Bachelorarbeit

Implementation and experimental comparison between the COMMUNICATION AVOIDING-GENERALIZED MINIMAL RESIDUAL METHOD and standard GMRES

Verfasser

Robert Ernstbrunner

angestrebter akademischer Grad

Bachelor of Science (BSc)

Wien, 2019

Studienkennzahl lt. Studienblatt:	A 033 521
Fachrichtung:	Informatik - Scientific Computing
Betreuerin / Betreuer:	Univ.-Prof. Dipl.-Ing. Dr. Wilfried Gansterer, M.Sc.

Contents

1 TODO

- describe arnoldi(s)
- describe arnoldi(s,t)
- describe CA-GMRES
- write implementation details
- write numerical experiments
- write performance experiments
- write introduction
- write abstract

2 Notation

Similar notation is considered as in Hoemmen et al. [9] and Grigori et al. [8].

Linear Algebra

- Greek letters denote scalars, lower case Roman letters denote vectors (or - based on the context - dimensions), capital Roman letters denote matrices.
- Capital letters with two subscripts, e.g. '$V_{m,n}$', denote matrices with m rows and n columns.
- Capital *Black letter* letters (e.g. V, Q and R in Black letters are represented by $\mathfrak{V}$, $\mathfrak{Q}$ and $\mathfrak{R}$ resp.) denote matrices that are composed out of other matrices.
- v_k denotes the k^{th} vector in a series of vectors v_0, v_1, ..., , v_k, v_{k+1}, ... of equal length.
- Similarly, V_k denotes the k^{th} matrix in a sequence of matrices $V_0, V_1, \ldots, V_k, V_{k+1}, \ldots$. Generally all these matrices have the same number of rows. They may or may not have the same number of columns.
- If V is a matrix consisting of s vectors $[v_1, v_2, \ldots, v_s]$, then $\underline{V} = [V, v_{s+1}]$. The underline denotes one more column at the end.
- If again, V is a matrix consisting of s vectors $[v_1, v_2, \ldots, v_s]$, then $\acute{V} = [v_2, v_3, \ldots, v_s]$. The acute denotes one column less at the beginning.
- As a consequence $\underline{\acute{V}}$ denotes one more column at the end and one less column at the beginning, e.g. $\underline{\acute{V}} = [\acute{V}, v_{s+1}]$.
- Depending on the context, both underline or/and acute letters can also refer to rows as well.
- $0_{m,n}$ is defined as an $m \times n$ matrix consisting of zeros and e_k denotes the k^{th} canonical vector with the dimension depending on the context.
- Matlab notation is used for addressing elements of matrices and vectors. For example, given a matrix A of size $n \times n$ and two sets of indices α and β, $A(\alpha, :)$ is a submatrix formed by the subset of the rows of A whose indices belong to α. Similarly, $A(\alpha, \beta)$ is a submatrix formed by the subset of the rows of A whose indices belong to α and the subset of the columns of A whose indices belong to β.

Terms and definitions

- **Fetching**: the movement of data. This could either be *reading* , *copying* or *sending* and *receiving* messages.
- **Ghosting**: the storage of redundant data that does not belong to a processors assigned domain.

Graph Notation

- $G(A)$ denotes the directed graph of A with $G(A) = \{V, E\}$ where $V(G(A))$ is the set of vertices of $G(A)$ and $E(G(A))$ is the set of edges of $G(A)$.
- $R(G(A), \alpha)$ denotes the set of vertices in $G(A)$ that are reachable from any vertex in the set α, including α.
- $R(G(A), \alpha, m)$ denotes the set of vertices in $G(A)$ that are reachable by paths of length at most m from any vertex in α, including α.

Abbreviations

- **MPK**: Matrix powers kernel
- **TSQR**: Tall and skinny QR factorization
- **CGS**: Classical Gram-Schmidt method
- **MGS**: Modified Gram-Schmidt method
- **BCGS**: Block Classical Gram-Schmidt method

Abstract

3 Introduction

As the CPU-memory performance gap widens the cost for communication increases as well.
Compared to arithmetic costs communication costs are much higher and the widening CPU-memory performance gap promotes the need for communication-avoiding algorithms.

The CA-GMRES algorithm was implemented in a shared-memory environment.
Communication avoiding GMRES is based on s-step GMRES [6]

The term *communication* generally denotes the movement of data either between different processors in the parallel case or between 'fast' and 'slow' memory in the sequential case, where 'fast' and 'slow' are relative to the two levels examined in the memory hierarchy (e.g., cache and DRAM, or DRAM and disk). Communication optimal algorithms do not eliminate communication completely, but they are constructed in a way such that reduction of communication is prioritized. This often results in new challenges, e.g., the CA-ILU(0) algorithm (section 8.1.1) has to balance between communication and redundant computations; CA-GMRES (section 8) incorporates additional techniques to deal with ill-conditioned basis vectors.

4 Related work

s-step methods, CA-ILU(0)

5 GMRES

The Generalized Minimal Residual Method (GMRES) was first introduced by Saad et al. [12] and is an iterative Krylov subspace method for solving large sparse linear systems. The GMRES method starts with an initial approximate solution x_0 and initial residual $r_0 = b - Ax_0$ and finds a correction z_k at iteration k which solves the least-squares problem

$$z_k := \text{argmin}_z \|b - A(x_0 + z)\|_2 \tag{5.1}$$

where z_k is determined in the Krylov subspace

$$\mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}.$$

The solution at iteration k is then formed by $x_k = x_0 + z_k$. Since $\{r_0, Ar_0, \ldots, A^{k-1}r_0\}$ is usually ill-conditioned the Arnoldi method is incorporated to produce $k+1$ orthonormal

basis vectors $\underline{Q} = [q_1, q_2, \ldots, q_k, q_{k+1}]$ with $q_1 = r_0 / \|r_0\|_2$ and a $k+1 \times k$ upper Hessenberg coefficient matrix $\underline{H}$ where

$$AQ = \underline{Q}\underline{H}.$$

With these conditions z_k can be defined as $z := Qy$ such that

$$\begin{aligned} \text{argmin}_z \|b - A(x_0 + z)\|_2 &= \text{argmin}_y \|r_0 - AQy\|_2 \\ &= \text{argmin}_y \|r_0 - \underline{Q}\underline{H}y\|_2. \end{aligned}$$

Since $q_1 = r_0 / \|r_0\|_2$ and $\underline{Q}$ is orthonormal, one has

$$\begin{aligned} \text{argmin}_y \|r_0 - \underline{Q}\underline{H}y\|_2 &= \text{argmin}_y \|\underline{Q}^T r_0 - \underline{H}y\|_2 \\ &= \text{argmin}_y \|\beta e_1 - \underline{H}y\|_2 \end{aligned} \tag{5.2}$$

with $\beta = \|r_0\|_2$. $\underline{H}$ is then factored into $\underline{H} = \underline{G}\underline{U}$ with square matrix $\underline{G}$ being a product of k Givens rotations, $\underline{U} = \begin{pmatrix} U \\ 0_{1,k} \end{pmatrix}$ and U being upper triangular. The triangular system to solve is then given by

$$y_k := \text{argmin}_y \left\|\beta \underline{G}^T e_1 - \underline{U}y\right\|_2$$

The solution is obtained by computing $x_k = x_0 + Qy_k$. Note that the absolute value of the last coordinate of $\beta \underline{G}^T e_1$ is $\|b - Ax_k\|_2$, the absolute residual at iteration k.

6 Computational kernels

In this thesis *computational kernels* define parts of an algorithm with significantly high costs, relatively speaking. These costs include both arithmetic operations and communication. The following kernels make up the essential building blocks of Arnoldi(s, t) and eventually CA-GMRES.

