D I P L O M A R B E I T

Automatic Solver Control for Linear Equation Systems

ausgeführt am
Institut für Mikroelektronik
der Technischen Universität Wien

unter Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Erasmus Langer
und
Projektass. Dipl.-Ing. Dr.techn. René Heinzl

durch

CLEMENS KLÖCKLER

Hollerweg 5-7/4/4
A-3430 Tulln, Österreich

Matr. Nr. 0027071
geboren am 4. Juni 1982, Wien

Tulln, im Mai 2009

_____

Kurzfassung

Das Ziel dieser Arbeit war es, eine automatische Steuerung für Gleichungslöser von linearen Gleichungssystemen zu entwickeln, die dem Anwender möglichst alle Entscheidungen bei der Lösung eines linearen Gleichungssystem abnehmen kann. Es soll, wenn möglich, den richtigen Gleichungslösertyp für eine Matrix auswählen, sowie eine geeignete Vorkonditionierung vornehmen. Falls der gewählte Löser nicht zum Ziel führt, sollten noch andere Lösungsvarianten ausgenutzt werden.
Der erste Teil der Arbeit beschäftigt sich mit dem notwendigen mathematischen Hintergrund von Matrizen. Außerdem werden die Lösungsalgorithmen von CG, BICGstab und GMRES erklärt und auch einige Vorkonditionierungsmethoden erläutert (Jacobi, unvollständige LR-Zerlegung und unvollständige Cholesky-Zerlegung). Um diese Algorithmen zu testen, wurden verschiedene Software Pakete, die eben diese anbieten, analysiert (Trilinos, PETSc, ITL, QQQ und Hypre), damit ein Pool an Gleichungslösern zur Verfügung gestellt werden kann. Um eine automatische Steuerung zu realisieren, braucht das Programm gewisse Matrixeigenschaften, auf denen die Entscheidungen fußen können. Dazu wurde ein Analysetool entwickelt, das eine Matrix analysiert und verschiedene wichtige Matrixeigenschaften zurückliefert (Bsp.: Symmetrie, Definitheit). Der Hauptteil der Arbeit beschäftigt sich mit dem Test verschiedener Matrizen, um zu sehen, welche Matrixeigenschaften als Entscheidungskriterien verwendet werden können. Aufgrund dieser Matrixeigenschaften wurden dann die Entscheidungen, die eine automatische Steuerung trifft, abgestimmt. Der letzte Teil der Arbeit zeigt das entwickelte Modul angewendet auf die gezeigten Testmatrizen, sowie den Lösungsstatus, und zugehörigen Residuen.

Abstract

This work describes the development of an automatic solver control for linear equations systems which was designed to help a user to find the optimal solver for a given matrix problem. The used solver algorithms were CG, BICGstab and GMRES, with the preconditioners Jacobi, incomplete LU factorization and incomplete Cholesky factorization. Several solver packages available on the market were tested for their performance (Trilinos, PETSc, QQQ, Hypre and ITL). In order for the automatic solver control to be able to make decisions, it needs to know several properties of a given matrix. For that reason an analyzing tool was developed. The main part of this work consists of a description of the testing of the different solver packages to find out how the various solvers perform and which matrix properties may have an influence on their performance. This testing helped in the adjustment of the decision criteria for the automatic solver control. The last part of this work describes the performance of the automatic solver control which was developed.

Acknowledgments

I want to thank Prof. Erasmus Langer, René Heinzl, and his team, Franz Stimpfl and Philipp Schwaha, for guiding and advising me through this work. Special thanks go to my study colleague Josef Weinbub, who was a helpful partner throughout the whole study especially with his knowledge of programming.

I want to thank my family for their support in all of my years of education since I started school at the age of six. I also want to thank all my friends who missed me during the last days of my completion of this work, especially Matthias and Ronny.

Very special thanks go to my beloved girlfriend, Elisabeth, who always had a shoulder to lean on in hard days.

dedicated to:

my little daughter Katharina
who lets me see the world through different eyes

Contents

Chapter 1

Introduction

Solving a linear equation system with a solver is a common task in computer science. Solvers are programs that are able to determine a solution for a mathematical problem, such as a linear equation system. An automatic solver control environment should be capable of evaluating the optimal approach to solve the problem statement. Additionally user interaction should be reduced to a minimum to keep the usage simple.

The major goal of this work is to implement a solver interface which is able to evaluate an equation system and process it accordingly. Due to the interface approach, several different software tools can be used. Each software tool provides a certain set of solvers and preconditioners. The automatic solver interface selects the solver which is likely to most efficiently solve this particular system of equations.

1.1 Motivation

The goal of this work is to develop an automatic solver control which tries to independently find the right options for solving an equation system. During the development of such a program several questions arise:

- Which different solver concepts are available? What are their advantages and disadvantages?
- What solver packages are available? What solvers and features do they offer? How is their performance?
- What can be done to improve the convergence of specific solvers (e.g., preconditioners)?
- Which matrix properties have an influence on the solvers and the preconditioners and therefore can help in making a decision?
- Does the solution fulfill the desired accuracy?

1.2 Organization

Chapter 2 offers the mathematical background of this work and discusses different properties a matrix may have (e.g., symmetry, positive definite). Then the solving concepts of direct (Gauss algorithm) and indirect solvers are introduced (Sections 2.2 and 2.3) with a deeper look at the indirect solver concepts **CG**, **BICGstab** and **GMRES**. One improvement (preconditioning) which can be used with an indirect

solver is discussed in Section 2.4, primarily investigating **Jacobi**, **incomplete LU factorization** and **incomplete Cholesky factorization**.
Chapter 3 discusses the five different solver packages that were tested: **Trilinos**, **PETSc**, **ITL**, **QQQ** and **Hypre**. Some of the functions the packages provide are introduced here along with the solving functions, and the special features of each package are also discussed.
The main part of the work consists of Chapters 4, 5 and 6. At first the development of a diagnostic tool which is able to analyze a matrix and find out its important properties is presented. Then a variety of solver packages are tested with different matrices: tridiagonal matrices (Section 5.1), Hilbert matrices (Section 5.2) and several sparse matrices resulting from scientific problems (Sections 5.3, 5.4, 5.5, 5.6). The tests are performed for two purposes: on the one hand to analyze the performance of the different solver packages, and on the other hand to investigate the effects of the matrix properties on different solver algorithms. The results of these tests are used to develop an automatic solver control which automatically decides which solver and preconditioner is chosen based on several matrix properties. The important tasks the program should fulfill, as well as tests to see whether the program is working, are discussed in Section 6.

Chapter 2

Linear Solvers

Before linear equation systems are analyzed, the mathematical background is introduced. First the mathematical toolkit that is needed for solving is described, including some matrix properties that may influence the solving. Then the solving algorithms are introduced.

2.1 Mathematical Toolkit

2.1.1 Vector Norm

Definition 1 (Norm) *Let V be a vector space over the field $\mathbb{R}$. A given norm of a vector $\mathbf{x} \in V$ needs to fulfill the properties [1]:*

- $\|\mathbf{x}\| \geq 0$, if $\|\mathbf{x}\| = 0 \Rightarrow \mathbf{x} = \mathbf{0}, \forall \mathbf{x} \in V$
- $\|\alpha \cdot \mathbf{x}\| = |\alpha| \cdot \|\mathbf{x}\|, \forall \mathbf{x} \in V, \forall \alpha \in \mathbb{R}$
- $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|, \forall \mathbf{x}, \mathbf{y} \in V$

One specific vector norm is the **Euclidean norm** which is used for this work [1].

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}, \quad \forall \mathbf{x} \in \mathbb{R}^n \tag{2.1}$$

2.1.2 Symmetry

Definition 2 (Symmetry) *A real-valued matrix **A** is called **symmetric** if it is equal to its transpose $\mathbf{A}^T$[1] [2].*

$$\mathbf{A} = \mathbf{A}^T \tag{2.2}$$

[1] The transpose is built by writing the rows of $\mathbf{A}$ as columns of $\mathbf{A}^T$

2.1.3 Eigenvalues

The **eigenvalues** of a real-valued matrix are all values λ which fulfill the equation [2]:

$$\mathbf{Ax} = \lambda\mathbf{x} \tag{2.3}$$

2.1.4 Positive Definite

Definition 3 (Positive Definite) *A symmetric real-valued matrix **A** is called **positive definite** if all eigenvalues of the matrix are positive [3]. A non-symmetric matrix is called **positive definite** if its symmetric part is positive definite.*

The symmetric part of a matrix can be built via:

$$\mathbf{A}_{symm} = \frac{1}{2}(\mathbf{A} + \mathbf{A}^T) \tag{2.4}$$

2.1.5 Gershgorin Circle Theorem

An estimation of the eigenvalues of a matrix can be made with the **Gershgorin circle theorem** [4]. If the matrix has only real values and is symmetric, the theorem simplifies to:

$$\begin{gathered}\mathbf{A} \in \mathbb{R}^{n\times n} \text{ with entries } a_{ij} \\ D_i[a_{ii} - R_i, a_{ii} + R_i], i = 1, ..., n \\ \text{with } R_i = \sum_{j=1, j\neq i}^{n} |a_{ij}|\end{gathered}$$

The closed sets D_i are called **Gershgorin discs** (Gershgorin circles). The theorem states that every eigenvalue resides within at least one of these discs.
If the off-diagonal matrix entries are small, it is possible to get a good approximation of the eigenvalues. Sometimes it is only of interest whether one of the eigenvalues is negative (e.g., for the definite). This may be possible to determine even if the non-diagonal entries have huge numbers.

2.1.6 Pivot Element

The **pivot element** of a matrix is the first element chosen by an algorithm. Zero pivot elements can lead to numerical instability of some algorithms [3]. For that reason it is necessary to guarantee that the pivot elements are not zero.

2.1.7 Krylov Subspace

Definition 4 (Krylov Subspace) *The **Krylov subspace** is generated by an $n \times n$ matrix **A** and a vector **b** with the length n [5]. It is represented by the span[2] of the vectors **b**, **Ab**, $\mathbf{A}^2$**b**, ..., $\mathbf{A}^{n-1}$**b**:*

$$\mathcal{K}_n = span\{\mathbf{b}, \mathbf{Ab}, \mathbf{A}^2\mathbf{b}, ..., \mathbf{A}^{n-1}\mathbf{b}\} \tag{2.5}$$

This subspace is used in algorithms to find the solution for a linear equation system.

[2]The span of a set of vectors is the set of all linear combinations that can be built with these vectors.

2.1.8 Linear Equation System

A linear equation system can be formulated as [1]:

$$\mathbf{Ax} = \mathbf{b} \tag{2.6}$$

If the matrix $\mathbf{A}$ has full rank[3] its inverse $\mathbf{A}^{-1}$ exists and there is only one single solution for the linear equation system.

$$\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{A}^{-1}\mathbf{b} \tag{2.7}$$

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \tag{2.8}$$

2.1.9 Condition Number

If a linear equation system is altered the **condition number** is a measure for the variation of the solution occurring if one element in the right-hand side vector is changed [4]:

$$\kappa(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\| \tag{2.9}$$

The smallest condition number for a matrix is one. The value of the condition number depends on the matrix norm used. If the condition number is $\gg 1$ the matrix is called ill-conditioned. A good condition leads to a small change in the solution; an ill condition leads to a large change.

If the **Euclidian**[4] **norm** is used, the condition number can be determined via the eigenvalues:

$$\kappa(\mathbf{A}) = \left| \frac{\lambda_{max}(\sqrt{\mathbf{A}\mathbf{A}^T})}{\lambda_{min}(\sqrt{\mathbf{A}\mathbf{A}^T})} \right| \tag{2.10}$$

2.1.10 Residual

If a solution $\mathbf{x}$ for a linear system is determined, the **residual** can be calculated by [6]:

$$\mathbf{r} = \mathbf{b} - \mathbf{Ax} \tag{2.11}$$

The norm of the residual is a measure for the error of the solution. The two most common methods of calculating the error of the solution are based on the **absolute** residual and on the **relative** residual. Using the Euclidian norm the **absolute** residual is:

$$\|\mathbf{r}\|_2 \tag{2.12}$$

[3] All vectors of the matrix are linearly independent

[4] The spectral norm is induced by the l_2 vector norm. Thus $\|\mathbf{A}\| = \sqrt{\lambda_{max}(\mathbf{A}^T\mathbf{A})}$

On the other hand the **relative** residual is determined by dividing the absolute residual by the norm of the right-hand side vector **b**. Again for the Euclidian norm:

$$\frac{\|\mathbf{r}\|_2}{\|\mathbf{b}\|_2} \tag{2.13}$$

2.2 Direct Solvers

After introducing the mathematical background now the two different categories for solving are investigated: **direct** and **indirect** solvers.

Direct solvers use matrix transformation, e.g., into an upper triangular matrix.

2.2.1 Gauss algorithm

One example for a direct solver is the **Gauss elimination algorithm** [2]. This algorithm uses row addition and subtraction to obtain a triangular matrix. Hence the solution vector can be determined.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{1n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & ... & a_{nn} \end{bmatrix}$$

$$\Rightarrow$$

$$\hat{\mathbf{A}} = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ 0 & \hat{a}_{22} & ... & \hat{a}_{1n} \\ \vdots & & \ddots & \vdots \\ 0 & ... & 0 & \hat{a}_{nn} \end{bmatrix}$$

Let the vector obtained on the right side be:

$$\hat{\mathbf{b}} = \begin{bmatrix} \hat{b}_1 \\ \vdots \\ \hat{b}_n \end{bmatrix}$$

The solution vector can be calculated by:

$$
\begin{aligned}
x_n &= \frac{\hat{b}_n}{\hat{a}_{nn}} \\
x_{n-1} &= \frac{\hat{b}_{n-1} - \hat{a}_{n-1n} \cdot x_n}{\hat{a}_{n-1n-1}} \\
&\vdots \\
x_1 &= \frac{\hat{b}_1 - \hat{a}_{12} \cdot x_2 - \cdots - \hat{a}_{1n} \cdot x_n}{\hat{a}_{11}}
\end{aligned}
$$

The Gaussian elimination can be easily implemented:

Algorithm:

Let $k, i, j, n \in \mathbb{N}$; $\mathbf{A} \in \mathbb{R}^{n \times n}$; $\mathbf{b}, \mathbf{x} \in \mathbb{R}^n$

for $k = 1, ..., n$

exchange column k with the column were $max_{j=k}^{n} |a_{kj}|$

for $i = k + 1, ..., n$

for $j = k, k + 1, ..., n$

$$a_{ij} = a_{kj} \cdot a_{k+1,k} - a_{ij} \cdot a_{kk}$$

$$b_i = b_k \cdot a_{k+1,k} - b_i \cdot a_{kk}$$

for $k = n, n - 1, ..., 1$

for $i = k, k + 1, ..., n$

$$x_i = \frac{b_k - (\sum_{j=k+1}^{n} a_{ij} \cdot x_j)}{a_{kk}}$$

It uses $n(n + 1)/2$ divisions, $(2n^3 + 3n^2 - 5)/6$ multiplications and $(2n^3 + 3n^2 - 5)/6$ subtractions, which is a total of around $2n^3/3$ floating point operations. Therefore the computational effort of the Gaussian elimination algorithm is $\mathcal{O}(n^3)$ and becomes very big for large n. A better performance for large matrices is given by the indirect solvers introduced in the next section.

2.3 Indirect Solvers

Several indirect solvers are based on iterative methods. In each iterative step the initial matrix is used and the matrix is not changed during the solving process. One group of indirect solvers use the **Krylov subspace**. The basic idea of iterative methods based on the **Krylov** subspace is that in each iteration step a vector will be multiplied by the matrix to achieve a new vector in the subspace base. To build the whole subspace the n^{th} multiplication of the matrix needs to be **not** equal to zero. Iterative methods that are based on the **Krylov** subspace are, e.g., **CG**, **BICGstab** and **GMRES**.

2.3.1 Conjugate Gradient

The conjugate gradient (**CG**) algorithm is the simplest **Krylov** subspace method [6]. The convergence of the algorithm is only guaranteed for symmetric positive definite matrices. For other matrices the method can diverge and another solving algorithm has to be used.

CG:

Constants used:

$\mathbf{A} \in \mathbb{R}^{n \times n};\ \mathbf{b} \in \mathbb{R}^n$

Variables used:

$\mathbf{p_k},\ \mathbf{r_k},\ \mathbf{x_k},\ \mathbf{p_{k+1}},\ \mathbf{r_{k+1}},\ \mathbf{x_{k+1}} \in \mathbb{R}^n;\ \ a_k,\ b_k \in \mathbb{R}$

Algorithm:

Let $\mathbf{x_0} \in \mathbb{R}^n$

$\mathbf{p_0} = \mathbf{r_0} = \mathbf{b} - \mathbf{A x_0}$

for $k = 0, 1, ...$

if $\mathbf{p_k} = 0$ then $\mathbf{x_k}$ is the solution

else

$$a_k = \frac{\mathbf{r_k}^T \mathbf{r_k}}{\mathbf{p_k}^T \mathbf{A p_k}},\ \ \mathbf{x_{k+1}} = \mathbf{x_k} + a_k \mathbf{p_k}$$

$$\mathbf{r_{k+1}} = \mathbf{r_k} - a_k \mathbf{A p_k},\ \ b_k = \frac{\mathbf{r_{k+1}}^T \mathbf{r_{k+1}}}{\mathbf{r_k}^T \mathbf{r_k}}$$

$$\mathbf{p_{k+1}} = \mathbf{r_{k+1}} + b_k \mathbf{p_k}$$

Each iteration of the **CG** algorithm applied on a matrix of size $n \times n$ has the following computational complexity:

- 1 matrix-vector multiplication
- 4 scalar products
- 6n floating point operations

As additional input besides the matrix and right-hand side vector, an initial guess is necessary.

2.3.2 Stabilized Bi-Conjugated Gradient

BICGstab (stabilized Bi-conjugated gradient) is an indirect method to solve a linear equation system [6]. It is a method based on the **Krylov** subspace and it is an enhancement of **CG**. The matrix to be treated does not need to be symmetric, but it has to be positive definite. (Note that this fact is in contrast to **CG**, where the matrix requires both attributes.)

BICGstab:

Constants used:

$\mathbf{A} \in \mathbb{R}^{n \times n}$; $\mathbf{b}, \hat{\mathbf{r_0}} \in \mathbb{R}^n$ ($\hat{\mathbf{r_0}}^T \mathbf{r_k} \neq 0$ has to be fulfilled in each iteration step)

Variables used:

$\mathbf{p_k}$, $\mathbf{r_k}$, $\mathbf{x_k}$, $\mathbf{v}$, $\mathbf{p_{k+1}}$, $\mathbf{r_{k+1}}$, $\mathbf{x_{k+1}}$, $\mathbf{s}$, $\mathbf{t} \in \mathbb{R}^n$; a_k, b_k, $w_{k+1} \in \mathbb{R}$

Algorithm:

Let $\mathbf{x_0} \in \mathbb{R}^n$

$\mathbf{r_0} = \mathbf{b} - \mathbf{A x_0} \neq 0$

Let $\hat{\mathbf{r_0}} \in \mathbb{R}^n$ with $\hat{\mathbf{r_0}}^T \mathbf{r_0} \neq 0$

$\mathbf{p_0} = \mathbf{r_0}$

for $k = 0, 1, ...$

$$a_k = \frac{\hat{\mathbf{r_0}}^T \mathbf{r_k}}{\hat{\mathbf{r_0}}^T \mathbf{A p_k}}$$

$$\mathbf{v} = \mathbf{A p_k}, \quad \mathbf{s} = \mathbf{r_k} - a_k \mathbf{v}, \quad \mathbf{t} = \mathbf{A s}$$

$$w_{k+1} = \frac{\mathbf{s}^T \mathbf{t}}{\mathbf{t}^T \mathbf{t}}$$

$$\mathbf{x_{k+1}} = \mathbf{x_k} + a_k \mathbf{p_k} + w_{k+1} \mathbf{s}, \quad \mathbf{r_{k+1}} = \mathbf{s} - w_{k+1} \mathbf{t}$$

if $\|\mathbf{r_{k+1}}\|_2$ small enough then STOP

else

$$b_k = \frac{\hat{\mathbf{r_0}}^T \mathbf{r_{k+1}}}{\hat{\mathbf{r_0}}^T \mathbf{r_k}} \frac{a_k}{w_{k+1}}$$

$$\mathbf{p_{k+1}} = \mathbf{r_{k+1}} + b_k (\mathbf{p_k} - w_{k+1} \mathbf{v})$$

Each iteration of the **BICGstab** algorithm applied on a matrix of size $n \times n$ has the following computational complexity:

- 2 matrix-vector products
- 4 scalar products
- 12n floating point operations

The algorithm requires two initial vectors besides the matrix, where the second one has a big influence on the convergence of the algorithm (the equation $\hat{\mathbf{r_0}}^T \mathbf{r_k} \neq 0$ has to be fulfilled until convergence, or else the algorithm fails).

2.3.3 Generalized Minimal Residual Method

GMRES (generalized minimal residual method) is an iterative method for solving a linear equation system [6]. The algorithm builds an orthonormal basis in the **Krylov** subspace, which can be used to calculate the solution of the linear equation system. The matrix does not have to fulfill any special properties for the algorithm to work. However if the matrix is ill-conditioned the algorithm may fail. Analytically the algorithm always converges because in the last iteration step the base of the **Krylov** subspace is completed and all vectors can be reached.

GMRES:

Constants used:

$\mathbf{A} \in \mathbb{R}^{n \times n};\ \mathbf{x_0},\ \mathbf{b} \in \mathbb{R}^n$

Variables used:

$\mathbf{w_k},\ \mathbf{x},\ \in \mathbb{R}^n;\ i = 1, ..., k:\ \ \mathbf{v_i}\ \in \mathbb{R}^n$

$k = 1, ..., n:\ \ s_{k+1},\ c_{k+1},\ \gamma_k \in \mathbb{R};\ \beta \in \mathbb{R};\ i = 1, ..., k:\ \ y_i \in \mathbb{R}$

$\mathbf{H}$[5] $\in \mathbb{R}^{(k+1) \times k}$ with elements h_{ij}.

Algorithm:

Let $\mathbf{x_0}\ \in \mathbb{R}^n$

$\mathbf{r_0} = \mathbf{b} - \mathbf{A x_0}$

If $\mathbf{r_0} = \mathbf{0}$, then END

$\mathbf{v_1} = \frac{\mathbf{r_0}}{\|\mathbf{r_0}\|_2}$

$\gamma_1 = \|\mathbf{r_0}\|_2$

for $k = 1, ..., n$

for $i = 1, ..., k$ do $h_{ik} = \mathbf{v_i}^T \mathbf{A v_k}$

$\mathbf{w_k} = \mathbf{A v_k} - \sum_{i=1}^{k} h_{ik} \mathbf{v_i}, \quad h_{k+1,k} = \|\mathbf{w_k}\|_2$

for $i = 1, ..., k-1$

do $\begin{pmatrix} h_{ik} \\ h_{i+1,k} \end{pmatrix} = \begin{pmatrix} c_{i+1} & s_{i+1} \\ s_{i+1} & -c_{i+1} \end{pmatrix} \begin{pmatrix} h_{ik} \\ h_{i+1,k} \end{pmatrix}$

$\beta = \sqrt{h_{kk}^2 + h_{k+1,k}^2}; \quad s_{k+1} = \frac{h_{k+1,k}}{\beta}$

$c_{k+1} = \frac{h_{kk}}{\beta}; \quad h_{kk} = \beta$

$\gamma_{k+1} = s_{k+1} \gamma_k; \quad \gamma_k = c_{k+1} \gamma_k$

if $\gamma_{k+1} \neq 0, \mathbf{v_{k+1}} = \frac{\mathbf{w_k}}{h_{k+1,k}}$

[5] This is called the **Hessenberg** matrix

else

for $i = k, ..., 1$

do $y_i = \frac{1}{h_{ii}} \left(\gamma_i - \sum_{j=i+1}^{k} h_{ik} y_j \right)$

$\mathbf{x} = \mathbf{x_0} + \sum_{i=1}^{k} y_i \mathbf{v_i}$ END

The computational complexity and allocated memory increase linearly with every iteration step (for **CG** and **BICGstab** the calculations are constant in every step). The complexity of **GMRES** at the m^{th} iteration step are:

- 1 matrix-vector multiplication
- m scalar products
- $\approx 3 \cdot m \cdot n$ floating point operations

Therefore it is sometimes recommended to restart the algorithm after a certain number of steps with the actual solution. The restart deletes all the previous base vectors. Hence the base can only have a restricted number of vectors which may be insufficient to calculate the solution. Like the **CG** algorithm, **GMRES** requires only the initial vector as input besides the matrix.

The Table 2.1 shows a comparison of the computational complexity of the different solvers.

solver	matrix-vector	scalar products	floating point
CG	1	4	6n
BICGstab	2	6	12n
GMRES	1	m	≈3·m·n

Table 2.1: Effort for a matrix with rank n: **CG** is the simplest solving algorithm and thus has the least operations. The computational effort for **BICGstab** algorithm is nearly twice the effort for **CG**. The complexity for **GMRES** depends on the iteration step m. The effort for the first few steps is lower than for the other solvers. But after several steps the effort is much larger than for the other solvers.

2.3.4 Other Krylov Methods

There are several other methods available which are also based on the **Krylov** subspace. Some are for computing the solution of a linear equation system; others are for the calculation of the eigenvalues:

- **BICG:** further enhancement of the **CG** algorithm for non-symmetric matrices, very unstable though [6].
- **CGS:** conjugate gradient squared, is a squared BICG [1].
- **QMR:** quasi minimum residual method, uses two **Krylov** subspaces built by the system matrix and its transpose [6].
- **TFQMR:** transpose free quasi minimum residual method, works without the transpose of the system matrix [1].

2.4 Preconditioners

Solving a linear system

$$\mathbf{Ax} = \mathbf{b}$$

by an iterative solver requires a well-conditioned matrix **A**. Otherwise the solver requires too many iterations or may fail completely. If **A** is ill-conditioned then one can multiply **A** by another matrix **P** such that $\mathbf{P}^{-1}\mathbf{A}$ has a smaller condition number than **A** [5].

This approach yields the following equation system:

$$\mathbf{P}^{-1}\mathbf{Ax} = \mathbf{P}^{-1}\mathbf{b}$$

The preconditioner **P** which is used should approximate the inverse of matrix **A** as well as possible. This is because if

$$\mathbf{P}^{-1} = \mathbf{A}^{-1}$$

then the linear system becomes

$$\mathbf{Ix} = \mathbf{A}^{-1}\mathbf{b}$$

The system would be solved by a simple matrix multiplication. In practice the preconditioner is not multiplied by the matrix directly; in fact, the residual

$$\mathbf{r_k} = \mathbf{b} - \mathbf{Ax_k}$$

is multiplied by the preconditioner. This approach reduces computational complexity because a matrix-matrix multiplication is more complex than two matrix-vector multiplications.

2.4.1 Jacobi Preconditioner

The simplest way to compute a preconditioner of a matrix **A** is the **Jacobi** preconditioner. The preconditioner **P** is the diagonal part of **A**:

$$\mathbf{P} = \mathbf{D}$$
$$p_{ij} = a_{ij}\delta_{ij} = \begin{cases} a_{ij} & i = j \\ 0 & \text{otherwise} \end{cases}$$
$$p_{ij}^{-1} = \frac{\delta_{ij}}{a_{ii}}.$$

2.4.2 ILU Preconditioner

The **Jacobi** preconditioner is very simple and may not improve the condition of the matrix enough to solve all kinds of linear equation systems. In this case a more complex preconditioner, like the incomplete LU factorization (**ILU**), is needed. The matrix **A** is decomposed into a lower[6] and an upper[7] triangular matrix $\mathbf{A} = \mathbf{LU}$. This approach eases the computation of the inverse of **A** because it is easier to invert a triangular matrix than a full matrix. (This is similar to the backward solving of the solution vector during a Gaussian elimination). If the decomposition of the matrix **A** into **LU** is only done for a certain number of matrix elements so that $\mathbf{A} \approx \mathbf{LU}$, it is called the incomplete LU decomposition [1].

[6] a matrix containing only zeros above the main diagonal
[7] a matrix containing only zeros under the main diagonal

A simple algorithm for **ILU**:

The **L** and **U** matrices can be stored in one matrix **M**. The upper part of **M** including the diagonal part is **U**. The lower part of **M** without the diagonal part is **L**. The diagonal part of **L** contains only ones. In the following algorithm only the nonzero elements of the matrix will be changed.

for $k = 1, ..., n-1$ do
    for $i = k+1, ..., n$ do
        $m_{ik} := m_{ik}/m_{kk}$
            for $j = k+1, ..., n$ do
                $m_{ij} := m_{ij} - m_{ik}m_{kj}$

The algorithm contains a division by the diagonal elements (pivot elements for **ILU**). Thus zeros in the main diagonal let the algorithm fail. A variation of the **ILU** preconditioner is the **ILUT** preconditioner which only keeps entries that are greater than a defined threshold (also called drop tolerance). This is used to save computation time by ignoring entries close to zero. Another important number is the ratio of fill, which defines how many nonzeros are allowed in the preconditioner matrix compared to the original matrix. This can be implemented by:

- Restricting the maximum number of row entries in the matrix
- Restricting the additional entries in a row compared to the original matrix
- Restricting the maximum number of entries in the preconditioner

2.4.3 Incomplete Cholesky Factorization

For a symmetric and positive definite matrix, the incomplete Cholesky (**IC**) factorization can be used as a preconditioner. It is similar to the **ILU** factorization. The Cholesky factorization is a direct solver for symmetric matrices. The matrix is decomposed into the product of a lower matrix and its transpose: $\mathbf{A} = \mathbf{L}\mathbf{L}^T$. The incomplete Cholesky factorization is only done for a limited number of matrix elements so the product is not exactly the matrix **A** [6].

$$\mathbf{A} \approx \mathbf{L}\mathbf{L}^T \tag{2.14}$$

Algorithm for decomposition into $\mathbf{L}\mathbf{D}\mathbf{L}^T$ (**D** diagonal matrix):

for $i = 1, ..., n$
    $d_i = a_{ii} - \sum_{k=1}^{i-1} d_k l_{ik}^2$
    for $j = i+1, ..., n$
        $d_i l_{ji} = a_{ji} - \sum_{k=1}^{i-1} d_k l_{jk} l_{ik}$

The **IC** preconditioner also can use drop tolerances and ratio of fill similar to **ILU**.

2.4.4 Other Preconditioners

There are several other preconditioners available:

- **Multilayer**/**Multigrid** preconditioners are only usable if the equation system offers an underlying mesh, e.g., a semiconductor simulation [7]. The preconditioner uses meshes with different numbers of elements. Starting with the smallest mesh, a preconditioner can be set up by using each mesh as a different preconditioner level. These levels will be added (Multilayer) or multiplied (Multigrid) to achieve the final preconditioner.
- Other iterative methods can be used as preconditioners, like the **Gauss-Seidel** algorithm or the **SOR** (successive over relaxation) [6].
- **Additive Schwarz** preconditioner splits the problem domain into smaller domains and adds the results of these [8].

2.5 Residual

The convergence of the solver is indicated by the residual norm. If the residual norm is small enough, the solving process is stopped. The two most common methods of calculating the break condition are based on the absolute residual and on the relative residual. If the absolute residual is used, the solving process stops when the norm of the residual gets smaller than the desired accuracy. In case of the 2-norm this would be

$$\|\mathbf{r_k}\|_2 < \epsilon \tag{2.15}$$

On the other hand the relative residual can be checked to determine the convergence. The relative residual is the absolute residual divided by the norm of the right-hand side vector **b**.

$$\frac{\|\mathbf{r_k}\|_2}{\|\mathbf{b}\|_2} < \epsilon \tag{2.16}$$

The choice of the convergence method can have an influence on the quality of the solution. If the norm of the b-vector is much larger than one, the absolute residual should be used as a break condition, because if the relative residual is used, many solutions are possible which won't be very accurate, due to the fact that the residual is divided by a large number which brings it closer to the break accuracy. On the other hand, if the norm of the **b**-vector is close to the break accuracy, it would be better to use the relative residual because using the true residual could cause the initial guess (zero in most cases) to be the solution, which may not be accurate enough.

Considering this, a good limit value for the decision of the residual would be 1. If the norm of **b** is greater than or equal to 1.0, the absolute residual can be used. The relative residual can be used if the norm is smaller than 1.0. If it is done the other way around, the iterations taken would be decreased but the solution might not be very accurate.

2.6 Loss of precision

A loss of precision is encountered when the solution vector returned is not usable as a solution for the linear system because it does not have the desired accuracy (regardless which of the residuals discussed in Section 2.5 is used). The reason for this problem is that the solvers all calculate a recursive residual which is updated every iteration step. This residual looks different for each of the three solvers introduced:

$$\textbf{CG} : \mathbf{r_{k+1}} = \mathbf{r_k} - a_k \mathbf{A p_k} \tag{2.17}$$

$$\textbf{BICGstab} : \mathbf{r_{k+1}} = \mathbf{s} - w_{k+1}\mathbf{t} \tag{2.18}$$

$$\textbf{GMRES} : \gamma_{k+1} = s_{k+1}\gamma_k, \gamma_{k+1} \hat{=} \|\mathbf{r_{k+1}}\| \tag{2.19}$$

The residuals of the different solver algorithms have in common that they are all built recursively from elements built in the current iteration step or in the one before. Therefore round-off errors which occur every iteration step can become very meaningful and may lead to a totally wrong recursive residual. If the solver only checks the recursive residual, it can accidently state that the problem is solved even if it is not. To avoid this problem, the true residual should be checked after the solver is finished:

$$\mathbf{r_{k+1}} = \mathbf{b} - \mathbf{A} \cdot \mathbf{x_{k+1}} \tag{2.20}$$

The problem is not really predictable, but it is possible to deal with it. The reason it occurs is due to round-off errors during the solving of an ill-conditioned matrix, so it would help to reformulate the linear equation system. To reformulate the equation system the underlying problem must be known and modified to acquire a matrix which is more solvable[1]. Only the user knows the problem behind the matrix and not the automatic solver control, so this is not an option here. Another way to deal with the problem is introduced here. First the residual of the solution obtained is checked. If it is smaller than one, and close to the desired accuracy, the solution vector can be used as an initial guess for a new solving process. In this case, the same solver can be used again with the old solution as an initial guess. Additionally the accuracy for convergence is decreased. It is possible that the loss of precision may occur again, so the resolving process should be carried out a few times. Each time the accuracy for convergence is decreased further.

