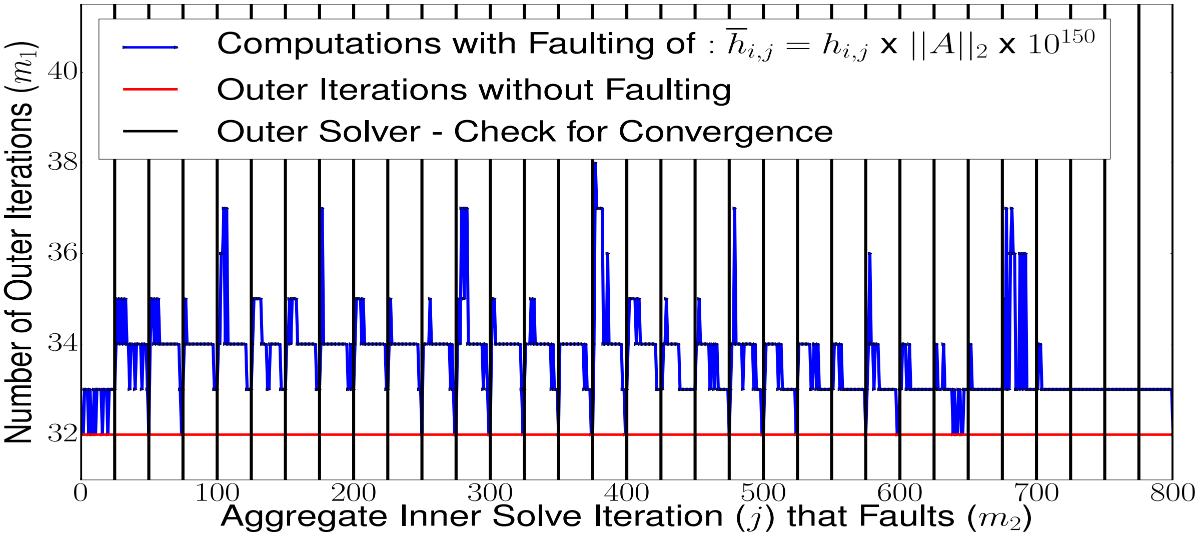


Figure 54 Faulting with single faults and error type 1 on the first MGS-iteration ($h_{i=1,j}$) during solving the adder_dcop_63 matrix for $i = 1$ and $j = 1, \ldots, 25$. The relative residuum is below 10^{-9}.

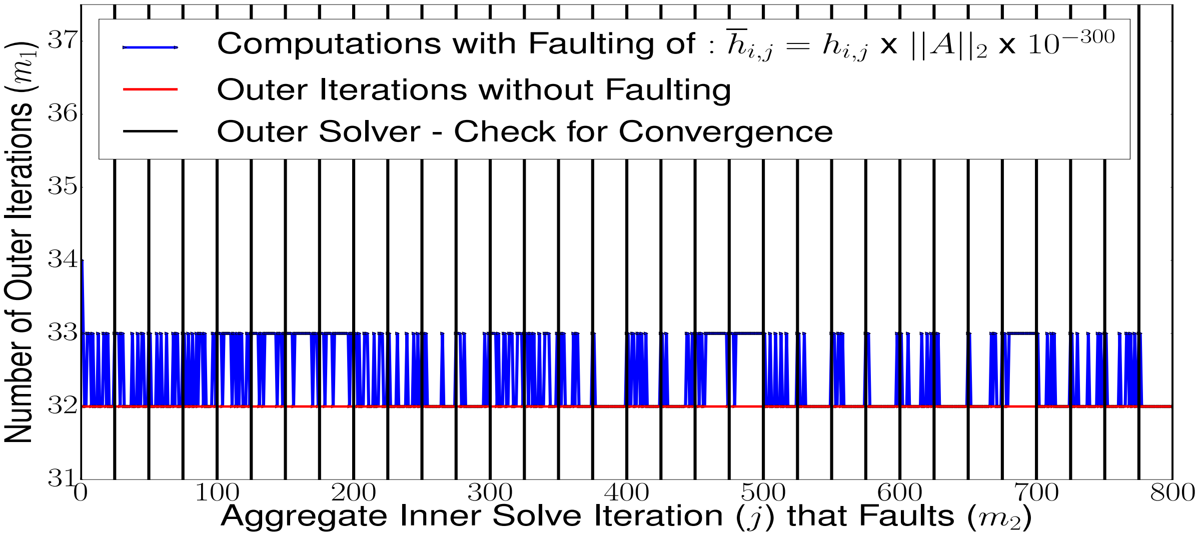


Figure 55 Faulting with single faults and error type 2 on the first MGS-iteration ($h_{i=1,j}$) during solving the adder_dcop_63 matrix for $i = 1$ and $j = 1, \ldots, 25$. The relative residuum is below 10^{-9}.

In figure 54 and figure 55 the adder_dcop_63 is solved, both figures show a different behavior as in [4] (other matrix). In this case the FT-GMRES takes 32 outer iterations whereas in the worst case 38 outer iterations are observed for a single fault. The inner solver of the FT-GMRES is initialized with zero values for vector $w_j (w_0)$ and it does 25 iterations ($j = 1, \ldots, 25$) for each execution. The relative residuum ($||r||_2 = ||Ax - b||_2 / \, ||b||_2$) is below 10^{-9} for all trials and computed solutions of x. Vertical bars in figure 54 and 55 indicate the start of a new inner solve of the FT-GMRES.

9.4.8 Faulting on the Givens rotation during solving the 2D Poisson problem - Inner solver initialized with zero values

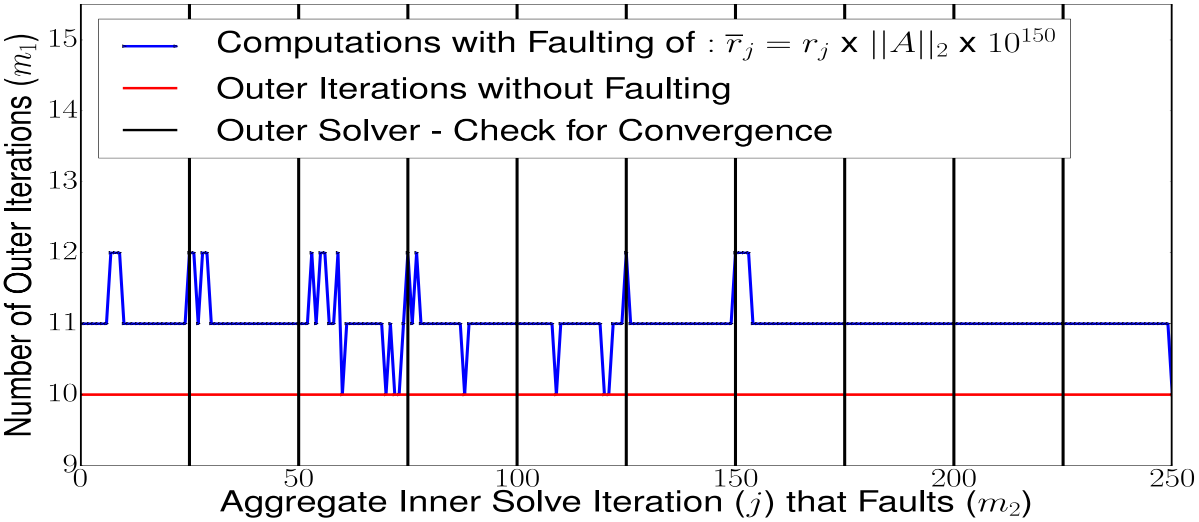


Figure 56 Faulting with single faults and error type 1 during applying the Givens rotation while solving the 2D Poisson matrix with a disturbed radius $\hat{r_j}$ ($\hat{r_j} = r_j \times ||A||_2 \times 10^{150}$).

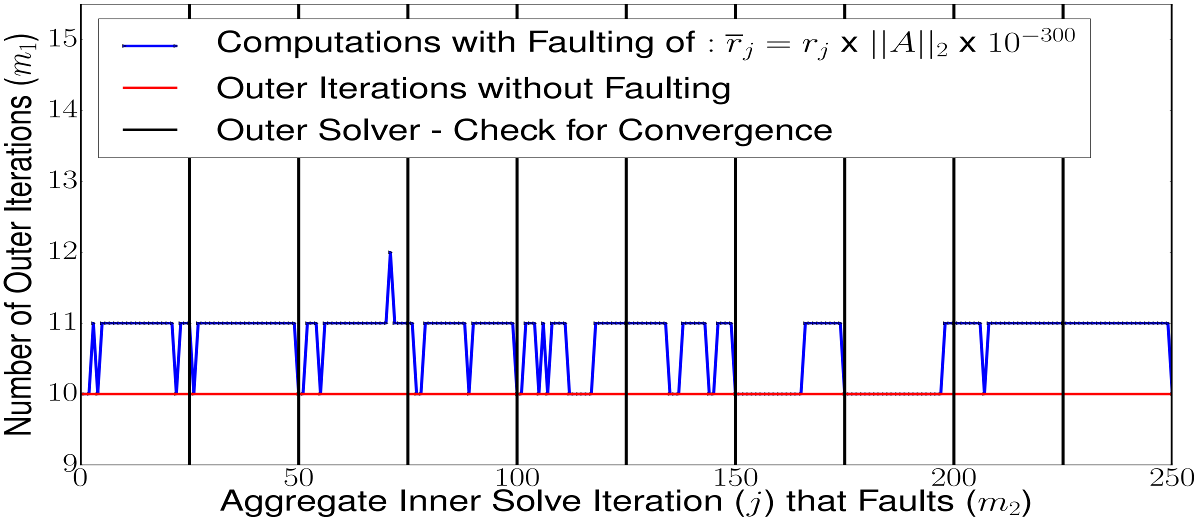


Figure 57 Faulting with single faults and error type 2 during applying the Givens rotation while solving the 2D Poisson matrix with a disturbed radius $\hat{r_j}$ ($\hat{r_j} = r_j \times ||A||_2 \times 10^{-300}$).

In figure 56 and 57 faulting during applying the Givens rotation is shown. At each iteration of j in the inner solver it is possible that there is a single fault during applying the Givens rotation. Faulting is only done once at each position in the inner solver for 10 × 25 trials (outer × iterations of the inner solver). The relative residuum ($||r||_2 = ||Ax - b||_2 / \; ||b||_2$) is below 10^{-9} for all trials. Vertical bars in figure 56 and 57 indicate the start of a new inner solve of the FT-GMRES.

9.4.9 Faulting on the Givens rotation during solving the adder_dcop_63 problem - Inner solver initialized with zero values

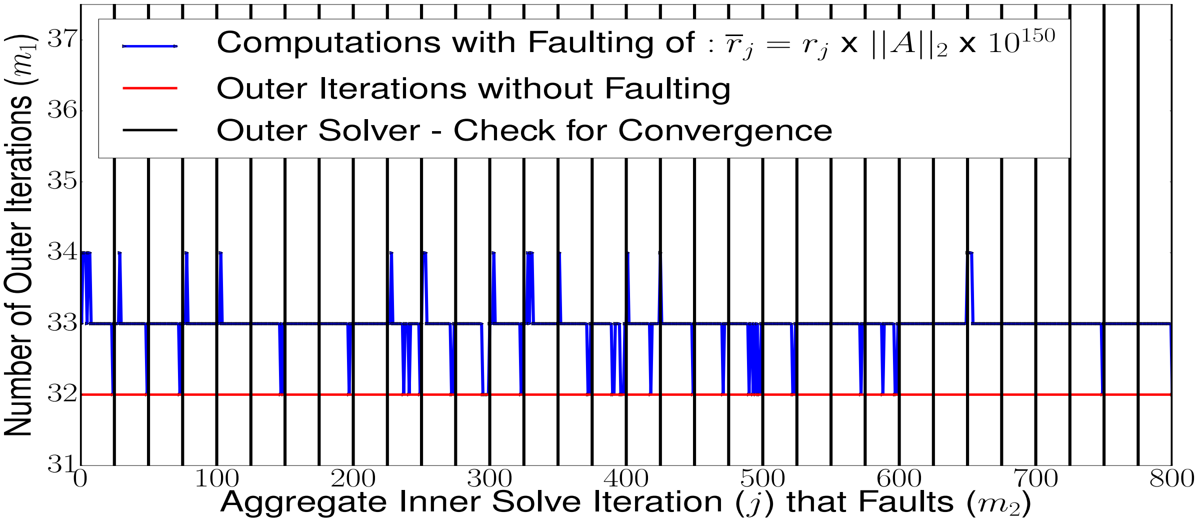


Figure 58 Faulting with single faults and error type 1 during applying the Givens rotation while solving the adder_dcop_63 matrix with a disturbed radius $\hat{r_j}$ ($\hat{r_j} = r_j \times ||A||_2 \times 10^{150}$).

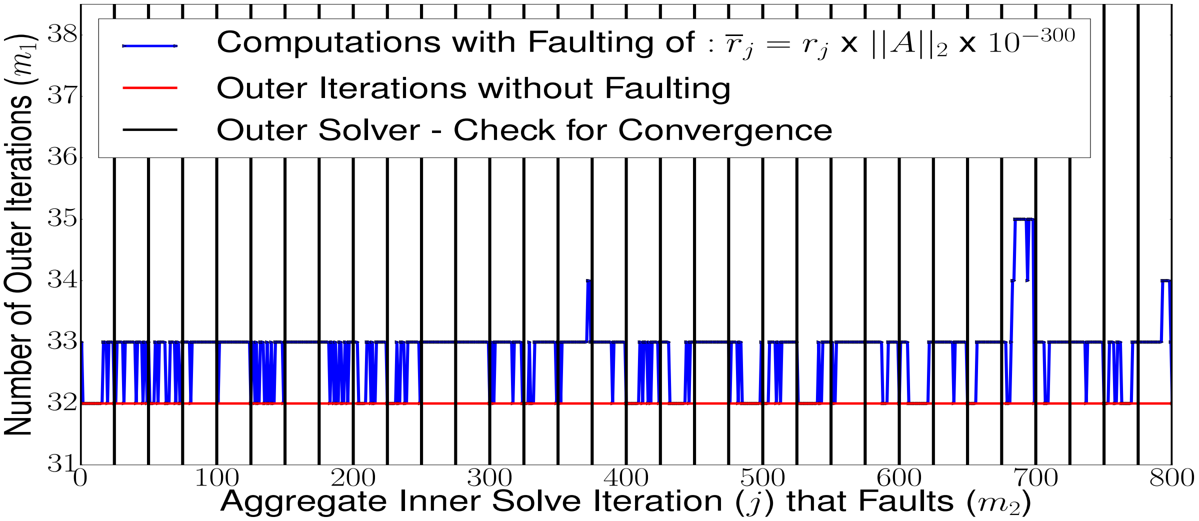


Figure 59 Faulting with single faults and error type 2 during applying the Givens rotation while solving the adder_dcop_63 matrix with a disturbed radius $\hat{r_j}$ ($\hat{r_j} = r_j \times ||A||_2 \times 10^{-300}$).

In figure 58 and 59 faulting during applying the Givens rotation is shown. At each iteration of j in the inner solver it is possible that there is a single fault during applying the Givens rotation. Faulting is only done once at each position in the inner solver for 32 × 25 trials (outer × iterations of the inner solver). The relative residuum ($||r||_2 = ||Ax - b||_2 / \ ||b||_2$) is below 10^{-9} for all trials. Vertical bars in figure 58 and 59 indicate the start of a new inner solve of the FT-GMRES.

9.4.10 Eigenvalue distribution of the 2D Poisson and adder_dcop _63 matrix

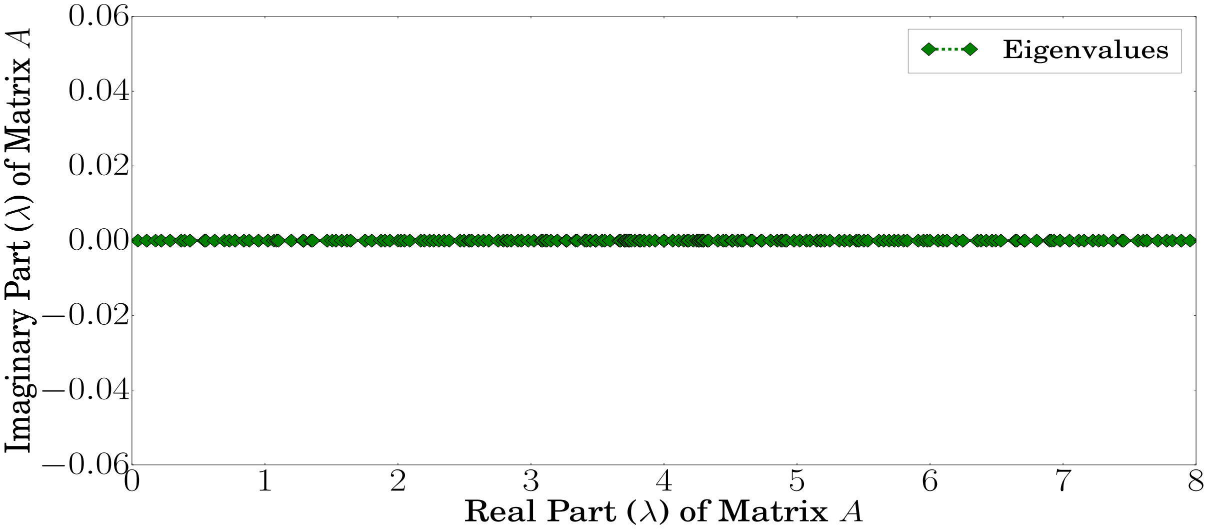


Figure 60 Eigenvalue distribution of the 2D Poisson matrix for a size of 100 × 100.

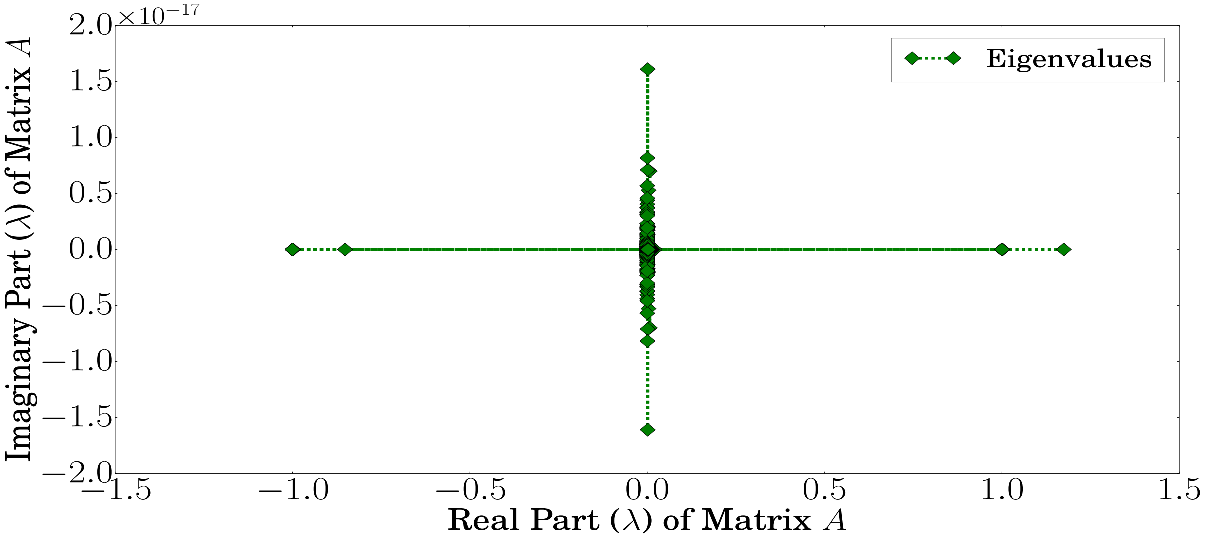


Figure 61 Eigenvalue distribution of the adder_dcop_63 matrix.

In figure 60 and 61 the eigenvalue distribution (λ) of both used problems is shown, in the case of the adder_dcop_63 matrix the eigenvalues are more centered except some outliers which is an indicator for super-lineare convergence like in [73] but in the case of the 2D Poisson matrix there are only real eigenvalues and all are on the same line. Superlineare convergence leads to the outcome that the used solver is able to solve the problem much faster than expected independent of the condition number ($||.||_2$). The used solver gets faster for each further iteration which leads to the effect of lowering the total number of iterations even for high condition numbers ($||.||_2$).

9.5 Faulting while solving the 2D Poisson and adder_dcop_63 problem with/without ILU-preconditioning in the inner solver of the FT-GMRES

In this section the 2D Poisson matrix is solved with and without ILU-preconditioning [6] but as well the adder_dcop_63 matrix taken from the sparse storage place for sparse matrices [10]. These experiments are done to show the effect of faulting during the orthogonalization process for computing $h_{i,j}$ (line 5 - 10 of algorithm 6) with and without preconditioning in the inner solver (line 4 of algorithm 14) of the FT-GMRES. Faulting is done with different kinds of perturbations like in section 8.1 such that a disturbed $\bar{h}_{i,j}$ is computed with $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times E_{perturbed}$ where $E_{perturbed}$ is the real perturbation during the orthogonalization process and can be chosen arbitrary. Preconditioning in this section is mainly done with the ILU-factorization [6] for speeding up the computation for solving $Ax = b$ with using the FT-GMRES solver.

How to apply ILU-preconditioning [6] for the inner solver of the F(T)-GMRES is shown in section 5.2.10.4 which is mainly done for the standard GMRES solver but can be easily adapted for the inner solver of the FT-GMRES (in line 4 of algorithm 14) as well as for the F-GMRES solver (in line 4 of algorithm 11) because it doesn't matter if the problem $Ax = b$ should be solved or $q_j = Mw_j$ for the standard GMRES solver as the inner solver. Therefore there is no contrast if also ILU-preconditioning is applied, in this case the standard matrix vector product $v_{j+1} = Aq_j$ has to be replaced for ILU-preconditioning with $v_{j+1} = L(\backslash U\backslash(Aq_j))$ where L and U are the factorized matrices of matrix A such that the product of LU is approximately A ($LU \approx A$). The relative error ($Error_{ILU}$) of this factorization is defined with $Error_{ILU} = ||LU - A||_2/||A_2||_2$ where L and U are a lower (L) and upper (U) tridiagonal matrix.

The most important data about the 2D Poisson matrix are shown in table 45 where also the adder_dcop_63 matrix is specified for comparison both used matrices are unequal in almost all properties. Something similar is already done in section 8.2 where the number of outer iterations (m_1) in the presence of a single fault is visualized for some specific positions like in section 9.2 to show the number of outer iterations (m_1) for each position of the orthogonalization process ($h_{i,j}$) in the case of a single fault for the inner solver of the FT-GMRES.

The main reason why the 2D Poisson matrix is used in this section is because in many other settings this matrix is also used and it is not really reasonable to apply ILU-factorization for a diagonal matrix because the used method would only divide through the absolute greatest element of it, this approach is also called diagonal scaling. So using ILU-preconditioning makes no really sense for diagonal matrices especially for observing the overhead in the presence of a single fault such that nothing really new can be observed. It is quite easy to apply preconditioning for a diagonal matrix just by dividing all values of the used matrix through the absolute greatest element of it such that all values are less or equal than one. This is not the case for the 2D Poisson matrix because there must be done some additional operations because of factorization to get an upper (U) and lower (L) matrix. The error of factorization[3] ($Error_{ILU}$) is chosen as great as possible because most work should be still done by the inner solver of the F(T)-GMRES.

In figure 65 solving the 2D Poisson problem is shown without ILU-preconditioning whereas in figure 67 the same problem is shown with an error of $Error_{ILU}{}^3 \approx 10^{-1}$. Applying ILU-preconditioning has some positive influence on the observed overhead but especially on the number of outer iterations in the presence of a single fault. It doesn't mean that ILU-preconditioning will always decrease the influence of faulting in the inner solver but the effect of faulting on average in the case of solving the 2D Poisson matrix can be decreased over all runs and positions of $h_{i,j}$.

[3] $Error_{ILU} = ||LU - A||_2/||A_2||_2$

9.5.1 Workload distribution of F-GMRES with ILU-preconditioning (2D Poisson matrix)

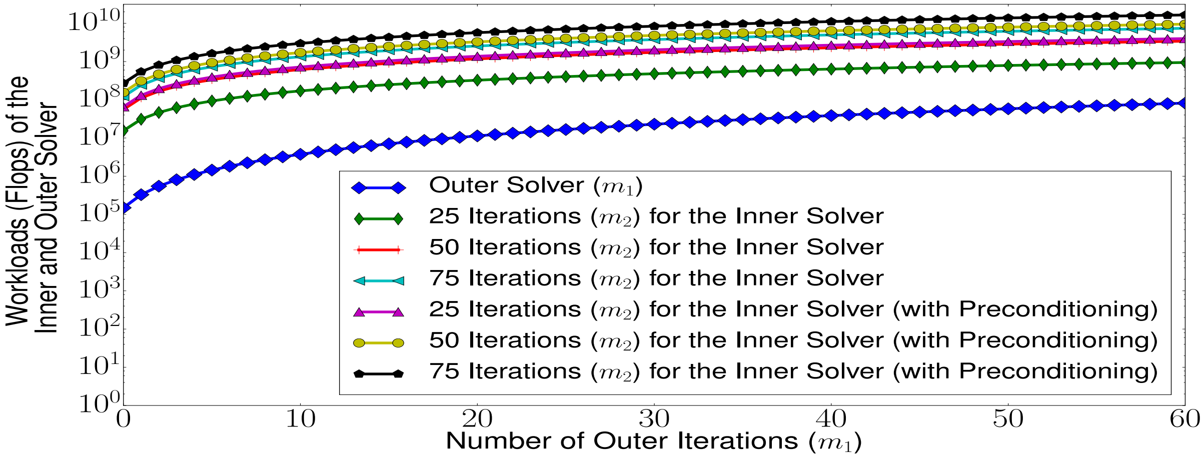


Figure 62 Workloads for the inner and outer solver of the F-GMRES with and without ILU-preconditioning in the case of solving the 2D Poisson matrix.

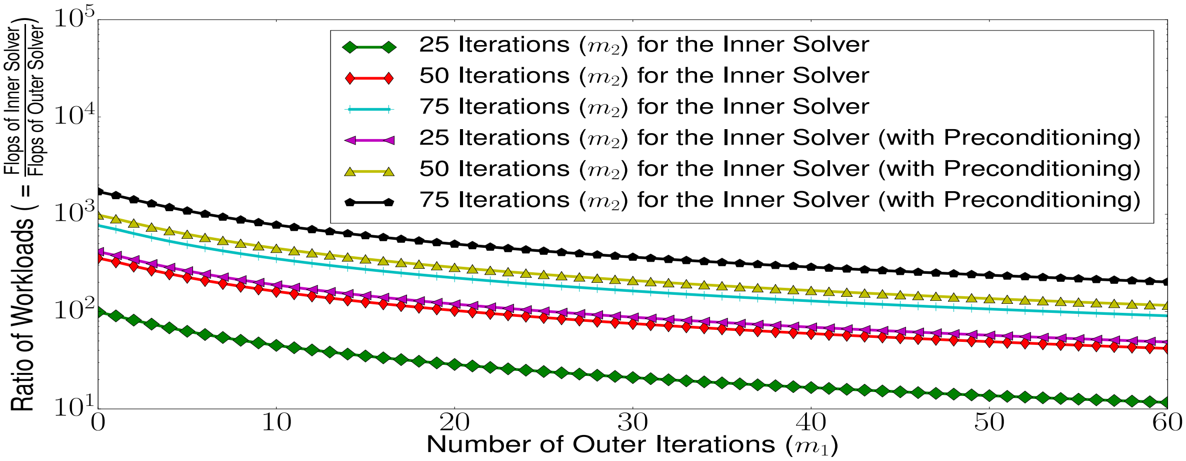
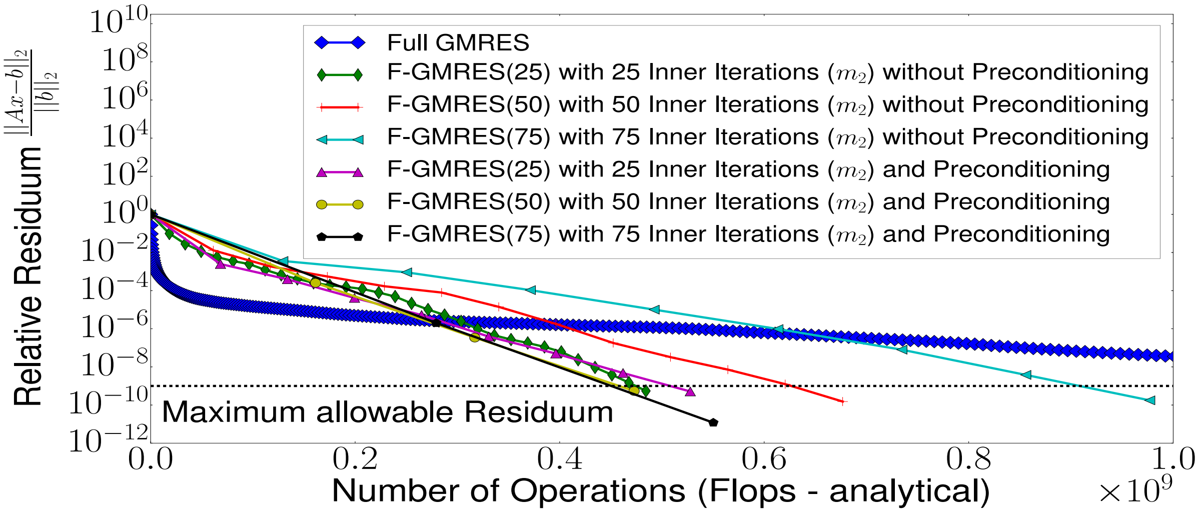


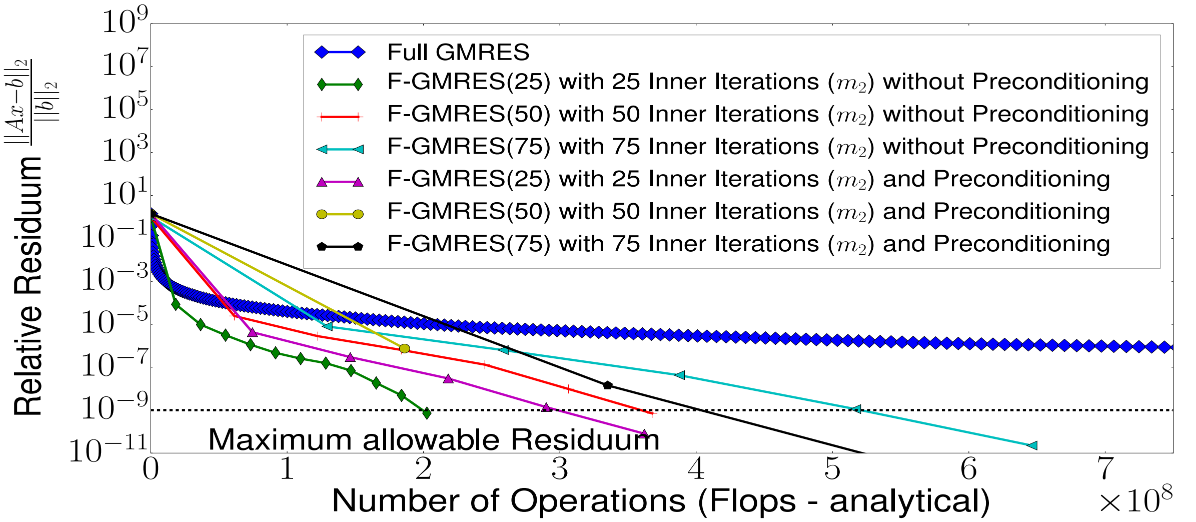
Figure 63 Ratio of workloads of the F-GMRES with and without ILU-preconditioning in the case of solving the 2D Poisson matrix.

In figure 63 the ratio of workloads between the inner and outer solver of the F-GMRES is shown with additional operations for ILU-preconditioning in the case of solving the 2D Poisson matrix (see section 9.4). In figure 62 the according flops for different numbers of iterations of the inner solver are shown like in section 6.1.9, the number of flops for the GMRES solver (see section 6.1.4) can be computed with $Flops_{GMRES}(m, n, A)$ where m is the number of iterations, $nnz(A)$ as the number of non-zero elements and n for the size of matrix A. For the F-GMRES the total workload without preconditioning is given by $Flops_{F-GMRES}(m_1, m_2, k, A) = m_1 \ Flops_{GMRES}(m_2, n, A) + Flops_{GMRES}(m_1, n, A)$ with m_1 for the number of outer iterations. In the worst case there are additional $2m_2n^2$ [47] operations for the inner solver with ILU-preconditioning. For a sparse matrix (2D Poisson) which is also a blocked-tridiagonal matrix forward and backward substitution can be estimated with $3lk^2$ flops [74][75] where k is just the block size such that $n = lk$ (2D Poisson: $n = k^2$) as like for a band matrix with $2k(k+1)l$ operations which is more efficient in terms of flops. Therefore the ratio of flops for the inner and outer solver is $\frac{m_1 \ Flops_{GMRES}(m_2,n,A)}{Flops_{GMRES}(m_1,n,A)} \approx \frac{m_2^2+m_2k(k+1)l}{m_1}$.

9.5.2 Residuum curves of GMRES and Flexible GMRES for solving the 2D Poisson matrix with/without ILU-preconditioning and different initializations for the inner solver



(a) Residuum curves (outer solver) for solving the 2D Poisson problem with and without ILU-preconditioning, the inner solver of the F-GMRES is initialized with random values for vector $w_j(w_0)$.

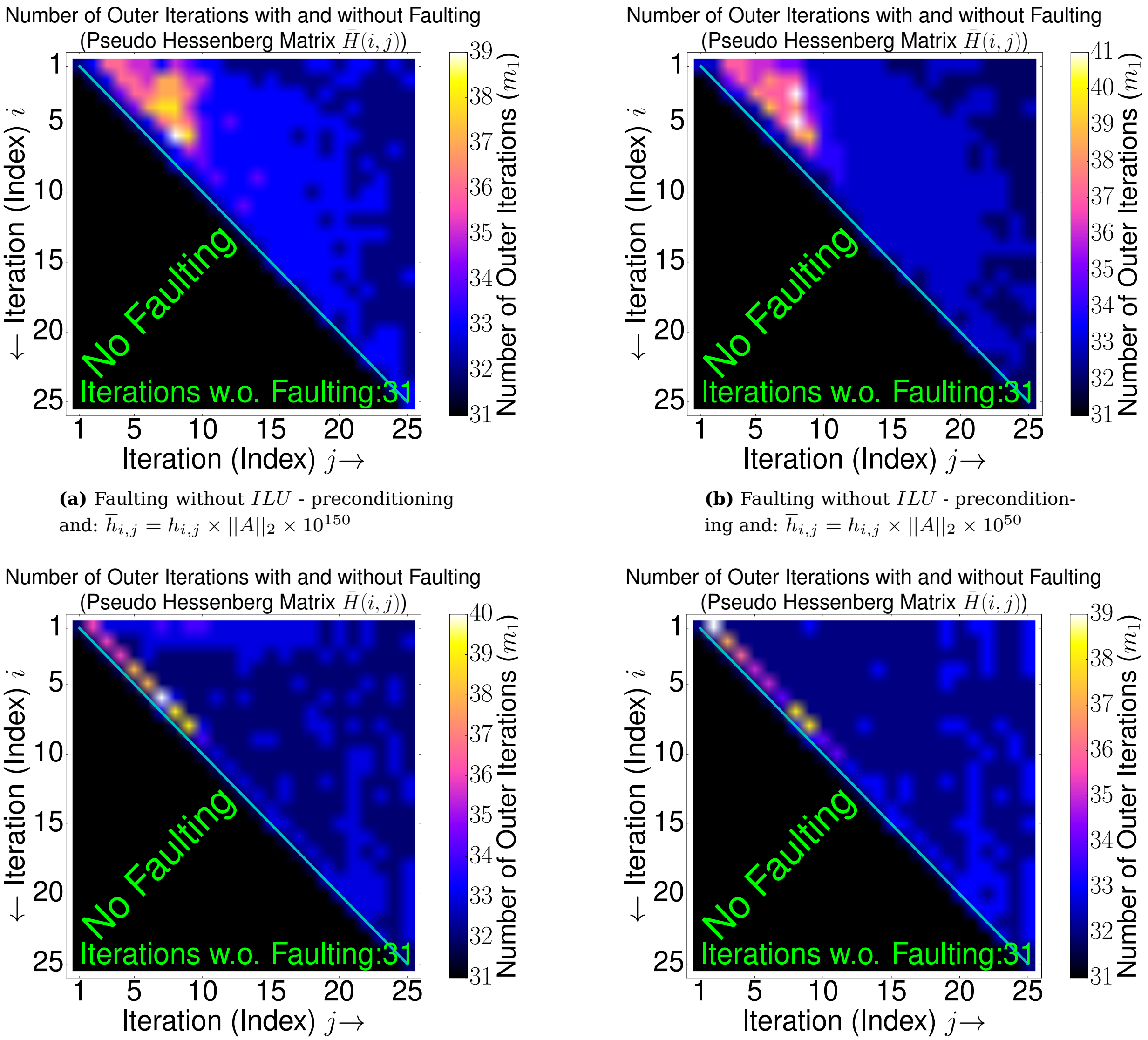


(b) Residuum curves (outer solver) for solving the 2D Poisson problem with and without ILU-preconditioning, the inner solver of the F-GMRES is initialized with zero values for vector $w_j(w_0)$.

Figure 64 Residuum curves (outer solver) for solving the 2D Poisson problem with and without ILU-preconditioning and different initializations for the inner solver (vector w_0) of the F-GMRES.

In figure 64 the according relative residuals ($||r||_2$) are plotted against the number of flops for the F-GMRES solver with different numbers of iterations for the inner solver (m_2). The number of flops is estimated approximately for all executions of the F-GMRES, for the needed number of iterations of the inner and outer solver as in section 6.1.4 to converge to the relative residuum of 10^{-9}. The error of factorization ($Error_{ILU}{}^3$) for matrix A with ILU-preconditioning is chosen below 10^{-1} as in figure 68 in the case of the 2D Poisson matrix. The according number of flops for ILU-preconditioning is also taken into account as in section 9.5.1 which is shown in figure 64. The number of flops for the F-GMRES solver with ILU-preconditioning is about 70 % more in contrast to without preconditioning for the standard solver in figure 64b to achieve the same residuum.

9.5.3 Faulting with different kinds of faults and without ILU-preconditioning during solving the 2D Poisson problem - Inner solver initialized with random values



(a) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

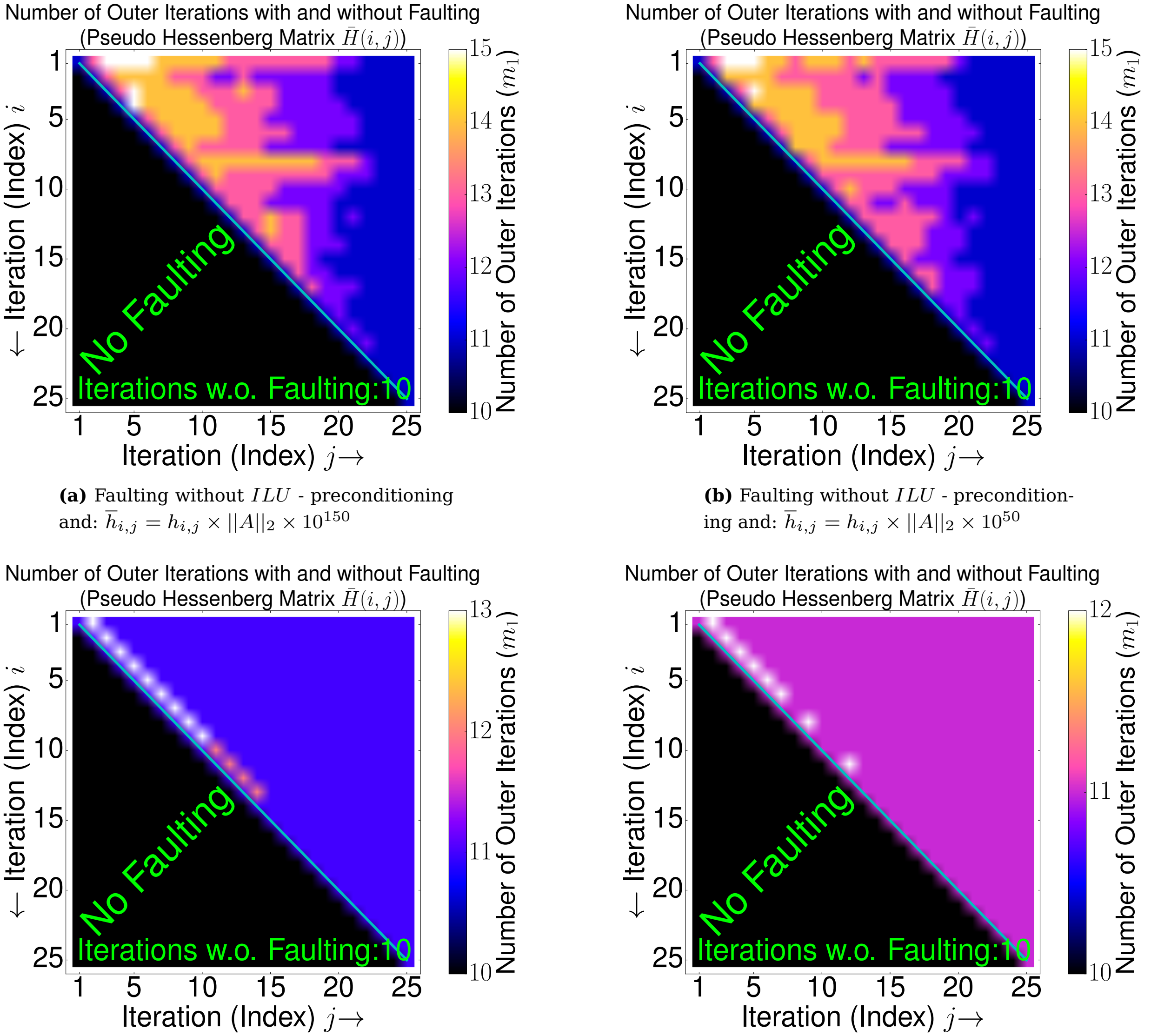
(c) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 1$

Figure 65 Number of outer iterations for a single fault on a specific position during orthogonalization ($h_{i,j}$) without ILU-preconditioning while solving the 2D Poisson matrix.

In figure 65 the 2D Poisson problem is solved. For the first trial in figure 65 it takes 31 outer iterations (m_1) for the F-GMRES without faulting whereas 25 iterations (m_2) are set for the inner solver. Faulting with large perturbations (10^{150}, 10^{50}) is more critical for all positions ($h_{i,j}$) than faulting with perturbations of 10^{10}. Figure 65c and 65d show that smaller perturbations are less problematic for all positions. The worst overhead is 28 % (41/32) because of the highest number of outer iterations in the presence of a single fault. The relative residuum is below 10^{-9} for all trials and computations.

9.5.4 Faulting with different kinds of faults and without ILU-preconditioning during solving the 2D Poisson problem - Inner solver initialized with zero values



(a) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

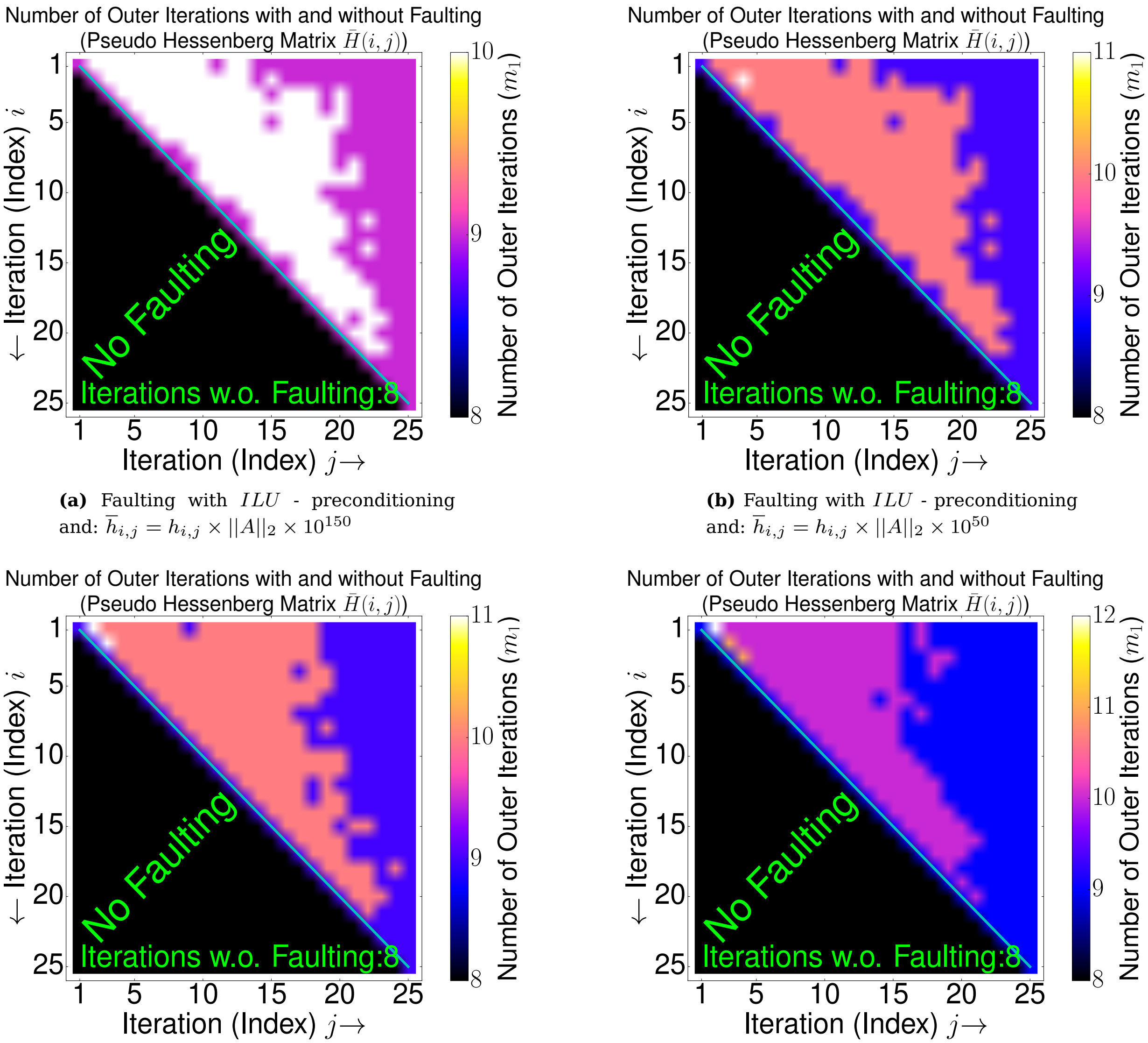
(c) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 1$

Figure 66 Number of outer iterations for a single fault on a specific position during orthogonalization ($h_{i,j}$) without ILU-preconditioning while solving the 2D Poisson matrix.

In figure 66 the 2D Poisson problem is solved. For the first trial in figure 66 it takes 10 outer iterations (m_1) for the F-GMRES without faulting whereas 25 iterations (m_2) are set for the inner solver. Faulting with large perturbations (10^{150}, 10^{50}) is more critical for all positions ($h_{i,j}$) than faulting with perturbations of 10^{10}. Figure 66c and 66d show that smaller perturbations are less problematic for all positions. The worst overhead is 36 % (15/11) because of the highest number of outer iterations in the presence of a single fault. The relative residuum is below 10^{-9} for all trials and computations.

9.5.5 Faulting with different kinds of faults and ILU-preconditioning during solving the 2D Poisson problem - Inner solver initialized with random values



(a) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

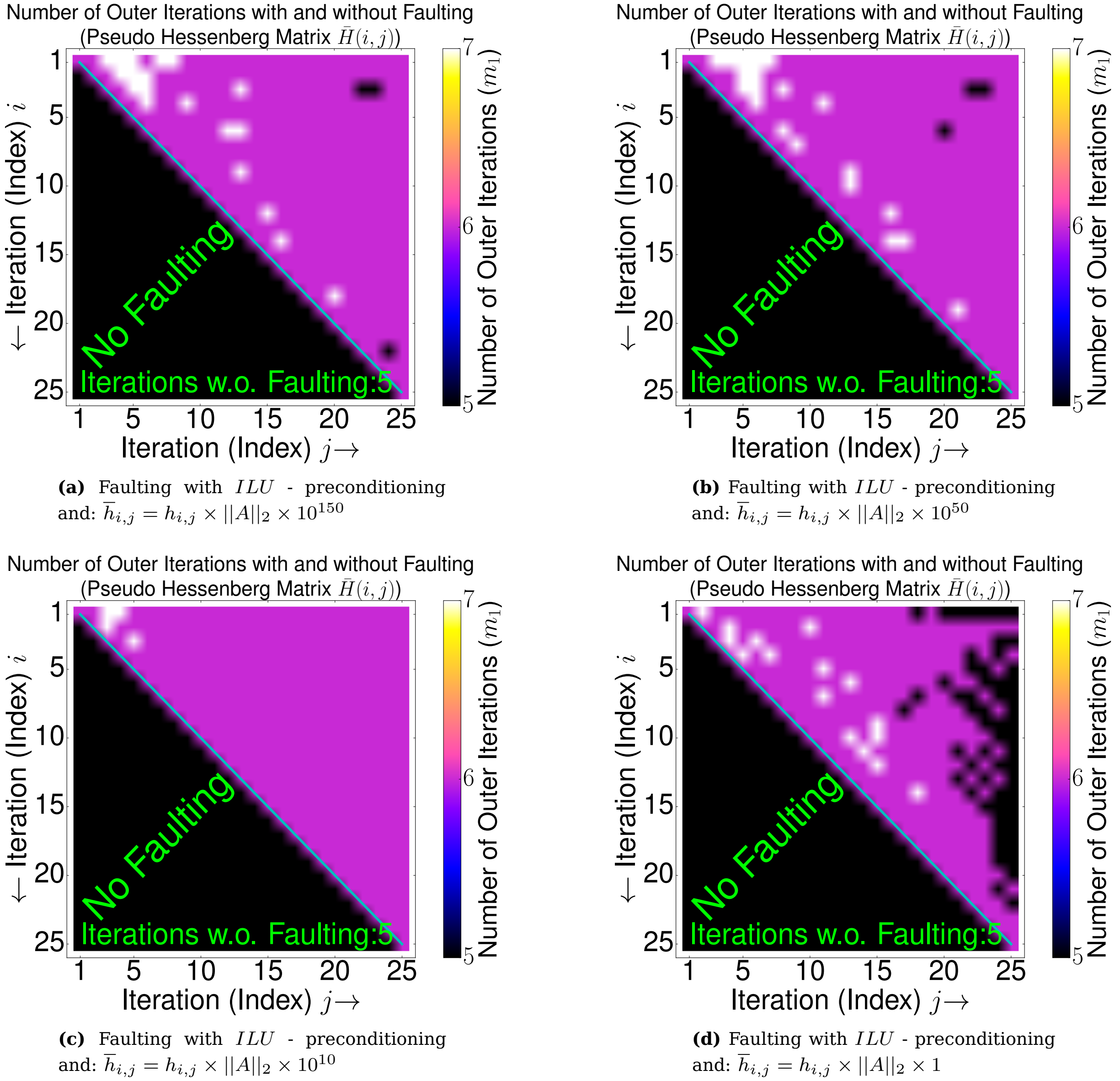
(c) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 1$

Figure 67 Number of outer iterations for a single fault on a specific position during orthogonalization ($h_{i,j}$) with ILU-preconditioning while solving the 2D Poisson matrix (and $Error_{ILU}{}^3 \approx 9 \times 10^{-2}$).

In figure 67 the 2D Poisson problem is solved with ILU-preconditioning. For the first trial in figure 67 it takes 8 outer iterations (m_1) for the F-GMRES without faulting whereas 25 iterations (m_2) are set for the inner solver. The worst overhead is 33 % (12/9) since in the worst case it takes three additional iterations in the presence of a single fault. The advantage of using ILU-preconditioning comes mainly by reducing the area where additional iterations are needed because of a single fault such that there is a slight improvement. The inner solver of the F(T)-GMRES builds an upper Hessenberg matrix as in formula 104. The relative residuum is below 10^{-9} for all trials.

9.5.6 Faulting with different kinds of faults and ILU-preconditioning during solving the 2D Poisson problem - Inner solver initialized with zero values



(a) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

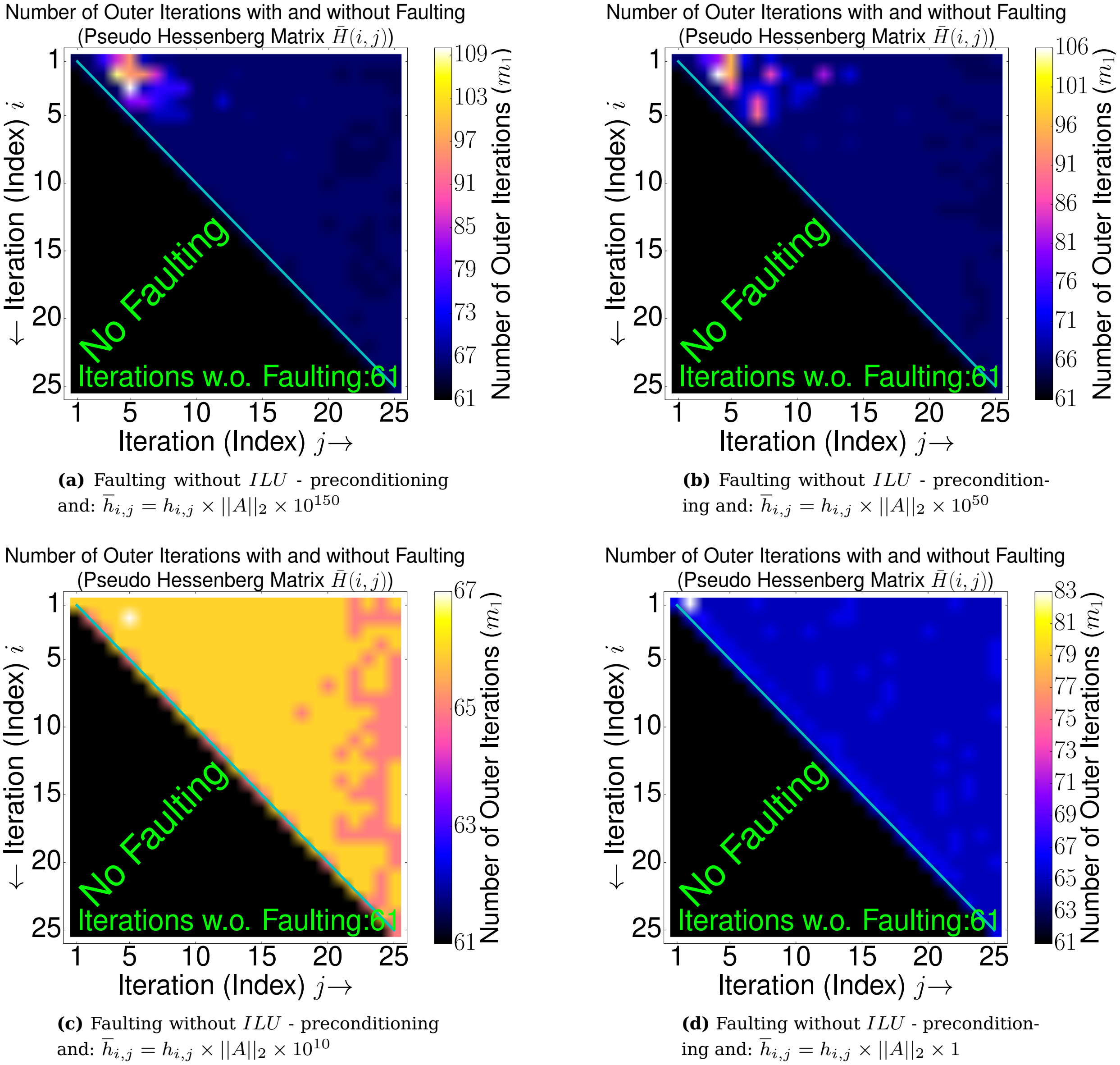
(c) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 1$

Figure 68 Number of outer iterations for a single fault on a specific position during orthogonalization ($h_{i,j}$) with ILU-preconditioning while solving the 2D Poisson matrix (and $Error_{ILU}{}^3 \approx 9{\times}10^{-2}$).

In figure 68 the 2D Poisson problem is solved with ILU-preconditioning. For the first trial in figure 68 it takes 5 outer iterations (m_1) for the F-GMRES without faulting whereas 25 iterations (m_2) are set for the inner solver. The worst overhead is 16 % (7/6) since in most cases it takes only one additional iteration in the presence of a single fault. The advantage of using ILU-preconditioning comes mainly by reducing the area where additional iterations are needed. There is an evident reduce of additional iterations. The inner solver is initialized with zero values, it builds an upper Hessenberg matrix as in formular 104. The relative residuum is below 10^{-9} for all computations.

9.5.7 Faulting with different kinds of faults and without ILU-preconditioning during solving the adder_dcop_63 problem - Inner solver initialized with random values



(a) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

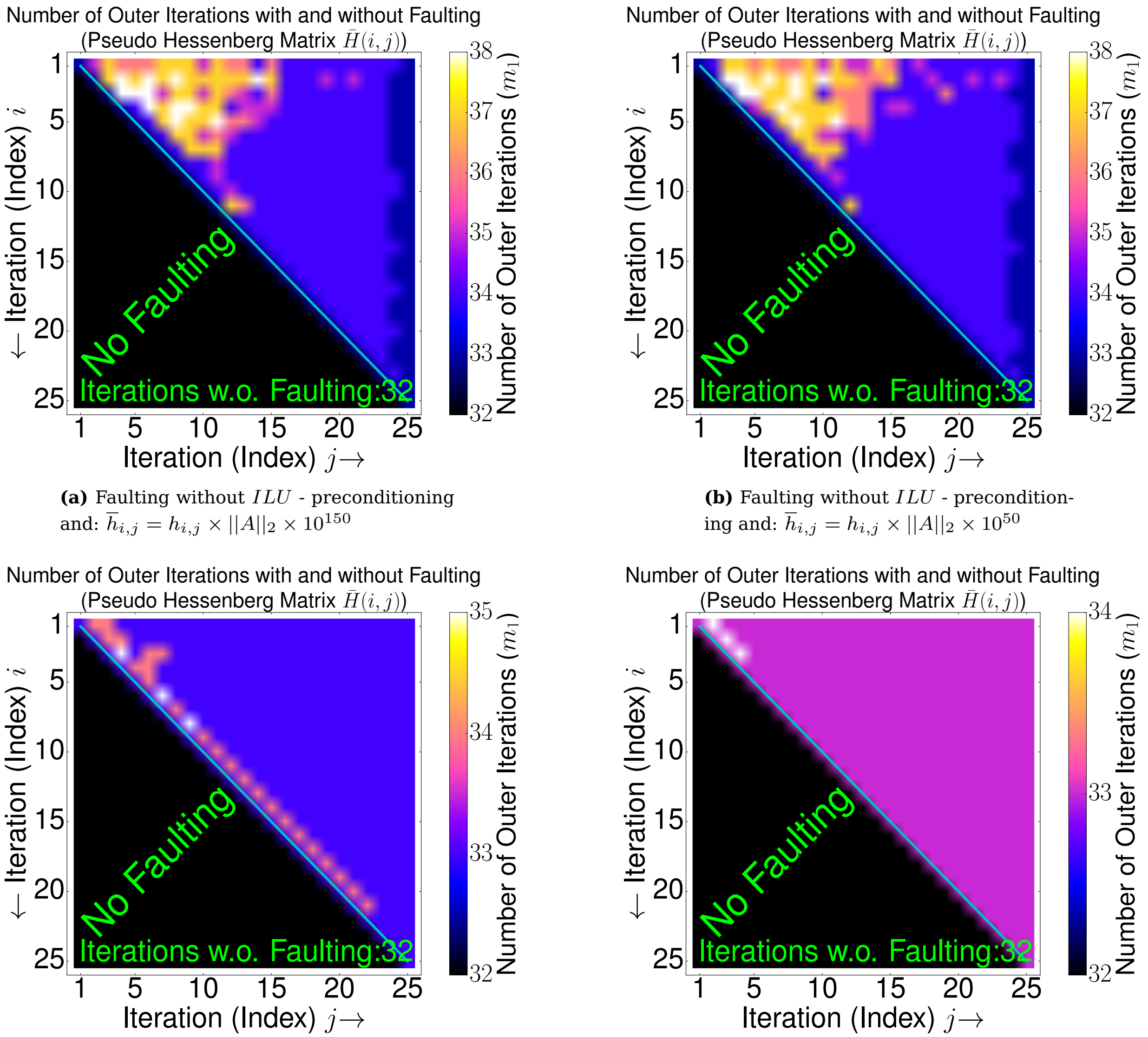
(c) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 1$

Figure 69 Number of outer iterations for a single fault on a specific position during orthogonalization ($h_{i,j}$) without ILU-preconditioning while solving the adder_dcop_63 matrix.

In figure 69 the adder_dcop_63 problem is solved where large perturbations (10^{150} and 10^{50}) are still a problem but the space is much more smaller where faulting is more critical than faulting during solving the 2D Poisson matrix. For this problem it takes 61 outer iterations (m_1) and 25 (m_2) for the inner solver without faulting of the F-GMRES, for the first trial and computation. The worst caused overhead is 76 % (109/62) which is high but only for single positions ($h_{i,j}$). The relative residuum is below 10^{-9} for all trials and computations.

9.5.8 Faulting with different kinds of faults and without ILU-preconditioning during solving the adder_dcop_63 problem - Inner solver initialized with zero values



(a) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

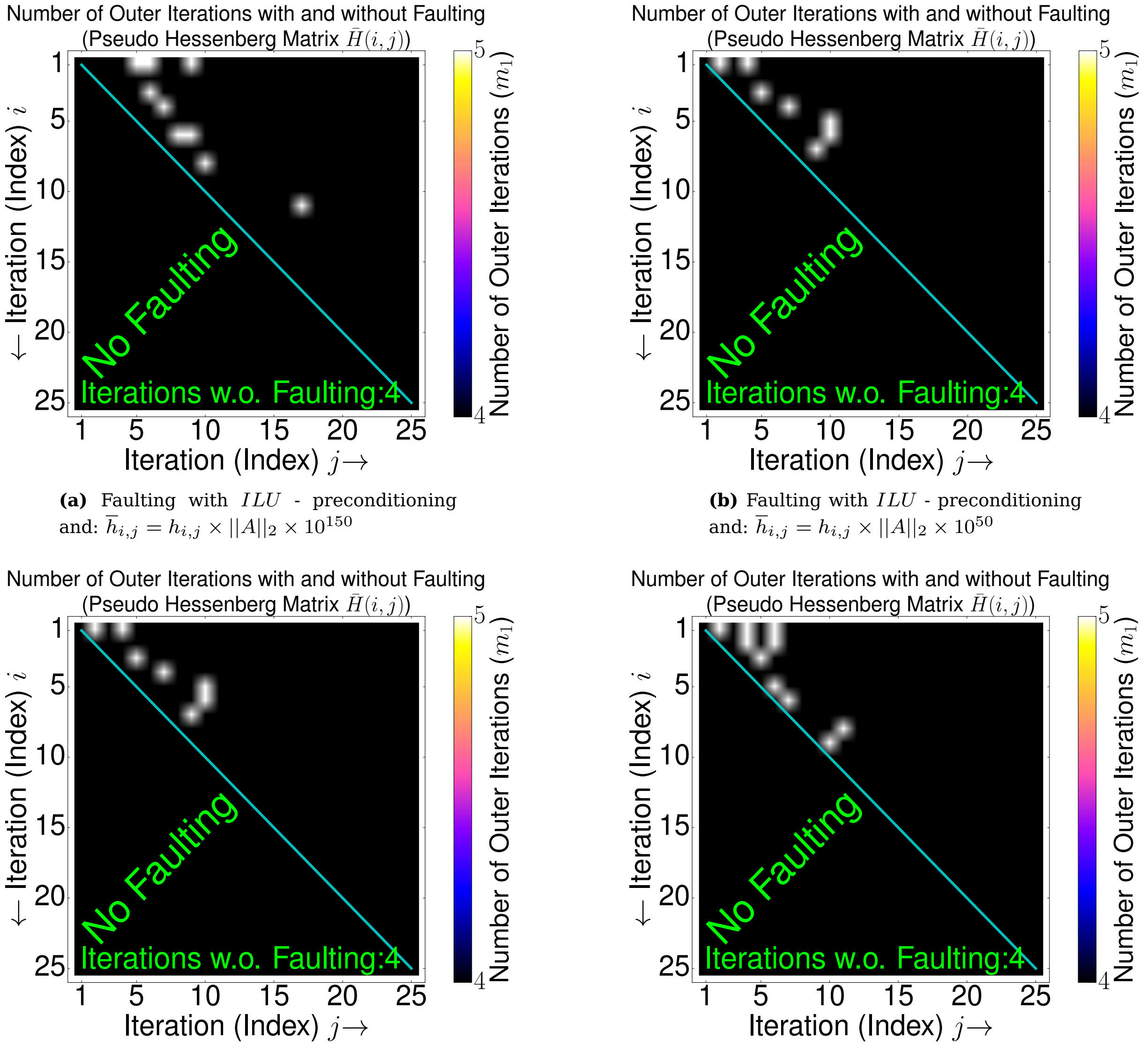
(c) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting without ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 1$

Figure 70 Number of outer iterations for a single fault on a specific position during orthogonalization ($h_{i,j}$) without ILU-preconditioning while solving the adder_dcop_63 matrix.