6.1 Matrix powers kernel

The matrix powers kernel, as described by Hoemmen et al. in [9], was not implemented in the context of this thesis. However, it is an essential part to avoid communication and therefore, will be briefly summarized here.
In its basic form, the MPK takes an $n \times n$ matrix A and a starting vector v_1 as input and produces s more vectors $Av, A^2v, \ldots, A^sv$. Since A is usually large and sparse, it makes sense to also look at the graph of A, namely $G(A)$ in order to apply known graph algorithms. In s-step methods, the MPK replaces the sparse matrix-vector products that generate the basis for the Krylov subspace $\mathcal{K}_{s+1}(A, v) = \left[v, Av, A^2v, \ldots, A^sv\right] = [v_1, v_2, \ldots, v_{s+1}]$. One invocation of the MPK produces the same amount of basis vectors as s sparse matrix-vector products. The MPK sends a factor of $\Theta(s)$ fewer messages than s SpMV invocations and the matrix has to be read from slow to fast memory only once. In order to achieve this, the data and the workload are distributed among P processors,

Algorithm 1 GMRES(m)

Input: $n \times n$ linear system $Ax = b$ and initial guess x_0

1: restart := true
2: **while** restart **do**
3: $r_0 := b - Ax_0$, $\beta := \|r_0\|_2$, $q_0 := r_0/\beta$, $\underline{Q}_0 := q_0$, $\underline{H}_0 := \varnothing$
4: **for** $k = 1$ to m **do**
5: Compute q_k and h_k using MGS-Arnoldi
6: Set $\underline{Q}_k := [Q_{k-1}, q_k]$ and $\underline{H}_k := [\underline{H}_{k-1}, h_k]$
7: Reduce h_k of $\underline{H}_k$ from upper Hessenberg to upper triangular form using k Givens rotations $G_1, G_2, \ldots, G_k$. Apply the same rotations in the same order to βe_1, resulting in the length $k+1$ vector ζ_k.
8: Element $k+1$ of ζ_k is the 2-norm (in exact arithmetic) of the current residual $r_{k+1} = b - Ax_{k+1}$ of the current solution x_{k+1}.
9: **if** converged **then**
10: restart = false, and exit for loop
11: **end if**
12: **end for**
13: Use the above reduction of $\underline{H}_k$ to upper triangular form and ζ_k to solve $y_k := \operatorname{argmin}_y \|\underline{H}_k y - \beta e_1\|_2$
14: Set $x_0 := x_0 + Q_k y_k$
15: **end while**

where each processor is assigned a part α of $A(\alpha, :)$ and $v_1(\alpha)$ with $\alpha \subseteq V(G(A))$. Then, each processor fetches $A(\eta, :)$ and $v_1(\eta)$, with $\eta = R(G(A), \alpha, s) - \alpha$ in order to compute s more vectors $v_2(\alpha), v_3(\alpha), \ldots, v_{s+1}(\alpha)$ without communication. In other words, to compute $v_2(\alpha)$, the superset $\beta = R(G(A), \alpha, 1)$ is required. To compute $v_3(\alpha)$, the set $R(G(A), \beta, 1) = R(G(A), \alpha, 2)$ must be available. In general, to compute $v_{s+1}(\alpha)$, the set $R(G(A), \alpha, s)$ must be at hand. Since

$$\alpha \subseteq R(G(A), \alpha, 1) \subseteq \ldots \subseteq R(G(A), \alpha, s-1) \subseteq R(G(A), \alpha, s)$$

it is clear, that larger steps eventually lead to increasing amounts of ghosted data and floating point operations.

6.1.1 Preconditioned matrix powers kernel

Iterative Krylov methods often require a preconditioner that, when applied, fundamentally changes the MPK. In order to avoid communication, highly parallelizable preconditioners come to mind. E.g., Nuentsa et al. [10] present their parallel GMRES with a multiplicative Schwarz preconditioner. Grigori et al. [8] developed *CA-ILU(0)*, a very interesting type of preconditioner that, at first glance, seems unfit for a parallel and communication-avoiding environment. Section 8.1.1 summarizes their work.

6.2 Tall and skinny QR

TSQR is a QR decomposition algorithm especially suited for $m \times n$ matrices, where $m \gg n$. TSQR uses a divide-and-conquer approach and therefore, works on a reduction tree structure. The highest form of parallelism is achieved if TSQR uses a binary tree. In the purely sequential case a linear (flat) tree comes into play. Hybrid algorithms use anything in between and the best tree structure may depend on the matrix size and underlying architecture as Demmel et al. explain in [5]. The parallel TSQR with a binary tree is summarized below. For a more detailed description, as well as a description of the sequential algorithm, see Demmel et al. [5].

Parallel TSQR First, the matrix A is split up into P parts with each submatrix having size $m/P \times n$. TSQR on a binary tree then passes $P-1$ stages where any fast and accurate QR factorization can be applied for each stage. Let's assume $P = 4$, then

$$A = \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{pmatrix}.$$

At stage zero the QR factorization for each submatrix A_i is computed, with

$$A_0 = Q_{00}R_{00}, \qquad A_1 = Q_{10}R_{10}, \qquad A_2 = Q_{20}R_{20} \quad \text{and} \quad A_3 = Q_{30}R_{30}.$$

The successive stage merges the R-factors and computes the next QR-factorizations

$$\begin{pmatrix} R_{00} \\ R_{10} \end{pmatrix} = Q_{01}R_{01} \qquad \text{and} \qquad \begin{pmatrix} R_{20} \\ R_{30} \end{pmatrix} = Q_{11}R_{11}.$$

This procedure is repeated until one last QR factorization is performed where the final R factor can be interpreted as the root of the tree. This results in the following decomposition

$$A = \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{pmatrix} = \left(\begin{array}{c|c|c|c} Q_{00} & & & \\ \hline & Q_{10} & & \\ \hline & & Q_{20} & \\ \hline & & & Q_{30} \end{array}\right) \cdot \left(\begin{array}{c|c} Q_{01} & \\ \hline & Q_{11} \end{array}\right) \cdot Q_{02} \cdot R_{02}. \tag{6.1}$$

Figure 1 shows, that this approach only requires $\mathcal{O}(\log P)$ messages on P processors (a factor of $\Theta(s)$ fewer messages than Householder QR or MGS). [5] show that, sequentially, the matrix is read only once, saving a factor of $\Theta(s)$ transferred data between levels of the memory hierarchy (compared to Householder QR or MGS). Of course, (6.1) is stored implicitly to save storage space.
The orthogonality of the Q factor computed by Classical or Modified Gram-Schmidt depends on the condition number of A. Householder QR, on the other hand, does not make any assumptions on $\mathcal{K}(A)$, it is *unconditionally* stable. Therefore, Householder QR

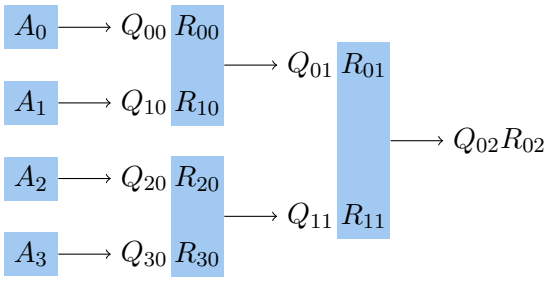

Figure 1: Parallel TSQR on a binary tree of four processors. The first subscript of the Q and R matrices indicates the sequence number for a stage, the second subscript is the stage number. The blue boxes represent the processors involved at each stage.

is a good choice for the local QR factorizations in TSQR, which makes TSQR inherently unconditionally stable as well.

For reasons explained in section 7.2.3 one might want TSQR to produce an R factor with real non-negative diagonal entries. Demmel et al. show in [4] how to modify the usual Householder QR in a numerically stable way so that such a factor is generated. This modified Householder QR factorization can then be incorporated, in order to let TSQR as well produce an R factor with real non-negative diagonal entries.

6.3 Block Gram-Schmidt

The Gram-Schmidt process takes a set of s linearly independent basis vectors $V = [v_1, \ldots, v_s]$ and creates an orthonormal basis that spans the same subspace as V. Unlike unblocked Gram-Schmidt methods, blocked Gram-Schmidt algorithms work on blocks of columns at a time instead of one column at a time. If the matrix consists of s columns this usually requires a factor of $\Theta(s)$ fewer messages and a factor of $\Theta(s)$ fewer data transfers between levels of the memory hierarchy.
In their performance analysis with a simplified parallel model, Hoemmen et al. [9] also show that blocked classical Gram-Schmidt is superior to a blocked modified Gram-Schmidt variant in terms of the messages sent between processors. However, they state that BCGS and BMGS contain similar accuracy properties as their unblocked versions. Therefore, BCGS is not as numerically stable as the blocked MGS variant. The modified Gram-Schmidt method is often used for basis orthogonalization in the Arnoldi iteration (section 7). Greenbaum et al. showed in [7], that it is the linear independence of the Arnoldi basis, not the orthogonality near machine precision, that is important when solving linear systems with GMRES. Therefore, the CA-GMRES algorithm (section 8) can make use of the classical Gram-Schmidt approach, that is presented below.

BCGS with TSQR Algorithm 2 shows a version of BCGS that incorporates TSQR in order to improve vector orthogonalization. BCGS orthogonalizes the $s + 1$ basis vectors

in $\underline{V_k}$, that are generated by the matrix powers kernel, against all previous basis vectors in Q by computing

$$\underline{V'_k} := (I - QQ^T)\underline{V_k} = \underline{V_k} - Q(Q^T\underline{V_k}). \tag{6.2}$$

TSQR then orthogonalizes these $k+1$ basis vectors with respect to each other. Combined, these two kernels do the work of updating a QR factorization with new columns. The advantage of TSQR and BCGS over unblocked MGS is, that they move asymptotically less data between levels of the memory hierarchy. Unlike MGS, BCGS consists almost entirely of dense matrix-matrix operations. TSQR improves orthogonality of the block columns. Therefore, if algorithm 2 is used for solving linear systems, reorthogonalization can be omitted entirely.

