Homework I
Iterative methods for sparse matrices

Moritz Wolter

October 22, 2015

1 Theoretical background

1.1 GMRES

The GMRES abbreviates generalized minimal residuals. It's idea is to solve $Ax = b$ by using a vector $x_n \in \mathcal{K}_n$ that minimizes the residual $r_n = b - Ax_n$ [1]. In general the process is implemented using Arnoldi-iterations to compute the Krylov subspace and the Hessenberg representation of the original problem. The least square problem is then solved by coupling a QR decomposition into the Arnoldi iterations.

1.2 BICGSTAB

The Bi-CGSTAB, method is a modified version of the biconjugate gradient method. The biconjugate gradient method uses recurrences which only require data from the last iteration, which keeps memory requirements under control. The better memory usage comes at the price of lost monotonic convergence. The original BCG algorithm does not converge nicely. The main added benefit of the Bi-CGSTAB algorithm is better convergence behavior. A table containing the most important methods is given below[2].

1.3 Preconditioning

The convergence of iterative methods depends on the properties of the matrix A, such as the position of it's eigenvalues or normality. It the spectrum is spread out or A is too far from normal iterative schemes might fail to converge. To fix such problems instead of solving:

$$Ax = b. \tag{1}$$

[1]Numerical Linear Algebra, Trefethen, Bau page 266

[2] as found in numerical linear algebra, Trefethen, Bau page 235.

	$Ax = b$	$Ax = \lambda x$
$A = A*$ (hermitian)	CG	Lanczos
$A \neq A*$ (non-hermitian)	GMRES, CGN, BCG et al.	Arnoldi

Table 1: Krylov subspace methods

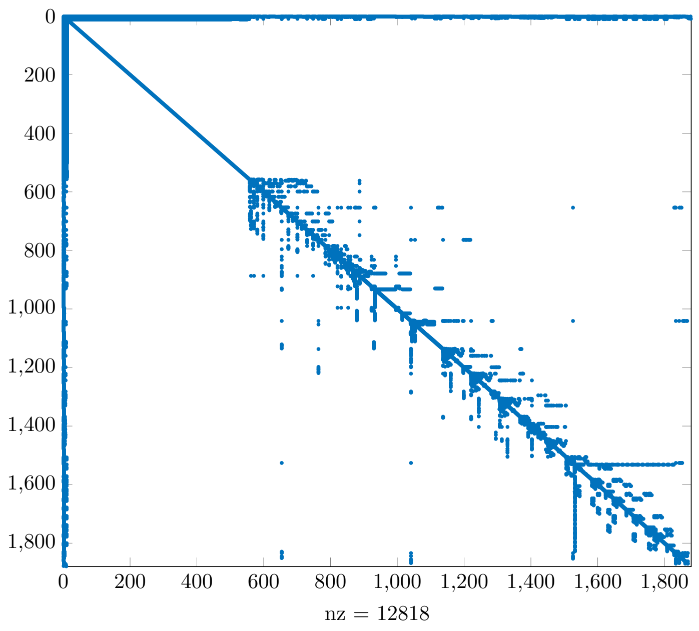

Figure 1: Sparsity representation of the `rajat12` matrix.

A matrix M^{-1}, the preconditioner is multiplied from the left to both sides of the equation [3]:

$$M^{-1}Ax = M^{-1}b. \tag{2}$$

The resulting system $M^{-1}A = B$ should be close to normal ($\bar{B}^T B \approx B\bar{B}^T$) and have a clustured spectrum.

2 Results

2.1 rajat12

Figure 1 shows the `rajat12` circuit matrix. This matrix is real and the matlab routine `issymmetric` finds it to be not symmetric. From a complex point of view it can thus be considered non-hermitian. However a closer look at Figure 1 reveals that the matrix is very close to being symmetric. In a first series of experiments the iterative GMRES method is used. The level of the incomplete LU-factorization, which is used for preconditioning is varied from between zero and one. Furthermore the amount of available vectors is increased successively from zero to one hundred.
Results for the time measurements are shown in Figure 2. Computations where run on a ThinkPad equipped with `8 GB` of ram and an i7-4800mq processor. The data shows a significant increase in computation time, for higher numbers of available vectors, with a zero level of fill. The same observation does not hold with a level of fill of one, here computations oscillate around an expected value of approximately two seconds.
Figure 3 shows convergence of the residuals. Here it is important to note, that sufficient accuracy is reached using 10 Vectors within $\approx$ 230 iterations for ILU level zero. It takes $\approx$ 20 when the level of fill is set to one, due to improved conditioning of the problem.