In figure 70 the adder_dcop_63 problem is solved where large perturbations (10^{150} and 10^{50}) are still a problem. The worst overhead for a single fault is given by 15 % (38/33) which is really less in comparison to figure 69. In figure 70c and 70d faulting is a problem on some specific positions ($h_{i,j}$) which means that smaller perturbations lead to some overhead like with faulting of 10^{10}. Solving this problem takes 32 outer (m_1) and 25 inner iterations (m_2) for the F-GMRES solver, for the first trial and computation without a fault, which is the half as in figure 69. The relative residuum is below 10^{-9} for all computations and trials.

9.5.9 Faulting with different kinds of faults and ILU-preconditioning during solving the adder_dcop_63 problem - Inner solver initialized with random values



(a) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

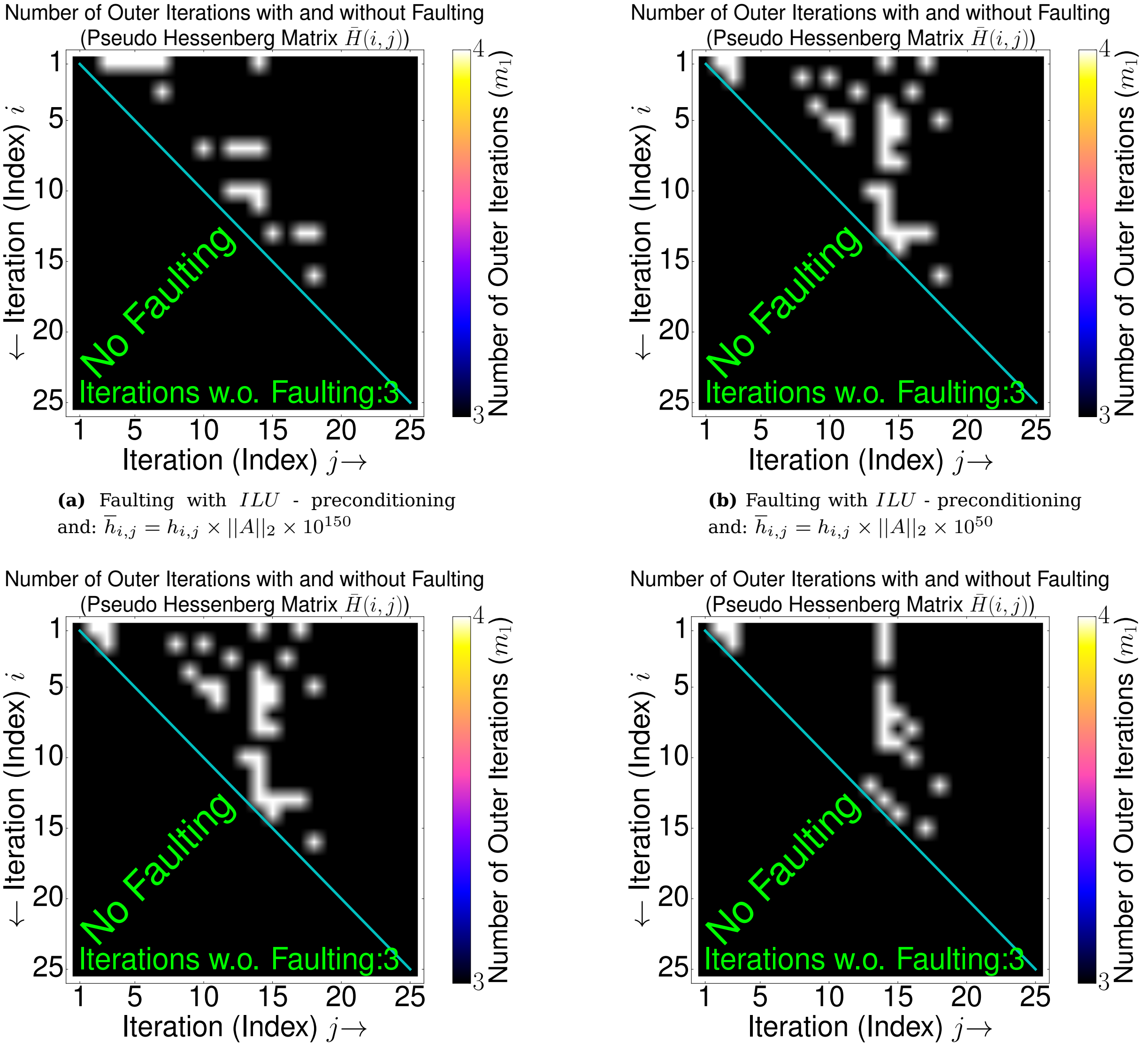
(c) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 1$

Figure 71 Number of outer iterations for a single fault on a specific position during orthogonalization with ILU-preconditioning while solving the adder_dcop_63 matrix (and $Error_{ILU}{}^3 \approx 3 \times 10^{-1}$).

In figure 71 the adder_dcop_63 problem is solved with ILU-preconditioning. For the first trial in figure 71 it takes 4 outer iterations (m_1) for the F-GMRES without faulting of the inner solver whereas 25 iterations (m_2) are set for the inner solver. There is no overhead caused according to the definition in formula 107. The advantage of using ILU-preconditioning comes mainly by reducing the area where additional iterations are needed because of a single fault such that there is a great improvement. The inner solver of the F(T)-GMRES builds an upper Hessenberg matrix as in formula 104. The relative residuum is below 10^{-9} for all trials and computations.

9.5.10 Faulting with different kinds of faults and ILU-preconditioning during solving the adder_dcop_63 problem - Inner solver initialized with zero values



(a) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}$

(c) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with ILU - preconditioning and: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 1$

Figure 72 Number of outer iterations for a single fault on a specific position during orthogonalization with ILU-preconditioning while solving the adder_dcop_63 matrix (and $Error_{ILU}{}^3 \approx 3 \times 10^{-1}$).

In figure 72 the adder_dcop_63 problem is solved with ILU-preconditioning. For the first trial in figure 72 it takes 3 outer iterations (m_1) for the F-GMRES without faulting of the inner solver whereas 25 iterations (m_2) are set for the inner solver. There is no overhead caused according to the definition in formula 107. The advantage of using ILU-preconditioning comes mainly by reducing the area where additional iterations are needed because of a single fault such that there is a great improvement. The inner solver of the F(T)-GMRES builds an upper Hessenberg matrix as in formula 104. The relative residuum is below 10^{-9} for all trials and computations.

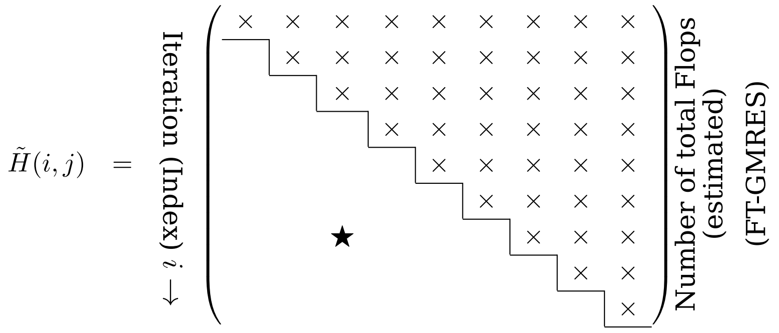
9.6 Faulting while solving the 2D Poisson and adder_dcop_63 problem with flexible preconditioning by the inner solver of the FT-GMRES

This section is mainly related to flexible preconditioning, flexible preconditioning mainly helps to decrease the needed number of iterations for the inner solver (m_1). Therefore it also helps to decrease the number of operations for the inner solver (line 4 of algorithm 14) and as well of the whole F(T)-GMRES solver because most of the work should be done in the inner solver of the F-GMRES and FT-GMRES solver. The inner solver is stopped if a specific accuracy (ϵ) for the solution vector (w_j) is achieved, this approach mainly avoids unnecessary work. This is also the main contrast between (right) preconditioning where always the same number of iterations (work) for the inner solver is applied and flexible preconditioning where different numbers of iterations (m_2) are possible for the inner solver.

There are different possibilities how flexible preconditioning can be done. The first possibility is to check the accuracy (ϵ_2) of the solution vector (w_j) in the inner solver with $||(q_{j+1} - Aw_{j+1})||_2/||q_{j+1}||_2 \leq \epsilon_2$. If the desired accuracy (ϵ_2) of the solution vector (w_j) is achieved the inner solver is stopped independent of the maximum number of iterations (m_2) for the inner solver. The expression $||(q_{j+1} - Aw_{j+1})||_2/||q_{j+1}||_2 \leq \epsilon_2$ is similar to $||(b - Ax||_2/||b||_2 \leq \epsilon$ which measures the accuracy of the outer solver (m_1) for vector x. The only thing which has changed is the right hand side of vector $q_j(b)$ and the according solution vector $w_j(x)$ where both expressions measure the accuracy (ϵ) of the inner (w_j) and outer solver (x) for the right hand side q_j and b.

The second possibility is just to check the linear in-dependency of two residuals with the expression of $1 - \frac{||r_j||_2}{||r_0||_2} \geq \delta_g$ [7] with $r_j = b - Ax_j$ as the current residuum and $r_0 = b - Ax_0$ as the start residuum some more information can be found in section 10.1. A value of $\delta_g = 1$ indicates linear in-dependency whereas a value $\delta_g = 0$ means that r_j and r_0 are still equivalent such that further iterations (m_2) have to be done for the inner solver of the F(T)-GMRES. Furthermore because of the fact that computing r_j is really expensive in terms of operations the approximation error (see section 5) is used instead with $||r_j||_2 \approx ||\sigma_2||_2^2$ and $||r_0||_2 \approx ||\sigma_0||_2^2$ such that $1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2} \geq \delta_g$. The F(T)-GMRES will do different iterations if flexible preconditioning is applied therefore the overhead is measured in floating point operations (see section 6.1.4), the related flops are shown in figure 110. In the case of flexible preconditioning with different numbers of iterations for the inner solver the total flops are summed up for each execution of the inner solver respective outer iteration. Only positions which are shown in figure 110 are tested with a single fault and the related error.

Matrix $\tilde{H}(i,j)$ which shows the number of total operations (estimated flops) in the presence of a single fault for each position of $h_{i,j}$ in the inner solver of the FT-GMRES:



(110)

Figure 73 Matrix $\tilde{H}(i,j)$ which visualizes where faulting is applied during the orthogonalization process ($h_{i,j}$) in the inner solver of the FT-GMRES with the according number of operations (flops).

9.6.1 The absolute change (δ) between the current (σ_j) and previous iteration (σ_{j-1}) in the inner solver of the F-GMRES - Inner solver initialized with zero values

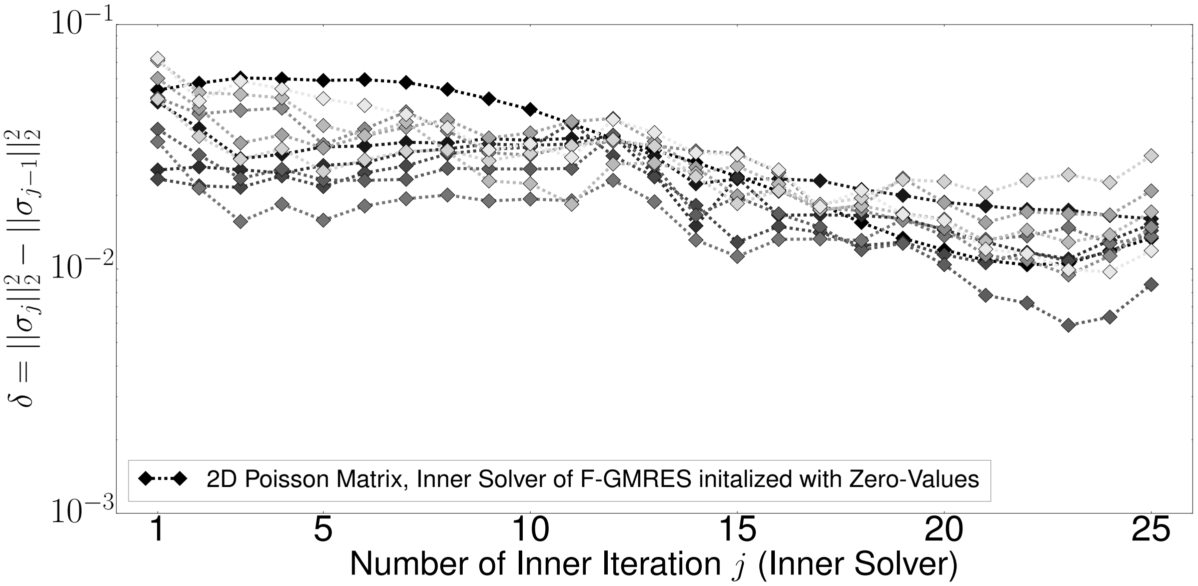


Figure 74 The absolute change (δ) between the current (σ_j) and previous iteration (σ_{j-1}) in the inner solver of the F-GMRES during solving the 2D Poisson matrix for 25 inner iterations (m_2).

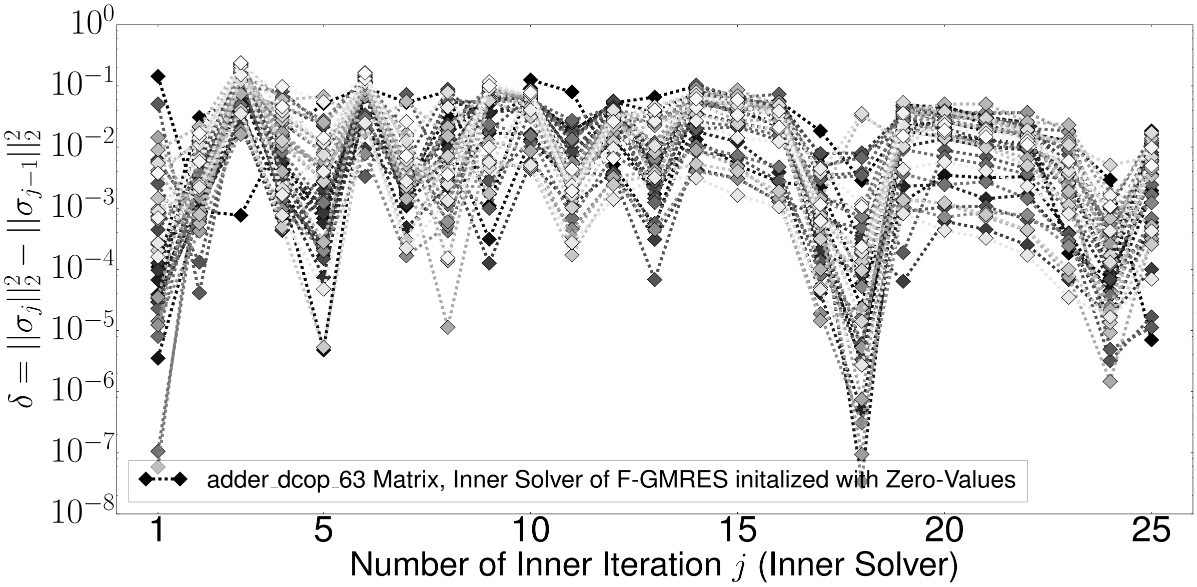


Figure 75 The absolute change (δ) between the current (σ_j) and previous iteration (σ_{j-1}) in the inner solver of the F-GMRES during solving the adder_dcop_63 matrix for 25 inner iterations (m_2).

In figure 74 and 75 the absolute change (δ) between the current (σ_j) and previous iteration (σ_{j-1}) in the inner solver of the F-GMRES is shown for all outer iterations (executions of the inner solver). In figure 75 there is a wider range for the change of the residuals (δ) in contrast to figure 74 where the 2D Poisson matrix is solved. The color goes from black (first outer iteration - first execution of the inner solver) to white (last outer iteration - last execution of the inner solver). The relative residuum is below 10^{-9} for all computations and trails of the F-GMRES in figure 74 and 75.

9.6.2 The relative change (δ_g) between the first (σ_0) and current iteration (σ_j) in the inner solver of the F-GMRES - Inner solver initialized with zero values

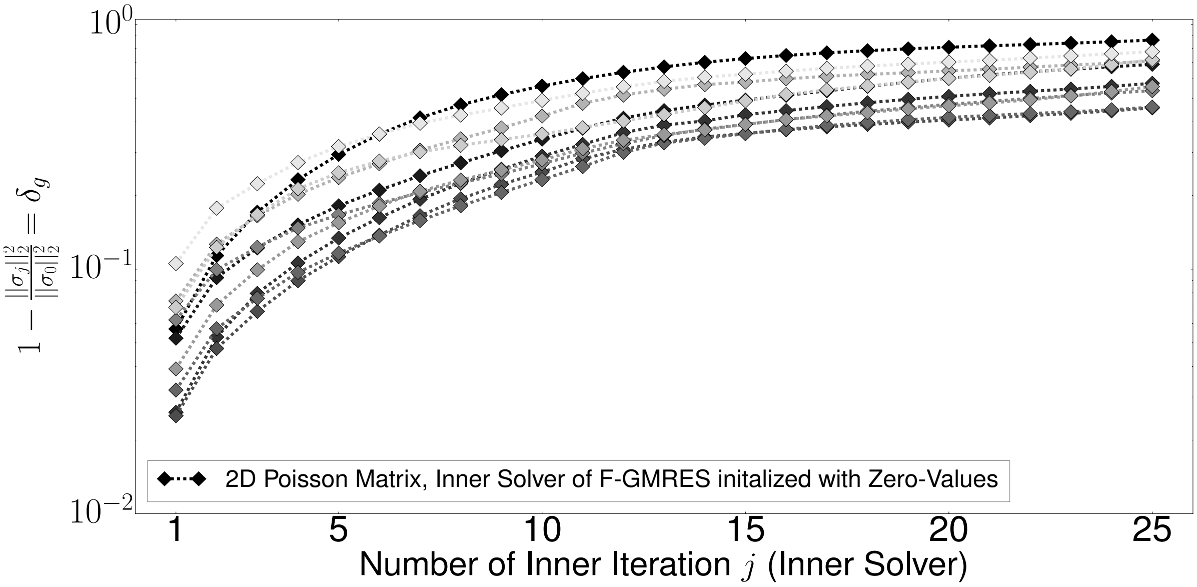


Figure 76 The relative change (δ_g) between the first (σ_0) and current iteration (σ_j) in the inner solver of the F-GMRES during solving the 2D Poisson matrix for 25 inner iterations (m_2).

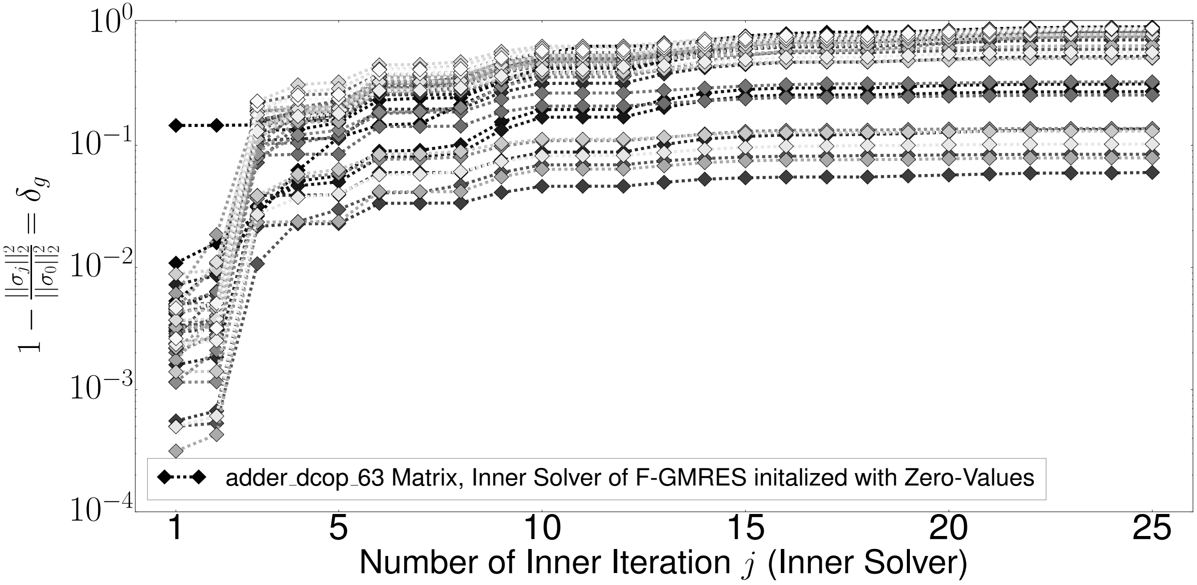


Figure 77 The relative change (δ_g) between the first (σ_0) and current iteration (σ_j) in the inner solver of the F-GMRES during solving the adder_dcop_63 matrix for 25 inner iterations (m_2).

In figure 76 and 77 the relative change (δ_g) between the first (σ_0) and current iteration (σ_j) in the inner solver of the F-GMRES is shown for all outer iterations (executions of the inner solver). These figures show that the value of δ_g decreases (tendencially) from 1 to 0 with increasing the number of outer iteration (execution of the inner solver), in the inner solver of the F-GMRES. It means it takes more iterations for the inner solver to achieve $\delta_g = 1$. The color goes from black (first outer iteration - first execution of the inner solver) to white (last outer iteration - last execution of the inner solver). The relative residuum is below 10^{-9} for all computations and trails of F-GMRES.

9.6.3 Adaptive controlling the number of iterations for the inner solver of the F-GMRES (flexible preconditioning) - Inner solver initialized with zero values

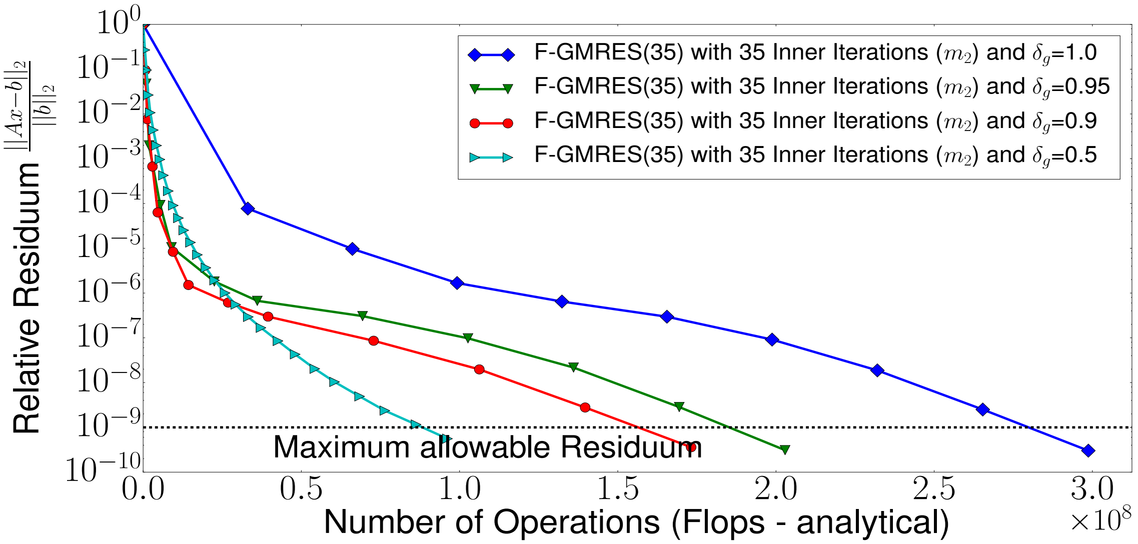


Figure 78 Solving the 2D Poisson matrix with controlling the number of inner iterations (m_2).

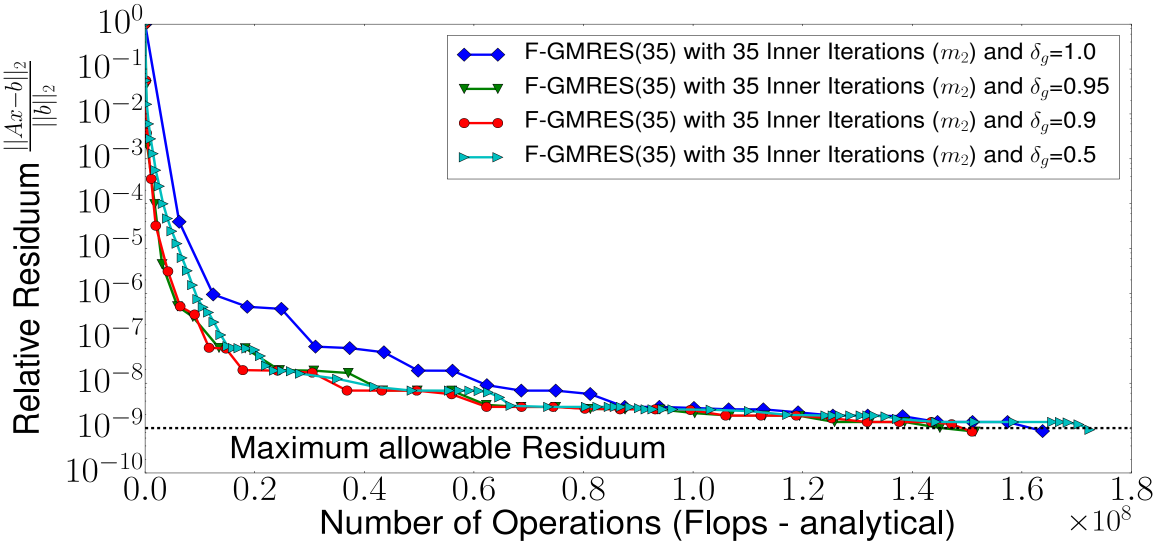


Figure 79 Solving the adder_dcop_63 matrix with controlling the number of inner iterations (m_2).

In figure 78 and 79 the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is examined in the inner solver of the F-GMRES. If a specific value for δ_g is achieved the inner solver is stopped with preconditioning and the outer solver is allowed to continue which leads to a reduction for the number of operations as well to a lowering for the number of iterations of the inner solver. Hence a value of $\delta_g = 1$ means full linear in-dependency such that the inner solver of the F-GMRES has to do preconditioning for all iterations (m_2). In figure 78 there is a reduction of flops with $\delta_g = 0.5$ of about 230 % against a computation with $\delta_g = 1$ but in figure 79 there is less reduction. There are no faults induced while solving these problems only flexible preconditioning is applied. Hence after some experiments with other matrices it was also possible to reduce the number of operations but not for each problem.

9.6.4 Adaptive controlling the number of iterations for the inner solver of the FT-GMRES (flexible preconditioning) - Inner solver initialized with zero values

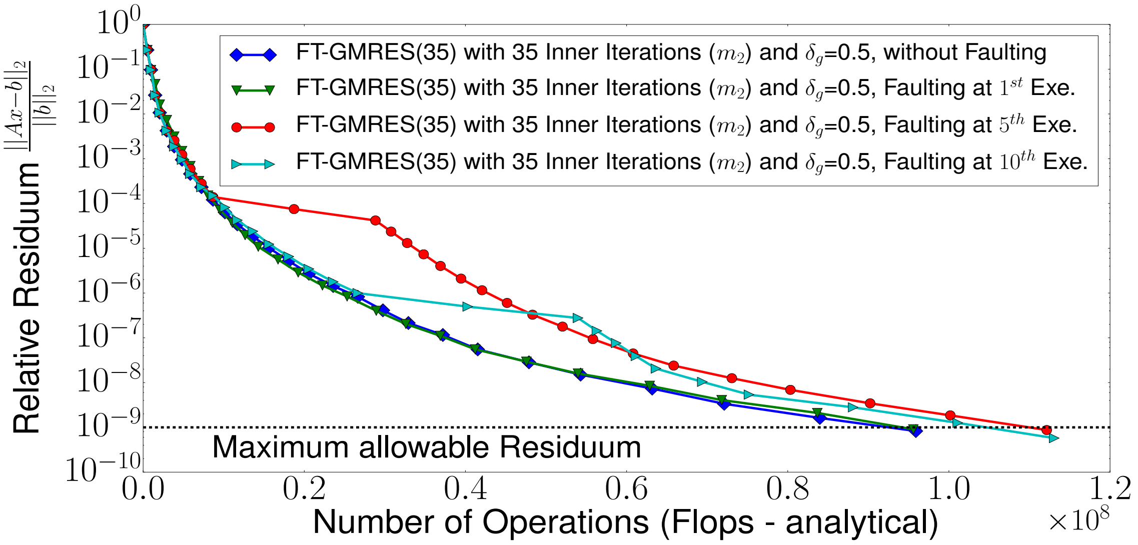


Figure 80 Solving the 2D Poisson matrix with controlling the number of iterations (m_2) for the inner solver. In this case faulting is applied during different executions of the inner solver.

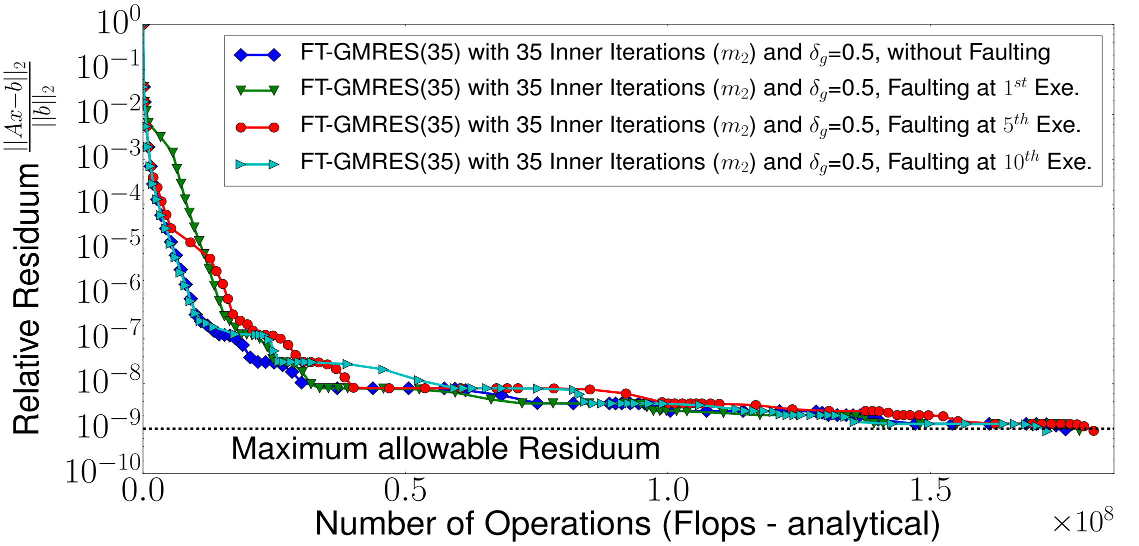
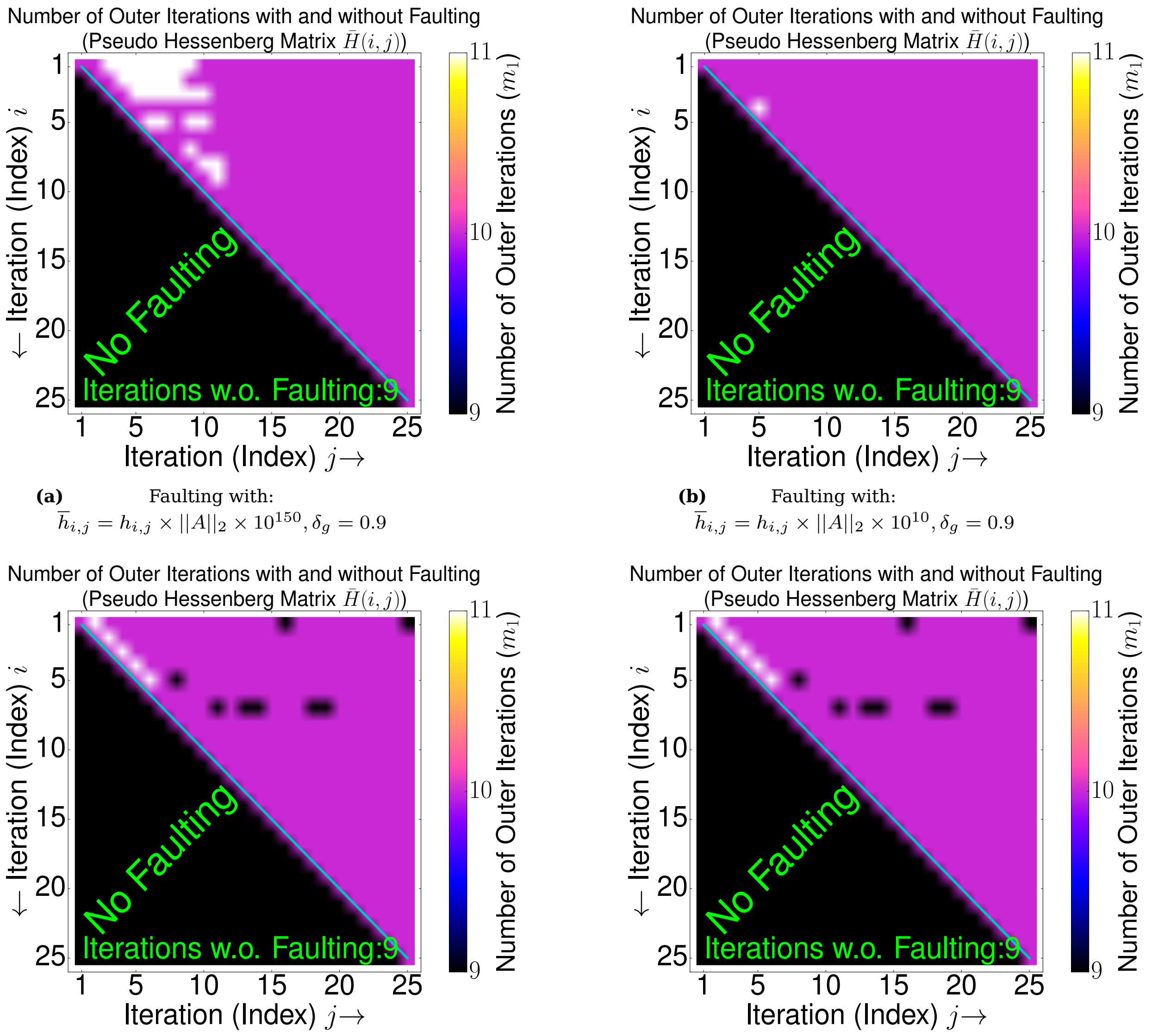


Figure 81 Solving the adder_dcop_63 matrix with controlling the number of iterations (m_2) for the inner solver. In this case faulting is applied during different executions of the inner solver.

As in section 9.6.3 flexible preconditioning is also applied in figure 80 and 81 while solving the 2D Poisson and adder_dcop_63 matrix. The inner solver of the F(T)-GMRES does at least 1 but at most 35 iterations (m_2). If a value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is achieved while preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. Both figures show the residual curves for solving the 2D Poisson and adder_dcop_63 matrix in the case of a single fault ($\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$ on $h_{1,2}$) during different executions of the inner solver. There are additional operations (flops) needed to decrease the effect of a single fault in the inner solver.

9.6.5 Faulting during solving the 2D Poisson problem with flexible preconditioning ($\delta_g = 0.9$) by the inner solver - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 0.9$

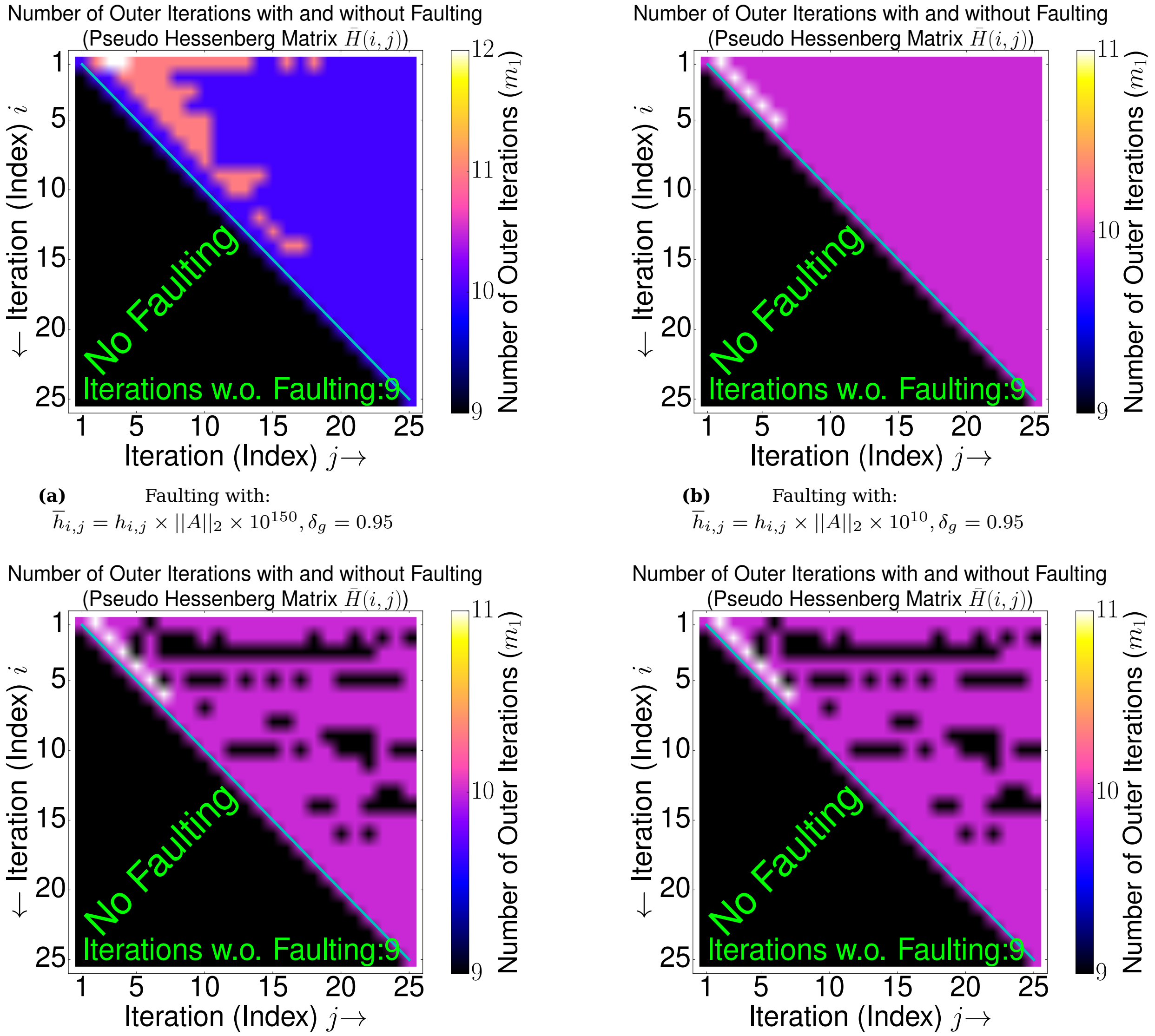
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}, \delta_g = 0.9$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}, \delta_g = 0.9$

Figure 82 Number of outer iterations for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the 2D Poisson matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 82 during solving the 2D Poisson matrix. The inner solver of the FT-GMRES does at least 25 but at most 35 iterations (m_2). If a value of $\delta_g = 0.9$ ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. It is assumed that during these additional iterations (between 25 to 35) there is no fault in the inner solver of the FT-GMRES. The worst overhead is 10 % (11/10) which is an improvement against 36 % in the presence of a single fault. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all computations and trials.

9.6.6 Faulting during solving the 2D Poisson problem with flexible preconditioning ($\delta_g = 0.95$) by the inner solver - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 0.95$

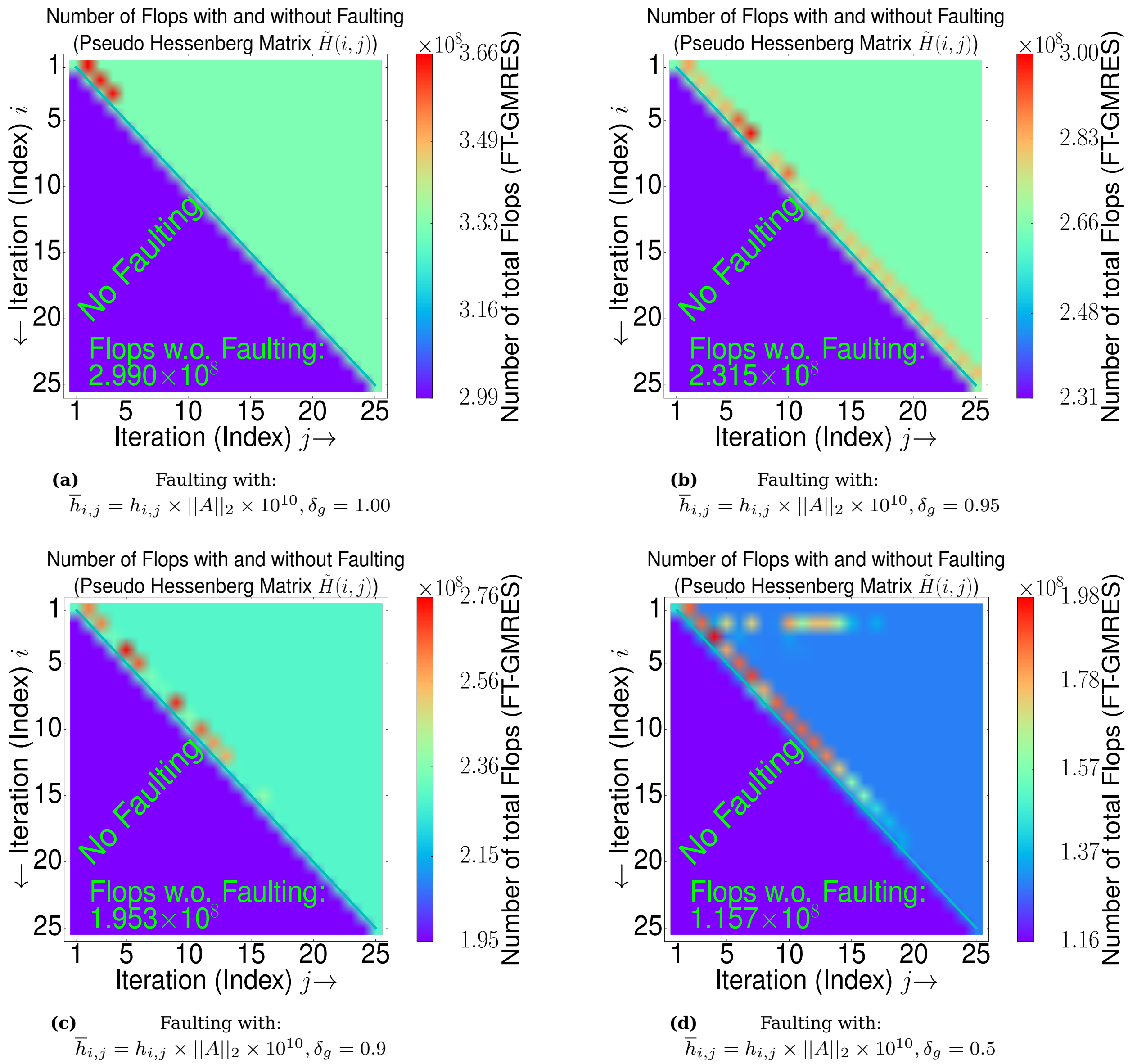
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}, \delta_g = 0.95$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}, \delta_g = 0.95$

Figure 83 Number of outer iterations for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the 2D Poisson matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 83 during solving the 2D Poisson matrix. The inner solver of the FT-GMRES does at least 25 but at most 35 iterations (m_2). If a value of $\delta_g = 0.95$ ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. It is assumed that during these additional iterations (between 25 to 35) there is no fault in the inner solver of the FT-GMRES. The worst overhead is 20 % (12/10) which is still an improvement against 36 % in the presence of a single fault. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all computations and trials.

9.6.7 Number of flops for solving the 2D Poisson problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 1.00$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 0.95$

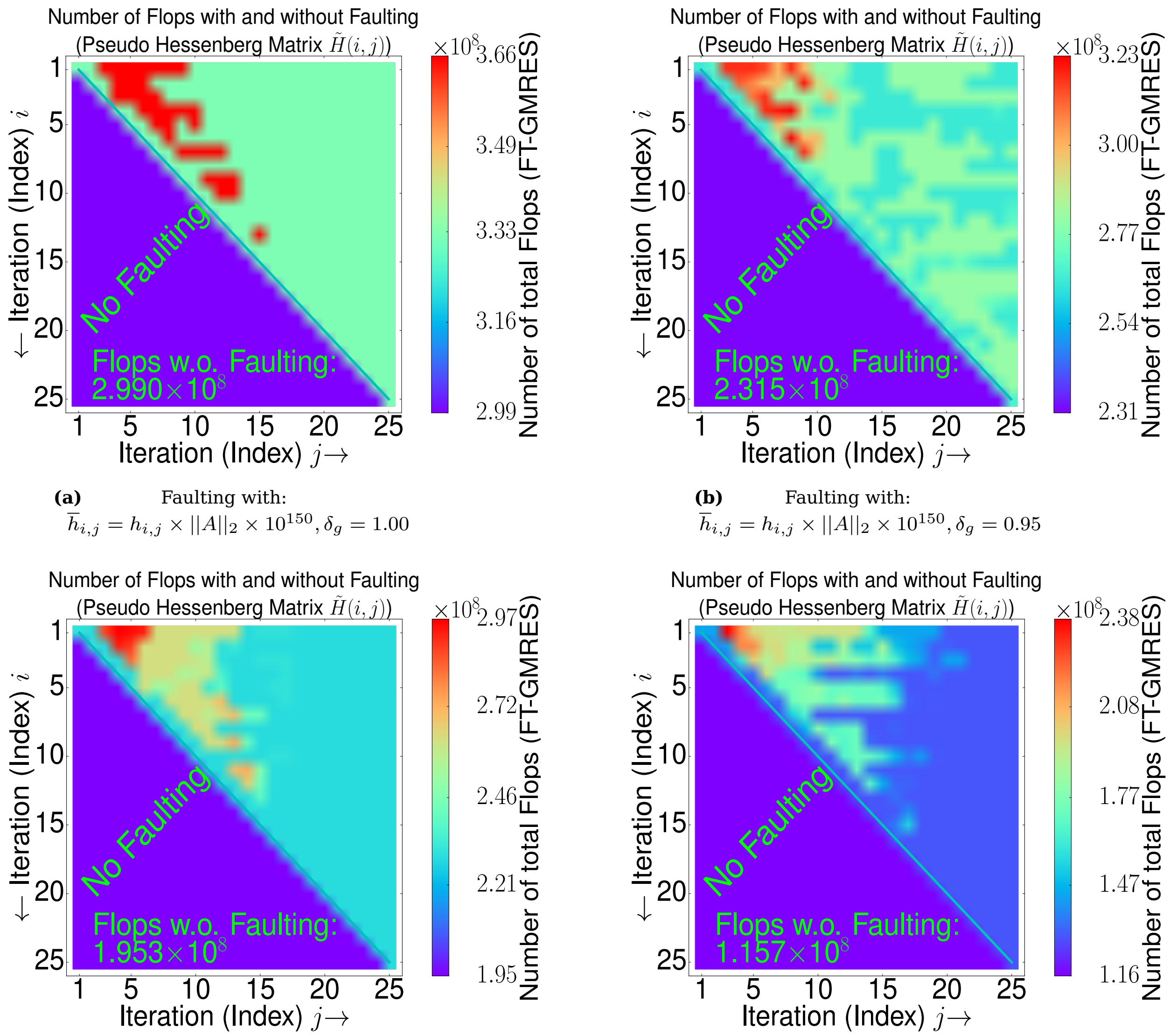
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 0.9$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 0.5$

Figure 84 Number of total flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the 2D Poisson matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 84 during solving the 2D Poisson matrix. The inner solver of the FT-GMRES does at least 5 but at most 35 iterations (m_2). In figure 84 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead (≈ 10 - 50 %) does not remain the same and the total number of flops is different for each different value of δ_g without faulting. In this case flexible preconditioning mainly helps to decrease the needed number of flops without faulting. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all computations.

9.6.8 Number of flops for solving the 2D Poisson problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 1.00$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

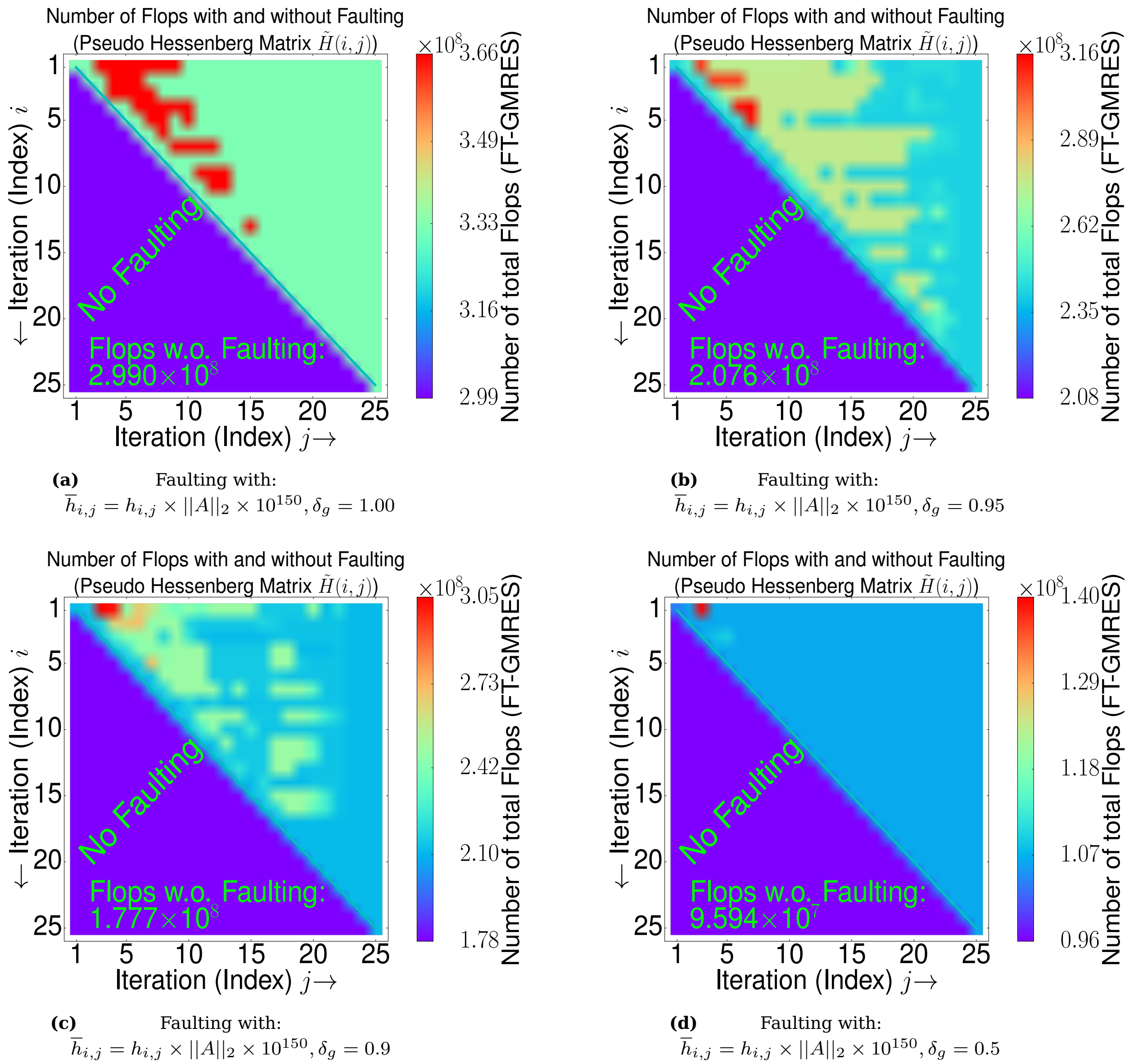
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.5$

Figure 85 Number of total flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the 2D Poisson matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 85 during solving the 2D Poisson matrix. The inner solver of the FT-GMRES does at least 5 but at most 35 iterations (m_2). In figure 85 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead (≈ 10 - 80 %) does not remain the same and the total number of flops is different for each different value of δ_g without faulting. In this case flexible preconditioning mainly helps to decrease the needed number of flops without faulting. The relative residuum ($||r||_2 = ||Ax-b||_2/||b||_2$) is below 10^{-9} for all computations.

9.6.9 Number of flops for solving the 2D Poisson problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 1.00$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

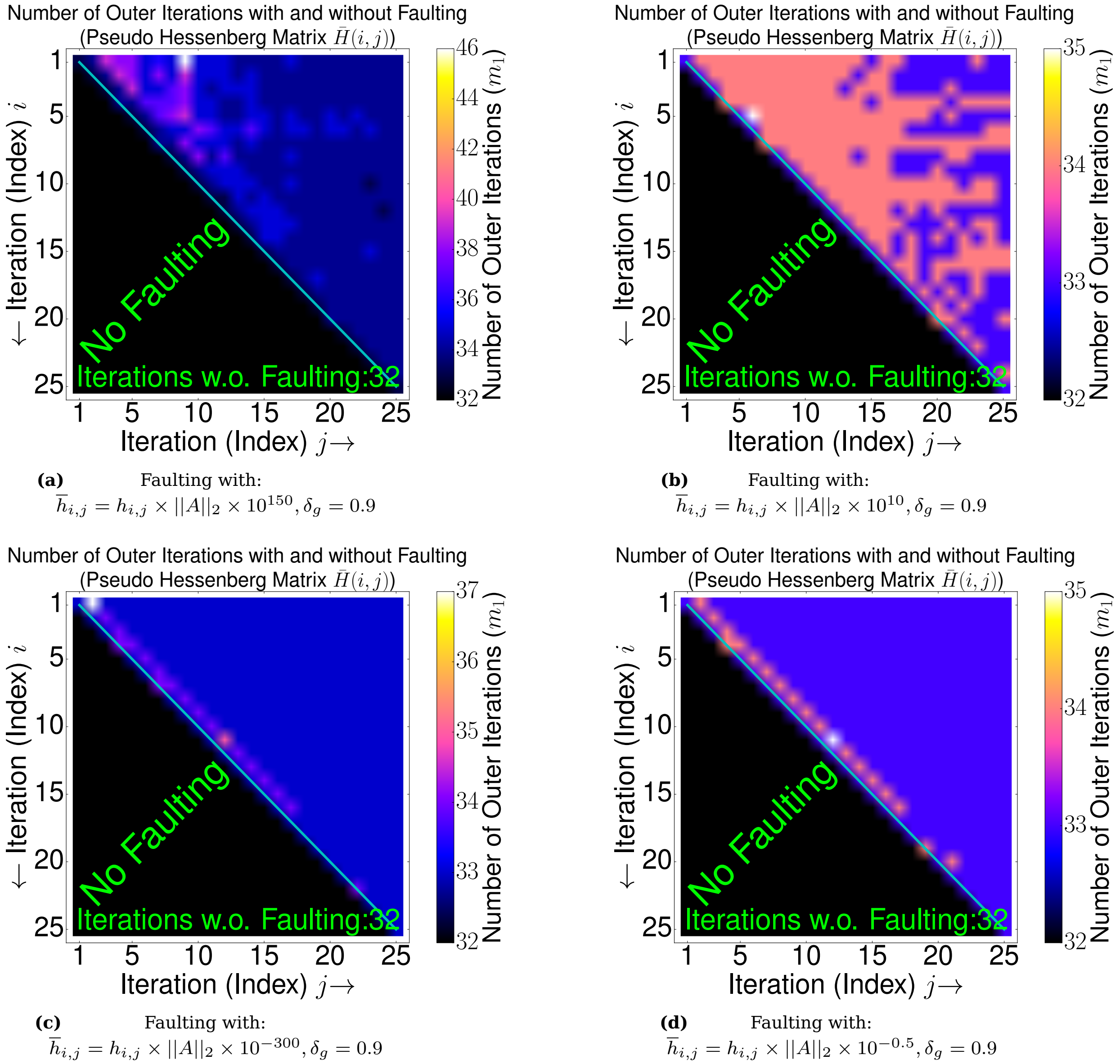
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.5$

Figure 86 Number of total flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the 2D Poisson matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 86 during solving the 2D Poisson matrix. The inner solver of the FT-GMRES does at least 1 but at most 35 iterations (m_2). In figure 86 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead ($\approx$ 10 - 60 %) does not remain the same and the total number of flops is different for each different value of δ_g without faulting. In this case flexible preconditioning mainly helps to decrease the needed number of flops without faulting. The relative residuum ($||r||_2 = ||Ax-b||_2/||b||_2$) is below 10^{-9} for all computations.

9.6.10 Faulting during solving the adder_dcop_63 problem with flexible preconditioning ($\delta_g = 0.9$) by the inner solver - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 0.9$

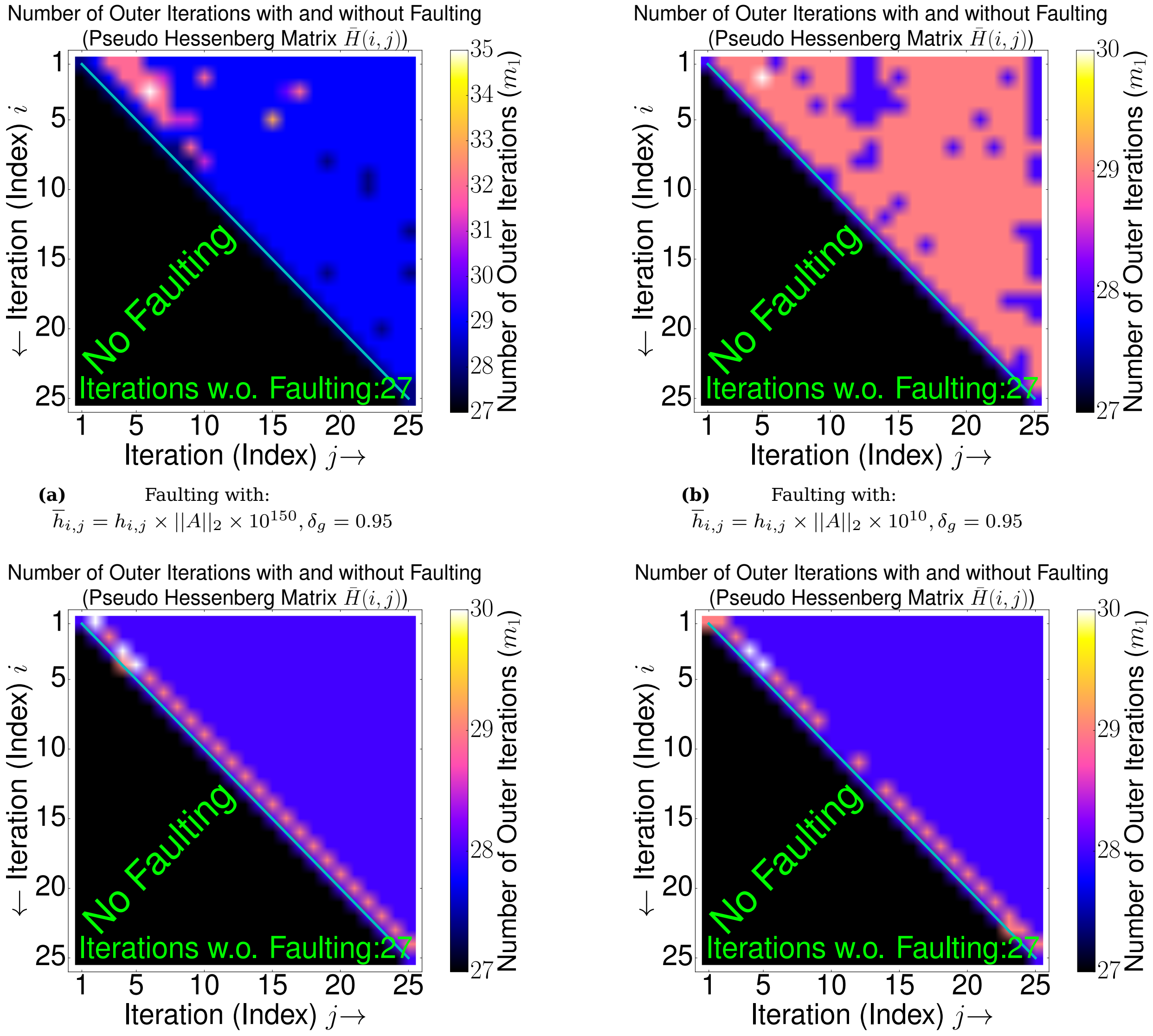
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}, \delta_g = 0.9$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}, \delta_g = 0.9$

Figure 87 Number of outer iterations for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the adder_dcop_63 matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 87 during solving the adder_dcop_63 matrix. The inner solver of the FT-GMRES does at least 25 but at most 35 iterations (m_2). If a value of $\delta_g = 0.90$ ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. It is assumed that during these additional iterations (between 25 to 35) there is no fault in the inner solver of the FT-GMRES. The worst overhead is 40 % (46/33) which is a decline from 15 % in the presence of a single fault. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all trials.

9.6.11 Faulting during solving the adder_dcop_63 problem with flexible preconditioning ($\delta_g = 0.95$) by the inner solver - Inner solver initialized with zero values



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 0.95$

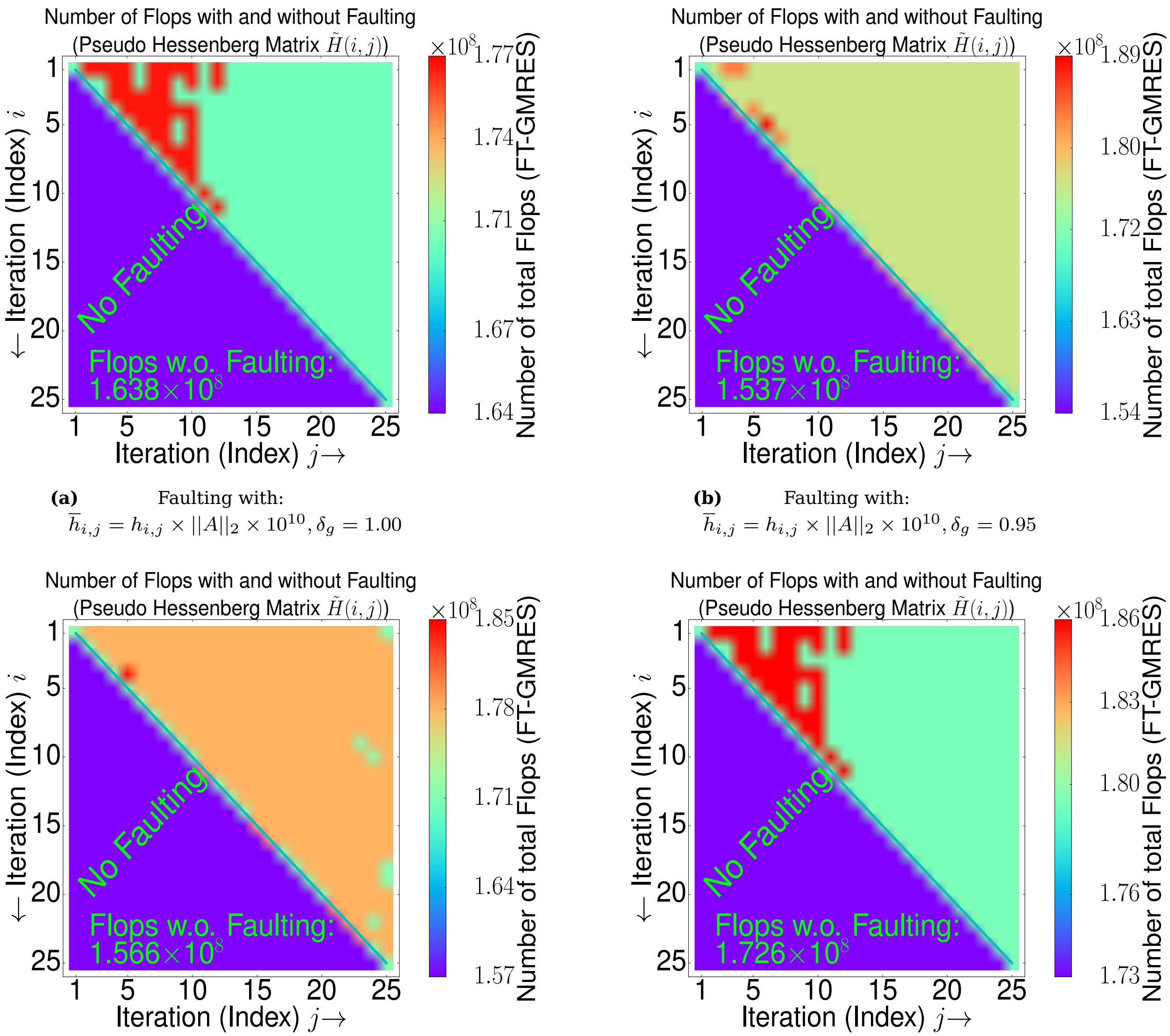
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}, \delta_g = 0.95$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}, \delta_g = 0.95$

Figure 88 Number of outer iterations for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the adder_dcop_63 matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 88 during solving the adder_dcop_63 matrix. The inner solver of the FT-GMRES does at least 25 but at most 35 iterations (m_2). If a value of $\delta_g = 0.95$ ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. It is assumed that during these additional iterations (between 25 to 35) there is no fault in the inner solver of the FT-GMRES. The worst overhead is 25 % (35/28) which is a decline from 15 % in the presence of a single fault. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all trials.