Algorithm 2 BCGS with TSQR

Input: $V = [V_1, V_2, \ldots, V_M]$ where V is $n \times m$. Each V_k is $n \times m_k$, with $\sum_{k=1}^{M} m_k = m$.
Output: $Q = [Q_1, \ldots, Q_M]$, where $\mathcal{R}(Q) = \mathcal{R}(V)$ and $\mathcal{R}([Q_1, \ldots, Q_k]) = \mathcal{R}([V_1, \ldots, V_k])$
Output: $R : m \times m$ upper triangular matrix.

```
1: for k = 1 to M do
2:     R_{1:k-1,k} := [Q_1, ..., Q_{k-1}]^T V_k
3:     V'_k := V_k - [Q_1, ..., Q_{k-1}] R_{1:k-1,k}
4:     Compute V'_k = Q_k R_kk via TSQR
5: end for
```

7 CA-Arnoldi

7.1 Arnoldi Iteration

The Arnoldi iteration is a method for solving sparse non-symmetric eigenvalue problems and was first introduced by W. Arnoldi in [1]. S steps of standard Arnoldi produce an $s+1 \times s$ upper Hessenberg matrix $\underline{H}$ and $m \times s+1$ orthonormal vectors $\underline{Q} = [q_1, q_2, \ldots, q_s, q_{s+1}]$, where

$$AQ = \underline{Q}\underline{H}. \tag{7.1}$$

There are many ways to orthogonalize successive basis vectors. Modified Gram-Schmidt is often employed because it performs numerically better compared to classical Gram-Schmidt (MGS based Arnoldi is outlined in algorithm 3). On the other hand, CGS is more suited for parallel implementations, because it provides fewer synchronization points. Walker [13] used Householder QR instead because it provides better orthogonalization than MGS. Since Householder QR requires the vectors to be available all at once, Walker produced s Monomial basis vectors first. Hoemmen et al. Bai et al. [3] later improved on Walkers work by replacing the (usually ill-conditioned) Monomial basis with the Newton basis instead. Greenbaum et al. [7] later proved, that the loss of orthogonality, that is caused by MGS, usually does not affect the solution of the linear system at all.

Algorithm 3 MGS based Arnoldi iteration

Input: $n \times n$ matrix A and starting vector v of size n

Output: Orthonormal $n \times s+1$ matrix $\underline{Q} = [Q, q_{s+1}]$, and a non-singular $s+1 \times s$ upper Hessenberg matrix $\underline{H}$ such that $AQ = \underline{QH}$

1: $\beta := \|v\|_2$, $q_1 := v/\beta$
2: **for** $j = 1$ to s **do**
3: $\quad w_j := Aq_j$
4: $\quad$ **for** $i = 1$ to j **do**
5: $\quad\quad h_{ij} := \langle w, q_i \rangle$
6: $\quad\quad w_j := w_j - h_{ij} q_i$
7: $\quad$ **end for**
8: $\quad h_{j+1,j} := \|w_j\|_2$
9: $\quad q_{j+1} := w_j / h_{j+1,j}$
10: **end for**

7.2 Arnoldi(s,t)

7.2.1 The Monomial basis

The Monomial basis in s-step Krylov methods is given by

$$\mathcal{K}_{s+1}(A, v) = [v, Av, A^2 v, \ldots, A^s v]$$

and has a change of basis matrix

$$\underline{B} = [\sigma_1 e_2, \sigma_2 e_3, \ldots, \sigma_s e_{s+1}].$$

with scaling factors $\sigma_1, \ldots, \sigma_s$ that satisfies

$$AV = \underline{VB}. \tag{7.2}$$

The Monomial basis is also known as the *power method* which is an iterative method for finding the principal eigenvalue and corresponding eigenvector of a matrix by repeatedly applying a starting vector to the matrix. If the matrix and starting vector satisfy certain conditions, the basis converges to the principal eigenvector. Ideally, a basis has orthogonal basis vectors and should not converge. In theory, the converged basis is still linearly independent in exact arithmetic. In machine precision, the basis vectors become inevitably dependent at some point.

The similarity between (7.2) and the Arnoldi relation (7.1) can be used in order to reconstruct the upper Hessenberg matrix $\underline{H}$ in (7.1). The vectors created from the QR factorization of $\underline{V}$ might differ from the ones created by the standard Arnoldi method by a unitary scaling (see section 7.2.3 for details). For simplicity, it is assumed, that the QR factorization $\underline{V} = \underline{QR}$ produces the same unitary vectors as the usual approach would.

From

$$\begin{aligned} AV &= \underline{VB} \\ AQR &= \underline{QRB} \\ AQ &= \underline{QRB}R^{-1} \end{aligned}$$

emerges

$$\underline{H} = \underline{RB}R^{-1}. \tag{7.3}$$

Since $\underline{H}$ is upper Hessenberg, $\underline{B}$ must be at least structurally upper Hessenberg as well. This is in fact the case for the Monomial basis.
The Monomial basis may be scaled in order to prevent rapid growth in the length of the basis vectors.
scaling the rapidly growing condition number and vector length $\rightarrow$ scale vector to lengh 1, but: $\rightarrow$ need a different basis.

7.2.2 The Newton basis

The Newton basis in s-step Krylov methods is given by

$$\mathcal{K}_{s+1}(A, v) = \left[v, (A - \theta_1 I)v, (A - \theta_2 I)(A - \theta_1 I)v, \ldots, \prod_{i=1}^{s}(A - \theta_i I)v \right]$$

and has a change of basis matrix

$$\underline{B} = \begin{pmatrix} \theta_1 & 0 & \ldots & 0 \\ \sigma_1 & \theta_2 & \ddots & \vdots \\ 0 & \sigma_2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \theta_s \\ 0 & 0 & \ldots & \sigma_s \end{pmatrix}$$

with scaling factors $\sigma_1, \ldots, \sigma_s$.

polinomial interpolation at shifts θ_1, θ_2, ..., θ_s.

Choosing the shifts find estimates of the eigenvalues of A (Ritz values).

The modified Leja ordering

Avoiding complex arithmetic Like the eigenvalues of a real matrix, the Ritz values can also occur as complex conjugate pairs. The modified Leja ordering ensures that these pairs are ordered consecutively with leading positive imaginary entries, i.e. $\theta_{j+1} = \overline{\theta}_j$ with $\Im(\theta_j) > 0$. Complex arithmetic doubles the storage and floating point operations and therefore, should be avoided. Instead of computing $v_{j+1} = (A - \theta_j I)v_j$ and $v_{j+2} = (A - \overline{\theta}_j I)v_{j+1}$ like one would normally do, Bai et al. [3] suggest that complex arithmetic can be skipped by setting

$$v_{j+1} = (A - \Re(\theta_j)I)v_j \tag{7.4}$$

and

$$v_{j+2} = (A - \Re(\theta_j)I)v_{k+1} + \Im(\theta_j)^2 v_j. \tag{7.5}$$

It can easily be shown that

$$\begin{aligned} v_{j+2} &= (A - \Re(\theta_j)I)^2 v_j + \Im(\theta_j)^2 v_j \\ &= (A - \overline{\theta}_j I)(A - \theta_j I)v_j. \end{aligned}$$

This also affects the change of basis matrix $\underline{B}$. If the Ritz values contain complex conjugate pairs, $\underline{B}$ is tridiagonal. E.g., if θ_1 through θ_s are real, with the exception of θ_j and θ_{j+1} being a complex conjugate pair, the change of basis matrix is given by

$$\underline{B} = \begin{pmatrix} \theta_1 & 0 & \dots & \dots & \dots & 0 \\ \sigma_1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \Re(\theta_j) & -\Im(\theta_j)^2 & \ddots & \vdots \\ \vdots & \ddots & \sigma_j & \Re(\theta_{j+1}) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \sigma_{j+1} & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \theta_s \\ 0 & \dots & \dots & \dots & 0 & \sigma_s \end{pmatrix}.$$

Performance notes Avoiding complex arithmetic in that way necessitates an extra SpMV operation in the Newton basis which, in the worst case, leads to as many floating point operations as in the Chebychev basis. However, in their performance analysis Hoemmen et al. [9] observed that the runtime of the Newton basis was still close to the runtime of the Monomial basis.
[9] further point out that this approach might lose accuracy when θ_{j-1} is real and θ_j and θ_{j+1} form a complex conjugate pair with $\Re(\theta_j) = \theta_{j-1}$. Then

$$\begin{aligned} v_j &= (A - \theta_{j-1}I)v_{j-1} \\ v_{j+1} &= (A - \Re(\theta_j)I)(A - \theta_{j-1}I)v_{j-1} \\ &= (A - \theta_{j-1}I)^2 v_{j-1} \end{aligned}$$

is equivalent to computing the Monomial basis with a possibly ill-conditioned matrix $A - \theta_{j-1}I$. This might occur, if the Ritz values reside within an ellipse with a long vertical axis and very short horizontal axis on the complex plane.