[3]Numerical linear algebra, Trefethen, Bau page 313.

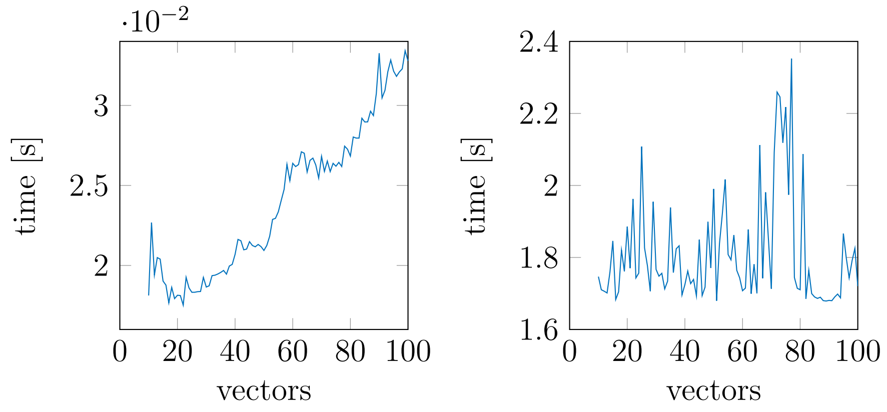

Figure 2: CPU-Time of running `./ilu --method gmres --file rajat12.mtx --nvectors $vectors --tolerance 1.e-10` with `--ilu-level 0`(left) and `--ilu-level 1`(right) the value of the `$vectors` variable is shown on the x axis.

Curiously the decrease in iterations does not lead to on overall reduction of computation time. Probably in this case single iterations are much faster with zero level of fill, which would account for the timing difference. An overall increase of computation time can be observed for more vectors with in the zero level of fill setting. This is probably due to the fact that the overhead caused by the additional vectors becomes significant only when the amount of total iterations is large. Interestingly, when more vectors are used fewer iterations are required until convergence is reached. Large amounts of vectors are probably beneficial in settings, where larger matrices are used. Memory requirements are shown in table 2. From the data it can be concluded that the higher ilu level does is neither advantageous in terms of memory consumption or computing time when the relatively small circuit matrix is considered.
Figure 4, shows the initial and preconditioned spectra. The normalizing effect of both preconditioners is clearly visible. Following the rule of thumb[4]:
"A preconditioner M is good if M^{-1}A is not too far from normal and its eigenvalues are clustered."
Both preconditioners work but the higher level of fill gives more clustering, therefore it is better in this case. The direct method implemented in the `mumps` requires `5.08 MB` of ram and finishes within $0.004035s = 4.035 * 10^{-3}s$. It is thus faster then any iterative scheme tried above, with more then acceptable memory consumption. Storing some data on the hard drive using the `--ooc` option increases the computation time to $0.007442s$. In this cases this is clearly not necessary, as the memory needed is available on almost any modern computer.

2.2 lhr01

Figure 5 shows the sparsity pattern of the `lhr01` light hydrocarbon recovery matrix. This matrix is real and not symmetric. From a complex point of view it again can be considered non-hermitian. However this matrix is a lot less symmetric then the circuit equations that have been explored earlier. The iterative methods implemented in the provided executable fail.