9.6.12 Number of flops for solving the adder_dcop_63 problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 1.00$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 0.95$

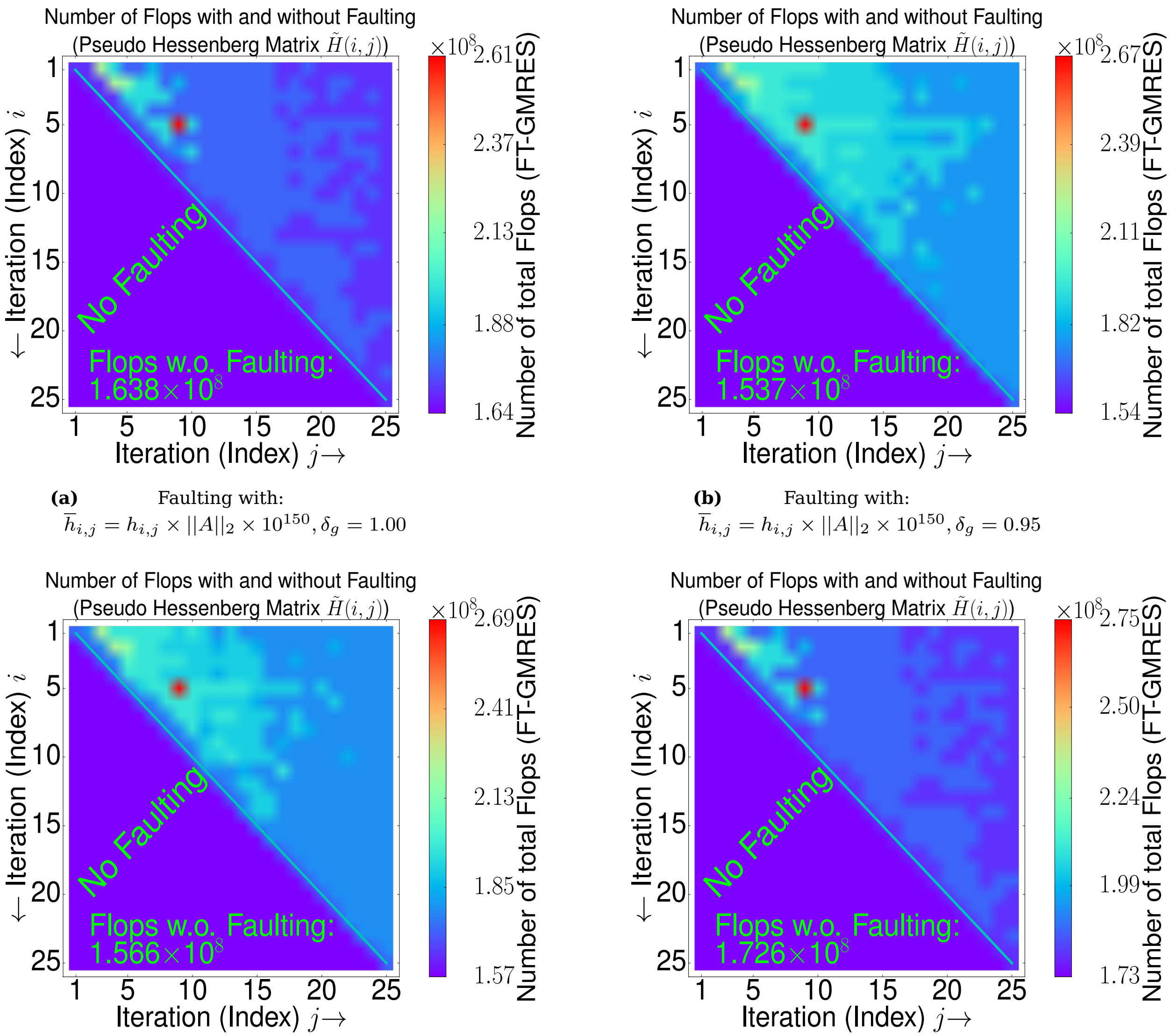
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 0.9$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 0.5$

Figure 89 Number of flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the adder_dcop_63 matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 89 during solving the adder_dcop_63 matrix. The inner solver of the FT-GMRES does at least 5 but at most 35 iterations (m_2). In figure 89 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead ($\approx$ 5 - 15 %) does nearly remain the same and the total number of flops is different for each different value of δ_g without faulting. In this case flexible preconditioning mainly helps to decrease the needed number of flops without faulting. The accuracy ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all computations.

9.6.13 Number of flops for solving the adder_dcop_63 problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 1.00$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

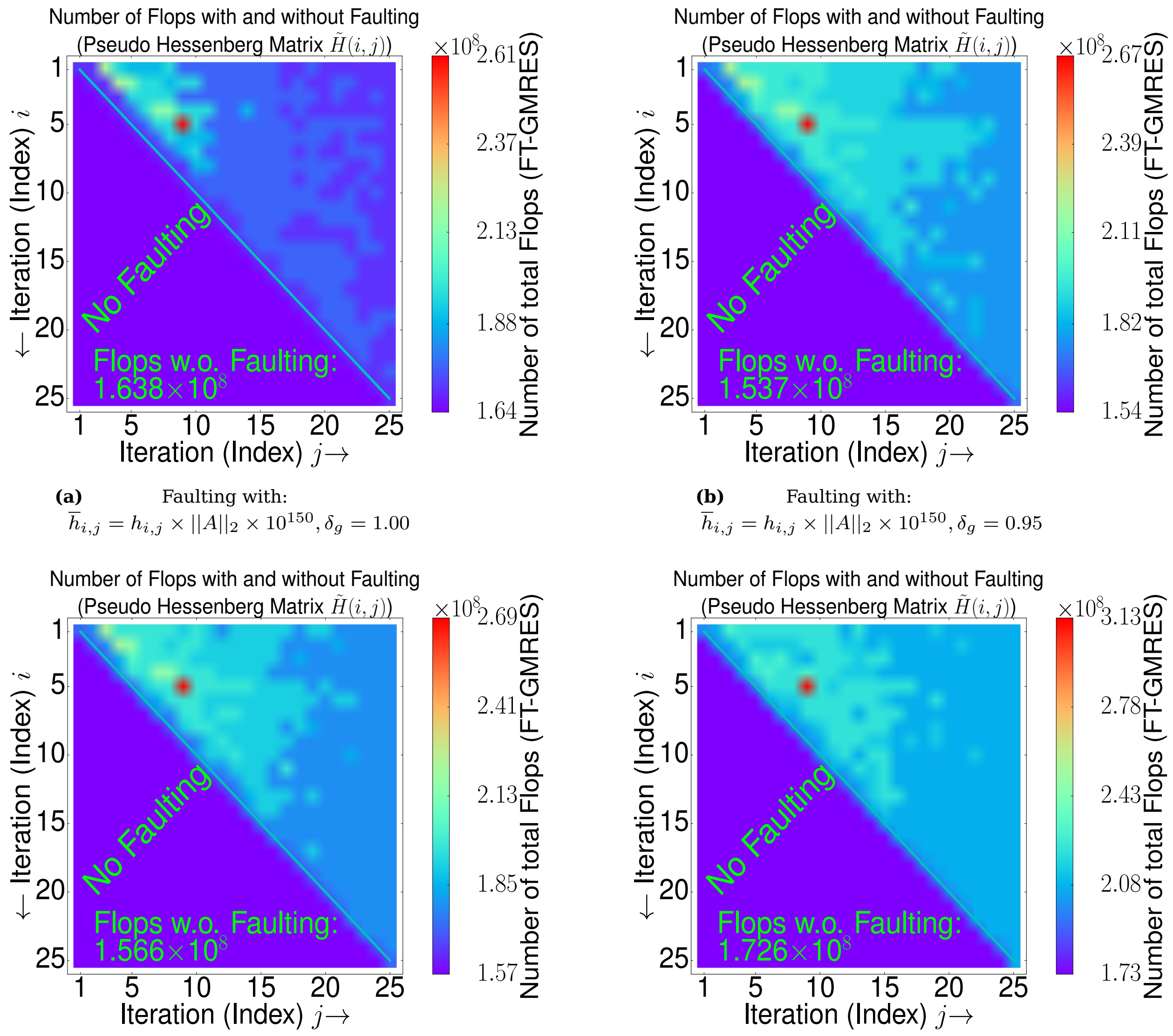
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.5$

Figure 90 Number of flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the adder_dcop_63 matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 90 during solving the adder_dcop_63 matrix. The inner solver of the FT-GMRES does at least 5 but at most 35 iterations (m_2). In figure 90 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead (≈ 15 - 70 %) does not remain the same and the total number of flops is different for each different value of δ_g without faulting. In this case flexible preconditioning mainly helps to decrease the needed number of flops without faulting. The accuracy ($||r||_2 = ||Ax - b||_2 / ||b||_2$) is below 10^{-9} for all computations.

9.6.14 Number of flops for solving the adder_dcop_63 problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with:
$\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 1.00$

(b) Faulting with:
$\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

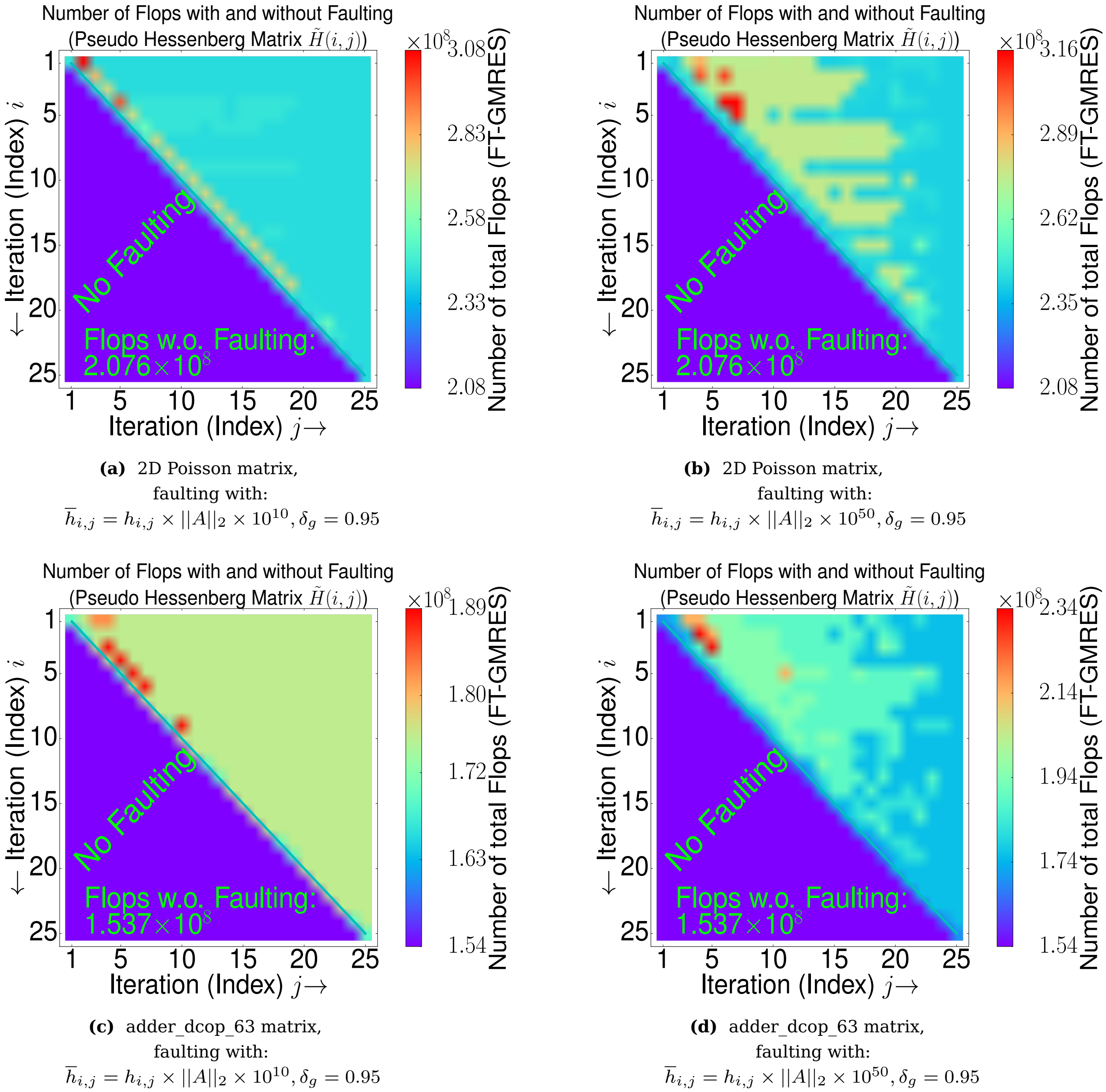
(c) Faulting with:
$\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

(d) Faulting with:
$\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.5$

Figure 91 Number of flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the adder_dcop_63 matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 91 during solving the adder_dcop_63 matrix. The inner solver of the FT-GMRES does at least 1 but at most 35 iterations (m_2). In figure 91 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead (≈ 15 - 70 %) does not remain the same and the total number of flops is different for each different value of δ_g without faulting. In this case flexible preconditioning mainly helps to decrease the needed number of flops without faulting. The accuracy ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all computations.

9.6.15 Further experiments for solving the 2D Poisson and adder_dcop_63 problem with flexible preconditioning and faulting - Inner solver initialized with zero values



(a) 2D Poisson matrix, faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 0.95$

(b) 2D Poisson matrix, faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}, \delta_g = 0.95$

(c) adder_dcop_63 matrix, faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}, \delta_g = 0.95$

(d) adder_dcop_63 matrix, faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{50}, \delta_g = 0.95$

Figure 92 Further experiments for solving the 2D Poisson and adder_dcop_63 problem with applying flexible preconditioning by the inner solver of the FT-GMRES in the presence of a fault.

As in section 9.6.9 and 9.6.14 the 2D Poisson and adder_dcop_63 problem is solved in figure 92 with applying flexible preconditioning by the inner solver of the FT-GMRES. The inner solver of the FT-GMRES does at least 1 but at most 35 iterations (m_2). If a specific value of $\delta_g = 0.95$ is achieved during preconditioning the inner solver is stopped and the outer solver is allowed to proceed. Figure 92 mainly shows that larger faults in the inner solver of the FT-GMRES lead to more operations (from figure 92a to 92b and from figure 92c to 92d) to negate the effect of poor preconditioning. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all computations.

9.6.16 Some notes on the flexible preconditioning with/without faulting

The main outcome is that flexible preconditioning helps to decrease the needed number of operations without a fault but it is not ensured that in the case of fault also a low number of operations is achieved. It is possible to reduce the total number of flops massively for the F-GMRES solver like in section 9.6.3 shown for the 2D Poisson and adder_dcop_63 matrix with applying flexible preconditioning for the inner solver of the F-GMRES. In most cases seen so far the F-GMRES needs less operations with observing the accuracy of the solution vector w_j where unnecessary iterations (m_2) and work for the inner solver of the F-GMRES are avoided in contrast to the standard case where only (right) preconditioning is applied with a fixed number of inner iterations (m_2). The problem is just to find optimal parameters such that the F-GMRES solver needs the lowest possible number of operations (flops) to converge to a specific residuum or to the same residuum as in the case where only (right) preconditioning is applied.

Overall observing the accuracy of the solution vector w_j in the inner solver of the F-GMRES (line 4 of algorithm 11) seems to be a good approach to avoid a large number of operations for the inner solver (m_2) where unnecessary work is done. The optimal parameters which are searched for flexible preconditioning are the smallest and greatest number of inner iterations (m_2) but also the value of δ_g which measures the linear in-dependency of two residual vectors ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) or the accuracy ϵ_2 for the solution vector w_j of the inner solver ($||(q_{j+1} - Aw_{j+1})||_2/||q_{j+1}||_2 \leq \epsilon_2$). It is far away of being obvious to find these optimal parameters such that the lowest possible number of operations is achieved but like said the inner solver of the F(T)-GMRES should do the most work.

The FT-GMRES can cope with different kinds of faults if flexible preconditioning is applied but the real overhead can be hidden in number of operations (flops) by the inner solver because in some cases the outer solver achieves the same number of outer iterations (m_1) with and without a fault whereas more work and more operations are done by the inner solver in the presence of a fault. The result is just that in the case of flexible preconditioning with a fault in the inner solver a similar overhead in the number of flops occurs as in the case without flexible preconditioning where the overhead appears in number of outer iterations (m_1). This is mainly shown in section 9.6.9 and 9.6.14 where nearly no overhead for the number of outer iterations is achieved but the number of total flops for FT-GMRES solver can be really high because of additional operations of the inner solver in the presence of a single fault.

Furthermore the cost (flops) of a fault in the inner solver of the FT-GMRES can be high which means that the inner solver of the FT-GMRES will always try to negate the effect of a fault but a fault will always lead to more operations (more outer or/and inner iterations). In the case of solving the 2D Poisson problem with flexible preconditioning and in the presence of a single fault the worst overhead ($\approx$ 10 - 80 %) can be sometimes really high but the average overhead is about 25 % like in the case of solving the adder_dcop_63 problem where for this problem the average overhead is around 20 %. Hence without flexible preconditioning (right preconditioning) a similar overhead of about 15 % can be observed in the case of solving the adder_dcop_63 problem whereas for solving the 2D Poisson problem a worst overhead of 36 % is observed.

The contrast is mainly owed how the overhead is estimated (flops vs number of outer iterations) and because of flexible preconditioning. There are also some outliers which cause this huge overhead mainly for solving these two problems. The outcome is this approach with flexible preconditioning mainly helps to decrease the needed number of operations without a fault whereas in the case of a single fault in the inner solver of the FT-GMRES a similar overhead as in the case without flexible preconditioning (right preconditioning) can occur or even higher. More aggressive preconditioning (lower values of δ_g) leads to more overhead, for more examples see section 9.10.

9.7 Results of some experiments (worst overhead) with the corresponding matrix properties

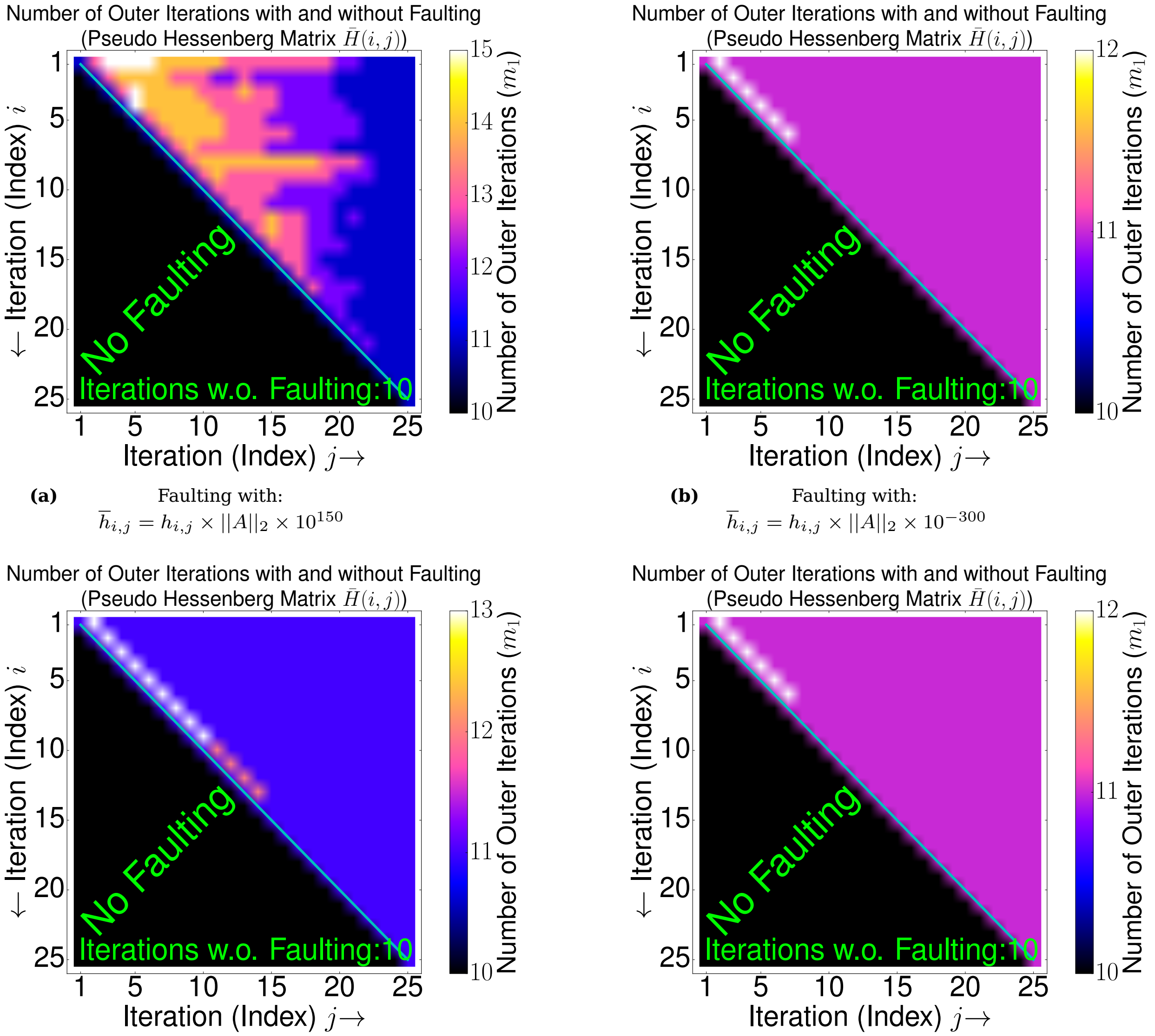
This section mainly summarizes the results of some experiments for different kinds of matrices [10] in table 6 which shows the worst overhead for different kinds of faults (10^{150} and 10^{-300}). The first part of this table is about the worst overhead for solving the 2D Poisson and adder_dcop_63 matrix whereas the second part shows the worst overhead in the presence of a single fault for other matrices which are discussed in this work but as well not. The methodology for all experiments is the same as in section 12. It also sums up some important properties like the condition number and the number of outer iterations (m_1) without a fault to solve the problem of $Ax = b$ and to converge to the same relative residuum as in the presence of a single fault. These huge overheads can occur because of stagnating convergence, after a specific accuracy the solver gets significant slower.

Name	$\|\|A\|\|_2$	Residuum ($\|\|r\|\|_2$)	Iterations	$\kappa(A)_2$	Overhead in %
2D Poisson	7.9014	10^{-9}	10	6.02×10^3	36%
adder_dcop_63	1.1743	10^{-9}	32	5.6107×10^3	15%
Pres_Poisson	26.0289	10^{-6}	30	2.03665×10^6	6.5%
Kuu	54.0821	10^{-9}	25	15758	15%
Na5	25.6604	10^{-9}	20	1212.23	19%
circuit_2	26.387	10^{-9}	28	131925	14%
mult_dcop_03	17.1762	10^{-9}	45	7.27261×10^{13}	9%
chem_master1	4.3071×10^3	$10^{-5.5}$	28	4.42×10^3	21%
ILL_Stokes	5.44287	10^{-6}	23	2.25289×10^9	12.5%
bcircuit	4.4662×10^4	$10^{-4.5}$	21	2.2087×10^9	18%
cavity06	15.7727	$10^{-4.5}$	30	672940	16%
cavity17	13.0849	10^{-6}	37	9.35083×10^6	16%
Chebyshev2	20277.2	10^{-5}	89	5.54075×10^{15}	166%
ebp1	0.208272	$10^{-10.5}$	30	5940.66	3%
fpga_dcop_03	12.284	10^{-5}	41	1.84648×10^{13}	33%
garon2	44.8683	$10^{-5.5}$	20	6.12558×10^7	9.5%
init_adder1	1.01806	10^{-9}	41	3.2654×10^7	72.5%
juba40k	1.0×10^{12}	$10^{-5.5}$	20	Inf	43%
k3plates	1.99511×10^{10}	10^{-6}	44	1.50352×10^{18}	4%
light_in_tissue	2.66355	10^{-7}	12	7821.9	77%
msc01050	2.10×10^7	10^{-6}	32	4.5826×10^{15}	58%
nasa2910	77.6164	10^{-4}	8	9.5271×10^{64}	11%
nos2	1.57283×10^{11}	10^{-6}	34	5.09958×10^9	11%
plat362	0.774279	10^{-6}	21	2.17822×10^{11}	36%
pores_2	6.08895×10^7	10^{-4}	66	1.09429×10^8	150%
qh1484	1.25679×10^{16}	10^{-4}	89	1.04449×10^{18}	116%
raefsky2	3.72481	10^{-9}	26	4251.948	7%
raefsky6	3.97745×10^{14}	10^{-9}	51	1.41323×10^{16}	7%
rajat19	10.9106	10^{-5}	49	1.09106×10^{10}	300%
shyy161	62.6441	$10^{-4.5}$	10	8.23×10^{266}	54%
SiNa	25.6159	10^{-9}	17	507.117	22%
wang3	0.268008	10^{-9}	12	6188.12	23%
sherman2	2.43842×10^9	$10^{-4.5}$	272	9.64333×10^{11}	8%
xenon1	5.2875×10^{28}	10^{-9}	36	1.6747×10^5	6%

Table 6 Worst overhead for a single fault and various disturbances for solving different matrices. For all experiments the inner solver of the FT-GMRES is initialized with zeros for vector $w_j(w_0)$.

9.8 Faulting with single faults while solving further problems (symmetric, unsymmetric)

9.8.1 Faulting during solving the 2D Poisson problem for different kinds of faults (symmetric problem) - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

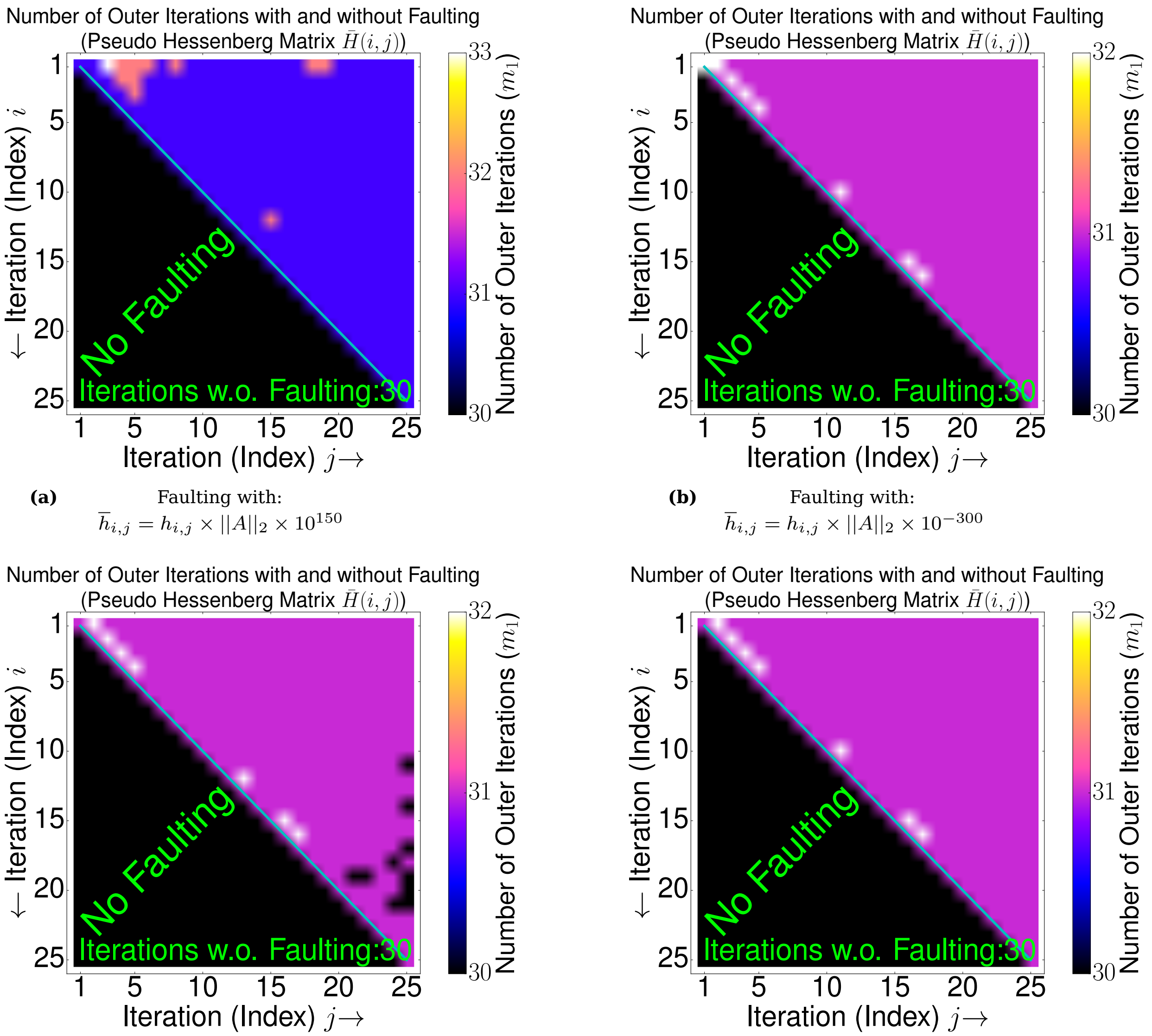
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 93 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the 2D Poisson matrix (, compare with figure 158).

In figure 93 the 2D Poisson matrix is solved and as well faulting during orthogonalization ($h_{i,j}$) of the inner solver is shown. The F(T)-GMRES takes 10 outer iterations (m_1) without a fault and for the first trial whereas in the worst case 15 outer iterations (m_1) are needed in the presence of a single fault such that the according overhead is about 36 % (15/11). For each trial and computation the relative residuum ($||r||_2 = ||Ax - b||_2)/||b||_2$) is below 10^{-9}, a symmetric problem is solved.

9.8.2 Faulting during solving the Pres_Poisson problem for different kinds of faults (symmetric problem) - Inner solver initialized with zero values



(a) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

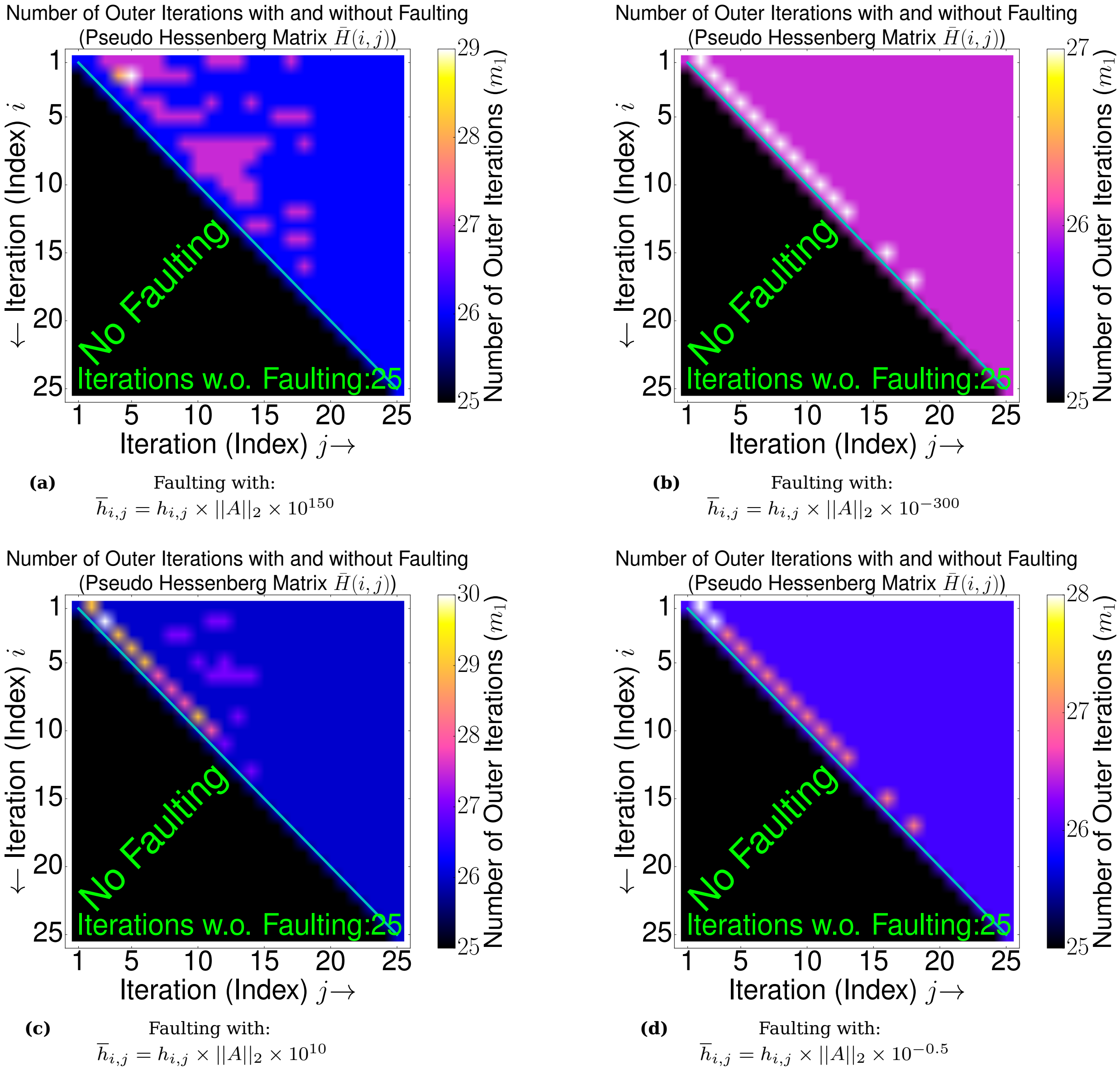
(c) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 94 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the Pres_Poisson matrix (, compare with figure 159).

In figure 94 the Pres_Poisson matrix is solved and as well faulting during orthogonalization ($h_{i,j}$) of the inner solver is shown. The F(T)-GMRES takes 30 outer iterations (m_1) without a fault and for the first trial whereas in the worst case 33 outer iterations (m_1) are needed in the presence of a single fault such that the according overhead is about 6.5 % (33/31). For each trial and computation the relative residuum ($||r||_2 = ||Ax-b||_2)/||b||_2$) is below 10^{-6}. In the case of the Pres_Poisson matrix the F(T)-GMRES solves a symmetric problem.

9.8.3 Faulting during solving the Kuu problem for different kinds of faults (symmetric problem) - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

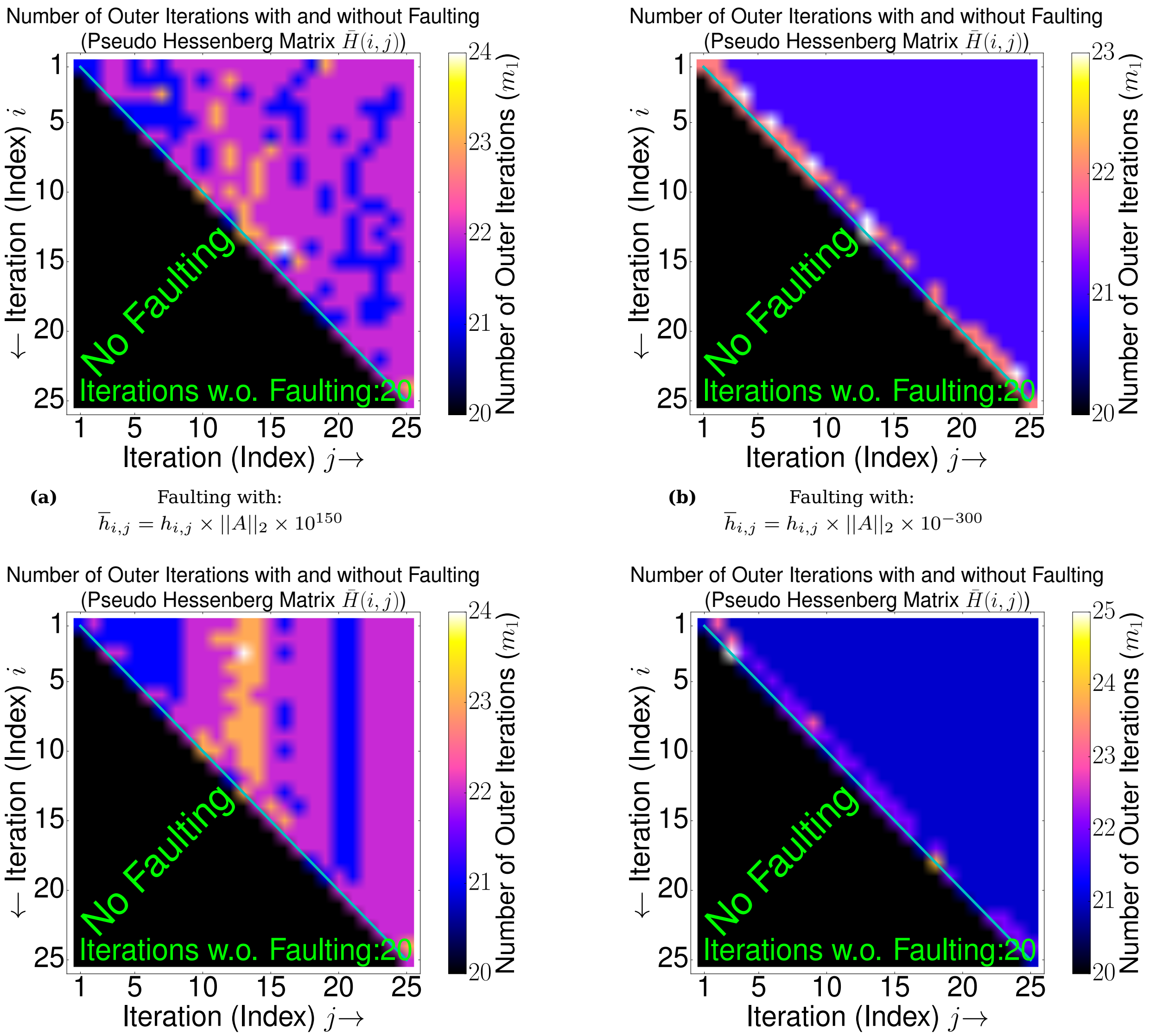
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 95 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the Kuu matrix (, compare with figure 160).

In figure 95 the Kuu matrix is solved and as well faulting during orthogonalization ($h_{i,j}$) of the inner solver is shown. The F(T)-GMRES takes 25 outer iterations (m_1) without a fault and for the first trial whereas in the worst case 30 outer iterations (m_1) are needed in the presence of a single fault such that the according overhead is about 15 % (30/26). For each trial and computation the relative residuum ($||r||_2 = ||Ax - b||_2)/||b||_2$) is below 10^{-9}. In the case of the Kuu matrix the F(T)-GMRES solves a symmetric problem.

9.8.4 Faulting during solving the Na5 problem for different kinds of faults (symmetric problem) - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

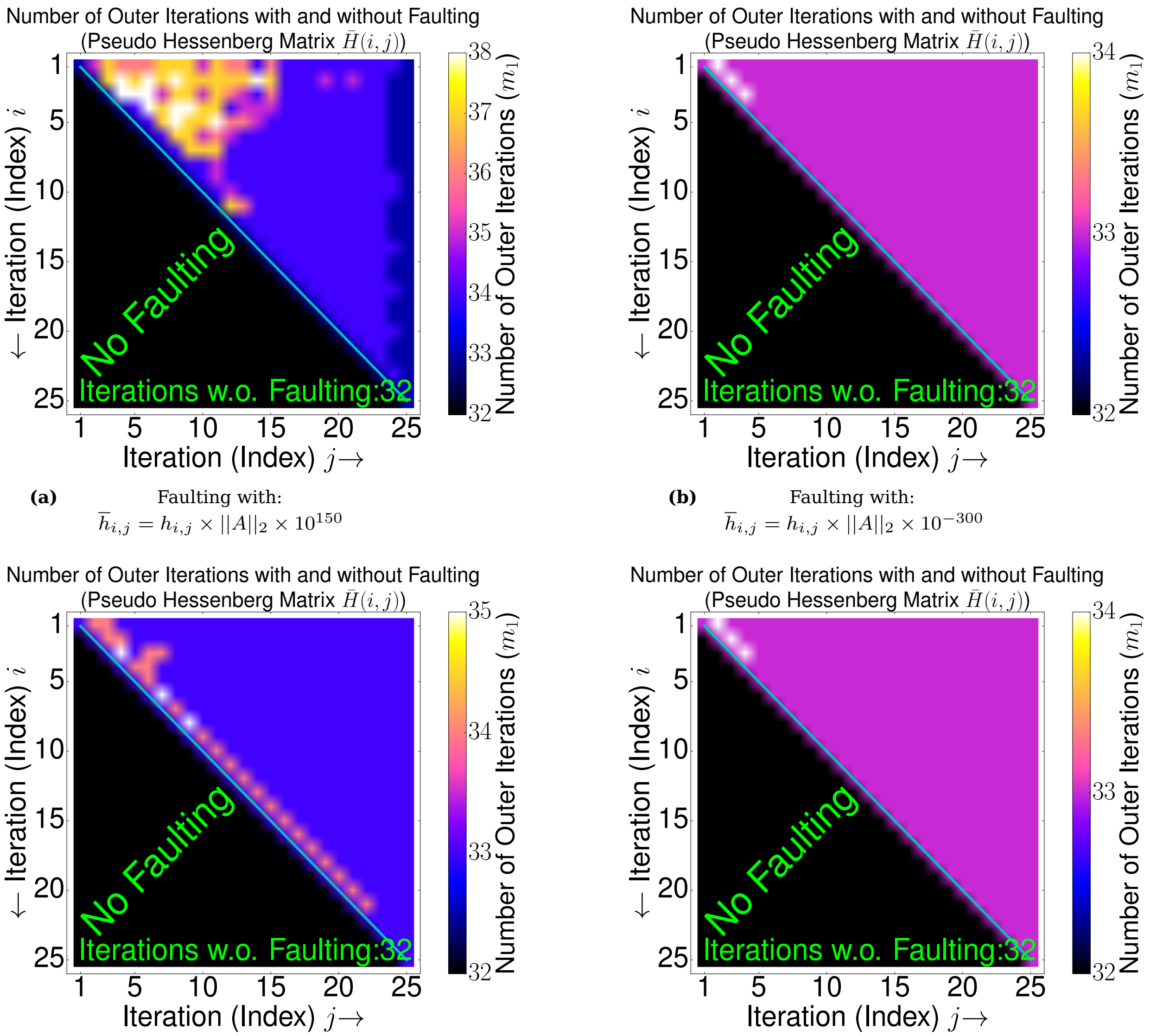
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 96 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the Na5 matrix (, compare with figure 161).

In figure 96 the Na5 matrix is solved and as well faulting during orthogonalization ($h_{i,j}$) of the inner solver is shown. The F(T)-GMRES takes 20 outer iterations (m_1) without a fault and for the first trial whereas in the worst case 25 outer iterations (m_1) are needed in the presence of a single fault such that the according overhead is about 19 % (25/21). For each trial and computation the relative residuum ($||r||_2 = ||Ax - b||_2)/||b||_2$) is below 10^{-9}. In the case of the Na5 matrix the F(T)-GMRES solves a symmetric problem.

9.8.5 Faulting during solving the adder_dcop_63 problem for different kinds of faults (un-symmetric problem) - Inner solver initialized with zero values



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

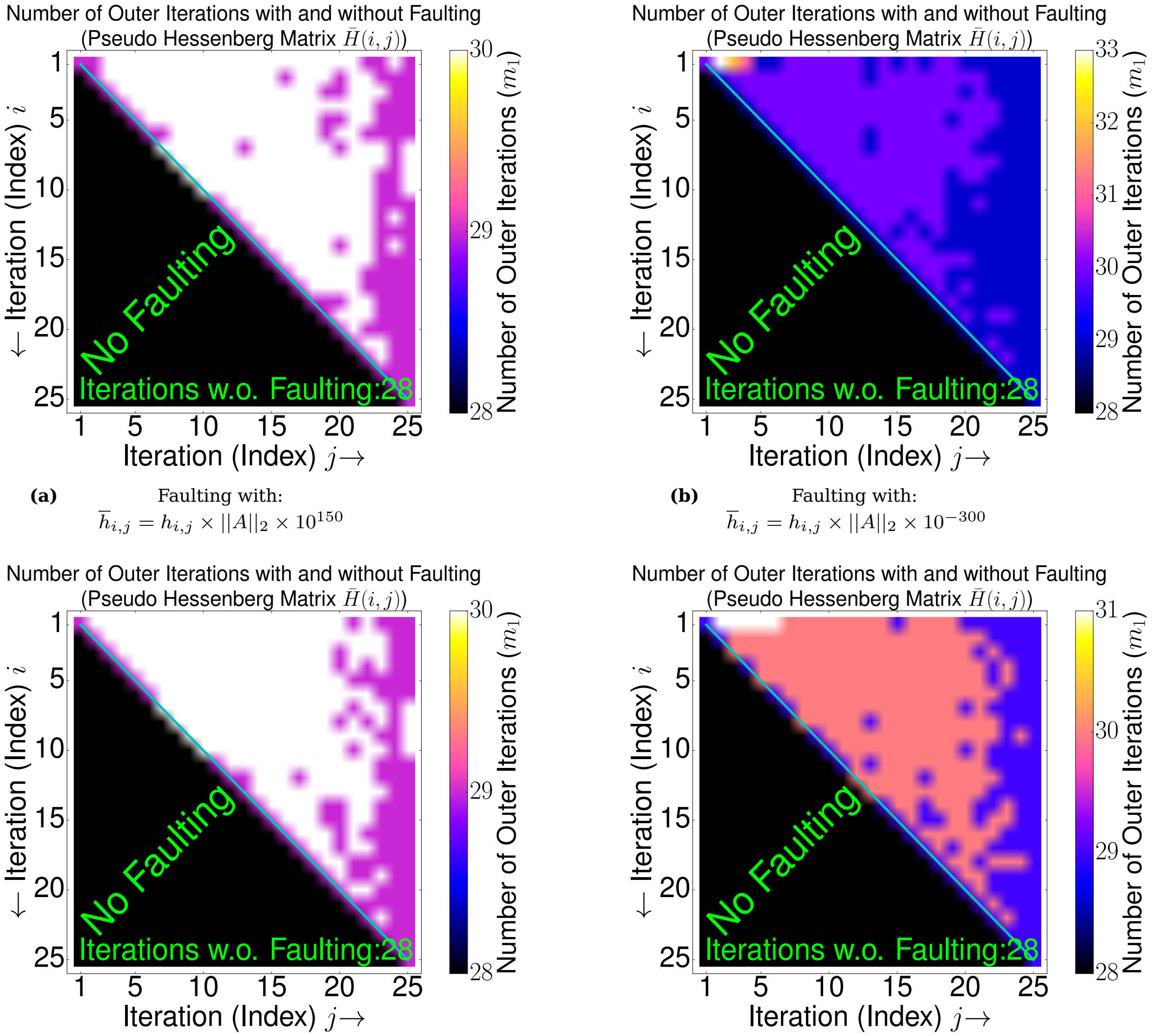
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 97 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the adder_dcop_63 matrix (, compare with figure 162).

In figure 97 the adder_dcop_63 matrix is solved and as well faulting during orthogonalization ($h_{i,j}$) of the inner solver is shown. The F(T)-GMRES takes 32 outer iterations (m_1) without a fault and for the first trial whereas in the worst case 38 outer iterations (m_1) are needed in the presence of a single fault such that the according overhead is about 15 % (38/33). For each trial and computation the relative residuum $(||r||_2 = ||Ax - b||_2)/||b||_2)$ is below 10^{-9}. In the case of the adder_dcop_63 matrix the F(T)-GMRES solves a un-symmetric problem.

9.8.6 Faulting during solving the circuit_2 problem for different kinds of faults (unsymmetric problem) - Inner solver initialized with zero values



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

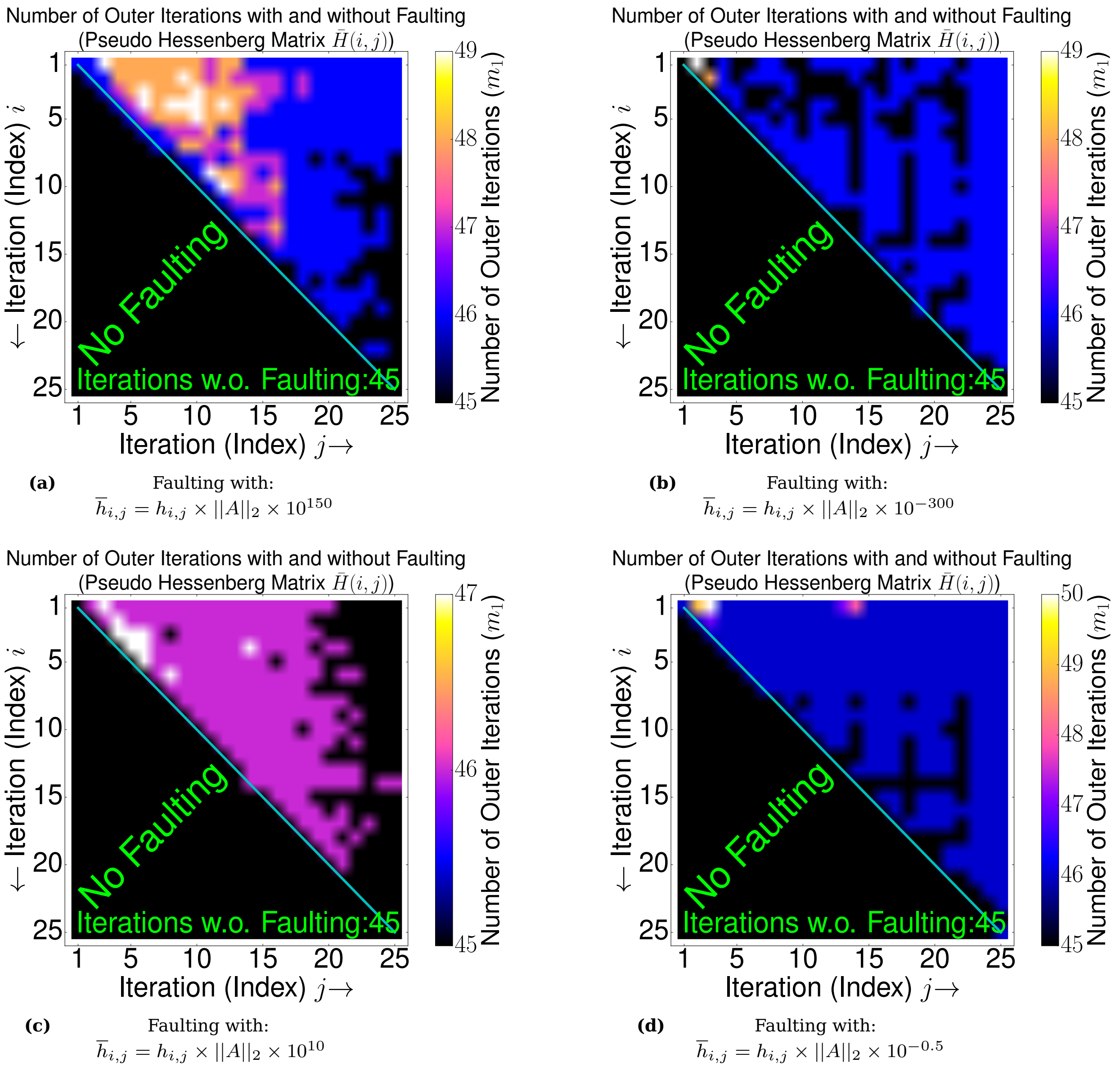
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 98 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the circuit_2 matrix (, compare with figure 163).

In figure 98 the circuit_2 matrix is solved and as well faulting during orthogonalization ($h_{i,j}$) of the inner solver is shown. The F(T)-GMRES takes 28 outer iterations without a fault and for the first trial whereas in the worst case 33 outer iterations (m_1) are needed in the presence of a single fault such that the according overhead is about 14 % (33/29). For each trial and computation the relative residuum ($||r||_2 = ||Ax - b||_2)/||b||_2$) is below 10^{-9}. In the case of the circuit_2 matrix the F(T)-GMRES solves an un-symmetric problem.

9.8.7 Faulting during solving the mult_dcop_03 problem for different kinds of faults (unsymmetric problem) - Inner solver initialized with zero values



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

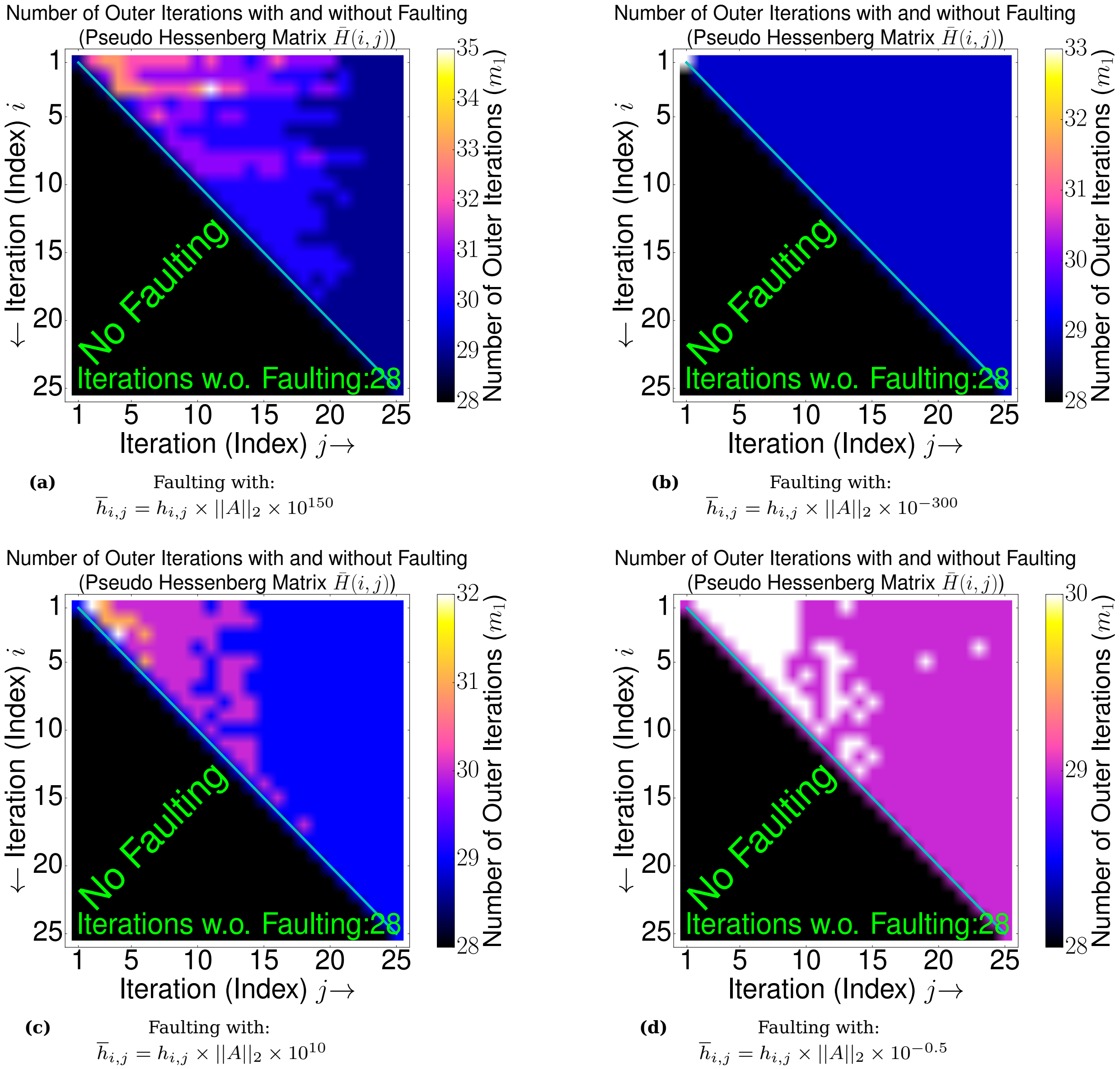
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 99 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the mult_dcop_03 matrix (, compare with figure 164).

In figure 99 the mult_dcop_03 matrix is solved and as well faulting during orthogonalization ($h_{i,j}$) of the inner solver is shown. The F(T)-GMRES takes 45 outer iterations without a fault and for the first trial whereas in the worst case 50 outer iterations (m_1) are needed in the presence of a single fault such that the according overhead is about 9 % (50/46). For each trial and computation the relative residuum ($||r||_2 = ||Ax - b||_2)/||b||_2$) is below 10^{-9}. In the case of the mult_dcop_03 matrix the F(T)-GMRES solves an un-symmetric problem.

9.8.8 Faulting during solving the chem_master1 problem for different kinds of faults (un-symmetric problem) - Inner solver initialized with zero values.



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

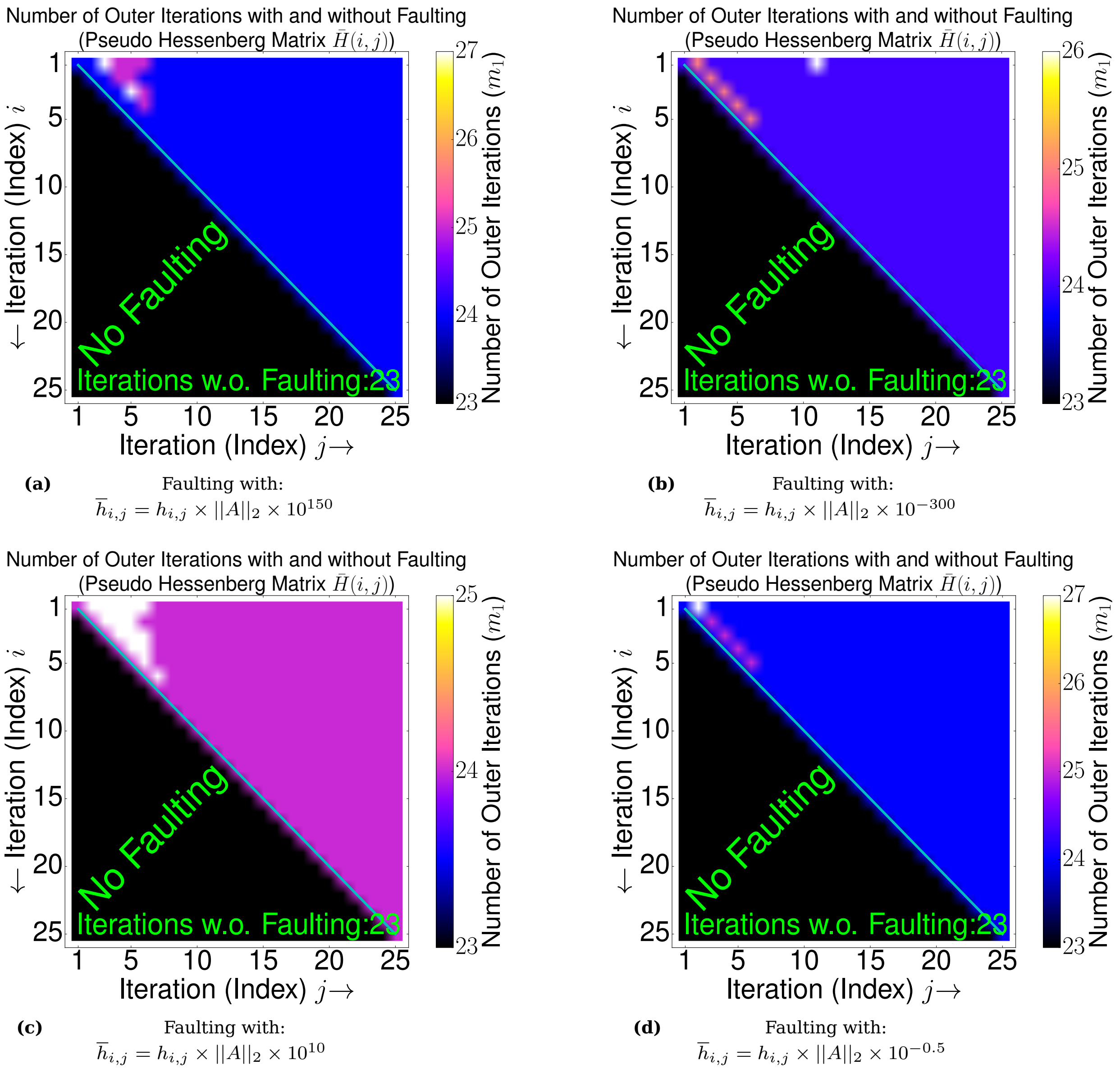
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 100 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the chem_master1 matrix (, compare with figure 165).

In figure 100 the chem_master1 matrix is solved and as well faulting during orthogonalization ($h_{i,j}$) of the inner solver is shown. The F(T)-GMRES takes 28 outer iterations (m_1) without a fault and for the first trial whereas in the worst case 35 outer iterations (m_1) are needed in the presence of a single fault such that the according overhead is about 21 % (35/29). For each trial and computation the relative residuum $(||r||_2 = ||Ax - b||_2)/||b||_2)$ is below $10^{-5.5}$. In the case of the chem_master1 matrix the F(T)-GMRES solves an un-symmetric problem.

9.8.9 Faulting during solving the ILL_Stokes problem for different kinds of faults (un-symmetric problem) - Inner solver initialized with zero values.



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

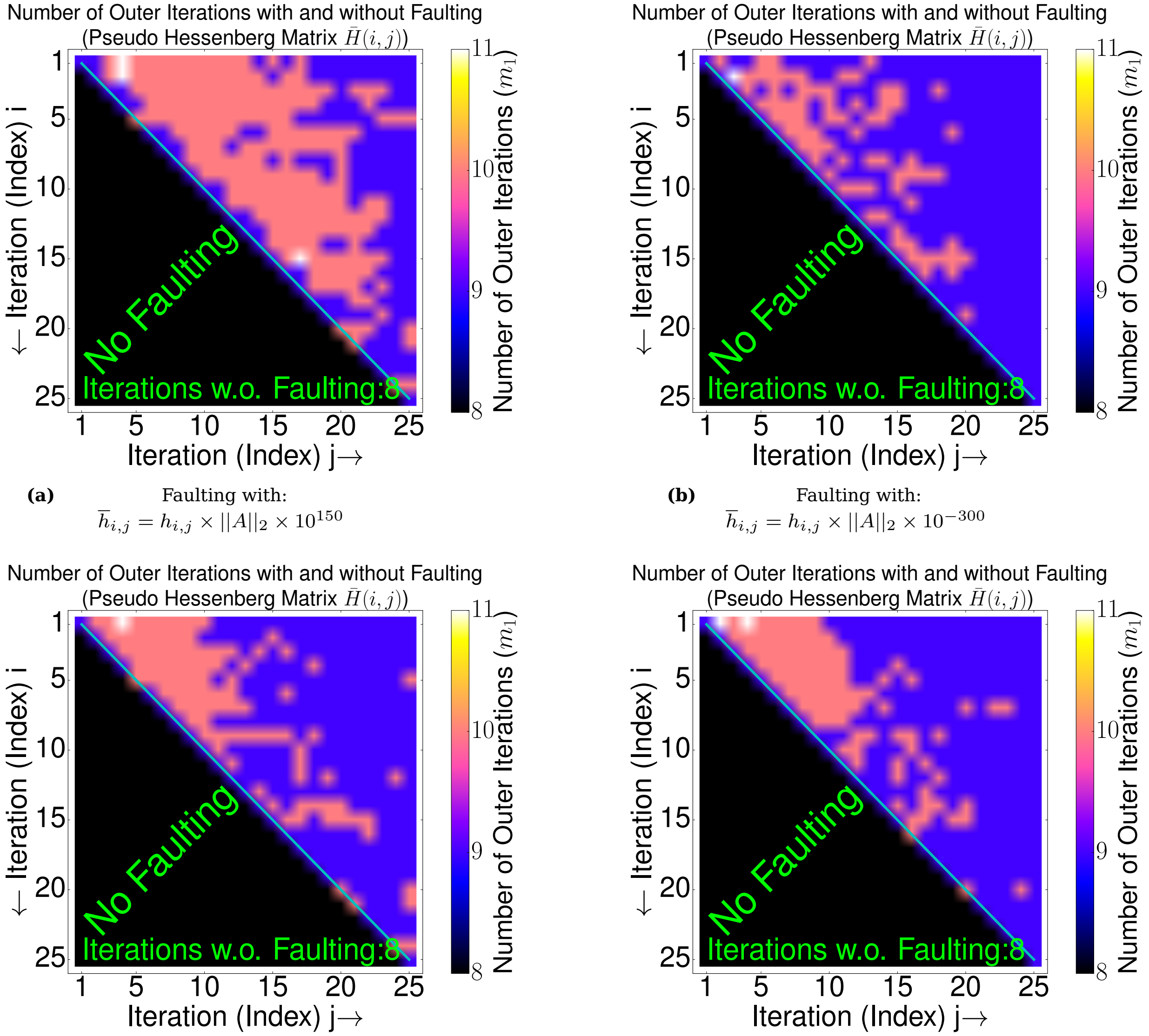
(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 101 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the ILL_Stokes matrix (, compare with figure 166).

In figure 101 the ILL_Stokes matrix is solved and as well faulting during orthogonalization ($h_{i,j}$) of the inner solver is shown. The FT-GMRES takes 23 outer iterations (m_1) without a fault and for the first trial whereas in the worst case 27 outer iterations (m_1) are needed in the presence of a single fault such that the according overhead is about 12.5 % (27/24). For each trial and computation the relative residuum ($||r||_2 = ||Ax - b||_2)/||b||_2$) is below 10^{-6}. In the case of the ILL_Stokes matrix the F(T)-GMRES solves an un-symmetric problem.