Algorithm 4 Arnoldi(s,t)

Input: $n \times n$ matrix A and starting vector v of size n

Output: An orthonormal n by $st+1$ matrix $\underline{\mathfrak{Q}} = [\mathfrak{Q}, q_{st+1}]$ and a non-singular $st+1$ by st upper Hessenberg matrix $\underline{H}$ such that $AQ = \underline{Q}\underline{H}$

1: $\beta := \|v\|_2$, $q_1 := v/\beta$, restart := false
2: **for** $k = 0$ to $t-1$ **do**
3: Fix basis conversion matrix $\underline{B}_k$
4: Compute $\underline{\acute{V}}$ via MPK
5: **if** $k = 0$ **then**
6: Compute the QR factorization $\underline{V}_0 = \underline{Q}_0\underline{R}_0$ via TSQR
7: $\underline{\mathfrak{Q}}_0 := \underline{Q}_0$
8: $\underline{\mathfrak{H}}_0 := \underline{R}_0\underline{B}_0R_0^{-1}$
9: **else**
10: $\underline{\acute{\mathfrak{R}}}_{k-1,k} := \underline{\mathfrak{Q}}_{k-1}^T\underline{\acute{V}}_k$
11: $\underline{\acute{V}}_k' := \underline{\acute{V}}_k - \underline{\mathfrak{Q}}_{k-1}\underline{\acute{\mathfrak{R}}}_{k-1,k}$
12: Compute QR factorization $\underline{\acute{V}}_k' = \underline{\acute{Q}}_k\underline{\acute{R}}_k$
13: Compute $\underline{\mathfrak{H}}_{k-1,k} := -\mathfrak{H}_{k-1}\mathfrak{R}_{k-1,k}R_k^{-1} + \underline{\mathfrak{R}}_{k-1,k}\underline{B}_kR_k^{-1}$
14: Compute $H_k := R_kB_kR_k^{-1} + \tilde{\rho}_k^{-1}b_kz_ke_s^T - h_{k-1}e_1e_{sk}^T\mathfrak{R}_{k-1,k}R_k^{-1}$
15: Compute $h_k := \tilde{\rho}_k^{-1}\rho_kb_k$
16: $\underline{\mathfrak{H}} := \begin{pmatrix} \mathfrak{H}_{k-1} & \mathfrak{H}_{k-1,k} \\ h_{k-1}e_1e_{sk}^T & H_k \\ 0_{1,sk} & h_ke_s^T \end{pmatrix}$
17: **end if**
18: **end for**

7.2.3 Scaling the first basis vector

Arnoldi(s,t) produces $\underline{\mathfrak{Q}}$ which differs from MGS-Arnoldi $\underline{\hat{\mathfrak{Q}}}$ by a unitary scaling $\underline{\Theta} = \mathrm{diag}(\theta_1, \theta_2, \ldots, \theta_{st}, \theta_{st+1})$ such that $\underline{\hat{\mathfrak{Q}}} = \underline{\mathfrak{Q}\Theta}$. $\rightarrow$

- QR factorization must not change direction of the first column
- compute $\theta_1 = \langle r_0, q_1 \rangle \ / \ \beta$
- compute q_1 via MGS-Arnoldi (this happens naturally when the first outer iteration is started with MGS-Arnoldi in order to compute Ritz values for the Newton basis)

7.2.4 QR factorization update

overlapping / non overlapping approach.

$[\underline{\mathfrak{Q}}_0, \underline{\acute{V}}_1] = [\underline{\mathfrak{Q}}_0, \underline{\acute{Q}}_1] \cdot \begin{pmatrix} I_{s+1,s+1} & \underline{\acute{\mathfrak{R}}}_{0,1} \\ 0_{s,s+1} & \underline{\acute{R}}_1 \end{pmatrix}$

BGS ...

Repartitioning the R factor:

$$\begin{aligned}
\mathfrak{R}_{k-1,k} &= \mathfrak{Q}_{k-1}^T V_k, \\
\underline{\mathfrak{R}}_{k-1,k} &= \mathfrak{Q}_{k-1}^T \underline{V}_k, \\
\acute{\mathfrak{R}}_{k-1,k} &= \underline{\mathfrak{Q}}_{k-1}^T \acute{V}_k, \\
\underline{\acute{\mathfrak{R}}}_{k-1,k} &= \underline{\mathfrak{Q}}_{k-1}^T \underline{\acute{V}}_k.
\end{aligned}$$

$$\begin{aligned}
\underline{\mathfrak{R}}_k &= \begin{pmatrix} I_{sk+1,sk+1} & \acute{\mathfrak{R}}_{k-1,k} \\ 0_{s,sk+1} & \acute{R}_k \end{pmatrix} = \begin{pmatrix} I_{sk,sk} & \underline{\mathfrak{R}}_{k-1,k} \\ 0_{s+1,sk} & \underline{R}_k \end{pmatrix} \\
\mathfrak{R}_k &= \begin{pmatrix} I_{sk+1,sk+1} & \acute{\mathfrak{R}}_{k-1,k} \\ 0_{s-1,sk+1} & \acute{R}_k \end{pmatrix} = \begin{pmatrix} I_{sk,sk} & \mathfrak{R}_{k-1,k} \\ 0_{s,sk} & R_k \end{pmatrix}
\end{aligned}$$

R_k and $\underline{\acute{R}}_k$ are $s \times s$ matrices, $\underline{R}_k$ is $s+1 \times s+1$ and $\acute{R}_k$ is $s-1 \times s-1$.

7.2.5 Reconstructing the upper Hessenberg matrix

flop optimization. how to apply Givens rotations
$AV_k = \underline{V}_k \underline{B}_k$

$A[\mathfrak{Q}_{k-1}, V_k] = [\mathfrak{Q}_{k-1}, \underline{V}_k]\underline{\mathfrak{B}}_k$

where $\underline{\mathfrak{B}}_k$ satisfies:

$$\underline{\mathfrak{B}}_k = \begin{pmatrix} \mathfrak{H}_{k-1} & 0_{sk,s} \\ h_{k-1} e_1 e_{sk}^T & \underline{B}_k \end{pmatrix} \tag{7.6}$$

with $\mathfrak{H}_0 := H_0$

$$\begin{aligned}
A & \quad [\underline{\mathfrak{Q}}_{k-1}, \acute{Q}_k] \cdot \begin{pmatrix} I_{sk+1,sk+1} & \acute{\mathfrak{R}}_{k-1,k} \\ 0_{s-1,sk+1} & \acute{R}_k \end{pmatrix} \\
= & \quad [\underline{\mathfrak{Q}}_{k-1}, \underline{\acute{Q}}_k] \cdot \begin{pmatrix} I_{sk+1,sk+1} & \underline{\acute{\mathfrak{R}}}_{k-1,k} \\ 0_{s,sk+1} & \underline{\acute{R}}_k \end{pmatrix} \cdot \begin{pmatrix} \mathfrak{H}_{k-1} & 0_{sk,s} \\ h_{k-1} e_1 e_{sk}^T & \underline{B}_k \end{pmatrix}
\end{aligned}$$

We have

$$A[\underline{\mathfrak{Q}}_{k-1}, \acute{Q}_k] = [\underline{\mathfrak{Q}}_{k-1}, \underline{\acute{Q}}_k]\underline{\mathfrak{H}}_k$$

therefore,

$$\underline{\mathfrak{H}}_k = \begin{pmatrix} I_{sk+1,sk+1} & \acute{\mathfrak{R}}_{k-1,k} \\ 0_{s,sk+1} & \underline{\acute{R}}_k \end{pmatrix} \cdot \begin{pmatrix} \mathfrak{H}_{k-1} & 0_{sk,s} \\ h_{k-1} e_1 e_{sk}^T & \underline{B}_k \end{pmatrix} \cdot \begin{pmatrix} I_{sk+1,sk+1} & \acute{\mathfrak{R}}_{k-1,k} \\ 0_{s-1,sk+1} & \acute{R}_k \end{pmatrix}^{-1}$$

$$\underline{\mathfrak{H}}_{k-1,k} := -\mathfrak{H}_{k-1} \mathfrak{R}_{k-1,k} R_k^{-1} + \underline{\mathfrak{R}}_{k-1,k} \underline{B}_k R_k^{-1} \tag{7.7}$$

A different approach In order to get better Eigenvalue approximations Erhel [6] computed $2s$ Ritz values, picked s of them and applied the Modified Leja Ordering. This improved the condition of the Newton basis and lead to (better) convergence in some cases. Hoemmen et al. [9] recommend the same approach. However, one has to consider that these $2s$ values could come in complex conjugate pairs and have to amount to s values eventually. Consider the following. If the $2s$ Ritz values solely consist of complex conjugate pairs and s is odd, there is no way this would fit without splitting a complex conjugate pair. To address this issue one could either omit the positive imaginary part of the last Ritz value or incorporate all computed values, i.e. apply $2s$ Ritz values over two outer iterations of Arnoldi(s,t). Since computing $2s$ Ritz values yields in better approximations of the eigenvalues, there is no reason why one should not incorporate them all. This slightly changes the way equation (7.7) is computed. Remember that, in order to avoid complex arithmetic, the consecutive order of a complex conjugate pair must be preserved. In the case where θ_s is the first entry of a complex conjugate pair, the first change of basis matrix $\underline{B}_{k-1}$ is connected to its consecutive change of basis matrix $\underline{B}_k$ by an additional entry right to the last Ritz value of $\underline{B}_{k-1}$ and above the first Ritz value of $\underline{B}_k$. $\underline{\mathfrak{B}}_k$ then differs from equation (7.6) with this additional entry

$$\underline{\mathfrak{B}}_k = \begin{pmatrix} \mathfrak{H}_{k-1} & -e_{sk}e_1^T\Im(\theta_s)^2 \\ h_{k-1}e_1e_{sk}^T & \underline{B}_k \end{pmatrix}. \tag{7.8}$$