[4]Numerical linear algebra, Trefethen, Bau, page 314

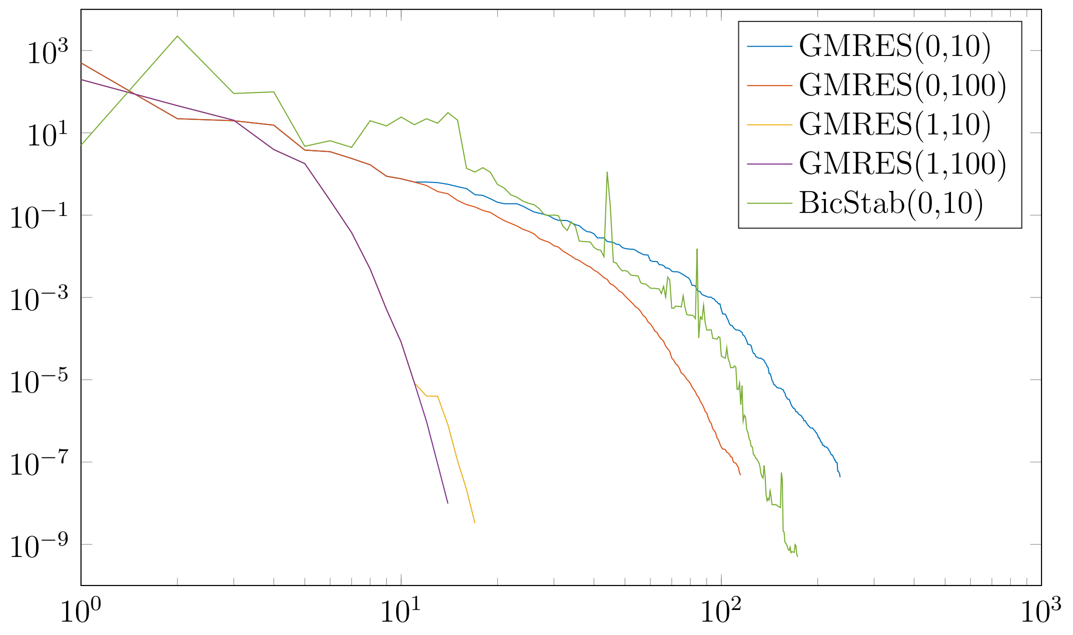

Figure 3: GMRES convergence with the rajat12 matrix with ILu level 0 and 1.

#vectors	ilu level of fill	memory	relative error
10	1	`155.9 MB`	$1.13517 \cdot 10^{-10}$
100	1	`157.3 MB`	$2.4985 \cdot 10^{-10}$
10	0	`3.1 MB`	$8.23038 \cdot 10^{-08}$
100	0	`4.35 MB`	$5.64681 \cdot 10^{-08}$

Table 2: Memory requirements of GMRES when run on the rajat12 matrix.

#vectors	ilu level of fill	memory	CPU time	relative error
10	1	`155.8 MB`	1.67045 s	$6.55193 \cdot 10^{-11}$
100	1	`155.823 MB`	1.94236 s	$6.55193 \cdot 10^{-11}$
10	0	`3.042 MB`	0.00815 s	$2.09145 \cdot 10^{-09}$
100	0	`3.042 MB`	0.01006 s	$2.09145 \cdot 10^{-09}$

Table 3: Memory and time requirements of bicgstab when run on the rajat12 matrix.

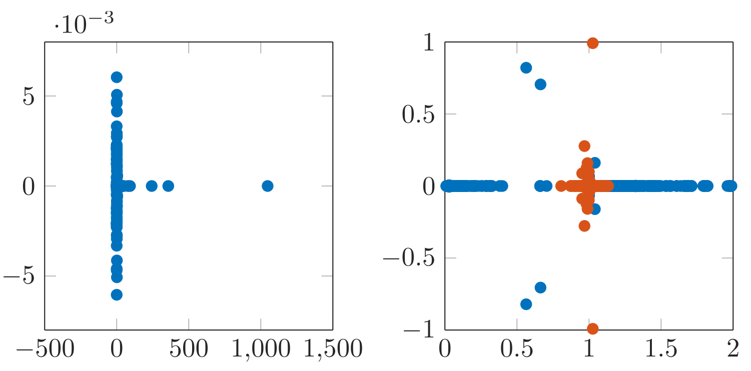

Figure 4: Original and preconditioned spectra of the rajat12 matrix. Preconditioned matrices are shown for 0 level of fill in blue and level 1 in red.