[1] E.g., if a simulation of a drift diffusion is calculated which has an underlying mesh, the modification of the mesh would lead to a new equation system. The modification should be done such that the new matrix has a better condition than the former.

Chapter 3

Related Solver Packages

The following sections provide an overview of the **Trilinos** package, the **QQQ**[1] package, the **PETSc** package, the **Hypre** package and the C++ matrix template library (**MTL/ITL**)[2].

3.1 Trilinos Solver Package

> *The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems. A unique design feature of Trilinos is its focus on packages.* [10]

The **Trilinos** solver package consists of many different packages like matrix manipulation tools (Epetra, EpetraExt, Tpetra, Jpetra, Kokkos), preconditioners (AztecOO, IFPACK, ML, Meros), linear solvers (Epetra, Teuchos, Pliris, AztecOO, Belos, Komplex, Amesos), eigenvalue solvers (Anasazi) and also nonlinear solvers (NOX, LOCA, MOOCHA, Rythmos) [11]. The **Krylov** subspace techniques are used by the package AztecOO which uses the Epetra package for matrix manipulation.

AztecOO supports the solvers **CG**, CGS (conjugate gradient squared), **BICGstab**, **GMRES** and TFQMR (Transpose Free Quasi-Minimal Residual). There is also a condition number estimator for the **CG** and **GMRES** solvers which estimates the condition number of the preconditioned equation system. This is connected with a higher cost of performance.

The preconditioners available are **Jacobi**, Neumann, least-square, Gauss-Seidel, **IC** and a few different **ILU** preconditioners. There is also the possibility of using a multilevel preconditioner by using the package ML.

The **Trilinos** package stores the matrix in CSR[3] format. For the matrix-vector multiplication and the scalar product, BLAS[4] functions are used.

[1] The QQQ solver package was developed at the Institute of Microelectronics, Vienna University of Technology

[2] some of the packages use **MPI** (message passing interface, a standard that allows communication among many different computers). For the sake of simplicity the **MPI** support has been deactivated [9].

[3] see Section B.1.1

[4] The BLAS (Basic Linear Algebra Subprograms) programming interface offers various functions for the computation of matrix products, matrix-vector products or scalar products [12].

3.1.1 CG

The **x**-vector used for the **CG** algorithm can be used as an initial guess. If this is not desired, the **x**-vector should be set to zero. The **CG** algorithm of **Trilinos** has several integrated break conditions which are not mentioned in Section 2.3.1. The solving process stops if the dot product of **p** and **Ap** is less than zero, indicating that the matrix is not positive definite:

```
1  if (p_ap_dot < 0 || AZ_breakdown_f(N, p, ap, p_ap_dot, proc_config)){
2    //possible breakdown
3  }
```

Additionally, it stops if these two vectors are under a certain absolute threshold because then their dot product would be close to zero showing, that they are orthogonal. Another possible break point is if the new calculated dot product of the residual is under a defined threshold:

```
1  if (fabs(r_z_dot) < brkdown_tol){
2     //possible breakdown
3  }
```

3.1.2 BICGstab

The **x**-vector can be used as an initial guess; if this is not intended, the **x**-vector should be set to zero. The vector $\hat{\mathbf{r_0}}$ can be set to $\mathbf{r_0}$; otherwise there is an option to generate a random vector. The **Trilinos** package has more checks implemented than the algorithm introduced in Section 2.3.2. Every time there is a division, the divisor is checked to see if it is under a certain threshold. This is done in two different ways. The values *rhon* ($= \hat{\mathbf{r_0}}^T \mathbf{r_k}$, see Section 2.3.2) and *sigma* ($= \hat{\mathbf{r_0}}^T \mathbf{Ap_k}$, see Section 2.3.2)are checked if they are smaller than the value *brkdown_tol* (which is initialized with DBL_EPSILON of C++):

```
1  if (fabs(rhon) < brkdown_tol) {
2    if (...)
3      brkdown_will_occur = AZ_TRUE;
4    else brkdown_tol = 0.1 * fabs(rhon);
5  }
```

Line 2 in the code snippet above holds a check if the two vectors leading to the value *rhon* are orthogonal. The check is valid if the following statement is true:

$$|\mathbf{v}^T\mathbf{w}| < 100 \cdot \|\mathbf{v}\|_2 \|\mathbf{w}\|_2 \cdot DBL\_EPSILON$$

If they are not orthogonal the *brkdown_tol* is decreased else a flag is set that a breakdown will occur. Analogous the check of sigma is implemented, only difference is that the iteration is aborted immediately instead of setting a flag:

```
1  if (fabs(sigma) < brkdown_tol) {
2    if (...)) {
3     ...
4          return;
5        }  else brkdown_tol = 0.1 * fabs(sigma);
6  }
```

For the value *dot_vec[1]* which contains the scalar product of $\mathbf{t}^T\mathbf{t}$ (see Section 2.3.2) a different check is done. The value is analyzed if it is smaller than the DBL_MIN of C++.

```
1  if (fabs(dot_vec[1]) < DBL_MIN) {
2        omega = 0.0;
3        brkdown_will_occur = AZ_TRUE;
4     } else omega = dot_vec[0] / dot_vec[1];
```

3.1.3 GMRES

The **GMRES** algorithm of **Trilinos** checks if the matrix **H** (see Section 2.3.3) is ill-conditioned. If this is the case the solving process is aborted. The reasons that the matrix is ill-conditioned can be caused by bad zero pivot handling of the preconditioner. Also this can appear if the matrix is singular[5].

3.1.4 Jacobi

The only option that can be set for the **Jacobi** preconditioner is the number of **Jacobi** steps (Default: 1).

3.1.5 ILU

The drop tolerance of the **ILU** factorization can be set (Default: 0.0) and also the ratio of fill compared to the original matrix can be set (Default: 3). If a zero pivot is encountered, the preconditioner uses the value *rownorm*, which is calculated from the values of the row, divided by the number of values and multiplied by the drop tolerance (if it is not equal to zero):

3.1.6 IC

The **IC** preconditioner of **Trilinos** offers the same options as the **ILU** preconditioner and thus has the same default values.

3.1.7 Loss of Precision

Trilinos is the only package that warns if the solution is not accurate and thus a loss of precision has occurred.

3.2 PETSc Solver Package

> *PETSc includes a large suite of parallel linear and nonlinear equation solvers that are easily used in application codes written in C, C++, Fortran and now Python.* [13]

PETSc (Portable, Extensible Toolkit for Scientific Computation) [14]: supports the **Krylov** subspace methods Richardson, **CG**, BICG, **GMRES**, **BICGstab**, CGS, two versions of TFQMR, and several others. The preconditioners provided are **Jacobi**, Block Jacobi, SOR, **incomplete Cholesky**, **ILU**, and additive Schwarz. It has the ability to use linear solvers as preconditioners and the option to combine different preconditioners. For this work **CG**, **BICGstab** and **GMRES** were used with **Jacobi**, **ILU** and **IC** preconditioners.

PETSc stores the matrix in a CSR[6] format and uses BLAS functions to calculate the scalar product. For the matrix-vector multiplication an internal function is used.

[5] A matrix is singular if its inverse does not exist

[6] see Section B.1.1

3.2.1 CG

The **CG** algorithm can be filled with an initial guess. To prevent a user from accidentally providing an initial guess, the usage of an initial guess must be activated explicitly. The **CG** algorithm breaks if the beta value is less than zero indicating that the preconditioned matrix is not positive definite:

```
else if (beta < 0.0) {
   ksp->reason = KSP_DIVERGED_INDEFINITE_PC;
   ierr = PetscInfo(ksp,"diverging due to indefinite preconditioner\n");CHKERRQ(ierr);
}
```

Another breakdown can be reached if the value *dpi* (equals the multiplication $\mathbf{p_k^T A p_k}$) is less than zero indicating that the matrix is indefinite or negative definite:

```
if (PetscRealPart(dpi) <= 0.0) {
   ksp->reason = KSP_DIVERGED_INDEFINITE_MAT;
   ierr = PetscInfo(ksp,"diverging due to indefinite or negative definite matrix\n");
   CHKERRQ(ierr);
   break;
}
```

PETSc also checks to see if the scalar product yields a valid number:

```
if PetscIsInfOrNanScalar(beta)
   SETERRQ(PETSC_ERR_FP,"Infinite or not-a-number generated in dot product");
```

3.2.2 BICGstab

For the **BICGstab** an initial guess can be set if the initial guess option of **PETSc** is activated. The vector $\hat{\mathbf{r_0}}$ is set to $\mathbf{r_0}$. **PETSc** avoids divisions by zero by checking divisors to see if they are zero:

```
if (d1 == 0.0) SETERRQ(PETSC_ERR_PLIB,"Divide by zero");
```

The algorithm breaks if the value *rho* (scalar product of the vector $\hat{\mathbf{r_0}}$ and the residual vector of the current iteration step) equals zero.

```
    if (rho == 0.0) {
      ksp->reason = KSP_DIVERGED_BREAKDOWN;
      break;
    }
```

3.2.3 GMRES

The **GMRES** code does not have significant changes to the algorithm in Section 2.3.3.

3.2.4 ILU

The **PETSc** package offers many functions to adjust the **ILU** factorization. The zero pivot limit can be defined (Default: $1 \cdot 10^{-12}$) and also how the preconditioner should act if it is encountered (e.g., shifting of nonzero, reordering of matrix etc.). The fill amount can be set via a factor that indicates how big the preconditioner is in comparison to the original matrix (Default: 1). Also a drop tolerance for matrix elements can be set to keep only values over the defined tolerance.

3.2.5 IC

The **IC** preconditioner uses the same default options as the **ILU** preconditioner.

3.2.6 Divergence tolerance

PETSc is the only package offering a divergence tolerance value. If the residual becomes larger than this value, the solving process is aborted (Default: $1 \cdot 10^5$).

3.3 Iterative Template Library/Matrix Template Library

The matrix template library 4 (MTL4) is currently under development by Peter Gottschling and Andrew Lumsdaine [15]. The latest version is only available from a public subversion repository and may be in an instable state (https://svn.osl.iu.edu/tlc/trunk/mtl4/trunk mtl4). Nevertheless the library is worth testing and provides some iterative solving methods. These are in the iterative template library (**ITL**) which is part of MTL4. At the moment there are **CG**, BICG and **BICGstab** solvers available and **Jacobi**, **ILU** and **IC** preconditioners.

ITL stores the matrix in a CSR[7] format and has internal functions for calculating the scalar product and matrix-vector multiplication. The following sections discuss the difference in code adaptation to the algorithms in Sections 2.3.1 and 2.3.2.

3.3.1 CG

The **x**-vector is used as an initial guess on input and as a solution vector on output. There are no significant changes to the algorithm in Section 2.3.1. The only check that is done is after each iteration loop which is used to break if the residual is lower than the break accuracy.

3.3.2 BICGstab

The **x**-vector is used as an initial guess on input and as a solution vector on output. For the vector $\hat{\mathbf{r_0}}$ the vector $\mathbf{r_0}$ is used. Two additional checks are implemented in the code which are not in the algorithm shown in Section 2.3.2. First the scalar product of $\hat{\mathbf{r_0}}^T \mathbf{r_0}$ is checked if it is equal to zero.

```
1 rho_1 = dot(rtilde, r);
2    if (rho_1 == Scalar(0.)) {
3      iter.fail(2, "bicg breakdown #1");
4     break;
5   }
```

Second the value *omega* $((\mathbf{s}^T\mathbf{t})/(\mathbf{t}^T\mathbf{t}))$ is checked if is equal to zero:

```
1  if (omega == Scalar(0.)) {
2    iter.fail(3, "bicg breakdown #2");
3    break;
4 }
```

[7] see Section B.1.1

3.3.3 Jacobi

In the code for the **Jacobi** preconditioner there is no check to see if the value from which the reciprocal is built is zero:

```
for (size_type i= 0; i < num_rows(A); ++i)
        inv_diag[i]= reciprocal(A[i][i]);
    }
```

3.3.4 ILU

The **ILU** code checks whether all main diagonal elements exist; it does not check whether they are close or equal to zero. There are no options which can be used for this preconditioner.

3.3.5 IC

There are no additional options which can be used for this preconditioner. The implementation of the code checks whether all diagonal elements are present, but it does not check whether they are zero or smaller than a certain limit.

3.4 QQQ Solver Package

The **QQQ** Solver was developed at the University of Technology in Vienna by Claus Fischer and Stephan Wagner [16]. It supports a direct solver and the indirect solvers **BICGstab** and **GMRES**. The only preconditioner available is an **ILU** factorization. It also has the possibility to pre-eliminate some critical rows and solve them with a direct solver. There is also a mechanism for scaling of the matrix elements.

One important difference from the other packages is that the solver always uses the zero vector as the initial guess. The user can not specify another initial guess. It is also the only package that uses the MCSR[8] matrix format. Also it is the only package that can not switch between relative and absolute residual, because it only provides the relative residual.

The package uses internal functions for the calculation of scalar products and matrix-vector products.

3.4.1 BICGstab

The initial guess for the algorithm is set to zero. The vector $\hat{\mathbf{r}_0}$ is set to $\mathbf{r_0}$. During the solving process the function qqqFromZero(**double**, **double**) is used several times to prevent division by zero. The function returns the maximum of the two given double values. The second double value is a reference value which will not be undercut. It is initialized with:

```
    Real const minsize
    = Real(1.0e6) * DBL_MIN;
```

One example of the usage is:

```
Number const dotrtv = qqqDotPrd(n,r0,v));
alpha = rho / qqqFromZero(dotrtv,minsize);
```

[8] see Section B.1.2

To avoid division by zero, the solving process is kept up and the solver may be able to find a solution where the other packages already gave up.

3.4.2 GMRES

The **GMRES** solver does not have significant changes to the algorithm introduced in Section 2.3.3.

3.4.3 ILU

Two values can be set for the **ILU** factorization. First the drop tolerance of the factorization can be defined to specify the limit at which a value is dropped (default: 1e-9). The second value is *fillMax* which stands for the allowed number of additional elements in each row (default: -1, which indicates that no additional element is allowed).

If a zero pivot is encountered, the preconditioning is aborted and the solving is stopped. However the **ILU** algorithm uses an internal sort function so that a zero pivot is rarely encountered, even if there are several zeros on the main diagonal of the initial matrix.

3.4.4 Residual for Convergence

QQQ only checks the convergence using the relative residual. There is no way to switch to the absolute residual, so if this is desired, the residual has to be modified to make the package useful for that case. The solving accuracy can be divided by the norm of the right-hand side vector. The package still will use the relative residual as a break condition but the new accuracy will simulate the absolute residual of the former accuracy:

$$\|\mathbf{r_k}\|_2 \leq \epsilon \tag{3.1}$$

$$\Rightarrow$$

$$\frac{\|\mathbf{r_k}\|_2}{\|\mathbf{b}\|_2} \leq \epsilon / \|\mathbf{b}\|_2 \tag{3.2}$$

$$\Rightarrow$$

$$\|\mathbf{r_k}\|_{rel} \leq \frac{\epsilon}{\|\mathbf{b}\|_2} \tag{3.3}$$

3.4.5 Pre-elimination, Sorting and Scaling

The **QQQ** package offers additional algorithms for improving the condition number of a given matrix. If the flags for pre-elimination, sorting and scaling are set, automatic versions of these algorithms are used. During the tests of the matrices it also was tested if they have much influence on the performance. The default values of these implementations did not give a significant increase in performance and the same results at nearly the same speed have been reached, so these features have not been investigated further.

3.5 Hypre Solver Package

> *Hypre is a library for solving large, sparse linear systems of equations on massively parallel computers.* [17]

Hypre stores the matrix in CSR[9] format and uses internal functions to calculate scalar product and matrix-vector multiplication. **Hypre** supports the indirect solvers **CG**, **BICGstab** and **GMRES**. The **ILU** preconditioner is the only introduced preconditioner that is provided by the package.

3.5.1 CG

The **CG** algorithm of **Hypre** can be started with an initial guess, which is stored in the **x**-vector given to the algorithm. If this is not intended, the **x**-vector should be set to zero. The **CG** algorithm breaks if the gamma (gamma stands for the scalar product of the residual and the preconditioned residual) value is under a certain threshold (also described in the original comment):

```
1  if ( (gamma<1.0e-292) && ((-gamma)<1.0e-292) ) {
2          hypre_error(HYPRE_ERROR_CONV);
3          break;
4       }
5       /* ... gamma should be >=0.  IEEE subnormal numbers are < 2**(-1022)=2.2e-308
6          (and >= 2**(-1074)=4.9e-324).  So a gamma this small means we're getting
7          dangerously close to subnormal or zero numbers (usually if gamma is small,
8          so will be other variables).  Thus further calculations risk a crash.
9          Such small gamma generally means no hope of progress anyway. */
```

3.5.2 BICGstab

The **BICGstab** algorithm can be filled with an initial guess, which is stored in the **x**-vector. If this is not intended, the **x**-vector should be set to zero. The vector $\hat{\mathbf{r_0}}$ is set to $\mathbf{r_0}$. The **BICGstab** algorithm of **Hypre** breaks if the value *temp* (scalar product of the residual and the preconditioned residual) is under a certain threshold avoiding a division by a number close to zero.

```
1  if (fabs(temp) >= epsmac) //epsmac = 1.e-128
2     alpha = res/temp;
3  else {
4     printf("BiCGSTAB broke down!! divide by near zero\n");
5     return(1);
6  }
```

Another breakdown occurs if the variable *res* (the scalar product of the vector $\hat{\mathbf{r_0}}$ and the residual vector of the current iteration step) falls under a defined threshold, avoiding a division by a number close to zero.

```
1  if (fabs(res) >= epsmac)
2     beta = 1.0/res;
3  else {
4     printf("BiCGSTAB broke down!! res=0 \n");
5     return(2);
6  }
```

A breakdown which often occurs on small matrices, is if the *gamma* (equals w_{k+1} in Section 2.3.2) value is smaller than a defined limit.

[9] see Section B.1.1

```
if (fabs(gamma) >= epsmac)
    (*(bicgstab_functions->ScaleVector))((beta*alpha/gamma),p);
else {
    printf("BiCGSTAB broke down!! gamma=0 \n");
    return(3);
}
```

3.5.3 GMRES

The **GMRES** algorithm has no significant changes from the algorithm introduced in Section 2.3.3.

3.5.4 ILU

For the **ILU** preconditioner the drop tolerance can be set to raise the speed of the preconditioning process (default: 0.0001). The second parameter that can be set is the maximum number of elements each row of the preconditioner matrix can have (default: 20).

Table 3.1 gives an overview of the introduced packages.

	Trilinos	**PETSc**	**ITL**	**QQQ**	**Hypre**
CG	YES	YES	YES	NO	YES
BICGstab	YES	YES	YES	YES	YES
GMRES	YES	YES	NO	YES	YES
Jacobi	YES	YES	YES	NO	NO
ILU	YES	YES	YES	YES	YES
IC	YES	YES	YES	NO	NO
Matrix-format	CSR	CSR	CSR	MCSR	CSR
residual norm	abs/rel	abs/rel	abs/rel	rel	abs/rel
initial guess	YES	YES	YES	NO	YES
loss of precision handling	YES	NO	NO	NO	NO
scalar product	BLAS	BLAS	internal	internal	internal
matrix-vector product	BLAS	internal	internal	internal	internal

Table 3.1: Packages Overview

Chapter 4

Diagnostic Tool

For the development of an automatic solver control which can decide which solvers and preconditioners are used, there need to be some matrix properties on which the decisions are based. Some kind of diagnostic tool is necessary which can find out the important facts of a matrix. This tool needs to be fast enough so that its use is relevant.

4.1 Matrix and Vector Container

4.1.1 Sparse Matrix Storage

The first thing that is needed for a matrix is a buffer where it is stored. The intended use was as intermediate storage between the reading of the matrix and the assembling of the specific internally stored matrices of the solvers (later it became a part of the diagnostic tool, too). This storage type needs the following properties:

1. only store the non-zeros
2. no order required for inserting of elements
3. possibility to read out only the nonzero elements
4. stores the dimension of the full matrix

For this purpose the boost multi_index[1] containers library was used. The advantage of the multi-index container is that it is possible to order the objects inserted by more than one order (e.g., a matrix can be ordered by row **and** by column index). This makes it possible to traverse the objects of the container differently. For the purpose of a sparse matrix the objects stored are the nonzero elements which have a row index, a column index and a value (e.g., **double**). The following **struct** represents that:

```
1 struct matrix_element {
2   int          row;
3   int          column;
4   double       value;
5   ...
6 };
```

[1] Boost Multi Index (http://www.boost.org/doc/libs/1_39_0/libs/multi_index/doc/index.html)

A multi-index container which can be ordered by row and column index can be defined by:

```
typedef multi_index_container< matrix_element,
    indexed_by<ordered_non_unique<tag<row>,
    BOOST_MULTI_INDEX_MEMBER(matrix_element,int,row)>,
    ordered_non_unique<tag<column>,
    BOOST_MULTI_INDEX_MEMBER(matrix_element,int,column)> >
> matrix_set_real;
```

The indices are ordered non-unique which indicates that an index can be inserted twice else for each row and column only one element would be allowed.

To add some error handling to the matrix container another **struct** is defined which also holds the dimension of the matrix, if it is initialized and a function to insert values into the matrix:

```
struct sparse_matrix {
  int dim;
  matrix_set_real data;
  bool initialized;

  void initialize(int dim_){...}

  sparse_matrix(int dim_){
    initialize(dim_);
  }

  void insert_value(int i, int j, double variable){
    if(initialized) {
      if((i<dim)&&(j>=0)&&(i<dim)&&(j>=0))
        data.insert(matrix_element<double>(i,j,variable));
      else throw MATRIX_INSERT_WRONG_INDEX;
    }else throw MATRIX_NOT_INITIALIZED;
  }

  int get_elements() {
    return data.size();
  }

  void print_matrix()    {...}
  int get_dimension(){
    return dim;
  }
};
```

The container does not explicitly circumvent double insertion of a matrix element, however this may lead to wrong solutions depending on the usage of the matrix and should be avoided.

There are two ways to read out the elements of the matrix. They can be accessed row-wise or column-wise which is indicated by the following code snippet:

```
typedef matrix_set_real::index<row>::type matrix_set_real_by_row;
typedef matrix_set_real::index<column>::type matrix_set_real_by_column;
```

To access them an iterator can be used, e.g., row-wise reading of elements and printing to terminal:

```
sparse_matrix matrix(1000);
...//insert elements
matrix_set_real_by_row::iterator iter_row;
for(iter_row = matrix.data.get<0>().begin();
    iter_row != matrix.data.get<0>().end(); ++iter_row) {
  std::cout << (*iter_row).row << std::endl;
  std::cout << (*iter_row).column << std::endl;
  std::cout << (*iter_row).value << std::endl;
}
```

Such a matrix format can now be used as intermediate storage between matrix reading and choosing the solver package where the equation system will be solved. The advantage of this format is that one can insert the elements of a matrix randomly. For other sparse matrix formats one needs to insert elements row wise or column wise. Also it is easy to traverse the nonzero elements of this matrix row- or column-wise. The disadvantages are that other formats may need less memory and that one needs to avoid multiple insertions of elements with the same indices.

4.1.2 Vector Storage

To store the right-hand side vector and the solution vector a simple class has been implemented that stores the dimension of the vector, if it is initialized and the vector elements. These are stored in a vector of the standard template library. The solution vector of a linear equation system is a dense vector so the format is dense.

```
class dense_vector {
  private:
    int dim;
    bool initialized;
    std::vector<double> data;
    ...
  public:

  dense_vector(int dimension){...}

  void insert_value(int row, double value){...}

  double get_value(int row){..}
  ...
}
```

The vector can be filled with the function insert_value (...) and an element can be read out by get_value (...) .

4.2 Matrix Entries

One interest lies in the relation of the numbers of diagonal and non-diagonal entries. E.g., a matrix with 50% diagonal elements may be perfect for the **Jacobi** preconditioner. For that reason the matrix storage type introduced (4.1.1) can be used to traverse through the elements and find out the numbers of diagonal and non-diagonal entries:

```
1  sparse_matrix matrix;
2  ...//fill matrix
3  matrix_set_real_by_row::iterator iter;
4  for(iter=matrix.data.get<0>().begin();
5      iter!=matrix.data.get<0>().end();++iter) {
6    if((*iter).row!=(*iter).column)//check for diagonal entry
7       nondiagonalelements++;
8     else    {
9      diagonalelements++;
10   }
11 }
```

With the number of non-zeros and diagonal elements, the diagonal-dominance (the diagonal elements percentage of all non-zeros) can be calculated.

Other interesting values are the average of the diagonal elements and the average of the non diagonal elements. If the diagonal average is greater than one while the average of the non diagonal elements is small (< 1), the **Jacobi** preconditioner may be suitable, because it guarantees only ones in the main diagonal. If all of the non-diagonal elements are very small, this will result in eigenvalues close to one, according to the Gershgorin theorem in Section 2.1.5. The diagonal average and non diagonal average can be calculated within the same loop as the diagonal elements:

```
1  { //traverse all matrix elements
2     if((*iter).row!=(*iter).column)//check for diagonal entry
3        value+=std::fabs((*iter).value);
4     else {
5        diagonalvalue+=std::fabs((*iter).value);
6     }
7  }
8  //average diagonalvalue
9  diagonalvalue=diagonalvalue/boost::lexical_cast<double>(diagonalelements);
10 //average nondiagonalvalue
11 value=value/boost::lexical_cast<double>(nonzeros-diagonalelements);
```

Also the maximum, minimum, absolute maximum and absolute minimum can be calculated.

4.3 Symmetry Check

The **IC** preconditioner is only available for symmetric matrices (see Section 2.4.3). A check whether the matrix is symmetric can decide if the preconditioner may be useable or not. The **IC** is a necessary preconditioner for the use of the **CG** solver, which needs a symmetric matrix for convergence.

The symmetry check can be implemented with the introduced sparse matrix storage format. The matrix is traversed simultaneously by a row and a column iterator. For a symmetric matrix the iterators always point on two values that are symmetric to each other (if no zeros have been added to the matrix). These values can used to determine if the matrix is symmetric. If the two values are not symmetric to each other one iterator tries to find the symmetric value of the other iterator. The following code snippet shows the symmetry check without the search for a symmetric value:

```
matrix_set_real_by_row::iterator iter_row=matrix.data.get<0>().begin();
matrix_set_real_by_column::iterator iter_col=matrix.data.get<1>().begin();

for(;(iter_row!=matrix.data.get<0>().end()) &&
      (iter_col!=matrix.data.get<1>().end());
    ++iter_row,++iter_col) {
  if((*iter_row).row!=(*iter_row).column){//check if element is on main diagonal
    if((*iter_col).column==(*iter_row).row){
      if((*iter_col).row==(*iter).column){
    if(std::fabs((*iter_row).value-(*iter_col).value)>SYMMETRIC_TOLERANCE) {
          symmetric=false;
      break;
    }else continue;
      }  else{
        ...//find symmetric element
      }
    } else {
   ...//find symmetric element
    }
  }
}
```

4.4 Positive Definite Matrices

CG and **BICGstab** require positive definite matrices to ensure convergence. The Gershgorin theorem can be used to estimate the minimum and the maximum eigenvalue or at least closed sets in which they are found. If the set containing the smallest eigenvalue is in the positive area of real values, the matrix is positive definite. If it overlaps with the negative area one may decide that it is nearly positive definite if its greater part is in the positive area. If the matrix is non-symmetric the symmetric part needs to be built first (according to Section 2.1.4). This is done with the help of the row and column iterators of the matrix which build a new matrix (its name is analyzer in the following code snippets).

```
sparse_matrix analyzer(matrix.get_dimension());

matrix_set_real_by_row::iterator iter_row=matrix.data.get<0>().begin();
matrix_set_real_by_column::iterator iter_col=matrix.data.get<1>().begin();

while((iter_row!=matrix.data.get<0>().end())&&
  (iter_col!=matrix.data.get<1>().end())){
   ...//build symmetric part
}
```

The **while**-loop is done until both iterators reached the end. If the iterators point on two symmetric values following **if**-branch is valid:

```
if( ( (*iter_row).row==(*iter_col).column) && ((*iter_col).row==(*iter_row).column)){
  if((*iter_col).column<=(*iter_col).row)
    analyzer.insert_value( (*iter_col).row , (*iter_col).column,
                           ((*iter_row).value + (*iter_col).value)*0.5);
  if( ( (*iter_row).row!=(*iter_row).column)&& ((*iter_row).column>= (*iter_row).row))
    analyzer.insert_value( (*iter_row).row , (*iter_row).column,
                           ((*iter_row).value + (*iter_col).value)*0.5);
  iter_row++;
  iter_col++;
}
```

To avoid a double insertion of an element on the main diagonal the second insertion is only made if row and column index are different. Additionally the column iterator only inserts an element if he points on the lower part of the matrix. Analogous the row iterator only inserts an element if he points on the upper part of the matrix. If the iterators would not be restricted that way each element would be inserted twice.

The two other **if**-branches are valid if one of the iterators did not traverse as far as the other. Then this iterator is increased till it catches up with the other one (additionally all elements it passes need to inserted as well). The following code snippet will introduce this for the row iterator:

```
else if(((*iter_row).row<(*iter_col).column) ||
   (((*iter_row).row==(*iter_col).column)&&
    ((*iter_col).row > (*iter_row).column))) {
  if((*iter_row).column>= (*iter_row).row){
    analyzer.insert_value( (*iter_row).row , (*iter_row).column,
                           (*iter_row).value*0.5);
    analyzer.insert_value( (*iter_row).column , (*iter_row).row,
                           (*iter_row).value*0.5);
  }
   iter_row++;
}
```

Analogous the code part for the column iterator can be implemented.

After the symmetric part is build the the Gershgorin's discs which contain the eigenvalues can be calculated. Three vectors (of the introduced dense vector format) are used to store the central point, the row radius and the column radius of each disc:

```
dense_vector radii_row(matrix.get_dimension());
dense_vector radii_col(matrix.get_dimension());
dense_vector central_points(matrix.get_dimension());
```

These are filled by traversing all elements of the analyzer matrix first row-wise to get the central points and the row radii.

```
//traverse row-wise
if((*iter_row).row==(*iter_row).column)
  central_points.insert_value((*iter_row).row,(*iter_row).value);
else
  radii_row.insert_value((*iter_row).row,
     radii_row.get_value((*iter_row).row)+std::fabs((*iter_row).value));
```

Then column-wise to determine the column radii.

```
//traverse column-wise
if((*iter_col).row!=(*iter_col).column)
  radii_col.insert_value((*iter_col).column,
     radii_col.get_value((*iter_col).column)+std::fabs((*iter_col).value));
```

After the three vectors are filled the two closed sets which contain the smallest and largest eigenvalue can be built. The important set for the definite is that which contains the smallest eigenvalue:

```
for(int i=1; i< central_points.get_dimension(); ++i){
  double buffer1, buffer2, radius;

  if(radii_col.get_value(i)>radii_row.get_value(i))
    radius=radii_col.get_value(i);
  else
    radius=radii_row.get_value(i);

  buffer1=central_points.get_value(i)+range;
  buffer2=central_points.get_value(i)-range;

  //check if the old maxima and minima of the smallest eigenvalue set are
  //dominating the new values
  if(buffer1<mineigenvaluemax)
    mineigenvaluemax=buffer1;
  if(buffer2<mineigenvaluemin)
    mineigenvaluemin=buffer2;
}
```

After the set for the smallest eigenvalue has been calculated, the decision whether the matrix is positive definite can be made. Therefore the minimum of the smallest eigenvalue needs to be positive to ensure the matrix is positive definite. If it is negative, but the greater part of the set is within $\mathbb{R}^+$, then one can state that the matrix "nearly positive definite". This statement will also be used in the next chapter if the matrix is only qualified as may be positive definite.

4.5 Condition Number

The condition number would be very helpful in deciding which solver or preconditioner may be used. The problem is that it is difficult to estimate before the solving process. There are some solvers which can estimate the condition number during the solving process, but the problem is that does not really help in the selection of the solver (the condition number is ready after the solving process has been finished).

Chapter 5

Matrices for Solver Testing

Several matrices have been used for testing of the solver packages and some of their different solvers. The break condition of the residual has been set to $1e^{-12}$. The relative residual was used for the test run because **QQQ** offers no explicit check for the absolute residual. The maximum number of iterations was set to the rank of the tested matrix. For the non-symmetric matrices **Jacobi** and **ILU** preconditioners have been tested. For symmetric matrices **IC** has been used as well. The value for the **GMRES** restart has been set to 1000. The results of the tests are listed in appendix A.

This chapter tries to detect the strengths and expose the weaknesses of the different code implementations of the solving and preconditioning algorithms. Several matrix properties are investigated to see if they may give a hint which solving options fit them. For the preconditioners, the default options the packages offer have been used, to see how the different options influence the solving process. The solving and assembly times were measured with a timer provided by the Epetra package of **Trilinos**. For the time of analysis the average value of a series of one hundred measured time values for each matrix was determined. The assembly time is the average value of a series of twenty time values. The results of the solving speed measurements are only discussed if the matrix has a rank greater than 10.