As mentioned before, equation (7.7) then changes to

$$\underline{\mathfrak{H}}_{k-1,k} := -\mathfrak{H}_{k-1}\mathfrak{R}_{k-1,k}R_k^{-1} + \underline{\mathfrak{R}}_{k-1,k}\underline{B}_kR_k^{-1} - \Im(\theta_s)^2e_{sk}e_1^TR_k^{-1}. \tag{7.9}$$

8 CA-GMRES

The CA-GMRES algorithm solves a different least-squares problem than (5.2):

$$\text{argmin}_y \left\|\beta e_1 - \underline{RB}R^{-1}y\right\|_2 \tag{8.1}$$

Algorithm 5 Newton-GMRES(s, t)

Input: $n \times n$ linear system $Ax = b$ and initial guess x_0

restart := true
while restart **do**
  $r_0 := b - Ax_0$, $\beta := \|r_0\|_2$, $q_1 := r_0/\beta$,
  for $k = 0$ to $t - 1$ **do**
    if $k = 0$ **then**
      Compute $\underline{Q}_0$ and $\underline{H}_0$ using MGS-Arnoldi
      Set $\underline{\mathfrak{Q}}_0 := \underline{Q}_0$ and $\underline{\mathfrak{H}}_0 := \underline{H}_0$
      Compute Ritz values from H_0 and fix basis conversion matrix $\underline{B}_k$
      Reduce $\underline{H}_0$ from upper Hessenberg to upper triangular form using s Givens rotations G_1, G_2, ..., G_s. Apply the same rotations in the same order to βe_1, resulting in the length $s + 1$ vector ζ_0.
    else
      Set $v_{sk+1} := q_{sk+1}$
      Compute $\acute{\underline{V}}_k$ where $v_{i+1} = (A - \theta_i I)v_i, i = sk + 1 : sk + s$
      $\acute{\underline{\mathfrak{R}}}_{k-1,k} := \underline{\mathfrak{Q}}_{k-1}^T \acute{\underline{V}}_k$
      $\acute{\underline{V}}_k' := \acute{\underline{V}}_k - \underline{\mathfrak{Q}}_{k-1} \acute{\underline{\mathfrak{R}}}_{k-1,k}$
      Compute QR factorization of $\acute{\underline{V}}_k' \to \acute{\underline{Q}}_k \acute{\underline{R}}_k$ using TSQR
      Compute $\underline{\mathfrak{H}}_{k-1,1} := -\mathfrak{H}_{k-1}\mathfrak{R}_{k-1,k}R_k^{-1} + \underline{\mathfrak{R}}_{k-1,k}\underline{B}_k R_k^{-1}$
      Compute $H_k := R_k B_k R_k^{-1} + \tilde{\rho}_k^{-1} b_k z_k e_s^T - h_{k-1} e_1 e_{sk}^T \mathfrak{R}_{k-1,k} R_k^{-1}$
      Compute $h_k := \tilde{\rho}_k^{-1} \rho_k b_k$
      $\underline{\mathfrak{H}}_k := \begin{pmatrix} \mathfrak{H}_{k-1} & \underline{\mathfrak{H}}_{k-1,k} \\ h_{k-1} e_1 e_{sk}^T & H_k \\ 0_{1,sk} & h_k e_s^T \end{pmatrix}$
      Apply Givens rotations G_1, ..., G_{sk} in order to $\begin{pmatrix} \underline{\mathfrak{H}}_{k-1,k} \\ \underline{H}_k \end{pmatrix}$.
      Reduce $\underline{H}_k$ to upper triangular form using s Givens rotations G_{sk+1}, ..., $G_{s(k+1)}$. Apply the rotations in the same order to $\begin{pmatrix} \zeta_{k-1} \\ 0_s \end{pmatrix}$, resulting in the length $s(k + 1) + 1$ vector ζ_k.
    end if
    Element $s(k + 1) + 1$ of ζ_k is the 2-norm (in exact arithmetic) of the current residual $r_{k+1} = b - Ax_{k+1}$ of the current solution x_{k+1}.
    if converged **then**
      restart = false, and exit for loop
    end if
  end for
  Use the above reduction of $\underline{\mathfrak{H}}_k$ to upper triangular form and ζ_k to solve $y_k := \operatorname{argmin}_y \left\|\underline{\mathfrak{H}}_k y - \beta e_1\right\|_2$
  Set $x_0 := x_0 + \mathfrak{Q}_k y_k$
end while

8.1 Preconditioning

Left, right, split, we consider left preconditioning ($M^{-1}Ax = M^{-1}b$) only. Scaling is a special type of preconditioning. [9] considered two types of scaling in order to prevent rapid basis vector growth:

1. Balancing: replacing A by $A' = DAD^{-1}$ with D diagonal.
2. Equilibration: replacing A by $A' = D_r A D_c$ with D_r and D_c diagonal.

In their experiments solving nonsymmetric linear systems with CA-GMRES [9] found that for practical problems, equilibration proved quite effective and almost made the basis type irrelevant. We observed something similar after applying the ILU(0) preconditioner.

8.1.1 CA-ILU(0) preconditioner

$M = LU$
Algorithm 5 in CA-GMRES apply M^{-1} to the red parts, i.e. replace $r_0 = b - Ax_0$ by $r_0 = M^{-1}(b - Ax_0)$ and $v_{i+1} = (A - \theta_i)v_i$ by $v_{i+1} = M^{-1}((A - \theta_i)v_i)$ summarize [8] (can be very long or short, dependent on overall length)

8.2 Convergence metrics

CA-GMRES produces cheap convergence metric, namely the relative residual $\|r_{k+1}\|_2$ / $\|r_0\|_2$. Might not be the best choice, depends too much on initial guess x_0.

If

- $\|x_0\|_2$ too large $\rightarrow \|r_0\|$ will be large and iteration will stop too early.
- $x_0 = 0$ harder to make the relative residual small if A is ill-conditioned and x_{k+1} lies nearly in the nullspace of A.

8.3 Implementation details

language: C++, libraries: intel MKL,
The Intel® Math Kernel Library has been optimized by exploiting both processor and system features and capabilities. Special care has been given to those routines that most profit from cache-management techniques. These especially include matrix-matrix operation routines such as dgemm(). In addition, code optimization techniques have been applied to minimize dependencies of scheduling integer and floating-point units on the results within the processor. The major optimization techniques used throughout the library include: • Loop unrolling to minimize loop management costs • Blocking of data to improve data reuse opportunities • Copying to reduce chances of data eviction from cache • Data prefetching to help hide memory latency • Multiple simultaneous operations (for example, dot products in dgemm) to eliminate stalls due to arithmetic unit pipelines

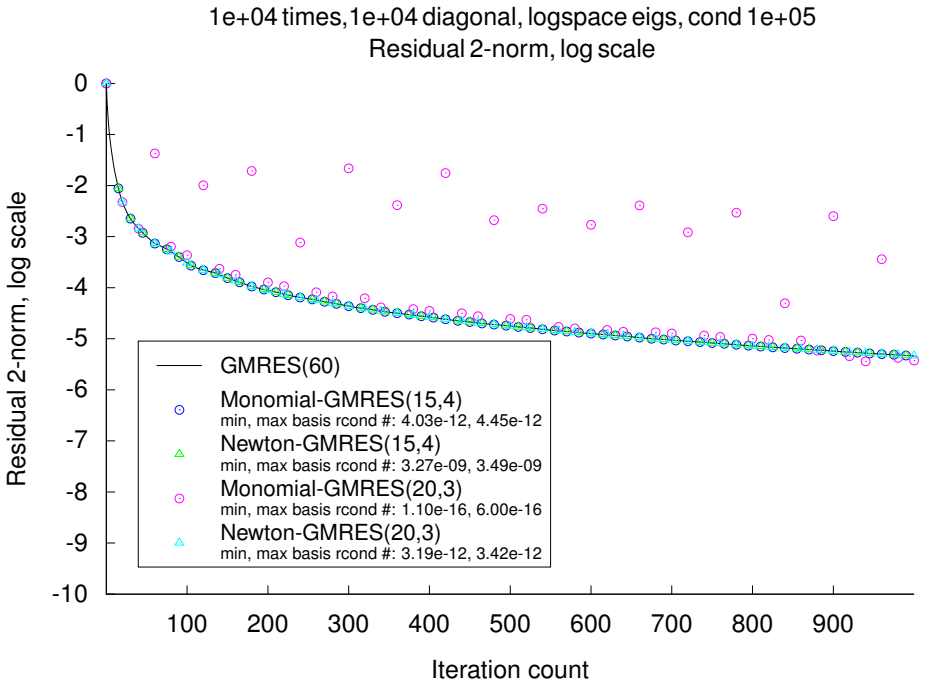

Figure 2: •••

• Use of hardware features such as the SIMD arithmetic units, where appropriate. These are techniques from which the arithmetic code benefits the most.
profiler: ??? (intel VTune Amplifier, TAU, ...) could not implement neither the MPK nor the CA-ILU(0) preconditioner due to time constraints. Also Modified Leja ordering does not deal with under / overflow in the product to maximize like in [9].