9.9 Faulting with single faults while solving further preconditioned problems with ILU-factorization (symmetric, un-symmetric)

9.9.1 Faulting during solving the Pres_Poisson problem for different kinds of faults (symmetric problem) - Inner solver initialized with zero values



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

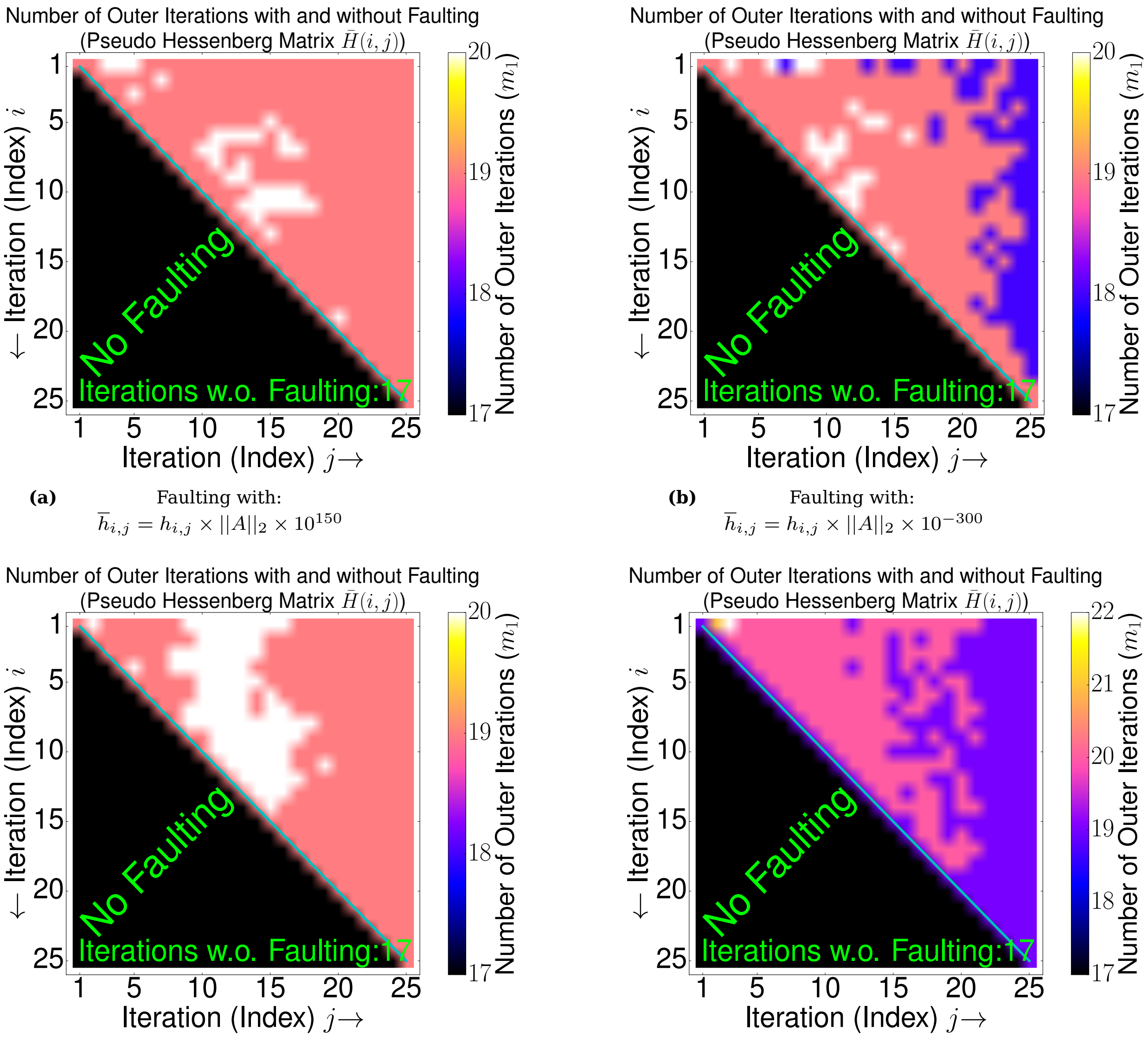
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 102 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the Pres_Poisson matrix with ILU-preconditioning (and $Error_{ILU}{}^3 \approx 3 \times 10^{-2}$). The inner solver builds an upper Hessenberg matrix as in formula 104.

As in figure 94 also the Pres_Poisson matrix is solved in figure 102 but with ILU-preconditioning. The F(T)-GMRES takes 8 outer iterations (m_1) without a fault whereas 25 iterations are set for the inner solver (m_2), there are 11 outer iterations needed in the presence of a single fault. For each trial the relative residuum is below 10^{-6}, the inner solver is initialized with zero values for $w_j(w_0)$. The worst overhead is 22 % (11/9) which is a decline against the un-preconditioned case (6.5 %). The value of $Error_{ILU}{}^3$ is chosen such that most of the work is still done by the inner solver.

9.9.2 Faulting during solving the Kuu problem for different kinds of faults (symmetric problem) - Inner solver initialized with zero values



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

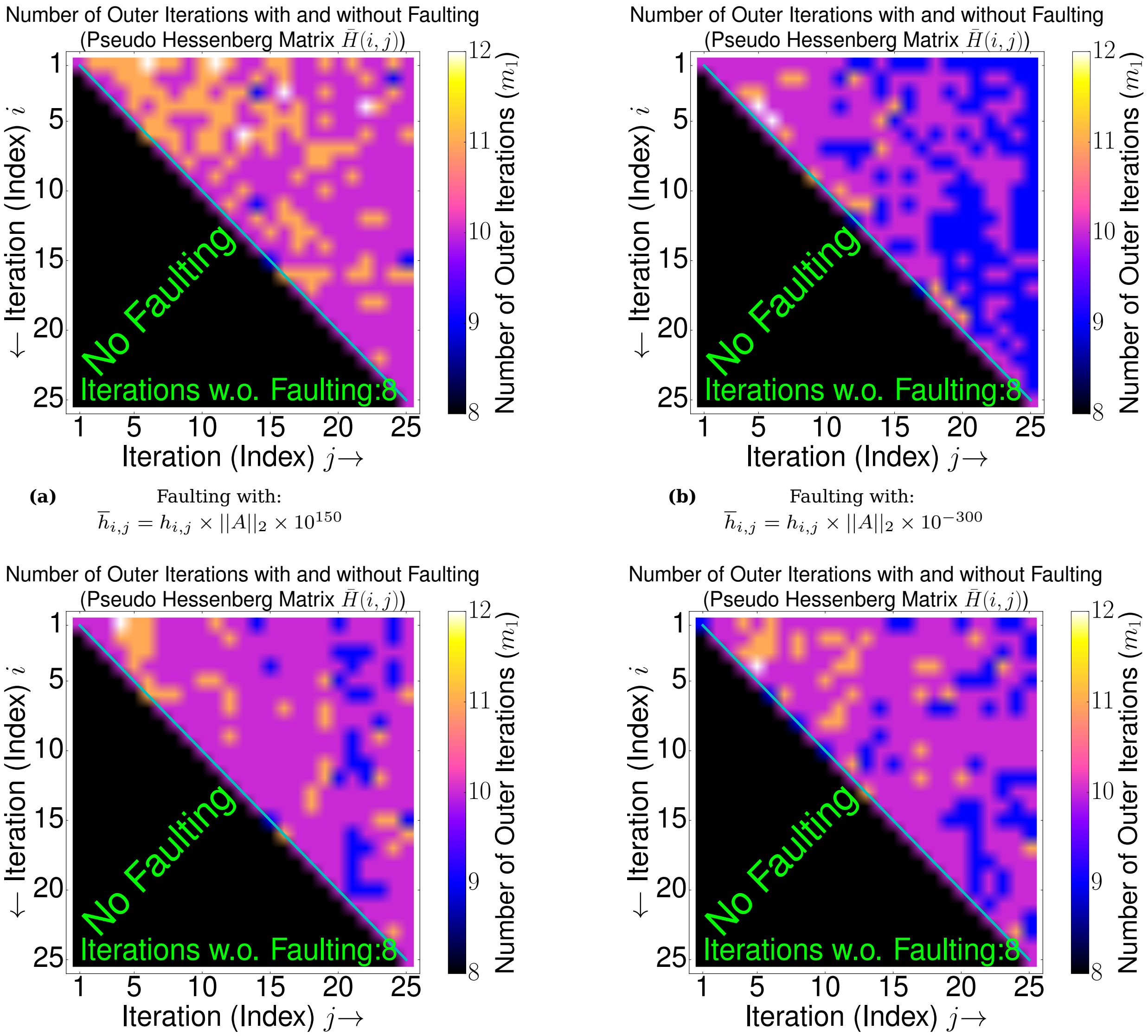
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 103 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the Kuu matrix with ILU-preconditioning (and $Error_{ILU}{}^3 \approx 3 \times 10^{-1}$). The inner solver of F(T)-GMRES builds an upper Hessenberg matrix as in formula 104.

As in figure 95 also the Kuu matrix is solved in figure 103 but with ILU-preconditioning. The F(T)-GMRES takes 17 outer iterations (m_1) without a fault and for the first trial whereas 25 iterations are set for the inner solver (m_2), there are 22 outer iterations needed in the presence of a single fault. For each trial the relative residuum is below 10^{-9}, the inner solver is initialized with zero values for vector $w_j(w_0)$. The worst overhead is 22 % (22/18) which is a decline against the un-preconditioned case (15 %). The value of $Error_{ILU}{}^3$ is chosen such that most of the work is still done by the inner solver (intuitively) because low values of $Error_{ILU}$ indicate more work for the ILU-decomposition.

9.9.3 Faulting during solving the Na5 problem for different kinds of faults (symmetric problem) - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

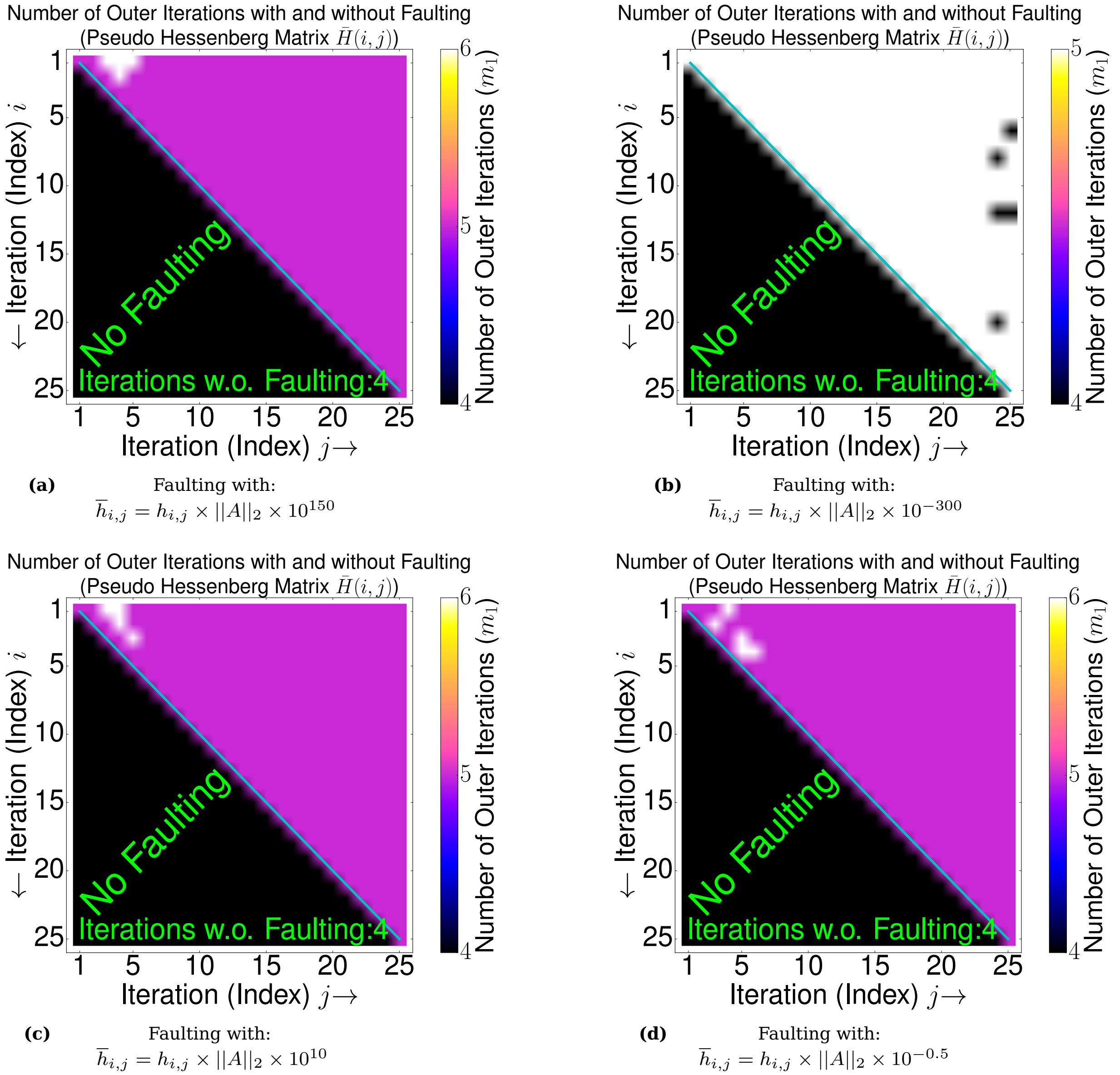
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 104 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the Na5 matrix with ILU-preconditioning (and $Error_{ILU}{}^3 \approx 5 \times 10^{-2}$). The inner solver of F(T)-GMRES builds an upper Hessenberg matrix as in formula 104.

As in figure 96 also the Na5 matrix is solved in figure 104 but with ILU-preconditioning. The F(T)-GMRES takes 8 outer iterations (m_1) without a fault and for the first trial whereas 25 iterations are set for the inner solver (m_2), there are 12 outer iterations needed in the presence of a single fault. For each trial the relative residuum is below 10^{-9}, the inner solver is initialized with zero values for vector $w_j(w_0)$. The worst overhead is 33 % (12/9) which is a decline against the unpreconditioned case (19 %). The value of $Error_{ILU}{}^3$ is chosen such that most of the work is still done by the inner solver (intuitively) because low values of $Error_{ILU}$ indicate more work for the ILU-decomposition.

9.9.4 Faulting during solving the circuit_2 problem for different kinds of faults (unsymmetric problem) - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

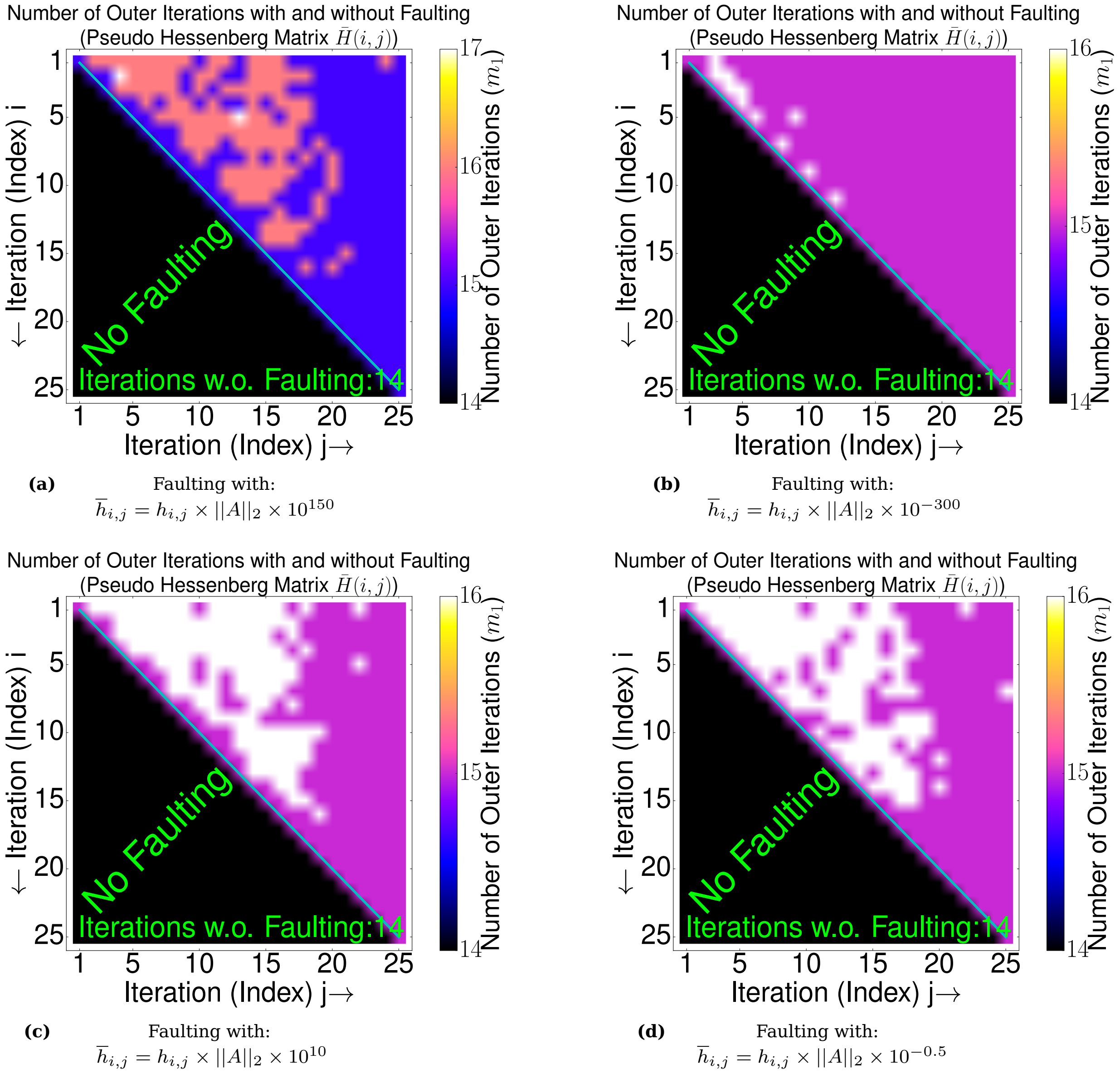
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 105 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the circuit_2 matrix with ILU-preconditioning (and $Error_{ILU}{}^3 \approx 998 \times 10^{-3}$). The inner solver of F(T)-GMRES builds an upper Hessenberg matrix as in formula 104.

As in figure 98 also the circuit_2 matrix is solved in figure 105 but with ILU-preconditioning. The F(T)-GMRES takes 4 outer iterations (m_1) without a fault and for the first trial whereas 25 iterations are set for the inner solver (m_2), there are 6 outer iterations (m_1) needed in the presence of a single fault. For each trial the relative residuum is below 10^{-9}, the inner solver is initialized with zero values for vector $w_j(w_0)$. The worst overhead is 20 % (6/5) which is a decline against the un-preconditioned case (14 %). The value of $Error_{ILU}{}^3$ is chosen such that most of the work is still done by the inner solver (intuitively) because low values of $Error_{ILU}$ indicate more work for the ILU-decomposition.

9.9.5 Faulting during solving the chem_master1 problem for different kinds of faults (un-symmetric problem) - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

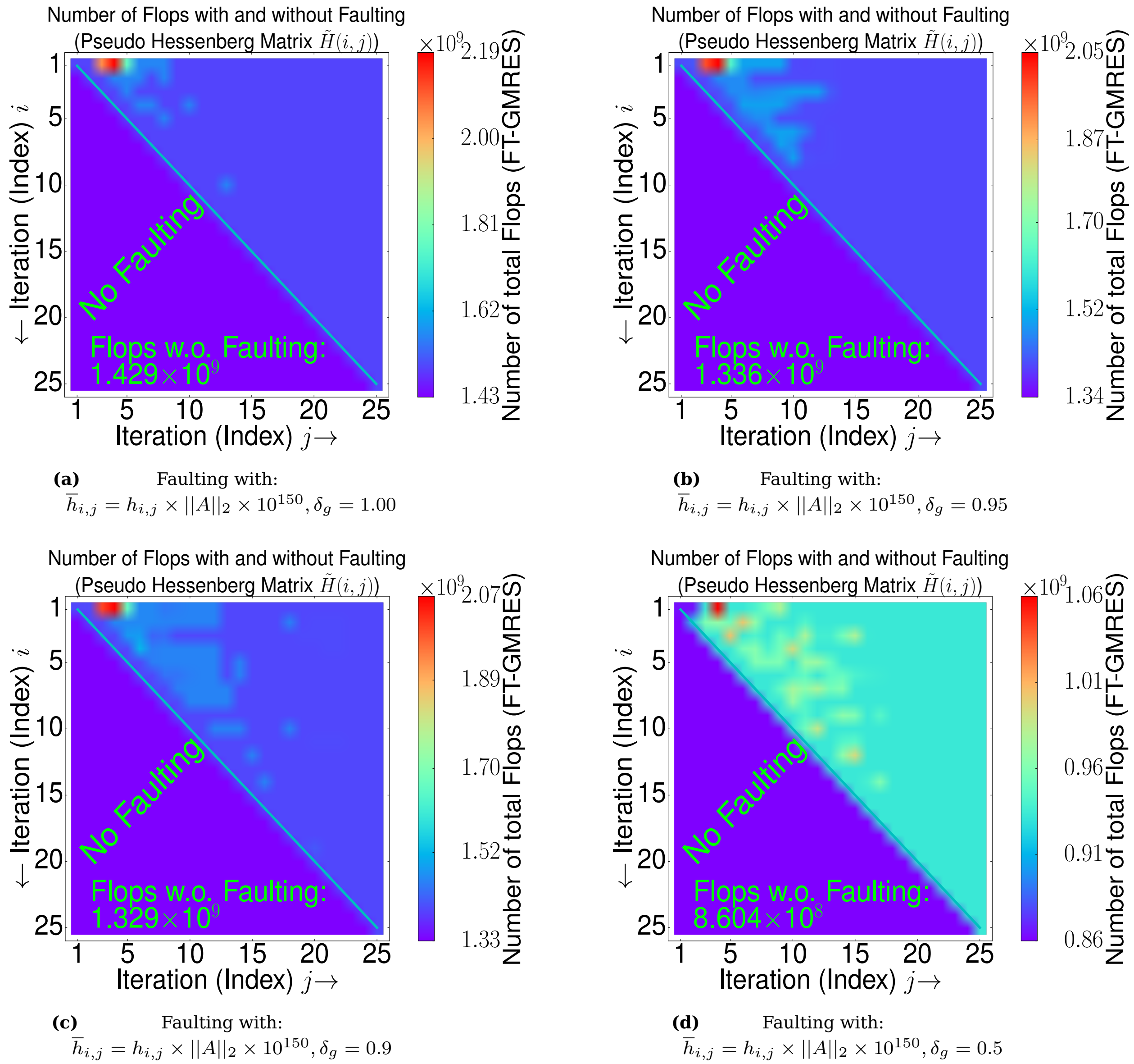
(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 106 Number of outer iterations for a single fault during orthogonalization ($h_{i,j}$) and for various disturbances while solving the chem_master1 matrix with ILU-preconditioning (and $Error_{ILU}{}^3 \approx 12 \times 10^{-2}$). The inner solver builds an upper Hessenberg matrix as in formula 104.

As in figure 100 also the chem_master1 matrix is solved in figure 106 but with ILU-preconditioning. The F(T)-GMRES takes 14 outer iterations (m_1) without a fault and for the first trial whereas 25 iterations are set for the inner solver (m_2), there are 17 outer iterations needed in the presence of a single fault. For each trial the relative residuum is below $10^{-5.5}$, the inner solver is initialized with zero values for vector $w_j(w_0)$. The worst overhead is 13 % (17/15) which is an improvement against the un-preconditioned case (21 %). The value of $Error_{ILU}{}^3$ is chosen such that most of the work is still done by the inner solver (intuitively) because low values of $Error_{ILU}$ indicate more work for the ILU-decomposition.

9.10 Faulting with single faults while solving further problems with applying flexible preconditioning by the inner solver of the FT-GMRES

9.10.1 Number of flops for solving the Pres_Poisson problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 1.00$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

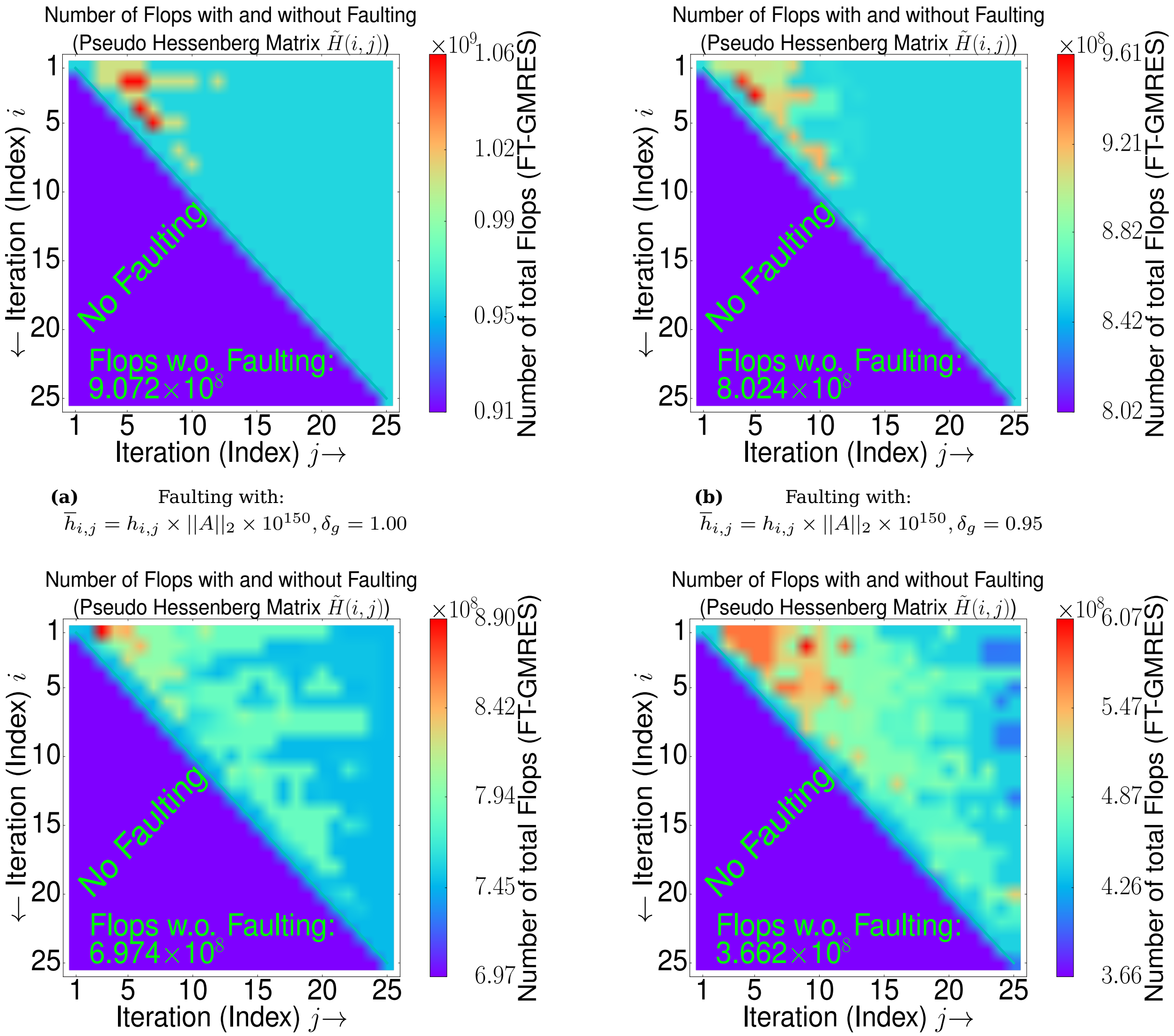
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.5$

Figure 107 Number of total flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the Pres_Poisson (sym.) matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 107 during solving the Pres_Poisson matrix. The inner solver of the FT-GMRES does at least 1 but at most 35 iterations (m_2). In figure 107 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead (≈ 10 - 40 %) does not remain the same for each δ_g. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-6} for all computations. The number of flops has been decreased from 1.429×10^9 to 8.604×10^8.

9.10.2 Number of flops for solving the Kuu problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 1.00$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

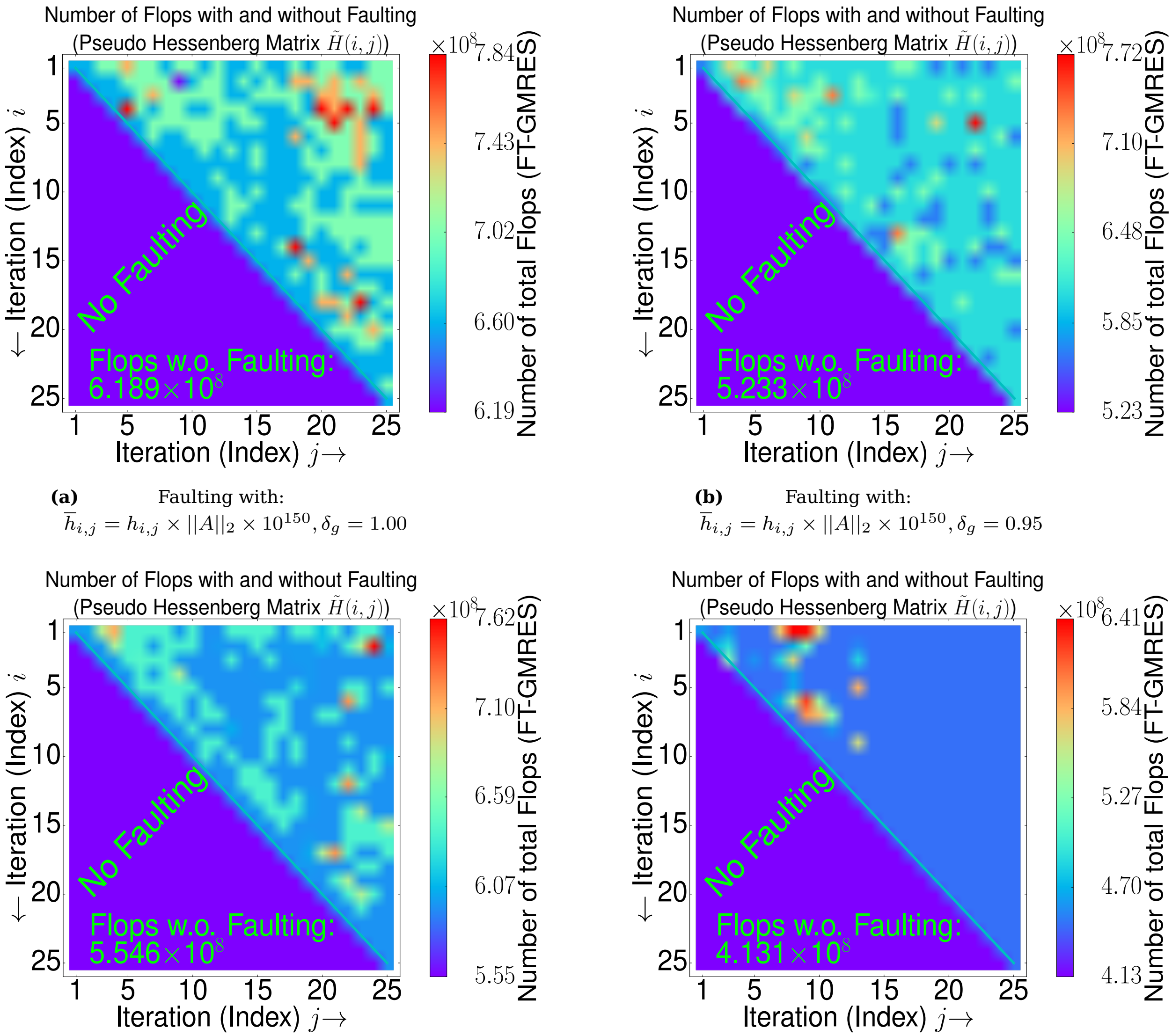
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.5$

Figure 108 Number of total flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the Kuu (symmetric) matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 108 during solving the Kuu matrix. The inner solver of the FT-GMRES does at least 1 but at most 35 iterations (m_2). In figure 108 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead (≈ 10 - 45 %) does not remain the same for each δ_g, lower values of δ_g lead to more overhead. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all computations and trials. The number of flops has been decreased from 9.072×10^8 to 3.662×10^8 for a failure free computation.

9.10.3 Number of flops for solving the Na5 problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 1.00$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

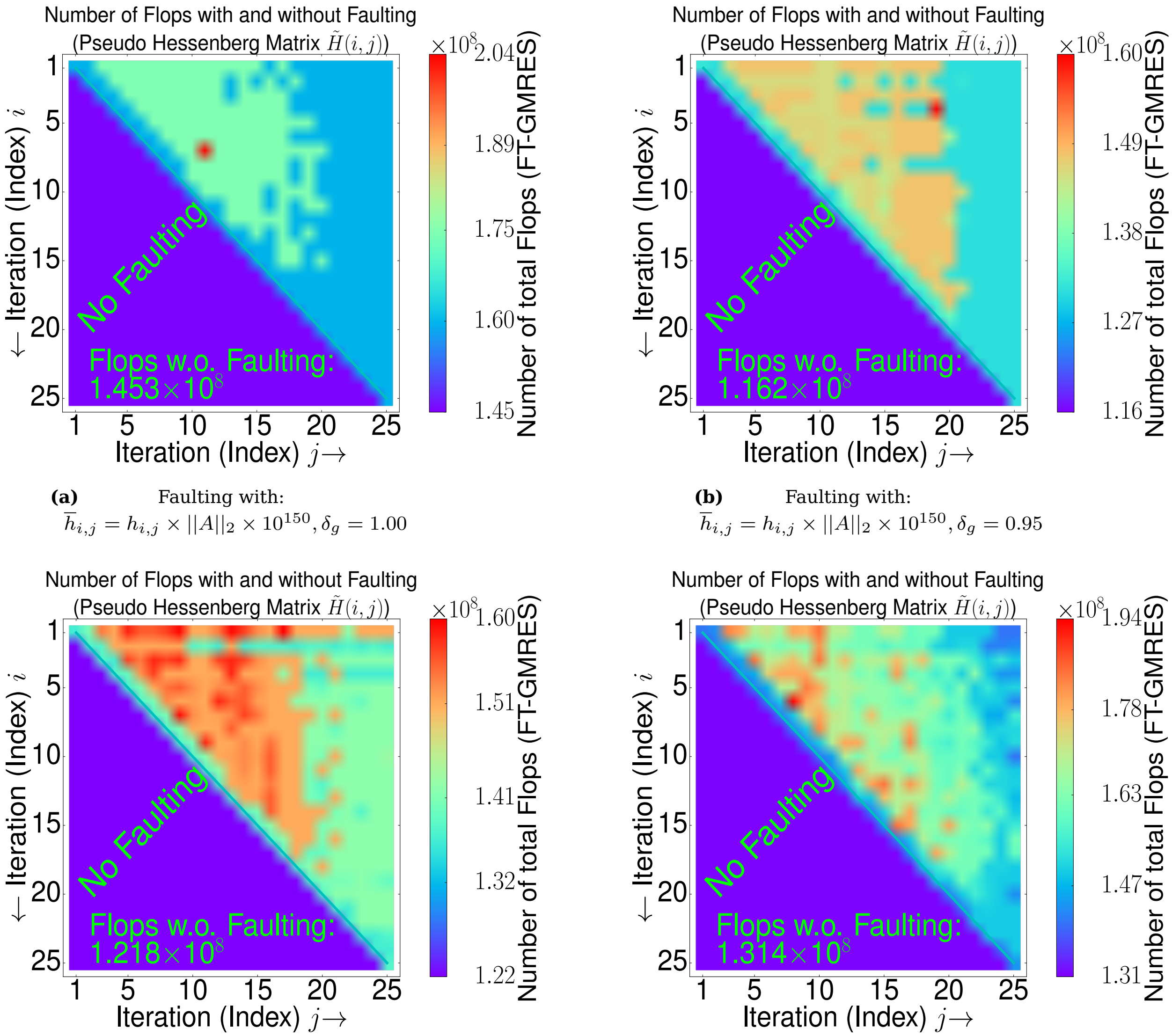
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.5$

Figure 109 Number of total flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the Na5 (symmetric) matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 109 during solving the Na5 matrix. The inner solver of the FT-GMRES does at least 1 but at most 35 iterations (m_2). In figure 109 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead (≈ 10 - 30 %) does not remain the same for each δ_g, lower values of δ_g lead to more overhead. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all computations and trials. The number of flops has been decreased from 6.189×10^8 to 4.131×10^8 for a failure free computation.

9.10.4 Number of flops for solving the circuit_2 problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 1.00$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

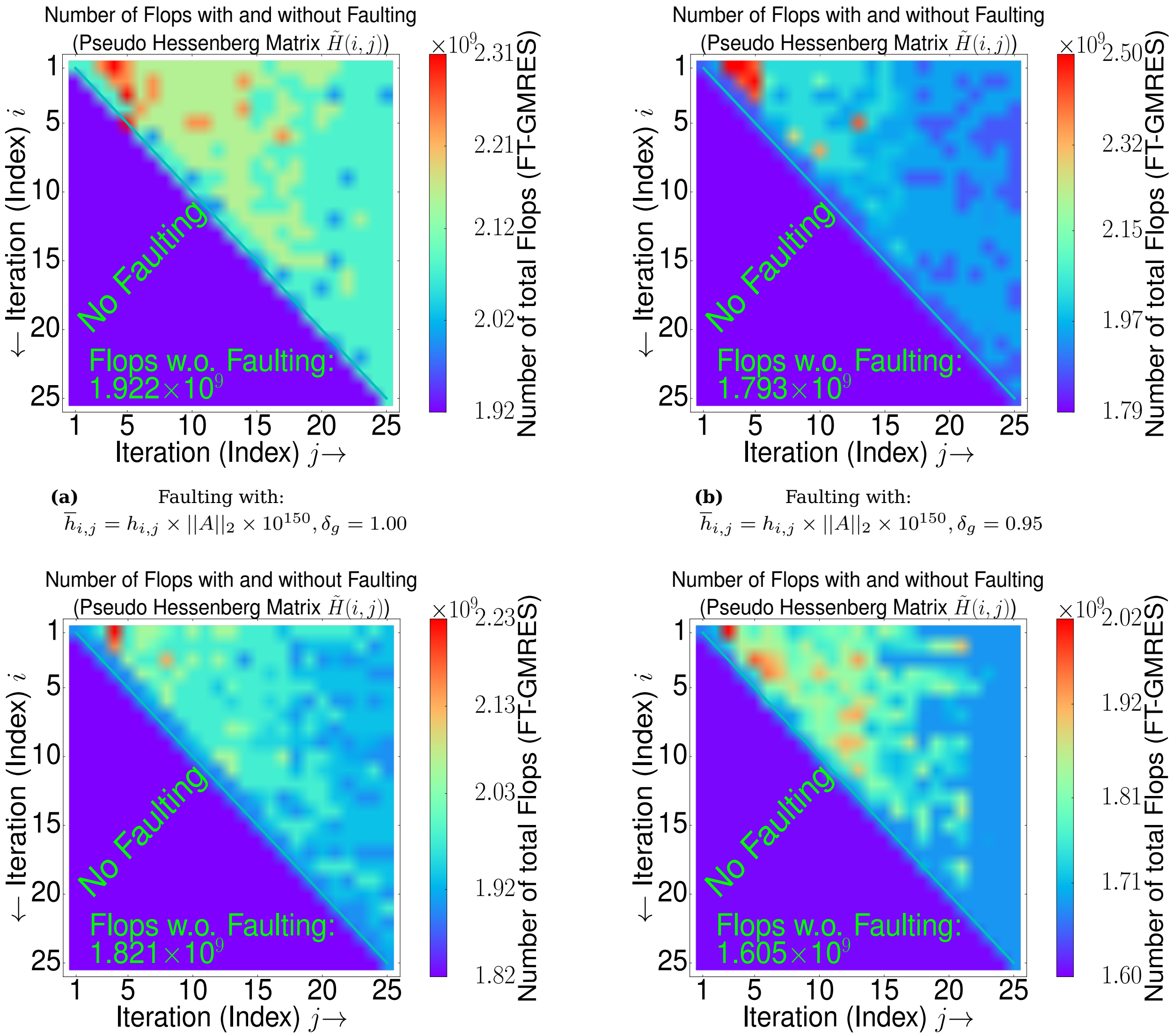
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.5$

Figure 110 Number of total flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the circuit_2 (un-symmetric) matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 110 during solving the circuit_2 matrix. The inner solver of the FT-GMRES does at least 1 but at most 35 iterations (m_2). In figure 110 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead (≈ 10 - 35 %) does not remain the same for each δ_g. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all trials. The number of flops has been decreased from 1.453×10^8 to 1.162×10^8 for a failure free run.

9.10.5 Number of flops for solving the mult_dcop_03 problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 1.00$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

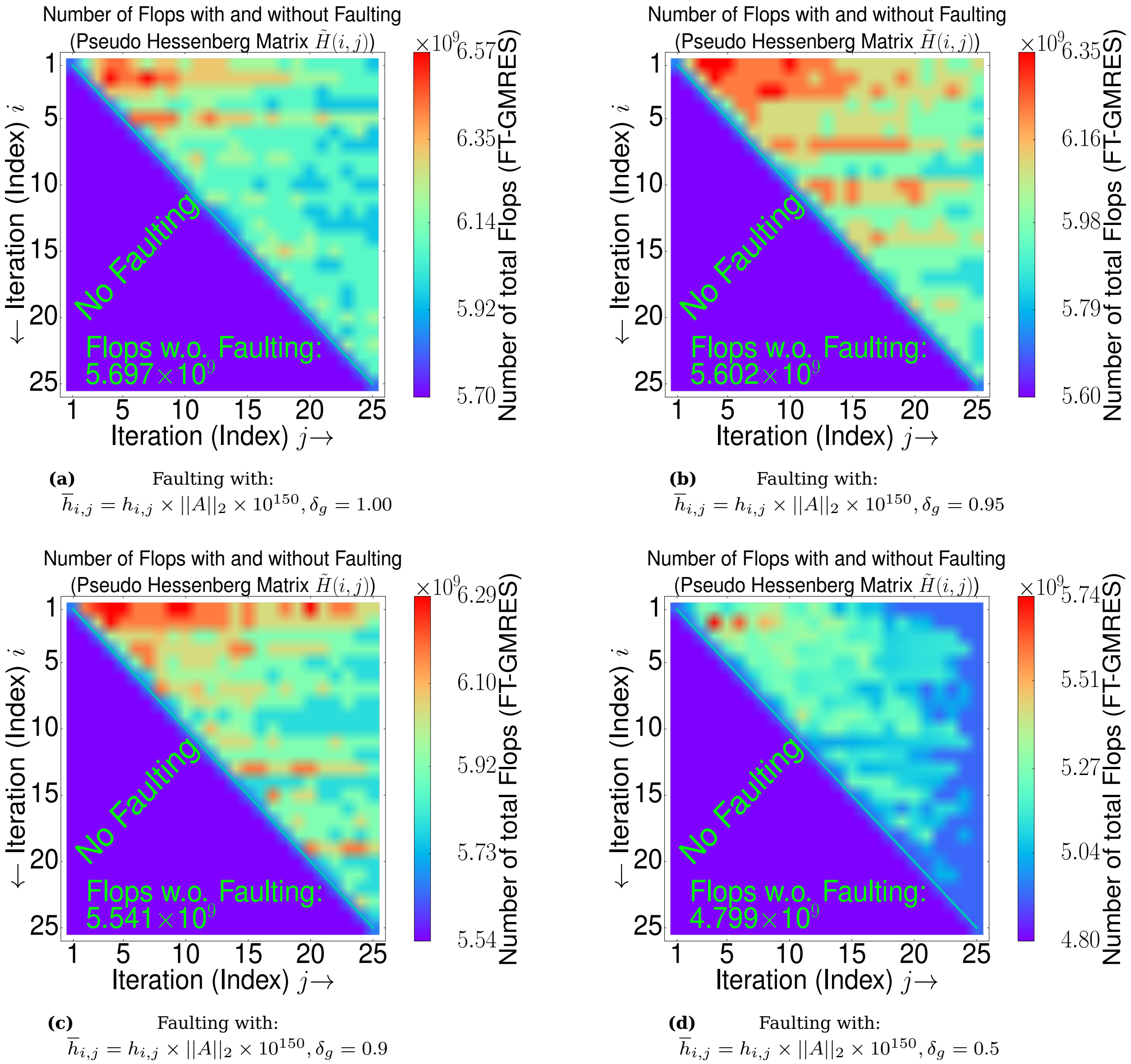
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.5$

Figure 111 Number of total flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the mult_dcop_03 (unsymmetric) matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 111 during solving the mult_dcop_03 matrix. The inner solver of the FT-GMRES does at least 1 but at most 35 iterations (m_2). In figure 111 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead (≈ 15 - 30 %) does not remain the same for each δ_g. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all computations. The number of flops has been decreased from 1.922×10^9 to 1.605×10^9.

9.10.6 Number of flops for solving the chem_master1 problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 1.00$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

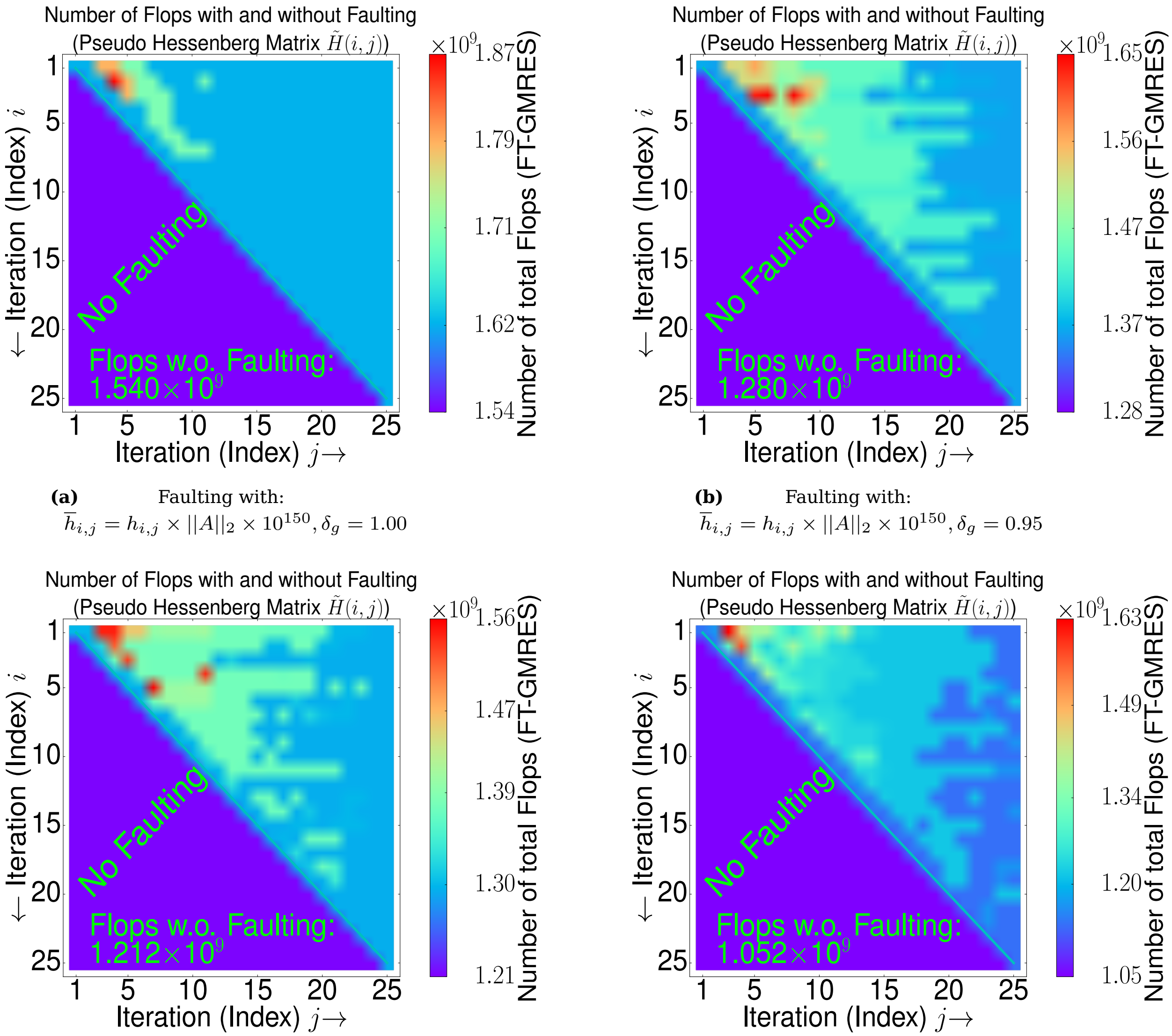
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.5$

Figure 112 Number of total flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the chem_master1 (unsymmetric) matrix.

As in section 9.6.4 flexible preconditioning is also applied in figure 112 during solving the chem_master1 matrix. The inner solver of the FT-GMRES does at least 1 but at most 35 iterations (m_2). In figure 112 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead (≈ 10 - 16 %) does remain the same for each δ_g. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below $10^{-5.5}$ for all trials. The number of flops has been decreased from 5.697×10^9 to 4.799×10^9.

9.10.7 Number of flops for solving the Ill_Stokes problem with flexible preconditioning by the inner solver and faulting - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 1.00$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.95$

(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.9$

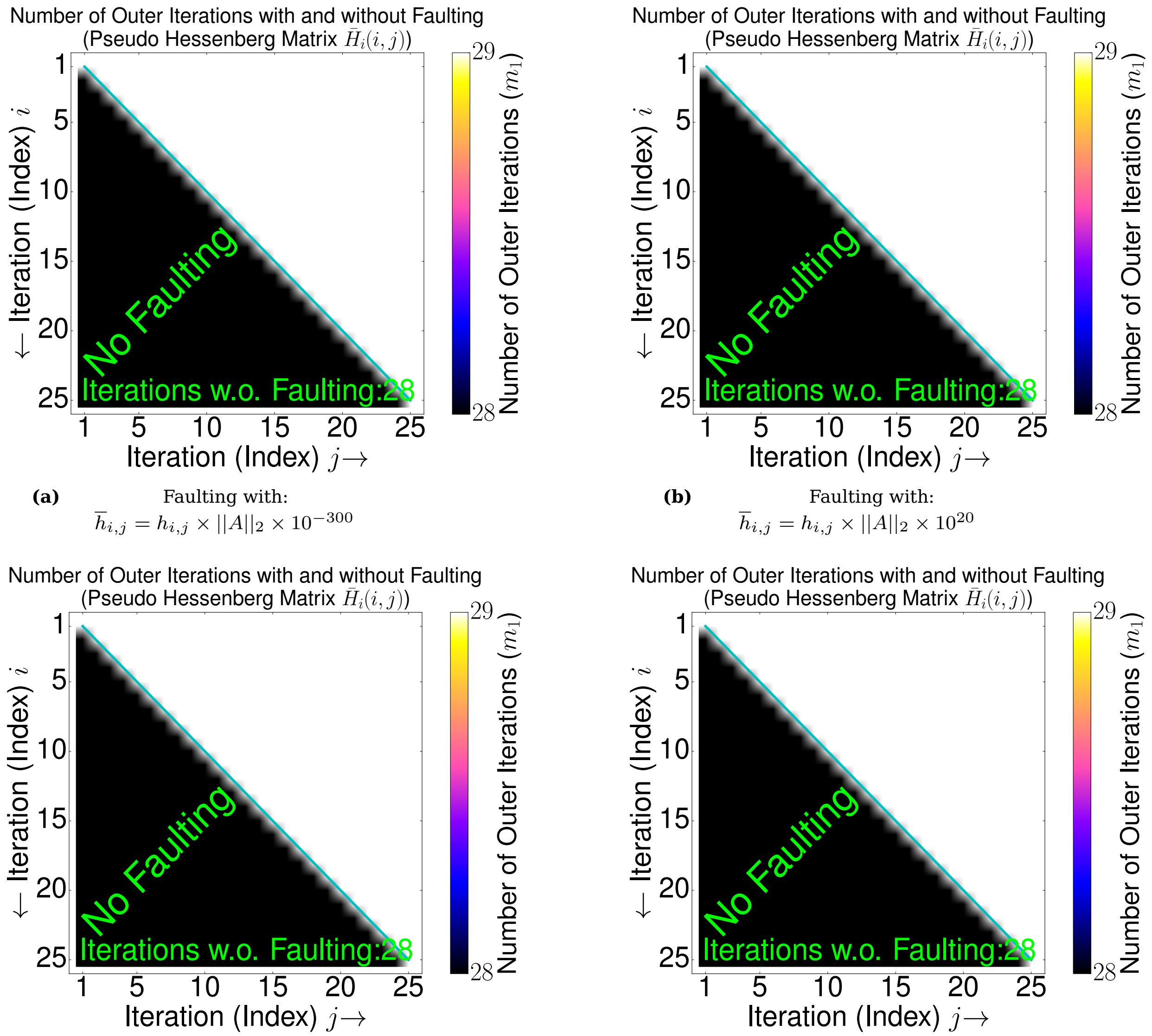
(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_g = 0.5$

Figure 113 Number of total flops for a single fault on a specific position during orthogonalization ($h_{i,j}$) of the inner solver with flexible preconditioning while solving the Ill_Stokes (un-symmetric) matrix. Lower values of δ_g lead to more overhead in the presence of a single fault.

As in section 9.6.4 flexible preconditioning is also applied in figure 113 during solving the Ill_Stokes matrix. The inner solver of the FT-GMRES does at least 1 but at most 35 iterations (m_2). In figure 113 always the same error is induced for a single fault but the value of δ_g ($\delta_g = 1 - \frac{||\sigma_j||_2^2}{||\sigma_0||_2^2}$) is changed, if this value is achieved during preconditioning of the inner solver then this solver is stopped and the outer solver is allowed to continue. The worst overhead (≈ 15 - 35 %) does not remain the same for each δ_g. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-6} for all trials. The number of flops has been decreased from 1.540×10^9 to 1.052×10^9 for a failure free run.

9.11 Faulting with multiple faults while solving further problems

9.11.1 Faulting during solving the circuit_2 problem along index i - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{20}$

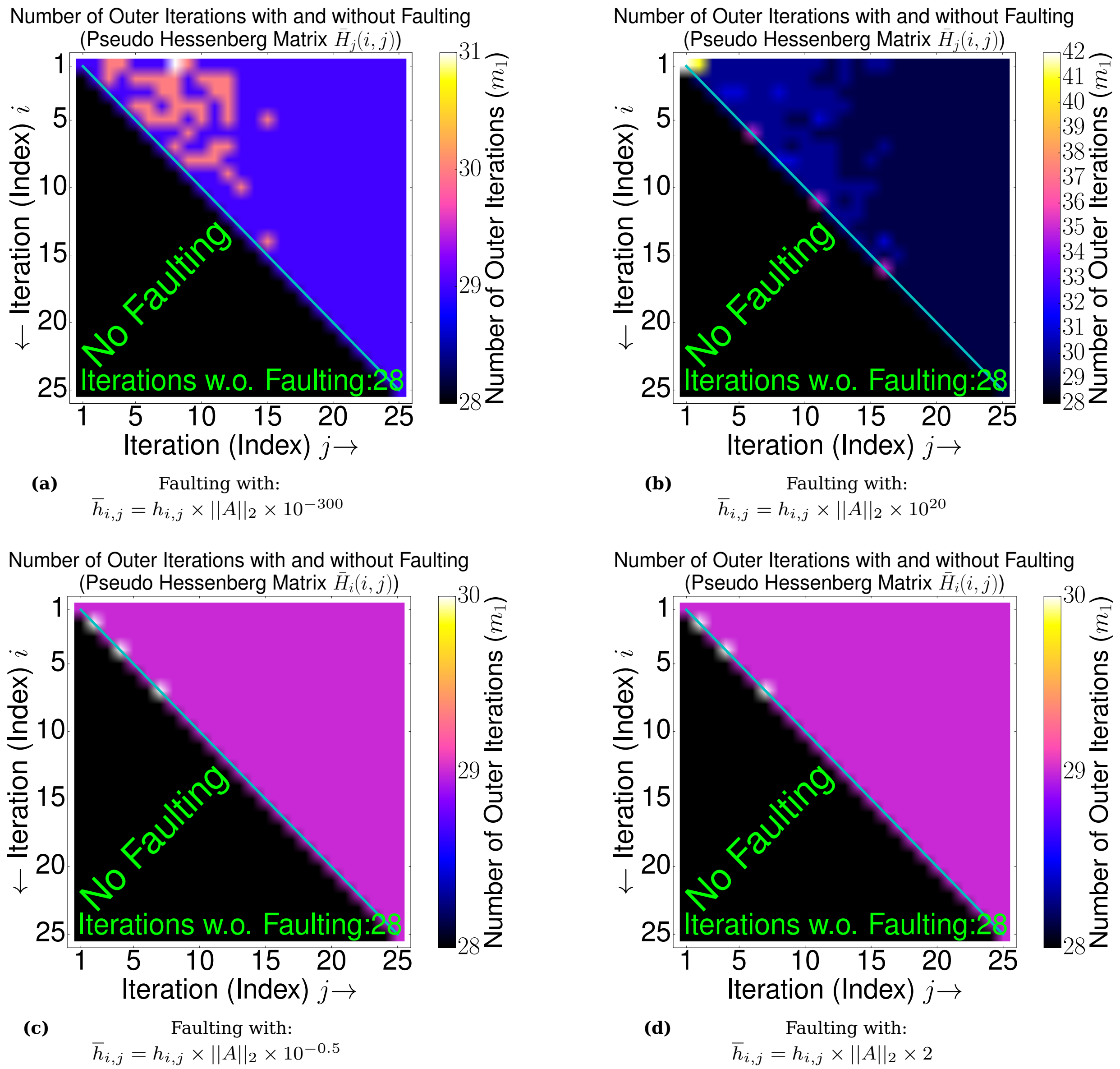
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 2$

Figure 114 Number of outer iterations for multiple faults during orthogonalization ($h_{i,j}$) of the inner solver along index i and for various disturbances while solving the circuit_2 problem.

In figure 114 faulting along index i with multiple faults is shown during solving the circuit_2 matrix. There is no overhead caused in figure 114, one additional iteration is needed for the FT-GMRES which is mainly because of achieving the same relative residuum ($||r||_2 = ||Ax - b||_2 / \ ||b||_2$) as the F-GMRES after computation (10^{-9}). The F-GMRES takes 28 outer iterations (m_1) without a fault for the first trial such that a relative residuum below 10^{-9} is achieved. Faulting along index i with multiple faults and with the same error seems to be almost uncritical in figure 114.

9.11.2 Faulting during solving the circuit_2 problem along index j - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(b) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{20}$

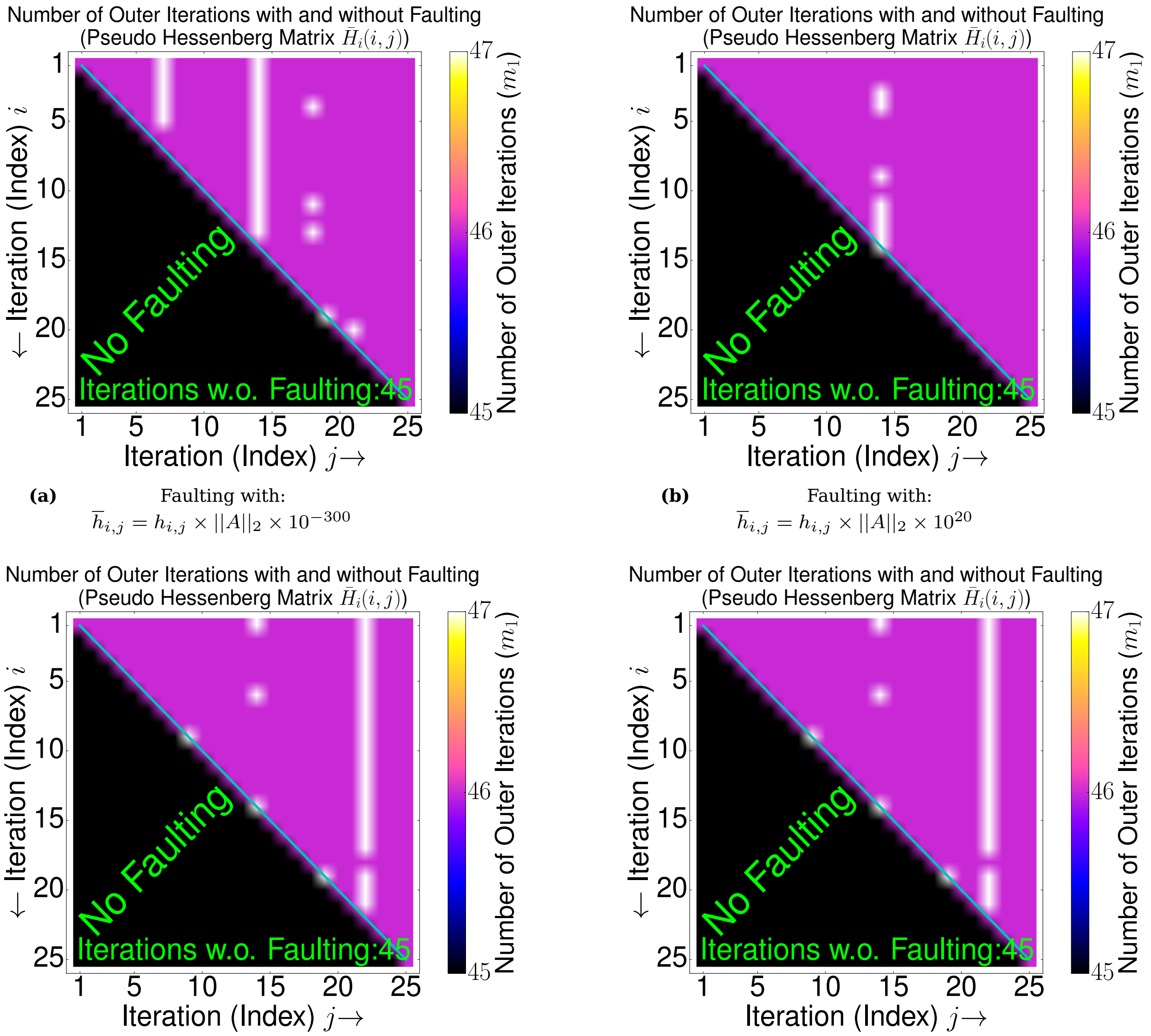
(c) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 2$

Figure 115 Number of outer iterations for multiple faults during orthogonalization ($h_{i,j}$) of the inner solver along index j and for various disturbances while solving the circuit_2 problem.

In figure 115 faulting along index j with multiple faults is shown during solving the circuit_2 problem. There is some overhead caused in figure 115b of about 45 % (42/29), one additional iteration is needed for the FT-GMRES which is mainly because of achieving the same relative residuum ($||r||_2 = ||Ax - b||_2 / \, ||b||_2$) as the F-GMRES after computation (10^{-9}). The F-GMRES takes 28 outer iterations (m_1) without a fault and for the first trial such that a relative residuum below 10^{-9} is achieved. Faulting along index j with multiple faults seems to be more critical in figure 115 especially in figure 115b. (Compare with figure 98.)

9.11.3 Faulting during solving the mult_dcop_03 problem along index i - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{20}$

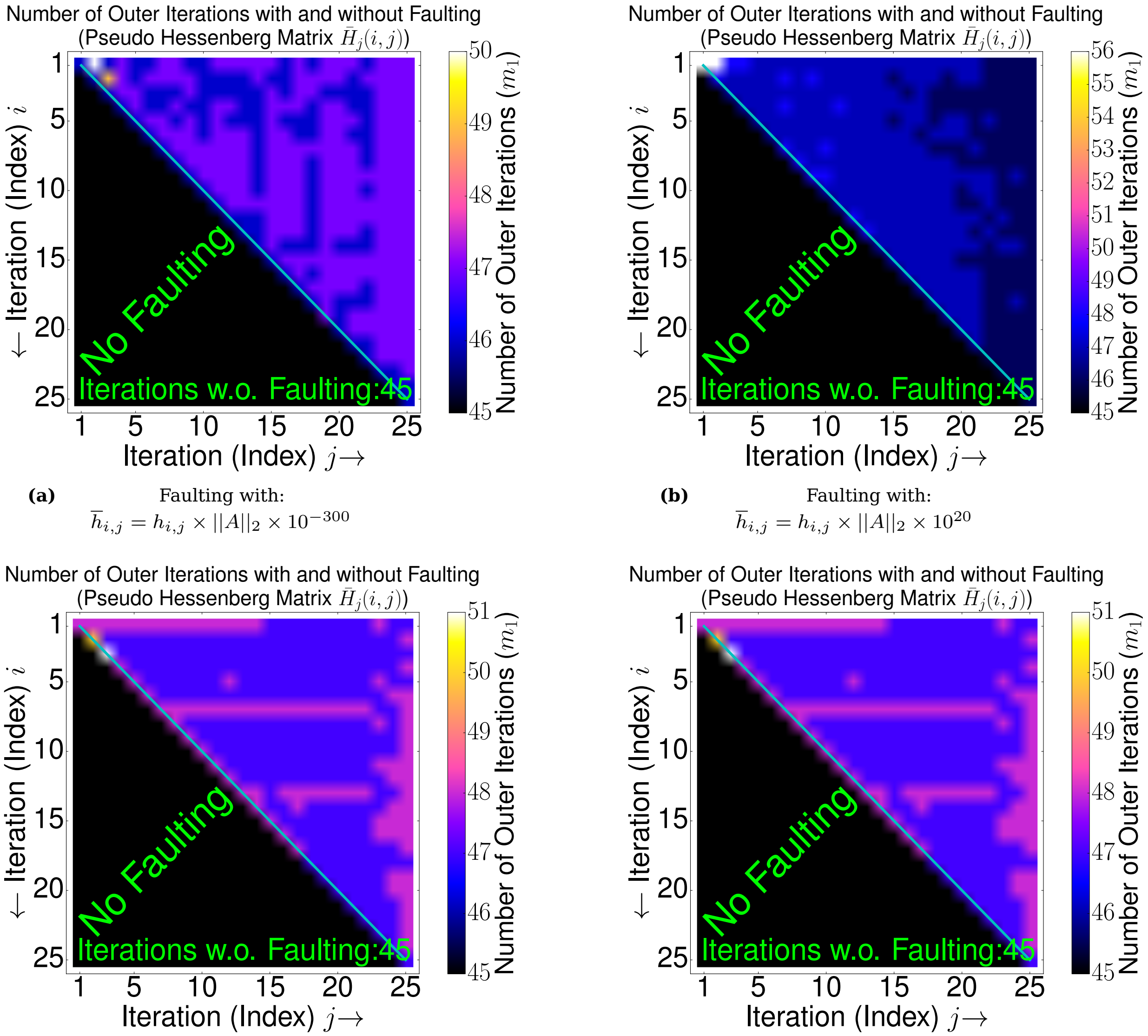
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 2$

Figure 116 Number of outer iterations for multiple faults during orthogonalization ($h_{i,j}$) of the inner solver along index i and for various disturbances while solving the mult_dcop_03 problem.