The implemented solver interface for each package has been compiled with the options: -pipe -pedantic -Wall -O3 -s. The option -03 has been used to guarantee full optimization by the compiler so each code can perform at its best. The solvers tested in these sections are:

- **Trilinos** package; Version 9.0.1
  Solvers: **CG**, **BICGstab** and **GMRES**
  Preconditioners: none, **Jacobi**, **ILU** and **IC**
- **PETSc** package; Version 3.0.0-p1
  Solvers: **CG**, **BICGstab** and **GMRES**
  Preconditioners: none, **Jacobi**, **ILU** and **IC**
- **ITL** package; Version mtl4 alpha-1
  Solvers: **CG** and **BICGstab**
  Preconditioners: none, **Jacobi**, **ILU** and **IC**
- **Hypre** package; Version 2.4.0b
  Solvers: **CG**, **BICGstab** and **GMRES**
  Preconditioners: none and **ILU**

- **QQQ** package:

  Solvers: **BICGstab** and **GMRES**

  Preconditioners: none and **ILU**

5.1 Tridiagonal Matrices

A good example of a tridiagonal matrix is the one obtained by the finite difference one-dimensional Laplace problem [11]. This matrix will be investigated in different sizes, starting with rank 2 (although the rank 2 version is not a real tridiagonal matrix). The matrix is symmetric and positive definite. It was basically used to test the solvers on small matrices, to see how they perform.

5.1.1 Rank 2

The rank 2 version of the matrix has the form:

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

Matrix Properties:

Dimension	2
Symmetric	YES
Positive Definite	YES
Value Range	-1 to 2
Absolute Minimum	1
Absolute Maximum	2
Diagonal Entries	2
Diagonal Zeros	0
Total Non-Zeros	4
Diagonal Average	2
Non-diagonal Average	1
Diagonal Dominance	0.5

Important Times:

Trilinos Assembly Time	2.93e-02 ms
PETSc Assembly Time	1.41e-01 ms
ITL Assembly Time	2.50e-03 ms
Hypre Assembly Time	8.05e-03 ms
QQQ Assembly Time	1.28e-03 ms
Time for Analysis	1.71e-01 ms

Results

All solvers computed a solution except **Hypre BICGstab** (see Table A.1) which encountered the error *"BiCGSTAB broke down!! gamma=0"* described earlier in Section 3.5.2.

5.1.2 Rank 3

The rank 3 version of the matrix has the form:

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

Matrix Properties:

Dimension	3
Diagonal Entries	3
Symmetric	YES
Diagonal Zeros	0
Positive Definite	YES
Total Non-Zeros	7
Value Range	-1 to 2
Diagonal Average	2
Absolute Minimum	1
Non-diagonal Average	1
Absolute Maximum	2
Diagonal Dominance	0.428

Important Times:

Trilinos Assembly Time	3.05e-02 ms
PETSc Assembly Time	1.41e-01 ms
ITL Assembly Time	2.60e-03 ms
Hypre Assembly Time	9.00e-03 ms
QQQ Assembly Time	1.34e-02 ms
Time for Analysis	1.51e-01 ms

Results

All solvers computed a solution except **BICGstab** of **Hypre** (Table A.2) which encountered the error *"BiCGSTAB broke down!! gamma=0"* described earlier in Section 3.5.2.

5.1.3 Rank 4

The rank 4 matrix has the form:

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

Matrix Properties:

Dimension	4
Symmetric	YES
Positive Definite	YES
Value Range	-1 to 2
Absolute Minimum	1
Absolute Maximum	2
Diagonal Entries	4
Diagonal Zeros	0
Total Non-Zeros	10
Diagonal Average	2
Non-diagonal Average	1
Diagonal Dominance	0.4

Important Times:

Trilinos Assembly Time	3.16e-02 ms
PETSc Assembly Time	1.39e-01 ms
ITL Assembly Time	2.85e-03 ms
Hypre Assembly Time	9.70e-03 ms
QQQ Assembly Time	1.34e-02 ms
Time for Analysis	1.52e-01 ms

Results

BICGstab of **Hypre** encountered the error *"BiCGSTAB broke down!! gamma=0"* described earlier in Section 3.5.2. All other solvers solved the equation system (Table A.3).

5.1.4 Rank 5

The rank 5 matrix has the form:

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

Matrix Properties:

Dimension	5
Symmetric	YES
Positive Definite	YES
Value Range	-1 to 2
Absolute Minimum	1
Absolute Maximum	2
Diagonal Entries	5
Diagonal Zeros	0
Total Non-Zeros	13
Diagonal Average	2
Non-diagonal Average	1
Diagonal Dominance	0.384

Important Times:

Trilinos Assembly Time	3.25e-02 ms
PETSc Assembly Time	1.46e-01 ms
ITL Assembly Time	3.35e-03 ms
Hypre Assembly Time	1.05e-02 ms
QQQ Assembly Time	1.42e-02 ms
Time for Analysis	1.56e-01 ms

Results

All solvers succeeded in solving the equation system as shown in Table A.4.

5.1.5 Rank 10

The rank 10 version of the matrix has the form:

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

Matrix Properties:

Dimension	10
Symmetric	YES
Positive Definite	YES
Value Range	-1 to 2
Absolute Minimum	1
Absolute Maximum	2
Diagonal Entries	10
Diagonal Zeros	0
Total Non-Zeros	28
Diagonal Average	2
Non-diagonal Average	1
Diagonal Dominance	0.3857

Important Times:

Trilinos Assembly Time	4.08e-02 ms
PETSc Assembly Time	1.46e-01 ms
ITL Assembly Time	6.00e-03 ms
Hypre Assembly Time	1.49e-02 ms
QQQ Assembly Time	1.56e-02 ms
Time for Analysis	1.64e-01 ms

Results

All solvers succeeded in computing a solution as shown in Table A.5.

5.2 Hilbert Matrices

The Hilbert matrix is symmetric, positive definite and ill-conditioned, especially for large dimensions. The matrix elements are built by:

$$h_{ij} = \frac{1}{i+j-1} \tag{5.1}$$

Normally it is used for testing solvers which build the inverse of a matrix (the inverse is known and has integer values). Here it is used to see the behavior of the iterative solvers on an ill-conditioned matrix.

5.2.1 Rank 4

The Hilbert matrix for rank 4 has the form:

$$A = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

Matrix Properties:

Dimension	4
Symmetric	YES
Positive Definite	YES
Value Range	-0.142857 to 1
Absolute Minimum	0.142857
Absolute Maximum	1
Diagonal Entries	4
Diagonal Zeros	0
Total Non-Zeros	16
Diagonal Average	0.419048
Non-diagonal Average	0.283333
Diagonal Dominance	0.25

Important Times:

Trilinos Assembly Time	3.43e-02 ms
PETSc Assembly Time	1.54e-01 ms
ITL Assembly Time	3.30e-03 ms
Hypre Assembly Time	1.01e-02 ms
QQQ Assembly Time	1.37e-02 ms
Time for Analysis	2.35e-01 ms

Results

Without a preconditioner the **CG** and **BICGstab** solvers are not able to calculate the solution in the given iteration limit (see Table A.6). The final residual of the solutions is $\approx 1 \cdot 10^{-8}$ and so an increase of the maximum iteration number would help to get closer to the desired residual.

The **Jacobi** preconditioner is only useful for the **GMRES** solver. The matrix is dense, so the **Jacobi** preconditioner would not be a considerable option anyway.

All solvers are able to solve the system with the **ILU** preconditioner except **BICGstab** of **Hypre**, which encountered the same error already discussed for the Laplace matrices rank 2 to 4 (Section 5.1).

The **IC** preconditioner performs perfectly here to make it possible for all solvers to converge. The reason is that the matrix meets the requirements for **IC**.

5.2.2 Rank 10

The matrix form is shown in Figure 5.1.

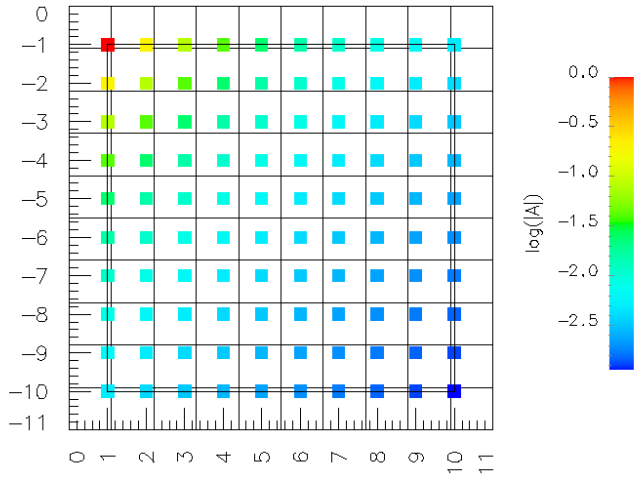

Figure 5.1: Hilbert matrix rank 10: Dimension = 10; **Value Range:** 0.0526316 to 1; **Absolute Minimum** = 0.0526316, **Absolute Maximum** = 1. (**NOTE:** The row indices are negative to simplify the visualization of the matrix in the right shape.)

Matrix Properties:

Symmetric	YES
Positive Definite	YES
Diagonal Entries	10
Diagonal Zeros	0
Total Non-Zeros	100
Diagonal Average	0.213326
Non-diagonal Average	0.124913
Diagonal Dominance	0.1

Important Times:

Trilinos Assembly Time	8.29e-02 ms
PETSc Assembly Time	1.84e-01 ms
ITL Assembly Time	3.22e-02 ms
Hypre Assembly Time	2.09e-02 ms
QQQ Assembly Time	2.36e-02 ms
Time for Analysis	1.78e-01 ms

Results

The system without a preconditioner is only solvable with **GMRES**, although **BICGstab** comes close with a final residual of around $1 \cdot 10^{-11}$ (see Table A.7).

The **Jacobi** preconditioner is unsuitable for this matrix. Only **Trilinos GMRES** is able to compute a solution with it. The reasons are that the matrix has a low diagonal dominance and also the diagonal average is too small.

The **ILU** preconditioner fails only for **Hypre CG**. The rest of the solvers are able to compute the solution of the equation system to a satisfying accuracy.

The **IC** preconditioner increases the performance of all solvers, letting them all converge. The reason is that the matrix is symmetric an positive definite.

5.2.3 Rank 100

The Hilbert matrix rank 100 has the form shown in Figure 5.2.

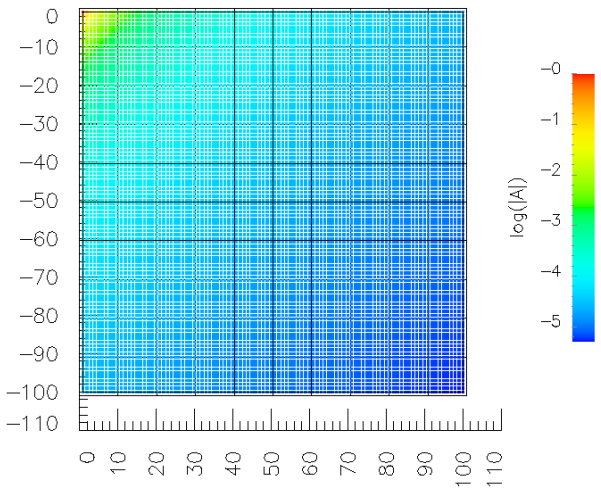

Figure 5.2: Hilbert matrix rank 100: Dimension = 100; **Value Range:** 0.00502513 to 1; **Absolute Minimum** = 0.00502513, **Absolute Maximum** = 1.

Matrix Properties:

Symmetric	YES
Positive Definite	YES
Diagonal Entries	100
Diagonal Zeros	0
Total Non-Zeros	10000
Diagonal Average	0.0328434
Non-diagonal Average	0.0136208
Diagonal Dominance	0.01

Important Times:

Trilinos Assembly Time	7.66 ms
PETSc Assembly Time	54.54 ms
ITL Assembly Time	5.81 ms
Hypre Assembly Time	5.11 ms
QQQ Assembly Time	1.69 ms
Time for Analysis	5.49 ms

Results

Without a preconditioner all solvers are able to compute a solution (see Table A.8).

With the **Jacobi** preconditioner only the **CG** solvers and **Trilinos GMRES** can calculate an accurate solution. The diagonal dominance is only one percent for this matrix. **BICGstab** of **Trilinos** and **ITL** abort the iteration because the vectors $\hat{\mathbf{r_0}}$ and $\mathbf{r_k}$ are orthogonal in the terminating iteration step (see Sections 3.1.2 and 3.3.2.

The **ILU** preconditioner fails for the **PETSc** and **Hypre** implementations. Their default values do not fit this matrix and so the solvers need too many iterations.

The **IC** preconditioners all fail although the **PETSc** solvers **BICGstab** and **GMRES** reach an accurate residual (the solver on the other hand returns that it has failed). For **CG** and **BICGstab** of **ITL** and **Trilinos** the residual yields ”nan” values. For **GMRES** of **Trilinos** the Hessenberg matrix is ill-conditioned caused by a worse preconditioning.

Performance

The solvers that reached the best speed performance and additionally computed an accurate solution are shown in Figure 5.3.

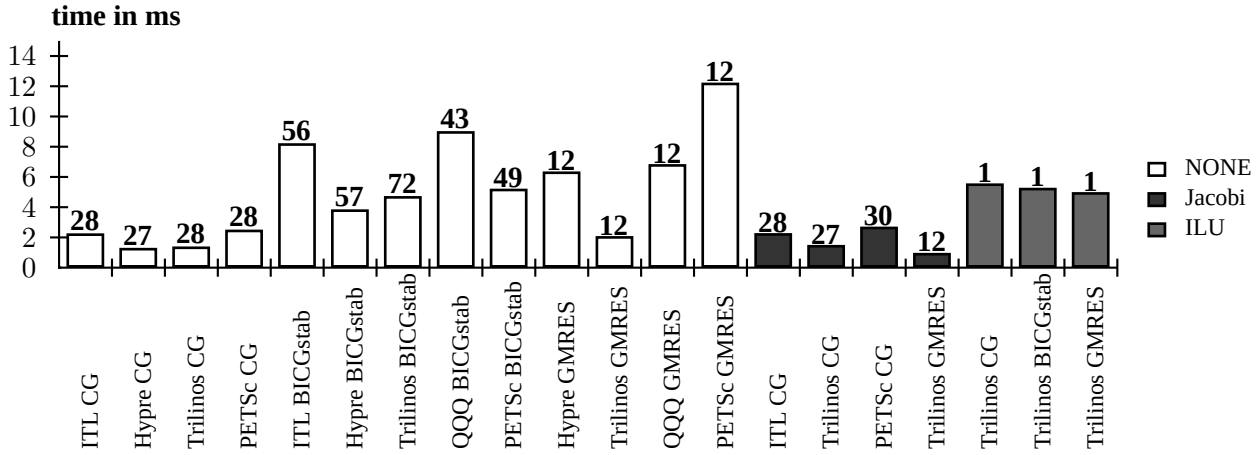

Figure 5.3: Performance for Hilbert rank 100: The fastest solver for this matrix is **Trilinos GMRES** with **Jacobi**. **CG** without preconditioner is also very fast considering that the matrix is not very well-conditioned. **ILU** preconditioning is very time inefficient for this matrix because it is dense and therefore much computational complexity is required. On the other hand only one iteration step is required afterwards. (**Note:** The number above each bar stands for the iterations used for computing the solution.)

5.2.4 Rank 1000

The Hilbert matrix rank 1000 has the form shown in Figure 5.4.

Matrix Properties:

Symmetric	YES
Positive Definite	YES
Diagonal Entries	1000
Diagonal Zeros	0
Total Non-Zeros	1000000
Diagonal Average	0.00443563
Non-diagonal Average	0.00138274
Diagonal Dominance	0.001

Important Times:

Trilinos Assembly Time	2.168 s
PETSc Assembly Time	429.678 s
ITL Assembly Time	0.991 s
Hypre Assembly Time	1.106 s
QQQ Assembly Time	0.217 s
Time for Analysis	0.743 s

Results

Without a preconditioner all solvers except **PETSc GMRES** (the solution reached an accuracy of $1.733 \cdot 10^{-12}$) are able to compute an accurate solution (see Table A.9).

The **Jacobi** preconditioner helps to compute a solution for all **CG** solvers and the **Trilinos GMRES**. For **BICGstab ITL** exceeds the maximum iterations, **Trilinos** encounters a breakdown because the vectors $\hat{\mathbf{r_0}}$ and $\mathbf{r_k}$ become orthogonal and **PETSc** reaches a residual bigger than its default divergence value.

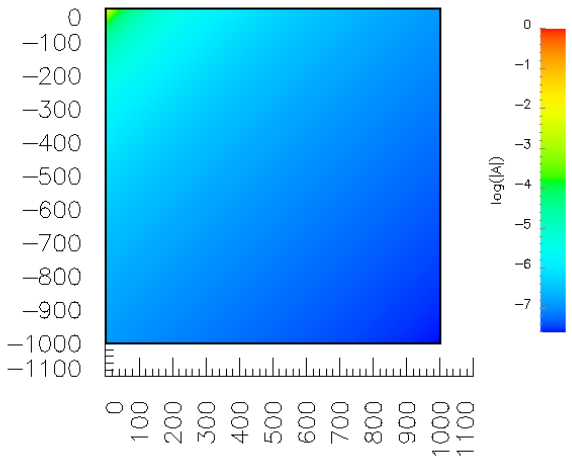

Figure 5.4: Hilbert matrix rank 1000: Dimension = 1000; **Value Range:** 0.00050025 to 1; **Absolute Minimum** = 0.00050025, **Absolute Maximum** = 1.

The **ILU** preconditioner of **PETSc** and **QQQ** fails completely (although the **QQQ** package returned that it solved the equation system for both solvers). **Trilinos ILU** solves with all solvers. **Hypre** and **ITL** only solve with one solver each (**Hypre GMRES** and **ITL BICGstab**).

The **IC** preconditioner fails completely although the **PETSc GMRES** computes a reasonable solution (but states that it has not solved the system, this seems to be a different loss of precision effect. The true residual computed after solving is more accurate than the recursive residual computed during the solving process (see Section 6).

Performance

The solver that solved the equation system fastest and reached the desired accuracy is **Hypre GMRES without** preconditioner (Figure 5.5).

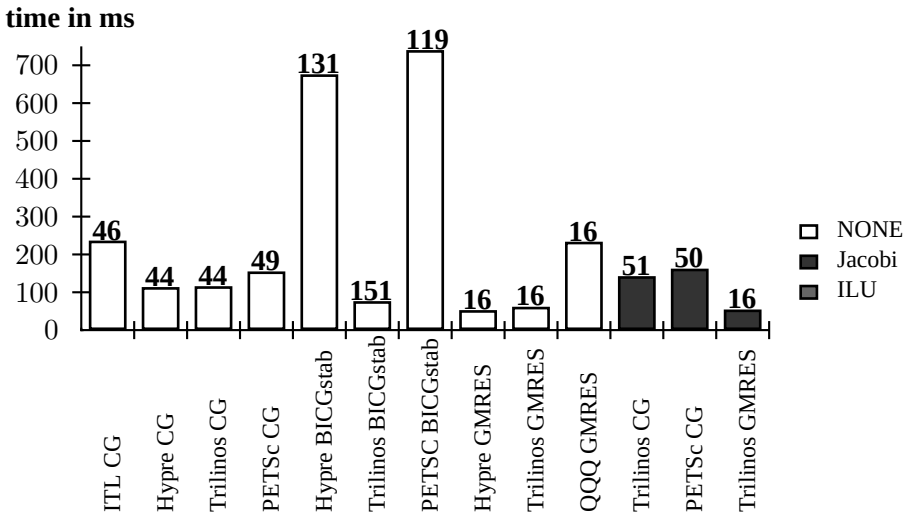

Figure 5.5: Hilbert rank 1000 performance: No preconditioner is able to improve the speed performance for this matrix. The fastest solver (**Hypre BICGstab**) uses no preconditioner. Considering that this is a dense matrix, using a preconditioner is associated with a high computational complexity.

5.3 Fidap Matrices

> *A visual repository of test data for use in comparative studies of algorithms for numerical linear algebra, featuring nearly 500 sparse matrices from a variety of applications, as well as matrix generation tools and services.* [18]

For testing the solvers, a few matrices have been downloaded from Matrix-market. These vary in form and range of values. Only matrices with an available right-hand side vector have been chosen.

The Fidap matrices are all generated by the Fidap package and are finite element modeling matrices (Source: Isaac Hasbani (Fluid Dynamics International), Barry Rackner (Minnesota Supercomputer Center))

5.3.1 Fidap001

The first matrix of the Fidap set has the form in Figure 5.6.

Matrix Properties:

Symmetric	YES
Positive Definite	NO
Diagonal Entries	181
Diagonal Zeros	35
Total Non-Zeros	3733
Diagonal Average	0.038429
Non-diagonal Average	0.00182772
Diagonal Dominance	0.048

Important Times:

Trilinos Assembly Time	2.72 ms
PETSc Assembly Time	16.98 ms
ITL Assembly Time	2.07 ms
Hypre Assembly Time	0.68 ms
QQQ Assembly Time	0.61 ms
Time for Analysis	2.76 ms

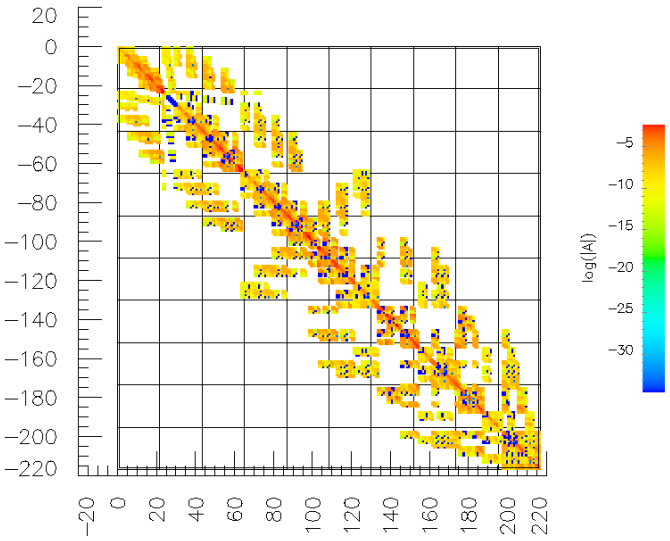

Figure 5.6: Matrix Fidap001: Dimension = 216; **Value Range:** -0.0302009 to 0.104451; **Absolute Minimum** = 5.96311e-21, **Absolute Maximum** = 0.104451.

Results

The **CG** and **BICGstab** solvers are not useful because the matrix is not positive definite (Table A.10). For the **CG** solvers this characteristic is discovered by the **PETSc** and **Trilinos** package breaking, the iteration.

The **Jacobi** preconditioner has problems because there are zeros on the main diagonal and the diagonal dominance is too small. Also the diagonal average is smaller than one.

The **ILU** preconditioner fails totally for the **ITL** package by throwing an exception caused by the zeros on the main diagonal. For the **BICGstab** only the **Trilinos** package has problems with **ILU**.

The **IC** preconditioner also throws an exception when using the **ITL** package because of the zeros on the main diagonal. The **Trilinos** and **PETSc** packages are able to solve this matrix with **BICGstab** and **GMRES** with the **IC**.

Performance

Trilinos BICGstab with **IC** preconditioner was the fastest solver for this matrix (Figure 5.7). The performance of the different **GMRES** implementations is also shown.

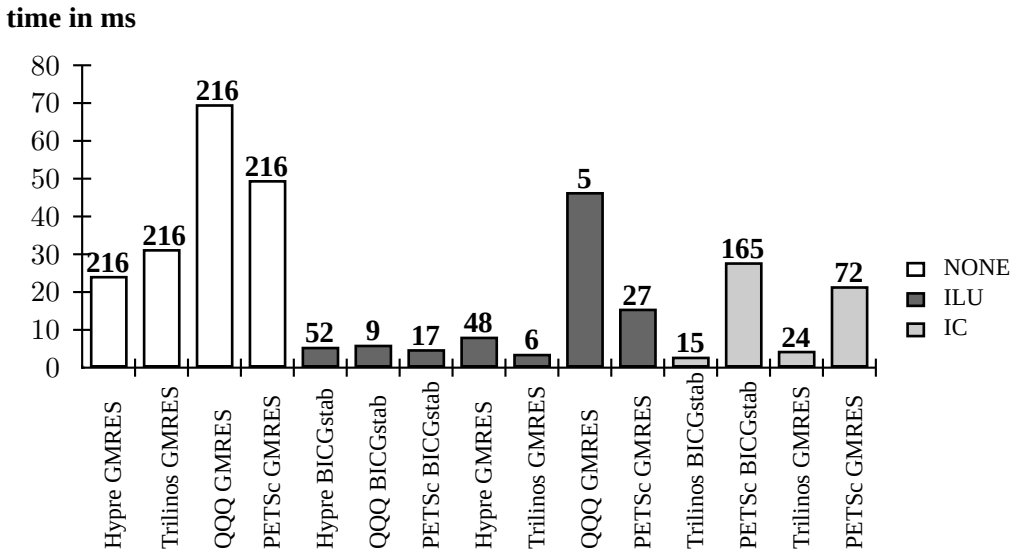

Figure 5.7: Fidap001 performance: Trilinos BICGstab with **IC** preconditioner solves this system fastest. The matrix is not positive definite so the **BICGstab** would not be chosen by the automatic solver control. The options chosen would be a **GMRES** solver with the **IC** preconditioner of **Trilinos**, which is the third fastest solver. For **GMRES** the **ILU** preconditioner would yield less iterations and thus be faster. The **GMRES** solvers without a preconditioner show the efficiency of the different code implementations.They all need 216 iterations but their speeds are different. **Hypre** is fastest, followed by **Trilinos** and **GMRES**. The slowest code implementation is from **QQQ**. This is also shown for the **ILU** preconditioner where **QQQ GMRES** needs 5 iterations but performs slowest.

5.3.2 Fidap002

The second matrix which was chosen from the Fidap set has the form in Figure 5.8.

Matrix Properties:

Symmetric	YES
Positive Definite	NO
Diagonal Entries	441
Diagonal Zeros	0
Total Non-Zeros	26807
Diagonal Average	6.03195e+06
Non-diagonal Average	680862
Diagonal Dominance	0.016

Important Times:

Trilinos Assembly Time	19.44 ms
PETSc Assembly Time	93.65 ms
ITL Assembly Time	17.10 ms
Hypre Assembly Time	10.74 ms
QQQ Assembly Time	4.28 ms
Time for Analysis	16.16 ms

Results

The equation system was not solvable to the accuracy of $1 \cdot 10^{-12}$ (see Table A.11). The **Trilinos** package encountered a loss of precision several times. Other solver packages returned that they have solved the system for some solving options, but the residual of the solution does not have the desired accuracy. Considering the matrix properties, an **IC** preconditioner with **GMRES** solver would be the choice of solving options. The loss of precision error will be investigated further in Chapter 6.

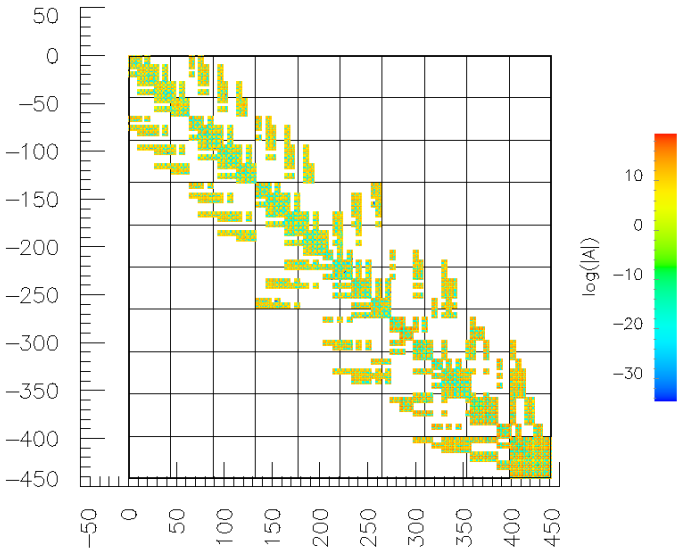

Figure 5.8: Matrix Fidap002: Dimension = 441; **Value Range:** -3.38963e+08 to 1.71342e+08; **Absolute Minimum** = 3.58608e-25, **Absolute Maximum** = 3.38963e+08.

Performance

There are no performance results for this matrix because no solver was able to compute the solution to the desired accuracy.

5.3.3 Fidap005

The third matrix from the Fidap set has the form shown in Figure 5.9.

Matrix Properties:

Symmetric	YES
Positive Definite	NO
Diagonal Entries	27
Diagonal Zeros	0
Total Non-Zeros	279
Diagonal Average	954735
Non-diagonal Average	280423
Diagonal Dominance	0.096

Important Times:

Trilinos Assembly Time	0.172 ms
PETSc Assembly Time	0.369 ms
ITL Assembly Time	9.52e-02 ms
Hypre Assembly Time	4.64e-02 ms
QQQ Assembly Time	4.45e-02 ms
Time for Analysis	0.375 ms

Results

The **Jacobi** preconditioner failed totally for this matrix (Table A.12). With the **ILU** and **IC** preconditioner some packages returned that they solved the equation system. But the relative residual is not below the desired accuracy. **Trilinos** encountered a loss of precision a few times.

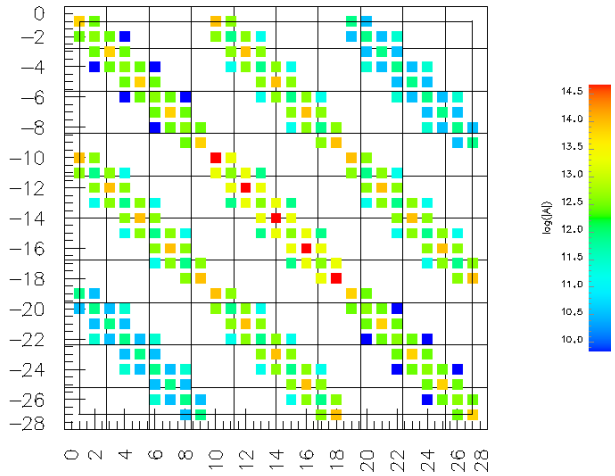

Figure 5.9: Matrix Fidap005: Dimension = 27; **Value Range:** -1.18519e+06 to 2.37038e+06; **Absolute Minimum** = 18518.6, **Absolute Maximum** = 2.37038e+06.

Performance

There are no performance results for this matrix because no solver was able to compute an accurate solution.

5.3.4 Fidapm05

The fourth matrix of the Fidap set has the form in Figure 5.10.

Matrix Properties:

Symmetric	YES
Positive Definite	NO
Diagonal Entries	27
Diagonal Zeros	15
Total Non-Zeros	445
Diagonal Average	2.72593
Non-diagonal Average	0.209569
Diagonal Dominance	0.06

Important Times:

Trilinos Assembly Time	0.320 ms
PETSc Assembly Time	0.403 ms
ITL Assembly Time	0.192 ms
Hypre Assembly Time	7.725e-02 ms
QQQ Assembly Time	7.395e-02 ms
Time for Analysis	0.503 ms

Results

Without being positive definite the **CG** and **BICGstab** solvers are not able to compute a solution without preconditioner (see Table A.13).

The **Jacobi** preconditioner is not able to improve the solving performance due to the zeros on the main diagonal and lets all solvers fail.

With the **ILU** preconditioner the **QQQ BICGstab** and the **PETSc BICGstab** are able to solve the system. On the other hand, the solution of the **GMRES** solver of **QQQ** is not accurate. The **IC** precon-

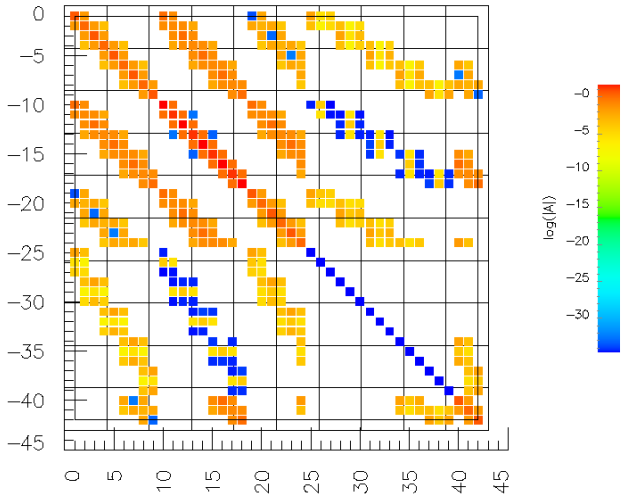

Figure 5.10: Matrix Fidapm005: Dimension = 42; **Value Range:** -1.06667 to 5.68889; **Absolute Minimum** = 1.38778e-17, **Absolute Maximum:** = 5.68889.

ditioner also fails totally.

Performance

The fastest solvers for this equation system which also offer accurate solutions are **QQQ BICGstab** and **PETSc BICGstab** both with **ILU** preconditioner (Figure 5.11).

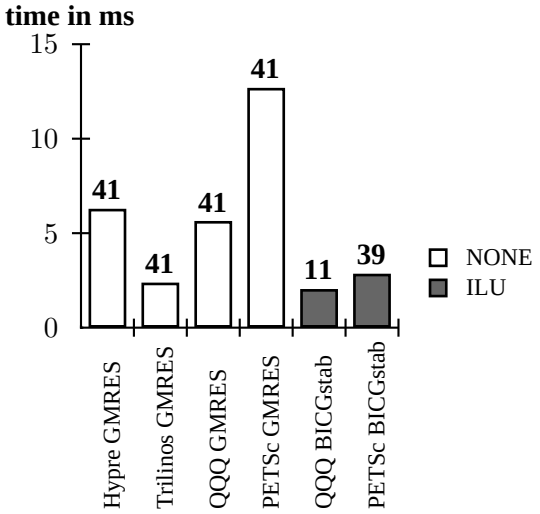

Figure 5.11: Fidapm05 performance: Two **BICGstab** solvers lead the performance results for this matrix. Since this matrix is not positive definite, each of these solvers would not be an option for an automatic solver control. **GMRES** solves this system without a preconditioner. All packages require the same number of iterations to reach an accurate solution. Here **Trilinos** is fastest, followed by **QQQ** and **Hypre**. **PETSc** has the slowest **GMRES** implementation for this matrix.

5.3.5 Fidap027

The fifth matrix chosen from the Fidap set has the form in Figure 5.12.