8.4 Numerical experiments

`It is known, but yet unexplained, that sometimes the norm of the Arnoldi residual converges to zero (for the Householder-GMRES this represents a typical behaviour) even after the true residual norm stagnates. For MGS-GMRES, however, the norms of the true and Arnoldi residuals stagnate obviously at about the same level. Greenbaum [7]`
How the true solution $\hat{x}$ was generated: $\hat{x}(k) = u(k) + sin(2\pi k/n)$, where the scalar $u(k)$ is chosen from a random uniform [-1, 1] distribution. $\hat{x}$ was chosen in this way because a completely random solution is usually nonphysical, but a highly nonrandom solution (such as a vector of all ones) might be near an eigenvector of the matrix (which would result in artificially rapid convergence of the iterative method).

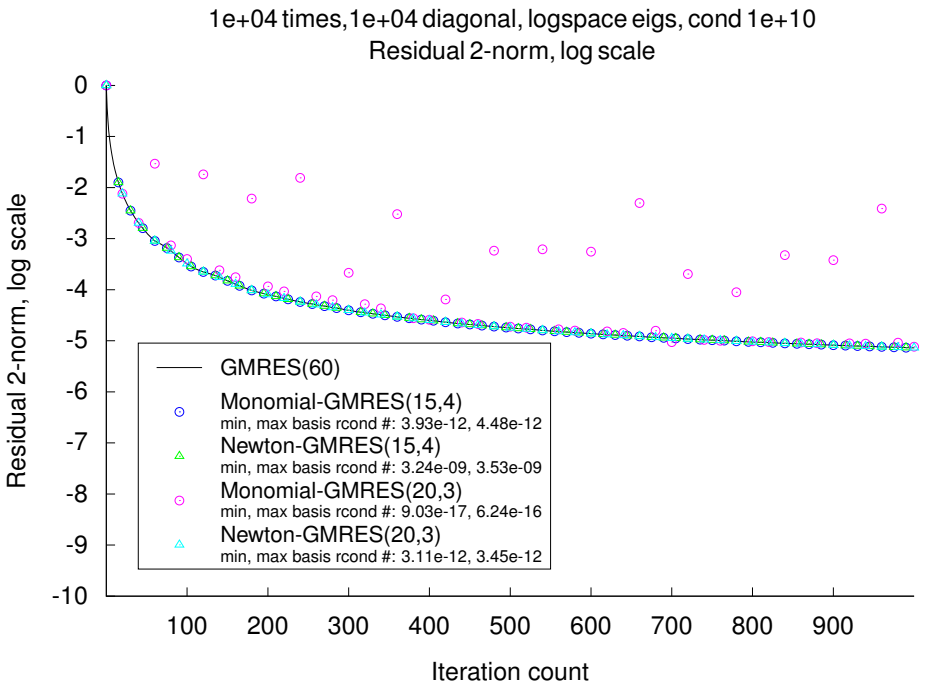

Figure 3: ●●●

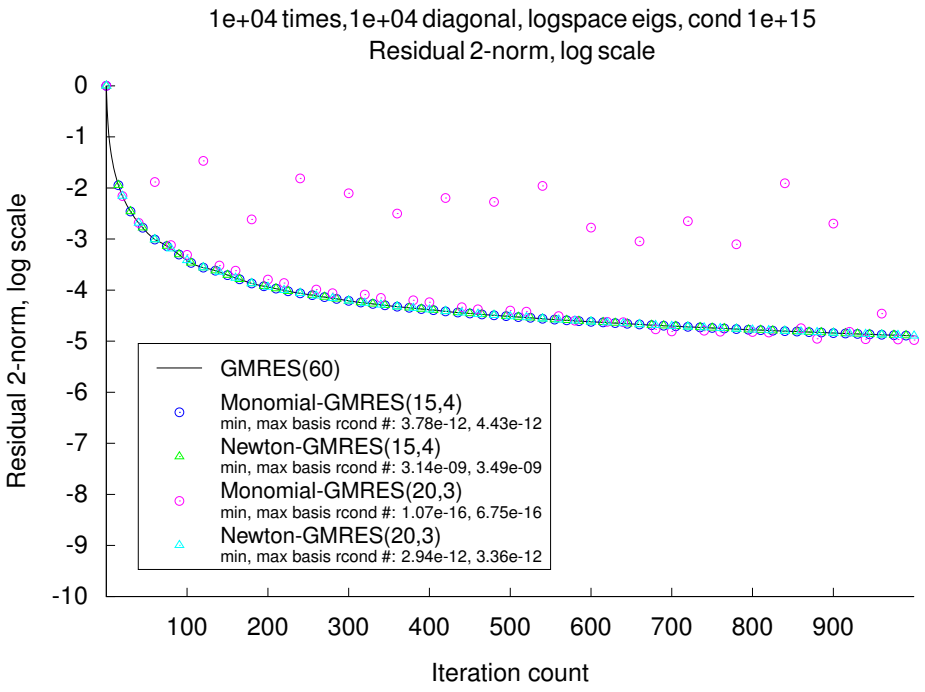

Figure 4: ●●●

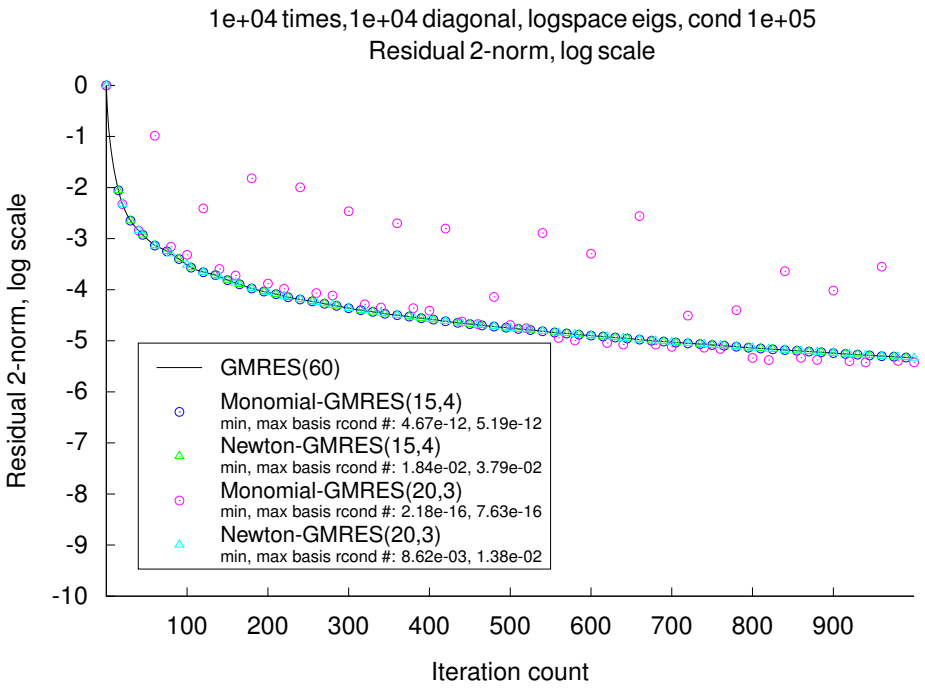

Figure 5: ●●●

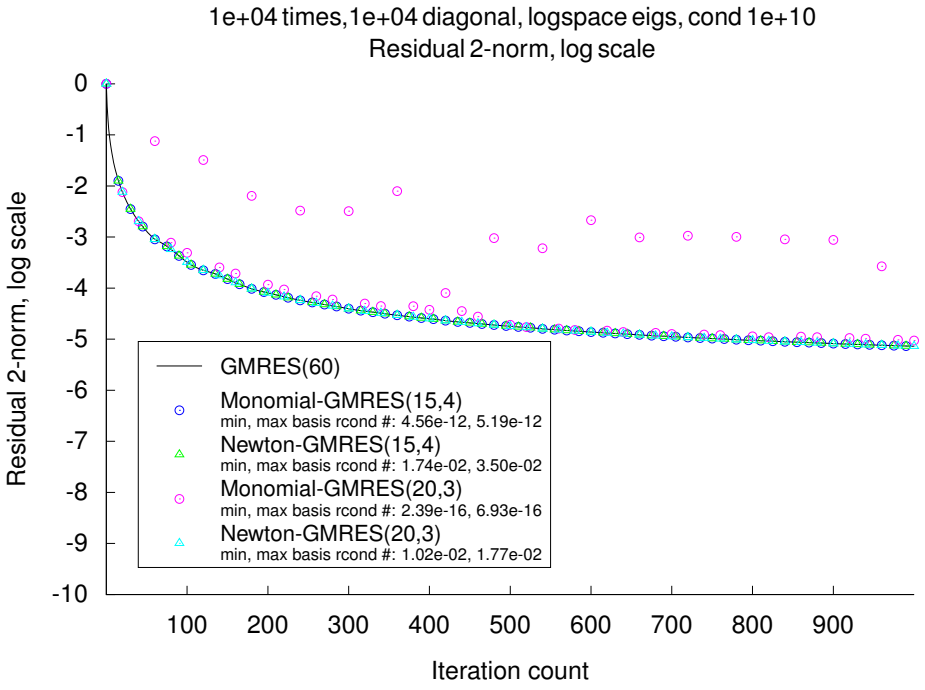