In figure 116 faulting along index i with multiple faults is shown during solving the mult_dcop_03 matrix. There is some overhead caused in figure 116b of about 2 % (47/46), one additional iteration is needed for the FT-GMRES which is mainly because of achieving the same relative residuum ($||r||_2 = ||Ax - b||_2 / \ ||b||_2$) as the F-GMRES after computation (10^{-9}). The F-GMRES takes 45 outer iterations (m_1) without a fault and for the first trial such that a relative residuum below 10^{-9} is achieved. Faulting along index i with multiple faults and with the same error seems to be almost uncritical in figure 116.

9.11.4 Faulting during solving the mult_dcop_03 problem along index j - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{20}$

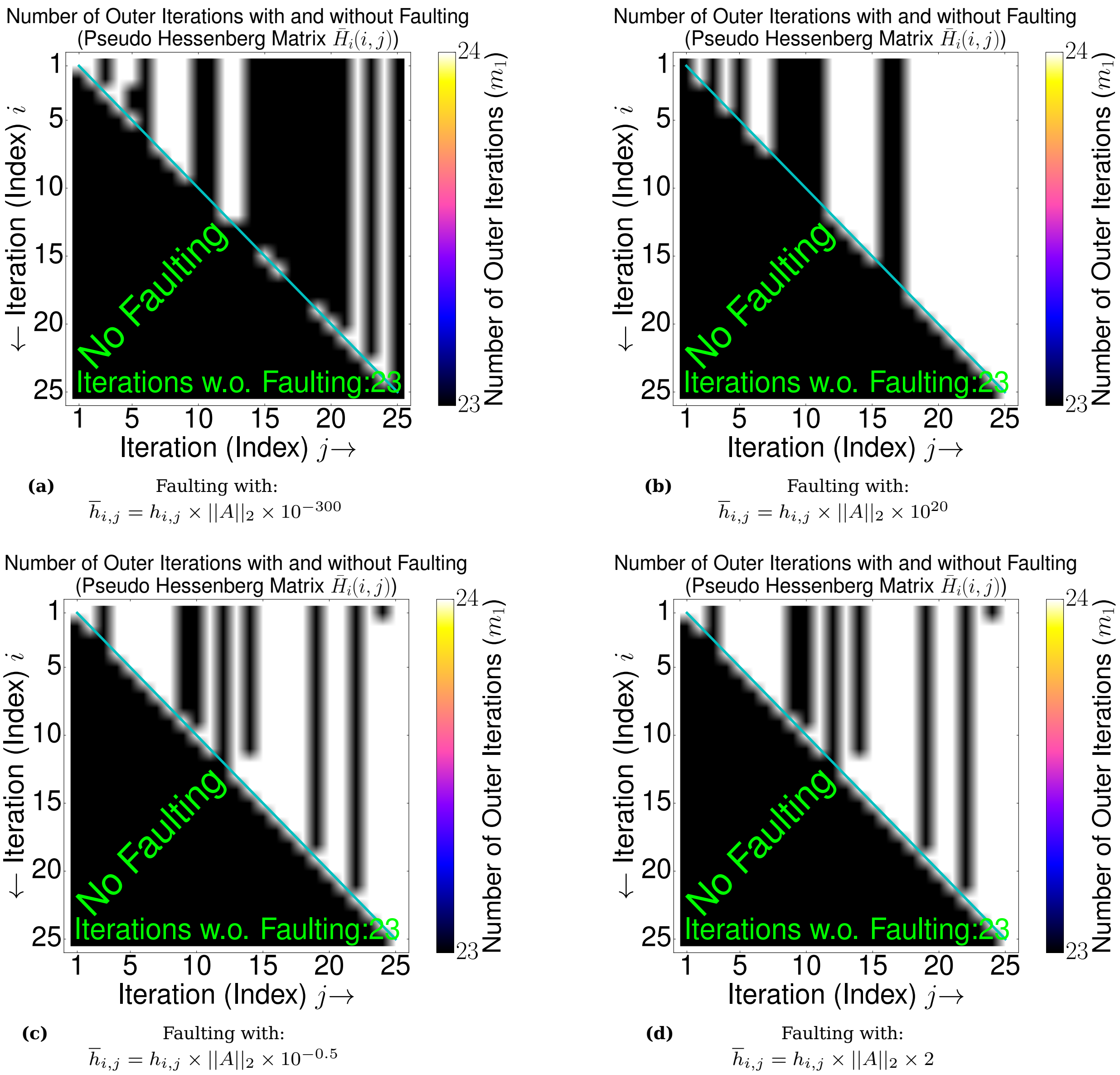
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 2$

Figure 117 Number of outer iterations for multiple faults during orthogonalization ($h_{i,j}$) of the inner solver along index j and for various disturbances while solving the mult_dcop_03 problem.

In figure 117 faulting along index j with multiple faults is shown during solving the mult_dcop_03 matrix. There is some overhead caused in figure 117b of about 22 % (56/46), one additional iteration is needed for the FT-GMRES which is mainly because of achieving the same relative residuum ($||r||_2 = ||Ax - b||_2/\ ||b||_2$) as the F-GMRES after computation (10^{-9}). The F-GMRES takes 45 outer iterations (m_1) without a fault and for the first trial such that a relative residuum below 10^{-9} is achieved. Faulting along index j with multiple faults seems to be more critical in figure 117 especially in figure 117b. (Compare with figure 99.)

9.11.5 Faulting during solving the ILL_Stokes problem along index i - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{20}$

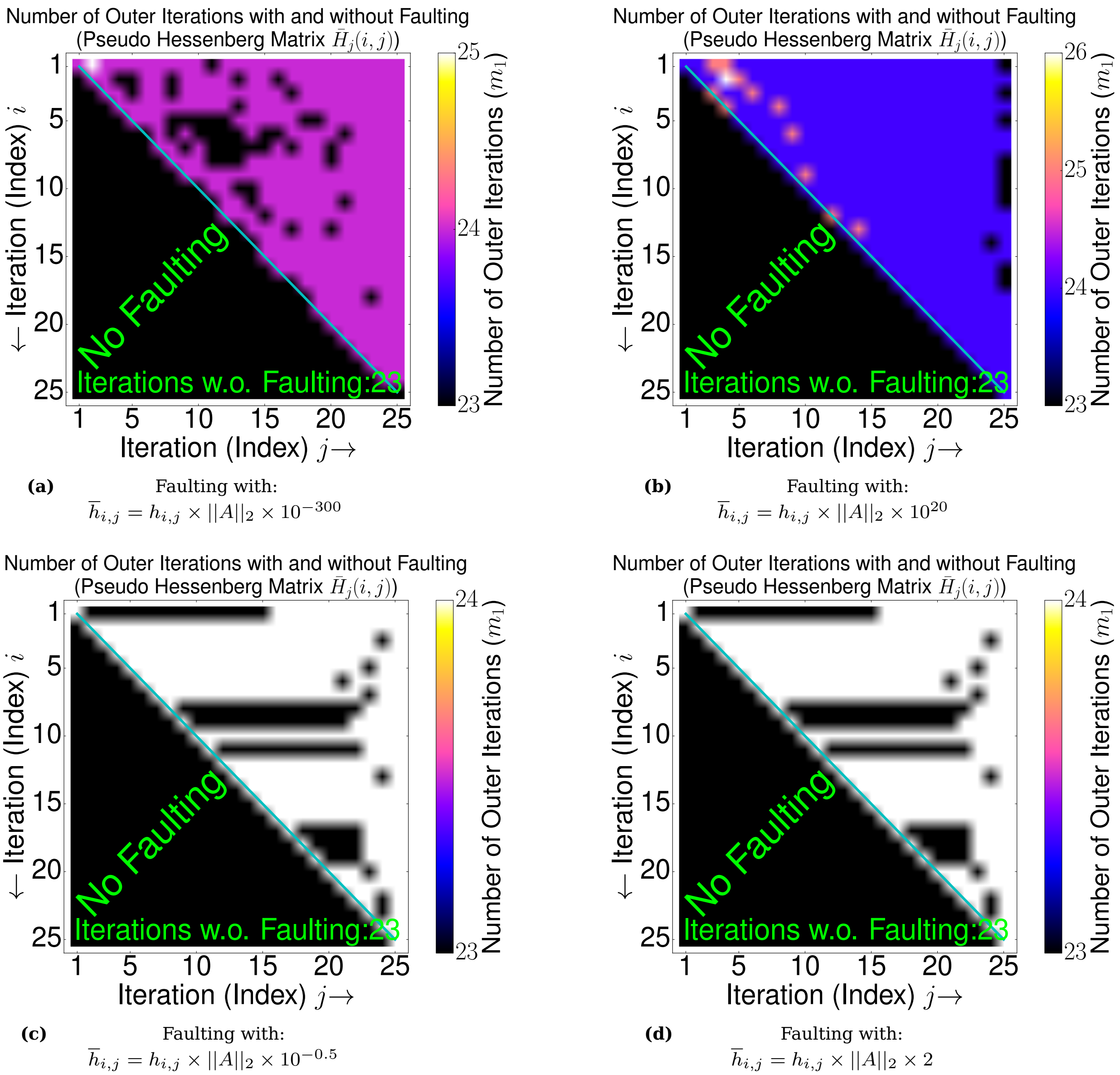
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 2$

Figure 118 Number of outer iterations for multiple faults during orthogonalization ($h_{i,j}$) of the inner solver along index i and for various disturbances while solving the ILL_Stokes problem.

In figure 118 faulting along index i with multiple faults is shown during solving the ILL_Stokes matrix. There is no obvious overhead caused in figure 118, one additional iteration is needed for the FT-GMRES which is mainly because of achieving the same relative residuum ($||r||_2 = ||Ax - b||_2/\ ||b||_2$) as the F-GMRES after computation (10^{-6}). The F-GMRES takes 23 outer iterations (m_1) without a fault and for the first trial such that a relative residuum below 10^{-6} is achieved. Faulting along index i with multiple faults and with the same error seems to be almost uncritical in figure 118.

9.11.6 Faulting during solving the ILL_Stokes problem along index j - Inner solver initialized with zero values (and with 25 iterations for the inner solver)



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{20}$

(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 2$

Figure 119 Number of outer iterations for multiple faults during orthogonalization ($h_{i,j}$) of the inner solver along index j and for various disturbances while solving the ILL_Stokes problem.

In figure 119 faulting along index j with multiple faults is shown during solving the ILL_Stokes matrix. There is some overhead caused in figure 119b of about 8 % (26/24), one additional iteration is needed for the FT-GMRES which is mainly because of achieving the same relative residuum ($||r||_2 = ||Ax - b||_2 / \ ||b||_2$) as the F-GMRES after computation (10^{-6}). The F-GMRES takes 23 outer iterations (m_1) without a fault and for the first trial such that a relative residuum below 10^{-6} is achieved. Faulting along index j with multiple faults seems to be more critical in figure 119 especially in 119b. (Compare with figure 101.)

9.12 Different disturbed Hessenberg matrices for single faults

9.12.1 Introduction

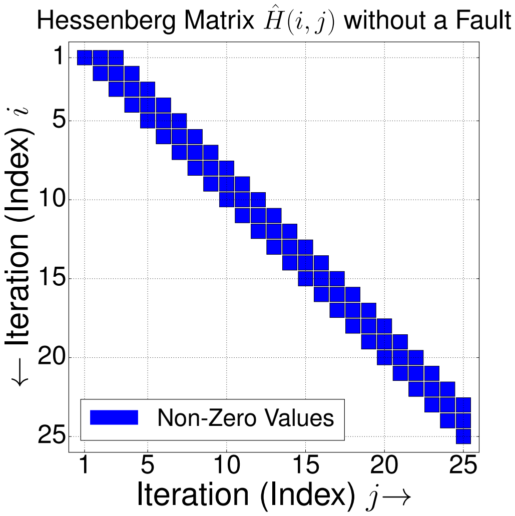


Figure 120 The final Hessenberg matrix (of the inner solver) without a fault in the case of solving a symmetric problem, after applying all Givens rotations (see section 5).

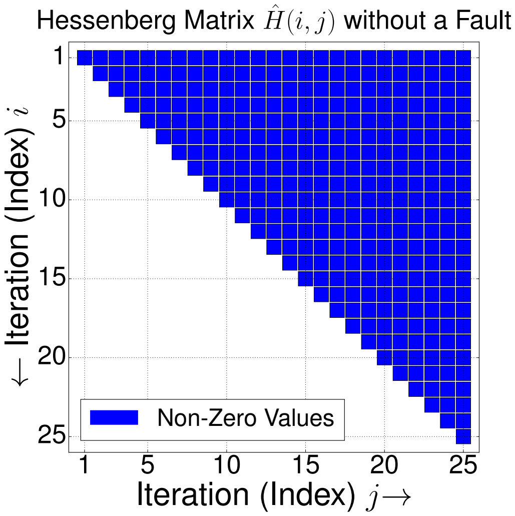
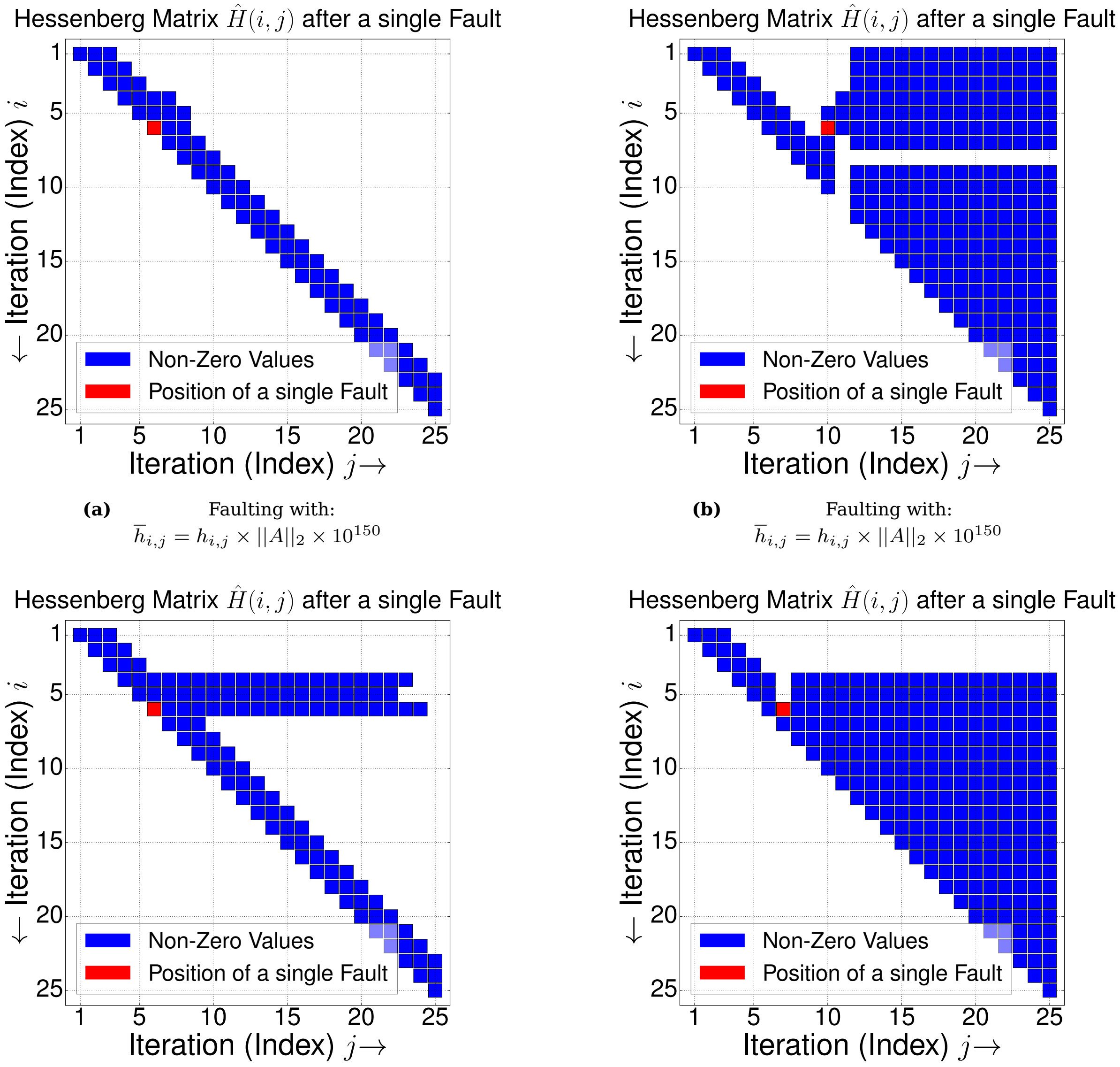


Figure 121 The final Hessenberg matrix (of the inner solver) without a fault in the case of solving an un-symmetric problem, after applying all Givens rotations (see section 5).

In figure 120 and 121 the final Hessenberg matrices after applying all Givens rotations are shown, in the case of applying 25 iterations (m_2) for the inner solver. Non-zero values of these matrices where the absolutes magnitudes are greater than a specific value (10^{-9}) are shown with a blue color in figure 120 and 121. If matrix A is symmetric then the GMRES solver builds a Hessenberg matrix like in figure 120 otherwise the problem is un-symmetric then matrix $\hat{H}$ has an appearance like in figure 121. Symmetric matrices are problems where the condition of $A = A^T$ is satisfied (, rows and columns are the same). (Hence multiple faults will also change the structure of the Hessenberg matrix like for faults in section 9.3 which is not shown in this master work).

9.12.2 Different disturbed Hessenberg matrices for solving the 2D Poisson problem



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

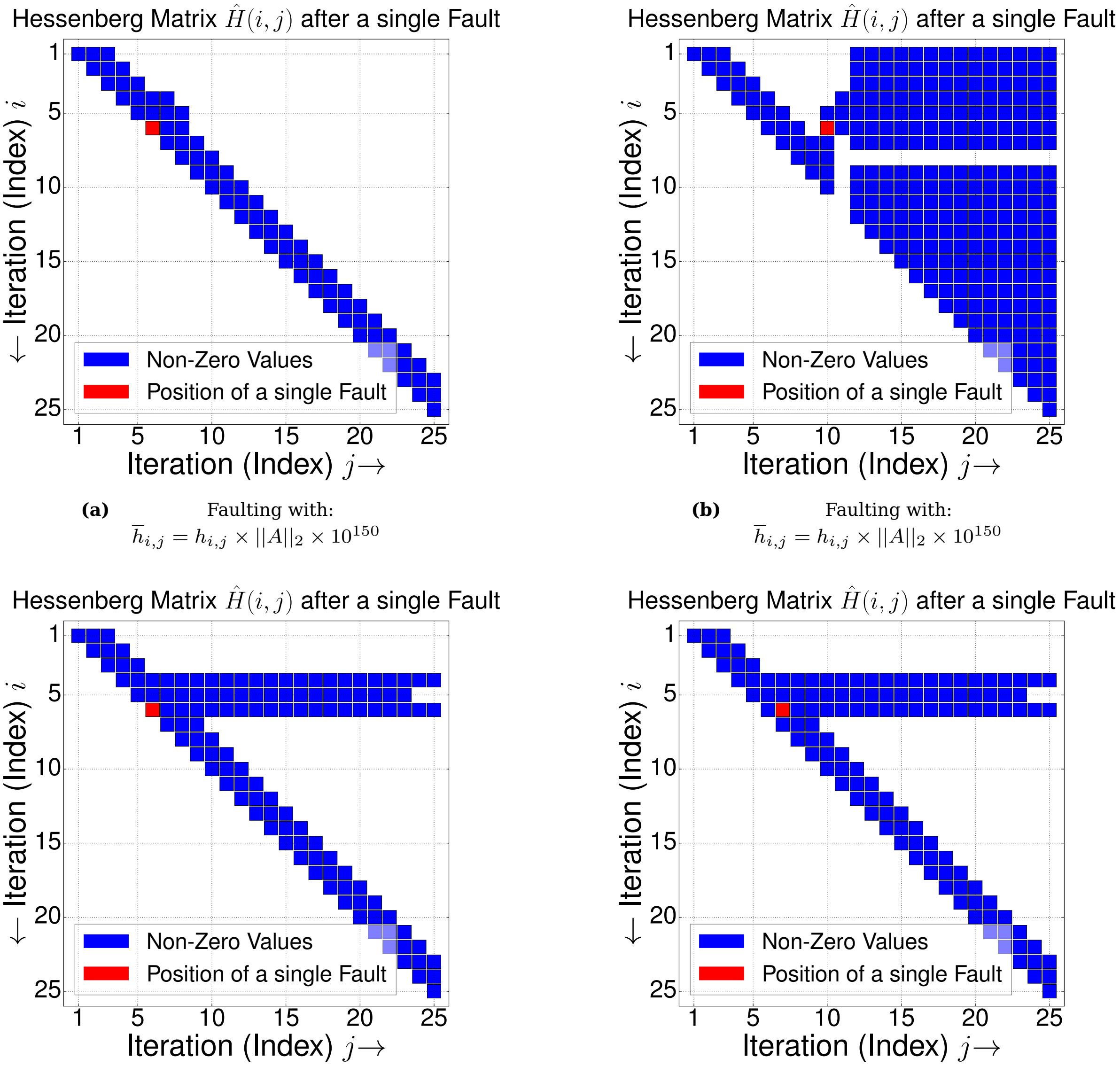
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

Figure 122 Different disturbed Hessenberg matrices (inner solver) for solving the 2D Poisson problem after a single fault on a specific position ($h_{i,j}$) in the inner solver of the FT-GMRES.

In figure 122 final Hessenberg matrices after 25 iterations (m_2) for a single fault on a specific position ($h_{i,j}$) during solving the 2D Poisson matrix are shown. In the case of this symmetric problem the inner solver of the FT-GMRES also builds a Hessenberg matrix which is at least an upper Hessenberg matrix but banded (tridiagonal) like in figure 120. Non-zero values which absolute magnitudes are greater than 10^{-9} are visualized with a blue color in figure 122 whereas the red color shows the position for a single fault of $h_{i,j}$. There is an oblivious change of the structure of the Hessenberg matrix because of a single fault, this deviation can be used for fault detection.

9.12.3 Different disturbed Hessenberg matrices for solving the Pres_Poisson problem



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

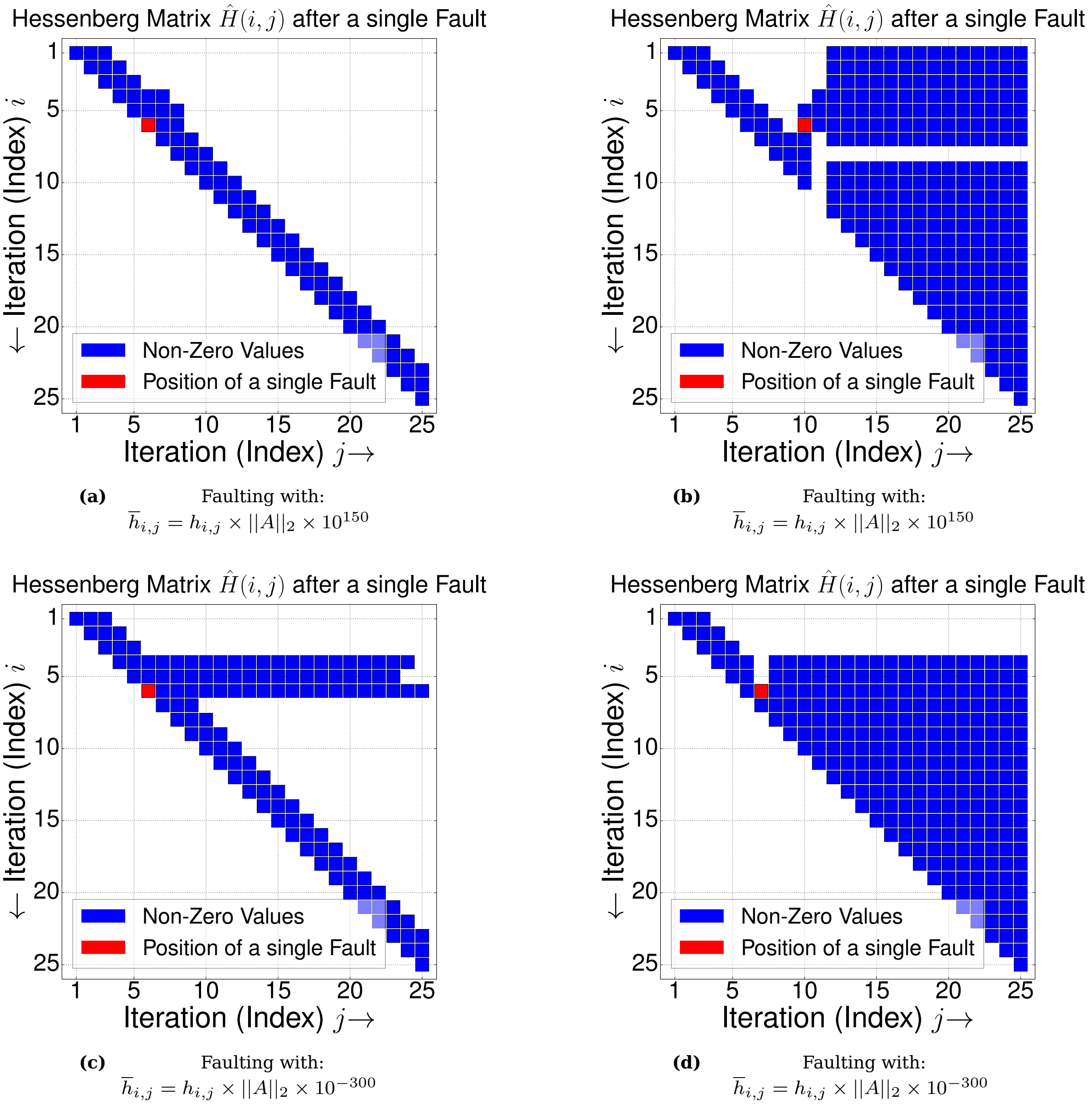
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

Figure 123 Different disturbed Hessenberg matrices (inner solver) for solving the Pres_Poisson problem after a single fault on a specific position ($h_{i,j}$) in the inner solver of the FT-GMRES.

In figure 123 final Hessenberg matrices after 25 iterations (m_2) for a single fault on a specific position ($h_{i,j}$) during solving the Pres_Poisson matrix are shown. In the case of this symmetric problem the inner solver of the FT-GMRES also builds a Hessenberg matrix which is at least an upper Hessenberg matrix but banded (tridiagonal) like in figure 120. Non-zero values which absolute magnitudes are greater than 10^{-9} are visualized with a blue color in figure 123 whereas the red color shows the position for a single fault of $h_{i,j}$. There is an oblivious change of the structure of the Hessenberg matrix because of a single fault, this deviation can be used for fault detection.

9.12.4 Different disturbed Hessenberg matrices for solving the Kuu problem



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

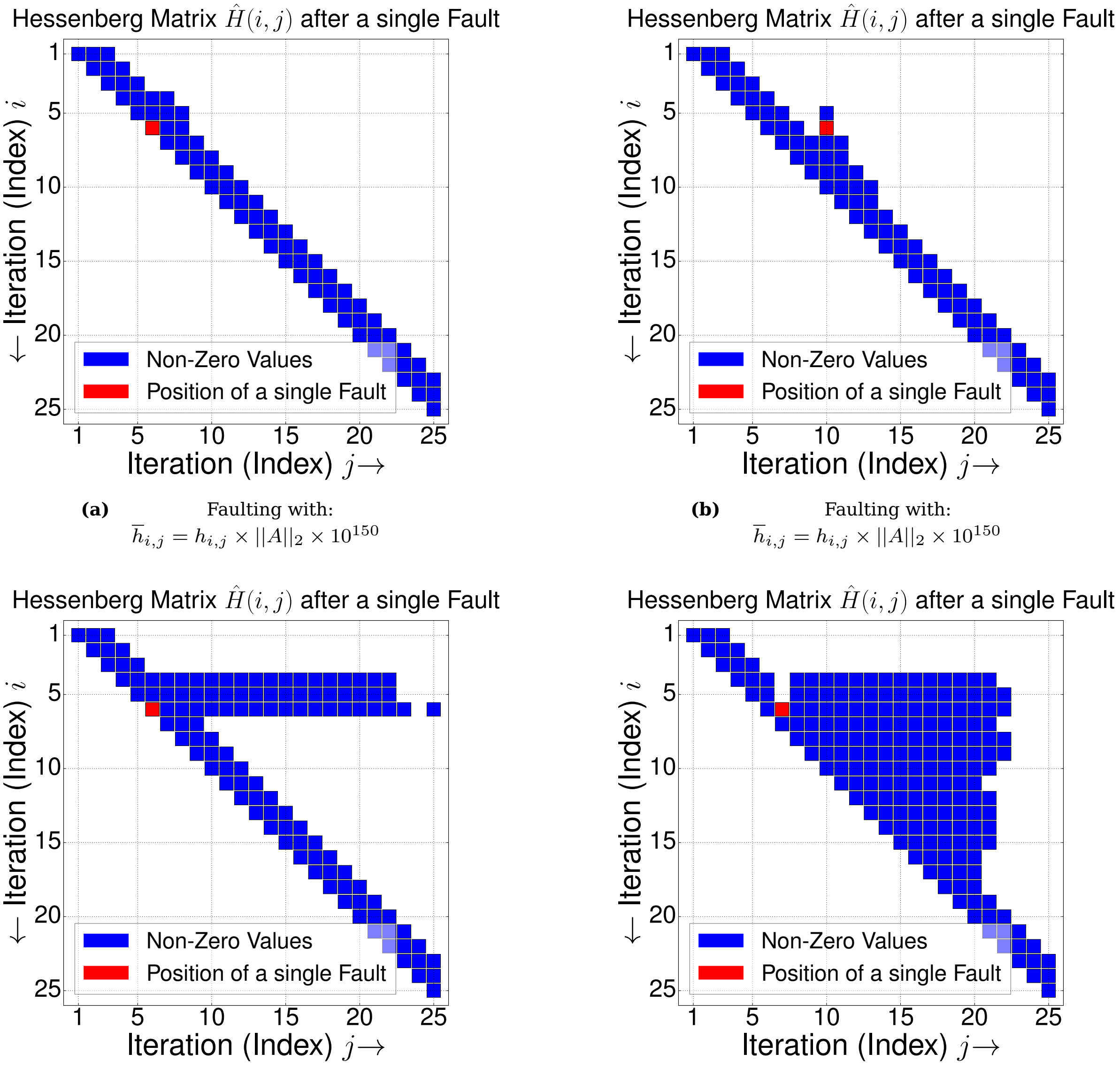
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

Figure 124 Different disturbed Hessenberg matrices (inner solver) for solving the Kuu problem after a single fault on a specific position ($h_{i,j}$) in the inner solver of the FT-GMRES.

In figure 124 final Hessenberg matrices after 25 iterations (m_2) for a single fault on a specific position ($h_{i,j}$) during solving the Kuu matrix are shown. In the case of this symmetric problem the inner solver of the FT-GMRES also builds a Hessenberg matrix which is at least an upper Hessenberg matrix but banded (tridiagonal) like in figure 120. Non-zero values which absolute magnitudes are greater than 10^{-9} are visualized with a blue color in figure 124 whereas the red color shows the position for a single fault of $h_{i,j}$. There is an oblivious change of the structure of the Hessenberg matrix because of a single fault, this deviation can be used for fault detection.

9.12.5 Different disturbed Hessenberg matrices for solving the adder_dcop_63 problem



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

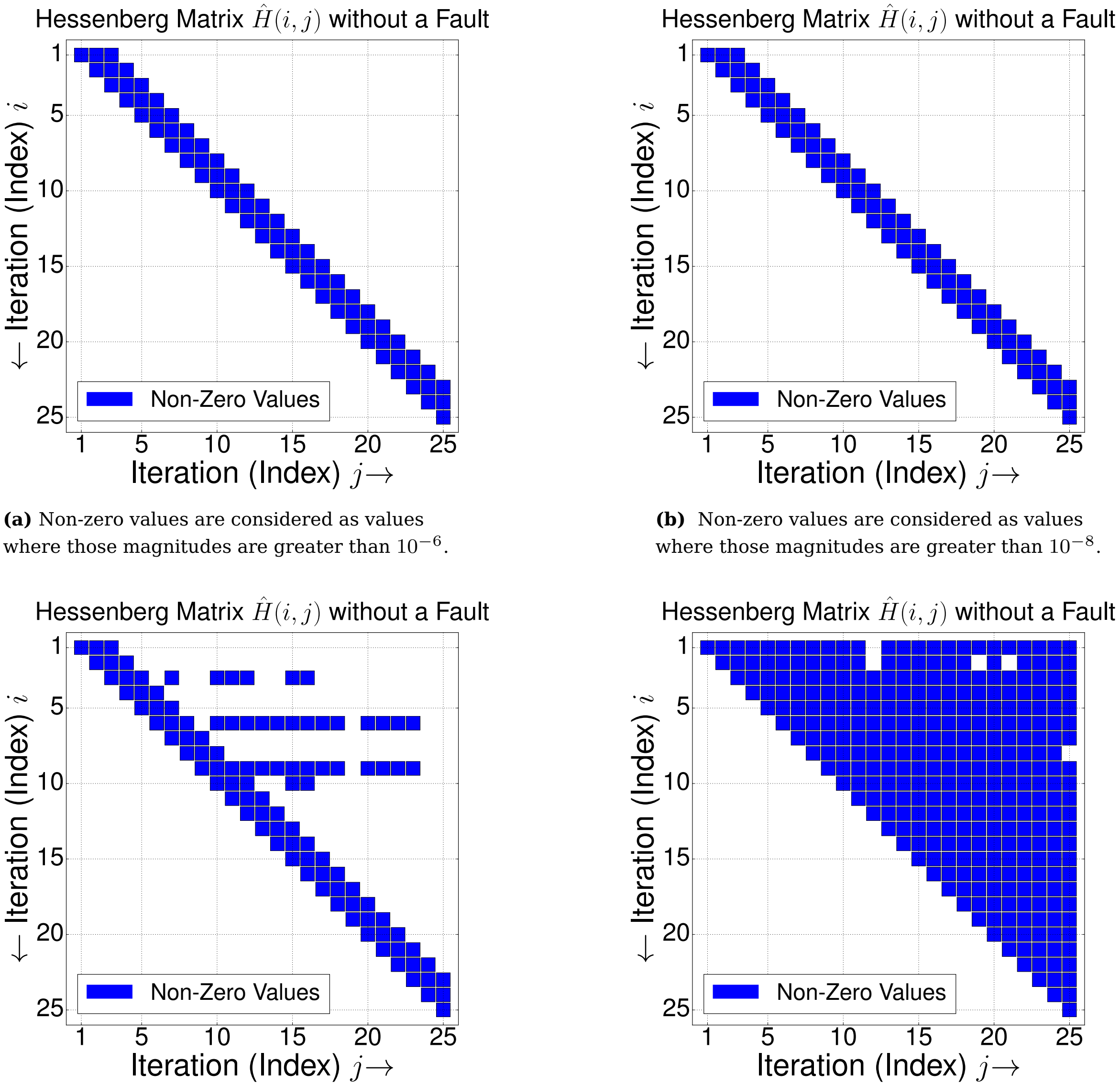
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

Figure 125 Different disturbed Hessenberg matrices (inner solver) for solving the adder_dcop_63 problem after a single fault on a specific position ($h_{i,j}$) in the inner solver of the FT-GMRES.

In figure 125 final Hessenberg matrices after 25 iterations (m_2) for a single fault on a specific position ($h_{i,j}$) during solving the adder_dcop_63 matrix are shown. In the case of this un-symmetric problem which is close to a symmetric problem (see section 9.12.6) the inner solver of the FT-GMRES builds unexpected a Hessenberg matrix which is at least an upper Hessenberg matrix but banded (tridiagonal) like in figure 120. Non-zero values which absolute magnitudes are greater than 10^{-6} are visualized with a blue color in figure 125 whereas the red color shows the position for a single fault of $h_{i,j}$. There is an oblivious change of the structure of the Hessenberg matrix.

9.12.6 Different Hessenberg matrices for solving the adder_dcop_63 problem (no faults)



(a) Non-zero values are considered as values where those magnitudes are greater than 10^{-6}.

(b) Non-zero values are considered as values where those magnitudes are greater than 10^{-8}.

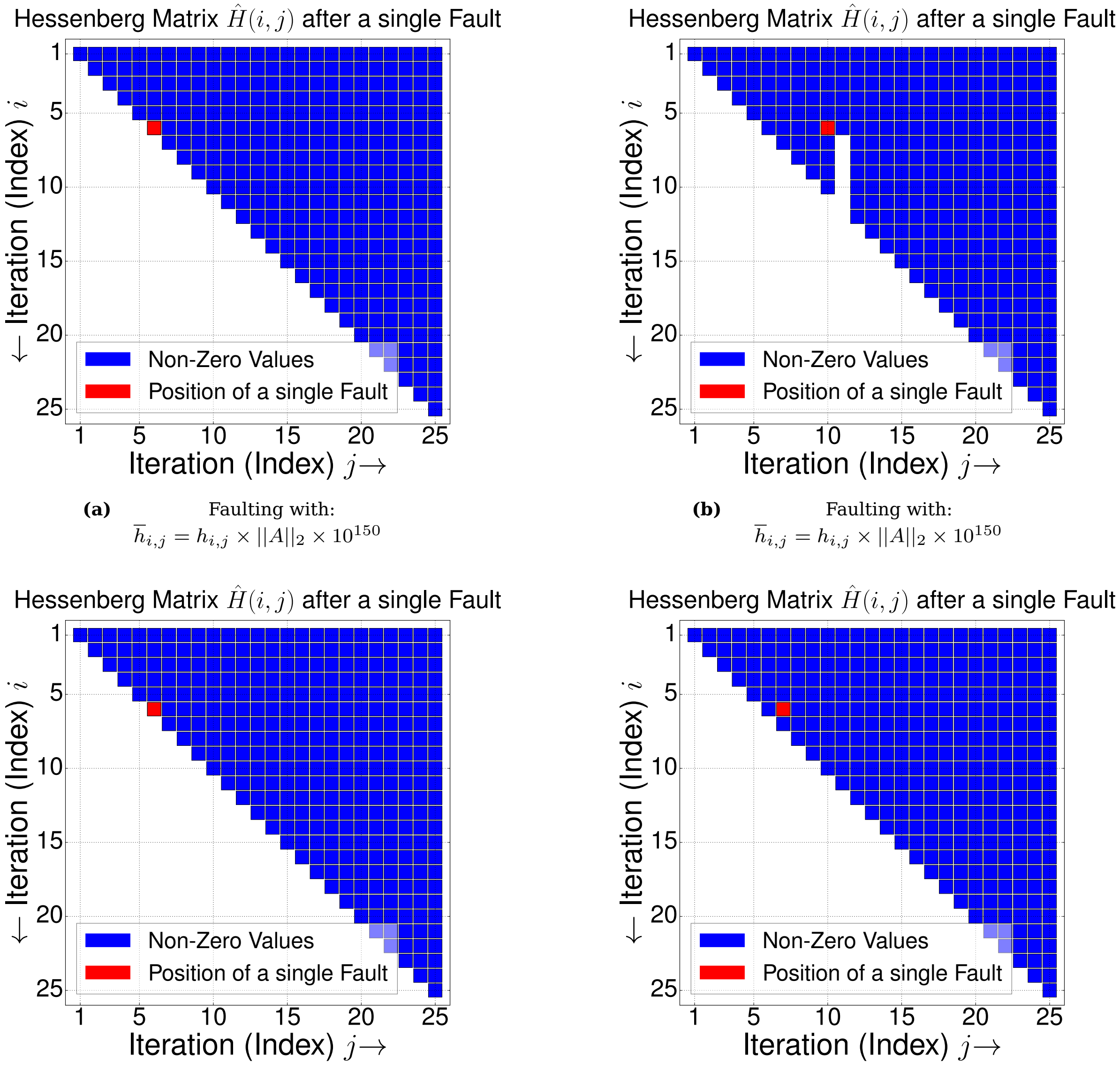
(c) Non-zero values are considered as values where those magnitudes are greater than 10^{-9}.

(d) Non-zero values are considered as values where those magnitudes are greater than 10^{-13}.

Figure 126 Different structures of the Hessenberg matrix in the inner solver of the F-GMRES through solving the adder_dcop_63 problem with values of various magnitudes as non-zero entries.

Figure 126 shows different Hessenberg matrices in the inner solver of the F-GMRES through solving the adder_dcop_63 problem with values of different magnitudes which are considered as non-zero entries of matrix $\hat{H}$. This problem is close to a symmetric matrix because matrix A can be decomposed in a symmetric (A_{sym}) and un-symmetric part (A_{unsym}) such that $A_{sym} = \frac{1}{2}(A + A^T)$ whereas $A_{unsym} = \frac{1}{2}(A - A^T)$ [69][76]. It follows that $A = A_{sym} + A_{unsym}$ then this also leads to $1 = ||A_{sym}||_2/||A||_2 + ||A_{unsym}||_2/||A||_2$ which measures the contribution of each part for matrix A. In this case $||A_{sym}||_2/||A||_2 \approx 1$ where $||A_{unsym}||_2/||A||_2 \approx 10^{-8}$ such that matrix A is nearly symmetric, so for this problem the (F-)GMRES builds unexpected a tridiagonal matrix (mainly).

9.12.7 Different disturbed Hessenberg matrices for solving the circuit_2 problem



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

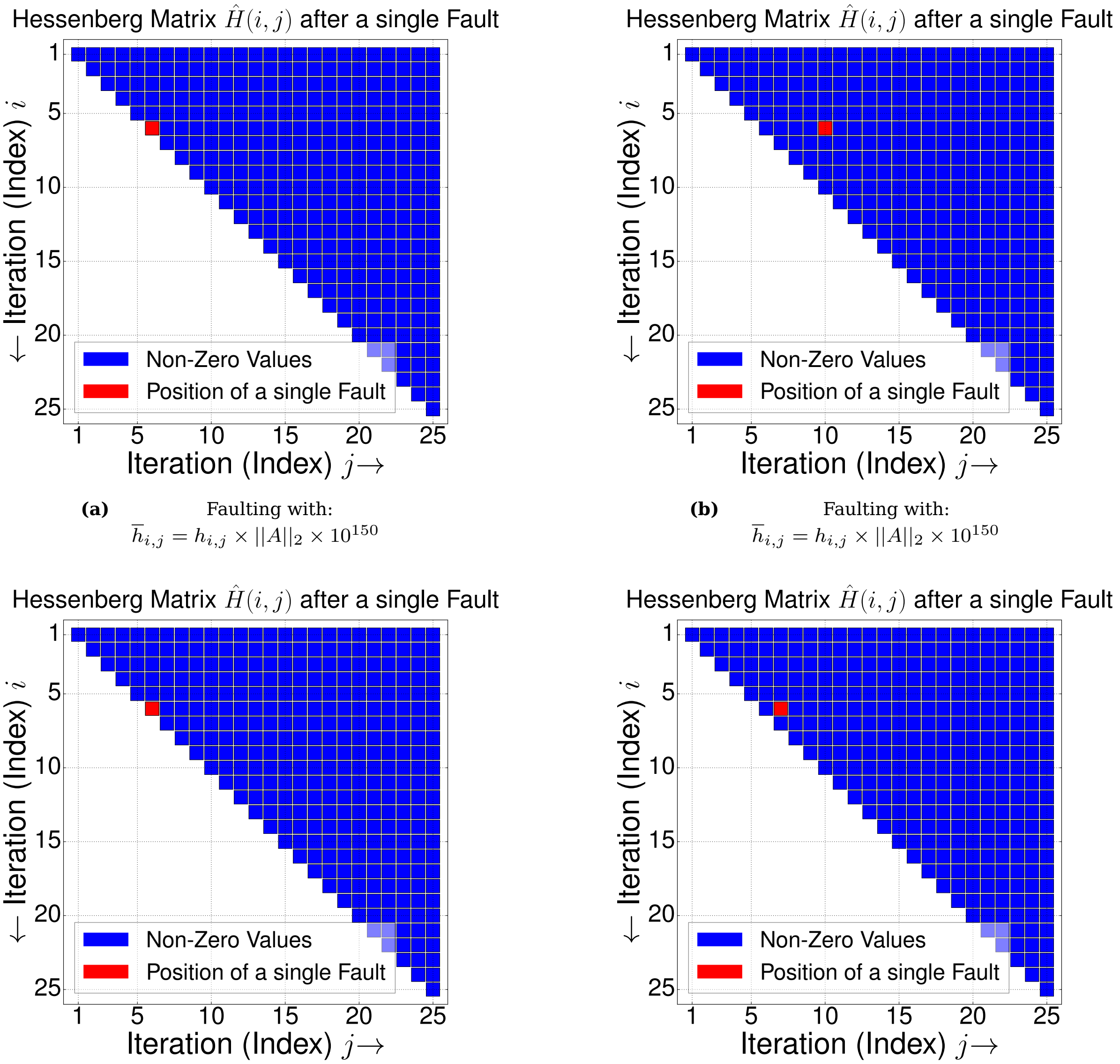
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

Figure 127 Different disturbed Hessenberg matrices (inner solver) for solving the circuit_2 problem after a single fault on a specific position ($h_{i,j}$) in the inner solver of the FT-GMRES.

In figure 127 final Hessenberg matrices after 25 iterations (m_2) for a single fault on a specific position ($h_{i,j}$) during solving the circuit_2 matrix are shown. In the case of this un-symmetric problem the inner solver of the FT-GMRES also builds a Hessenberg matrix which is at least an upper Hessenberg matrix and therefore like in figure 121. Non-zero values which absolute magnitudes are greater than 10^{-9} are visualized with a blue color in figure 127 whereas the red color shows the position for a single fault of $h_{i,j}$. There is an oblivious change of the structure of the Hessenberg matrix because of a single fault, this deviation can be used for fault detection.

9.12.8 Different disturbed Hessenberg matrices for solving the mult_dcop_03 problem



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

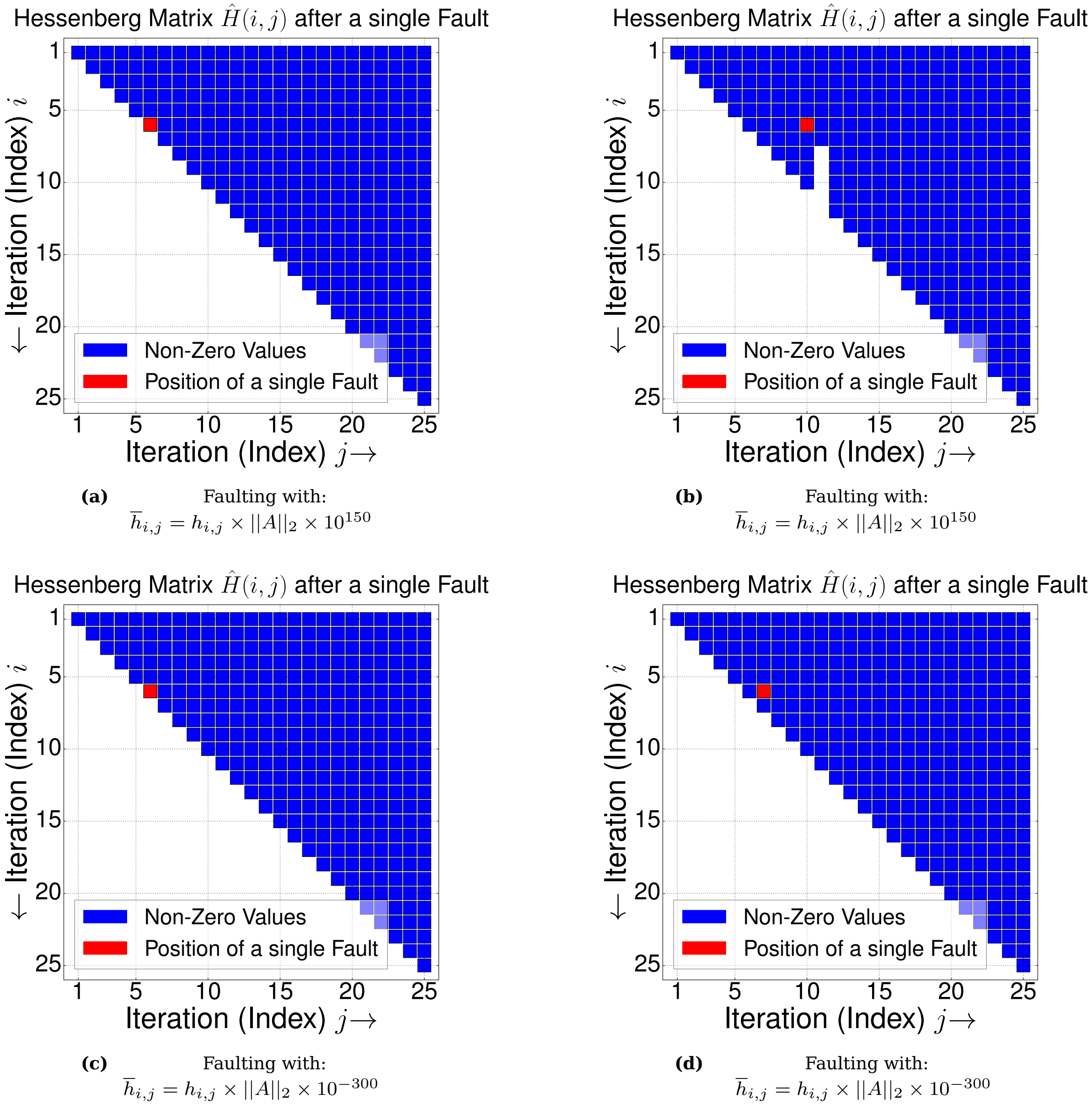
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

Figure 128 Different disturbed Hessenberg matrices (inner solver) for solving the mult_dcop_03 problem after a single fault on a specific position ($h_{i,j}$) in the inner solver of the FT-GMRES.

In figure 128 final Hessenberg matrices after 25 iterations (m_2) for a single fault on a specific position ($h_{i,j}$) during solving the mult_dcop_03 problem are shown. In the case of this un-symmetric problem the inner solver of the FT-GMRES also builds a Hessenberg matrix which is at least an upper Hessenberg matrix and therefore like in figure 121. Non-zero values which absolute magnitudes are greater than 10^{-9} are visualized with a blue color in figure 128 whereas the red color shows the position for a single fault of $h_{i,j}$. There is no change of the structure of the Hessenberg matrix because of a single fault therefore no fault can be detected in this case.

9.12.9 Different disturbed Hessenberg matrices for solving the chem_master1 problem



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

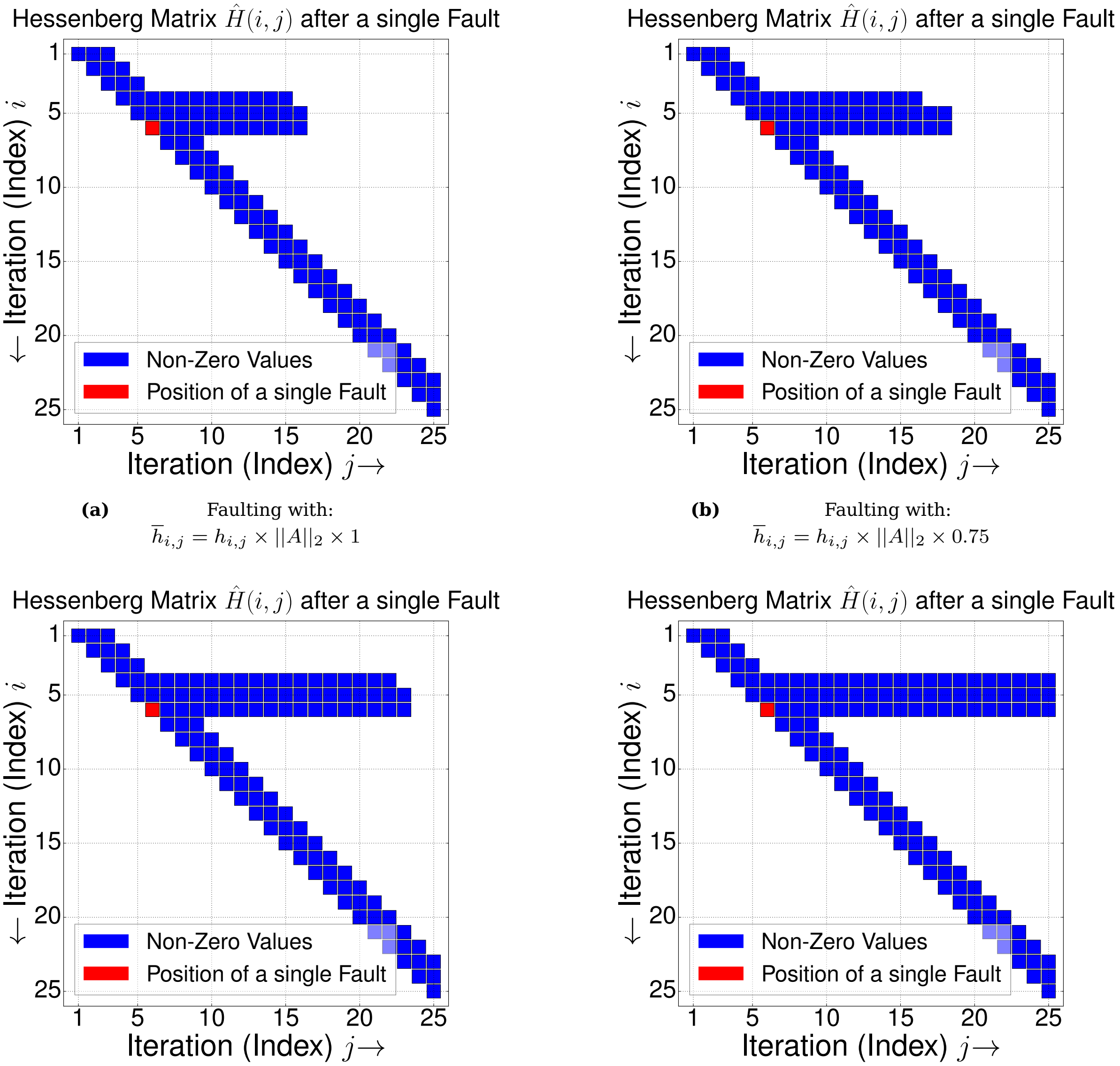
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

Figure 129 Different disturbed Hessenberg matrices (inner solver) for solving the chem_master1 problem after a single fault on a specific position ($h_{i,j}$) in the inner solver of the FT-GMRES.

In figure 129 final Hessenberg matrices after 25 iterations (m_2) for a single fault on a specific position ($h_{i,j}$) during solving the chem_master1 problem are shown. In the case of this un-symmetric problem the inner solver of the FT-GMRES also builds a Hessenberg matrix which is at least an upper Hessenberg matrix and therefore like in figure 121. Non-zero values which absolute magnitudes are greater than 10^{-9} are visualized with a blue color in figure 129 whereas the red color shows the position for a single fault of $h_{i,j}$. There is an oblivious change of the structure of the Hessengerg matrix because of a single fault, this deviation can be used for fault detection.

9.12.10 Sensibility of banded matrices in the presence of a fault (2D Poisson)



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 1$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 0.75$

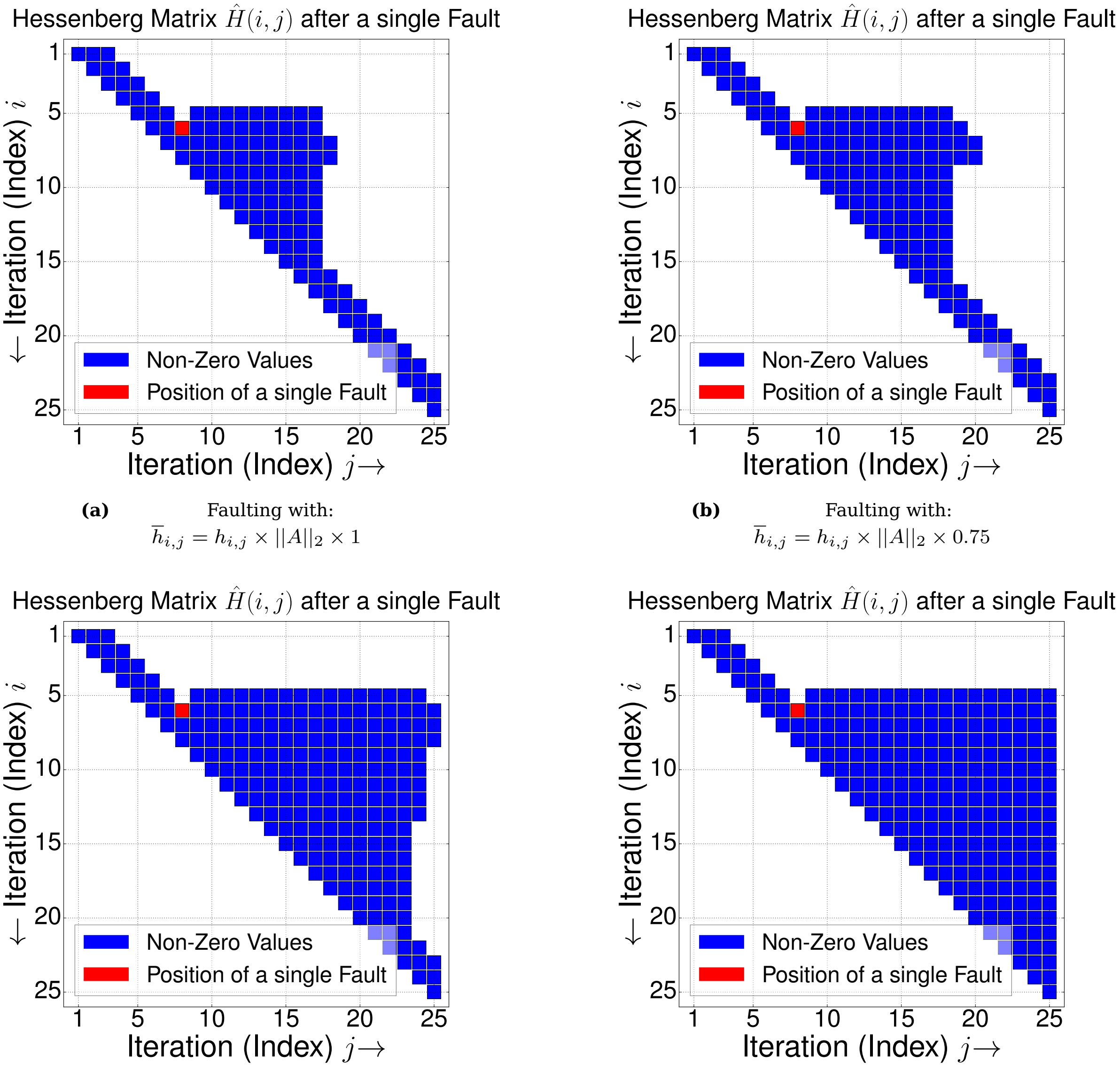
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 0.5$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 0.25$

Figure 130 Different disturbed Hessenberg matrices (inner solver) for solving the 2D Poisson problem after a single fault on a specific position ($h_{i,j}$) in the inner solver of the FT-GMRES.

In figure 130 final Hessenberg matrices after 25 iterations (m_2) for a single fault on a specific position ($h_{i,j}$) during solving the 2D Poisson matrix are shown. In the case of this symmetric problem the inner solver of the FT-GMRES also builds a Hessenberg matrix which is at least an upper Hessenberg matrix but banded (tridiagonal) like in figure 120. Non-zero values which absolute magnitudes are greater than 10^{-9} are visualized with a blue color in figure 130 whereas the red color shows the position for a single fault of $h_{i,j}$. There is an oblivious change of the structure of the Hessenberg matrix because of a single fault, this deviation can be used for fault detection. Figure 130 shows that even small disturbances can change the structure in the presence of a fault.

9.12.11 Sensibility of banded matrices in the presence of a fault (2D Poisson)



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 1$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 0.75$

(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 0.5$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 0.25$

Figure 131 Different disturbed Hessenberg matrices (inner solver) for solving the 2D Poisson problem after a single fault on a specific position ($h_{i,j}$) in the inner solver of the FT-GMRES.

In figure 131 final Hessenberg matrices after 25 iterations (m_2) for a single fault on a specific position ($h_{i,j}$) during solving the 2D Poisson matrix are shown. In the case of this symmetric problem the inner solver of the FT-GMRES also builds a Hessenberg matrix which is at least an upper Hessenberg matrix but banded (tridiagonal) like in figure 120. Non-zero values which absolute magnitudes are greater than 10^{-9} are visualized with a blue color in figure 131 whereas the red color shows the position for a single fault of $h_{i,j}$. There is an oblivious change of the structure of the Hessenberg matrix because of a single fault, this deviation can be used for fault detection. Figure 131 shows that even small disturbances can change the structure in the presence of a fault.

10 Improvements and recommendations for Fault Tolerant GMRES (FT-GMRES)

10.1 Introduction

This section is mainly dedicated to the FT-GMRES (see section 6.2.4) and how to detect and correct faults in the inner solver of the FT-GMRES (line 4 of algorithm 14). One obvious disadvantage is if faults are undetected in the FT-GMRES then these will perhaps lead to a wrong solution in the inner solver where those uncorrected faults may also increase the number of (outer) iterations for the outer solver (m_1). Furthermore because of the poor preconditioning of the inner solver the converge speed for computing the solution might degenerates for solving $Ax = b$. Degeneration leads to poor preconditioning and to more outer iterations but as well to more operations of the used solver which is mainly a slow down of the computation speed.

Invariants are variables or conditions that do not change their properties independently of the computation step which give the possibility to detect specific faults in the inner solver these invariants can be also used for correcting some faults. Invariants can be also seen as properties which are don't allowed to change and are independent. Some invariants which are known so far are shown in table 7. These give the possibility to detect some specific faults where some of them are really cheap to compute but others can not be computed efficiently. So if the condition ($\leq, \geq$) of one of these invariants is hurt a correction step must be applied to resilient the related fault for a better converge speed and to ensure correct results in the inner solver of the FT-GMRES.

One better approach would be to bound errors such that the influence of a fault is totally negated. There is always the problem if the number of operations is it too high to detect a specific fault then there is no real decrease in terms of floating point operations but as well a reduce of the energy requirements. Another question rises if it is really worth to correct every fault since the overhead through additional operations could be too high which should be avoided.

Description	Invariant	Correction step
Lose of orthogonality. (Not applicable)	$\|\|I - q_j^T q_j\|\|_2 \leq \epsilon_{Ortho}$ [48]	Re-orthogonalize $H(:, j)$.
Values of $\hat{H}(i, j)$. ("Applicable")	$h_{i,j} \leq \|\|A\|\|_2$ [4]	Re-orthogonalize $H(:, j)$.
Explicit residuum. (Not applicable)	$\|\|r_j\|\|_2 = \|\|b - Ax_j\|\|_2$ $(\|\|r_j\|\| \leq \|\|r_{j-1}\|\|)$	Restarting the inner solver. // (Continue with the outer solver.)
Implicit residuum. (Not applicable)	$\|\|r_j\|\|_2 = \|\|\hat{H}(i : j + 1, 1 : j)y - \beta e_1\|\|_2$ $\|\|r_j\|\|_2 \leq \|\|r_{j-1}\|\|_2$	Restarting the inner solver. // (Continue with the outer solver.)
Relative change of $\|\|r\|\|_2$. (Applicable)	$1 - \frac{\|\|\sigma_j\|\|_2^2}{\|\|\sigma_{j-1}\|\|_2^2} \geq \delta_l$ [7] $\|\|r_j\|\|_2, \|\|r_{j-1}\|\|_2 \approx \|\|\sigma_j\|\|_2^2, \|\|\sigma_{j-1}\|\|_2^2$	Restarting the inner solver. // (Continue with the outer solver.)
Relative change of $\|\|r\|\|_2$. (Applicable)	$1 - \frac{\|\|\sigma_j\|\|_2^2}{\|\|\sigma_0\|\|_2^2} \leq \delta_g$ [7] $\|\|r_0\|\|_2, \|\|r_j\|\|_2 \approx \|\|\sigma_0\|\|_2^2, \|\|\sigma_j\|\|_2^2$	Continue with the inner solver. // Continue with the outer solver.

Table 7 Possible methods to detect and correct faults in the inner solver of the FT-GMRES those which are known so far (from the authors view). Applicable means in terms of floating point operations, if the according approach is cheap to apply or not in the sense of flops.

The first possibility would be to check the lose of orthogonality [48] in the GMRES algorithm (see algorithm 6) of vector q_j because it must be satisfied that $q_j^T q_j = 1$ which is not applicable to detect faults caused by some bit-flips during computing the Hessenberg matrix. The main reason is if the vector q_j is faulty it is still possible that this vector is orthogonal to itself or to other vectors. Furthermore finding a good value for ϵ_{Ortho} is hard or the algorithm will always re-orthogonalize even when no fault is occurred. Therefore checking the lose of orthogonality is not applicable for detecting faults in the inner solve of the FT-GMRES.