Matrix Properties:

Symmetric	NO
Positive Definite	NO
Diagonal Entries	728
Diagonal Zeros	192
Total Non-Zeros	23775
Diagonal Average	0.0182834
Non-diagonal Average	0.000924957
Diagonal Dominance	0.032

Important Times:

Trilinos Assembly Time	29.02 ms
PETSc Assembly Time	920.81 ms
ITL Assembly Time	25.92 ms
Hypre Assembly Time	13.44 ms
QQQ Assembly Time	6.26 ms
Time for Analysis	29.02 ms

Results

The matrix is symmetric but not positive definite so the **CG** and **BICGstab** fail (see Table A.14). Without a preconditioner the **PETSc GMRES** fails, in contrast to the other **GMRES** solvers.

The **Jacobi** preconditioner is not usable for this matrix, mainly because of the zeros on the main diagonal.

The zeros on the main diagonal also let the Trilinos **ILU** preconditioner fail. The **Hypre ILU** is not able to calculate a solution but does not deliver a numerical breakdown. The **QQQ BICGstab** with **ILU** preconditioner solves the equation system. The **PETSc** version of this solver only reaches a solution close to the accuracy desired. On the other hand the **GMRES** solvers of these packages are returning an inaccurate solution.

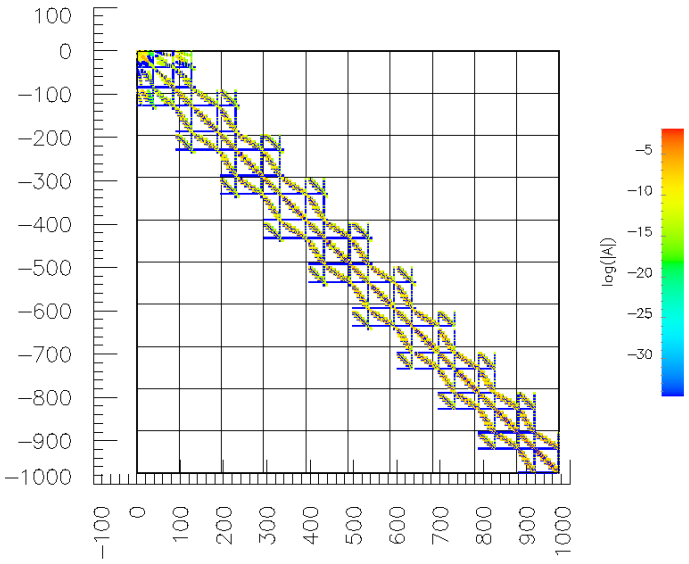

Figure 5.12: Matrix Fidap027: Dimension = 974; **Value Range:** -0.0706368 to 0.156271; **Absolute Minimum** = 3.2967e-35, **Absolute Maximum** = 0.156271.

Performance

This is another matrix that is not positive definite but has **BICGstab** as fastest solver (Figure 5.13).

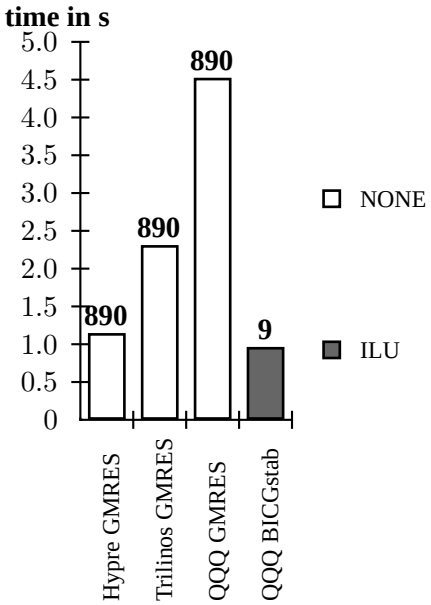

Figure 5.13: Fidap027 performance: QQQ BICGstab with **ILU** solved fastest, although the matrix is not positive definite. **Hypre GMRES** solves the system without a preconditioner slightly slower than the **QQQ BICGstab**.

5.3.6 Fidap028

The sixth matrix of the Fidap set has the form in Figure 5.14.

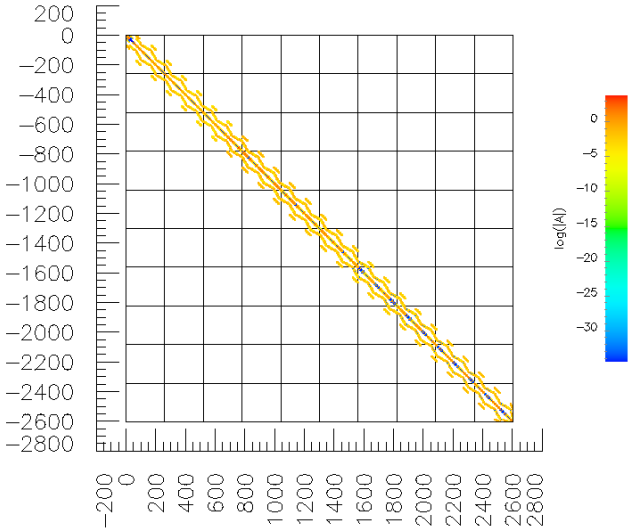

Figure 5.14: Matrix Fidap028: Dimension = 2603; **Value Range:** -24.6163 to 49.3352; **Absolute Minimum** = 6.31089e-30, **Absolute Maximum** = 49.3352.

Matrix Properties:

Symmetric	NO
Positive Definite	NO
Diagonal Entries	1853
Diagonal Zeros	750
Total Non-Zeros	70861
Diagonal Average	8.6464
Non-diagonal Average	0.485261
Diagonal Dominance	0.026

Important Times:

Trilinos Assembly Time	51.36 ms
PETSc Assembly Time	991.75 ms
ITL Assembly Time	51.70 ms
Hypre Assembly Time	18.67 ms
QQQ Assembly Time	12.18 ms
Time for Analysis	59.29 ms

Results

The **CG** and **BICGstab** solvers do not work because the matrix is not positive definite (see Table A.15).

The **Jacobi** preconditioner fails due to the zeros on the main diagonal.

The **ILU** preconditioners of **QQQ** and **PETSc** help the **BICGstab** and **GMRES** solvers to calculate an tolerable solution. However the desired accuracy is not reached with any of the solvers.

Performance

There are no performance results for this matrix because no solver was able to compute an accurate solution.

5.4 Sherman Matrices

The Sherman set consists of matrices used for oil reservoir simulation by Andy Sherman (Source: Andy Sherman, Nolan and Associates, Houston, TX.).

5.4.1 Sherman2

The first matrix of the Sherman set that was chosen has the form in Figure 5.15.

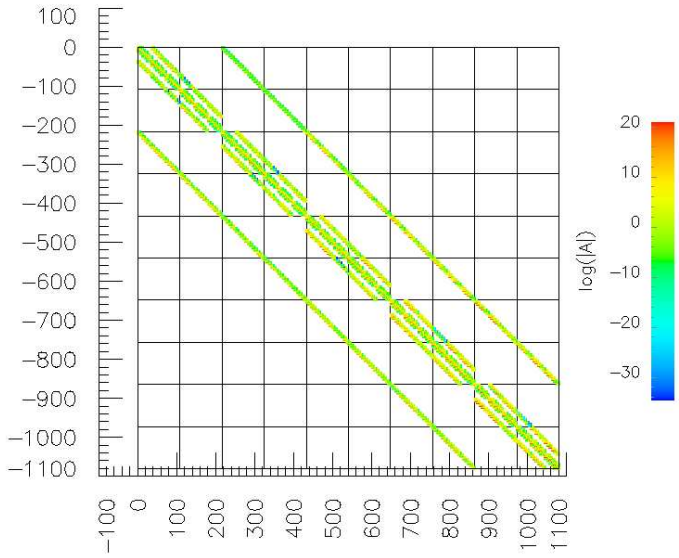

Figure 5.15: Matrix Sherman2: Dimension = 1080; **Value Range:** -1.11233e+09 to 1.33985e+09; **Absolute Minimum** = 1.48821e-17, **Absolute Maximum** = 1.33985e+09.

Matrix Properties:

Symmetric	NO
Positive Definite	NO
Diagonal Entries	1080
Diagonal Zeros	0
Total Non-Zeros	22958
Diagonal Average	2.56142e+06
Non-diagonal Average	4.3234e06
Diagonal Dominance	0.047

Important Times:

Trilinos Assembly Time	14.74 ms
PETSc Assembly Time	98.56 ms
ITL Assembly Time	12.05 ms
Hypre Assembly Time	3.72 ms
QQQ Assembly Time	3.26 ms
Time for Analysis	20.21 ms

Results

The matrix is neither symmetric nor positive definite so it is obvious that **CG** and **BICGstab** do not work (Table A.16). Without a preconditioner only the **GMRES** solvers converge except the **GMRES** of the **PETSc** package which uses too many iterations.

The **Jacobi** preconditioner completely fails letting the **GMRES** solvers diverge which were able to solve it without the preconditioner. The low diagonal dominance of 4.7 percent is a good reason for not using it.

The **ILU** preconditioner lets the **BICGstab** and **GMRES** solvers of the packages be able to get to an accurate solution except the **QQQ** versions. These both calculate a wrong solution with a residual $\gg 1$ which indicates that the **QQQ ILU** preconditioner failed here.

Performance

The fastest solvers for this matrix are **PETSc BICGstab** and **PETSc GMRES** (Figure 5.16).

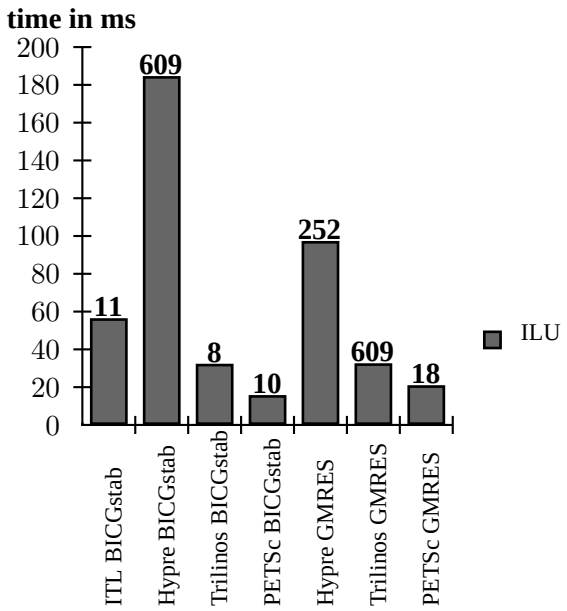

Figure 5.16: Sherman2 performance: PETSC has the two fastest solvers for this matrix with **BICGstab** and **GMRES**, both using the **ILU** preconditioner. The **ILU** preconditioner of **Hypre** and **Trilinos** perform equally well here because the **GMRES** of both packages needed 609 iterations. For this matrix the default values of the **PETSc ILU** preconditioner fit best, decreasing the number of iterations the most.

5.4.2 Sherman3

The second matrix used from the Sherman set is shown in the Figure 5.17.

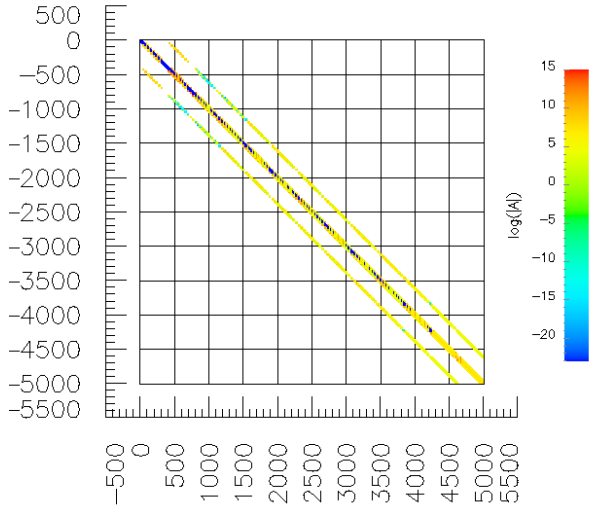

Figure 5.17: Matrix Sherman3: Dimension = 5005; **Value Range:** -1.72445e+06 to 3.44925e+06; **Absolute Minimum** = 1e-10, **Absolute Maximum** = 3.44925e+06.

Matrix Properties:

Symmetric	NO
Positive Definite	YES
Diagonal Entries	5005
Diagonal Zeros	0
Total Non-Zeros	20033
Diagonal Average	14610.2
Non-diagonal Average	4864.83
Diagonal Dominance	0.249

Important Times:

Trilinos Assembly Time	8.96 ms
PETSc Assembly Time	1.266 s
ITL Assembly Time	3.02 ms
Hypre Assembly Time	4.34 ms
QQQ Assembly Time	1.95 ms
Time for Analysis	12.45 ms

Results

The matrix is not symmetric but is positive definite. The **BICGstab** is not able to solve the equation system without a preconditioner (see Table A.17). The **GMRES** solvers, on the other hand, deliver an inaccurate solution.

With the **Jacobi** preconditioner the performance is slightly better. All solver types are reaching a tolerable accuracy around $1 \cdot 10^{-11}$, but no solver reaches the convergence limit, although some solvers state that they are successful.

The **ILU** preconditioner also encounters loss of precision errors (in case of the **QQQ GMRES** this error is around $1 \cdot 10^{7}$). The only solvers which did not encounter a considerable round-off error are the **Hypre BICGstab**, **Hypre GMRES** and **Trilinos GMRES**.

Performance

There are only three solvers that solve the equation system in an acceptable amount of time (Figure 5.18).

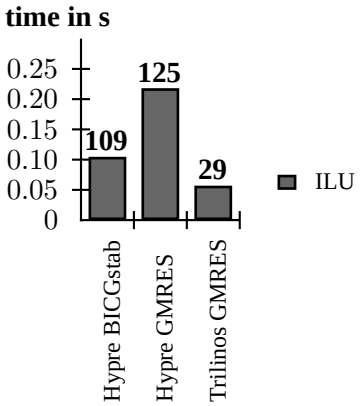

Figure 5.18: Sherman3 performance: Trilinos GMRES with an **ILU** preconditioner is the fastest solver for this matrix. **Trilinos BICGstab** would have been slightly faster but it encountered a loss of precision error so it is not in the diagram. The **ILU** preconditioner of **Trilinos** decreases the number of iterations more than **Hypre ILU** and therefore the **GMRES** solver of **Trilinos** is fastest. The matrix is positive definite so the **BICGstab** solver would be the choice for an automatic solver control.

5.4.3 Sherman5

The third matrix chosen from the Sherman set has the form shown in Figure 5.19.

Matrix Properties:

Symmetric	NO
Positive Definite	NO
Diagonal Entries	3312
Diagonal Zeros	0
Total Non-Zeros	20793
Diagonal Average	60.4114
Non-diagonal Average	23.1693
Diagonal Dominance	0.159

Important Times:

Trilinos Assembly Time	12.33 ms
PETSc Assembly Time	445.19 ms
ITL Assembly Time	7.87 ms
Hypre Assembly Time	3.67 ms
QQQ Assembly Time	2.54 ms
Time for Analysis	15.98 ms

Results

The matrix is neither symmetric nor positive definite, so the **CG** and **BICGstab** solvers fail without a preconditioner (see Table A.18). The **GMRES** solvers are able to solve the system except the **PETSc GMRES** which exceeds the maximum iterations.

The **Jacobi** preconditioner encounters a loss of precision and delivers an inaccurate solution for **BICGstab** and **GMRES**.

The **ILU** preconditioner also encounters a loss of precision except for the **Hypre BICGstab** and the **GMRES** solvers of **Trilinos** and **Hypre**.

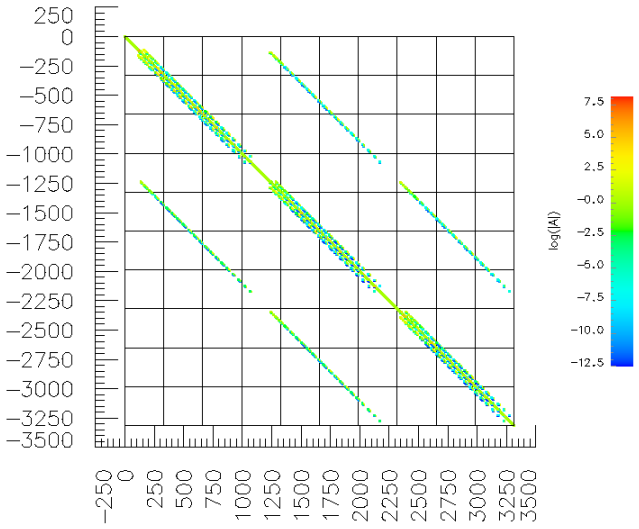

Figure 5.19: Matrix Sherman5: Dimension = 3312; **Value Range:** -3557.32 to 785.866; **Absolute Minimum** = 3.68974e-06, **Absolute Maximum** = 3557.32.

Performance

For this matrix only three solvers are able to calculate an accurate solution shown in Figure 5.20.

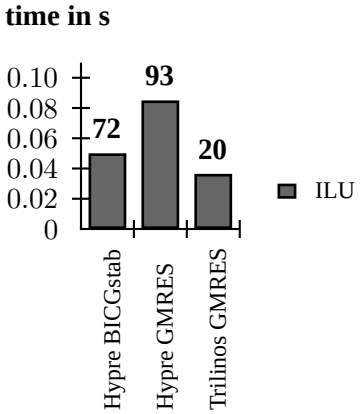

Figure 5.20: Sherman5 performance: The **ILU** preconditioner of **Trilinos** decreases the number of iterations more than **Hypre ILU** and therefore the **GMRES** solver of **Trilinos** is fastest. **Trilinos BICGstab** would have been slightly faster but encountered a loss of precision error so it is not shown in the Figure.

5.5 DRIVCAV Matrices

The matrices in the DRIVCAV set are from 2D modeling of fluid flow in driven cavities. These matrices are non-symmetric and indefinite and are normally hard to solve with **Krylov** solvers because of the difficulty in finding a good preconditioner. The intended use of these matrices is for testing iterative solvers. (Source: Andrew Chapman, University of Minnesota, chapman@msi.umn.edu)

5.5.1 E05r0100

The matrix e05r0100 has the form shown in Figure 5.21.

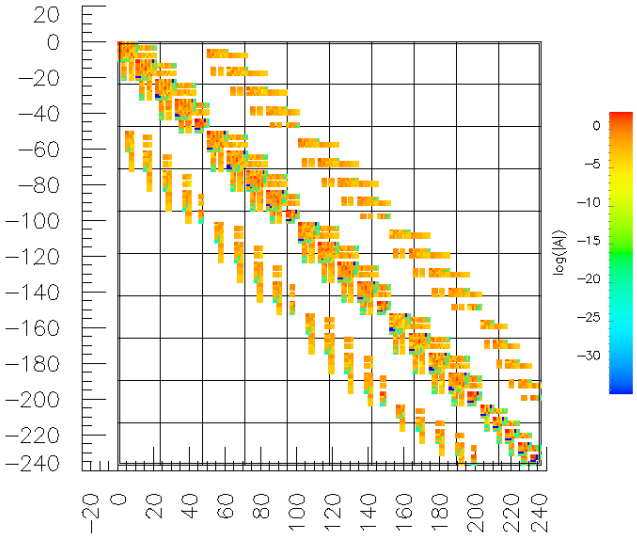

Figure 5.21: Matrix E05r0100: Dimension = 236; **Value Range:** -5.87668 to 10.1705; **Absolute Minimum** = 3.46945e-18, **Absolute Maximum** = 10.1705.

Matrix Properties:

Symmetric	NO
Positive Definite	NO
Diagonal Entries	162
Diagonal Zeros	74
Total Non-Zeros	5760
Diagonal Average	6.26831
Non-diagonal Average	0.29926
Diagonal Dominance	0.028

Important Times:

Trilinos Assembly Time	3.73 ms
PETSc Assembly Time	28.38 ms
ITL Assembly Time	2.88 ms
Hypre Assembly Time	1.16 ms
QQQ Assembly Time	0.83 ms
Time for Analysis	6.75 ms

Results

Without a preconditioner the equation system is only solvable by the **GMRES** solvers of **Hypre**, **Trilinos** and **QQQ** (see Table A.19). The matrix is not symmetric and not positive definite so it is obvious that **CG** and **BICGstab** fail.

The **Jacobi** preconditioner has problems solving this equation system because of the zeros on the main diagonal. Also the diagonal dominance is very low.

For the **ILU** preconditioner the only solver able to solve the system is the **QQQ BICGstab**. **PETSc BICGstab** (final residual=$1.01673 \cdot 10^{-12}$), **QQQ GMRES** (final residual = $2.93189 \cdot 10^{-12}$) and **PETSc GMRES** (final residual = $1.71488 \cdot 10-11$) state that they solve too, but their final residual is slightly greater than the desired accuracy. The zero pivot shifting of **PETSc** and the internal ordering of **QQQ ILU** are better strategies for this matrix than the zero pivot handling of **Trilinos** and **Hypre** which only replace the zeros. **Trilinos** states that the Hessenberg matrix is ill-conditioned which is due to the zero pivot handling.

Performance

The fastest solver for this matrix is **QQQ BICGstab** with **ILU** preconditioner as shown in Figure 5.22.

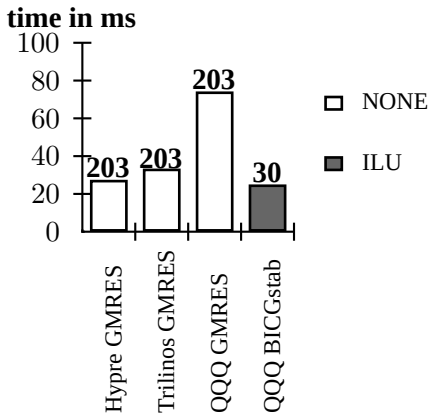

Figure 5.22: E05r0100 performance: QQQ BICGstab with an **ILU** preconditioner not only solved this system the fastest, but it was also the only one which solved with a preconditioner. Three **GMRES** solved without a preconditioner, whereby **Hypre** was fastest, followed by **Trilinos** and **QQQ**.

5.5.2 E20r5000

The second matrix tested of the DRIVCAV set has the form in Figure 5.23.

Matrix Properties:

Symmetric	NO
Positive Definite	NO
Diagonal Entries	3042
Diagonal Zeros	1199
Total Non-Zeros	126328
Diagonal Average	6.26187
Non-diagonal Average	2.37942
Diagonal Dominance	0.024

Important Times:

Trilinos Assembly Time	0.088 s
PETSc Assembly Time	3.456 s
ITL Assembly Time	0.094 s
Hypre Assembly Time	0.034 s
QQQ Assembly Time	0.022 s
Time for Analysis	0.230 s

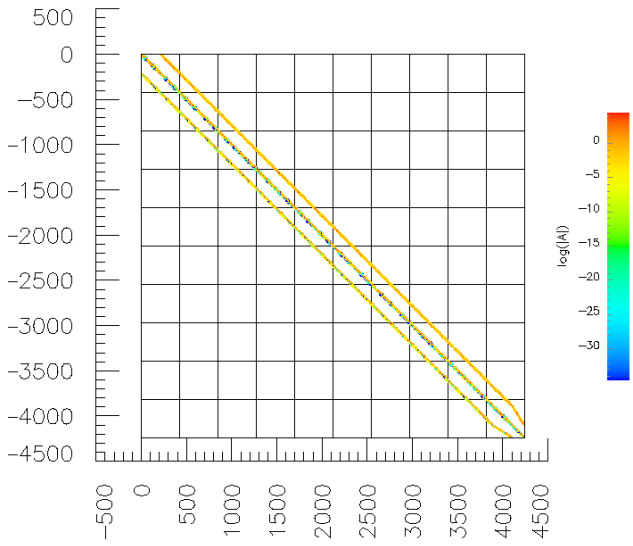

Figure 5.23: Matrix E20r5000: Dimension = 4241; **Value Range:** -74.7776 to 93.9809; **Absolute Minimum** = 3.25261e-19, **Absolute Maximum** = 93.9809.

Results

This matrix is a numerical challenge for the solvers. There are over 28 percent zeros on the main diagonal, so the preconditioners are working at their limits. Without a preconditioner, no solver is able to solve the system. The best residuals reached are around $1 \cdot 10^{-1}$ by the **GMRES** solver (see Table A.20).

The **Jacobi** preconditioner is not able to decrease the condition number enough to be able to solve the matrix.

The **ILU** preconditioner of **ITL** throws an exception because of the zeros on the main diagonal. The zero pivot handling of **Trilinos** also fails and leads to ”nan” residuals for the **CG** and **BICGstab** solvers. The only solvers which deliver a tolerable solution are the **QQQ GMRES** and **BICGstab** solvers which reach a residual of around $1 \cdot 10^{-8}$.

Performance

There are no performance results for this matrix because no solver was able to reach the desired accuracy for the solution.

5.6 Hamm Matrices

The matrices in this package are computer design components from Steve Hamm. (Source: Steve Hamm, Motorola Inc. Semiconductor Systems Design Technology, 3501 Bluestein Blvd. MD-M2, Austin, TX 78762 USA (hamm@ssdt-bluestein.sps.mot.com))

5.6.1 Add20

The first matrix in the Hamm set is the model of a 20-bit adder and has the form in Figure 5.24.

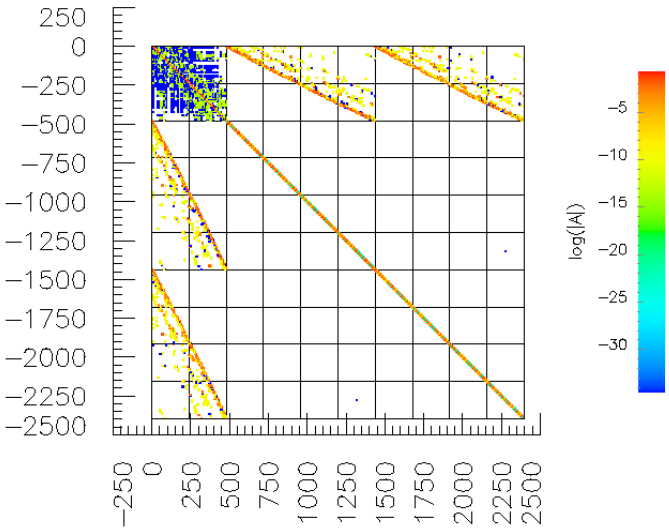

Figure 5.24: Matrix Add20: Dimension = 2395; **Value Range:** -0.285714 to 0.4981081; **Absolute Minimum** = 8.57445e-13, **Absolute Maximum** = 0.498101.

Matrix Properties:

Symmetric	NO
Positive Definite	YES
Diagonal Entries	2395
Diagonal Zeros	0
Total Non-Zeros	13139
Diagonal Average	0.140144
Non-diagonal Average	0.0237017
Diagonal Dominance	0.182

Important Times:

Trilinos Assembly Time	9.55 ms
PETSc Assembly Time	66.93 ms
ITL Assembly Time	5.75 ms
Hypre Assembly Time	4.75 ms
QQQ Assembly Time	2.15 ms
Time for Analysis	11.71 ms

Results

The matrix is positive definite so the **BICGstab** solver works for all packages except **Trilinos** which encounters a numerical breakdown (Table A.21) because the vectors $\hat{\mathbf{r}}_0$ and $\mathbf{r}_k$ become orthogonal.

The **ILU** preconditioner performs well for this matrix especially for the **Trilinos** package which is able to solve the system with all 3 solvers. All **BICGstab** and **GMRES** implementations solve with the **ILU** preconditioner.

Performance

Trilinos CG, **GMRES** and **BICGstab** with **ILU** preconditioner had the fastest performance (Figure 5.25).

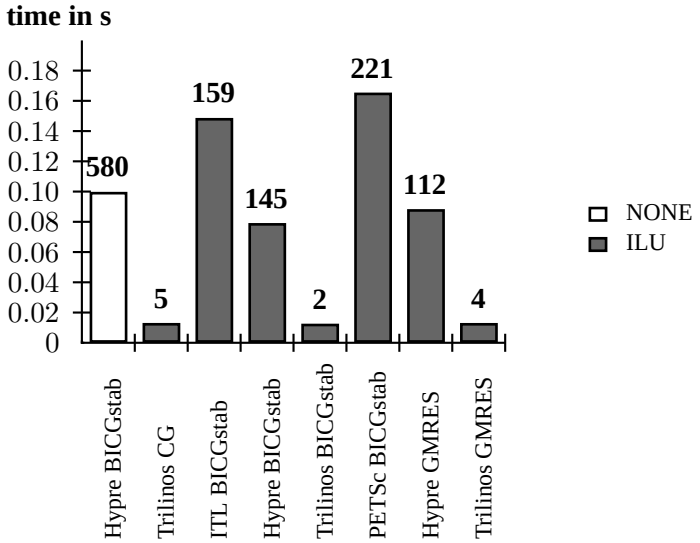

Figure 5.25: Add20 performance: The **ILU** preconditioner of **Trilinos** performs very well, reducing the iterations to 2,4 and 5 for the solvers. **Trilinos BICGstab** was able to solve the solution fastest using 2 iterations. The **ILU** preconditioners of the other packages did not improve the performance of their solvers that well.

5.6.2 Memplus

The second matrix in the Hamm set is the model of a memory circuit and has the form shown in Figure 5.26.

Matrix Properties:

Symmetric	NO
Positive Definite	NEARLY
Diagonal Entries	17758
Diagonal Zeros	0
Total Non-Zeros	97350
Diagonal Average	0.0257426
Non-diagonal Average	0.00447144
Diagonal Dominance	0.182

Important Times:

Trilinos Assembly Time	0.079 s
PETSc Assembly Time	3.592 s
ITL Assembly Time	0.051 s
Hypre Assembly Time	0.052 s
QQQ Assembly Time	0.019 s
Time for Analysis	0.135 s

Results

The matrix is nearly positive definite and three of the five **BICGstab** solvers are able to solve the equation system (see Table A.22). **Trilinos BICGstab** encounters a numerical breakdown because the vectors $\hat{\mathbf{r_0}}$ and $\mathbf{r_k}$ become orthogonal. The **ILU** preconditioner of **Trilinos** helps the **CG** solver to solve the system.

QQQ GMRES, **PETSC BICGstab** and **PETSC GMRES** with **ILU** stated that they solved the equation system, although the solutions are slightly above the desired accuracy caused by round-off errors

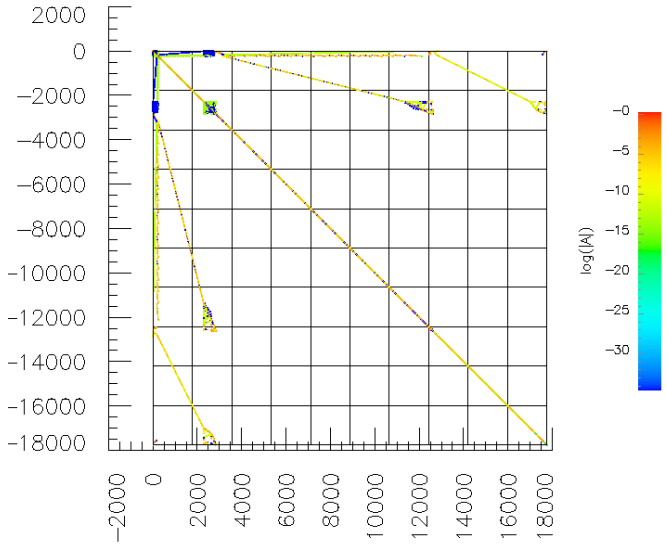

Figure 5.26: Matrix Memplus: Dimension = 17758; **Value Range:** -0.15873 to 140965; **Absolute Minimum** = 1.36431e-26, **Absolute Maximum** = 1.40965.

($1.44025 \cdot 10^{-12}$ for **QQQ**, 3.57992^{-12} for **PETSc GMRES** and $2.27627 \cdot 10^{-12}$ for **PETSc BICGstab**). **QQQ BICGstab** is able to determine an accurate solution.

Performance

The Trilinos **BICGstab** with **ILU** preconditioner is the fastest solver for this matrix as shown in Figure 5.27.

5.7 Saddle point matrix

A **Saddle point matrix** has the form [19]:

$$\mathbf{A} = \begin{bmatrix} \mathbf{B} & \mathbf{C} \\ \mathbf{C}^T & \mathbf{0} \end{bmatrix}$$

Such a matrix can lead to trouble using **Krylov** subspace methods because the zeros on the main diagonal lead to zero pivot errors. If such a system needs to be solved using these preconditioners, the matrix needs to be modified (e.g., exchange rows to obtain a diagonal without zeros). The saddle point matrix used for testing has the form:

$$A = \begin{bmatrix} 1 & 2 & 5 & 1 & 4 \\ 3 & 4 & 6 & 2 & 5 \\ 8 & 8 & 9 & 3 & 6 \\ 1 & 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 \end{bmatrix}$$

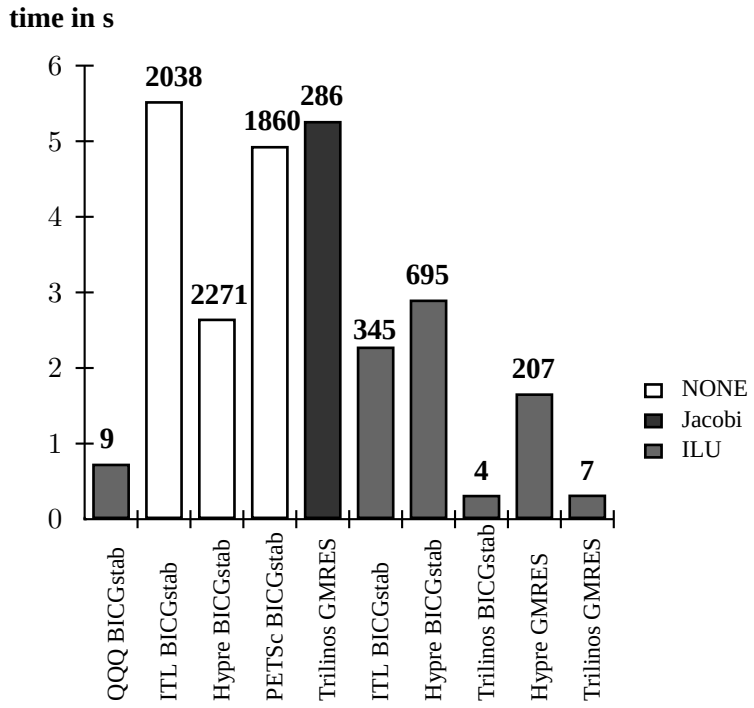

Figure 5.27: Memplus performance: The **Trilinos BICGstab** and **GMRES** solvers with an **ILU** preconditioner are the fastest here. For this matrix the **BICGstab** solver would be preferred because the matrix is defined as nearly positive definite. The **ILU** preconditioner of **Trilinos** reduces the iterations of the solving process to a minimum compared to the other packages except **QQQ BICGstab**.