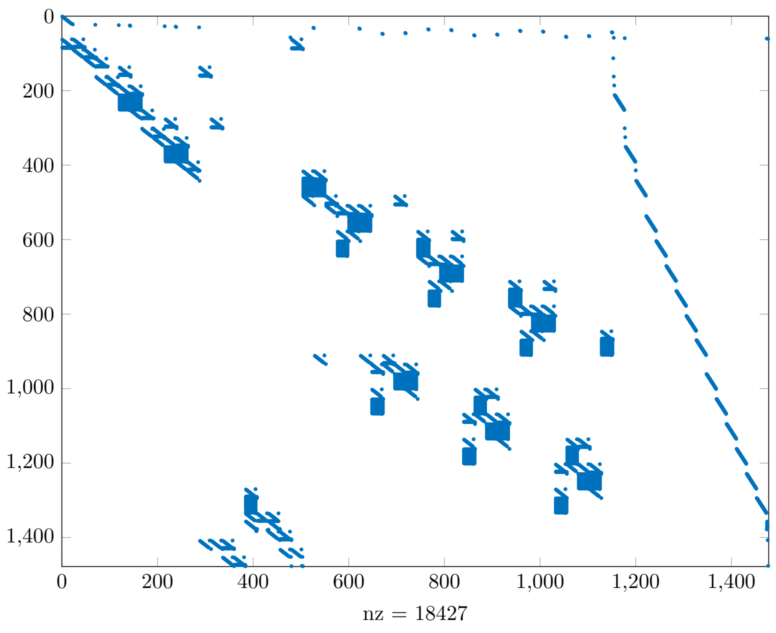

Figure 5: Sparsity representation of the `lhr01` matrix.

#vectors	ilu level of fill	memory	time	relative error
10	1	`9.473 GB`	61.5548s	20.8122
100	1	`9.561 GB`	81.6827s	1.15364
10	0	`1.037 GB`	48.2886s	91.2033
100	0	`1.125 GB`	66.3984s	3.18241

Table 4: Time, memory and accuracy test results of various GMRES computations, on the ship 003 matrix. The preconditioner's level of fill as well as the number of available vectors have been varied.

Printing the error message:
`ILU factorization failed on equation 21.`
In this case the Gaussian elimination process is unstable. Instabilities could arise if an entry in the L matrix is very large. If the L matrix is then multiplied with the corresponding U, the original A is not obtained [5]. In our case an obtained preconditioner would not be usable. The direct solver however is able to solve the problem in $0.009011s$ using `7.365 MB`. Interestingly enough the direct solver uses a LU-decomposition as well to solve the problem. The two problems are sightly different the incomplete LU seeks to solve $A \approx LU$. The direct methods attempts to solve $LUx = Pb$. These two problems will be implemented differently. Without looking at the code it is difficult to diagnose the problem.

2.3 ship003

Figure 7 shows the sparsity pattern of the `ship03` matrix. Results of iterative computations using the GMRES-algorithm are shown in table 4. The results here are similar to what was observed earlier. Decreasing the level of fill or vector number leads to faster