A good way to detect large faults in the Hessenberg matrix $\hat{H}$ is using the expression of $h_{j+1,j} \leq ||A||_2$ which makes detecting large perturbations (e.g. 10^{10}, 10^{150}) really easy during the orthogonalization process for computing values for $h_{i,j}$ (line 5 - 10 of algorithm 6). If one of these values ($h_{i,j}$) of the Hessenberg matrix $\hat{H}$ is greater than the norm ($||.||_2$) of the used matrix A then a fault occurred during the orthogonalization process ($h_{i,j}$). This kind of error detecting is mainly based on applying a norm for the matrix vector product of $v_{j+1} = Aq_j$ which is performed in the GMRES solver (line 4 of algorithm 6). After applying a norm ($||.||_2$) and knowing the fact that $||q_j||_2 = 1$ (in the case of the GMRES) it follows that $||v_{j+1}||_2 \leq ||A||_2$. From that point it is obvious that $h_{j+1,j} = ||v_{j+1}||_2$ [4] and with substituting of the inequality this formula is transformed to $h_{j+1,j} \leq ||A||_2$ then this also leads to $h_{i,j} \leq ||A||_2$ because this condition must be satisfied for each value of the Hessenberg matrix.

Computing the norm of $||.||_2$ is in general not applicable for large matrices therefore another norm must be used. If a fault is detected re-orthogonalization must be applied which means that all values of matrix $\hat{H}(:,j)$ must be recomputed (all rows of matrix $\hat{H}(:,j)$ at current column position j) at the current column position and iteration j before the Givens rotation is applied. The norm of $||.||_2$ for a matrix A can be computed approximately with $||.||_2 \leq \sqrt{n} \times ||.||_\infty$ [6] where n is the size of matrix A whereas $||.||_\infty$ is the absolute greatest element of matrix A.

The next possibility would be to check the residuals of $||r||_2 = ||\hat{H}(i : j + 1, 1 : j)y - \beta e_1||_2$ or $||r||_2 = ||Ax - b||_2$ with comparing two residuals between two iterations ($||r_j||_2 \leq ||r_{j-1}||_2$). If an increase of the current residual ($||r_j||$) against the previous ($||r_{j-1}||$) is detected the inner solver should be restarted or with the outer solver can be continued. Continue with the outer solver is not advisable because this will lead to more iterations for the outer solver because of poor preconditioning of the inner solver. This approach is in general not applicable because it needs too much operations for the matrix vector products of $\hat{H}(i : j + 1, 1 : j)y$ and Ax. Computing the residuals ($||r_j||_2$) for each iteration j makes the convergence behavior of the solution vector x observable but this approach cannot be used for error detecting in general because of the fact that the GMRES algorithm will always try to minimize the implicit ($||r||_2 = ||\hat{H}(i : j + 1, 1 : j)y - \beta e_1||_2$) as well the explicit residual ($||r||_2 = ||Ax - b||$) even when the Hessenberg matrix is not correct because of a fault. Furthermore in the case of the implicit residual the value of β is always fixed to the normalized value of $||r_0||_2$ (the starting residuum) which leads to the next reason that using the residuals is not applicable in general.

The approximation error $||\sigma||_2^2$ which is computed in algorithm 5 during applying the Givens rotation is approximately the same like the residuum such that $||\sigma||_2^2 \approx ||r||_2$ and can be used to build an indicator to detect a fault in the inner solver of the FT-GMRES. So the value of $||\sigma||_2^2$ can be used instead of the explicit ($||r||_2 = ||Ax - b||_2$) and implicit residuum ($||r||_2 = ||\hat{H}(i : j + 1, 1 : j)y - \beta e_1||_2$) because of a similar behavior in terms of convergence and magnitude like the residuum ($||r||_2$). In general it helps to decrease the number of floating point operations a lot because computing the residuum $||r||_2$ will always need an additional sparse vector product (Ax or Hy). How to compute the value of $||\sigma||_2^2$ is shown in algorithm 5 with applying the Givens rotation.

With using the approximation errors ($||\sigma||_2^2$) it is possible to detect stagnating convergence in the inner solver of the FT-GMRES between two iterations j and $j - 1$ if two values $||\sigma_{j-1}||_2^2$ and $||\sigma_j||_2^2$ are nearly the same ($||\sigma_{j-1}||_2^2 \approx ||\sigma_j||_2^2$) such that the solution is not improving anymore this fact can be used to build an additional error detector δ_l. It also means that $\frac{||\sigma_j||_2^2}{||\sigma_{j-1}||_2^2} \approx 1$ if both values are the same and the solution is not converging to a desired value anymore in the inner solver of the FT-GMRES because the current value of $||\sigma_j||_2^2$ and previous computed $||\sigma_{j-1}||_2^2$ are nearly the same such that there is no relevant change between two iterations. So the expression $\frac{||\sigma_j||_2^2}{||\sigma_{j-1}||_2^2} - 1 = \delta_l$ checks if $||\sigma_j||_2^2$ and $||\sigma_{j-1}||_2^2$ are nearly the same such that $\frac{||\sigma_j||_2^2}{||\sigma_{j-1}||_2^2} \approx 1$ and with the given threshold value δ_l. Linear dependency follows if $\delta_l = 0$ whereas $\delta_l = 1$ means that the last and previous computed residuum are still the same such that no residual reduction is done.

The previous computed value of $||\sigma_{j-1}||_2^2 \approx ||r_{j-1}||_2$ and the current value of $||\sigma_j||_2^2 \approx ||r_j||_2$ can be used as a detector to check if the convergence speed in the inner solver is stagnating. This method of how to detect stagnating convergence is described in [7]. So if a fault is detected in the inner solver, this solver can be restarted but by also reusing the current solution computed so far for the outer solver which should be used as the new start solution of $w_j(w_0)$, for the next execution of the inner solver. For example if a fault is detected at iteration j in the inner solver and there are a certain number iterations at all then the last remaining iterations are used for the next execution of the inner solver which is done in this master work. The main problem is to find a reasonable value for δ_l to check for equality of $||\sigma_j||_2^2$ and $||\sigma_{j-1}||_2^2$. In general the value of δ_l can be chosen arbitrary which is just an scalar. If those eigenvalues (λ) of the used matrix A are known perhaps a better value can be found for the value of δ_l. The following equation is related to the converge behavior (residual reduction) of the GMRES solver for positive definite matrices:

$$\|r_m\|_2 \leq \left(1 - \frac{\lambda_{\min}^2(1/2(A^T + A))}{\lambda_{\max}(A^T A)}\right)^{m/2} \|r_0\|_2 \text{ [5][3][47]} \qquad (111)$$

, where r_m denotes the current residuum at iteration m and r_0 as the starting residuum, λ_{min} and λ_{max} are the smallest and greatest eigenvalue (λ) of the used matrix A. This approximation for the residual reduction needs a transpose of the used matrix A and a sparse matrix multiplication but also some additions which is not applicable for large matrices because of too much floating point operations. This formula also says that the residuum ($\|r_m\|_2$) always decreases monotonic with a certain factor. On the other side it should be also possible to detect stagnating convergence in the inner solver because the residuum ($||r_m||_2$) must decrease monotonic for a certain factor and a specific number of iterations which is shown in formula 111. The main problem is that the eigenvalues (λ) of matrix A are not known in general, a better approach would be to use formula 112 with the unknown constant ϵ this is only a factor which determines how much the residuum $\|r_0\|_2$ respective $||\sigma_0||_2^2$ should be reduced after m iterations with using the GMRES solver.

$$\|r_m\|_2 \leq \epsilon \times \|r_0\|_2 \qquad (112)$$

The value of ϵ mainly depends on the number of iterations and the according convergence rate of the used solver, therefore this value (ϵ) is only valid for m iterations and can be computed with the help of $||r_m||_2$ and $||r_0||_2$. There are maybe some in-accuratenesses which lead to the problem that in some cases the relation in formula 112 is not satisfied even when the convergence rate is not decreased because of a to "rough" bound. This formula 112 is just similar to formula 111 but by knowing nothing about the matrix A and the according eigenvalues (λ) of matrix A.

So a possibility of how to determine the value of δ_l is shown in section 10.2 where some iterations are done in a failure free execution of the GMRES (F-GMRES) solver to compute the value of δ_l. The main idea is just that value of ϵ should be determined by the solver itself because there must be a similar residual reduction for each execution of the inner solver of the F-GMRES. From this point of view it should be perhaps also possible to compute and reuse the value of δ_l in the same way and if the value of δ_l is lower than the computed one then it should indicates a fault.

10.1.1 Faulting without any improvements on a diagonal matrix with high condition number - Inner solver initialized with zero values

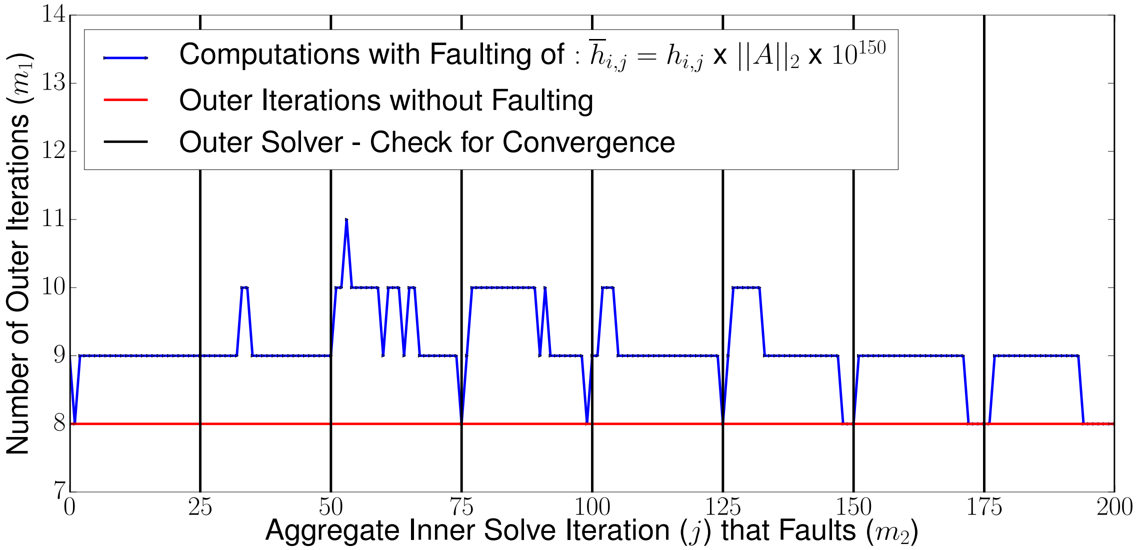


Figure 132 Faulting with error type 1 on a diagonal matrix with condition number 1425 on the first MGS ($h_{1,j}$). There are no improvements done in the case of the FT-GMRES for fault detection.

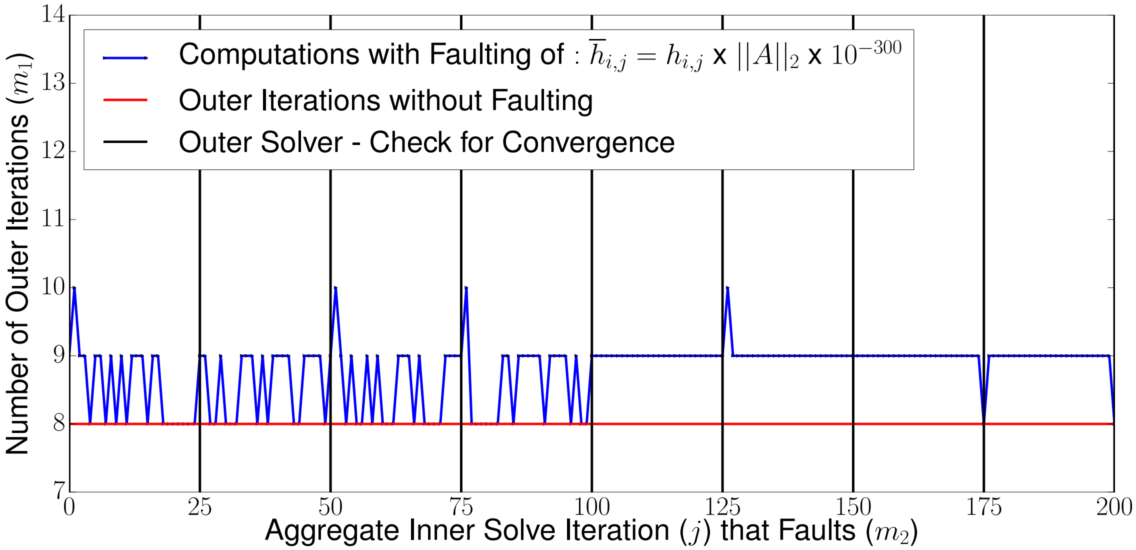


Figure 133 Faulting with error type 2 on a diagonal matrix with condition number 1425 on the first MGS ($h_{1,j}$). There are no improvements done in the case of the FT-GMRES for fault detection.

In figure 132 and 133 there are no improvements done for the inner solver of the FT-GMRES. The used matrix is built like in section 7.1 for problem 1 (see formula 102). This diagonal matrix D has 1000 entries where the lowest value (d_1) is set to 1.0 and the greatest (d_n) to 1425 which leads to a matrix with condition number 1425. In both cases the inner solver of the FT-GMRES is initialized with zero values for vector $w_j(w_0)$. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all computations and trials. In this case the worst overhead is about 22 % (11/9).

10.1.2 Faulting on a diagonal matrix with high condition number and using the norm ($h_{i,j} \leq ||A||_2$) - Inner solver initialized with zero values

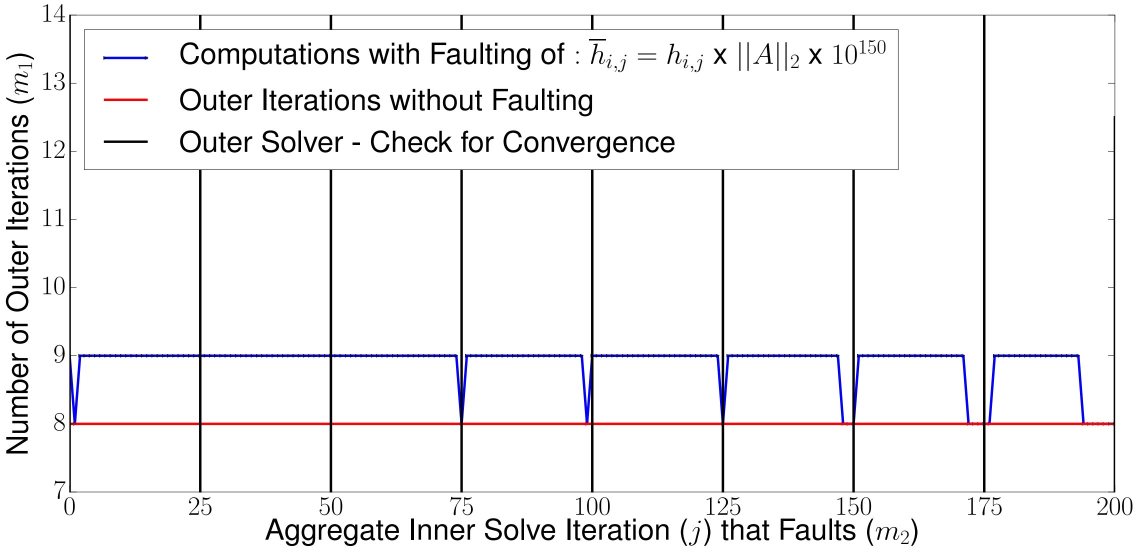


Figure 134 Faulting with error type 1 on a diagonal matrix with condition number 1425 on the first MGS ($h_{1,j}$) and using the norm of matrix A, it is possible to detect perturbations like 10^{150}.

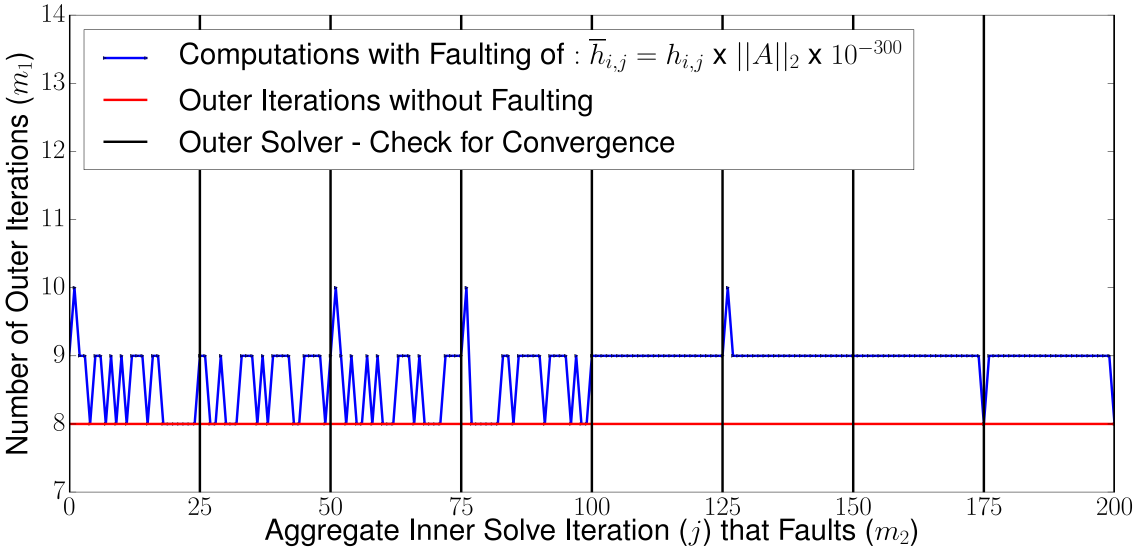


Figure 135 Faulting with error type 2 on a diagonal matrix with condition number 1425 on the first MGS ($h_{1,j}$) and using the norm of matrix A, perturbations like 10^{-300} cannot be detected.

In figure 134 the norm of matrix A is used to detect faults during the orthogonalization process in the inner solver of the FT-GMRES. If the condition $h_{i,j} \leq ||A||_2$ is hurt re-orthogonalization is applied which means that all values of the Hessenberg matrix at current iteration j must be re-computed ($\hat{H}(:,j)$). It is not possible to detect and correct perturbations like 10^{-300} which is shown in figure 135. The used matrix D is built like in section 7.1 for problem 1 (see formula 102). In both cases the inner solver of the FT-GMRES is initialized with zero values for vector $w_j(w_0)$.

10.1.3 Faulting on a diagonal matrix with high condition number and using the relative change (δ_l) - Inner solver initialized with zero values

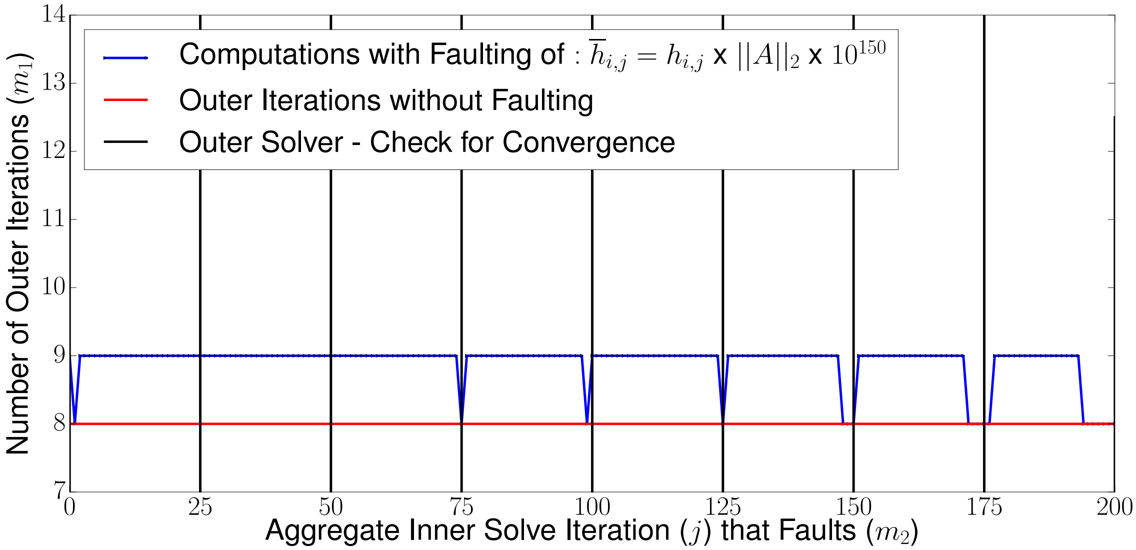


Figure 136 Faulting with error type 1 on a diagonal matrix with condition number 1425 on the first MGS ($h_{1,j}$). In this case all faults are detected in the inner solver of the FT-GMRES.

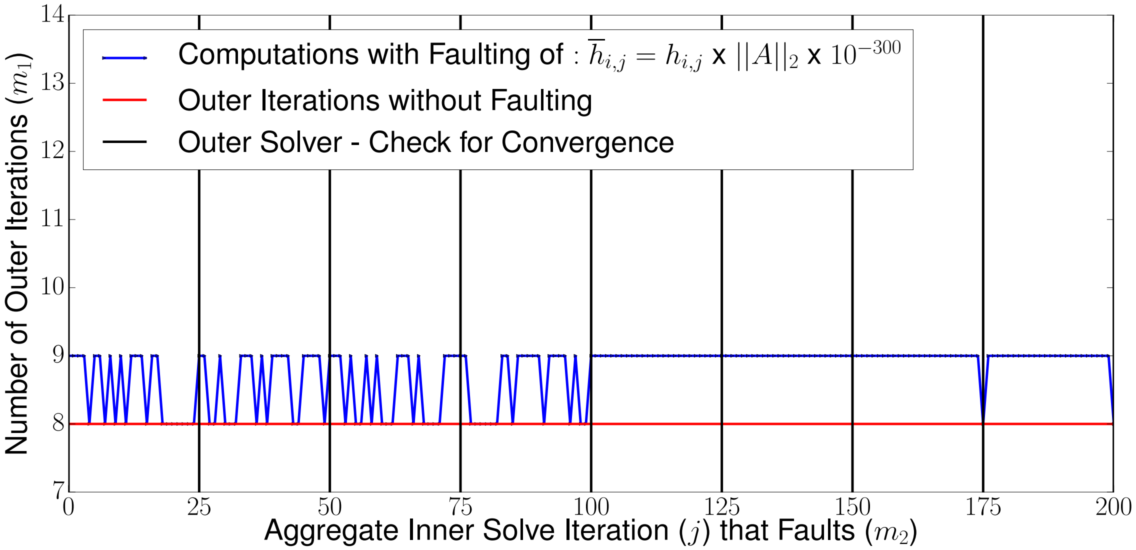


Figure 137 Faulting with error type 2 on a diagonal matrix with condition number 1425 on the first MGS ($h_{1,j}$). In this case all faults are detected in the inner solver of the FT-GMRES.

The relative change δ_l can be used to correct perturbations like 10^{150} and 10^{-300} which is shown in figure 136 and 137. After a fault is detected restarting of the inner solver is applied with the last remaining iterations and with the current solution w_j. The value of δ_l is set to $10^{-1,5}$ in this case for fault detection. The used matrix D is built like in section 7.1 for problem 1 (see formula 103). In both cases the inner solver of the FT-GMRES is initialized with zero values for vector $w_j(w_0)$. A similar behavior can be found in section 10.2.8 and 10.2.9 in the case of the 2D Poisson problem.

10.2 Fault detection through the relative change between two residuals

10.2.1 Introduction

One further possibility to detect faults in the inner solver of the FT-GMRES is to observe the relative change between two residuals with δ_l like mentioned in section 10.1 but there are some problems which have to be solved before. In general the eigenvalues (λ) of the used matrix A are not known and the value of δ_l for fault detection (see section 10.1) is more or less related to the eigenvalues (λ) of the used matrix A. One possibility would be for estimating this value δ_l that the GMRES or Flexible GMRES can do some iterations in a failure free run without doing too much iterations then this computed value δ_l can be maybe used afterwards to detect some faults during solving the problem of $Ax = b$ with the Fault Tolerant GMRES (FT-GMRES). This is only a possibility at the moment and not a general approach to solve $Ax = b$ and must be proven. The value of δ_l which is computed with $1 - \frac{||\sigma_j||_2^2}{||\sigma_{j-1}||_2^2} = \delta_{l(j)}$ can be used in different ways, one possibility is just to use the lowest value of all computed values of $\delta_{l(1)}$,$\delta_{l(2)}$,$\delta_{l(3)}$,...,$\delta_{l(m)}$ to get the value for δ_l where restarting has to be done for the inner solver of the FT-GMRES.

There are different ways but some further experiments and approaches have to be done to improve the computation of δ_l for fault detection in the FT-GMRES. In general the use of δ_l will not detect every fault if the convergence is stagnating in the inner solver of the FT-GMRES but this value should help to detect and correct some further faults and can be used with other improvements. It is also possible to reuse the computed solution x of the F-GMRES or GMRES which is computed in a failure free run for the FT-GMRES. This is perhaps a good way to accelerate the solving process for the FT-GMRES and to lower the number of floating point operations because in the standard GMRES solver the number of operations tends to increase for each new iteration through the orthogonalization process. In figure 138 the main concept of how to determine the value of δ_l is shown with reusing the solution vector x, this is more or less an idea at the moment.

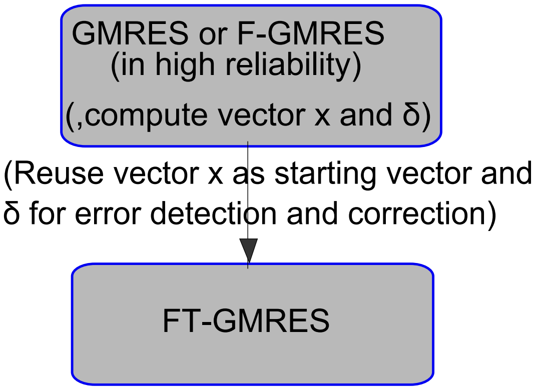

GMRES or F-GMRES
(in high reliability)
(,compute vector x and δ)
(Reuse vector x as starting vector and
δ for error detection and correction)
FT-GMRES

Figure 138 Illustration of a possibility of how to compute the relative change (δ_l) and with also reusing the solution vector x of the F-GMRES or GMRES solver for the FT-GMRES.

For the next sections there are different values of δ_l computed which visualize the relative change between two residuals ($||r||_2$) respective approximation errors ($||\sigma||_2^2$) in the inner solver of the F(T)-GMRES. These plots also show the value of δ_l for different trials if the convergence speed is stagnating in the presence of a single fault in the inner solver of the FT-GMRES. Different perturbations like 10^{-300} and 10^{150} are induced in the inner solver of the FT-GMRES on positions of $h_{1,2}$ during the orthogonalization process for some trials which lead to stagnation of the convergence speed. Furthermore the value of δ_l is also computed for the very first (25) iterations in the case of the GMRES solver, the GMRES solver has always the same initialization as the inner solver.

10.2.2 Computation of the relative change (δ_l) during solving the 2D Poisson problem - GMRES and inner solver of F(T)-GMRES initialized with random values

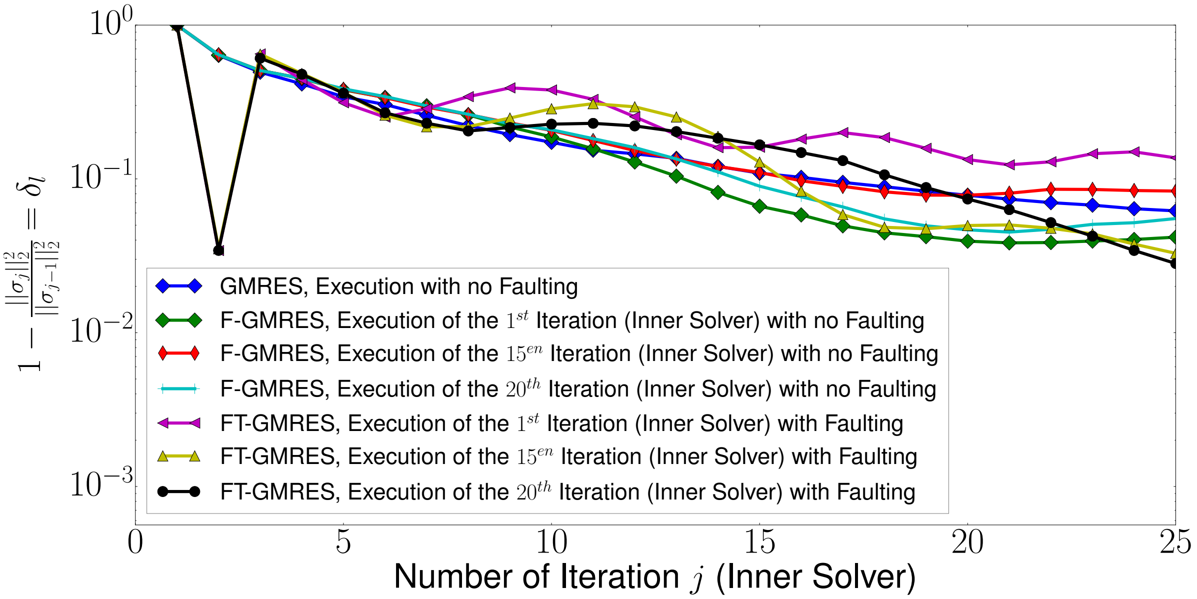


Figure 139 The relative change (δ_l) in the inner solver of F(T)-GMRES during solving the 2D Poisson matrix and faulting (single) with $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$ for $h_{1,2}$.

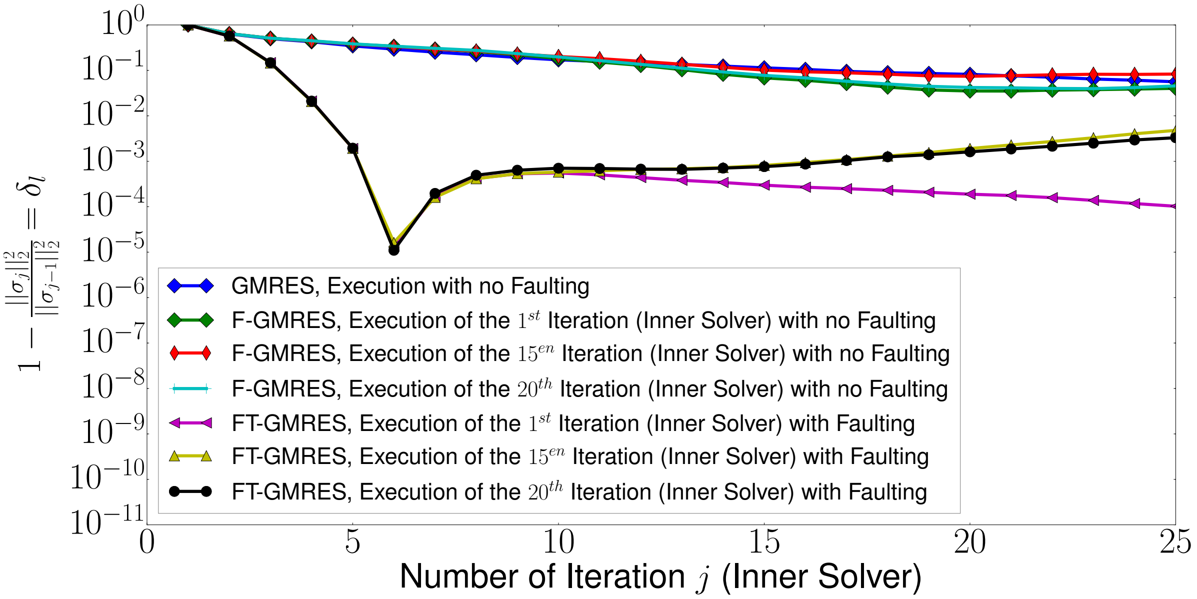


Figure 140 The relative change (δ_l) in the inner solver of F(T)-GMRES during solving the 2D Poisson matrix and faulting (single) with $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$ for $h_{1,2}$

In figure 139 and 140 there are some some indexes (3, 6) where the value of δ_l is lower than without faulting. It indicates that a fault during the execution of the inner solver occurred so restarting should be done. The inner solver of F(T)-GMRES is initialized with random values for vector $w_j(w_0)$. Both figures visualize the relative change (δ_l) in the inner solver for different executions of it. So each plot shows the relative change for different executions of the inner solver.

10.2.3 Computation of the relative change (δ_l) during solving the 2D Poisson problem - GMRES and inner solver of F(T)-GMRES initialized with zero values

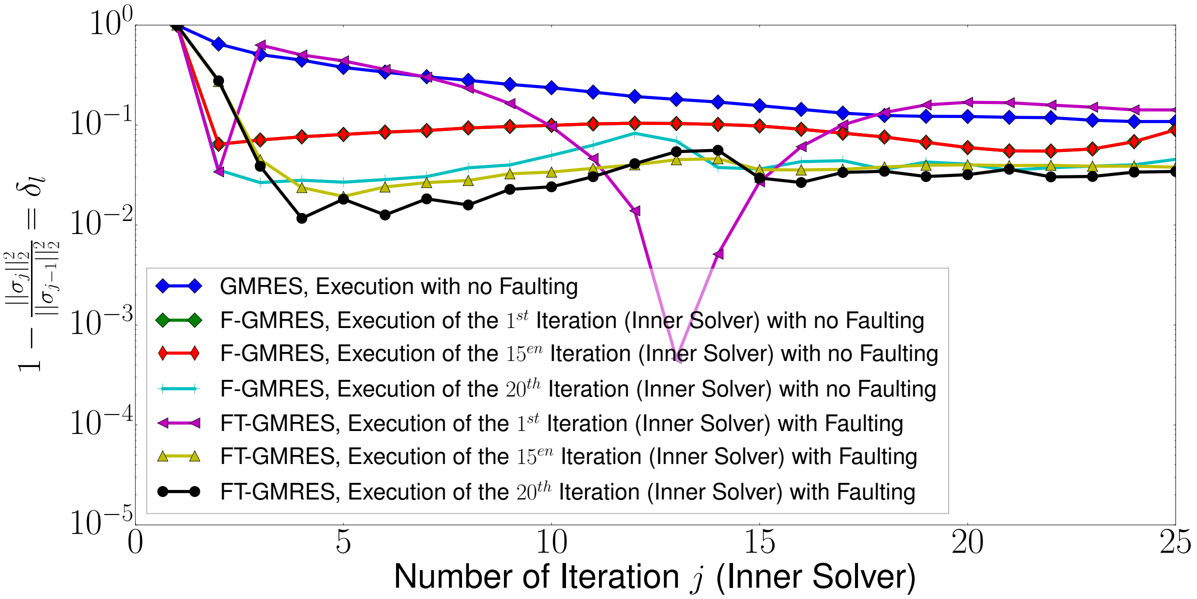


Figure 141 The relative change (δ_l) in the inner solver of F(T)-GMRES during solving the 2D Poisson matrix and faulting (single) with $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$ for $h_{1,2}$.

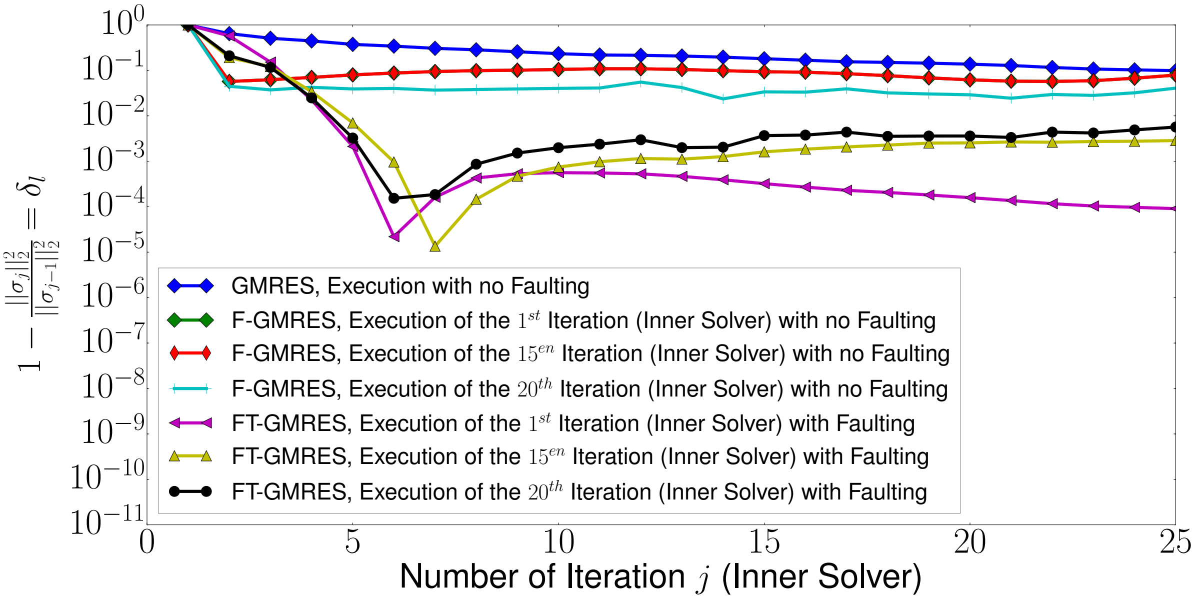


Figure 142 The relative change (δ_l) in the inner solver of F(T)-GMRES during solving the 2D Poisson matrix and faulting (single) with $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$ for $h_{1,2}$.

In figure 141 and 142 there are some indexes (6, 13) where the value of δ_l is lower than without faulting. It indicates that a failure during the execution of the inner solver occurred so restarting should be done. The inner solver of the F(T)-GMRES is initialized with zero values for vector $w_j(w_0)$. Both figures visualize the relative change (δ_l) in the inner solver for different executions of it. So each plot shows the relative change for different executions of the inner solver.

10.2.4 Computation of all relative changes (δ_l) during solving the 2D Poisson problem - GMRES and inner solver of F-GMRES initialized with random and zero values

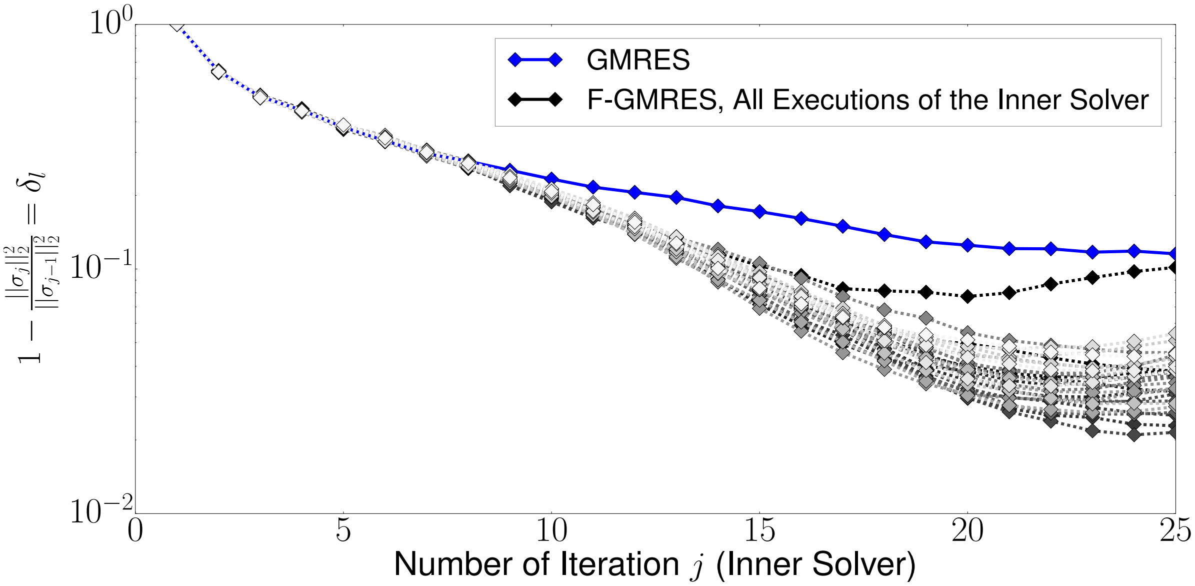


Figure 143 All relative changes (δ_l) in the inner solver during solving the 2D Poisson matrix. The inner solver of the F-GMRES for vector $w_j(w_0)$ is initialized with random values.

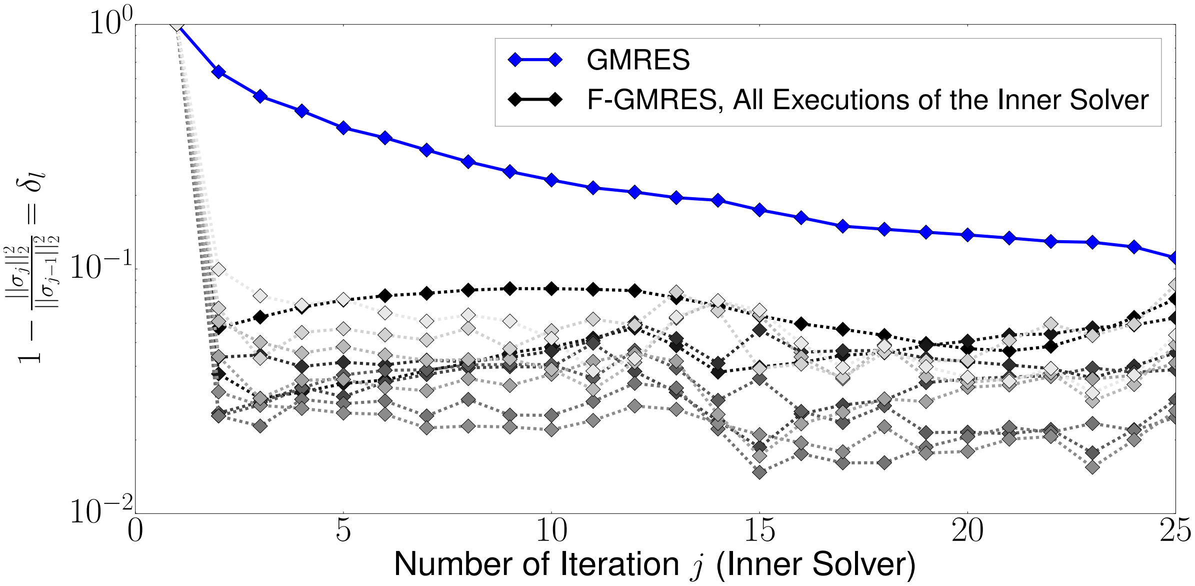


Figure 144 All relative changes (δ_l) in the inner solver during solving the 2D Poisson matrix. The inner solver of the F-GMRES for vector $w_j(w_0)$ is initialized with zero values.

In figure 143 and 144 the value of δ_l is computed for all executions of the inner solver of the F-GMRES (with no faulting), in the case of solving the 2D Poisson matrix. The value of δ_l is also computed for the GMRES solver, for the very first 25 iterations and with the same initialization as the inner solver of the F-GMRES for random values but as well with zero values for vector $w_j(w_0)$. The color goes from black (first outer iteration - first execution of the inner solver) to white (last outer iteration - last execution of the inner solver). The value of δ_l decreases, the solver gets slower.

10.2.5 Computation of the relative change (δ_l) during solving the adder_dcop_63 problem - GMRES and inner solver of F(T)-GMRES initialized with random values

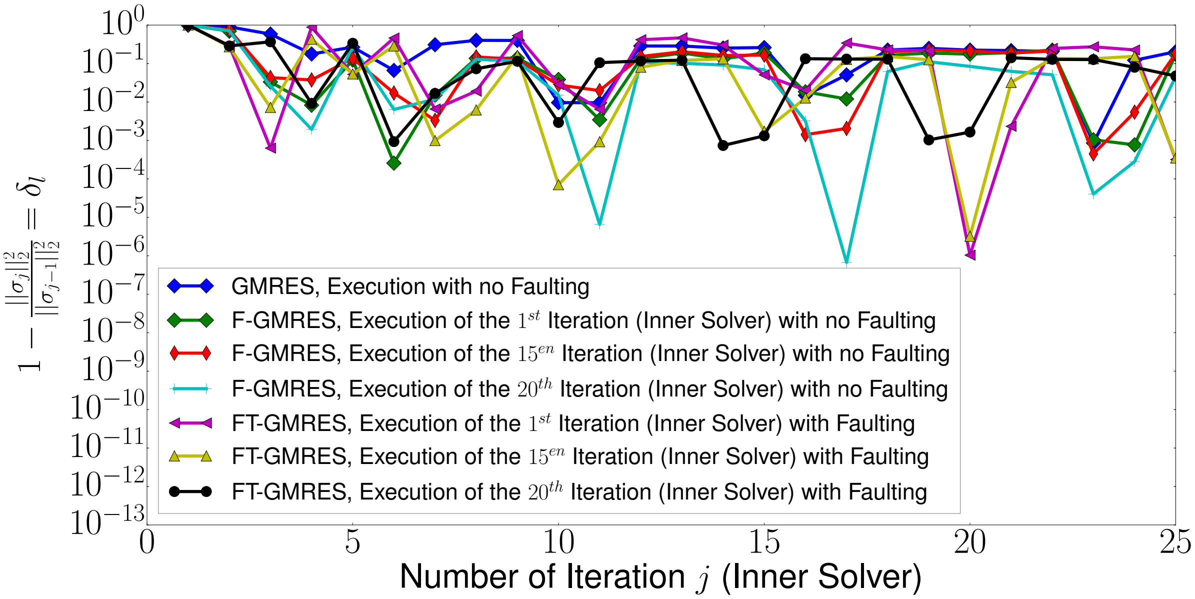


Figure 145 The relative change (δ_l) in the inner solver of F(T)-GMRES during solving the adder_dcop_63 matrix and faulting with $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$ for $h_{1,2}$.

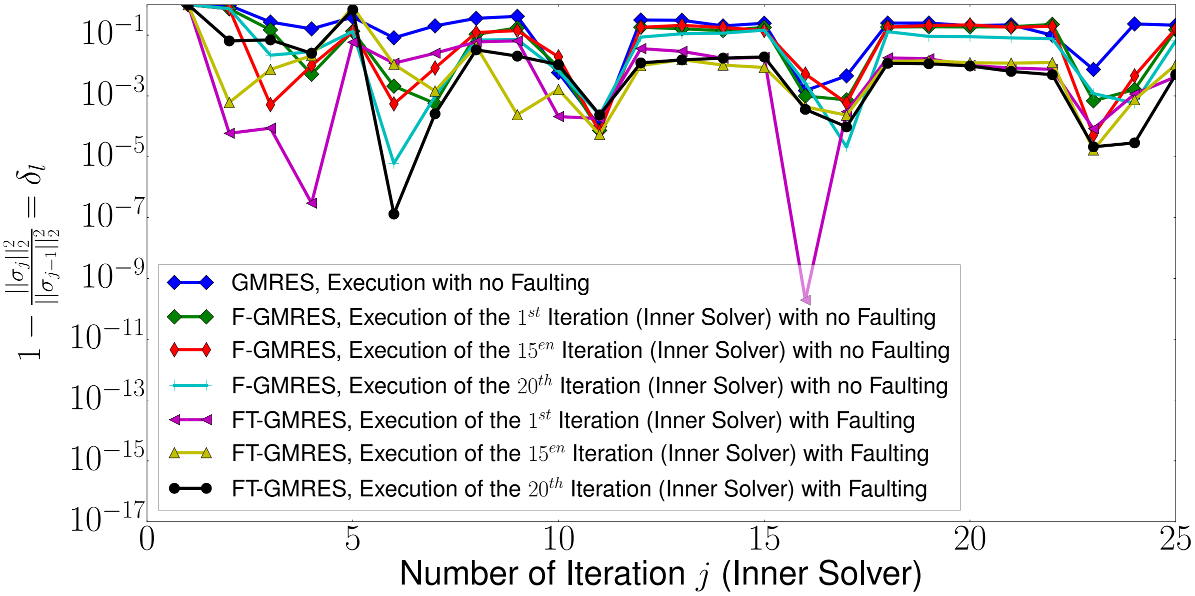


Figure 146 The relative change (δ_l) in the inner solver of F(T)-GMRES during solving the adder_dcop_63 matrix and faulting with $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$ for $h_{1,2}$.

In figure 145 and 146 the adder_dcop_63 matrix is solved which has the ability to tolerate some faults. It is hard to find a value for δ_l where restarting should be done because of the shape for δ_l, maybe it is not needed. The inner solver of F(T)-GMRES is initialized with random values for vector $w_j(w_0)$. Both figures visualize the relative change (δ_l) in the inner solver for different executions of it. So each plot shows the relative change for different executions of the inner solver.

10.2.6 Computation of the relative change (δ_l) during solving the adder_dcop_63 problem - GMRES and inner solver of F(T)-GMRES initialized with zero values

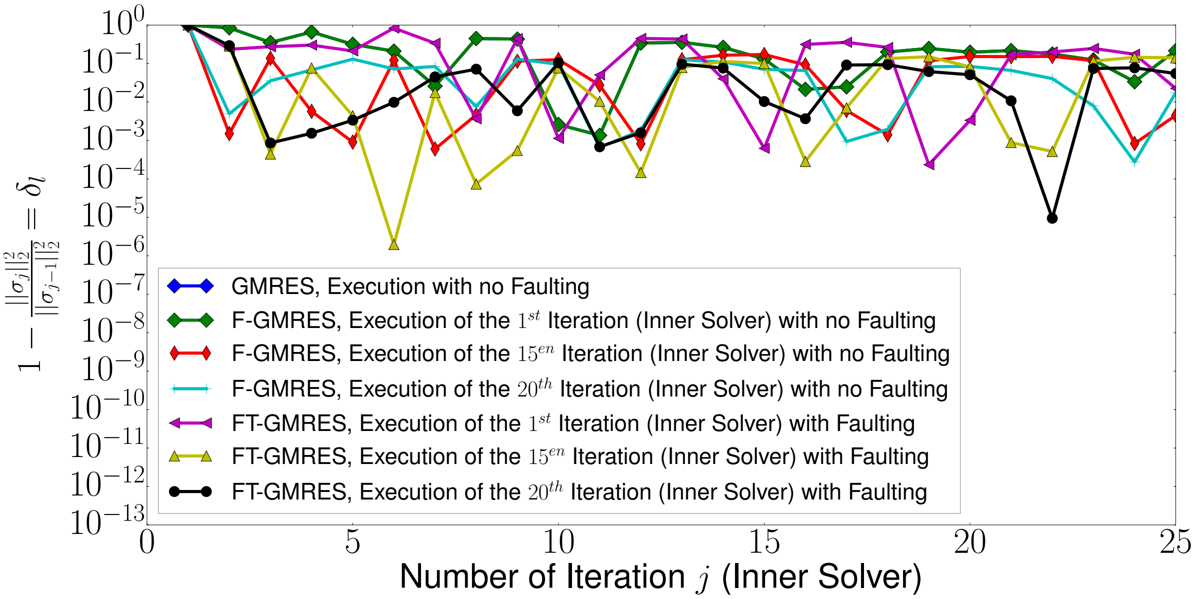


Figure 147 The relative change (δ_l) in the inner solver of F(T)-GMRES during solving the adder_dcop_63 matrix and faulting (single) with $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$ for $h_{1,2}$.

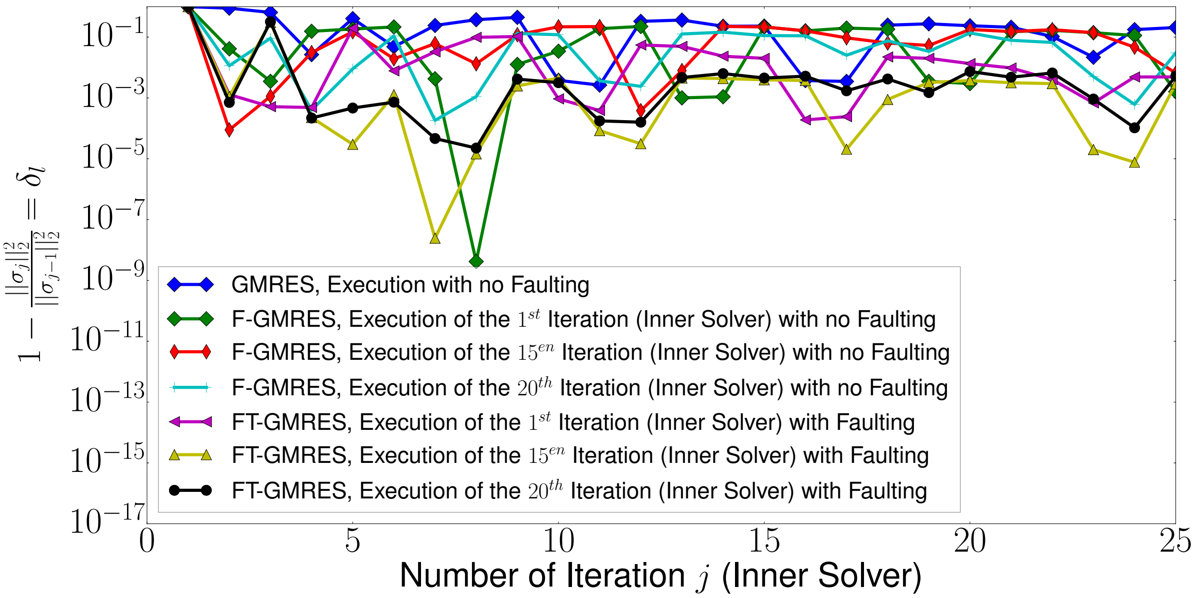


Figure 148 The relative change (δ_l) in the inner solver of F(T)-GMRES during solving the adder_dcop_63 matrix and faulting (single) with $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$ for $h_{1,2}$.

In figure 147 and 148 the adder_dcop_63 matrix is solved which has the ability to tolerate some faults. It is hard to find a value for δ_l where restarting should be done because of the shape for δ_l, maybe it is not needed. The inner solver of F(T)-GMRES is initialized with zero values for vector $w_j(w_0)$. Both figures visualize the relative change (δ_l) in the inner solver for different executions of it. So each plot shows the relative change for different executions of the inner solver.

10.2.7 Computation of all relative changes (δ_l) during solving the adder_dcop_63 problem - GMRES and inner solver of F-GMRES initialized with random/zero values

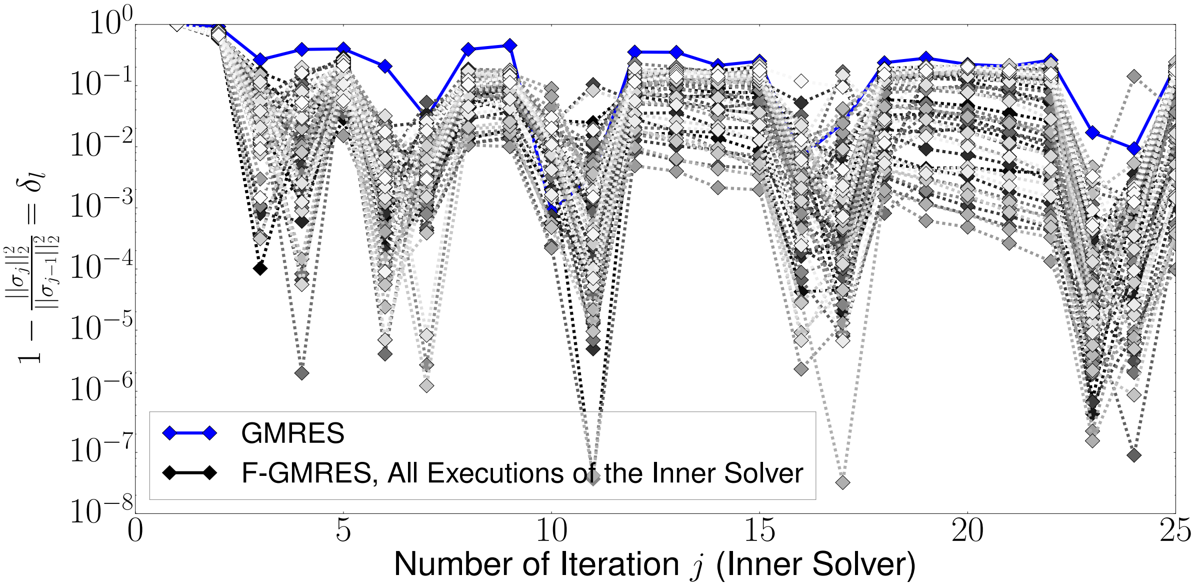


Figure 149 All relative changes (δ_l) in the inner solver during solving the adder_dcop_63 matrix. The inner solver of the F-GMRES for vector $w_j(w_0)$ is initialized with random values.

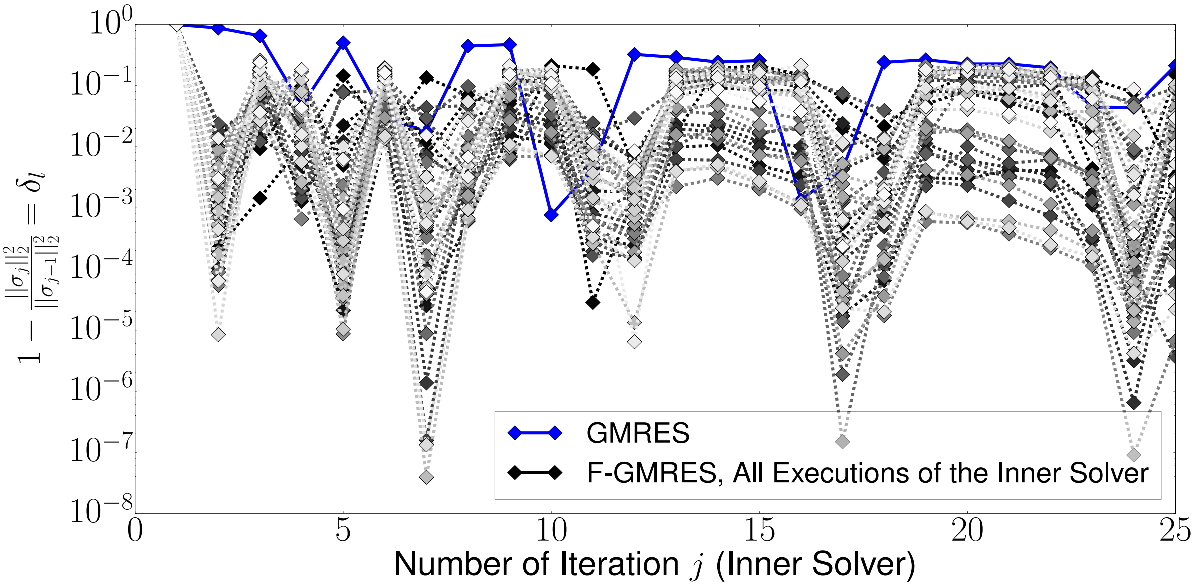
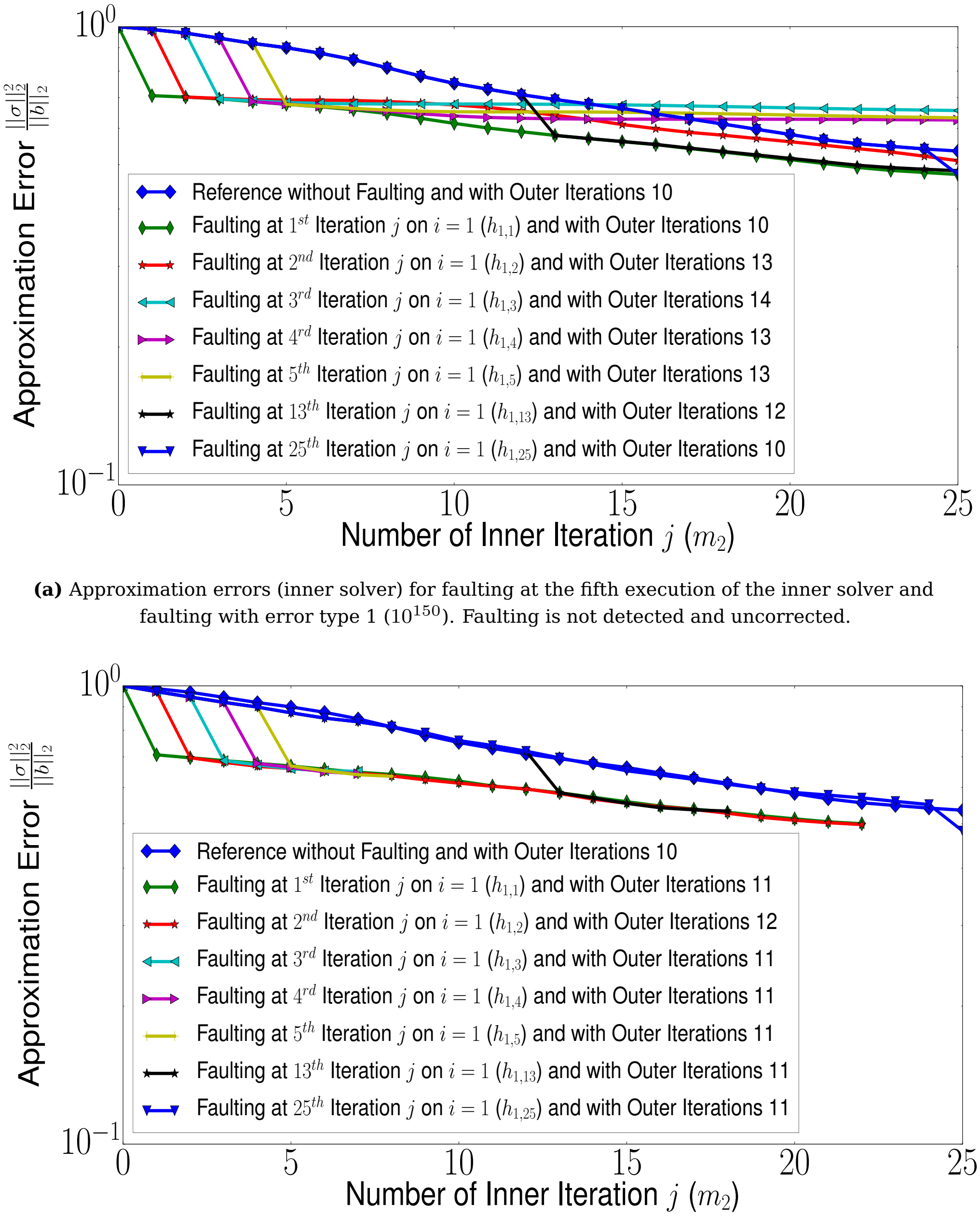


Figure 150 All relative changes (δ_l) in the inner solver during solving the adder_dcop_63 matrix. The inner solver of the F-GMRES for vector $w_j(w_0)$ is initialized with zero values.

In figure 149 and 150 the value of δ_l is computed for all executions of the inner solver of the F-GMRES (with no faulting), in the case of solving the adder_dcop_63 matrix. The value of δ_l is also computed for the GMRES solver, for the very first 25 iterations and with the same initialization as the inner solver of the F-GMRES for random values but as well with zero values for vector $w_j(w_0)$. The color goes from black (first outer iteration - first execution of the inner solver) to white (last outer iteration - last execution of the inner solver). The value of δ_l decreases, the solver gets slower.

10.2.8 Approximation errors with/without correcting stagnating convergence for solving the 2D Poisson matrix (error type 1) - Inner solver initialized with zero values



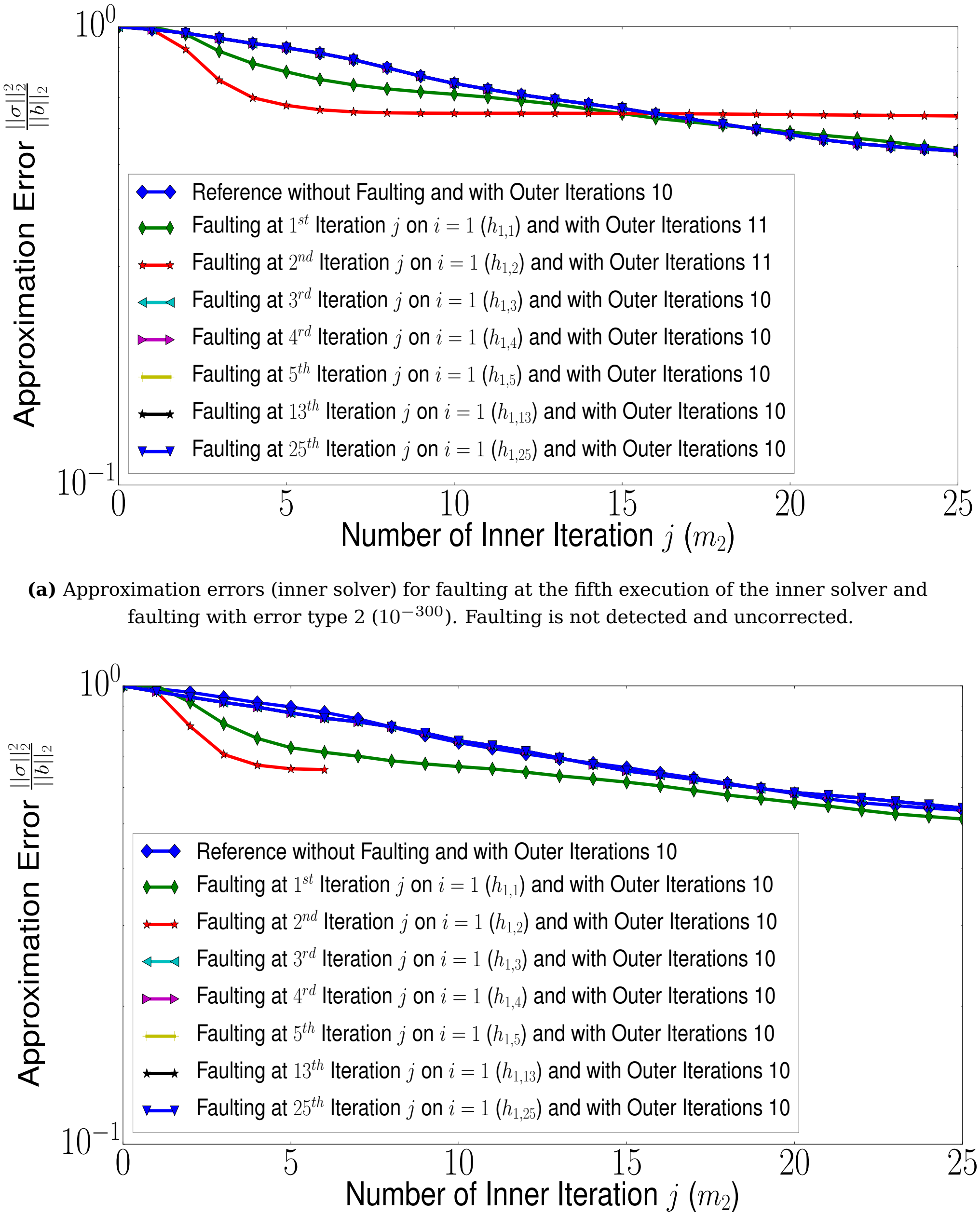
(a) Approximation errors (inner solver) for faulting at the fifth execution of the inner solver and faulting with error type 1 (10^{150}). Faulting is not detected and uncorrected.