Matrix Properties:

Dimension	5
Symmetric	NO
Positive Definite	NO
Value Range	1 to 9
Absolute Minimum	1
Absolute Maximum	9
Diagonal Entries	3
Diagonal Zeros	2
Total Non-Zeros	21
Diagonal Average	4.66667
Non-diagonal Average	4.11111
Diagonal Dominance	0.142

Important Times:

Trilinos Assembly Time	3.69e-02 ms
PETSc Assembly Time	0.15 ms
ITL Assembly Time	3.5e-03 ms
Hypre Assembly Time	1.14e-02 ms
QQQ Assembly Time	1.49e-02 ms
Time for Analysis	0.25 ms

Results

The matrix shows that all **Jacobi** preconditioners do have troubles caused by the zeros on the main diagonal, except the one from **PETSc** which solves the system together with the **GMRES** solver. With this matrix the **ILU** preconditioners do not have that much trouble, although the normally well performing **ILU** of **PETSc** is not able to compute an adequate solution in the given maximum iterations.

5.8 Overall Results

Several matrix properties influence the performance of the different solvers and preconditioners. Also different implementations of the solvers and preconditioners by the packages lead to different results during the solving of a linear equation system (e.g., different solving times, different residuals achieved, and different numbers of iterations).

There are two important matrix properties which have a big influence on the performance of a solving algorithm: **symmetry** and **positive definite**. The symmetry of the matrix is required for the **CG** solver and the **IC** preconditioner. They also need the matrix to be positive definite, although the **IC** preconditioner could solve systems which have no positive definite matrix as well. If the matrix is not positive definite this can be detected during solving by **Trilinos** and **PETSc**, which will lead to a breakdown of the solving process (e.g., see Section 5.3.1). **ITL** and **Hypre** do not detect this and therefore exceed the maximum number of iterations.

The **BICGstab** solver also requires a positive definite matrix. All matrices stated as nearly positive definite could be treated by the solver as well. Also several matrices that are not positive definite could be treated with BICGstab (but this is not predictable). The biggest problem of the **BICGstab** solver is that if the vectors $\mathbf{\hat{r}_0}$ and $\mathbf{r_k}$ become orthogonal (see **BICGstab** algorithm in Section 2.3.2), then the algorithm stops without convergence. **Trilinos** is the most sensitive package in this case (e.g., see Sections 5.3.1, 5.2.4, 5.6.1, and 5.6.2). **QQQ** avoids this problem, as described in Section 3.4.1.

All three preconditioners encounter problems if there is no handling of zero pivot elements. The solver packages use different strategies when a zero pivot element is encountered. For some preconditioners this is not even tested and the preconditioner is used anyway, resulting in a divergence of the solving process most of the time (after a division by zero or by a number close to zero). The **ITL** package checks whether all diagonal elements exist (and throws an exception if not, so the solving process is stopped and cannot be continued), but it does not check whether these are equal to zero (see Section 3.3). This leads to problems when a zero is inserted explicitly into the main diagonal. **Hypre** and **Trilinos** replace an encountered zero pivot, and so the solving process can be kept alive (**Trilinos** replaces a zero pivot with the *rownorm* (see Section 3.1.5) and **Hypre** uses the drop tolerance (see Section 3.5.4)). The **PETSc** package uses one of several different shifting options to eliminate zero pivots (Section 3.2), which seems to be the best method to deal with zero pivots (see Sections 5.3.2, 5.3.3 and 5.3.5). **QQQ** aborts the solving process if a zero pivot is encountered (see Section 3.4.3), although this rarely happens (never in the tests of this chapter), due to the internal sort algorithms used by the **ILU** factorization.

The comparison of the different implementations of the **ILU** factorization shows that the fill factor has great influence on the iterations until convergence is reached. The **Trilinos** package, with a fill ratio of 3, needs below 10 iterations most of the time until convergence is reached (e.g., see Section 5.6.1 and 5.6.2). On the other hand, the **PETSc** package with a fill ratio of 1 performs very much worse.

Trilinos is the only package which gives a warning that a solution may not be accurate. The other solver packages may return in their error code that the system was solved (e.g., for the matrices in Sections 5.3.2, 5.4.2 and 5.4.3), but it can happen that the solution is not accurate or totally wrong (e.g., if the residual is $\gg 1$). This is particularly important because if the user does not check whether the solution is accurate, the program using the solution may work with false parameters.

ITL is very easy to implement and does not need much knowledge of solvers or preconditioners because there are not many options which can be set. The disadvantage of this package is that there is no handling of zero pivots when they are encountered. The solving process cannot be continued. Also the **GMRES** algorithm, which is one of the most effective solving algorithms, has not been implemented yet.

Hypre has a limited choice of preconditioners (no **Jacobi** or **IC** preconditioner). The **BICGstab** algorithm has problems with small matrices ($<$ rank 5; e.g., see Sections 5.1.1 to 5.1.3). The zero pivot

handling of the **Hypre** preconditioner works but could be better. There are a few solving options which can be set to improve the performance of the solving and which require little knowledge of the solvers.

Trilinos has the fastest solvers and also is the only package that can detect a loss of precision. Additionally its **GMRES** algorithm has been implemented to terminate if the Hessenberg matrix is ill-conditioned (e.g., in Section 5.2.4). On the other hand, it has problems with zeros on the main diagonal if a preconditioner is used (e.g., see Sections 5.3.2 and 5.7). Several different options can be set to improve the solving process.

PETSc has the slowest solvers, regarding the time for solving a linear equation system, compared to the other packages (also the **GMRES** solver often uses more iterations than the implementations of the other packages), but, on the other hand, error detecting is well designed because every function returns an error code that is immediately checked. There are many options which can be altered to improve the solving process, but they also require a deep knowledge of the ongoing processes. **PETSc** also offers the best zero pivot handling of all the packages.

QQQ sometimes performs best on the matrices (e.g., Section 5.5.1); for other matrices it delivers completely wrong solutions (e.g., see Sections 5.4.1 and 5.4.2). It does not really offer zero pivot handling, but the **ILU** preconditioner uses sorting algorithms during the factorization which can prevent zero pivots most of the time. This also results in a larger amount of time for computing the **ILU** factorization than the implementations of other packages. **QQQ** also offers algorithms not available for the other packages, like pre-elimination or automatic scaling. However, a test showed that these do not improve the performance significantly if used with their default settings.

Figures 5.28 and 5.29 show how many of the given equation systems each package was able to solve and how many solving combinations were successful.

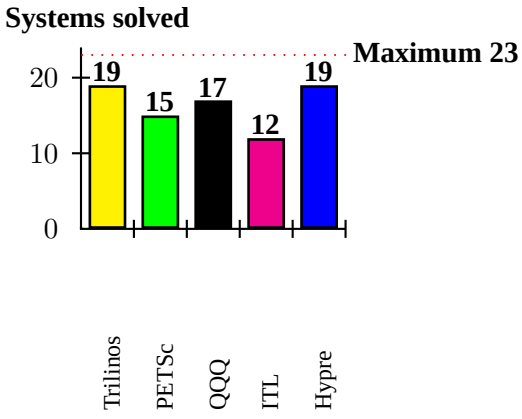

Figure 5.28: Number of solvable equation systems per package: Trilinos and **Hypre** were both able to solve 19 of the 23 given systems. These are followed by **QQQ**, which solved 17 different systems. **PETSc** and **ILU** solved the least amount of equation systems with 15 and 12 different solved systems, respectively.

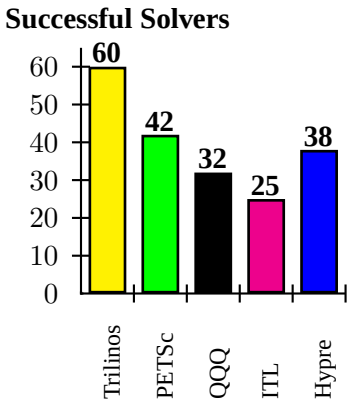

Figure 5.29: Weighted Score of the Packages: Each solving combination that was tested yields one weighted point if it has reached the accuracy of $1 \cdot 10^{-12}$. The five Laplace matrices have not been taken into account for this diagram because they would have distorted the result. **Trilinos** has the most successful solving combinations and also provides the most different solving combinations. **PETSc** has the same amount of combinations but is not that successful because it often only reaches an accuracy between $1 \cdot 10^{-11}$ and $1 \cdot 10^{-12}$. **ITL** has the second most solving combinations but the least successful ones. At the moment **ITL** only provides solvers which are restricted to a specific kind of matrices. With a **GMRES** algorithm there would have been more successful solving runs.

Chapter 6

Automatic Solver Control Interface

Figure 6.1 shows the components of the automatic solver control interface that was developed which handles the solving of a linear equation system. It contains elements that analyze the matrix, components which can decide on the different solving options, and a post solving handling of the solution which checks whether the system has been solved. The user needs to insert the matrix and the right-hand side vector into the solver. When the solving process is started, the matrix will be analyzed and then passed to the solver package that will solve it. The right-hand side vector is analyzed to decide which residual should be used as a break condition. When the solving process is finished, the solution will be checked to see if it has reached a defined accuracy (three choices are available for the user). If not, the error code delivered will be handled. This may result in a new solving run with different solving options.

6.1 Number Analyzer

The number analyzer fulfills the following tasks:

- Calculating the diagonal dominance percentage
- Determining the number of zeros in the main diagonal
- Calculation of the diagonal average and the nondiagonal average

The main task of the number analyzer is to determine the number of zeros on the main diagonal, because they cause errors when using a preconditioner without zero pivot handling. Another useful number is the diagonal dominance, which indicates how many percent of the matrix elements are within the main diagonal. Other useful numbers are the diagonal average and the nondiagonal average. If a matrix has big numbers on the main diagonal and small numbers off the main diagonal, the **Jacobi** preconditioner would make the main diagonal one and the nondiagonal values smaller. Considering the Gershgorin circle theorem (see Section 2.1.5) and that the condition is defined by the eigenvalues of the matrix $\mathbf{A} \cdot \mathbf{A}^T$ (see Chapter 2) the matrix obtained would have eigenvalues close to 1 and therefore a small condition number.

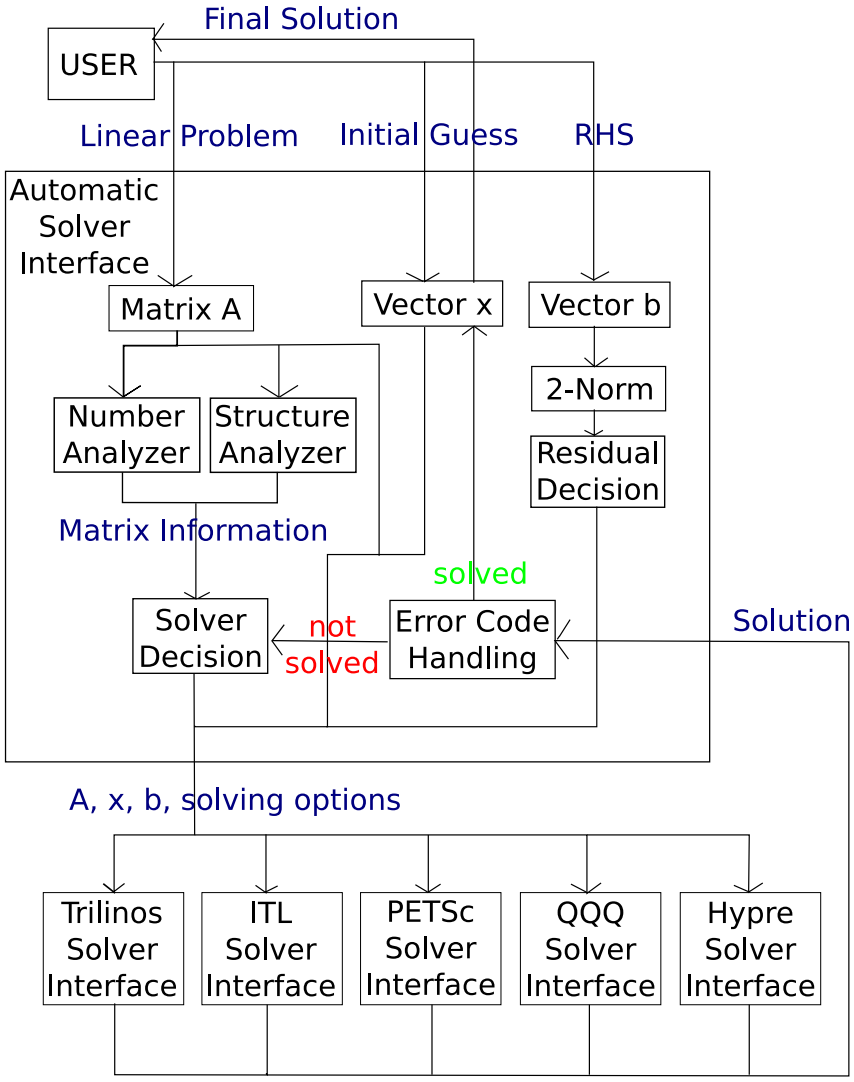

Figure 6.1: This figure illustrates how the automatic solver control interface works. The user states a linear equation system by inserting the matrix and the right-hand side into the solver interface (the initial guess is not required). The matrix is analyzed by numbers and structure. The 2-norm of the **b** vector is used for the residual decision. The Solver Decision module decides which solver package and options are used. Afterwards all necessary information is passed to the chosen solver package. After the solving process, the Error Code Handling module checks the error the solver package returned and also guarantees that the solution has the desired accuracy. In case the system was not solved, other approaches are used.

6.2 Structure Analyzer

The structure analyzer checks to see if the matrix is symmetric and positive definite. If the matrix does not fulfill one of these characterizations, several solver flags are disabled:

- not symmetric: **CG** and **IC** deactivated
- not positive definite: **CG** and **BICGstab** deactivated

6.3 Residual Decision

The norm of the **b**-vector is used to make a decision if the absolute or relative residual should be used. If the norm is greater than or equal to one, the absolute residual is used; if the norm is smaller, the relative residual is used.

6.4 Solver Decision

The data which is obtained by the analyzers can be used to decide which solver package, solver, and preconditioner may be used. This tool is used to eliminate improper solving options. This is implemented using Boolean variables.

The **CG** and **IC** algorithms require a symmetric matrix as input. If the matrix is not symmetric these methods may fail and lead to a crash of the program. So if the matrix is not symmetric, these methods are not used.

CG also requires the matrix to be positive definite as well as **BICGstab**. If it is not, the methods are not used.

The **Jacobi** preconditioner can be used when the percentage of the diagonal dominance is larger than a defined threshold. It may also be used if the average diagonal value is larger than 1, while the average nondiagonal value is smaller than 1:

```
1 if((diagonaldominance>JACOBI_LIMIT)||((diagonalaverage>1.0)&&(nondiagonalaverage<1.0)))
2     jacobi_flag=true;
3 else jacobi_flag=false;
```

If there are zeros in the main diagonal, the **Trilinos ILU** preconditioner and the preconditioners of **ITL** have serious problems. These packages should not be used in this case. **Hypre**, **PETSc** and **QQQ** offer a better handling for this case. Also the **Jacobi** preconditioner should be turned off because it is affected as well.

To decide whether the current solver configuration is valid, a function is needed that returns **false** if not valid. There are two reasons the configuration can be invalid. The first reason can be that some of the solving options have been disabled by the decision module:

```
1 if((preconditioner_id==IC)&&(symmetric_flag==false))   return false;
2 if((preconditioner_id==JACOBI)&&(jacobi_flag==false)) return false;
3 if((solver_id==CG)&&((symmetric_flag==false)||(positive_definite_flag==false)))
4     return false;
5 if((solver_id==BICGSTAB)&&(positive_definite_flag==false)) return false;
```

Second, the solving options can be invalid because the defined package does not have the required solvers or preconditioners.

```
switch(solver_package_id){
        case TRILINOS: if(trilinos_flag==true) return true;
           else return false;
           break;
        case HYPRE: if((preconditioner_id!=ILU)||(hypre_flag==false)) return false;
           else return true;
           break;
        case ITL: if((solver_id==GMRES)||(itl_flag==false)) return false;
           else return true;
           break;
        case PETSC: return true;
           break;
        case QQQ: if((solver_id==CG)||(preconditioner_id!=ILU))  return false;
           else return true;
}
```

If the solver configuration is invalid it needs to be changed which will be discussed in the next section.

6.5 Error Code Handling

The error code that is returned by the solver package used can be used to make further decisions. Also, even if the error code states that the problem was solved, it is possible that a numerical problem occurred and that the true residual is far off the desired accuracy.

The errors which can occur are:

- loss of precision
- maximum iterations exceeded
- different types of numerical breakdowns

The possible options for handling the errors are:

- change solver package
- change preconditioner
- change solver
- solve again with new accuracy
- solve again with more iterations

The first three options can be merged into one single function which cycles through all solver packages, preconditioners and solvers. The first thing changed is the solver package, so if a solver type seems good for a problem, it will be tested with all solver packages, starting with the fastest one. If it fails, the next package will be used. The packages will be cycled in the following order (starting with the fastest): **Trilinos**, **ITL**, **Hypre**, **PETSc** and **QQQ**. If the **QQQ** package is reached, the solver package is reset to **Trilinos** and the preconditioner is increased (preconditioner cycle order: **Jacobi**, **IC** and **ILU**).

```
if(solver_package_id==QQQ){//check if last package reached
   if(preconditioner_id==ILU){//check if last preconditioner reached
      ...
      }
   } else {
      solver_package_id=TRILINOS;//reset solver package
      preconditioner_id++;//use next preconditioner
   }
} else
   solver_package_id++; //use next solver_package
```

If the preconditioners are cycled through as well, the solver will be increased (solver cycle order: **CG**, **BICGstab** and **GMRES**), until all possible options have been used.

```
if(solver_id==GMRES)
         return false;//no more option available
      else {
         solver_package_id=TRILINOS;//reset solver_package
         preconditioner_id=JACOBI;//reset preconditioner
         solver_id++;
```

Every time the solving options are changed, a check is required to see if they are valid .

6.5.1 Loss of Precision Handling

The **Trilinos** package is the only package that explicitly states a loss of precision error. For the other packages, this error is handled implicitly if the solver package returned that everything is all right but the residual check fails.

The loss of precision error can be approached by resolving the equation system with a better accuracy and with the solution of the last run as the initial guess. Important variables are:

```
int solve_counter;//counts the solve numbers
double accuracy;// desired accuracy
double solve_accuracy;//accuracy handed to the solver, can vary to the above number
double residual;//residual which is calculated after the solving process
double old_residual;//residual of the last run before current run
```

The implementation in the code is as follows:

```
if((solve_counter<SOLVE_LOOP_MAX)&&(old_residual!=residual)) {
   solve_counter++;
   old_residual=residual;//residual is the same as the last run
   solve_accuracy=solve_accuracy/10.0;
   if(residual >1.0)
      reset_initialguess();
}
```

The procedure is repeated until a defined number of solving processes is reached or the residual of the last solution is equal to the new residual. Also the solution will only be used again if it is smaller than 1. The **else**-tree resets the solve_counter , the solve_accuracy and the old_residual . It also changes the solving options:

```
else {
    solve_counter=0;
    solve_accuracy=accuracy;
    decision=increase_solver();
    old_residual=0.0;
}
```

6.5.2 Maximum Iterations Exceeded

If the maximum number of iterations are exceeded, then depending on the residual of the solution it may be helpful to restart the solving process with an increased number of iterations. The restart will only be used if the norm of the residual from the last run is smaller than 1. In the other case, the solver cycling will continue. There are two different resolving branches depending on whether the solving package is **PETSc** or not. **PETSc** is the slowest solver, so only one solving run is used here. The other packages use two solving runs at most. The iterations are multiplied by 10 every time, so more than two solving runs do not make sense because the time to divergence would be too long.

```
if((solver_package_id==PETSC)&&(residual<1.0)&&(solve_counter<SOLVE_IT_MAX_PETSC)){
    iterations*=10;
    if(iterations>MAX_ITERATIONS)
    iterations=MAX_ITERATIONS;
    solve_counter++;
}
```

The **else**-branch resets the variables which were used.

6.5.3 Numerical Problems

For the numerical problems, the only possible solution is to change the solver options, because the numerical breakdown is nested within the algorithm used and would likely occur again if the same options were used again.

```
solve_counter=0;//resets if it was used in the current run
decision=increase_solver(); //use next solving option
if(residual>1.0) //keep residual if it is smaller than 1
    reset_initialguess();
```

6.6 Tests

There are three different accuracy limits the user can choose from: $1 \cdot 10^{-8}$, $1 \cdot 10^{-10}$ and $1 \cdot 10^{-12}$. It is not guaranteed that these limits are reached. In case of divergence, the solution with the lowest residual reached can be accessed if desired.

6.6.1 Break tolerance $1 \cdot 10^{-8}$

Tests with the automatic solver control showed that for all matrices except the matrix E20r5000 and the matrix Sherman2 a solution with an accuracy of $1 \cdot 10^{-8}$ could be computed. Which solvers have been used and how many times a new solving run was necessary until the accuracy was reached are shown in the Figures 6.3 and 6.4 (Figure 6.2 explains how the diagrams can be read).

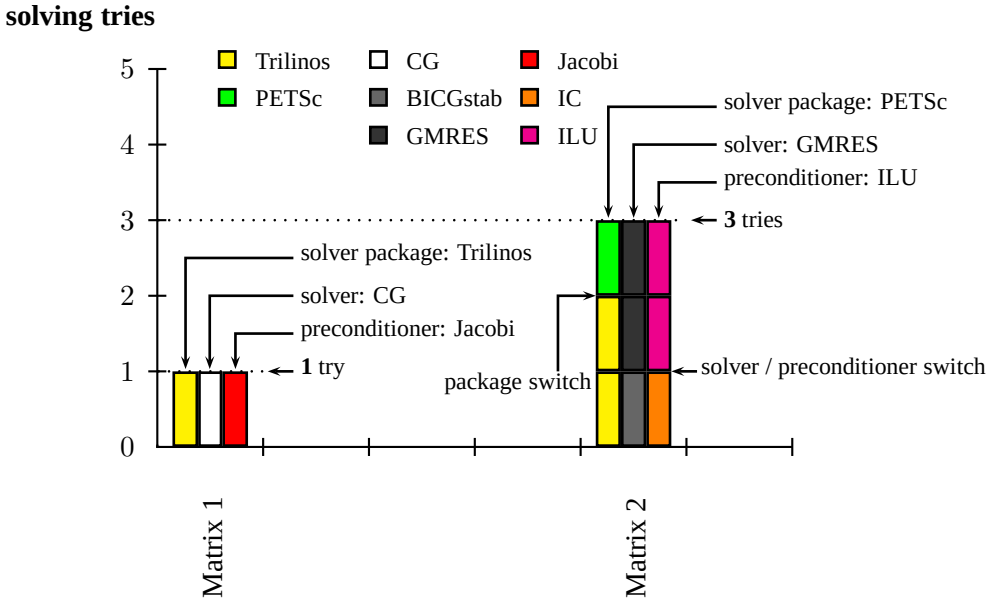

Figure 6.2: Diagram explanation manual: The diagram of the Matrix 1 indicates that its equation system was solved with one try and the solving options Trilinos, CG and Jacobi. The second system was solved in 3 tries. The first try was done with Trilinos, BICGstab and IC. The second try with Trilinos, GMRES and ILU. The third try with PETSc, GMRES and ILU. These diagrams show the solving options that were used by the automatic solver control interface and also the number of solving tries used to reach an accurate solution (**NOTE**: This is only an explanation diagram which may not indicate the real switch options the automatic solver control uses).

A positive example of handling a loss of precision can be seen in Figure 6.5. The handling does not work every time, which can be seen in the matrix E20r5000 which reached the accuracy $3.35 \cdot 10^{-8}$ (Figure 6.6) and the matrix Sherman2 that had 1.1^{-7} as best residual (Figure 6.7).

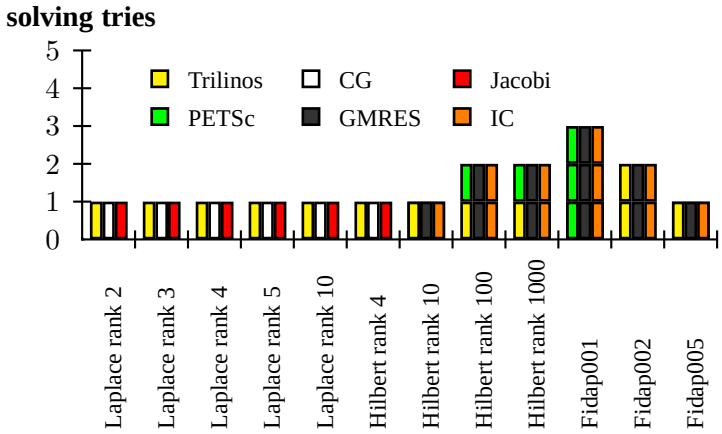

Figure 6.3: Test results for accuracy $1 \cdot 10^{-8}$: The Laplace matrices are all symmetric and positive definite, therefore the **CG** solver is used by the program. The **Jacobi** preconditioner is chosen because the diagonal dominance is greater than 20%. The **Trilinos** package is the first one used and successfully solves these systems. The same solving options are used for the Hilbert matrix rank4. The linear equations systems of the three other Hilbert matrices are solved with the **IC** preconditioner and the **GMRES** solver because they are not positive definite. For the rank 100 and 1000 matrices, **Trilinos** is not able to solve the systems so the package is switched to **PETSc**. The Fidap001 matrix has zeros in the main diagonal, so **Trilinos** is skipped and instead **PETSc** is used as the first package. The matrix is symmetric but not positive definite, therefore **GMRES** and **IC** are used. Its system is solved on the third try. The matrix Fidap002 has no zeros on the main diagonal, so **Trilinos** was used but a loss of precision occurred so the system was only solvable on the second try. The equation system of the matrix Fidap005 was solvable in one try.

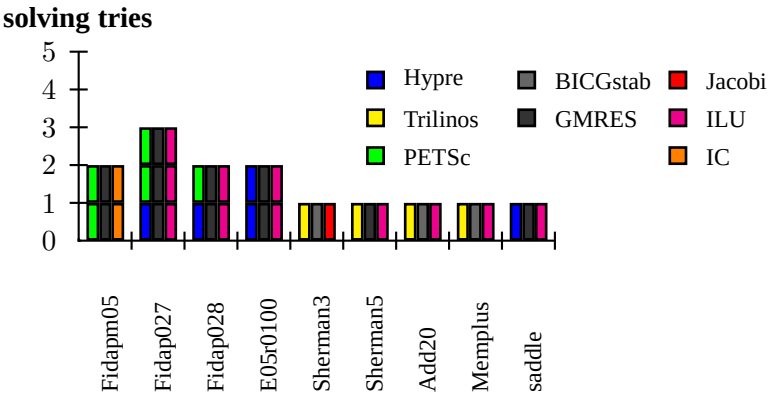

Figure 6.4: Test results for accuracy $1 \cdot 10^{-8}$: The matrix Fidapm05 has zeros in the main diagonal, so the first package is not **Trilinos**. The matrix is symmetric but not positive definite so the options chosen are **GMRES** and **IC** from the **PETSc** package. A loss of precision makes a second run necessary. The matrices Fidap027 and Fidap028 are neither symmetric nor positive definite and have zeros in the main diagonal. The options for the first run were **GMRES** and **ILU** from the **Hypre** package, which did not result in a solution. A second run (and even a third run for Fidap027) with the **PETSc** package was required. The matrix E05r0100 has zeros in the main diagonal and is neither symmetric or positive definite. The **Hypre** package solved the system in two runs using **GMRES** and **ILU**. The rest of the equation systems could be solved with the first decision of the automatic solver control module.

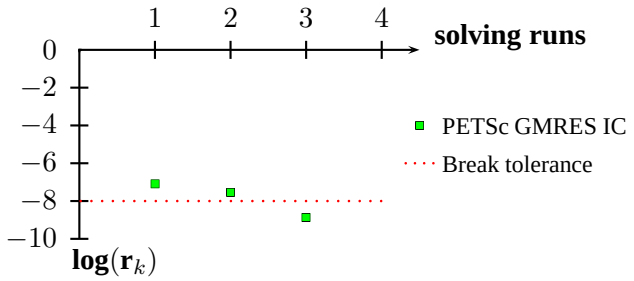

Figure 6.5: Results for the matrix Fidap001: The true residual of the matrix is not beyond the given limit for convergence after the first solving run, so the solving process is restarted. The last solution is used as the new initial guess, and the accuracy is decreased by a factor of 10. After two restarts the solution fulfills the given limit.

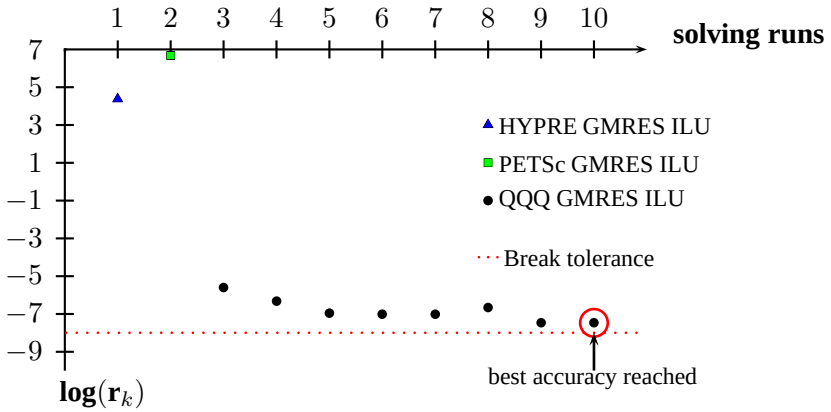

Figure 6.6: Results for Matrix E20r5000: The equation system containing the matrix E20r5000 was not solvable to the accuracy $1 \cdot 10^{-8}$. **Hypre** and **PETSc** failed totally by returning a residual number much bigger than 1. With the **QQQ GMRES** solver with **ILU** preconditioner it was possible by restarting the solving process repeatedly (while the accuracy was decreased at every restart) to reach a solution close to $1 \cdot 10^{-8}$ (**NOTE:** Normally the restart process also uses the last solution, if it is already smaller than 1. This option is not available because **QQQ** resets the solution every time).

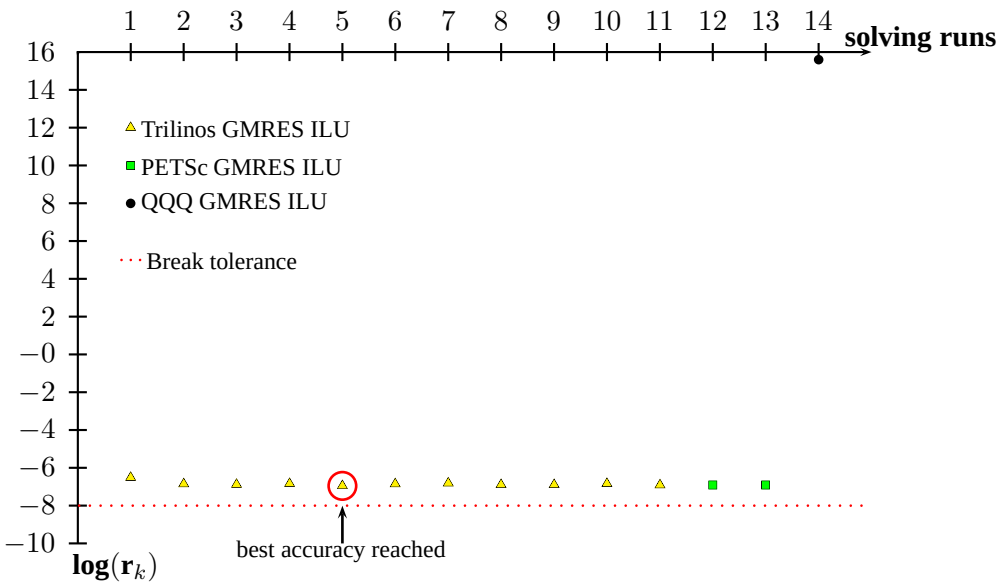

Figure 6.7: Results for the matrix Sherman2: The solution with the best accuracy is reached after 5 solving tries. Then the residual norm of the solution could not be decreased further. The **QQQ** solver totally failed here by calculating a solution with a residual norm around $1 \cdot 10^{16}$.

6.6.2 Break tolerance $1 \cdot 10^{-10}$

The solvable matrices are shown in the diagrams 6.8 and 6.9.

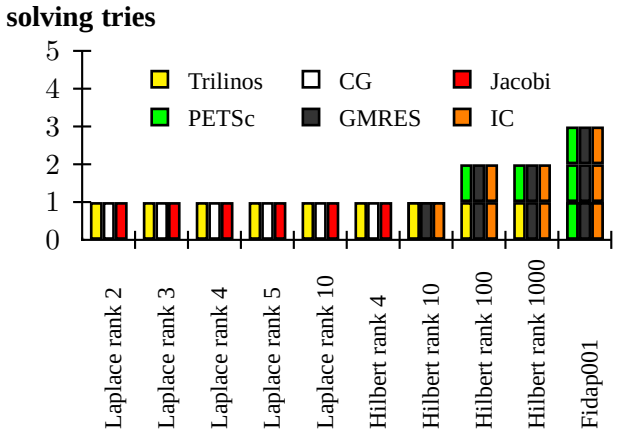

Figure 6.8: Test results for accuracy $1 \cdot 10^{-10}$: This diagram does not show much difference from the diagram for the accuracy $1 \cdot 10{-}8$. One big difference is that the equation systems Fidap002 and Fidap005 were not solvable anymore, so they are missing here. All the other matrices had the same solving options and tries as for the accuracy $1 \cdot 10{-}8$.