[5]Numerical Linear Algebra, Trefethen, Bau, page 163

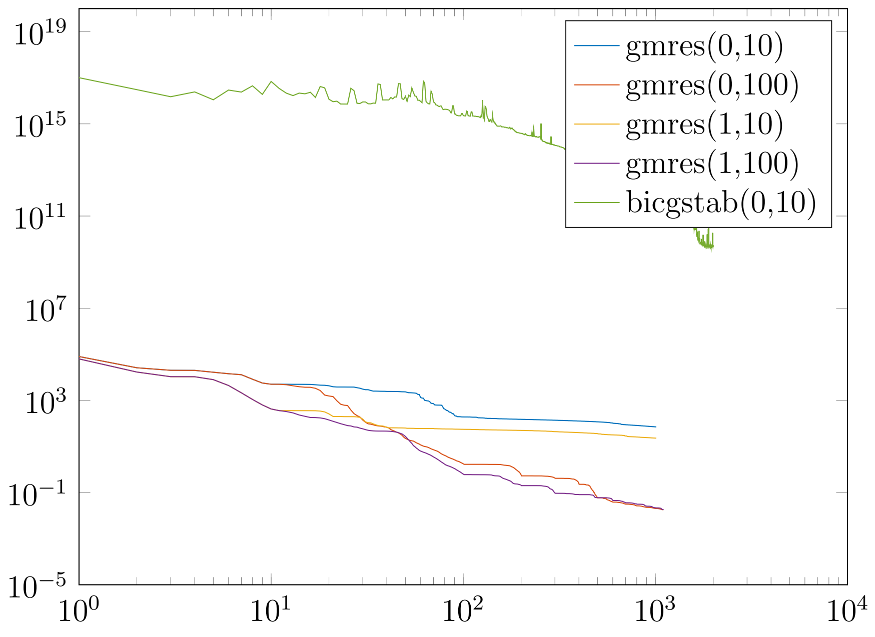

Figure 6: GMRES convergence with the ship matrix with ILu level 0 and 1.

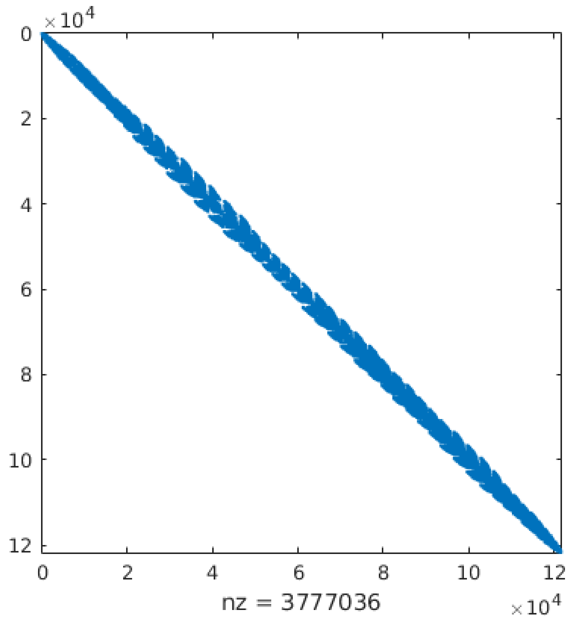

Figure 7: Visualisation of the sparsity pattern of the `ship003`-matrix

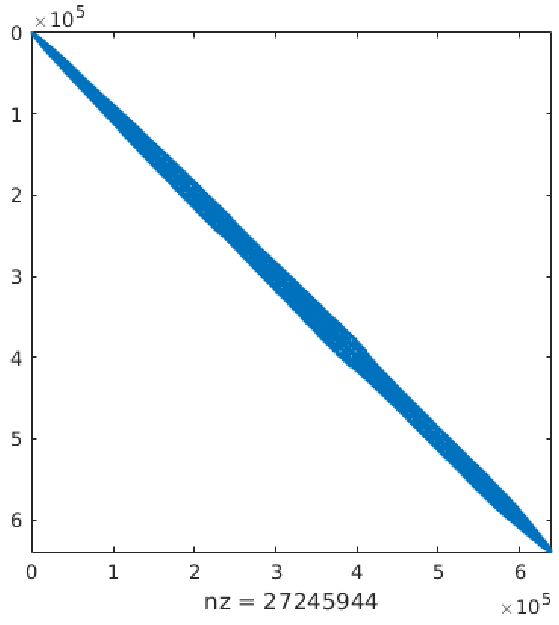

Figure 8: Visualisation of the sparsity pattern of the `fault639`-matrix

#vectors	ilu level of fill	memory	CPU time	relative error
10	1	??	269.7s	0.962255
100	1	??	465 s	0.82716
10	0	`3.723 GB`	181.4s	0.892187
100	0	`4.184 GB`	357.1s	0.0304293

Table 5: Memory requirements of GMRES when run on the fault matrix.

computation time and higher error. Convergence plots are given in Figure 6. Interestingly the GMRES algorithms only find the correct solution, if the methods have at least 100 vectors at their disposal. In contrast to the rajat matrix, which was experimented with earlier, the level of fill is not so important. Indicating, that both preconditioners are of similar quality. Unfortunately the matrix is too large for eigenvalue computations on the machine available, making further investigation impossible. The bicgstab algorithm does not converge, as the solution it computes is useless it's memory usage has not been measured. Again the direct method outperforms the iterative ones producing a solution with an error of $1.273 * 10^{-10}$ within $21.7s$ using `2.996 GB` of RAM.