(b) Approximation errors (inner solver) for faulting at the fifth execution of the inner solver and faulting with error type 1 (10^{150}). Faulting is detected and corrected.

Figure 151 Approximation errors for solving the 2D Poisson problem with and without fault detection. On the top there are no improvements done in contrast to the bottom side.

In figure 151 the 2D Poisson matrix is solved. Faulting is applied during the fifth execution of the inner solver. On the top in figure 151a there is no fault detection applied in contrast to figure 151b with observing the approximation errors (δ_l). For fault detection a value of $\delta_l = 10^{-2}$ is chosen because the lowest computed value of the F-GMRES for solving this problem is $\delta_l \approx 6 \times 10^{-2}$.

10.2.9 Approximation errors with/without correcting stagnating convergence for solving the 2D Poisson matrix (error type 2) - Inner solver initialized with zero values



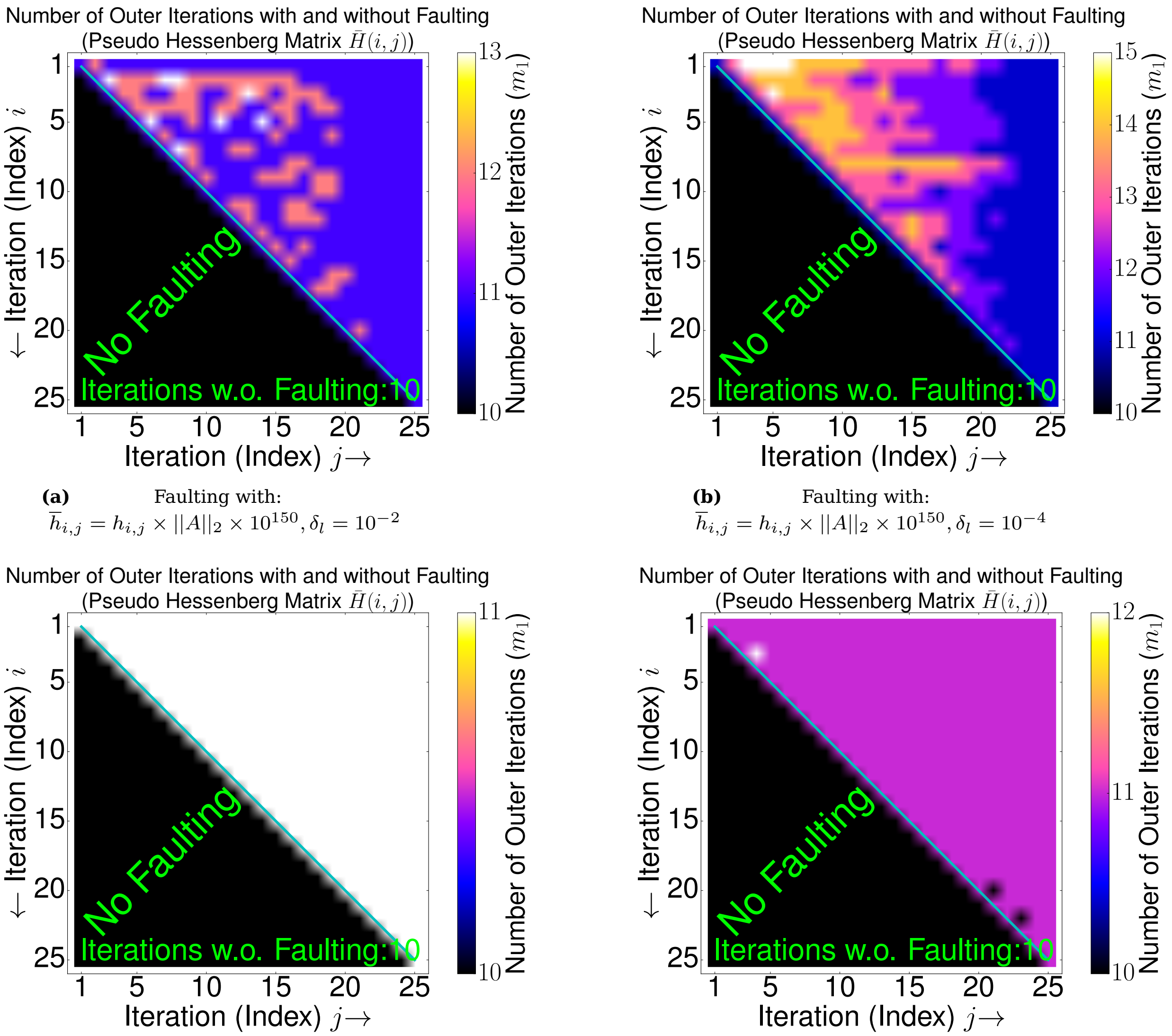
(a) Approximation errors (inner solver) for faulting at the fifth execution of the inner solver and faulting with error type 2 (10^{-300}). Faulting is not detected and uncorrected.

(b) Approximation errors (inner solver) for faulting at the fifth execution of the inner solver and faulting with error type 2 (10^{-300}). Faulting is detected and corrected.

Figure 152 Approximation errors for solving the 2D Poisson problem with and without fault detection. On the top there are no improvements done in contrast to the bottom side.

In figure 152 the 2D Poisson matrix is solved. Faulting is applied during the fifth execution of the inner solver. On the top in figure 152a there is no fault detection applied in contrast to figure 152b with observing the approximation errors (δ_l). For fault detection a value of $\delta_l = 10^{-2}$ is chosen because the lowest computed value of the F-GMRES for solving this problem is $\delta_l \approx 6 \times 10^{-2}$.

10.2.10 Fault detection during solving the 2D Poisson problem with different values for the relative change (δ_l) - Inner solver initialized with zero values



(a) Faulting with:
$\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_l = 10^{-2}$

(b) Faulting with:
$\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}, \delta_l = 10^{-4}$

(c) Faulting with:
$\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}, \delta_l = 10^{-2}$

(d) Faulting with:
$\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}, \delta_l = 10^{-4}$

Figure 153 Fault detection during solving the 2D Poisson matrix with different values for the relative change (δ_l). If a fault is detected then the inner solver of the FT-GMRES is restarted with the last remaining iterations such that the total number of iterations is always the same (25).

In figure 153 different values for δ_l are used in the FT-GMRES for fault detection while solving the 2D Poisson problem, this figure shows that a lot of faults can be detected with $\delta_l = 10^{-2}$ in contrast to $\delta_l = 10^{-4}$ for different kinds of faults. Nearly all faults can be detected with $\delta_l = 10^{-2}$ in figure 153a with observing the relative change between two residuals (δ_l). Figure 153 shows fault detection for a single fault in the inner solver of the FT-GMRES. The inner solver of the Fault Tolerant GMRES (FT-GMRES) is initialized with zero values for vector $w_j(w_0)$. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) is below 10^{-9} for all trials and computations.

10.2.11 Some notes on the parameters for fault detection in the case of the FT-GMRES

In general observing the residual change (δ_l) for fault protection needs a lot of knowledge for the used problem which makes this approach really hard to apply, so this approach should be only used if the convergence behavior is already known for the according matrix A. In the case of solving the 2D Poisson problem with the FT-GMRES solver the inner solver could be protected from a single fault but not in every case which is shown in figure 153 which depends on the value for δ_l where restarting of the inner solver of the FT-GMRES should be done. In the case of solving the adder_dcop_63 matrix it was not possible to find any reasonable value for δ_l where restarting of the inner solver should be done because of the shape for δ_l which is shown in figure 148 and 146.

This is mainly related to the fact that the convergence speed in the inner solver of the F(T)-GMRES is not always the same especially in the case of solving the adder_dcop_63 matrix. In general the change of the residuals respective approximation errors is just crucial for applying fault detection with observing the relative change (δ_l) because if the residual reduction from one iteration to the next iteration is far away of being constant or not within a specific (small) range of values then it is hard to find a reasonable value for δ_l where restarting of the inner solver should be done.

So only in the case of solving the 2D Poisson problem the inner solver could be protected from a single fault if the FT-GMRES solver is used. For solving the 2D Poisson problem with the FT-GMRES the value of δ_l for restarting the inner solver of the FT-GMRES is chosen such that this value is lower than all computed values of δ_l with the F-GMRES while solving the same problem. In figure 153 a value of 10^{-2} and 10^{-4} is chosen for δ_l where restarting of the inner solver of the FT-GMRES should be done which is lower than all computed values of δ_l with the F-GMRES ($\delta_l \approx 6 \times 10^{-2}$).

This idea in section 10.2.1 and figure 138 can be discarded because the value of δ_l changes in general for each outer iteration respective execution of the inner solver of the F-GMRES. Furthermore if the shape for δ_l changes too much for the according matrix then no reasonable value can be applied where restarting should be done for the inner solver of the FT-GMRES. Solving the 2D Poisson matrix with detecting stagnating converge is just an exception where this kind of approach can be used maybe there are other problems where this approach can be applied.

10.3 Fault detection through checking the structure of matrix $\hat{H}$

10.3.1 Introduction

One further possibility to detect faults in the inner solver of the FT-GMRES (line 4 of algorithm 14) is to check the structure of the Hessenberg matrix after applying the Givens rotation (see algorithm 5) but because of the fact that there can be two different structures for the Hessenberg matrix there must be two cases considered depending on the properties of matrix A.

In the general case an upper Hessenberg matrix will be built like in formula 114 shown for 10 iterations after applying the Givens rotation which deletes non-zero elements below the main diagonal. For symmetric problems if following property holds such that $A = A^T$ where matrix A and the according transposed matrix A^T are the same if rows and columns are exchanged then a tridiagonal or band matrix after applying the Givens rotation will be built like in formula 113, before applying the Givens rotation it is indeed a tridiagonal matrix like in formula 105.

Tridiagonal matrix (Banded matrix) - after all Givens rotations

$$\hat{H}(i,j) \quad = \quad \begin{array}{cc} & \rightarrow \text{Iteration (Index) } j \\ \downarrow \text{Iteration (Index) } i & \begin{bmatrix} x & x & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x & x & x & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & x & x & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x & x & x & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & x & x & x & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x & x & x & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x \end{bmatrix} \end{array} \tag{113}$$

Upper Hessenberg matrix - after all Givens rotations

$$\hat{H}(i,j) \quad = \quad \begin{array}{cc} & \rightarrow \text{Iteration (Index) } j \\ \downarrow \text{Iteration (Index) } i & \begin{bmatrix} x & x & x & x & x & x & x & x & x & x \\ 0 & x & x & x & x & x & x & x & x & x \\ 0 & 0 & x & x & x & x & x & x & x & x \\ 0 & 0 & 0 & x & x & x & x & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x & x & x & x \\ 0 & 0 & 0 & 0 & 0 & x & x & x & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x \end{bmatrix} \end{array} \tag{114}$$

In the case of a banded or tridiagonal matrix like in formula 113 the total number of non-zero elements can be checked but also the non-zero values for each row can be verified which must be in this case 1 up to 3 depending on the according row number and iteration. In contrast of an upper Hessenberg matrix like in formula 114 the total number of non-zero elements can be as well verified which are for each iteration j exactly $0.5j(j+1)$. Values where the absolute magnitudes are greater than 10^{-9} are considered as non-zero entries marked with "x" in formula 113 and 114.

10.3.2 Absolute lowest value of the Hessenberg matrix in the inner solver of the F-GMRES for solving different problems - Inner solver initialized with zero values

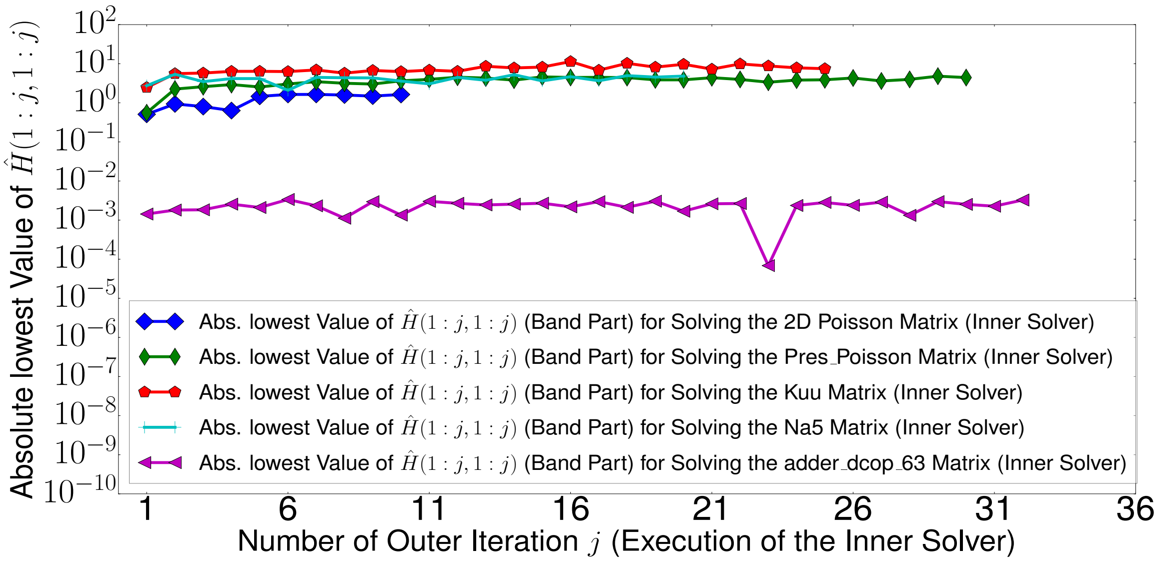


Figure 154 Absolute lowest value of the Hessenberg matrix (band part of $\hat{H}(1:j,1:j)$, see formula 113) in the inner solver of the F-GMRES for solving different (un-)symmetric problems.

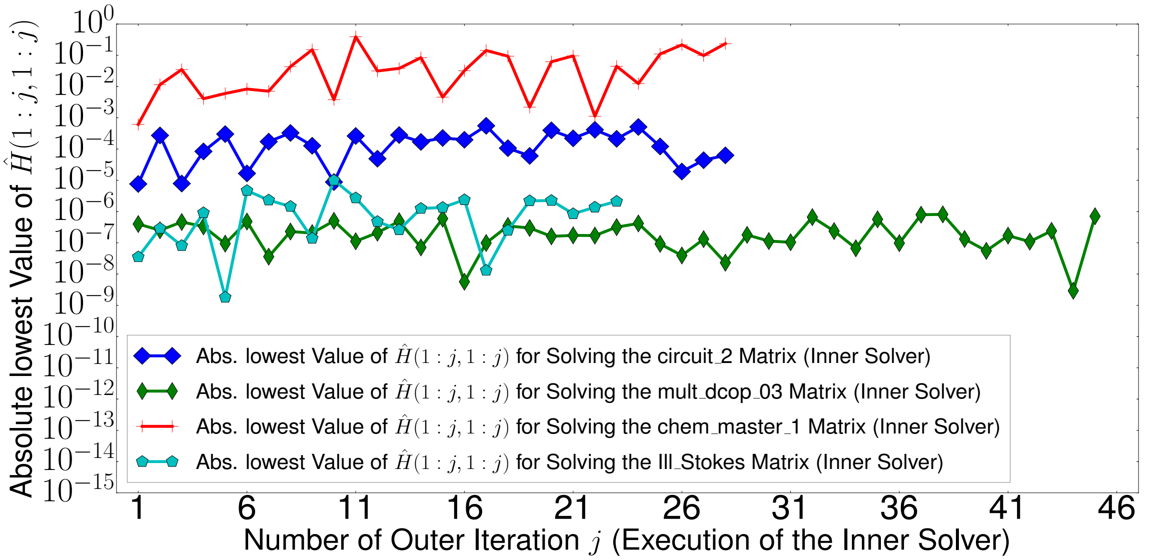


Figure 155 Absolute lowest value of the Hessenberg matrix (upper triangular part of $\hat{H}(1:j,1:j)$, see formula 114) in the inner solver of the F-GMRES for solving different un-symmetric problems.

In figure 154 and 155 the absolute lowest value of the Hessenberg matrix is shown for different problems (symmetric and un-symmetric) and for each execution of the inner solver of the F-GMRES such that a specific accuracy is achieved. In the case of solving a symmetric problem only the band part is considered after applying all Givens rotations and in the case of solving an un-symmetric problem the whole upper triangular part is taken for computing the absolute lowest value of $\hat{H}(1:j,1:j)$. For all problems in figure 154 and 155 the inner solver does 25 iterations. For solving the adder_dcop_63 problem the inner solver builds unexpected a tridiagonal matrix (see figure 126).

10.3.3 Absolute largest value of the Hessenberg matrix in the inner solver of the F-GMRES for solving different problems - Inner solver initialized with zero values

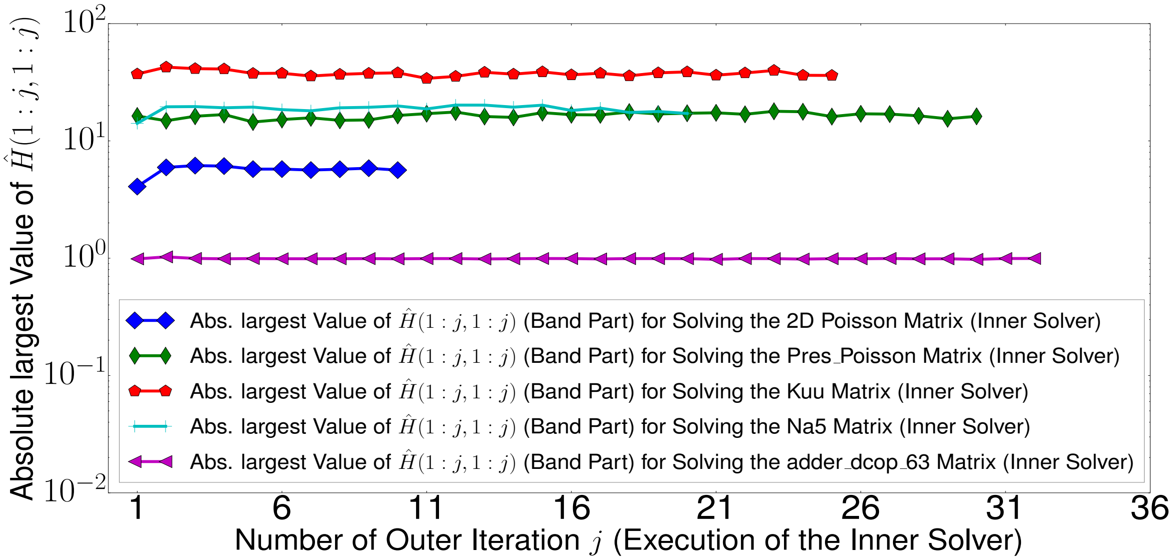


Figure 156 Absolute largest value of the Hessenberg matrix (band part of $\hat{H}(1 : j, 1 : j)$, see formula 113) in the inner solver of the F-GMRES for solving different (un-)symmetric problems.

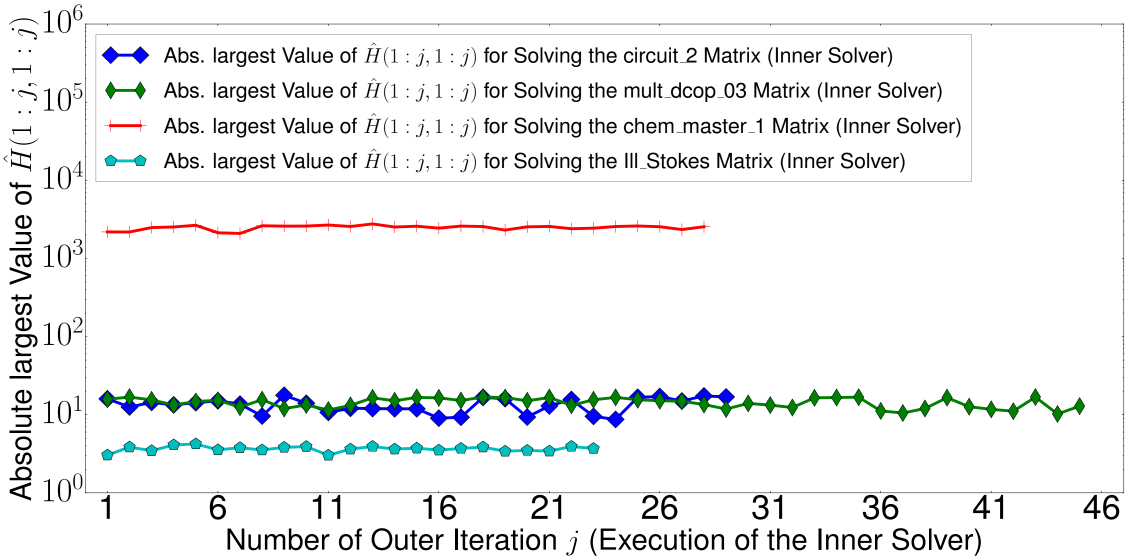
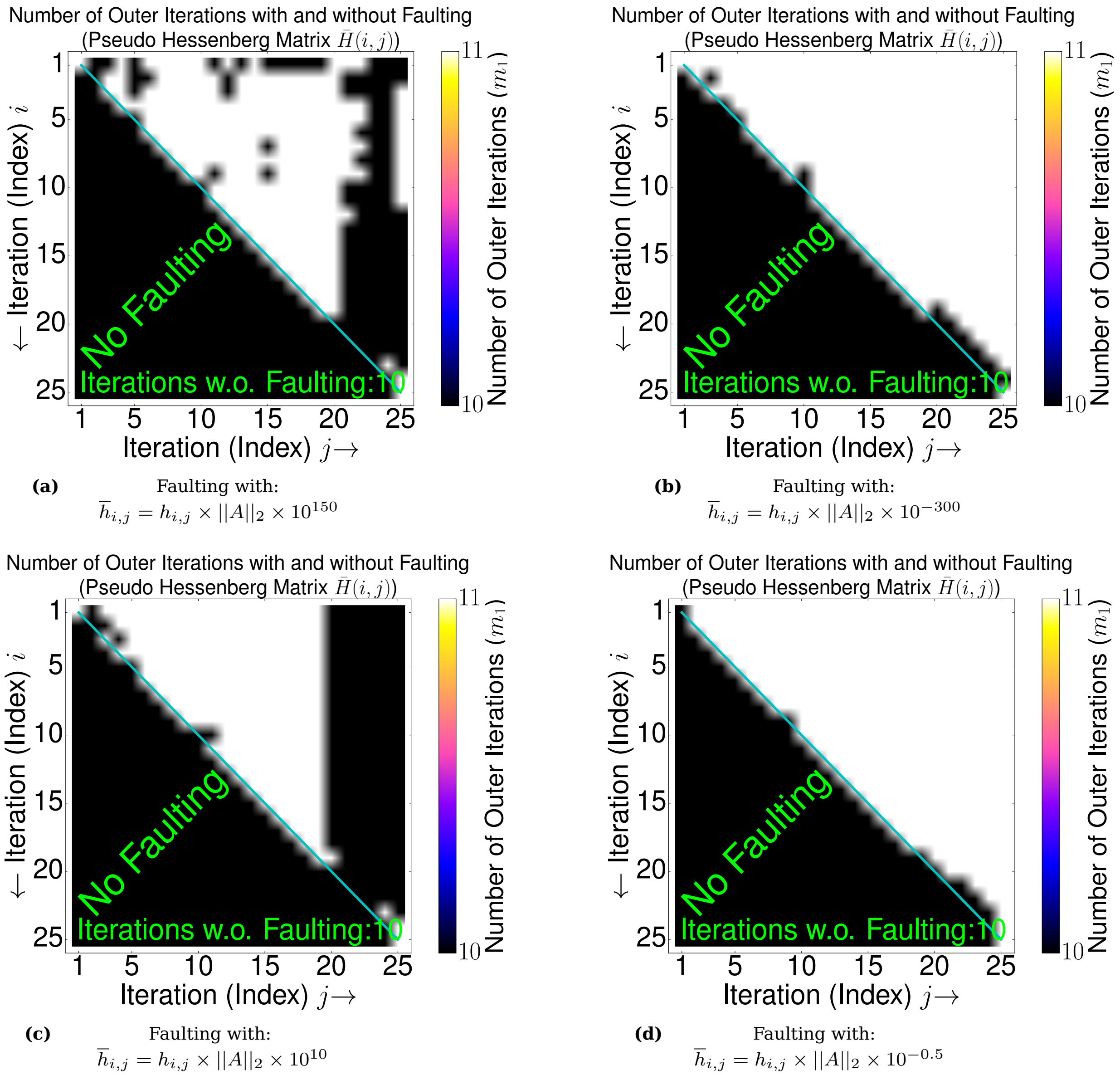


Figure 157 Absolute largest value of the Hessenberg matrix (upper triangular part of $\hat{H}(1 : j, 1 : j)$, see formula 114) in the inner solver of the F-GMRES for solving different un-symmetric problems.

In figure 156 and 157 the absolute largest value of the Hessenberg matrix is shown for different problems (symmetric and un-symmetric) and for each execution of the inner solver of the F-GMRES such that a specific accuracy is achieved. In the case of solving a symmetric problem the whole upper triangular matrix is considered after applying all Givens rotations and as well as in the case of solving an un-symmetric problem for computing the absolute largest value of $\hat{H}(1 : j, 1 : j)$. For all problems in figure 156 and 157 the inner solver does 25 iterations. In the case of solving the adder_dcop_63 problem the inner solver builds unexpected a tridiagonal matrix (see figure 126).

10.3.4 Fault detection during solving the 2D Poisson problem with checking the structure of the Hessenberg matrix - Inner solver initialized with zero values



(a) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

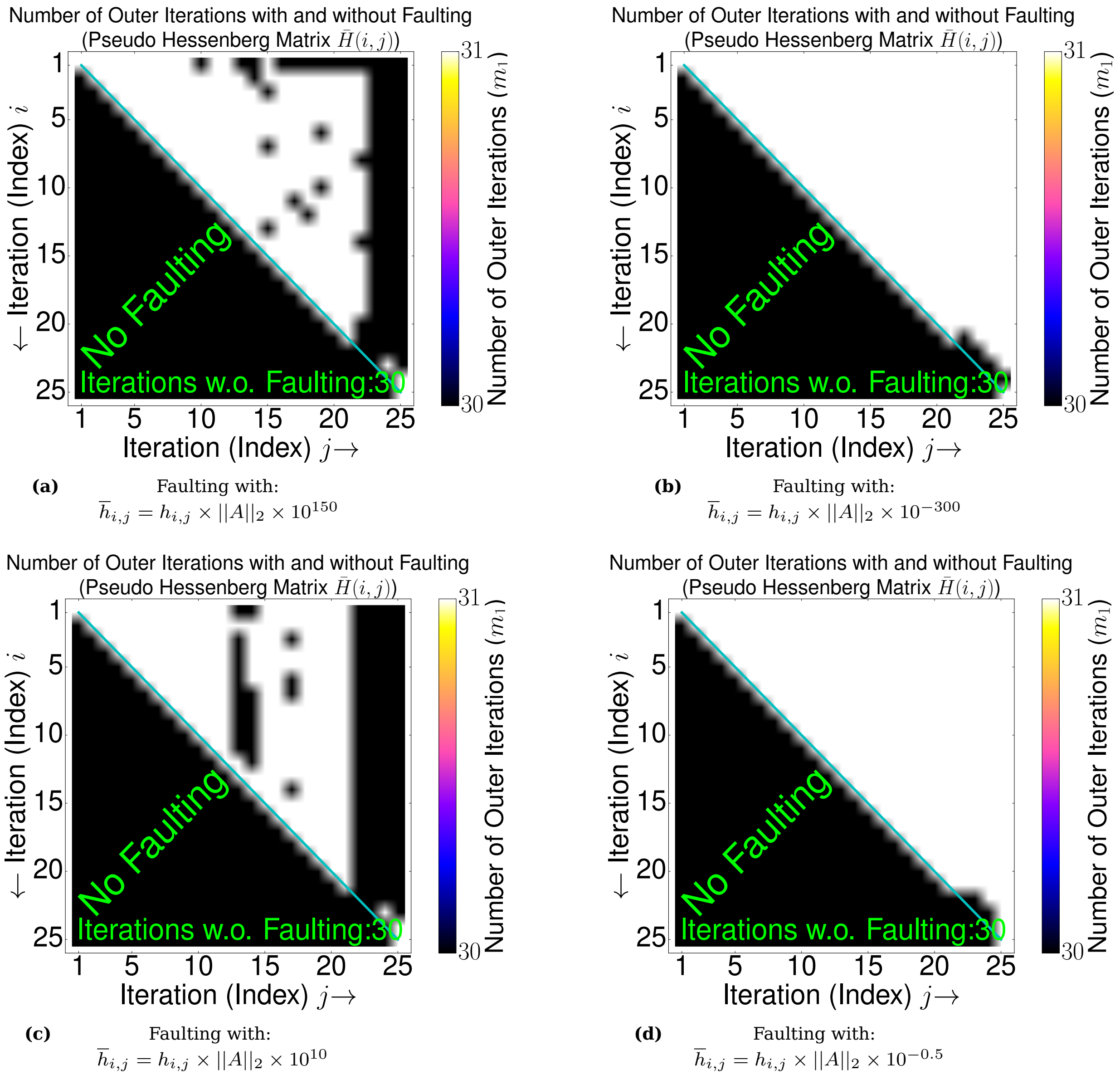
(c) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 158 Fault detection during solving the 2D Poisson problem for different kinds of faults with checking the structure of the Hessenberg matrix in the inner solver of the FT-GMRES.

In figure 158 fault detection during solving the 2D Poisson problem is shown. For this matrix the inner solver of the FT-GMRES builds a tridiagonal or band matrix like in formula 113 in the case of this symmetric problem. The FT-GMRES takes 10 outer iterations (m_1) without a fault whereas 25 iterations are set for the inner solver (m_2). All faults in the inner solver of the FT-GMRES are corrected with restarting the inner solver at the current iteration j and reusing the solution (w_j). Values of $\hat{H}$ where the absolute magnitudes are greater than 10^{-9} are considered as non-zero values. Fault detection is mainly applied for single faults in figure 158 for each position of $h_{i,j}$. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of all trials and computations is below 10^{-9}.

10.3.5 Fault detection during solving the Pres_Poisson problem with checking the structure of the Hessenberg matrix - Inner solver initialized with zero values



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

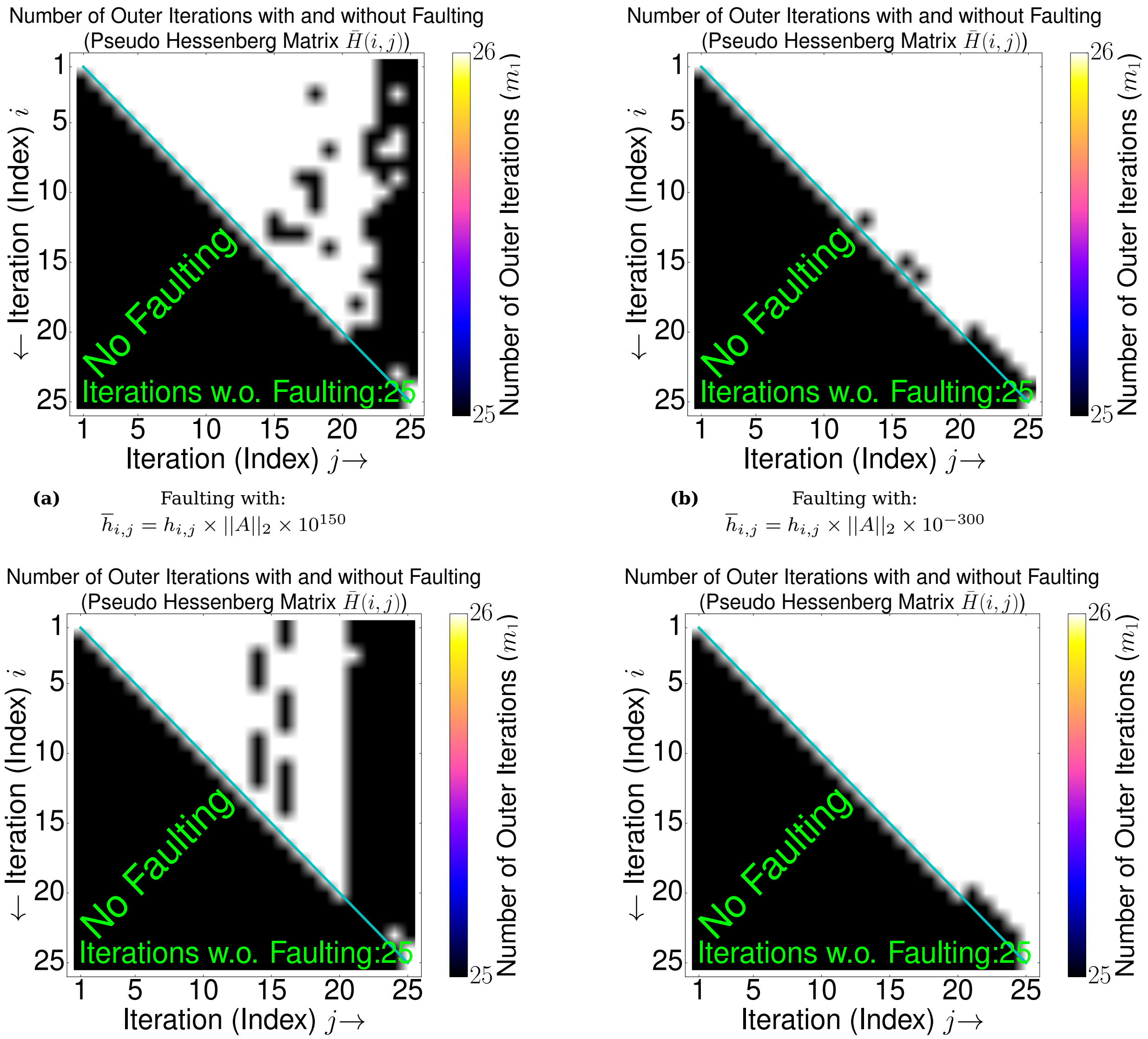
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 159 Fault detection during solving the Pres_Poisson problem for different kinds of faults with checking the structure of the Hessenberg matrix in the inner solver of the FT-GMRES.

In figure 159 fault detection during solving the Pres_Poisson problem is shown. For this matrix the inner solver of the FT-GMRES builds a tridiagonal or band matrix like in formula 113 in the case of this symmetric problem. The FT-GMRES takes 30 outer iterations (m_1) without a fault whereas 25 iterations are set for the inner solver (m_2). All faults in the inner solver of the FT-GMRES are corrected with restarting the inner solver at the current iteration j and reusing the solution (w_j). Values of $\hat{H}$ where the absolute magnitudes are greater than 10^{-9} are considered as non-zero values. Fault detection is mainly applied for single faults in figure 159 for each position of $h_{i,j}$. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of all trials and computations is below 10^{-6}.

10.3.6 Fault detection during solving the Kuu problem with checking the structure of the Hessenberg matrix - Inner solver initialized with zero values



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

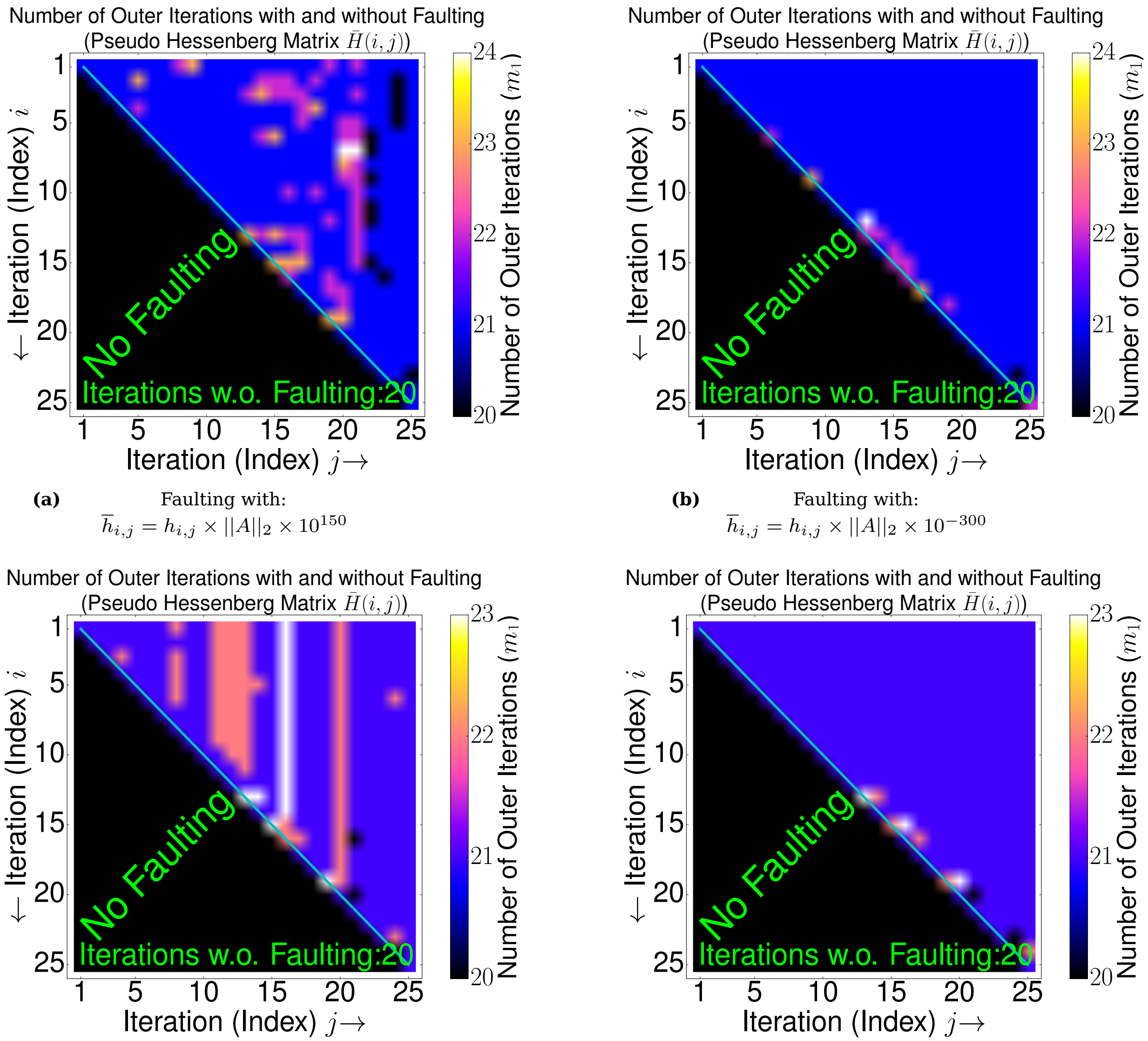
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 160 Fault detection during solving the Kuu problem for different kinds of faults with checking the structure of the Hessenberg matrix in the inner solver of the FT-GMRES.

In figure 160 fault detection during solving the Kuu problem is shown. For this matrix the inner solver of the FT-GMRES builds a tridiagonal or band matrix like in formula 113 in the case of this symmetric problem. The FT-GMRES takes 25 outer iterations (m_1) without a fault whereas 25 iterations are set for the inner solver (m_2). All faults in the inner solver of the FT-GMRES are corrected with restarting the inner solver at the current iteration j and reusing the solution (w_j). Values of $\hat{H}$ where the absolute magnitudes are greater than 10^{-9} are considered as non-zero values. Fault detection is mainly applied for single faults in figure 160 for each position of $h_{i,j}$. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of all trials and computations is below 10^{-9}.

10.3.7 Fault detection during solving the Na5 problem with checking the structure of the Hessenberg matrix - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

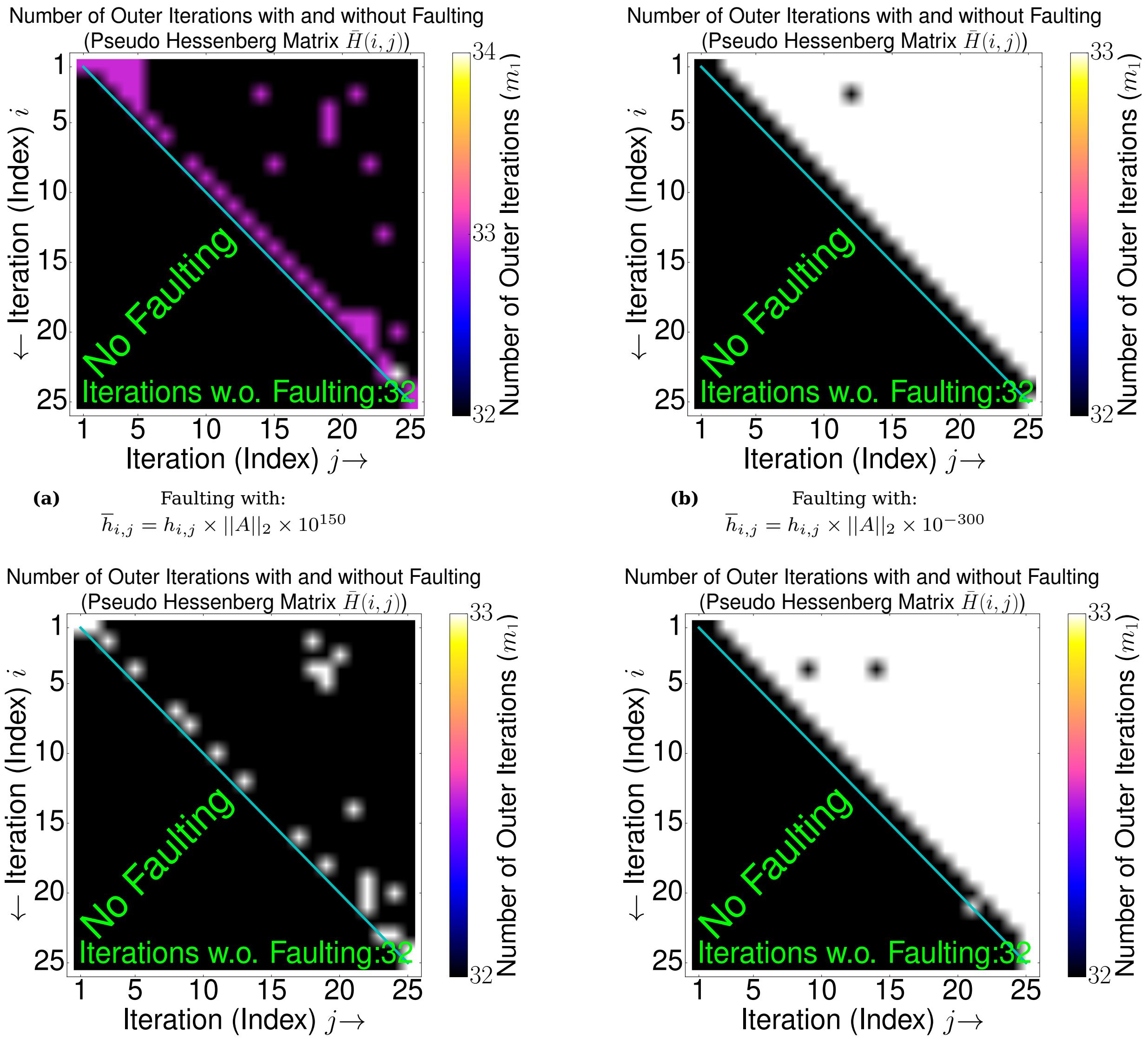
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 161 Fault detection during solving the Na5 problem for different kinds of faults with checking the structure of the Hessenberg matrix in the inner solver of the FT-GMRES.

In figure 161 fault detection during solving the Na5 problem is shown. For this matrix the inner solver of the FT-GMRES builds a tridiagonal or band matrix like in formula 113 in the case of this symmetric problem. The FT-GMRES takes 20 outer iterations (m_1) without a fault whereas 25 iterations are set for the inner solver (m_2). Some faults in the inner solver of the FT-GMRES are corrected with restarting the inner solver at the current iteration j and reusing the solution (w_j). Values of $\hat{H}$ where the absolute magnitudes are greater than 10^{-9} are considered as non-zero values. Fault detection is mainly applied for single faults in figure 161 for each position of $h_{i,j}$. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of all trials and computations is below 10^{-9}.

10.3.8 Fault detection during solving the adder_dcop_63 problem with checking the structure of the Hessenberg matrix - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

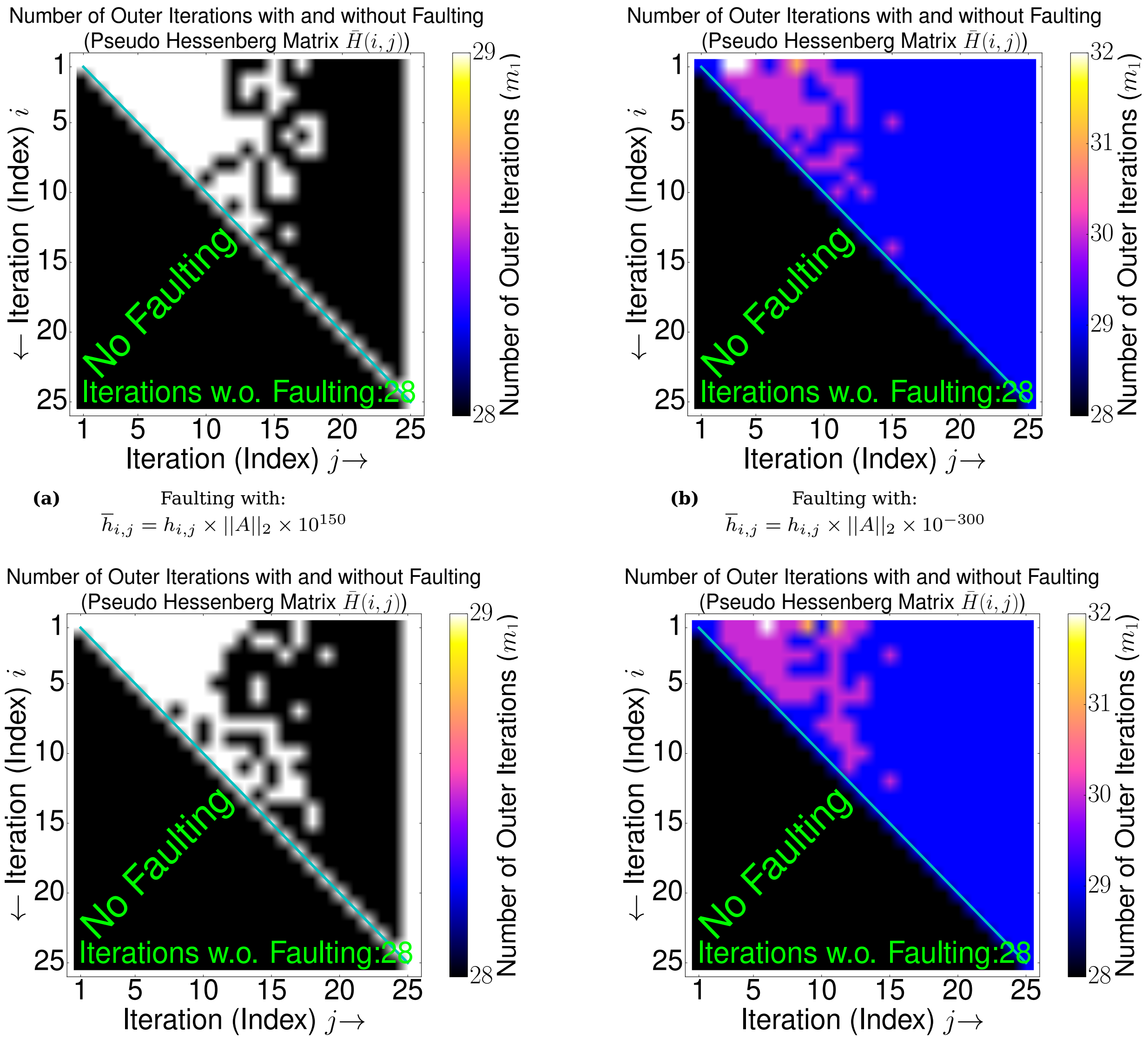
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 162 Fault detection during solving the adder_dcop_63 problem for different kinds of faults with checking the structure of the Hessenberg matrix in the inner solver of the FT-GMRES.

In figure 162 fault detection during solving the adder_dcop_63 matrix is shown. For this matrix the inner solver of the FT-GMRES builds a tridiagonal or band matrix like in formula 113 in the case of this un-symmetric problem which is unexpected. The FT-GMRES takes 32 outer iterations (m_1) without a fault whereas 25 iterations are set for the inner solver. Nearly all faults in the inner solver of the FT-GMRES are corrected with restarting the inner solver at iteration j and reusing the solution (w_j). Values of $\hat{H}$ where the absolute magnitudes are greater than 10^{-6} are considered as non-zero values. Fault detection is mainly applied for single faults in figure 162 for each position of $h_{i,j}$. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of all trials and computations is below 10^{-9}.

10.3.9 Fault detection during solving the circuit_2 problem with checking the structure of the Hessenberg matrix - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

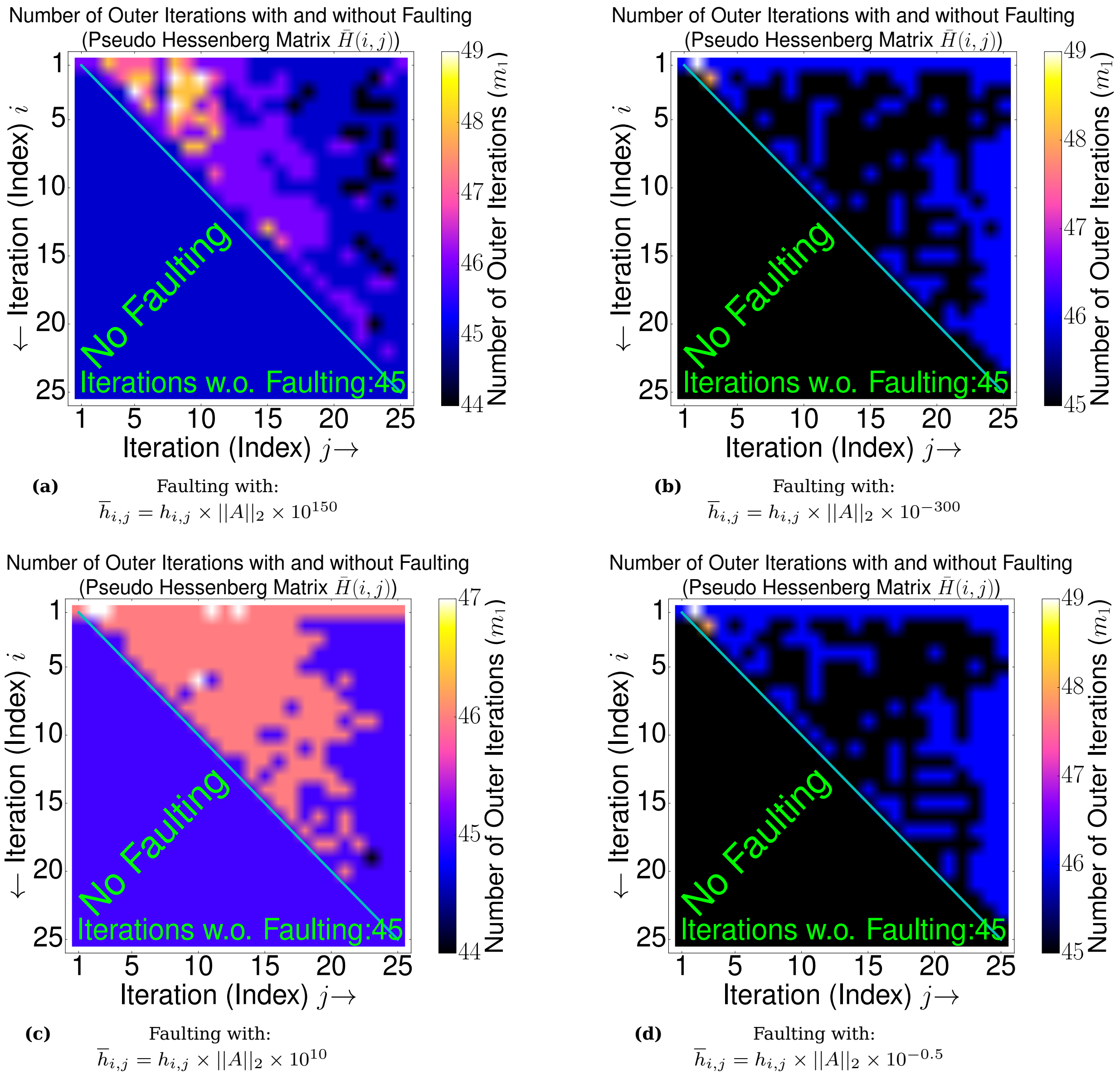
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 163 Fault detection during solving the circuit_2 problem for different kinds of faults with checking the structure of the Hessenberg matrix in the inner solver of the FT-GMRES.

In figure 163 fault detection during solving the circuit_2 matrix is shown. For this matrix the inner solver of the FT-GMRES builds an upper Hessenberg matrix like in formula 114 in the case of this un-symmetric problem. The FT-GMRES takes 28 outer iterations (m_1) without a fault whereas 25 iterations are set for the inner solver (m_2). Some faults in the inner solver of the FT-GMRES are corrected with restarting the inner solver at the current iteration j and reusing the solution (w_j). Values of $\hat{H}$ where the absolute magnitudes are greater than 10^{-9} are considered as non-zero values. Fault detection is mainly applied for single faults in figure 163 for each position of $h_{i,j}$. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of all trials and computations is below 10^{-9}.

10.3.10 Fault detection during solving the mult_dcop_03 problem with checking the structure of the Hessenberg matrix - Inner solver initialized with zero values



(a) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

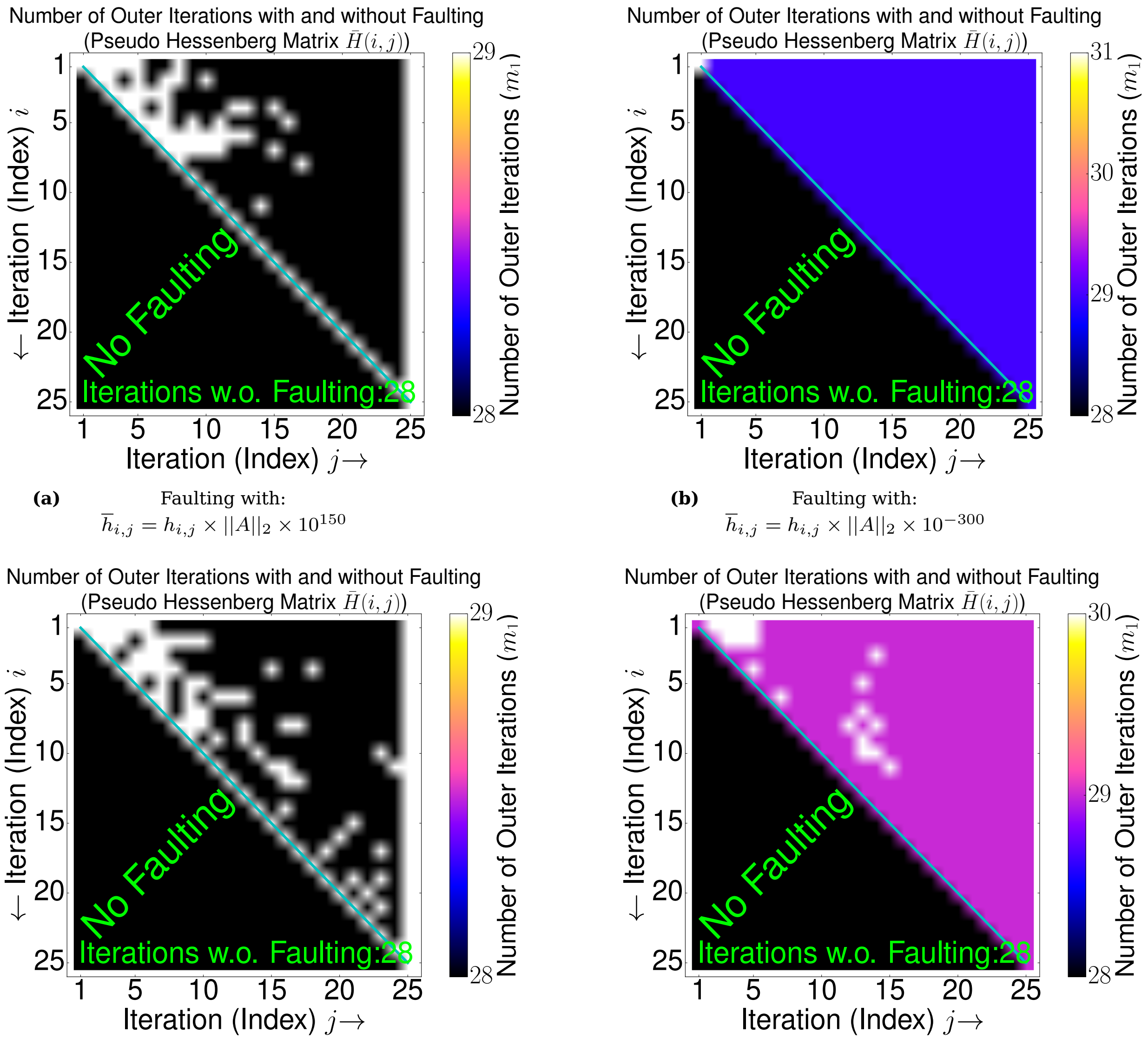
(c) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 164 Fault detection during solving the mult_dcop_03 problem for different kinds of faults with checking the structure of the Hessenberg matrix in the inner solver of the FT-GMRES.

In figure 164 fault detection during solving the mult_dcop_03 matrix is shown. For this matrix the inner solver of the FT-GMRES builds an upper Hessenberg matrix like in formula 114 in the case of this un-symmetric problem. The FT-GMRES takes 45 outer iterations (m_1) without a fault whereas 25 iterations are set for the inner solver (m_2). Some faults in the inner solver of the FT-GMRES are corrected with restarting at the current iteration j and reusing the solution (w_j). Values of $\hat{H}$ where the absolute magnitudes are greater than 10^{-9} are considered as non-zero values. Fault detection is mainly applied for single faults in figure 164 for each position of $h_{i,j}$. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of all trials and computations is below 10^{-9}.

10.3.11 Fault detection during solving the chem_master1 problem with checking the structure of the Hessenberg matrix - Inner solver initialized with zero values



(a) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

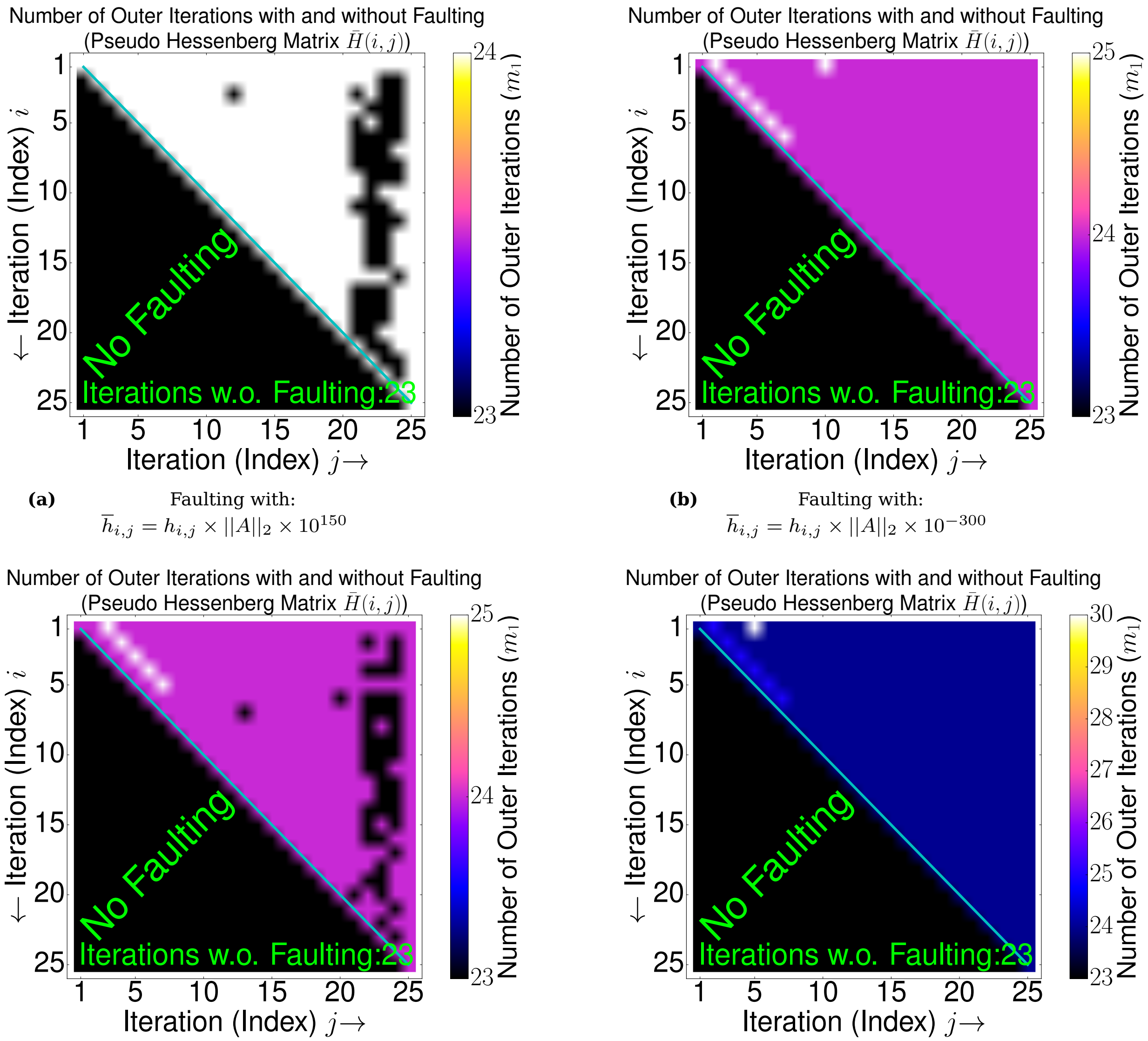
(c) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with: $\overline{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 165 Fault detection during solving the chem_master1 problem for different kinds of faults with checking the structure of the Hessenberg matrix in the inner solver of the FT-GMRES.

In figure 165 fault detection during solving the chem_master1 matrix is shown. For this matrix the inner solver of the FT-GMRES builds an upper Hessenberg matrix like in formula 114 in the case of this un-symmetric problem. It takes 28 outer iterations (m_1) without a fault whereas 25 iterations are set for the inner solver (m_2). Nearly all faults in the inner solver of the FT-GMRES are corrected with restarting the inner solver at the current iteration j and reusing the solution (w_j). Values of $\hat{H}$ where the absolute magnitudes are greater than 10^{-9} are considered as non-zero values. Fault detection is mainly applied for single faults in figure 165 for each position of $h_{i,j}$. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of all trials and computations is below $10^{-5.5}$.

10.3.12 Fault detection during solving the ILL_Stokes problem with checking the structure of the Hessenberg matrix - Inner solver initialized with zero values



(a) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$

(b) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-300}$

(c) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{10}$

(d) Faulting with:
$\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{-0.5}$

Figure 166 Fault detection during solving the ILL_Stokes problem for different kinds of faults with checking the structure of the Hessenberg matrix in the inner solver of the FT-GMRES.

In figure 166 fault detection during solving the ILL_Stokes matrix is shown. For this matrix the inner solver of the FT-GMRES builds an upper Hessenberg matrix like in formula 114 in the case of this un-symmetric problem. It takes 23 outer iterations (m_1) without a fault whereas 25 iterations are set for the inner solver (m_2). Some faults in the inner solver of the FT-GMRES are corrected with restarting the inner solver at the current iteration j and reusing the solution (w_j). Values of $\hat{H}$ where the absolute magnitudes are greater than 10^{-9} are considered as non-zero values. Fault detection is mainly applied for single faults in figure 166 for each position of $h_{i,j}$. The relative residuum ($||r||_2 = ||Ax - b||_2/||b||_2$) of all trials and computations is below 10^{-6}.

10.4 The Extended Fault Tolerant GMRES (EFT-GMRES) method

In algorithm 15 there is shown how fault correction is done (in this master work) with checking the structure of the Hessenberg matrix and other properties in the inner solver of the FT-GMRES, this is mainly related for the line 3 up to 8. This algorithm shows how fault correction is done for the FT-GMRES in section 10.2 and 10.3 if for example a deviation of the structure of the Hessenberg matrix is detected at iteration j (m_3) or another kind of fault is detected. The inner solver is restarted with the last remaining iterations ($m_2 - m_3$) and solution (w_j) if a fault is detected at a specific iteration j (m_3) in the inner solver such that the total number of iterations is at most m_2. There is also the possibility to recompute the last computation (inner solver) with all iterations for the inner solver (m_2) and last solution w_j which is then a full restart for the inner solver.