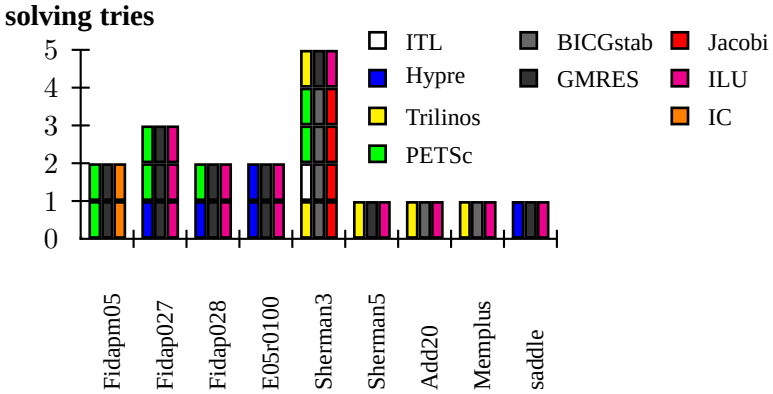

Figure 6.9: Test results for accuracy $1 \cdot 10^{-10}$: All matrices behave the same way they did with accuracy $1 \cdot 10^{-8}$ except the matrix Sherman2 which now needs 5 tries, unlike to one try before.

The systems which were not able to be solved are Fidap002 (Figure 6.10) which was only solvable to an accuracy of 3.54^{-9} and Fidap005 (Figure 6.11) which only reached an accuracy of 2.25^{-10}. Obviously, the linear equation systems E20r5000 and Sherman2 which have not been solvable to an accuracy of $1 \cdot 10^{-8}$ could not be solved as well.

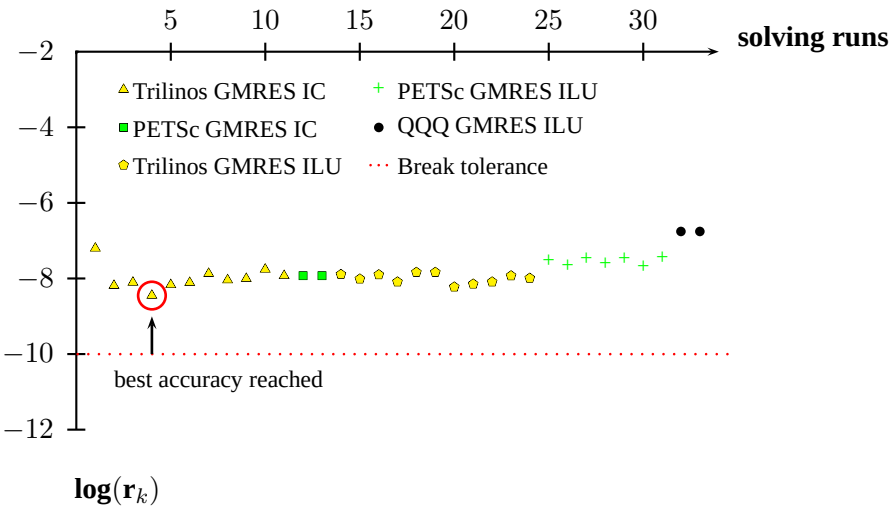

Figure 6.10: Results for the matrix Fidap002: The matrix Fidap002 already had a loss of precision with the accuracy $1 \cdot 10^{-8}$. There the desired accuracy was reachable by restarting the solving process several times. Now the best solution is reached after 4 solving runs and then no more improvement is made. The solution does not reach the convergence limit of $1 \cdot 10^{-10}$.

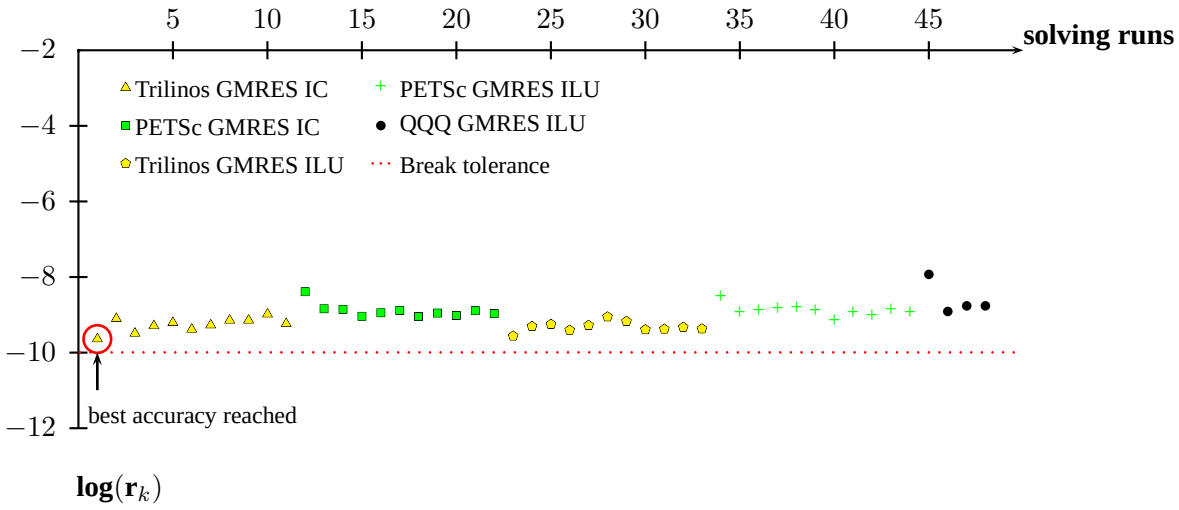

Figure 6.11: Results for the matrix Fidap005: The matrix Fidap005 did not encounter any problems with the accuracy $1 \cdot 10^{-8}$. Only one solving try was necessary to get to the desired solution. Now the best solution is also calculated at the first try, but it does not fulfill the convergence tolerance of $1 \cdot 10^{-10}$. No switch of the solving options helps to get a more accurate solution.

6.6.3 Break tolerance $1 \cdot 10^{-12}$

With the higher accuracy, more equation system appeared that were not solvable to the desired tolerance. The systems which could be solved are shown in the Figures 6.12 and 6.13.

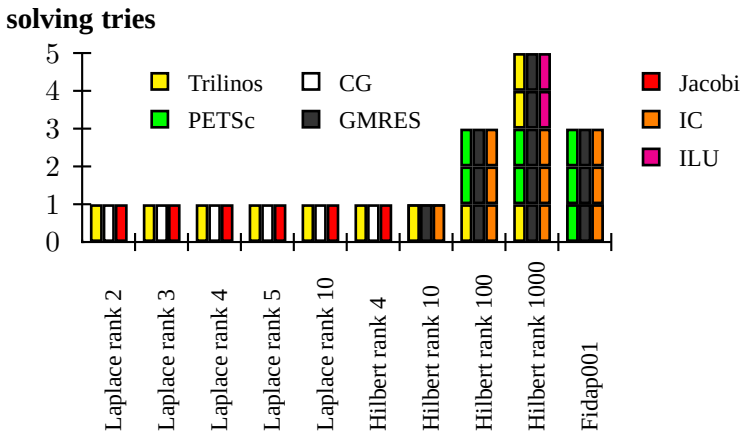

Figure 6.12: Test results for accuracy $1 \cdot 10^{-12}$: The matrix Hilbert rank 100 now needs an additional solving try to compute the solution with an accuracy of $1 \cdot 10^{-12}$. The matrix Hilbert rank 1000 even needs five tries now and cycles not only the solving packages, but also the preconditioner is changed from **IC** to **ILU**.

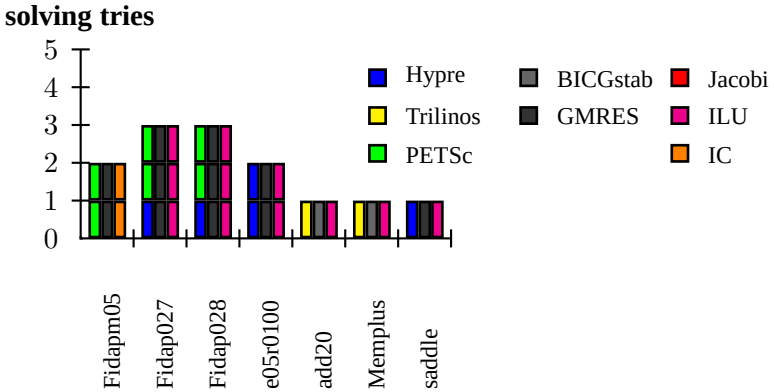

Figure 6.13: Test results for accuracy $1 \cdot 10^{-12}$: The equation systems Sherman3 and Sherman5 are missing here because they are not solvable anymore. The matrix Fidap028 now needs one more solving try than before.

The Sherman3 matrix had 8.42^{-11} as best residual (Figure 6.14). The matrix Sherman5 reached a residual of 2.27^{-11} (Figure 6.15).

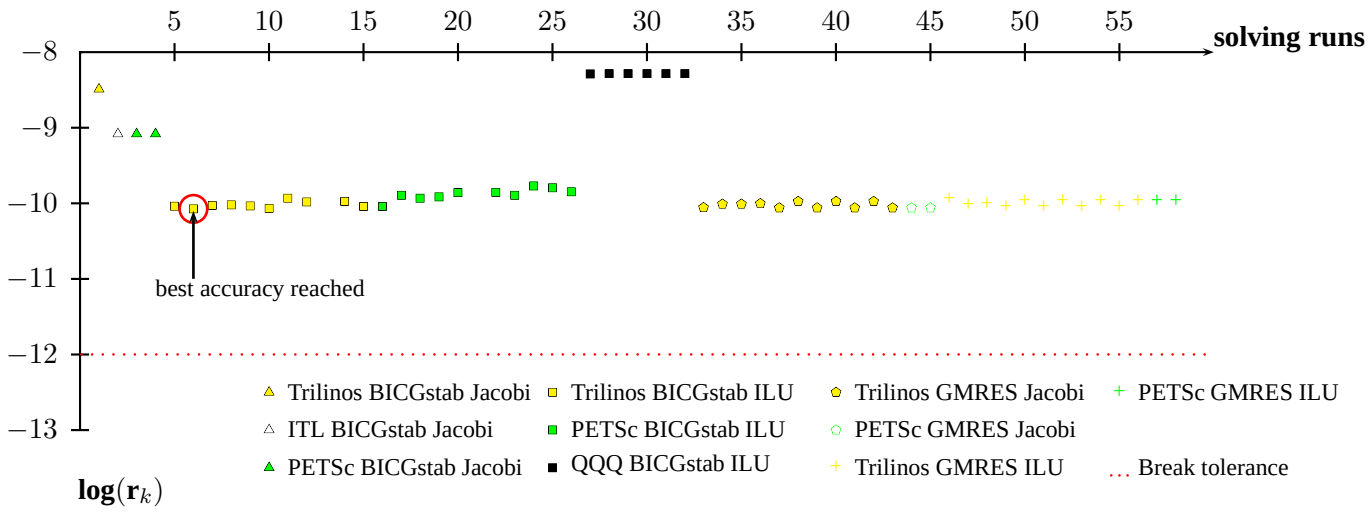

Figure 6.14: Results for the matrix Sherman3: The best solution is reached after six solving tries and three switches of the solving options. Further restarting of the solving process and even more switching of the options does not improve the solution anymore. (**NOTE:** The **QQQ GMRES** solver with **ILU** preconditioner was the last solving try but is not shown in the diagram because it had a residual around $1 \cdot 10^{10}$).

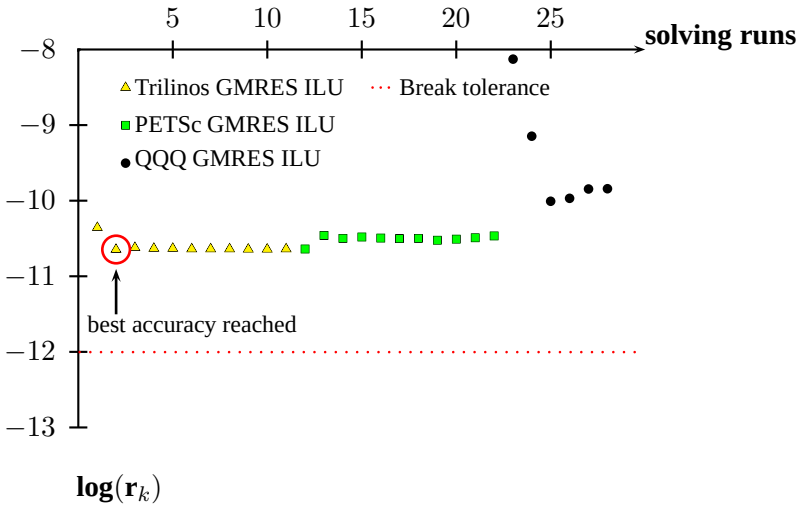

Figure 6.15: Results for the matrix Sherman5: The best solution was reached after the second solving try. More solving tries and switching of the options did not increase the accuracy of the solution anymore nor did they help to reach convergence.

6.6.4 Conclusion of the Tests

The test with the three different accuracy levels showed that decreasing the accuracy margin of the convergence causes more solvers to have problems with computing accurate solutions for the matrices. For the low accuracy $1 \cdot 10^{-8}$ most matrices have been solvable with the first chosen options; only a package switch sometimes had to be made. When the accuracy tolerance was decreased, more solver and preconditioner switches occurred and also more matrices had loss of precision errors.

Regarding the loss of precision error it was not always possible to improve the solution further by restarting the solving process. For the matrices it worked with a lower accuracy a higher accuracy resulted in an unsolvable loss of precision error. These matrices are close to the numerical limits and therefore it is very hard to improve the accuracy further.

6.7 Usage of the Automatic Solver Control

At first the automatic solver control interface needs to be initialized with the dimension of the equation system:

```
1 asi::automatic_solver_control_interface  si(dimension);
```

Then the matrix and the right-hand side need to be filled:

```
1 {
2    ... //loop for passing matrix to solver interface
3    si.insert_value(row,column,value);
4 }
5
6 {
7    ...//loop for passing b vector to solver
8    si.init_rhs(i,value);
9 }
```

After that the diagnostic tool can be activated (si. activate_diagnostic ()). Before solving the desired accuracy may be chosen (e.g., low accuracy si.set_low_accuracy ()) and the printlevel optionally can be set (si. set_printlevel (1)). Because some solver packages have output which cannot be suppressed, there is some output even if no output is chosen. All warnings provided by these packages are not valid because they are handled anyway.

Then the solving process can be started with si.solve() and will return a **true** if it succeeds. If a **false** is returned the user can check the value of the best residual reached. If he tolerates it he may access the best solution with:

```
1 si.get_best_x_value(int row);//returns the double value with the index row
```

Chapter 7

Conclusion

The development of an automatic solver control has been presented. Three different indirect solver algorithms (**CG**, **BICGstab**, and **GMRES**) have been tested. Five different packages (**Trilinos**, **Hypre**, **PETSc**, **QQQ** and **ITL**) which provide these algorithms have been compared to each other to find out the differences in code implementation (Chapters 3 and 5). The preconditioners **Jacobi**, **ILU** and **IC** have been introduced to improve the solver process of some linear equation systems.

In this work several matrix properties have been investigated to find out which solvers fit a particular type of matrix best. The **CG** solver and the **IC** preconditioner require a matrix that is symmetric and positive definite. The **BICGstab** solver also requires a positive definite matrix. Another important matrix property is the number of zeros in the main diagonal. The strategies of the different solver packages for zero pivot elements are important to success in solving certain matrices. The **PETSc** and **QQQ** packages both have good strategies and therefore had the least problems with matrices containing zeros on the main diagonal. Last but not least, the relation between the number of diagonal and off-diagonal elements is a decision criteria for the **Jacobi** preconditioner.

These matrix properties are checked by a matrix analyzer which has been implemented. The performance of this module was reasonable and helped the automatic solver control to decide which solving options fit best for a matrix.

An error that often occurred during the testing of the matrices is a loss of precision of the residual. This can lead to inaccurate solutions. Chapter 6 introduced a strategy to handle this problem. The equation system is solved repeatedly to improve the accuracy of the residual. This led to reasonable results for some of the matrices. Chapter 6 also showed the performance of the automatic solver control with three different accuracy limits. For the accuracy $1 \cdot 10^{-8}$ the automatic solver control was able to solve all given systems except two. A few of the matrices needed more than one solving run to reach the desired accuracy. Switching of the solver package helped in some cases. For other matrices the loss of precision handling was successful. The automatic solver control also delivered reasonable results for the other accuracy limits.

There are more solvers and preconditioners available on the market which could be added to the automatic solver control for further improvement.

Appendix A

Test results

This section contains the test results of all the matrices tested in Chapter 5. The tables contain the solver package, the solver type, the preconditioner, how many iterations the solving process took, the status, the solving time and the absolute and relative residual calculated after the solving process. The error codes returned by the packages are represented by the following **enum** values:

```
1 enum solvererror{
2   SOLVED=0, MAX_ITERATIONS_EXCEEDED, LOSS_OF_PRECISION,
3   NUMERICAL_BREAKDOWN, ILL_CONDITIONED, UNKNOWN_ERROR, INDEFINITE_PC,
4   DROP_TOLERANCE, INDEFINITE_MAT
5 };
```

The solver combinations tested were:

- **Trilinos** package

  Solvers: **CG**, **BICGstab** and **GMRES**; Preconditioners: none, **Jacobi**, **ILU** and **IC**

- **PETSc** package

  Solvers: **CG**, **BICGstab** and **GMRES**; Preconditioners: none, **Jacobi**, **ILU** and **IC**

- **ITL** package

  Solvers: **CG** and **BICGstab**; Preconditioners: none, **Jacobi**, **ILU** and **IC**

- **Hypre** package

  Solvers: **CG**, **BICGstab** and **GMRES**; Preconditioners: none and **ILU**

- **QQQ** package:

  Solvers: **BICGstab** and **GMRES**; Preconditioners: none and **ILU**

The **IC** preconditioner was only used with symmetric matrices. When any of the solver combinations mentioned is missing from a table, this indicates that the solver was not able to reach a defined program end (e.g., an exception was thrown).

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	2	0	0.000264s	0	0
Hypre	CG	NONE	1	0	0.000188s	0	0
Trilinos	CG	NONE	1	0	0.00025s	0	0
PETSc	CG	NONE	1	0	0.000603s	0	0
ITL	BICGstab	NONE	1	0	0.000255s	0	0
Hypre	BICGstab	NONE	0	3	0.000157s	nan	nan
Trilinos	BICGstab	NONE	1	0	0.000232s	0	0
QQQ	BICGstab	NONE	1	0	0.000208s	3.14018e-16	2.22045e-16
PETSc	BICGstab	NONE	1	0	0.000594s	0	0
Hypre	GMRES	NONE	1	0	0.006222s	3.14018e-16	2.22045e-16
Trilinos	GMRES	NONE	1	0	0.000269s	3.14018e-16	2.22045e-16
QQQ	GMRES	NONE	1	0	0.004661s	3.14018e-16	2.22045e-16
PETSc	GMRES	NONE	1	0	0.01812s	3.14018e-16	2.22045e-16
ITL	CG	Jacobi	2	0	0.000273s	0	0
Trilinos	CG	Jacobi	1	0	0.000263s	0	0
PETSc	CG	Jacobi	1	0	0.000634s	0	0
ITL	BICGstab	Jacobi	1	0	0.000269s	0	0
Trilinos	BICGstab	Jacobi	1	0	0.000234s	0	0
PETSc	BICGstab	Jacobi	1	0	0.000614s	0	0
Trilinos	GMRES	Jacobi	1	0	0.000253s	3.14018e-16	2.22045e-16
PETSc	GMRES	Jacobi	1	0	0.017948s	3.14018e-16	2.22045e-16
ITL	CG	ILU	2	0	0.000298s	0	0
Hypre	CG	ILU	1	0	0.000194s	0	0
Trilinos	CG	ILU	1	0	0.000482s	0	0
PETSc	CG	ILU	1	0	0.001542s	0	0
ITL	BICGstab	ILU	1	0	0.000286s	0	0
Hypre	BICGstab	ILU	0	3	0.000172s	nan	nan
Trilinos	BICGstab	ILU	1	0	0.000411s	0	0
QQQ	BICGstab	ILU	1	0	0.00019s	3.14018e-16	2.22045e-16
PETSc	BICGstab	ILU	1	0	0.001494s	0	0
Hypre	GMRES	ILU	1	0	0.004202s	3.14018e-16	2.22045e-16
Trilinos	GMRES	ILU	1	0	0.000498s	3.14018e-16	2.22045e-16
QQQ	GMRES	ILU	1	0	0.004665s	3.14018e-16	2.22045e-16
PETSc	GMRES	ILU	1	0	0.019647s	3.14018e-16	2.22045e-16
ITL	CG	IC	2	0	0.000376s	4.96507e-16	3.51083e-16
Trilinos	CG	IC	1	0	0.000486s	0	0
PETSc	CG	IC	1	0	0.001646s	0	0
ITL	BICGstab	IC	1	0	0.000286s	2.22045e-16	1.57009e-16
Trilinos	BICGstab	IC	1	0	0.000379s	0	0
PETSc	BICGstab	IC	1	0	0.001264s	0	0
Trilinos	GMRES	IC	1	0	0.00046s	3.14018e-16	2.22045e-16
PETSc	GMRES	IC	1	0	0.018328s	3.14018e-16	2.22045e-16

Table A.1: Laplace Rank 2 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	3	0	8.9e-05s	0	0
Hypre	CG	NONE	2	0	8.9e-05s	0	0
Trilinos	CG	NONE	2	0	0.000113s	0	0
PETSc	CG	NONE	2	0	0.000328s	0	0
ITL	BICGstab	NONE	2	0	9.5e-05s	5.43896e-16	3.84593e-16
Hypre	BICGstab	NONE	2	0	8.5e-05s	5.43896e-16	3.84593e-16
Trilinos	BICGstab	NONE	2	0	0.000104s	5.43896e-16	3.84593e-16
QQQ	BICGstab	NONE	2	0	0.000107s	0	0
PETSc	BICGstab	NONE	2	0	0.000332s	5.43896e-16	3.84593e-16
Hypre	GMRES	NONE	2	0	0.005149s	3.14018e-16	2.22045e-16
Trilinos	GMRES	NONE	2	0	0.001377s	3.14018e-16	2.22045e-16
QQQ	GMRES	NONE	2	0	0.004485s	6.28037e-16	4.44089e-16
PETSc	GMRES	NONE	2	0	0.010632s	4.44089e-16	3.14018e-16
ITL	CG	Jacobi	3	0	0.000101s	0	0
Trilinos	CG	Jacobi	2	0	0.000126s	0	0
PETSc	CG	Jacobi	2	0	0.000354s	0	0
ITL	BICGstab	Jacobi	2	0	9.8e-05s	5.43896e-16	3.84593e-16
Trilinos	BICGstab	Jacobi	2	0	0.000109s	5.43896e-16	3.84593e-16
PETSc	BICGstab	Jacobi	2	0	0.000337s	5.43896e-16	3.84593e-16
Trilinos	GMRES	Jacobi	2	0	0.000121s	3.14018e-16	2.22045e-16
PETSc	GMRES	Jacobi	2	0	0.010194s	4.44089e-16	3.14018e-16
ITL	CG	ILU	2	0	0.000126s	2.48253e-16	1.75542e-16
Hypre	CG	ILU	1	0	9.4e-05s	2.48253e-16	1.75542e-16
Trilinos	CG	ILU	1	0	0.000216s	2.48253e-16	1.75542e-16
PETSc	CG	ILU	1	0	0.000662s	2.48253e-16	1.75542e-16
ITL	BICGstab	ILU	1	0	0.000108s	2.48253e-16	1.75542e-16
Hypre	BICGstab	ILU	0	3	8.4e-05s	0	0
Trilinos	BICGstab	ILU	1	0	0.000155s	0	0
QQQ	BICGstab	ILU	1	0	8.4e-05s	7.36439e-16	5.20741e-16
PETSc	BICGstab	ILU	1	0	0.000634s	1.57009e-16	1.11022e-16
Hypre	GMRES	ILU	1	0	0.0032s	3.14018e-16	2.22045e-16
Trilinos	GMRES	ILU	1	0	0.000239s	3.14018e-16	2.22045e-16
QQQ	GMRES	ILU	1	0	0.00446s	1.57009e-16	1.11022e-16
PETSc	GMRES	ILU	1	0	0.011198s	3.14018e-16	2.22045e-16
ITL	CG	IC	2	0	0.000109s	0	0
Trilinos	CG	IC	1	0	0.000253s	2.48253e-16	1.75542e-16
PETSc	CG	IC	1	0	0.000883s	2.48253e-16	1.75542e-16
ITL	BICGstab	IC	1	0	9.8e-05s	0	0
Trilinos	BICGstab	IC	1	0	0.000153s	0	0
PETSc	BICGstab	IC	1	0	0.000502s	1.57009e-16	1.11022e-16
Trilinos	GMRES	IC	1	0	0.00023s	3.14018e-16	2.22045e-16
PETSc	GMRES	IC	1	0	0.010533s	3.14018e-16	2.22045e-16

Table A.2: Laplace rank 3 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	3	0	0.000284s	0	0
Hypre	CG	NONE	2	0	0.000208s	0	0
Trilinos	CG	NONE	2	0	0.000266s	0	0
PETSc	CG	NONE	2	0	0.00063s	0	0
ITL	BICGstab	NONE	2	0	0.000296s	0	0
Hypre	BICGstab	NONE	0	3	0.00018s	nan	nan
Trilinos	BICGstab	NONE	2	0	0.00025s	0	0
QQQ	BICGstab	NONE	2	0	0.000215s	6.28037e-16	4.44089e-16
PETSc	BICGstab	NONE	2	0	0.000612s	0	0
Hypre	GMRES	NONE	2	0	0.006287s	3.14018e-16	2.22045e-16
Trilinos	GMRES	NONE	2	0	0.001544s	3.14018e-16	2.22045e-16
QQQ	GMRES	NONE	2	0	0.004627s	3.14018e-16	2.22045e-16
PETSc	GMRES	NONE	2	0	0.018159s	3.14018e-16	2.22045e-16
ITL	CG	Jacobi	3	0	0.000294s	0	0
Trilinos	CG	Jacobi	2	0	0.000278s	0	0
PETSc	CG	Jacobi	2	0	0.000668s	0	0
ITL	BICGstab	Jacobi	2	0	0.000359s	0	0
Trilinos	BICGstab	Jacobi	2	0	0.000262s	0	0
PETSc	BICGstab	Jacobi	2	0	0.000645s	0	0
Trilinos	GMRES	Jacobi	2	0	0.000277s	3.14018e-16	2.22045e-16
PETSc	GMRES	Jacobi	2	0	0.017549s	3.14018e-16	2.22045e-16
ITL	CG	ILU	2	0	0.000306s	1.57009e-16	1.11022e-16
Hypre	CG	ILU	1	0	0.000198s	1.57009e-16	1.11022e-16
Trilinos	CG	ILU	1	0	0.00049s	1.57009e-16	1.11022e-16
PETSc	CG	ILU	1	0	0.001502s	3.33067e-16	2.35514e-16
ITL	BICGstab	ILU	1	0	0.000291s	1.57009e-16	1.11022e-16
Hypre	BICGstab	ILU	1	0	0.000218s	0	0
Trilinos	BICGstab	ILU	1	0	0.000429s	3.14018e-16	2.22045e-16
QQQ	BICGstab	ILU	1	0	0.000196s	4.96507e-16	3.51083e-16
PETSc	BICGstab	ILU	1	0	0.001455s	1.57009e-16	1.11022e-16
Hypre	GMRES	ILU	1	0	0.004284s	1.57009e-16	1.11022e-16
Trilinos	GMRES	ILU	1	0	0.0005s	2.71948e-16	1.92296e-16
QQQ	GMRES	ILU	1	0	0.004586s	1.57009e-16	1.11022e-16
PETSc	GMRES	ILU	1	0	0.019218s	4.96507e-16	3.51083e-16
ITL	CG	IC	2	0	0.000299s	4.96507e-16	3.51083e-16
Trilinos	CG	IC	1	0	0.000494s	3.33067e-16	2.35514e-16
PETSc	CG	IC	1	0	0.001608s	3.33067e-16	2.35514e-16
ITL	BICGstab	IC	1	0	0.000286s	3.84593e-16	2.71948e-16
Trilinos	BICGstab	IC	1	0	0.000386s	0	0
PETSc	BICGstab	IC	1	0	0.001239s	1.57009e-16	1.11022e-16
Trilinos	GMRES	IC	1	0	0.000467s	2.48253e-16	1.75542e-16
PETSc	GMRES	IC	1	0	0.018313s	4.96507e-16	3.51083e-16

Table A.3: Laplace rank 4 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	5	0	0.000113s	8.59975e-16	3.2504e-16
Hypre	CG	NONE	4	0	0.000106s	5.55112e-16	2.09812e-16
Trilinos	CG	NONE	4	0	0.000128s	5.55112e-16	2.09812e-16
PETSc	CG	NONE	4	0	0.000359s	2.71948e-16	1.02787e-16
ITL	BICGstab	NONE	4	0	0.000128s	5.55112e-16	2.09812e-16
Hypre	BICGstab	NONE	4	0	9.9e-05s	1.57009e-16	5.93439e-17
Trilinos	BICGstab	NONE	4	0	0.000111s	3.14018e-16	1.18688e-16
QQQ	BICGstab	NONE	4	0	0.000109s	7.69185e-16	2.90725e-16
PETSc	BICGstab	NONE	4	0	0.000362s	3.84593e-16	1.45362e-16
Hypre	GMRES	NONE	4	0	0.005221s	8.382e-16	3.1681e-16
Trilinos	GMRES	NONE	4	0	0.001395s	1.04738e-15	3.95873e-16
QQQ	GMRES	NONE	4	0	0.004774s	1.52226e-15	5.75361e-16
PETSc	GMRES	NONE	4	0	0.010735s	5.43896e-16	2.05573e-16
ITL	CG	Jacobi	5	0	0.000112s	8.59975e-16	3.2504e-16
Trilinos	CG	Jacobi	4	0	0.000137s	5.55112e-16	2.09812e-16
PETSc	CG	Jacobi	4	0	0.000399s	2.71948e-16	1.02787e-16
ITL	BICGstab	Jacobi	4	0	0.000127s	5.55112e-16	2.09812e-16
Trilinos	BICGstab	Jacobi	4	0	0.000122s	3.14018e-16	1.18688e-16
PETSc	BICGstab	Jacobi	4	0	0.000376s	3.84593e-16	1.45362e-16
Trilinos	GMRES	Jacobi	4	0	0.000144s	1.04738e-15	3.95873e-16
PETSc	GMRES	Jacobi	4	0	0.010281s	5.43896e-16	2.05573e-16
ITL	CG	ILU	2	0	0.000129s	2.48253e-16	9.3831e-17
Hypre	CG	ILU	1	0	0.000102s	6.66134e-16	2.51775e-16
Trilinos	CG	ILU	1	0	0.000214s	2.22045e-16	8.3925e-17
PETSc	CG	ILU	1	0	0.000704s	6.66134e-16	2.51775e-16
ITL	BICGstab	ILU	1	0	0.000106s	2.48253e-16	9.3831e-17
Hypre	BICGstab	ILU	1	0	9.7e-05s	1.11022e-16	4.19625e-17
Trilinos	BICGstab	ILU	1	0	0.000167s	3.33067e-16	1.25887e-16
QQQ	BICGstab	ILU	1	0	9e-05s	6.66134e-16	2.51775e-16
PETSc	BICGstab	ILU	1	0	0.000694s	6.75322e-16	2.55248e-16
Hypre	GMRES	ILU	1	0	0.003189s	5.43896e-16	2.05573e-16
Trilinos	GMRES	ILU	1	0	0.000239s	2.22045e-16	8.3925e-17
QQQ	GMRES	ILU	1	0	0.00448s	6.66134e-16	2.51775e-16
PETSc	GMRES	ILU	1	0	0.011167s	6.66134e-16	2.51775e-16
ITL	CG	IC	2	0	0.000114s	5.97873e-16	2.25975e-16
Trilinos	CG	IC	1	0	0.000252s	2.22045e-16	8.3925e-17
PETSc	CG	IC	1	0	0.000885s	2.22045e-16	8.3925e-17
ITL	BICGstab	IC	1	0	0.000108s	5.97873e-16	2.25975e-16
Trilinos	BICGstab	IC	1	0	0.000154s	3.33067e-16	1.25887e-16
PETSc	BICGstab	IC	1	0	0.000539s	2.22045e-16	8.3925e-17
Trilinos	GMRES	IC	1	0	0.000238s	2.22045e-16	8.3925e-17
PETSc	GMRES	IC	1	0	0.010801s	6.66134e-16	2.51775e-16