2.4 Fault639

The biggest matrix considered in this series of experiments is the `fault639`-matarix. Table 5 and figure 9 contain more information on the performance of the iterative methods under consideration. Unfortunately `valgrind` crashed during memory measurements for the very memory intensive computations with iLU level of fill one. The bicgstab method again is far from convergence. However the `gmres` method run with zero level of fill iLU preconditioning and 100 vectors converges to a good solution. Running for approximately five minutes this iterative scheme performs much better then the direct methods, which took around 22 minutes to compute a solution with error of the magnitude 10^{-12} on eupen

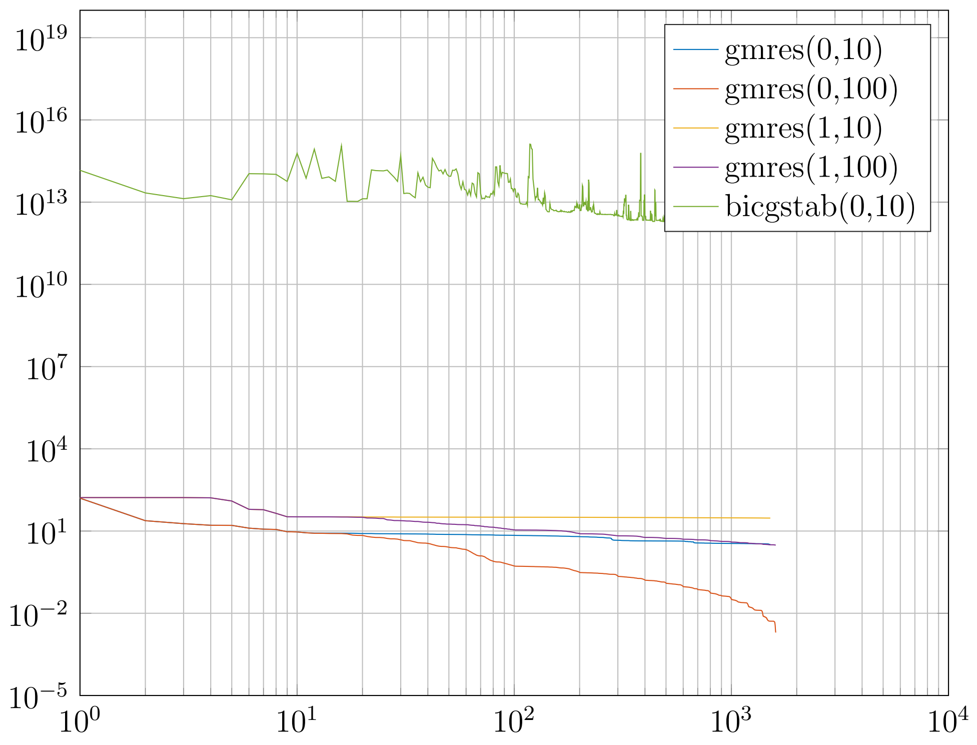

Figure 9: GMRES convergence with the fault matrix with ILu level 0 and 1.

in the computer science department. In order to run the direct methods it was necessary to store data on the hard drive during computations (`--ooc` option). Interestingly on ESAT's vierre64 server with `32 GB` ram and no `--ooc` computations took five hours to compute a solution of accuracy 10^{-10}. Probably because data did exceed the total memory available and the system started to swap and I/O operations on esat's network drives are slow.

3 Conclusion

When it comes to choosing a direct or indirect method it is crucial to consider the size of the problem. Direct methods have outperformed the iterative ones for the small problems considered in this report. When the problem is very large direct methods take very long and need a lot of memory. The memory requirements are particularly problematic since swapping might be necessary, which will slow down computations further. Iterative solvers can work well if a preconditioner is available that nicely clusters the eigenvalues and keeps the matrix close to normal. If that is the case iterative methods will greatly diminish computation time and memory consumption.