Algorithm 15 The Extended Fault Tolerant GMRES (EFT-GMRES) algorithm without restarts:

Input: Matrix A, right hand side b and initial starting vector x_0 and ϵ as the accuracy for solving $Ax = b$ and m_1 for the number of outer iterations and m_2 for the iterations of the inner solver.

Output: Approximate solution of vector x_{m_1} for $m_1 > 0$.

```
1:  $r_0 = b - Ax_0$, $\beta = ||r_0||_2$, $q_1 = r_0/\beta$   ...▷ Compute initial residuum vector, first basis vector $q_1$.
2:  for $j = 1, 2, ...$until convergence and $j < m_1$ do
3:    Solve $q_j = M_j w_j$ for $w_j$   ...▷ Do preconditioning in unreliable mode like
      $[w_j, m_3, restart] = GMRES(M_j, q_j, w_0, m_2, \dots)$.
4:    $m_4 = m_2$
5:    while $m_4 \geq 0$ or $restart! = None$ do
6:      $m_4 = m_4 - m_3$   ...▷ Lower number of total iterations for the inner solver.
7:      $[w_j, m_3, restart] = GMRES(M_j, q_j, w_j, m_4, \dots)$   ...▷ Restart inner solver.
8:    end while
9:    $v_{j+1} = Aw_j$   ...▷ Perform matrix vector product.
10:   ..▷ Orthogonalize basis vector $q_j$ (line 11-15).
11:   for $i = 1, 2, ...j$ do
12:     $h_{i,j} = q_i^T v_{j+1}$
13:     $v_{j+1} = v_{j+1} - h_{i,j} q_i$
14:   end for
15:   $h_{j+1,j} = ||v_{j+1}||_2$
16:   ▷ Do rank revealing decomposition [49].
17:   if $\hat{H}(j{+}1, j) < \epsilon_{Machine}$ then   ...▷ Stopped if machine precision is achieved
      because $\hat{H}(j+1, j) \approx 0$ so no further residual reduction.
18:     if $\hat{H}(1:j, 1:j)$ no full rank then
19:       Did not converged.
20:     else
21:       Solution is $x_{j-1}$.
22:     end if
23:   end if
24:   $q_{j+1} = v_{j+1}/h_{j+1,j}$   ...▷ New basis vector $q_{j+1}$.
25:   $y_j = argmin_y||\hat{H}(i : j+1, 1 : j)y - \beta e_1||_2$   ...▷ Solve least square problem.
26:   $x_j = x_0 + [q_1, q_2, \dots, q_j]y_j$   ...▷ Compute solution $x_j$.
27: end for
28: Solution found $x_{j-1}$.
```

11 Conclusions and final results

The F-GMRES (FT-GMRES) seems to be a good approach in terms of number of floating point operations for the inner solver. As more iterations are done in the inner solver of the F-GMRES (FT-GMRES) the fraction between the outer (m_1) and inner solver (m_2) will increase which means if the number of iterations for the inner solver is high then the density or fraction of workloads will be also high ($\frac{m_2^2}{m_1}$). This fraction will be especially high if matrix A has a lot of non-zero elements. The main outcome is that most of the operations are done in the inner solver and in unreliable.

In contrast the F-CG has always a constant workload ratio (m_2) between the outer and inner solver independent of the non-zero elements of matrix A. This outcome is mainly related to the fact that the number of operations of the GMRES solver during the orthogonalization process ($h_{i,j}$) rises quadratically but in the case of the CG solver there is always constant work for each iteration.

The initialization of flexible methods as well for any iterative method is just crucial which means in the standard case the inner solver should be initialized with zero values to achieve the best speed up and to ensure convergence if nothing is known like the inverse norm of matrix A ($||A^{-1}||_2$). How to get an initial value for vector w_j is mainly based on the fact that the problem $q_j = M_j w_j$ should be solved for vector w_j in the inner solver then let vector $w_j = M_j^{-1} q_j$ and with using the norm ($||.||_2$) it follows that $||w_j||_2 \leq ||M_j^{-1}||_2 ||q_j||_2$ and by knowing that $||q_j||_2 \leq 1$ it leads to $||w_j||_2 \leq ||M_j^{-1}||_2$. If the norm of matrix A is smaller than 1 such that $||A||_2 \leq 1$ and let $M_j \approx A$ then this implies that vector w_j should be initialized with large values but in this case it makes preconditioning obsolete of the F-GMRES solver because the GMRES solver is preferred in this case ($||A||_2 \leq 1$) so vector w_j should be always initialized with zero values in the F(T)-GMRES solver to get the best speed up.

Faulting can be done with different kinds of perturbations, in the case of a diagonal matrix faulting with small perturbation coefficients ($E_{perturbed} \leq 1$) is only critical on some specific positions like for positions of $h_{1,2}$, $h_{2,3}$, $h_{3,4}$, ... Also for other kinds of matrices where an upper Hessenberg matrix will be built, faulting with large and as well small perturbations seems to be critical but in the case of large perturbations there is a bigger chance that an overhead occurs for all positions of $h_{i,j}$. Further experiments with the 2D Poisson matrix which are not discussed in detail in this master work also show that those positions ($h_{1,2}$, $h_{2,3}$, $h_{3,4}$, ...) are really sensitive for some disturbances. So if the total disturbance ($||A||_2 \times E_{perturbed}$) is unequal to 1 at positions of $h_{1,2}$, $h_{2,3}$, $h_{3,4}$, ... then there is a big chance that some overhead occurs in number of outer iterations (m_2) for the FT-GMRES. One explanation is just that small disturbances can also change the structure of the Hessenberg matrix in the inner solver of the FT-GMRES.

In the case of un-symmetric problems where an upper Hessenberg matrix is build faulting is as well crucial at positions of $h_{1,2}$, $h_{2,3}$, $h_{3,4}$, ... but other positions seems to be also critical but they are less affected. In the case of multiple faults, faulting of a whole row of the Hessenberg matrix in the inner solver of the FT-GMRES seems to be more critical than faulting of a single column.

If the inner solver is initialized with zero values for vector $w_j(w_0)$ then an overhead of about 0 to 300 % can be observed and on average of about 15 to 20 %. It is hard to determine the overhead in general but other experiments respective matrices in this work show that also a similar overhead can be observed which means an analog overhead should be observed intuitively for other matrices. The overhead for diagonal matrices doesn't really depend on the condition number because the overhead is nearly constant (between 10 % to 30 %) of all used diagonal matrices in this master work. It mainly means that the overhead depends more on the convergence rate and on the eigenvalue distribution but not entirely on the smallest and greatest eigenvalue (λ).

Sometimes ILU-preconditioning can help to decrease the overhead as in the case of solving the 2D Poisson matrix of about 36,4 % to 14.3 % in the presence of a single fault. There is also a big drawback because the additional number of flops for preconditioning is really high of about 70 % more against the not preconditioned case without faulting. It should be mentioned forward and backward substitution is counted really pessimistic which means there will be perhaps more gain. Not in every case preconditioning could help to decrease the overhead but for these problems the overhead was nearly the same as in the not preconditioned case. It is in general not worth to apply ILU-preconditioning to reduce the overhead for the FT-GMRES because the caused overhead is similar as in the not preconditioned case without considering the work for factorizing of matrix A.

Observing the relative change between two residuals in the inner solver of the FT-GMRES is a further possibility to reduce the overhead in the presence of a fault but should be only used if the convergence behavior is already known for the used matrix A. It should be pointed out that using the relative change for fault detection needs a lot of knowledge about the converge behavior of each used matrix so it is hard to apply this approach in general for the FT-GMRES.

Checking the structure of the so called Hessenberg matrix looks much more promising and can be used for more matrices to solve the problem of $Ax = b$. Not in every case the used FT-GMRES solver could be protected against a fault but there was really less effort to achieve a good result where lot of faults are detected which means a far better result should be possible, the main problem is still what is considered as non-zero values. Fault detection for symmetric problems is easier because there is more information about the structure of the Hessenberg matrix where in the case of un-symmetric problems the structure is sometimes unchanged because of a fault. Hence for really efficient implementations of the GMRES algorithm where only those values for the tridiagonal matrix are computed and stored a recommendation is to add some additional values $(3 + 2)$ where in the standard case 3 values for each row of the Hessenberg matrix are used such that it is possible to apply the Givens rotation for other positions which are usually zero or almost zero (before applying the Givens rotation) but with the aim to recognize faults during the orthogonalization process and to check the structure of the Hessenberg matrix.

If applied so it is not needed to apply the orthogonalization process for those other positions except those from the tridiagonal matrix but the Givens rotation must be still applied for these additional values and positions to detect faults during the orthogonalization process and to check the structure of the Hessenberg matrix. Multiple faults will also change the structure of the Hessenberg matrix and can be as well detected during the whole orthogonalization process. In general faults like 10^{150} during the orthogonalization process and with the fault methodology of $\bar{h}_{i,j} = h_{i,j} \times ||A||_2 \times 10^{150}$ can be detected easier in contrast to faults with 10^{-300} because in the case of un-symmetric problems a value $(h_{i,j})$ in the Hessenberg matrix will not necessary set to be zero because there is still the Givens rotation which overwrites this value maybe such that this value is not zero. In the case of symmetric problems values $(h_{i,j})$ of the Hessenberg matrix are really sensitive against any kind of perturbation such that a structural change should be recognized.

Faults like $E_{perturbed} = 10^{150}$ will lead to more operations (GMRES tries to minimize the residuum) whereas faults like 10^{-300} results in stagnating convergence because of a structural change of the Hessenberg matrix, every fault can change the structure. Flexible preconditioning mainly helps to decrease the number of operations to converge in an error free run. It can help to minimize the number of operations for an error free computation but a fault in the inner solver of the FT-GMRES will always lead to more operations to decrease the negative effect because of poor preconditioning. A negated fault in the inner solver with flexible preconditioning has similar the same costs (flops) as an not corrected error with right preconditioning. Checking the rank of the Hessenberg matrix in the inner solver of the FT-GMRES does not necessary reveal a fault but should be always done especially for the (F-)GMRES solver. Something which is not done in this master work is to check the numerical stability of the Hessenberg matrix like for example the condition number.

References

[1] D. Gans, “Energy-efficient and cost-effective reliability design in memory systems,” tech. rep., 2014. Graduate Theses and Dissertations, Paper 13710.

[2] D. Gans, “ECC Brings Reliability and Power Efficiency to Mobile Devices.” Website, 2015. EE|times, Online available under: http://www.eetimes.com/author.asp?section_id=36&doc_id=1325816; retrieved on December 2016.

[3] Y. Saad, “A Flexible Inner-outer Preconditioned GMRES Algorithm,” *SIAM J. Sci. Comput.*, vol. 14, no. 2, pp. 461 – 469, 1993.

[4] J. Elliott, M. Hoemmen, and F. Mueller, “Evaluating the Impact of SDC on the GMRES Iterative Solver,” in *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, IPDPS ’14, (Washington, DC, USA), pp. 1193 – 1202, IEEE Computer Society, 2014.

[5] Y. Saad and M. H. Schultz, “GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems,” *SIAM J. Sci. Stat. Comput.*, vol. 7, pp. 856 – 869, July 1986.

[6] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical mathematics*. Texts in applied mathematics, New York, Berlin: Springer, 2000.

[7] M. A. A. Gomes-Ruggiero, V. A. L. R. Lopes, and J. V. Toledo-Benavides, “A safeguard approach to detect stagnation of GMRES(m) with applications in Newton-Krylov methods,” *Computational and Applied Mathematics*, vol. 27, pp. 175 – 199, 2008.

[8] “Reliability.” Website, Last modified January 22, 2016. CCCP, Online available under: http://cccp.eecs.umich.edu/research/reliability.php; retrieved on December 2016.

[9] C. Slayman, “Soft error trends and mitigation techniques in memory devices,” in *Reliability and Maintainability Symposium (RAMS), 2011 Proceedings - Annual*, pp. 1 – 5, Januar 2011.

[10] “Sparse Matrix Collection.” Website. University of Florida, Online available under: https://www.cise.ufl.edu/research/sparse/matrices/; retrieved on December 2016.

[11] E. W. Dijkstra, “Self-stabilizing Systems in Spite of Distributed Control,” *Commun. ACM*, vol. 17, no. 11, pp. 643 – 644, 1974.

[12] P. G. Bridges, K. B. Ferreira, M. A. Heroux, and M. Hoemmen, “Fault-tolerant linear solvers via selective reliability,” *CoRR*, vol. abs/1206.1390, 2012.

[13] K.-H. Huang and J. A. Abraham, “Algorithm-Based Fault Tolerance for Matrix Operations,” *IEEE Trans. Comput.*, vol. 33, pp. 518 – 528, June 1984.

[14] E. Agullo, L. Giraud, A. Guermouche, J. Roman, and M. Zounon, “Towards resilient parallel linear Krylov solvers: recover-restart strategies,” Research Report RR-8324, INRIA, July 2013.

[15] Y. Notay, “Flexible Conjugate Gradients,” *SIAM Journal on Scientific Computing*, vol. 22, no. 4, pp. 1444 – 1460, 2000.

[16] J. R. Shewchuk, “An Introduction to the Conjugate Gradient Method Without the Agonizing Pain,” tech. rep., Pittsburgh, PA, USA, 1994.

[17] G. Bronevetsky and B. de Supinski, “Soft Error Vulnerability of Iterative Linear Algebra Methods,” in *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ICS ’08, (New York, NY, USA), pp. 155 – 164, ACM, 2008.

[18] Z. Zheng, A. A. Chien, and K. Teranishi, *Fault Tolerance in an Inner-Outer Solver: A GVR-Enabled Case Study*, pp. 124 – 132. Cham: Springer International Publishing, 2015.

[19] J. Elliott, M. Hoemmen, and F. Mueller, “Tolerating Silent Data Corruption in Opaque Preconditioners,” *CoRR*, vol. abs/1404.5552, 2014.

[20] P. E. Bjørstad, M. Dryja, and E. Vainikko, “Parallel Implementation of a Schwarz Domain Decomposition Algorithm,” in *Proceedings of the Third International Workshop on Applied Parallel Computing, Industrial Computation and Optimization*, PARA ’96, (London, UK, UK), pp. 46 – 56, Springer, 1996.

[21] J. Elliott, M. Hoemmen, and F. Mueller, “A Numerical Soft Fault Model for Iterative Linear Solvers,” in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC ’15, (New York, NY, USA), pp. 271 – –274, ACM, 2015.

[22] P. Du, P. Luszczek, S. Tomov, and J. Dongarra, “Soft Error Resilient QR Factorization for Hybrid System with GPGPU,” in *Proceedings of the Second Workshop on Scalable Algorithms for Large-scale Systems*, ScalA ’11, (New York, NY, USA), pp. 11 – 14, ACM, 2011.

[23] “Magma Library.” Website. University of Tennessee, Department of Electrical Engineering and Computer Science, Magma HPC Library, Online available under: `http://icl.cs.utk.edu/magma/`; retrieved on December 2016.

[24] A. Schøll, C. Braun, M. A. Kochte, and H. J. Wunderlich, “Efficient on-line fault-tolerance for the preconditioned conjugate gradient method,” in *2015 IEEE 21st International On-Line Testing Symposium (IOLTS)*, pp. 95–100, July 2015.

[25] Z. Chen, “Online-ABFT: An Online Algorithm Based Fault Tolerance Scheme for Soft Error Detection in Iterative Methods,” *SIGPLAN Not.*, vol. 48, no. 8, pp. 167 – 176, 2013.

[26] G. Aupy, A. Benoit, T. Hérault, Y. Robert, F. Vivien, and D. Zaidouni, “On the Combination of Silent Error Detection and Checkpointing,” in *PRDC - The 19th IEEE Pacific Rim International Symposium on Dependable Computing - 2013*, (Vancouver, Canada), IEEE, 2013.

[27] J. Elliott, M. Hoemmen, and F. Mueller, “Exploiting Data Representation for Fault Tolerance,” in *Proceedings of the 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA ’14, (Piscataway, NJ, USA), pp. 9 – 16, IEEE Press, 2014.

[28] J. Elliott, M. Hoemmen, and F. Mueller, “Quantifying the Impact of Single Bit Flips on Floating Point Arithmetic,” tech. rep., 2013. ORNL REPORT: ORNL/TM-2013/282.

[29] J. Langou, Z. Chen, G. Bosilca, and J. Dongarra, “Recovery Patterns for Iterative Methods in a Parallel Unstable Environment,” *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 102 – 116, 2008.

[30] Z. Chen, “Algorithm-based Recovery for Iterative Methods Without Checkpointing,” in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC ’11, (New York, NY, USA), pp. 73 – 84, ACM, 2011.

[31] J. Sloan, R. Kumar, and G. Bronevetsky, “An algorithmic approach to error localization and partial recomputation for low-overhead fault tolerance,” in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 1 – 12, June 2013.

[32] P. Sao and R. Vuduc, “Self-stabilizing Iterative Solvers,” in *Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA ’13, (New York, NY, USA), pp. 4:1 – 4:8, ACM, 2013.

[33] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* Society for Industrial and Applied Mathematics, 1994.

[34] J. Peterson and J. Burkardt, “Iterative Methods for Solving $Ax = b$.” Lecture Notes. Online available under: `http://people.sc.fsu.edu/~jpeterson/linear_algebra_part3.pdf`; retrieved on December 2016.

[35] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2nd ed., 2003.

[36] M. Botchev, “Numerical Linear Algebra.” Lecture Notes of Lecture 9, 2006. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.329.9777&rep=rep1&type=pdf`; retrieved on December 2016.

[37] D. P. O’Leary, “Notes on Some Methods for Solving Linear Systems.” Lecture Notes, 2007. Online available under: `https://www.cs.umd.edu/users/oleary/a600/cgnotes.pdf`; retrieved on December 2016.

[38] M. H. Gutknecht, *A Brief Introduction to Krylov Space Methods for Solving Linear Systems*, pp. 53 – 62. Berlin, Heidelberg: Springer, 2007.

[39] G. Fasshauer, “Arnoldi Iteration and GMRES.” Lecture Notes. Online available under: `http://math.iit.edu/~fass/477577_Chapter_14.pdf`; retrieved on December 2016.

[40] Wikipedia, “Jacobi Method.” Online available under: `https://en.wikipedia.org/wiki/Jacobi_method`; retrieved on December 2016.

[41] J. R. Senning, “Computing and Estimating the Rate of convergence.” Lecture Notes, 2015. Online available under: `http://www.math-cs.gordon.edu/courses/ma342/handouts/rate.pdf`; retrieved on December 2016.

[42] N. Gmati and B. Philippe, *Comments on the GMRES Convergence for Preconditioned Systems*, pp. 40 – 51. Berlin, Heidelberg: Springer, 2008.

[43] V. Simoncini and D. B. Szyld, “Theory of Inexact Krylov Subspace Methods and Applications to Scientific Computing,” *SIAM Journal on Scientific Computing*, vol. 25, no. 2, pp. 454 – 477, 2003.

[44] V. Simoncini and D. B. Szyld, “On the Occurrence of Superlinear Convergence of Exact and Inexact Krylov Subspace Methods,” *SIAM Review*, vol. 47, no. 2, pp. 247 – 272, 2005.

[45] J. A. Sifuentes, M. Embree, and R. B. Morgan, “GMRES Convergence for Perturbed Coefficient Matrices, with Application to Approximate Deflation Preconditioning,” *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 3, pp. 1066–1088, 2013.

[46] G. Lube, “Arnoldi Verfahren.” Website, 2006 - 2014. Online available under: `https://lp.uni-goettingen.de/get/text/2022`; retrieved on December 2016.

[47] C. Kanzow, *Numerik linearer Gleichungssysteme: Direkte und iterative Verfahren*. Berlin: Springer, 2005.

[48] L. Giraud, J. Langou, and M. Rozloznik, “The loss of orthogonality in the Gram-Schmidt orthogonalization process,” *Computers and Mathematics with Applications*, vol. 50, no. 7, pp. 1069 – 1075, 2005.

[49] G. W. Stewart, “Updating a Rank-Revealing ULV Decomposition,” *SIAM Journal on Matrix Analysis and Applications*, vol. 14, no. 2, pp. 494 – 499, 1993.

[50] G. L. G. Sleijpen and H. A. van der Vorst, “Krylov subspace methods for large linear systems of equations.” Lecture Notes, 1993. Online available under: http://people.sc.fsu.edu/~jpeterson/linear_algebra_part3.pdf; retrieved on December 2016.

[51] M. Smith, “RAM reliability: Soft errors.” Website, 1998. Online available under: http://www.ida.liu.se/labs/eslab/SNDFT/docs/ram-soft.html; retrieved on December 2016.

[52] C. Mims, “Why CPUs Aren’t Getting Any Faster.” Website, 2010. Blog, Online available under: https://www.technologyreview.com/s/421186/why-cpus-arent-getting-any-faster/; retrieved on December 2016.

[53] E. L. Bosworth, “The Power Wall.” Website, 2010. Columbus State University, TSYS School of Computer Science, Online available under: http://www.edwardbosworth.com/My5155_Slides/Chapter01/ThePowerWall.htm; retrieved on December 2016.

[54] “Hard disk drive reliability and MTBF / AFR.” Seagate, Online available under: http://knowledge.seagate.com/articles/en_US/FAQ/174791en?language=en_US; retrieved on December 2016.

[55] M. Travers, “CPU Power Consumption Experiments and Results Analysis of Intel i7-4820K,” tech. rep. Technical Report Series NCL-EEE-MICRO-TR-2015-197.

[56] M. Abe and V. Hudson, “Nanoelectronics for 2020 and Beyond.” Website, 2013. electronics360, Online available under: http://electronics360.globalspec.com/article/183/nanoelectronics-for-2020-and-beyond; retrieved on December 2016.

[57] E. L. Bosworth, “Overcoming the Memory Wall.” Report. Oregon State University, Online available under: http://blogs.oregonstate.edu/ece570/files/2012/02/Report.pdf; retrieved on December 2016.

[58] IT-Wissen, “DRAM Memory.” Website. Online available under: http://www.itwissen.info/definition/lexikon/dynamic-random-access-memory-DRAM-Dynamisches-RAM.html; retrieved on December 2016.

[59] Wikipedia, “History of supercomputingr.” Website. Online available under: http://en.wikipedia.org/wiki/History_of_supercomputing; retrieved on December 2016.

[60] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, “The Soft Error Problem: An Architectural Perspective,” in *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, HPCA ’05, (Washington, DC, USA), pp. 243 – 247, IEEE Computer Society, 2005.

[61] B. Schroeder, E. Pinheiro, and W.-D. Weber, “DRAM Errors in the Wild: A Large-scale Field Study,” in *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS ’09, (New York, NY, USA), pp. 193 – 204, ACM, 2009.

[62] V. Degalahal, R. Ramanarayanan, N. Vijaykrishnan, Y. Xie, and M. J. Irwin, “The Effect of Threshold Voltages on the Soft Error Rate,” in *Proceedings of the 5th International Symposium on Quality Electronic Design*, ISQED ’04, (Washington, DC, USA), pp. 503 – 508, IEEE Computer Society, 2004.

[63] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, “Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors,”

in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 361 – 372, June 2014.

[64] N. DeBardeleben, S. Blanchard, V. Sridharan, S. Gurumurthi, J. Stearley, K. Ferreira, and J. Shalf, “Extra Bits on SRAM and DRAM Errors - More Data From the Field,” *Silicon Errors in Logic - System Effects (SELSE-10), Stanford University*, April 1, 2014.

[65] P. Ellerman, “Calculating Reliability using FIT & MTTF: Arrhenius HTOL Model,” tech. rep., 2012. "Online available under: `http://www.microsemi.com/document-portal/doc_view/124041-calculating-reliability-using-fit-mttf-arrhenius-htol-model`.

[66] Wikipedia, “Jaguar Supercomputer.” Website. Online available under: `https://en.wikipedia.org/wiki/Jaguar_(supercomputer)`; retrieved on December 2016.

[67] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, “Memory errors in modern systems: The good, the bad, and the ugly,” *SIGPLAN Not.*, vol. 50, pp. 297–310, Mar. 2015.

[68] H. Anzt, V. Heuveline, and B. Rocker, “Mixed Precision Iterative Refinement Methods for Linear Systems: Convergence Analysis Based on Krylov Subspace Methods,” in *Proceedings of the 10th International Conference on Applied Parallel and Scientific Computing - Volume 2*, PARA’10, (Berlin, Heidelberg), pp. 237 — 247, Springer, 2012.

[69] G. H. Golub and C. F. Van Loan, *Matrix Computations (3rd Ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[70] P. G. Bridges, K. B. Ferreira, M. A. Heroux, and M. Hoemmen, “Fault-tolerant linear solvers via selective reliability,” tech. rep., 2012. arXiv:1206.1390v1.

[71] J. van den Eshof and G. L. G. Sleijpen, “Inexact Krylov Subspace Methods for Linear Systems,” *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 1, pp. 125 – 153, 2004.

[72] C. Vuik, “New Insights in GMRES-like Methods with Variable Preconditioners,” *J. Comput. Appl. Math.*, vol. 61, pp. 189 – 204, July 1995.

[73] V. Simoncini and D. B. Szyld, “On the Occurrence of Superlinear Convergence of Exact and Inexact Krylov Subspace Methods,” *SIAM Review*, vol. 47, no. 2, pp. 247–272, 2005.

[74] O. Axelsson, *Iterative Solution Methods*. New York, NY, USA: Cambridge University Press, 1994.

[75] J. E. Flaherty, “Block Tridiagonal Systems.” Website, 2005. Online available under: `http://www.cs.rpi.edu/~flaherje/pdf/lin10.pdf`; retrieved on Januar 2016.

[76] Wikipedia, “Symmetric matrix.” Online available under: `https://en.wikipedia.org/wiki/Symmetric_matrix`; retrieved on December 2016.

[77] Netlib, “GMRES Code.” Listening. Details of this algorithm are described in: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, Online available under: `http://www.netlib.org/templates/matlab/gmres.m`; retrieved on December 2016.

12 Parameters

Figure 17, 18, diagonal matrix, first value = 1, last value controlled, size = 1000×1000, $||r||_2 \leq 10^{-9}$
Figure 19, 20, diagonal matrix, first value = 1, last value controlled, size = 1000×1000, $||r||_2 \leq 10^{-9}$
Figure 21, 22, diagonal matrix, first value = 1, last value controlled, size = 1000×1000, $||r||_2 \leq 10^{-9}$
Figure 23, 24, diagonal matrix, first value = 1, last value controlled, size = 1000×1000, $||r||_2 \leq 10^{-9}$
Figure 25, diagonal matrix, first value = 1, last value controlled, size = 1000×1000, $||r||_2 \leq 10^{-9}$

Figure 26, diagonal matrix, randomized values, size = 1000×1000, $||r||_2 \leq 10^{-9}$
Figure 29, 30, diagonal matrix, first value = 1, last value = 142.5, size = 1000×1000, $||r||_2 \leq 10^{-9}$
Figure 31, 32, diagonal matrix, first value = 1, last value = 142.5, size = 1000×1000, $||r||_2 \leq 10^{-9}$
Figure 34, diagonal matrix, first value = 1, last value = 142.5, size = 1000×1000, $||r||_2 \leq 10^{-9}$
Figure 35, 36, diagonal matrix, first value = 1, last value = 1425, size = 1000×1000, $||r||_2 \leq 10^{-9}$
Figure 37, 38, diagonal matrix, first value = 1, last value = 14250, size = 1000×1000, $||r||_2 \leq 10^{-9}$
Figure 41, 42, diagonal matrix, first value = 1, last value = 142.5, size = 1000×1000, $||r||_2 \leq 10^{-9}$
Figure 43, 44, diagonal matrix, first value = 1, last value = 1425, size = 1000×1000, $||r||_2 \leq 10^{-9}$
Figure 46, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 47, adder_dcop_63 matrix, size = 1813×1813, $||r||_2 \leq 10^{-9}$
Figure 48, 49, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 50, 51, adder_dcop_63 matrix, size = 1813×1813, $||r||_2 \leq 10^{-9}$
Figure 52, 53, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 54, 55, adder_dcop_63 matrix, size = 1813×1813, $||r||_2 \leq 10^{-9}$
Figure 56, 57, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 58, 59, adder_dcop_63 matrix, size = 1813×1813, $||r||_2 \leq 10^{-9}$
Figure 62, 63, 64, 65, 66, 67, 68, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 69, 70, 71, 72, adder_dcop_63 matrix, size = 1813×1813, $||r||_2 \leq 10^{-9}$
Figure 74, 76, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 75, 77, adder_dcop_63 matrix, size = 1813×1813, $||r||_2 \leq 10^{-9}$
Figure 78, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 79, adder_dcop_63 matrix, $||r||_2 \leq 10^{-9}$
Figure 80, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 81, adder_dcop_63 matrix, $||r||_2 \leq 10^{-9}$
Figure 82, 83, 84, 85, 86, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 87, 88, 89, 90, 91, adder_dcop_63 matrix, $||r||_2 \leq 10^{-9}$
Figure 92, 2D Poisson matrix & adder_dcop_63 matrix, $||r||_2 \leq 10^{-9}$
Figure 93, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 94, Pres_Poisson, size = 14822×14822, $||r||_2 \leq 10^{-6}$
Figure 95, Kuu matrix, size = 7102×7102, $||r||_2 \leq 10^{-9}$
Figure 96, Na5 matrix, size = 5832×5832, $||r||_2 \leq 10^{-9}$
Figure 97, adder_dcop_63 matrix, size = 1813×1813, $||r||_2 \leq 10^{-9}$
Figure 98, circuit_2 matrix, size = 4510×4510, $||r||_2 \leq 10^{-9}$
Figure 99, mult_dcop_03, size = 25187×25187, $||r||_2 \leq 10^{-9}$
Figure 100, chem_master1 matrix, size = 40401×40401, $||r||_2 \leq 10^{-5.5}$
Figure 101, Ill_Stokes matrix, size = 20896×20896, $||r||_2 \leq 10^{-6}$
Figure 102, Pres_Poisson, size = 14822×14822, $||r||_2 \leq 10^{-6}$
Figure 103, Kuu matrix, size = 7102×7102, $||r||_2 \leq 10^{-9}$
Figure 104, Na5 matrix, size = 5832×5832, $||r||_2 \leq 10^{-9}$
Figure 105, circuit_2 matrix, size = 4510×4510, $||r||_2 \leq 10^{-9}$
Figure 106, chem_master1 matrix, size = 40401×40401, $||r||_2 \leq 10^{-5.5}$
Figure 107, Pres_Poisson, size = 14822×14822, $||r||_2 \leq 10^{-6}$
Figure 108, Kuu matrix, size = 7102×7102, $||r||_2 \leq 10^{-9}$
Figure 109, Na5 matrix, size = 5832×5832, $||r||_2 \leq 10^{-9}$
Figure 110, circuit_2 matrix, size = 4510×4510, $||r||_2 \leq 10^{-9}$
Figure 111, mult_dcop_03, size = 25187×25187, $||r||_2 \leq 10^{-9}$
Figure 112, chem_master1 matrix, size = 40401×40401, $||r||_2 \leq 10^{-5.5}$

Figure 113, Ill_Stokes matrix, size = 20896×20896, $||r||_2 \leq 10^{-6}$
Figure 114, 115, circuit_2 matrix, size = 4510×4510, $||r||_2 \leq 10^{-9}$
Figure 116, 117, mult_dcop_03 matrix, size = 25187×25187, $||r||_2 \leq 10^{-9}$
Figure 118, 119, Ill_Stokes matrix, size = 20896×20896, $||r||_2 \leq 10^{-6}$
Figure 122, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 123, Pres_Poisson, size = 14822×14822, $||r||_2 \leq 10^{-6}$
Figure 124, Kuu matrix, size = 7102×7102, $||r||_2 \leq 10^{-9}$
Figure 125, 126, adder_dcop_63 matrix, size = 1813×1813, $||r||_2 \leq 10^{-9}$
Figure 127, circuit_2 matrix, size = 4510×4510, $||r||_2 \leq 10^{-9}$
Figure 128, mult_dcop_03, size = 25187×25187, $||r||_2 \leq 10^{-9}$
Figure 129, chem_master1 matrix, size = 40401×40401, $||r||_2 \leq 10^{-5.5}$
Figure 130, 131, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 132, 133, 134, 135, 136, 137, diagonal matrix, first value = 1, last value = 1425, $||r||_2 \leq 10^{-9}$
Figure 139, 140, 141, 142, 143, 144, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 145, 146, 147, 148, 149, 150, adder_dcop_63 matrix, size = 1813×1813, $||r||_2 \leq 10^{-9}$
Figure 151, 152, 153, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 158, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
Figure 159, Pres_Poisson, size = 14822×14822, $||r||_2 \leq 10^{-6}$
Figure 160, Kuu matrix, size = 7102×7102, $||r||_2 \leq 10^{-9}$
Figure 161, Na5 matrix, size = 5832×5832, $||r||_2 \leq 10^{-9}$
Figure 162, adder_dcop_63 matrix, size = 1813×1813, $||r||_2 \leq 10^{-9}$
Figure 163, circuit_2 matrix, size = 4510×4510, $||r||_2 \leq 10^{-9}$
Figure 164, mult_dcop_03, size = 25187×25187, $||r||_2 \leq 10^{-9}$
Figure 165, chem_master1 matrix, size = 40401×40401, $||r||_2 \leq 10^{-5.5}$
Figure 166, Ill_Stokes matrix, size = 20896×20896, $||r||_2 \leq 10^{-6}$
Figure 167, 168, 169, 170, 171, 2D Poisson matrix, size = 10000×10000,$||r||_2 \leq 10^{-9}$
Figure 172, 173, 174, 175, 176, 2D Poisson matrix, size = 10000×10000, $||r||_2 \leq 10^{-9}$
$||r||_2 \ldots$ maximum allowable relative residuum for the solution vector x ($||r||_2 = ||Ax - b||_2/||b||_2$)

For all trials and tests the right hand side b is always computed with $b = Ax_{solution}$ and $x_{solution} = ones()$ (a vector with ones), the outer solver is always initialized with $x_{start} = zeros()$ (zero values). All experiments in this master work are done with Matlab and Python, Python is used to visualize.

13 Appendix

13.1 Formulas for computing the number of flops (Matlab)

```
%faktor...(input) how much multiplikations and addtions for each non-zero element of matrix A,
%nnz...(input) number of non-zeros of matrix A,
%n...(input) length of solution vector x,
%m...(input) number of iterations,
%m1...(input) number of outer iterations of F(T)-GMRES, %m2...(input) number of inner iterations of F(T)-GMRES,
%flops...(output) number of floating point operations at iteration m (m1)

% Flops of the standard GMRES solver
function [flops] = gmres_flops(m,n,nnz,faktor)
 flops=faktor*m*nnz+2*m^(2)*n+3*m^(2)+m^(2)+2*m*n;
end

% Flops of the flexible GMRES (F(T)-GMRES) solver for a fixed number of inner iterations (m2)
function [flops] = gmres_flexible_flops(m1,m2,n,nnz,faktor)
 flops= gmres_flops(m1,n,nnz,faktor)+(m1)*gmres_flops(m2,n,nnz,faktor)+faktor*nnz*(m1+1)+(m1+1)*n;
end

```

```
% Flops of the preconditioned GMRES solver for a fixed number of iterations (m)
% faktor2 are the number of operations which have to be done for forward and backward substitution
function [flops] = gmres_precon_flops(nnz,m,n,num,faktor,faktor2)
flops=gmres_flops(nnz,m,n,faktor)+m*faktor2;
end

% Flops of the preconditioned flexible GMRES (F(T)-GMRES) solver for a fixed number of inner iterations (m2)
function [flops] = gmres_flexible_precon_flops(m1,m2,n,nnz,faktor,faktor2)
 flops= gmres_flops(m1,n,nnz,faktor)+(m1)*gmres_precon_flops(m2,n,nnz,faktor,faktor2)+faktor*nnz*(m1+1)+(m1+1)*n;
end

% Flops of the flexible GMRES (F(T)-GMRES) solver for different iterations of the inner solver
%m is just an array which stores the number of inner iterations  (m2) for each execution of it
function [flops] = gmres_flexible_flops(k,nnz,m,n,faktor)
flops=[]

%Flops of flexible GMRES (F(T)-GMGRES) for a fixed number of iterations for the inner solver
if length(m)==1
    flops= gmres_flops(nnz,k,n,faktor)+k*gmres_flops(nnz,m,n,faktor)+faktor*nnz*(k+1)+(k+1)*n;
end

%Flops of flexible GMRES (FT-GMGRES) with variable numbers of iterations for the inner solver
if length(m)>1
    flops_= gmres_flops(nnz,k,n,faktor);
    flops__=0;
    for i=1:k
        flops__=flops__+gmres_flops(nnz,m(i),n,faktor);
    end
    flops=flops_+flops__+(k+1)*n+faktor*nnz*(k+1);
end
end

% Flops of the standard conjugate gradient (CG) solver
function [flops] = conjugate_flops(m,n,nnz,faktor)
 flops=faktor*m*nnz+m*(10*n)+n;
end

% Flops of the flexible conjugate gradient (F-CG) solver for a fixed number of inner iterations (m2)
function [flops] = conjugate_flexible_flops(m1,m2,n,nnz,faktor)
 flops=conjugate_flops(m1,n,nnz,faktor)+ (m1)*conjugate_flops(m2,n,nnz,faktor)+faktor*(m1+1)*nnz+(m1+1)*n;
end
```

13.2 Preconditioned GMRES algorithm [77]

```
function [x, error, iter, flag] = gmres( A, x, b, M, restrt, max_it, tol)
% -- Iterative template routine --
%      Univ. of Tennessee and Oak Ridge National Laboratory
%      October 1, 1993
%      Details of this algorithm are described in "Templates for the
%      Solution of Linear Systems: Building Blocks for Iterative
%      Methods", Barrett, Berry, Chan, Demmel, Donato, Dongarra,
%      Eijkhout, Pozo, Romine, and van der Vorst, SIAM Publications,
%      1993. (ftp netlib2.cs.utk.edu; cd linalg; get templates.ps).
% [x, error, iter, flag] = gmres( A, x, b, M, restrt, max_it, tol )
% gmres.m solves the linear system Ax=b using the Generalized Minimal residual method with restarts.
% input    A          REAL nonsymmetric positive definite matrix
%          x          REAL initial guess vector
%          b          REAL right hand side vector
%          M          REAL preconditioner matrix
%          restrt     INTEGER number of iterations between restarts
%          max_it     INTEGER maximum number of iterations
%           tol       REAL error tolerance
% output    x         REAL solution vector
%           error     REAL error norm
%           iter      INTEGER number of iterations performed
%           flag      INTEGER: 0 = solution found to tolerance, 1 = no convergence given max_it
    iter = 0;                                          % initialization
    flag = 0;
    bnrm2 = norm( b );
    if  ( bnrm2 == 0.0 ), bnrm2 = 1.0; end
    r = M \ ( b-A*x ); error = norm( r ) / bnrm2;
    if ( error < tol ) return, end
    [n,n] = size(A);                                   % initialize workspace
    m = restrt;
    V(1:n,1:m+1) = zeros(n,m+1);
    H(1:m+1,1:m) = zeros(m+1,m);
    cs(1:m) = zeros(m,1);
    sn(1:m) = zeros(m,1);
    e1     = zeros(n,1);
    e1(1) = 1.0;
    for iter = 1:max_it,                               % begin iteration
        r = M \ ( b-A*x );
        V(:,1) = r / norm( r );
        s = norm( r )*e1;
        for i = 1:m,                                   % construct orthonormal
      w = M \ (A*V(:,i));                          % basis using Gram-Schmidt
      for k = 1:i,
        H(k,i)= w'*V(:,k);
        w = w - H(k,i)*V(:,k);
      end
      H(i+1,i) = norm( w );
      V(:,i+1) = w / H(i+1,i);
      for k = 1:i-1,                               % apply Givens rotation
               temp      =  cs(k)*H(k,i) + sn(k)*H(k+1,i);
               H(k+1,i) = -sn(k)*H(k,i) + cs(k)*H(k+1,i);
               H(k,i)    = temp;
      end
      [cs(i),sn(i)] = rotmat( H(i,i), H(i+1,i) ); % form i-th rotation matrix
           temp   = cs(i)*s(i);                        % approximate residual norm
           s(i+1) = -sn(i)*s(i);
      s(i)   = temp;
           H(i,i) = cs(i)*H(i,i) + sn(i)*H(i+1,i);
           H(i+1,i) = 0.0;
      error  = abs(s(i+1)) / bnrm2;
      if ( error <= tol ),                         % update approximation
         y = H(1:i,1:i) \ s(1:i);                  % and exit
              x = x + V(:,1:i)*y;
         break;
      end
        end
        if ( error <= tol ), break, end
        y = H(1:m,1:m) \ s(1:m);
        x = x + V(:,1:m)*y;                            % update approximation
        r = M \ ( b-A*x )                              % compute residual
        s(i+1) = norm(r);
        error = s(i+1) / bnrm2;                        % check convergence
        if ( error <= tol ), break, end;
    end
    if ( error > tol ) flag = 1; end;                  % converged
% END of gmres.m
```

13.3 Faulting with FT-GMRES while solving the 2D Poisson matrix

13.3.1 Inner solver initialized with the previous computed basis vector ($w_j(w_0) = w_{j-1}$)

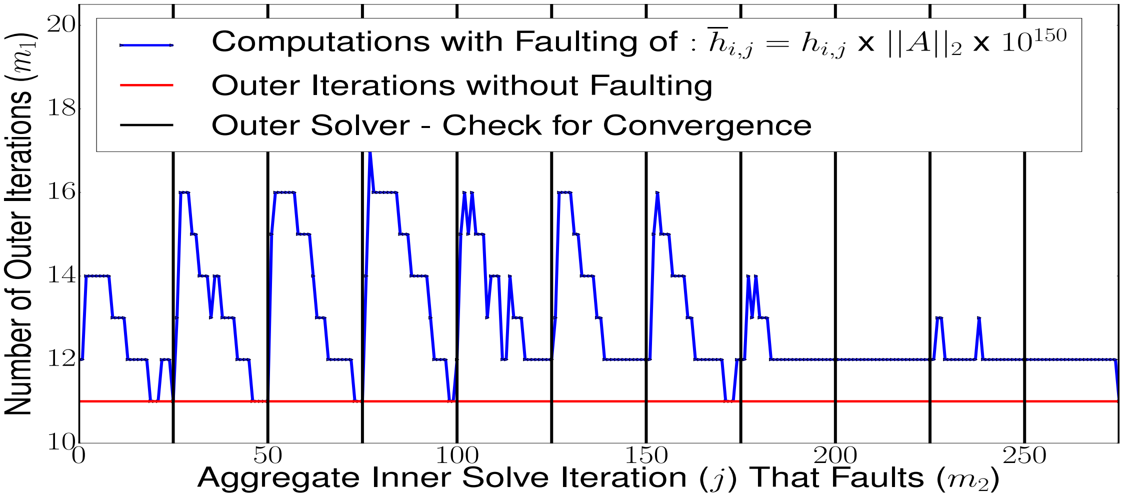


Figure 167 Faulting with error type 1 on the first MGS-iteration ($h_{i=1,j}$) during solving the 2D Poisson matrix for $i = 1$ and $j = 1, \ldots, 25$.

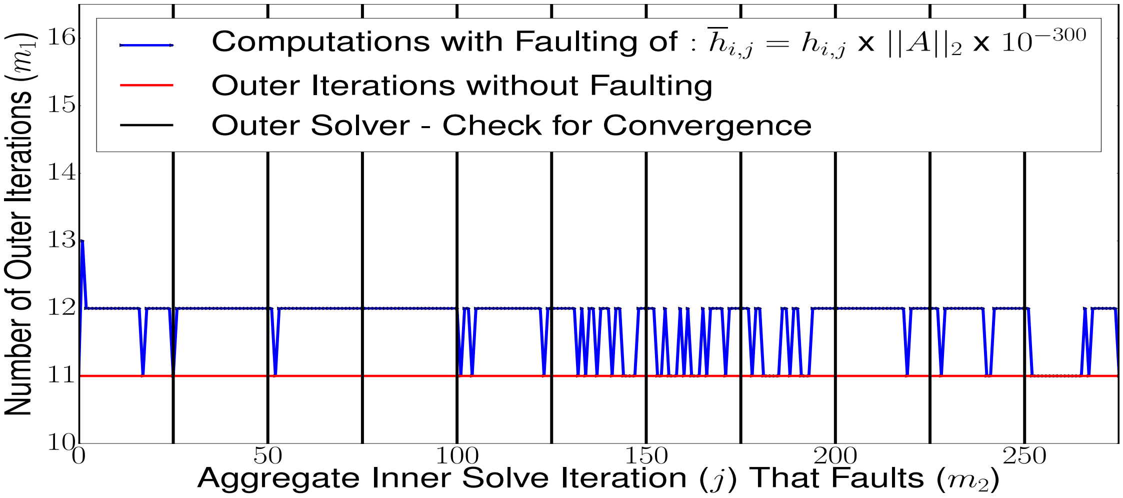


Figure 168 Faulting with error type 2 on the first MGS-iteration ($h_{i=1,j}$) during solving the 2D Poisson matrix for $i = 1$ and $j = 1, \ldots, 25$.

13.3.2 Inner solver initialized with random values ($w_j(w_0) = randn()$)

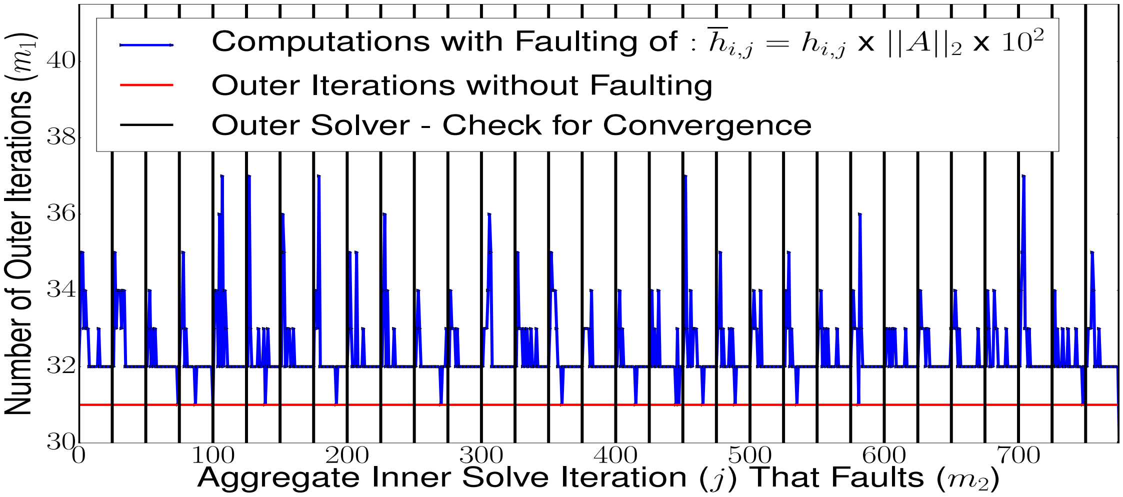


Figure 169 Faulting with error type 1 on the first MGS-iteration ($h_{i=1,j}$) during solving the 2D Poisson matrix for $i = 1$ and $j = 1, \ldots, 25$.

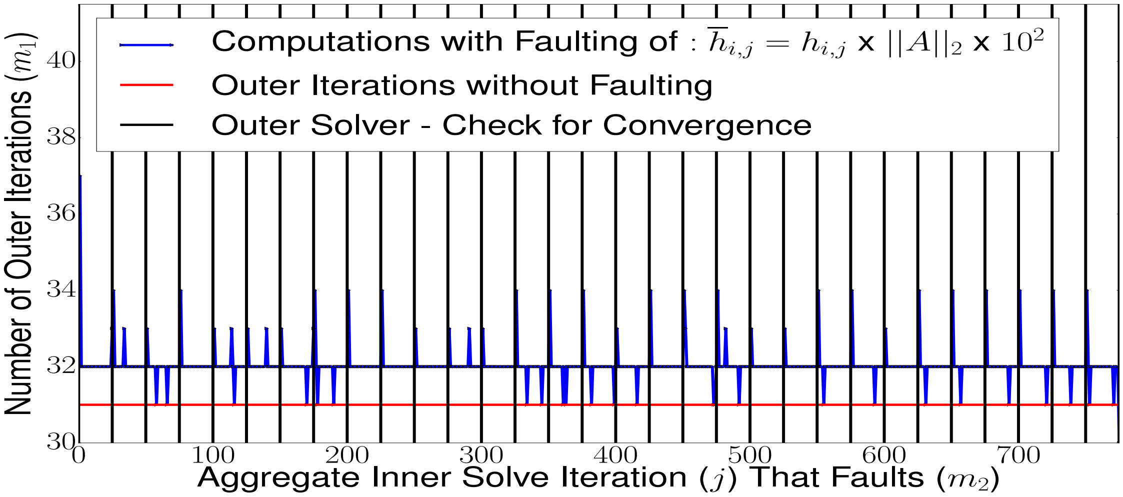


Figure 170 Faulting with error type 2 on the first MGS-iteration ($h_{i=1,j}$) during solving the 2D Poisson matrix for $i = 1$ and $j = 1, \ldots, 25$.

13.3.3 Residuum curves for solving the 2D Poisson matrix

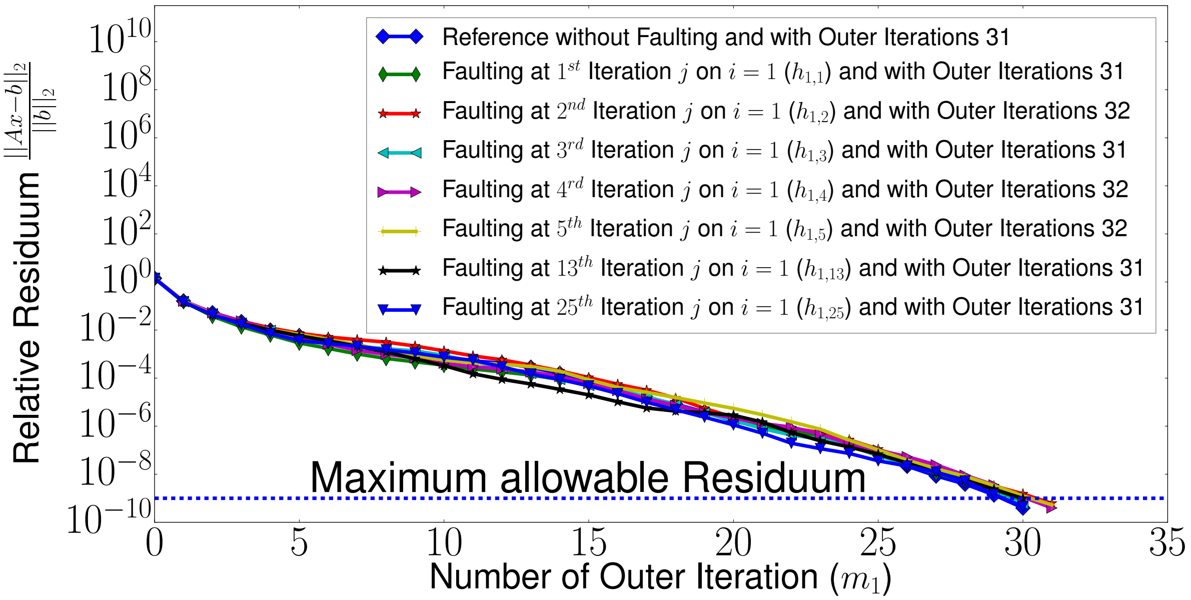


Figure 171 Trend of the explicit residuum and faulting with error type 1 during solving the 2D Poisson matrix. Faulting at 5^{th} outer iteration. Inner solver initialized with random values.

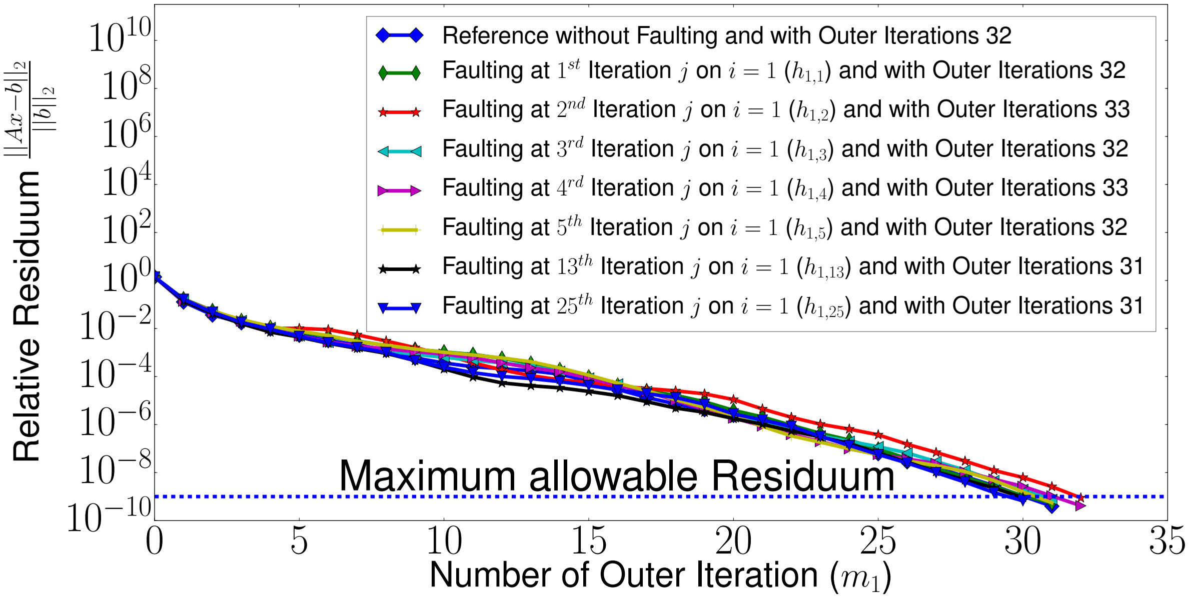


Figure 172 Trend of the explicit residuum and faulting with error type 2 during solving the 2D Poisson matrix. Faulting at 5^{th} outer iteration. Inner solver initialized with random values.

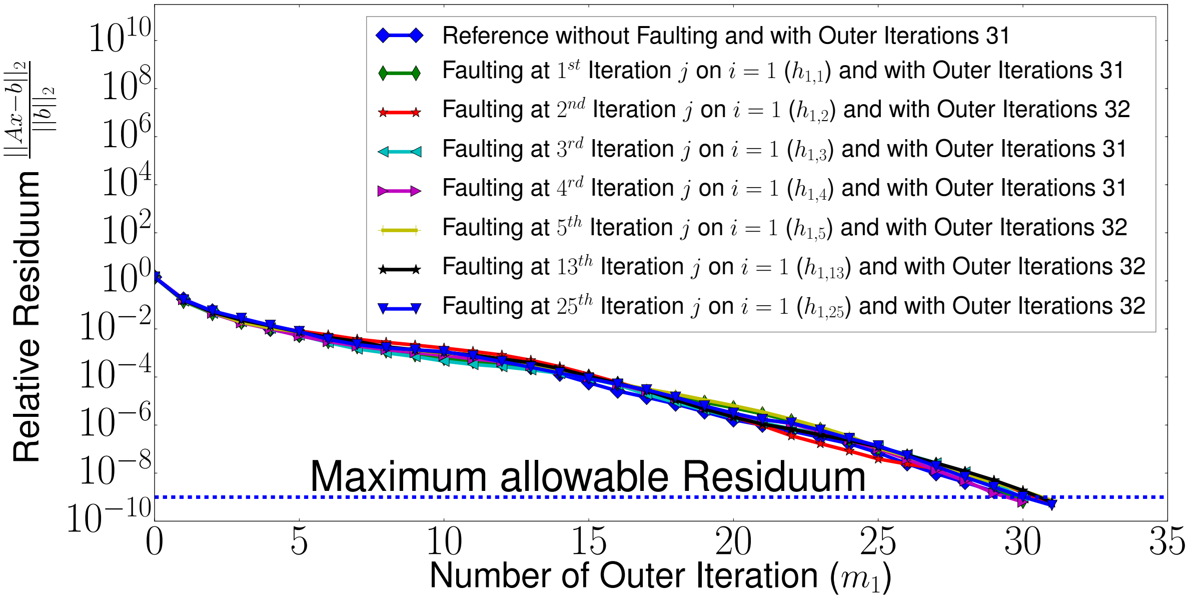


Figure 173 Trend of the explicit residuum and faulting with error type 1 during solving the 2D Poisson matrix. Faulting at 15^{th} outer iteration. Inner solver initialized with random values.

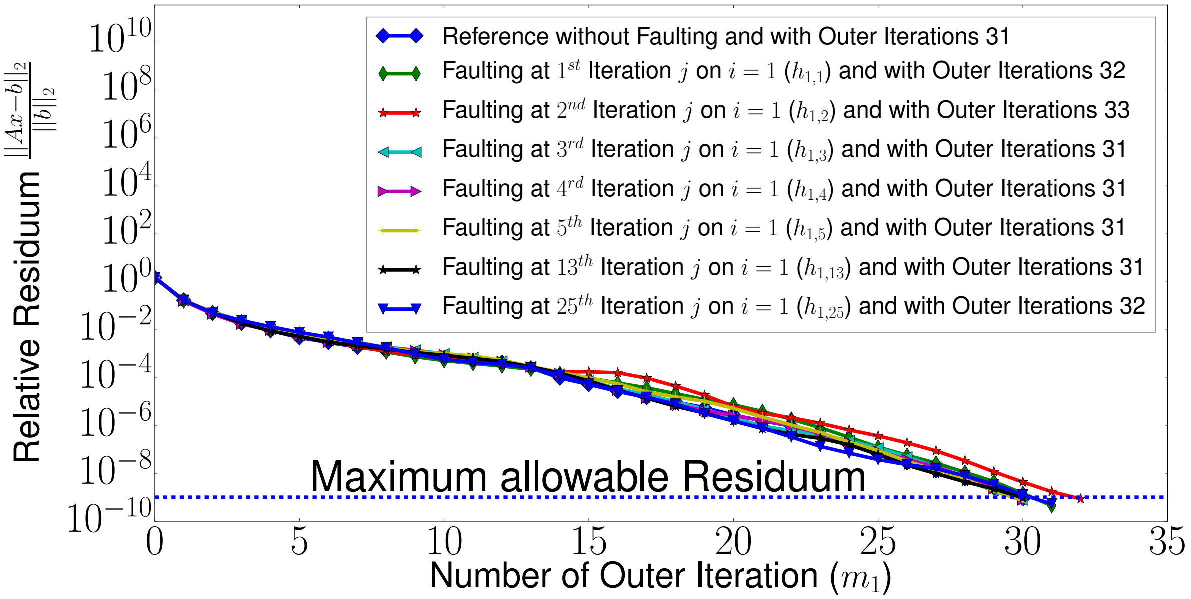


Figure 174 Trend of the explicit residuum and faulting with error type 2 during solving the 2D Poisson matrix. Faulting at 15^{th} outer iteration. Inner solver initialized with random values.

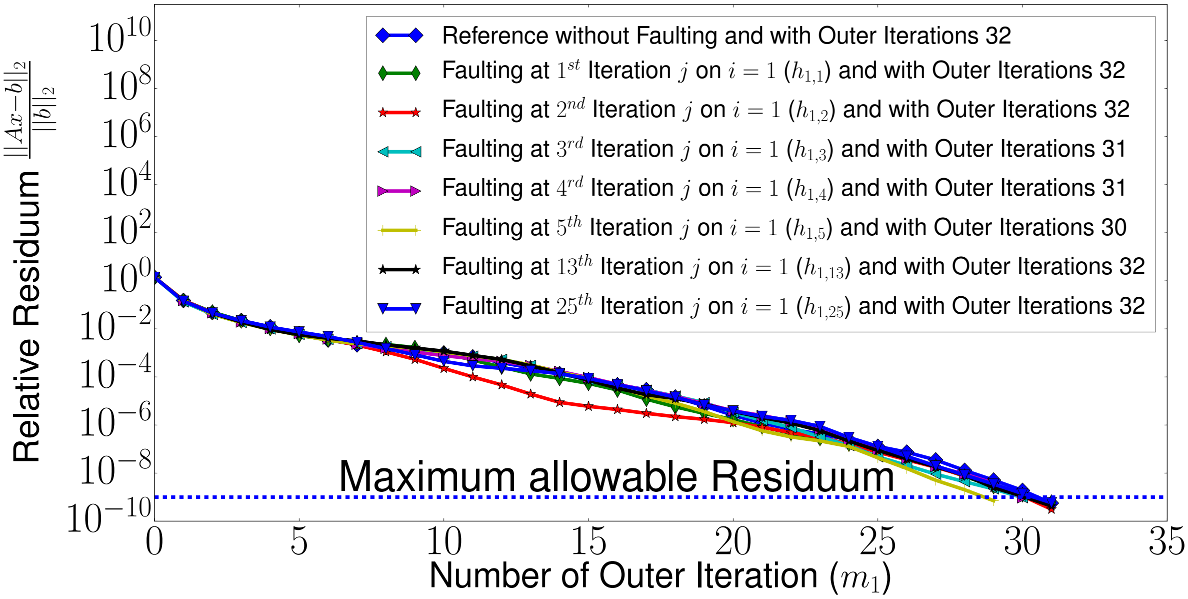


Figure 175 Trend of the explicit residuum and faulting with error type 1 during solving the 2D Poisson matrix. Faulting at 30^{th} outer iteration. Inner solver initialized with random values.

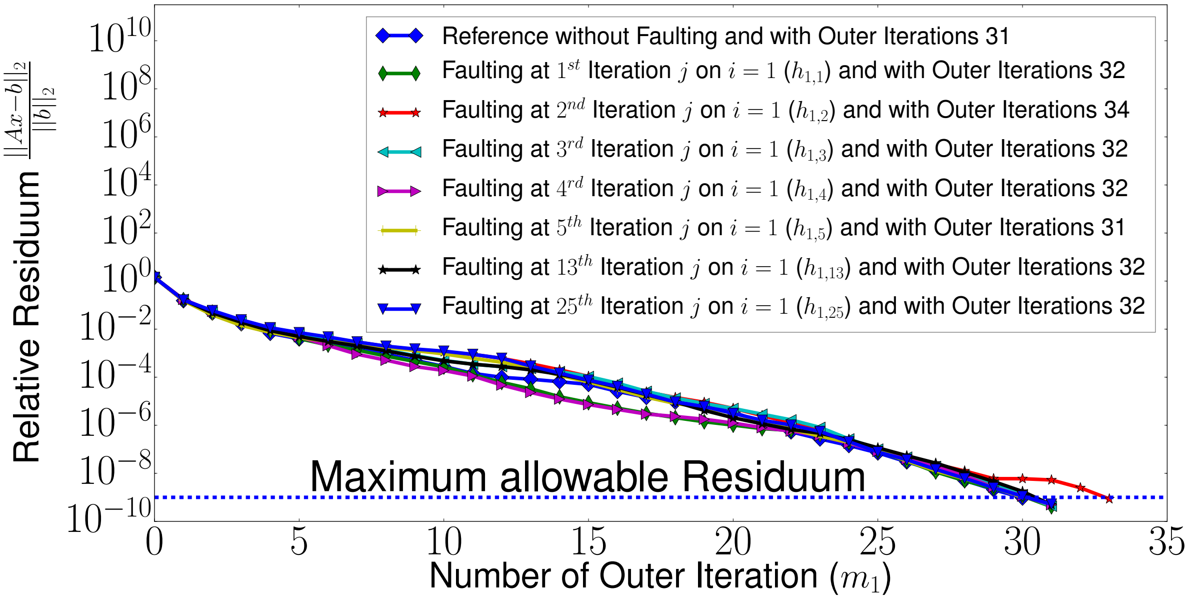


Figure 176 Trend of the explicit residuum and faulting with error type 2 during solving the 2D Poisson matrix. Faulting at 30^{th} outer iteration. Inner solver initialized with random values.

13.4 Abstract (German - short version)

Iterative Gleichungslöser geben die Möglichkeit grosse Gleichungssysteme effizient zu lösen. Es wird ein Vektor x gesucht um das Problem $Ax = b$ zu berechnen für eine Matrix A wobei diese wiederum dünn besetzt ist. Für spärlich belegte Matrizen gilt dass diese zwar sehr gross sind, aber nur sehr wenige Elemente haben die ungleich Null sind. Während des Lösungsvorganges in Computersystemen kann es zu sogenannten Bitflips im Hauptspeicher kommen, welche das Ergebnis des gewählten Gleichungslöser verfälschen und daher die berechnete Lösung x nicht korrekt sein kann. Diese Fehler im Hauptspeicher werden durch verschiedene Methoden erkannt. Wie zum Beispiel durch den sogenannten ECC (Error Correcting Code) - Speicher welcher Bitflips erkennen und korrigieren kann.

Diese Methoden führen aber wiederum zu einem erhöhten Energieverbrauch des gesamten Computersystems. Es gibt daher verschiedene Ansätze um diesen Trend entgegen zu wirken. Eine Möglichkeit ist es zwei Gleichungslöser zu kombinieren, wobei dieser aus einen äusseren und inneren Löser besteht. Im Inneren dürfen wiederum Bitflips passieren, wobei der Äussere das Ergebnis überprüft ob das Gleichungssystem $Ax = b$ gelöst und der richtige Vektor x berechnet wurde.

Im Fall des FT-GMRES (Fault Tolerant GMRES) gibt es einen inneren und äusseren Gleichungslöser. Dieser besteht aus zwei GMRES Lösern welcher wiederum das Problem $Ax = b$ für verschiedene Arten von Matrizen berechnen kann. Diese Arbeit beschreibt vor allem wie sich diese Fehler im inneren Gleichungslöser mit verschiedenen Arten von Bitflips auf den Äusseren auswirken. Ein weiteres Ziel dieser Arbeit ist wie Fehler auch erkannt werden können um den Nebenwirkungen von Bitflips entgegen zu wirken. Der FT-GMRES soll möglichst wenig durch Bitflips im inneren Löser beeinflusst werden.