Table A.4: Laplace rank 5 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	6	0	0.000353s	2.71948e-16	1.92296e-16
Hypre	CG	NONE	5	0	0.000269s	4.44089e-16	3.14018e-16
Trilinos	CG	NONE	5	0	0.000325s	4.44089e-16	3.14018e-16
PETSc	CG	NONE	5	0	0.000744s	4.44089e-16	3.14018e-16
ITL	BICGstab	NONE	5	0	0.000429s	1.17495e-15	8.30815e-16
Hypre	BICGstab	NONE	5	0	0.000288s	4.96507e-16	3.51083e-16
Trilinos	BICGstab	NONE	5	0	0.000312s	2.1065e-15	1.48952e-15
QQQ	BICGstab	NONE	5	0	0.000227s	2.64364e-15	1.86933e-15
PETSc	BICGstab	NONE	5	0	0.000737s	3.14018e-16	2.22045e-16
Hypre	GMRES	NONE	5	0	0.006462s	4.44089e-16	3.14018e-16
Trilinos	GMRES	NONE	5	0	0.001597s	6.08094e-16	4.29988e-16
QQQ	GMRES	NONE	5	0	0.004681s	1.38667e-15	9.80522e-16
PETSc	GMRES	NONE	5	0	0.018655s	8.88178e-16	6.28037e-16
ITL	CG	Jacobi	6	0	0.000364s	2.71948e-16	1.92296e-16
Trilinos	CG	Jacobi	5	0	0.00034s	4.44089e-16	3.14018e-16
PETSc	CG	Jacobi	5	0	0.000772s	4.44089e-16	3.14018e-16
ITL	BICGstab	Jacobi	5	0	0.000444s	1.17495e-15	8.30815e-16
Trilinos	BICGstab	Jacobi	5	0	0.000326s	2.1065e-15	1.48952e-15
PETSc	BICGstab	Jacobi	5	0	0.00076s	3.14018e-16	2.22045e-16
Trilinos	GMRES	Jacobi	5	0	0.000353s	6.08094e-16	4.29988e-16
PETSc	GMRES	Jacobi	5	0	0.017783s	8.88178e-16	6.28037e-16
ITL	CG	ILU	2	0	0.000313s	2.71948e-16	1.92296e-16
Hypre	CG	ILU	1	0	0.000207s	2.71948e-16	1.92296e-16
Trilinos	CG	ILU	1	0	0.000495s	2.71948e-16	1.92296e-16
PETSc	CG	ILU	1	0	0.001518s	3.33067e-16	2.35514e-16
ITL	BICGstab	ILU	1	0	0.000303s	2.71948e-16	1.92296e-16
Hypre	BICGstab	ILU	1	0	0.000228s	0	0
Trilinos	BICGstab	ILU	1	0	0.000424s	0	0
QQQ	BICGstab	ILU	1	0	0.000203s	3.84593e-16	2.71948e-16
PETSc	BICGstab	ILU	1	0	0.001479s	3.33067e-16	2.35514e-16
Hypre	GMRES	ILU	1	0	0.004408s	2.71948e-16	1.92296e-16
Trilinos	GMRES	ILU	1	0	0.000509s	4.00297e-16	2.83052e-16
QQQ	GMRES	ILU	1	0	0.004716s	9.15513e-16	6.47366e-16
PETSc	GMRES	ILU	1	0	0.019294s	1.92296e-16	1.35974e-16
ITL	CG	IC	2	0	0.000304s	4.44089e-16	3.14018e-16
Trilinos	CG	IC	1	0	0.000499s	6.66134e-16	4.71028e-16
PETSc	CG	IC	1	0	0.00163s	6.66134e-16	4.71028e-16
ITL	BICGstab	IC	1	0	0.000294s	4.44089e-16	3.14018e-16
Trilinos	BICGstab	IC	1	0	0.00039s	0	0
PETSc	BICGstab	IC	1	0	0.001263s	0	0
Trilinos	GMRES	IC	1	0	0.000473s	3.33067e-16	2.35514e-16
PETSc	GMRES	IC	1	0	0.018294s	6.66134e-16	4.71028e-16

Table A.5: Laplace rank 10 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	4	1	0.000224s	4.20719e-09	1.53968e-09
Hypre	CG	NONE	4	1	0.000176s	4.21041e-09	1.54086e-09
Trilinos	CG	NONE	4	1	0.000311s	5.14823e-09	1.88407e-09
PETSc	CG	NONE	4	1	0.000526s	1.7421e-10	6.37547e-11
ITL	BICGstab	NONE	4	1	0.00027s	3.16189e-08	1.15714e-08
Hypre	BICGstab	NONE	4	1	0.000156s	5.80364e-08	2.12393e-08
Trilinos	BICGstab	NONE	4	1	0.000297s	3.69441e-08	1.35202e-08
QQQ	BICGstab	NONE	4	1	0.00016s	2.71695e-08	9.94306e-09
PETSc	BICGstab	NONE	4	1	0.000516s	1.19062e-08	4.35726e-09
Hypre	GMRES	NONE	4	0	0.005288s	4.57757e-16	1.67523e-16
Trilinos	GMRES	NONE	4	0	0.00149s	2.22045e-16	8.12604e-17
QQQ	GMRES	NONE	4	0	0.004482s	5.55112e-16	2.03151e-16
PETSc	GMRES	NONE	4	0	0.011031s	5.08768e-16	1.86191e-16
ITL	CG	Jacobi	4	1	0.000229s	9.41892e-10	3.44699e-10
Trilinos	CG	Jacobi	4	1	0.000332s	3.19936e-09	1.17085e-09
PETSc	CG	Jacobi	4	1	0.000557s	2.50357e-09	9.16217e-10
ITL	BICGstab	Jacobi	4	1	0.000278s	2.19324e-06	8.02646e-07
Trilinos	BICGstab	Jacobi	4	1	0.00031s	2.19327e-06	8.02657e-07
PETSc	BICGstab	Jacobi	4	1	0.000522s	2.4326e-06	8.90245e-07
Trilinos	GMRES	Jacobi	4	0	0.000235s	2.48253e-16	9.08519e-17
PETSc	GMRES	Jacobi	4	0	0.010467s	6.97054e-13	2.55097e-13
ITL	CG	ILU	2	0	0.000211s	0	0
Hypre	CG	ILU	1	0	0.000147s	0	0
Trilinos	CG	ILU	1	0	0.00035s	0	0
PETSc	CG	ILU	1	0	0.001137s	0	0
ITL	BICGstab	ILU	1	0	0.000198s	0	0
Hypre	BICGstab	ILU	0	3	0.000128s	nan	nan
Trilinos	BICGstab	ILU	1	0	0.000287s	0	0
QQQ	BICGstab	ILU	1	0	0.000142s	2.71948e-16	9.95232e-17
PETSc	BICGstab	ILU	1	0	0.001105s	0	0
Hypre	GMRES	ILU	1	0	0.003308s	1.11022e-16	4.06302e-17
Trilinos	GMRES	ILU	1	0	0.000363s	1.11022e-16	4.06302e-17
QQQ	GMRES	ILU	1	0	0.004539s	1.04266e-14	3.81578e-15
PETSc	GMRES	ILU	1	0	0.01158s	1.04266e-14	3.81578e-15
ITL	CG	IC	2	0	0.000205s	0	0
Trilinos	CG	IC	1	0	0.000367s	5.20741e-16	1.90572e-16
PETSc	CG	IC	1	0	0.00128s	2.71948e-16	9.95232e-17
ITL	BICGstab	IC	1	0	0.000195s	0	0
Trilinos	BICGstab	IC	1	0	0.000265s	0	0
PETSc	BICGstab	IC	1	0	0.000892s	2.48253e-16	9.08519e-17
Trilinos	GMRES	IC	1	0	0.000346s	2.22045e-16	8.12604e-17
PETSc	GMRES	IC	1	0	0.010994s	7.03219e-15	2.57353e-15

Table A.6: Hilbert rank 4 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	10	1	0.00031s	6.927e-08	1.47075e-08
Hypre	CG	NONE	10	1	0.000325s	1.53567e-07	3.26056e-08
Trilinos	CG	NONE	10	1	0.000489s	2.08801e-07	4.4333e-08
PETSc	CG	NONE	10	1	0.000795s	2.37618e-07	5.04514e-08
ITL	BICGstab	NONE	10	1	0.000644s	1.40222e-10	2.97722e-11
Hypre	BICGstab	NONE	10	1	0.00031s	1.41216e-10	2.99833e-11
Trilinos	BICGstab	NONE	10	1	0.000588s	1.06781e-10	2.26718e-11
QQQ	BICGstab	NONE	10	1	0.000258s	1.20492e-10	2.5583e-11
PETSc	BICGstab	NONE	10	1	0.000908s	1.28341e-10	2.72495e-11
Hypre	GMRES	NONE	7	0	0.005538s	3.42065e-13	7.26277e-14
Trilinos	GMRES	NONE	7	0	0.001651s	3.42137e-13	7.26431e-14
QQQ	GMRES	NONE	7	0	0.00477s	3.42054e-13	7.26253e-14
PETSc	GMRES	NONE	7	0	0.011307s	3.69428e-13	7.84375e-14
ITL	CG	Jacobi	10	1	0.000481s	1.43939e-07	3.05613e-08
Trilinos	CG	Jacobi	10	1	0.000617s	4.53603e-08	9.63096e-09
PETSc	CG	Jacobi	10	1	0.000922s	5.31169e-08	1.12779e-08
ITL	BICGstab	Jacobi	10	1	0.00068s	4.5955e-06	9.75723e-07
Trilinos	BICGstab	Jacobi	9	3	0.000616s	4.5954e-06	9.75702e-07
PETSc	BICGstab	Jacobi	10	1	0.000918s	0.000665968	0.000141399
Trilinos	GMRES	Jacobi	7	0	0.000407s	3.43116e-13	7.28508e-14
PETSc	GMRES	Jacobi	10	1	0.01095s	1.78515e-10	3.79026e-11
ITL	CG	ILU	2	0	0.000436s	0	0
Hypre	CG	ILU	10	1	0.000408s	6.63835e-08	1.40947e-08
Trilinos	CG	ILU	1	0	0.000514s	0	0
PETSc	CG	ILU	6	0	0.001753s	6.37775e-16	1.35413e-16
ITL	BICGstab	ILU	1	0	0.000405s	0	0
Hypre	BICGstab	ILU	3	0	0.000285s	5.73006e-13	1.21662e-13
Trilinos	BICGstab	ILU	1	0	0.000452s	0	0
QQQ	BICGstab	ILU	5	0	0.000306s	3.00062e-13	6.37097e-14
PETSc	BICGstab	ILU	5	0	0.001712s	5.32444e-16	1.13049e-16
Hypre	GMRES	ILU	2	0	0.003408s	5.57132e-13	1.18291e-13
Trilinos	GMRES	ILU	1	0	0.000549s	6.47366e-16	1.3745e-16
QQQ	GMRES	ILU	3	0	1.39602s	8.382e-16	1.77968e-16
PETSc	GMRES	ILU	4	0	0.006615s	3.76986e-15	8.00422e-16
ITL	CG	IC	2	0	0.000285s	0	0
Trilinos	CG	IC	1	0	0.000391s	0	0
PETSc	CG	IC	3	0	0.001437s	4.57757e-16	9.71916e-17
ITL	BICGstab	IC	1	0	0.000272s	0	0
Trilinos	BICGstab	IC	1	0	0.000292s	0	0
PETSc	BICGstab	IC	2	0	0.000967s	4.96507e-16	1.05419e-16
Trilinos	GMRES	IC	1	0	0.000375s	6.47366e-16	1.3745e-16
PETSc	GMRES	IC	3	0	0.011162s	3.84593e-16	8.16573e-17

Table A.7: Hilbert rank 10 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	28	0	0.002271s	3.19848e-12	2.00532e-13
Hypre	CG	NONE	27	0	0.001311s	1.92737e-12	1.20838e-13
Trilinos	CG	NONE	28	0	0.001402s	1.21157e-12	7.59605e-14
PETSc	CG	NONE	28	0	0.002516s	2.90365e-12	1.82047e-13
ITL	BICGstab	NONE	56	0	0.008232s	2.81127e-12	1.76255e-13
Hypre	BICGstab	NONE	57	0	0.003858s	1.58843e-11	9.95884e-13
Trilinos	BICGstab	NONE	72	0	0.004737s	1.03206e-11	6.47061e-13
QQQ	BICGstab	NONE	43	0	0.009032s	1.03347e-11	6.47943e-13
PETSc	BICGstab	NONE	49	0	0.005225s	1.02169e-11	6.40559e-13
Hypre	GMRES	NONE	12	0	0.006365s	1.20622e-12	7.56252e-14
Trilinos	GMRES	NONE	12	0	0.00209s	1.20636e-12	7.56337e-14
QQQ	GMRES	NONE	12	0	0.006851s	1.20658e-12	7.56475e-14
PETSc	GMRES	NONE	12	0	0.01224s	2.61782e-12	1.64127e-13
ITL	CG	Jacobi	28	0	0.002286s	2.0414e-12	1.27987e-13
Trilinos	CG	Jacobi	27	0	0.001505s	5.66709e-12	3.55304e-13
PETSc	CG	Jacobi	30	0	0.002715s	7.83941e-12	4.91499e-13
ITL	BICGstab	Jacobi	50	5	0.007504s	4.62261e-05	2.89819e-06
Trilinos	BICGstab	Jacobi	16	3	0.001539s	4.89168e-05	3.06689e-06
PETSc	BICGstab	Jacobi	100	1	0.009801s	0.012549	0.00078677
Trilinos	GMRES	Jacobi	12	0	0.000979s	1.26878e-12	7.95473e-14
PETSc	GMRES	Jacobi	100	1	0.020344s	4.19833e-08	2.63219e-09
ITL	CG	ILU	2	0	0.043619s	9.6064e-14	6.02283e-15
Hypre	CG	ILU	100	1	0.007659s	355.635	22.2969
Trilinos	CG	ILU	1	0	0.005575s	2.74205e-15	1.71915e-16
PETSc	CG	ILU	65	8	0.012643s	7.4135e-08	4.64797e-09
ITL	BICGstab	ILU	1	0	0.043425s	7.4244e-15	4.6548e-16
Hypre	BICGstab	ILU	100	1	0.010346s	0.000294372	1.84559e-05
Trilinos	BICGstab	ILU	1	0	0.005286s	3.00376e-15	1.88324e-16
QQQ	BICGstab	ILU	8	0	0.120315s	1.46383e-11	9.17761e-13
PETSc	BICGstab	ILU	100	1	0.019756s	4.41937e-09	2.77077e-10
Hypre	GMRES	ILU	100	1	0.011315s	1.40817e-07	8.82863e-09
Trilinos	GMRES	ILU	1	0	0.004998s	2.96244e-15	1.85733e-16
QQQ	GMRES	ILU	100	0	1.92864s	9.84294e-14	6.17113e-15
PETSc	GMRES	ILU	100	1	0.022275s	2.35607e-13	1.47716e-14
ITL	CG	IC	2	5	0.059313s	nan	nan
Trilinos	CG	IC	1	3	0.043062s	nan	nan
PETSc	CG	IC	100	1	0.020455s	9.04494e-10	5.67081e-11
ITL	BICGstab	IC	1	5	0.059175s	nan	nan
Trilinos	BICGstab	IC	1	3	0.044934s	nan	nan
PETSc	BICGstab	IC	100	1	0.023955s	3.63423e-13	2.27851e-14
Trilinos	GMRES	IC	1	3	0.040762s	15.95	1
PETSc	GMRES	IC	100	1	0.029418s	4.0757e-12	2.5553e-13

Table A.8: Hilbert rank 100 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	46	0	0.236592s	2.10316e-11	4.12673e-13
Hypre	CG	NONE	44	0	0.11377s	2.0448e-11	4.01222e-13
Trilinos	CG	NONE	44	0	0.116052s	2.04162e-11	4.00598e-13
PETSc	CG	NONE	49	0	0.155108s	3.1622e-11	6.20473e-13
ITL	BICGstab	NONE	125	0	1.29165s	2.12603e-11	4.1716e-13
Hypre	BICGstab	NONE	131	0	0.676206s	1.79792e-11	3.5278e-13
Trilinos	BICGstab	NONE	151	0	0.765636s	3.34408e-11	6.56162e-13
QQQ	BICGstab	NONE	143	0	3.80804s	4.80739e-11	9.43286e-13
PETSc	BICGstab	NONE	119	0	0.740253s	4.78309e-11	9.38518e-13
Hypre	GMRES	NONE	16	0	0.052864s	2.02465e-11	3.97269e-13
Trilinos	GMRES	NONE	16	0	0.061722s	2.02466e-11	3.97271e-13
QQQ	GMRES	NONE	16	0	0.233662s	2.02469e-11	3.97277e-13
PETSc	GMRES	NONE	18	0	0.065736s	8.8348e-11	1.73353e-12
ITL	CG	Jacobi	48	0	0.248s	4.46602e-11	8.76305e-13
Trilinos	CG	Jacobi	51	0	0.142618s	2.39416e-11	4.69773e-13
PETSc	CG	Jacobi	50	0	0.162431s	4.44202e-11	8.71595e-13
ITL	BICGstab	Jacobi	1000	1	10.2725s	0.000259781	5.09732e-06
Trilinos	BICGstab	Jacobi	17	3	0.093759s	0.00177622	3.48523e-05
PETSc	BICGstab	Jacobi	798	7	4.93399s	3.69026e+07	724087
Trilinos	GMRES	Jacobi	16	0	0.054442s	2.47582e-11	4.85795e-13
PETSc	GMRES	Jacobi	1000	1	5.97811s	1.7516e-06	3.43691e-08
ITL	CG	ILU	191	5	36.0003s	nan	nan
Hypre	CG	ILU	1000	1	3.94704s	38.5607	0.756622
Trilinos	CG	ILU	1	0	2.07958s	2.71255e-12	5.32245e-14
PETSc	CG	ILU	285	8	4.63593s	2.14401e-06	4.20689e-08
ITL	BICGstab	ILU	1	0	33.863s	1.68221e-12	3.30077e-14
Hypre	BICGstab	ILU	1000	1	6.63145s	0.000274096	5.37821e-06
Trilinos	BICGstab	ILU	1	0	2.08542s	1.50324e-12	2.9496e-14
QQQ	BICGstab	ILU	730	0	94.2204s	nan	nan
PETSc	BICGstab	ILU	771	7	12.9524s	9.79664	0.192226
Hypre	GMRES	ILU	460	0	2.79304s	3.67954e-11	7.21984e-13
Trilinos	GMRES	ILU	1	0	2.10316s	2.24399e-12	4.40306e-14
QQQ	GMRES	ILU	479	0	20.5125s	0.734338	0.0144089
PETSc	GMRES	ILU	1000	1	12.2122s	2.01663e-08	3.95695e-10
ITL	CG	IC	2	5	62.8814s	nan	nan
Trilinos	CG	IC	1	3	53.2338s	nan	nan
PETSc	CG	IC	1000	1	10.8735s	2.10377e-09	4.12793e-11
ITL	BICGstab	IC	1	5	62.8922s	nan	nan
Trilinos	BICGstab	IC	1	3	53.4243s	nan	nan
PETSc	BICGstab	IC	1000	1	19.9753s	3.06529e-07	6.01459e-09
Trilinos	GMRES	IC	1	3	53.2426s	50.9643	1
PETSc	GMRES	IC	1000	1	13.6084s	9.25948e-12	1.81686e-13

Table A.9: Hilbert rank 1000 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	216	1	0.009474s	2.03408e-06	0.00104844
Hypre	CG	NONE	216	1	0.004741s	3.07566e-06	0.00158531
Trilinos	CG	NONE	14	3	0.000463s	0.00697383	3.59457
PETSc	CG	NONE	14	8	0.001925s	0.00697383	3.59457
ITL	BICGstab	NONE	216	1	0.018788s	2.43245e-05	0.0125378
Hypre	BICGstab	NONE	216	1	0.007994s	4.54973e-05	0.023451
Trilinos	BICGstab	NONE	216	1	0.007586s	1.49028e-05	0.00768145
QQQ	BICGstab	NONE	216	1	0.018279s	1.58748e-05	0.00818249
PETSc	BICGstab	NONE	216	1	0.020728s	2.28318e-05	0.0117684
Hypre	GMRES	NONE	216	0	0.024286s	5.73681e-17	2.95697e-14
Trilinos	GMRES	NONE	216	0	0.031384s	2.68214e-17	1.38247e-14
QQQ	GMRES	NONE	216	0	0.069713s	1.0976e-16	5.65745e-14
PETSc	GMRES	NONE	216	0	0.049644s	1.41014e-16	7.26837e-14
ITL	CG	Jacobi	2	5	0.000843s	nan	nan
Trilinos	CG	Jacobi	31	3	0.000969s	0.000161776	0.0833852
PETSc	CG	Jacobi	10	5	0.00179s	0.0220573	11.3692
ITL	BICGstab	Jacobi	1	5	0.000765s	nan	nan
Trilinos	BICGstab	Jacobi	50	3	0.002085s	0.000161776	0.0833852
PETSc	BICGstab	Jacobi	216	1	0.021064s	1.07939e-05	0.00556357
Trilinos	GMRES	Jacobi	26	4	0.00128s	0.000161106	0.0830401
PETSc	GMRES	Jacobi	216	1	0.049476s	1.74919e-05	0.00901597
Hypre	CG	ILU	216	1	0.010609s	0.00228652	1.17856
Trilinos	CG	ILU	216	1	0.015368s	1.08371e-13	5.58586e-11
PETSc	CG	ILU	1	6	0.001867s	0.0019401	1
Hypre	BICGstab	ILU	52	0	0.005539s	1.5862e-15	8.17587e-13
Trilinos	BICGstab	ILU	2	3	0.003389s	2.0337e-07	0.000104825
QQQ	BICGstab	ILU	9	0	0.006086s	4.80331e-16	2.47581e-13
PETSc	BICGstab	ILU	17	0	0.004904s	6.04178e-16	3.11416e-13
Hypre	GMRES	ILU	48	0	0.008215s	1.6735e-15	8.62583e-13
Trilinos	GMRES	ILU	6	0	0.003677s	4.82309e-16	2.486e-13
QQQ	GMRES	ILU	5	0	0.04648s	1.99352e-17	1.02753e-14
PETSc	GMRES	ILU	27	0	0.015625s	2.07441e-14	1.06923e-11
Trilinos	CG	IC	1	3	0.001949s	0.0019401	1
PETSc	CG	IC	1	8	0.001861s	0.0019401	1
Trilinos	BICGstab	IC	15	0	0.002922s	1.21323e-15	6.25344e-13
PETSc	BICGstab	IC	165	0	0.027896s	5.43052e-17	2.79909e-14
Trilinos	GMRES	IC	24	0	0.004478s	7.1161e-16	3.6679e-13
PETSc	GMRES	IC	72	0	0.02156s	1.70953e-13	8.81154e-11

Table A.10: Fidap001 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	441	1	0.075503s	72.935	10.5997
Hypre	CG	NONE	441	1	0.032999s	1413.43	205.416
Trilinos	CG	NONE	1	3	0.000448s	6.88082	1
PETSc	CG	NONE	1	8	0.001905s	6.88082	1
ITL	BICGstab	NONE	441	1	0.147967s	480.292	69.8016
Hypre	BICGstab	NONE	441	1	0.059716s	2690.68	391.041
Trilinos	BICGstab	NONE	441	1	0.058336s	2461.82	357.78
QQQ	BICGstab	NONE	441	1	0.170636s	433.943	63.0657
PETSc	BICGstab	NONE	441	1	0.094213s	268.37	39.0026
Hypre	GMRES	NONE	441	1	0.148011s	8.59851e-07	1.24963e-07
Trilinos	GMRES	NONE	182	2	0.050207s	1.11923e-07	1.62659e-08
QQQ	GMRES	NONE	441	1	0.569367s	0.808758	0.117538
PETSc	GMRES	NONE	441	1	0.281105s	0.00921185	0.00133877
ITL	CG	Jacobi	441	1	0.074537s	28.129	4.08804
Trilinos	CG	Jacobi	1	3	0.000713s	6.88082	1
PETSc	CG	Jacobi	1	6	0.001856s	6.88082	1
ITL	BICGstab	Jacobi	441	1	0.147733s	389.591	56.6198
Trilinos	BICGstab	Jacobi	441	1	0.060231s	391.585	56.9097
PETSc	BICGstab	Jacobi	441	1	0.093884s	21.1755	3.07747
Trilinos	GMRES	Jacobi	431	2	0.236927s	1.19691e-06	1.73949e-07
PETSc	GMRES	Jacobi	441	1	0.279951s	355.103	51.6076
ITL	CG	ILU	441	1	0.244323s	2.69292e-07	3.91366e-08
Hypre	CG	ILU	441	1	0.049234s	6.88087	1.00001
Trilinos	CG	ILU	19	2	0.022473s	6.06136e-08	8.80906e-09
PETSc	CG	ILU	2	6	0.007213s	17.4231	2.53212
ITL	BICGstab	ILU	396	0	0.358785s	2.70652e-07	3.93343e-08
Hypre	BICGstab	ILU	441	1	0.092369s	8.92243e+10	1.29671e+10
Trilinos	BICGstab	ILU	4	2	0.020178s	1.36996e-08	1.99099e-09
QQQ	BICGstab	ILU	29	0	0.473228s	1.39157e-07	2.02238e-08
PETSc	BICGstab	ILU	441	1	0.17704s	0.0617003	0.00896701
Hypre	GMRES	ILU	441	1	0.162195s	4.27121e+10	6.20741e+09
Trilinos	GMRES	ILU	6	2	0.019838s	6.42809e-10	9.34204e-11
QQQ	GMRES	ILU	2	0	0.290719s	6.88082	1
PETSc	GMRES	ILU	441	1	0.330102s	0.076763	0.0111561
ITL	CG	IC	2	5	0.1389s	nan	nan
Trilinos	CG	IC	113	2	0.029655s	6.98804e-08	1.01558e-08
PETSc	CG	IC	1	5	0.011784s	6.88082	1
ITL	BICGstab	IC	1	5	0.139376s	nan	nan
Trilinos	BICGstab	IC	166	2	0.062446s	6.26726e-07	9.1083e-08
PETSc	BICGstab	IC	441	1	0.256739s	924.007	134.287
Trilinos	GMRES	IC	56	2	0.025388s	6.28489e-08	9.13392e-09
PETSc	GMRES	IC	441	1	0.369817s	0.0293541	0.00426608

Table A.11: Fidap002 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	27	1	0.000409s	0.0603102	0.923308
Hypre	CG	NONE	27	1	0.000318s	0.065567	1.00379
Trilinos	CG	NONE	27	1	0.000409s	0.0829542	1.26997
PETSc	CG	NONE	27	1	0.000851s	38.524	589.777
ITL	BICGstab	NONE	27	1	0.000642s	0.0823034	1.26001
Hypre	BICGstab	NONE	27	1	0.000352s	0.0738414	1.13046
Trilinos	BICGstab	NONE	27	1	0.000437s	0.52968	8.10904
QQQ	BICGstab	NONE	27	1	0.000372s	0.0722653	1.10633
PETSc	BICGstab	NONE	27	1	0.000986s	0.0724119	1.10858
Hypre	GMRES	NONE	27	1	0.00563s	1.20693e-10	1.84772e-09
Trilinos	GMRES	NONE	27	2	0.000625s	2.2913e-10	3.50782e-09
QQQ	GMRES	NONE	27	1	0.01276s	7.1664e-07	1.09713e-05
PETSc	GMRES	NONE	27	1	0.011314s	7.13564e-09	1.09242e-07
ITL	CG	Jacobi	27	1	0.000399s	0.0759657	1.16298
Trilinos	CG	Jacobi	27	1	0.000439s	0.186635	2.85725
PETSc	CG	Jacobi	27	1	0.0009s	3.09106	47.322
ITL	BICGstab	Jacobi	27	1	0.000668s	0.897626	13.742
Trilinos	BICGstab	Jacobi	27	1	0.000466s	0.10358	1.58574
PETSc	BICGstab	Jacobi	27	1	0.001008s	0.0472817	0.72385
Trilinos	GMRES	Jacobi	27	2	0.000627s	2.31564e-10	3.54508e-09
PETSc	GMRES	Jacobi	27	1	0.011113s	0.0305118	0.467115
ITL	CG	ILU	20	0	0.000784s	9.5835e-11	1.46717e-09
Hypre	CG	ILU	27	1	0.000392s	2.33284	35.7142
Trilinos	CG	ILU	2	2	0.000497s	8.77969e-11	1.34411e-09
PETSc	CG	ILU	1	6	0.000836s	0.0653197	1
ITL	BICGstab	ILU	23	0	0.001163s	2.32982e-10	3.56679e-09
Hypre	BICGstab	ILU	27	1	0.000462s	0.176004	2.6945
Trilinos	BICGstab	ILU	1	2	0.000426s	8.39537e-11	1.28527e-09
QQQ	BICGstab	ILU	8	0	0.001293s	1.08723e-10	1.66447e-09
PETSc	BICGstab	ILU	18	0	0.001341s	3.25371e-09	4.9812e-08
Hypre	GMRES	ILU	27	1	0.003555s	3.59605e-11	5.50531e-10
Trilinos	GMRES	ILU	2	2	0.000528s	1.04463e-11	1.59925e-10
QQQ	GMRES	ILU	3	0	1.34561s	6.97003e-11	1.06706e-09
PETSc	GMRES	ILU	11	0	0.005876s	2.08155e-08	3.18671e-07
ITL	CG	IC	2	5	0.000512s	nan	nan
Trilinos	CG	IC	2	2	0.000458s	5.31794e-11	8.1414e-10
PETSc	CG	IC	17	0	0.001919s	2.54017e-10	3.88882e-09
ITL	BICGstab	IC	1	5	0.000461s	nan	nan
Trilinos	BICGstab	IC	1	2	0.000346s	5.94119e-11	9.09556e-10
PETSc	BICGstab	IC	16	0	0.001146s	1.08307e-10	1.65811e-09
Trilinos	GMRES	IC	2	2	0.001703s	1.47092e-11	2.25187e-10
PETSc	GMRES	IC	15	0	0.011292s	3.32734e-10	5.09393e-09

Table A.12: Fidap005 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	42	1	0.000999s	1.44987e-06	2.21965e-05
Hypre	CG	NONE	42	1	0.000755s	2.62163e-06	4.01354e-05
Trilinos	CG	NONE	11	3	0.000428s	0.0844426	1.29276
PETSc	CG	NONE	11	8	0.000848s	0.0844426	1.29276
ITL	BICGstab	NONE	42	1	0.001776s	0.000553869	0.00847936
Hypre	BICGstab	NONE	42	1	0.000867s	0.000673926	0.0103173
Trilinos	BICGstab	NONE	42	1	0.000973s	0.000612725	0.00938039
QQQ	BICGstab	NONE	42	1	0.000814s	0.000412576	0.00631626
PETSc	BICGstab	NONE	42	1	0.001929s	0.00068205	0.0104417
Hypre	GMRES	NONE	41	0	0.006291s	8.80907e-16	1.34861e-14
Trilinos	GMRES	NONE	41	0	0.002384s	9.8949e-16	1.51484e-14
QQQ	GMRES	NONE	41	0	0.005646s	1.99601e-15	3.05576e-14
PETSc	GMRES	NONE	41	0	0.01269s	6.27715e-15	9.60989e-14
ITL	CG	Jacobi	2	5	0.000271s	nan	nan
Trilinos	CG	Jacobi	17	3	0.000568s	0.00911024	0.139472
PETSc	CG	Jacobi	6	8	0.000765s	0.262144	4.01325
ITL	BICGstab	Jacobi	1	5	0.00026s	nan	nan
Trilinos	BICGstab	Jacobi	32	3	0.000865s	0.00911024	0.139472
PETSc	BICGstab	Jacobi	42	1	0.001967s	1.76768e-05	0.000270619
Trilinos	GMRES	Jacobi	16	4	0.000624s	0.00891275	0.136448
PETSc	GMRES	Jacobi	42	1	0.012228s	2.71819e-07	4.16136e-06
Hypre	CG	ILU	42	1	0.001145s	0.142913	2.1879
Trilinos	CG	ILU	1	3	0.001481s	nan	nan
PETSc	CG	ILU	2	6	0.001329s	1.44603	22.1377
Hypre	BICGstab	ILU	42	1	0.001399s	2.03919e-12	3.12185e-11
Trilinos	BICGstab	ILU	1	3	0.001556s	nan	nan
QQQ	BICGstab	ILU	11	0	0.002045s	4.72651e-14	7.23596e-13
PETSc	BICGstab	ILU	39	0	0.002855s	2.94172e-14	4.50358e-13
Hypre	GMRES	ILU	42	1	0.004654s	2.81272e-12	4.30608e-11
Trilinos	GMRES	ILU	1	3	0.001381s	0.0653197	1
QQQ	GMRES	ILU	2	0	0.083597s	9.52165e-09	1.4577e-07
PETSc	GMRES	ILU	42	1	0.013707s	1.41543e-08	2.16693e-07
Trilinos	CG	IC	1	3	0.001163s	nan	nan
PETSc	CG	IC	5	8	0.001897s	0.125969	1.9285
Trilinos	BICGstab	IC	1	3	0.001364s	nan	nan
PETSc	BICGstab	IC	42	1	0.00291s	1.35377e-06	2.07253e-05
Trilinos	GMRES	IC	1	3	0.001198s	0.0653197	1
PETSc	GMRES	IC	42	1	0.013159s	1.03194e-08	1.57983e-07

Table A.13: Fidapm05 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	974	1	0.260724s	0.000735395	1.46914
Hypre	CG	NONE	974	1	0.100546s	0.00144761	2.89198
Trilinos	CG	NONE	10	3	0.001441s	0.00325129	6.49529
PETSc	CG	NONE	10	8	0.00767s	0.00325129	6.49529
ITL	BICGstab	NONE	974	1	0.492048s	1.66841e-05	0.0333308
Hypre	BICGstab	NONE	974	1	0.194319s	2.36322e-05	0.0472113
Trilinos	BICGstab	NONE	974	1	0.189586s	2.24524e-05	0.0448545
QQQ	BICGstab	NONE	974	1	0.578349s	2.96936e-05	0.0593205
PETSc	BICGstab	NONE	974	1	0.485854s	1.41414e-05	0.0282511
Hypre	GMRES	NONE	890	0	1.148s	3.74739e-16	7.48638e-13
Trilinos	GMRES	NONE	890	0	2.31263s	3.78224e-16	7.556e-13
QQQ	GMRES	NONE	890	0	4.52542s	3.90853e-16	7.8083e-13
PETSc	GMRES	NONE	974	1	2.82672s	1.75492e-11	3.50591e-08
ITL	CG	Jacobi	2	5	0.00451s	nan	nan
Trilinos	CG	Jacobi	122	3	0.015285s	1.35809e-05	0.0271314
PETSc	CG	Jacobi	36	8	0.014303s	0.000378492	0.756136
ITL	BICGstab	Jacobi	1	5	0.004305s	nan	nan
Trilinos	BICGstab	Jacobi	974	1	0.2067s	1.3598e-05	0.0271655
PETSc	BICGstab	Jacobi	974	1	0.502673s	4.64069e-06	0.00927097
Trilinos	GMRES	Jacobi	95	4	0.032582s	1.35565e-05	0.0270827
PETSc	GMRES	Jacobi	974	1	2.83895s	8.48254e-06	0.0169461
Hypre	CG	ILU	974	1	0.219448s	0.0190443	38.0458
Trilinos	CG	ILU	1	3	0.133364s	nan	nan
PETSc	CG	ILU	2	6	0.012554s	0.000336813	0.672871
Hypre	BICGstab	ILU	974	1	0.40124s	0.192553	384.675
Trilinos	BICGstab	ILU	1	3	0.140919s	nan	nan
QQQ	BICGstab	ILU	9	0	0.096451s	4.95299e-16	9.89488e-13
PETSc	BICGstab	ILU	19	0	0.031221s	6.12014e-16	1.22266e-12
Hypre	GMRES	ILU	974	1	1.56428s	0.564703	1128.14
Trilinos	GMRES	ILU	1	3	0.125118s	0.000500561	1
QQQ	GMRES	ILU	14	0	0.1562s	0.00120665	2.4106
PETSc	GMRES	ILU	29	0	0.039871s	1.75185e-15	3.49976e-12

Table A.14: Fidap027 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	2603	1	1.46291s	0.269311	0.190171
Hypre	CG	NONE	2603	1	0.579195s	0.163464	0.115428
Trilinos	CG	NONE	12	3	0.003191s	17.5205	12.3719
PETSc	CG	NONE	12	8	0.015875s	17.5205	12.3719
ITL	BICGstab	NONE	1699	5	1.84716s	0.56624	0.399844
Hypre	BICGstab	NONE	2603	1	1.10887s	0.29724	0.209893
Trilinos	BICGstab	NONE	117	3	0.048506s	0.861267	0.608175
QQQ	BICGstab	NONE	2603	1	3.18844s	0.253346	0.178897
PETSc	BICGstab	NONE	2603	1	2.50979s	1.10728	0.781896
Hypre	GMRES	NONE	2603	1	10.0036s	0.000106624	7.52915e-05
Trilinos	GMRES	NONE	2603	1	22.8405s	0.000106748	7.53792e-05
QQQ	GMRES	NONE	2603	1	36.5571s	0.000158039	0.000111598
PETSc	GMRES	NONE	2603	1	18.607s	0.000106715	7.53555e-05
ITL	CG	Jacobi	2	5	0.009695s	nan	nan
Trilinos	CG	Jacobi	263	3	0.059078s	0.692432	0.488954
PETSc	CG	Jacobi	4	8	0.01061s	30.7697	21.7277
ITL	BICGstab	Jacobi	1	5	0.009138s	nan	nan
Trilinos	BICGstab	Jacobi	2603	1	1.12053s	9.91998e+13	7.0049e+13
PETSc	BICGstab	Jacobi	2603	1	2.51247s	2.02845e-05	1.43237e-05
Trilinos	GMRES	Jacobi	134	4	0.152653s	0.667513	0.471357
PETSc	GMRES	Jacobi	2603	1	18.1686s	4.21404e-08	2.9757e-08
Hypre	CG	ILU	2603	1	1.27041s	11.5292	8.14122
Trilinos	CG	ILU	1	3	0.205077s	nan	nan
PETSc	CG	ILU	1	6	0.019881s	1.41615	1
Hypre	BICGstab	ILU	2603	1	2.48026s	5.27602e+18	3.72561e+18
Trilinos	BICGstab	ILU	1	3	0.222067s	nan	nan
QQQ	BICGstab	ILU	28	0	1.23864s	2.47651e-12	1.74876e-12
PETSc	BICGstab	ILU	106	0	0.226963s	4.39446e-12	3.1031e-12
Hypre	GMRES	ILU	2603	1	11.3511s	1.26717e+08	8.94797e+07
Trilinos	GMRES	ILU	1	3	0.183214s	1.41615	1
QQQ	GMRES	ILU	16	0	0.645006s	13.4038	9.46493
PETSc	GMRES	ILU	89	0	0.169645s	2.57675e-11	1.81954e-11

Table A.15: Fidap028 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	1080	1	0.207139s	6.40783e+24	3.67414e+16
Hypre	CG	NONE	1080	1	0.089774s	7.19293e+25	4.1243e+17
Trilinos	CG	NONE	1	3	0.000464s	1.74403e+08	1
PETSc	CG	NONE	1	8	0.003869s	1.74403e+08	1
ITL	BICGstab	NONE	1080	1	0.412768s	5.29378e+07	0.303537
Hypre	BICGstab	NONE	1080	1	0.169693s	4.01545e+06	0.0230239
Trilinos	BICGstab	NONE	1080	1	0.162694s	4.12918e+06	0.023676
QQQ	BICGstab	NONE	1080	1	0.596152s	4.25507e+06	0.0243978
PETSc	BICGstab	NONE	1080	1	0.448228s	1.76822e+08	1.01387
Hypre	GMRES	NONE	792	0	0.985469s	0.000147067	8.43257e-13
Trilinos	GMRES	NONE	793	0	2.00764s	0.000146049	8.37419e-13
QQQ	GMRES	NONE	792	0	3.90627s	0.000136862	7.84745e-13
PETSc	GMRES	NONE	1080	1	3.21818s	119969	0.000687883
ITL	CG	Jacobi	1080	1	0.216848s	3.8619e+10	221.435
Trilinos	CG	Jacobi	4	3	0.001031s	1.90169e+08	1.0904
PETSc	CG	Jacobi	2	6	0.005147s	1.90174e+08	1.09043
ITL	BICGstab	Jacobi	1080	1	0.425011s	6.06008e+61	3.47475e+53
Trilinos	BICGstab	Jacobi	1080	1	0.172153s	7.61446e+81	4.366e+73
PETSc	BICGstab	Jacobi	10	7	0.009196s	3.79207e+13	217431
Trilinos	GMRES	Jacobi	235	4	0.16251s	9047.93	5.18793e-05
PETSc	GMRES	Jacobi	1080	1	3.21645s	7.30407e+10	418.803
ITL	CG	ILU	1080	1	0.503734s	9.10391e+26	5.22003e+18
Hypre	CG	ILU	1080	1	0.166851s	1.34245e+09	7.69738
Trilinos	CG	ILU	1	3	0.028584s	1.74403e+08	1
PETSc	CG	ILU	1	6	0.00672s	1.74403e+08	1
ITL	BICGstab	ILU	11	0	0.056491s	4.51981e-05	2.59158e-13
Hypre	BICGstab	ILU	609	0	0.184664s	0.000165714	9.50178e-13
Trilinos	BICGstab	ILU	8	0	0.032393s	3.48335e-05	1.99729e-13
QQQ	BICGstab	ILU	1080	1	26.7222s	2.82451e+13	161952
PETSc	BICGstab	ILU	10	0	0.015809s	3.57623e-05	2.05055e-13
Hypre	GMRES	ILU	252	0	0.097379s	0.000101986	5.84769e-13
Trilinos	GMRES	ILU	14	0	0.032646s	1.60593e-05	9.20812e-14
QQQ	GMRES	ILU	1	0	0.040773s	4.01676e+15	2.30314e+07
PETSc	GMRES	ILU	18	0	0.021028s	7.78913e-05	4.46616e-13

Table A.16: Sherman2 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	5005	1	1.90615s	0.0771292	0.000331295
Hypre	CG	NONE	5005	1	0.770484s	0.0428037	0.000183856
Trilinos	CG	NONE	5005	1	0.778495s	0.0428973	0.000184258
PETSc	CG	NONE	5005	1	1.61091s	0.0633232	0.000271994
ITL	BICGstab	NONE	5005	1	3.3448s	1.81115e-05	7.77946e-08
Hypre	BICGstab	NONE	5005	1	1.44806s	7.06643e-05	3.03526e-07
Trilinos	BICGstab	NONE	5005	1	1.45572s	0.000102011	4.38169e-07
QQQ	BICGstab	NONE	5005	1	4.47374s	0.00706575	3.03497e-05
PETSc	BICGstab	NONE	5005	1	3.06883s	0.0341576	0.000146718
Hypre	GMRES	NONE	5005	1	8.64103s	2.43545e-10	1.0461e-12
Trilinos	GMRES	NONE	785	2	11.1773s	6.71055e-09	2.8824e-11
QQQ	GMRES	NONE	1320	0	31.4356s	1.87887e-09	8.07034e-12
PETSc	GMRES	NONE	1566	0	19.0528s	2.65669e-10	1.14114e-12
ITL	CG	Jacobi	923	0	0.333632s	2.57125e-09	1.10444e-11
Trilinos	CG	Jacobi	922	2	0.159619s	2.9281e-09	1.25772e-11
PETSc	CG	Jacobi	891	0	0.288987s	2.89222e-09	1.2423e-11
ITL	BICGstab	Jacobi	524	0	0.365448s	1.84798e-09	7.93769e-12
Trilinos	BICGstab	Jacobi	490	3	0.160675s	3.23295e-09	1.38866e-11
PETSc	BICGstab	Jacobi	458	0	0.3237s	1.63235e-08	7.01147e-11
Trilinos	GMRES	Jacobi	482	2	4.13227s	2.0575e-09	8.83765e-12
PETSc	GMRES	Jacobi	3212	0	42.5148s	2.16487e-08	9.29882e-11
ITL	CG	ILU	172	0	0.150296s	1.52056e-09	6.53129e-12
Hypre	CG	ILU	5005	1	2.17707s	1070.29	4.59724
Trilinos	CG	ILU	5005	1	2.12393s	4.73887e-09	2.0355e-11
PETSc	CG	ILU	160	0	0.104792s	1.32474e-09	5.69018e-12
ITL	BICGstab	ILU	101	0	0.165563s	9.28012e-10	3.98611e-12
Hypre	BICGstab	ILU	109	0	0.104683s	1.90999e-10	8.20401e-13
Trilinos	BICGstab	ILU	20	2	0.049299s	6.84653e-10	2.94081e-12
QQQ	BICGstab	ILU	36	0	2.80833s	5.12919e-09	2.20315e-11
PETSc	BICGstab	ILU	117	0	0.153035s	8.09647e-10	3.4777e-12
Hypre	GMRES	ILU	125	0	0.216134s	1.99124e-10	8.55301e-13
Trilinos	GMRES	ILU	29	0	0.056956s	1.98647e-10	8.53252e-13
QQQ	GMRES	ILU	2	0	0.105566s	9.56087e+09	4.10671e+07
PETSc	GMRES	ILU	877	0	10.9977s	7.74872e-08	3.32833e-10

Table A.17: Sherman3 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	1899	5	0.570678s	nan	nan
Hypre	CG	NONE	3312	1	0.416329s	1.72522e+23	2.77915e+21
Trilinos	CG	NONE	1	3	0.000538s	62.0774	1
PETSc	CG	NONE	1	8	0.010466s	62.0774	1
ITL	BICGstab	NONE	198	5	0.113038s	126.741	2.04167
Hypre	BICGstab	NONE	3312	1	0.783425s	2.75118e-06	4.43186e-08
Trilinos	BICGstab	NONE	1113	3	0.247757s	16.3764	0.263807
QQQ	BICGstab	NONE	3312	1	3.9362s	3.74627e-05	6.03484e-07
PETSc	BICGstab	NONE	3312	1	1.87454s	3.03477e-08	4.88869e-10
Hypre	GMRES	NONE	1929	0	7.42302s	5.86559e-11	9.44884e-13
Trilinos	GMRES	NONE	1820	0	20.5701s	2.38273e-11	3.83833e-13
QQQ	GMRES	NONE	1667	0	28.0344s	6.22901e-10	1.00343e-11
PETSc	GMRES	NONE	3312	1	28.8769s	0.0178954	0.000288276
ITL	CG	Jacobi	517	5	0.164849s	nan	nan
Trilinos	CG	Jacobi	1	3	0.001135s	62.0774	1
PETSc	CG	Jacobi	1	8	0.010081s	62.0774	1
ITL	BICGstab	Jacobi	179	0	0.107159s	3.4952e-10	5.6304e-12
Trilinos	BICGstab	Jacobi	180	2	0.044537s	8.30062e-10	1.33714e-11
PETSc	BICGstab	Jacobi	152	0	0.097521s	1.50881e-09	2.43053e-11
Trilinos	GMRES	Jacobi	165	2	0.261708s	3.75975e-10	6.05655e-12
PETSc	GMRES	Jacobi	1075	0	9.37321s	1.47833e-09	2.38143e-11
ITL	CG	ILU	3312	1	2.10788s	19871.1	320.102
Hypre	CG	ILU	3312	1	0.97036s	5.94224e+19	9.57231e+17
Trilinos	CG	ILU	1	3	0.026437s	62.0774	1
PETSc	CG	ILU	2	6	0.015656s	167.044	2.6909
ITL	BICGstab	ILU	30	0	0.066178s	1.28677e-10	2.07285e-12
Hypre	BICGstab	ILU	72	0	0.050053s	5.07197e-11	8.1704e-13
Trilinos	BICGstab	ILU	15	2	0.036147s	1.18253e-10	1.90492e-12
QQQ	BICGstab	ILU	29	0	2.23036s	1.1653e-10	1.87718e-12
PETSc	BICGstab	ILU	31	0	0.046716s	4.15241e-10	6.68909e-12
Hypre	GMRES	ILU	93	0	0.085033s	5.50109e-11	8.86166e-13
Trilinos	GMRES	ILU	20	0	0.03647s	4.62161e-11	7.44492e-13
QQQ	GMRES	ILU	10	0	0.621464s	8.82663e-09	1.42188e-10
PETSc	GMRES	ILU	41	0	0.054181s	2.10242e-09	3.38677e-11

Table A.18: Sherman5 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	236	1	0.015485s	2739.91	379.126
Hypre	CG	NONE	236	1	0.009264s	2739.91	379.126
Trilinos	CG	NONE	236	1	0.008046s	2739.91	379.126
PETSc	CG	NONE	236	1	0.024425s	2739.91	379.126
ITL	BICGstab	NONE	236	1	0.027717s	0.108113	0.0149598
Hypre	BICGstab	NONE	236	1	0.011747s	0.57305	0.0792938
Trilinos	BICGstab	NONE	236	1	0.012032s	0.712718	0.0986201
QQQ	BICGstab	NONE	236	1	0.02732s	0.727792	0.100706
PETSc	BICGstab	NONE	236	1	0.023994s	1.09461	0.151463
Hypre	GMRES	NONE	203	0	0.027475s	2.30884e-12	3.19479e-13
Trilinos	GMRES	NONE	203	0	0.033416s	2.49509e-12	3.45249e-13
QQQ	GMRES	NONE	203	0	0.074245s	3.65898e-12	5.06299e-13
PETSc	GMRES	NONE	236	1	0.069171s	0.763829	0.105692
ITL	CG	Jacobi	2	5	0.000976s	nan	nan
Trilinos	CG	Jacobi	236	1	0.009394s	1473.95	203.953
PETSc	CG	Jacobi	236	1	0.016473s	80555.5	11146.6
ITL	BICGstab	Jacobi	1	5	0.000871s	nan	nan
Trilinos	BICGstab	Jacobi	236	1	0.012502s	11.0212	1.52502
PETSc	BICGstab	Jacobi	236	1	0.0257s	0.430728	0.0596006
Trilinos	GMRES	Jacobi	72	4	0.006319s	4.15136	0.574431
PETSc	GMRES	Jacobi	236	1	0.06867s	0.0446516	0.00617852
Hypre	CG	ILU	236	1	0.015513s	617886	85497.9
Trilinos	CG	ILU	6	3	0.004147s	1.31075e+47	1.81371e+46
PETSc	CG	ILU	1	6	0.002842s	7.22691	1
Hypre	BICGstab	ILU	236	1	0.027159s	178.613	24.7149
Trilinos	BICGstab	ILU	236	1	0.034673s	2197.5	304.071
QQQ	BICGstab	ILU	27	0	0.025025s	4.5019e-12	6.22935e-13
PETSc	BICGstab	ILU	30	0	0.008727s	7.34779e-12	1.01673e-12
Hypre	GMRES	ILU	236	1	0.038846s	5.11491e-09	7.07758e-10
Trilinos	GMRES	ILU	8	4	0.004467s	7.78616	1.07738
QQQ	GMRES	ILU	12	0	0.087214s	2.11885e-11	2.93189e-12
PETSc	GMRES	ILU	42	0	0.026555s	1.23933e-10	1.71488e-11

Table A.19: E05r0100 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	4241	1	3.92968s	6.70081e+06	230868
Hypre	CG	NONE	4241	1	1.60714s	6.70081e+06	230868
Trilinos	CG	NONE	4241	1	1.5396s	6.70081e+06	230868
PETSc	CG	NONE	269	7	0.231691s	290896	10022.4
ITL	BICGstab	NONE	139	5	0.255446s	167.285	5.76361
Hypre	BICGstab	NONE	4241	1	3.15319s	175.065	6.03166
Trilinos	BICGstab	NONE	59	3	0.042719s	184.725	6.36446
QQQ	BICGstab	NONE	4241	1	9.75308s	163.881	5.64631
PETSc	BICGstab	NONE	19	7	0.050993s	1.0405e+06	35849.3
Hypre	GMRES	NONE	4241	1	27.7054s	15.9229	0.548603
Trilinos	GMRES	NONE	4241	1	64.176s	15.9229	0.548603
QQQ	GMRES	NONE	4241	1	101.779s	23.0544	0.794309
PETSc	GMRES	NONE	4241	1	50.5551s	15.9229	0.548603
ITL	CG	Jacobi	2	5	0.014851s	nan	nan
Trilinos	CG	Jacobi	4241	1	1.68298s	1.27595e+06	43961.3
PETSc	CG	Jacobi	138	7	0.124222s	92277	3179.29
ITL	BICGstab	Jacobi	1	5	0.013411s	nan	nan
Trilinos	BICGstab	Jacobi	1667	3	1.25287s	1642.31	56.5839
PETSc	BICGstab	Jacobi	4241	1	6.70229s	233.808	8.05556
Trilinos	GMRES	Jacobi	4241	1	65.8724s	15.9288	0.548806
PETSc	GMRES	Jacobi	4241	1	50.2777s	16.4152	0.565565
Hypre	CG	ILU	4241	1	4.28862s	8.04596e+16	2.77213e+15
Trilinos	CG	ILU	1	3	0.302958s	nan	nan
PETSc	CG	ILU	1	6	0.036216s	29.0244	1
Hypre	BICGstab	ILU	4241	1	8.0576s	29.0218	0.999911
Trilinos	BICGstab	ILU	1	3	0.330591s	nan	nan
QQQ	BICGstab	ILU	45	0	6.63505s	1.06817e-09	3.68024e-11
PETSc	BICGstab	ILU	382	7	1.26225s	3.95147e+12	1.36143e+11
Hypre	GMRES	ILU	4241	1	33.2303s	4.86836e+13	1.67733e+12
Trilinos	GMRES	ILU	1	3	0.236314s	29.0244	1
QQQ	GMRES	ILU	34	0	4.60562s	1.08444e-07	3.73629e-09
PETSc	GMRES	ILU	4241	1	54.4999s	1.65029e+06	56858.5

Table A.20: E20r5000 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	2395	1	0.473019s	8.06422e-13	0.00813262
Hypre	CG	NONE	2395	1	0.21942s	8.06422e-13	0.00813262
Trilinos	CG	NONE	2395	1	0.228695s	8.06422e-13	0.00813262
PETSc	CG	NONE	2395	1	0.531212s	8.06422e-13	0.00813262
ITL	BICGstab	NONE	553	0	0.21512s	8.28338e-23	8.35364e-13
Hypre	BICGstab	NONE	580	0	0.099845s	2.85412e-23	2.87832e-13
Trilinos	BICGstab	NONE	257	3	0.050144s	2.35238e-17	2.37233e-07
QQQ	BICGstab	NONE	715	0	15.8321s	8.15104e-23	8.22018e-13
PETSc	BICGstab	NONE	632	0	0.244964s	9.84551e-23	9.92901e-13
Hypre	GMRES	NONE	458	0	0.775713s	9.90202e-23	9.986e-13
Trilinos	GMRES	NONE	459	0	1.6528s	9.57723e-23	9.65846e-13
QQQ	GMRES	NONE	461	0	2.97669s	8.97193e-23	9.04803e-13
PETSc	GMRES	NONE	1090	0	6.82447s	9.65895e-23	9.74087e-13
ITL	CG	Jacobi	2395	1	0.516277s	6.26454e-12	0.0631767
Trilinos	CG	Jacobi	2395	1	0.248544s	6.26454e-12	0.0631767
PETSc	CG	Jacobi	2395	1	0.450245s	6.26454e-12	0.0631767
ITL	BICGstab	Jacobi	502	0	0.205913s	9.19034e-23	9.26828e-13
Trilinos	BICGstab	Jacobi	63	3	0.013401s	9.2773e-14	0.000935599
PETSc	BICGstab	Jacobi	949	0	0.387008s	9.33245e-23	9.4116e-13
Trilinos	GMRES	Jacobi	172	0	0.195794s	8.87523e-23	8.95051e-13
PETSc	GMRES	Jacobi	2395	1	14.5466s	3.29967e-15	3.32766e-05
ITL	CG	ILU	2395	1	1.03061s	8.36297e-13	0.0084339
Hypre	CG	ILU	2395	1	0.615881s	1.30097e-10	1.31201
Trilinos	CG	ILU	5	0	0.013115s	1.86936e-24	1.88522e-14
PETSc	CG	ILU	2395	1	0.780597s	8.36297e-13	0.0084339
ITL	BICGstab	ILU	159	0	0.14873s	8.7275e-23	8.80152e-13
Hypre	BICGstab	ILU	145	0	0.079206s	7.95843e-23	8.02593e-13
Trilinos	BICGstab	ILU	2	0	0.012627s	9.89167e-23	9.97557e-13
QQQ	BICGstab	ILU	6	0	15.5565s	2.51039e-23	2.53168e-13
PETSc	BICGstab	ILU	221	0	0.165501s	5.70359e-23	5.75197e-13
Hypre	GMRES	ILU	112	0	0.088412s	9.48827e-23	9.56874e-13
Trilinos	GMRES	ILU	4	0	0.013186s	2.05001e-23	2.0674e-13
QQQ	GMRES	ILU	5	0	0.333652s	4.00445e-24	4.03841e-14
PETSc	GMRES	ILU	1057	0	6.97491s	6.92011e-23	6.9788e-13

Table A.21: Add20 test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	17758	1	24.6101s	7.70234e-12	0.365786
Hypre	CG	NONE	17758	1	10.8274s	7.70234e-12	0.365786
Trilinos	CG	NONE	17758	1	12.0438s	7.70234e-12	0.365786
PETSc	CG	NONE	17758	1	21.8778s	7.70234e-12	0.365786
ITL	BICGstab	NONE	2038	0	5.53111s	1.98111e-23	9.40835e-13
Hypre	BICGstab	NONE	2271	0	2.6545s	1.99094e-23	9.45501e-13
Trilinos	BICGstab	NONE	164	3	0.230141s	1.34628e-14	0.000639352
QQQ	BICGstab	NONE	2458	0	6.72217s	1.88561e-23	8.9548e-13
PETSc	BICGstab	NONE	1860	0	4.93828s	1.87908e-23	8.92379e-13
Hypre	GMRES	NONE	914	0	22.6791s	2.07233e-23	9.84156e-13
Trilinos	GMRES	NONE	914	0	54.1267s	2.08688e-23	9.91063e-13
QQQ	GMRES	NONE	1031	0	101.167s	2.0618e-23	9.79156e-13
PETSc	GMRES	NONE	1465	0	59.6993s	2.07945e-23	9.87536e-13
ITL	CG	Jacobi	17758	1	24.9021s	8.70321e-11	4.13318
Trilinos	CG	Jacobi	17758	1	12.9512s	8.70321e-11	4.13318
PETSc	CG	Jacobi	17758	1	22.5687s	8.70321e-11	4.13318
ITL	BICGstab	Jacobi	553	5	1.60083s	1.97465e-17	9.37767e-07
Trilinos	BICGstab	Jacobi	18	3	0.031162s	1.33868e-13	0.00635744
PETSc	BICGstab	Jacobi	755	0	2.19001s	3.43885e-23	1.63312e-12
Trilinos	GMRES	Jacobi	286	0	5.26988s	1.87699e-23	8.91389e-13
PETSc	GMRES	Jacobi	11535	0	550.505s	2.69647e-21	1.28056e-10
ITL	CG	ILU	17758	1	54.5601s	1.50292e-10	7.1374
Hypre	CG	ILU	17758	1	35.7351s	7.98189e-13	0.0379062
Trilinos	CG	ILU	8	0	0.324985s	2.9158e-24	1.38472e-13
PETSc	CG	ILU	17758	1	39.8092s	1.50292e-10	7.1374
ITL	BICGstab	ILU	345	0	2.28419s	1.71749e-23	8.15638e-13
Hypre	BICGstab	ILU	695	0	2.90697s	1.68983e-23	8.02505e-13
Trilinos	BICGstab	ILU	4	0	0.32395s	3.54429e-24	1.68319e-13
QQQ	BICGstab	ILU	9	0	0.735679s	5.08324e-24	2.41404e-13
PETSc	BICGstab	ILU	387	0	1.97006s	4.79313e-23	2.27627e-12
Hypre	GMRES	ILU	207	0	1.66554s	2.10036e-23	9.97468e-13
Trilinos	GMRES	ILU	7	0	0.32606s	3.08071e-24	1.46304e-13
QQQ	GMRES	ILU	8	0	4.76651s	3.03273e-23	1.44025e-12
PETSc	GMRES	ILU	2124	0	100.289s	7.53823e-23	3.57992e-12

Table A.22: Memplus test results

package	solver	precond	iterations	status	solving time	absolute residual	rel residual
ITL	CG	NONE	5	1	0.000112s	0.982146	0.0220387
Hypre	CG	NONE	5	1	0.0001s	0.982146	0.0220387
Trilinos	CG	NONE	2	3	0.030383s	3.07395	0.0689775
PETSc	CG	NONE	2	8	0.009716s	3.07395	0.0689775
ITL	BICGstab	NONE	5	1	0.000442s	1.20431e-09	2.70239e-11
Hypre	BICGstab	NONE	5	1	0.000239s	1.25082e-09	2.80676e-11
Trilinos	BICGstab	NONE	5	1	0.000474s	1.40992e-09	3.16377e-11
QQQ	BICGstab	NONE	5	1	0.040176s	2.67244e-09	5.99679e-11
PETSc	BICGstab	NONE	5	1	0.000695s	1.39461e-09	3.12941e-11
Hypre	GMRES	NONE	5	0	0.006386s	8.37906e-15	1.88021e-16
Trilinos	GMRES	NONE	5	0	0.023519s	8.1886e-15	1.83747e-16
QQQ	GMRES	NONE	5	0	0.04472s	9.56597e-15	2.14654e-16
PETSc	GMRES	NONE	5	0	0.018645s	7.01436e-14	1.57398e-15
ITL	CG	Jacobi	2	5	0.000291s	nan	nan
Trilinos	CG	Jacobi	2	3	0.000458s	13.8536	0.310865
PETSc	CG	Jacobi	2	8	0.000617s	4.86385	0.109142
ITL	BICGstab	Jacobi	1	5	0.000272s	nan	nan
Trilinos	BICGstab	Jacobi	5	1	0.000483s	3.65538	0.0820243
PETSc	BICGstab	Jacobi	5	1	0.000727s	2.03328e-05	4.56255e-07
Trilinos	GMRES	Jacobi	4	4	0.000505s	3.42932	0.0769517
PETSc	GMRES	Jacobi	5	0	0.017973s	8.8185e-13	1.97881e-14
Hypre	CG	ILU	1	0	0.000197s	0	0
Trilinos	CG	ILU	4	0	0.000581s	5.58757e-12	1.25382e-13
PETSc	CG	ILU	1	6	0.001527s	44.5646	1
Hypre	BICGstab	ILU	0	3	0.000177s	nan	nan
Trilinos	BICGstab	ILU	2	0	0.000461s	3.62256e-12	8.12879e-14
QQQ	BICGstab	ILU	3	0	0.040145s	1.75032e-12	3.92761e-14
PETSc	BICGstab	ILU	5	1	0.001595s	0.335308	0.0075241
Hypre	GMRES	ILU	1	0	0.004303s	7.16072e-15	1.60682e-16
Trilinos	GMRES	ILU	4	2	0.000933s	0.00125206	2.80953e-05
QQQ	GMRES	ILU	1	0	0.057151s	8.82118e-14	1.97942e-15
PETSc	GMRES	ILU	5	1	0.01966s	3.61698e-06	8.11628e-08

Table A.23: Saddle point matrix results

Appendix B

Sparse Matrix Formats

B.1 Sparse Matrix Formats

B.1.1 Compressed Sparse Row Matrix Format (CSR)

The CSR (Compressed Sparse[1] Row) format stores a matrix in three arrays: *values*, *columns* and *rowIndex*. The *values* array stores the nonzero elements of the sparse matrix [16]. The I-th element of the *columns* array contains the column indices of the I-th value of the *values* array. The J-th element of the *rowIndex* array contains the index of the element in the *values* array, which is the first nonzero in the row j of the matrix. The number of elements in the *values* and *columns* arrays is equal to the number of nonzero elements in the matrix. The *rowIndex* array contains the dimension of the matrix plus one elements. The last element of *rowIndex*is the number of elements in *values* plus one. The following example illustrates the usage:

$$A = \begin{bmatrix} 1 & 0 & 5 & 0 \\ 0 & 4 & 6 & 0 \\ 1 & 0 & 0 & 2 \\ 1 & 0 & 3 & 1 \end{bmatrix}$$

The Matrix A in CSR format would be stored like this (index starting at 0):

$values = [1, 5, 4, 6, 1, 2, 1, 3, 1]$
$columns = [0, 2, 1, 2, 0, 3, 0, 2, 3]$
$rowIndex = [0, 2, 4, 6, 10]$

B.1.2 Modified Compressed Sparse Row Matrix Format (MCSR)

The MCSR (Modified Compressed Sparse Row) format stores the diagonal of the matrix explicitly [16]. There are two arrays used: *values* and index. Both arrays are split into a lower and an upper part. The lower part of the *values* array contains the main diagonal elements of the matrix. Thus the number of elements stored equals the dimension of the matrix. If the diagonal contains zeros they must be saved as well. The element with the index dimension plus one is unused in the *values* array. The upper part of the *values* array contains the off diagonal non zeros of the matrix stored row wise. The J-th element from the lower part of the *index* array contains the index of the *values* array with the first off diagonal element of the J-th row. The element dimension plus one holds the number of elements stored in the two arrays.

[1] In contrast to a dense matrix, a sparse matrix stores the nonzero elements only.

The upper part of the *index* array holds the column indices of the elements stored at the same index in the *values* array. The matrix A mentioned in Section B.1.1 would be stored like this (index starting at 0):

$values = [1, 4, 0, 1, *, 5, 6, 1, 2, 1, 3]$
$index = [5, 6, 7, 9, 11, 2, 2, 0, 3, 0, 2]$

Bibliography

[1] A. Meister, *Numerik linearer Gleichungssysteme*, 2nd ed. Vieweg, 2005.

[2] G. Fischer, *Lineare Algebra*, 16th ed. Vieweg, 2008.

[3] N. Koeckler and H. R. Schwarz, *Numerische Mathematik*, 5th ed. Teubner, 2004.

[4] H. Dirschmid, *Skriptum aus Matritzennumerik*, Technische Universität Wien, 1998.

[5] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. SIAM, 2003.

[6] J. Stoer and R. Bulirsch, *Numerische Mathematik 2*, 5th ed. Springer, 2005.

[7] M. Schinnerl, J. Schöberl, M. Kaltenbacher, and R. Lerch, *Multigrid Methods for the 3D Simulation of Nonlinear Magneto-Mechanical Systems*. IEEE Transactions Magnetics, 2002, no. Vol 38(3), pp 1497-1511.

[8] A. Toselli and O. Widlund, *Domain Decomposition Methods - Algorithms and Theory*, 1st ed. Springer, 2005.

[9] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI - The Complete Reference*, 2nd ed. MIT Press, 1998.

[10] M. A. Heroux, J. M. Willenbring, and R. Heaphy, *Trilinos Developers Guide*, 2003, no. SAND2003-1898.

[11] M. Sala, M. A. Heroux, and D. M. Day, *Trilinos Tutorial*, 2007, no. SAND2004-2189.

[12] L. S. Blackforda, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, *An Updated Set of Basic Linear Algebra Subprograms (BLAS)*, 2002.

[13] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient Management of Parallelism in Object Oriented Numerical Software Libraries*. Birkhäuser Press, 1997.

[14] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc Users Manual*, 2008, no. ANL-95/11 - Revision 3.0.0.

[15] P. Gottschling and D. Lindbo, *Generic Compressed Sparse Matrix Insertion: Algorithms and Implementations in MTL4 and FEniCS*, 2009.

[16] S. Wagner, *Small-Signal Device and Circuit Simulation*. Dissertation, TU Wien, 2005, http://www.iue.tuwien.ac.at/phd/wagner/.

[17] R. Falgout, A. Cleary, J. Jones, E. Chow, V. Henson, C. Baldwin, P. Brown, P. Vassilevski, and U. M. Yang, *Hypre User's Manual*, 2008.

[18] *Matrix Market*, http://math.nist.gov/MatrixMarket.

[19] D. Braess, *Finite Elemente. Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*, 4th ed. Springer-Verlag, 2003.