Figure 6: ●●●

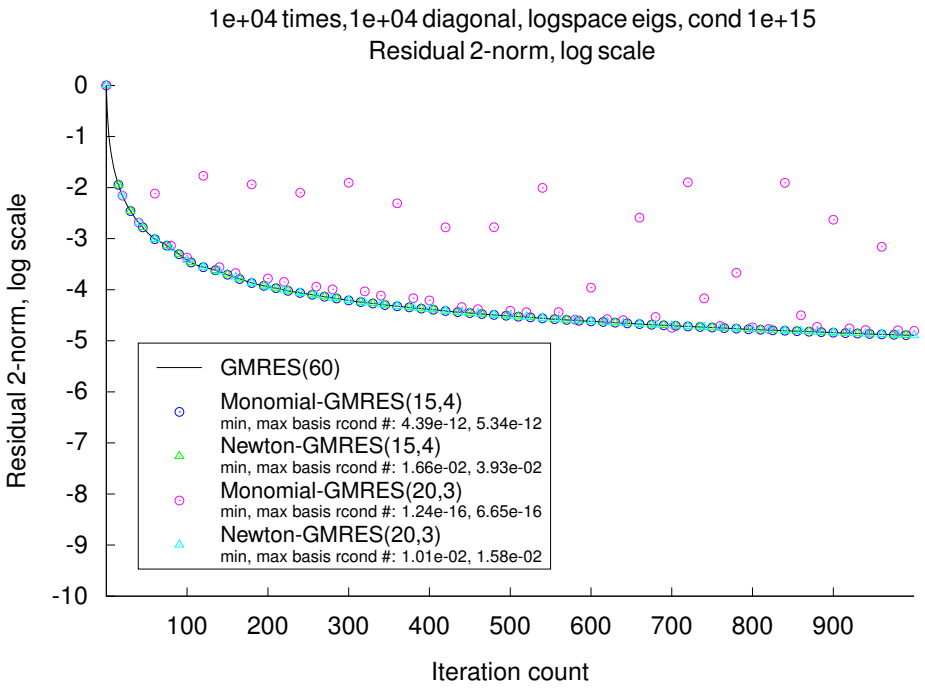

Figure 7: ●●●

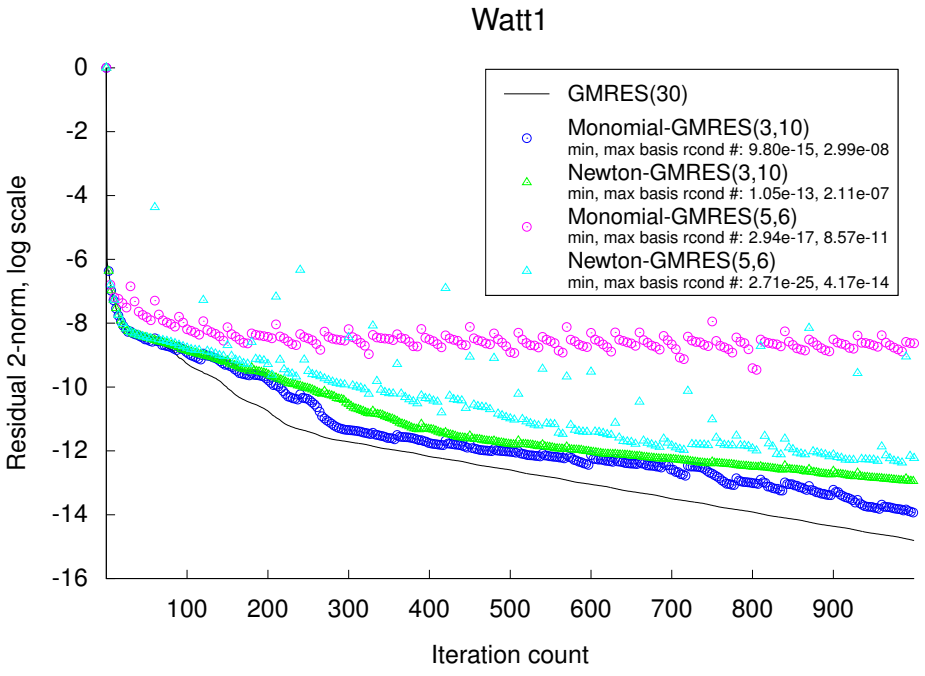

Figure 8: not scaled

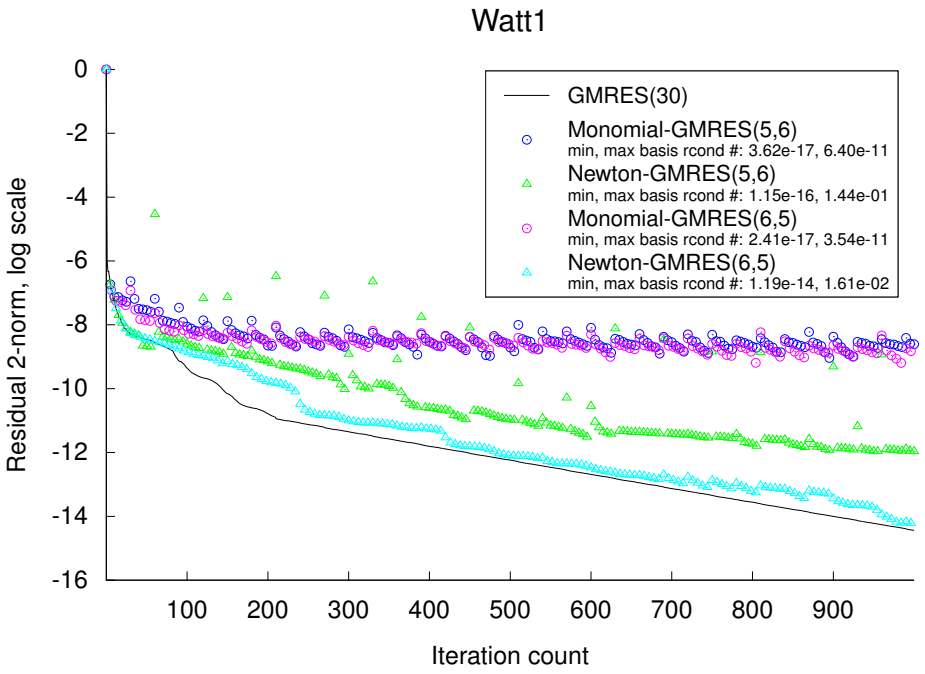

Figure 9: ●●●

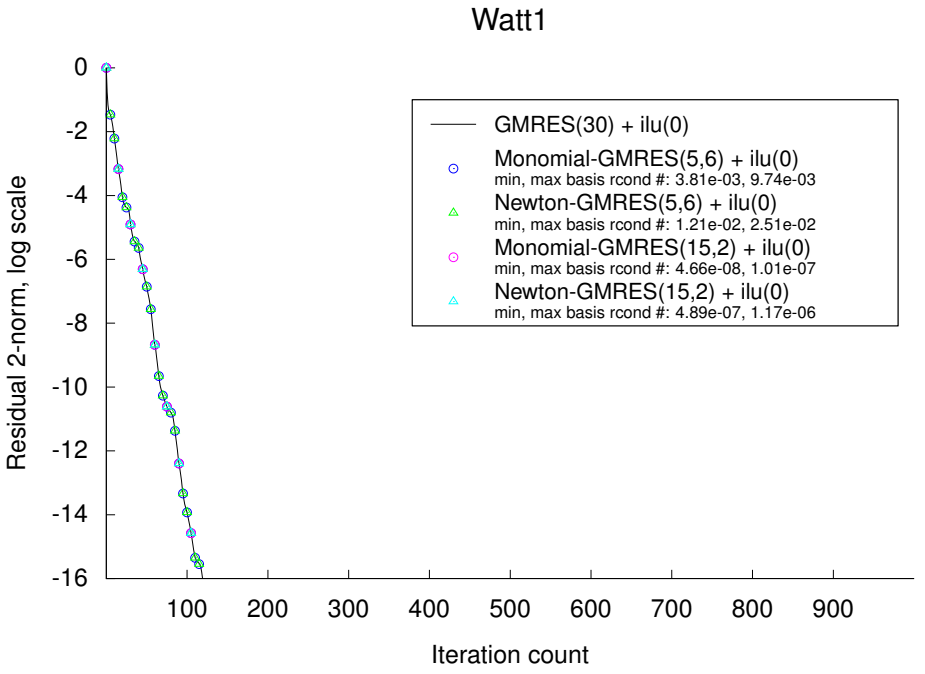

Figure 10: without basis vector scaling

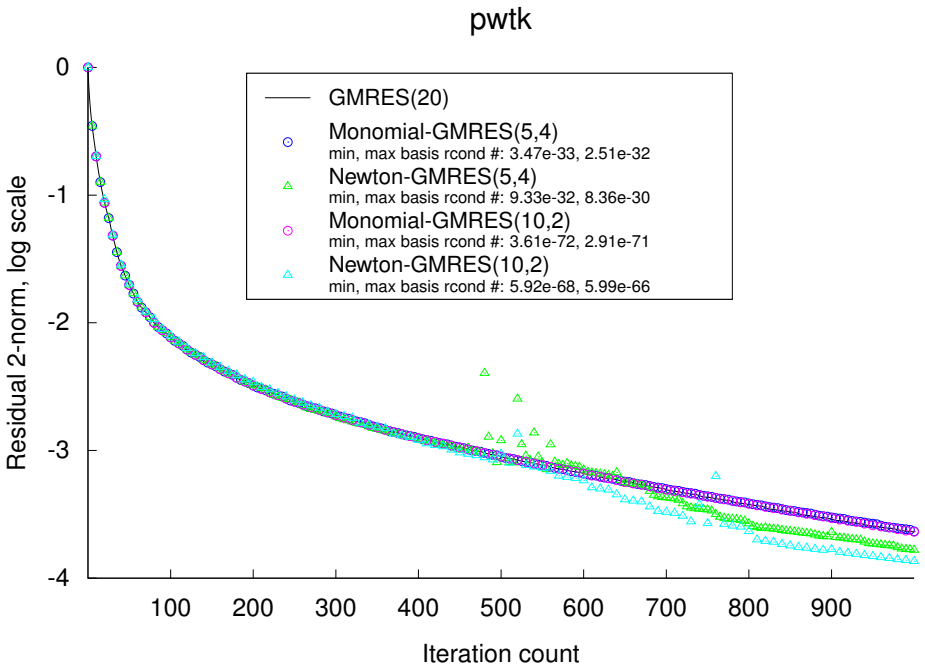

Figure 11: ●●●

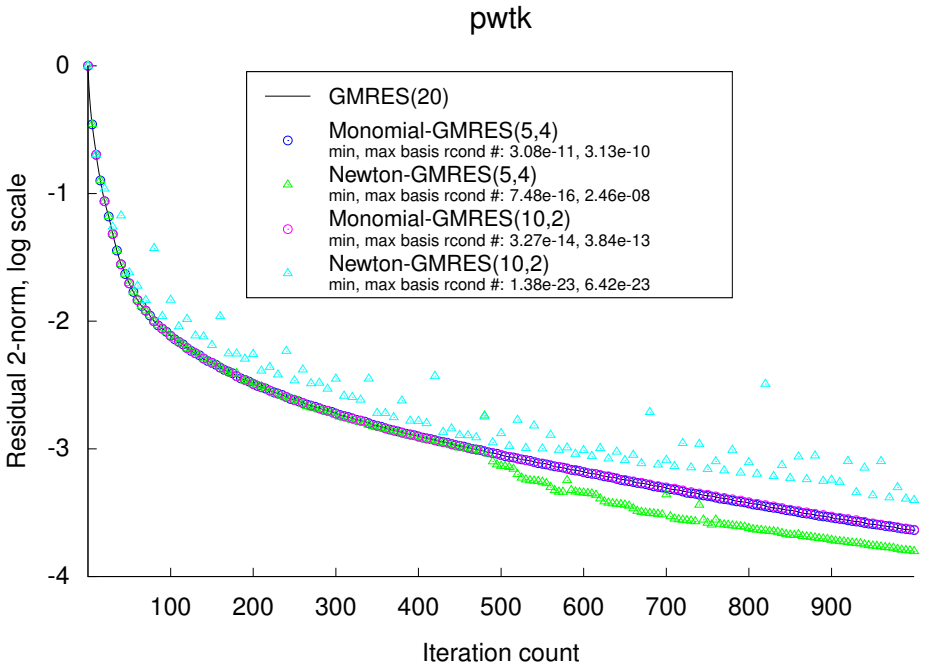

Figure 12: ●●●

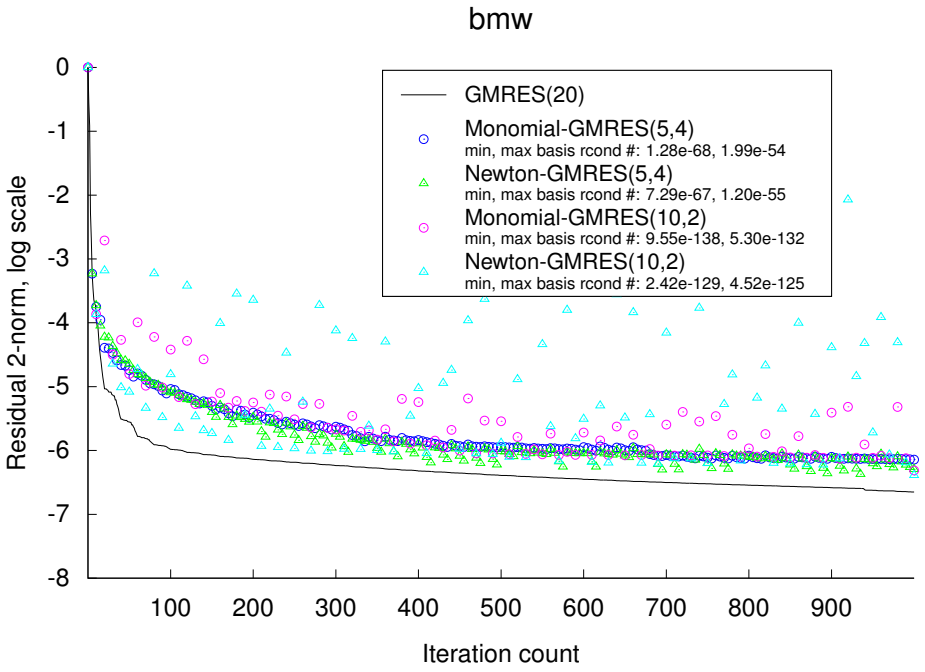

Figure 13: ●●●

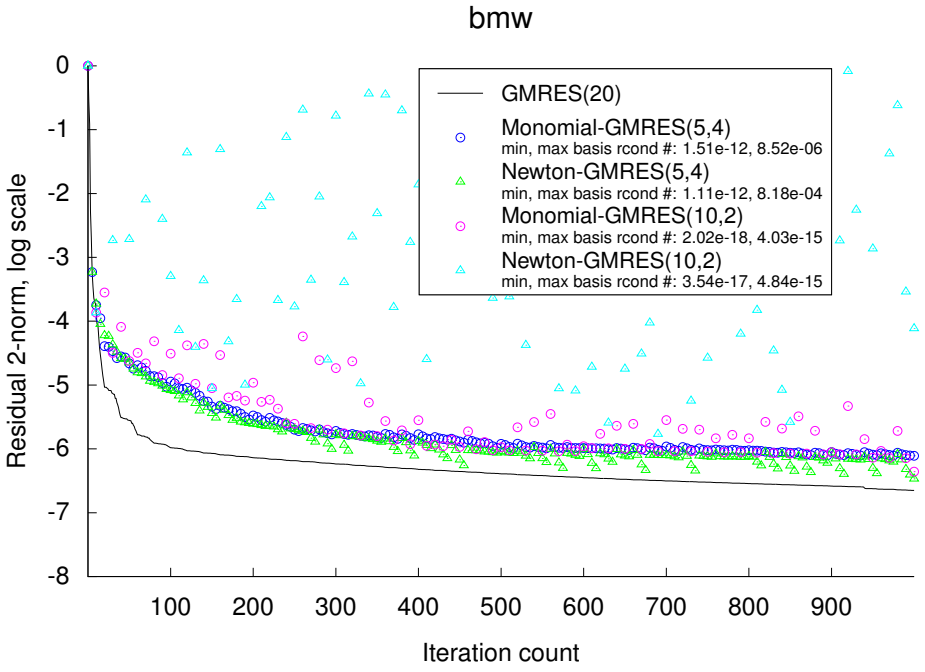

Figure 14: ●●●

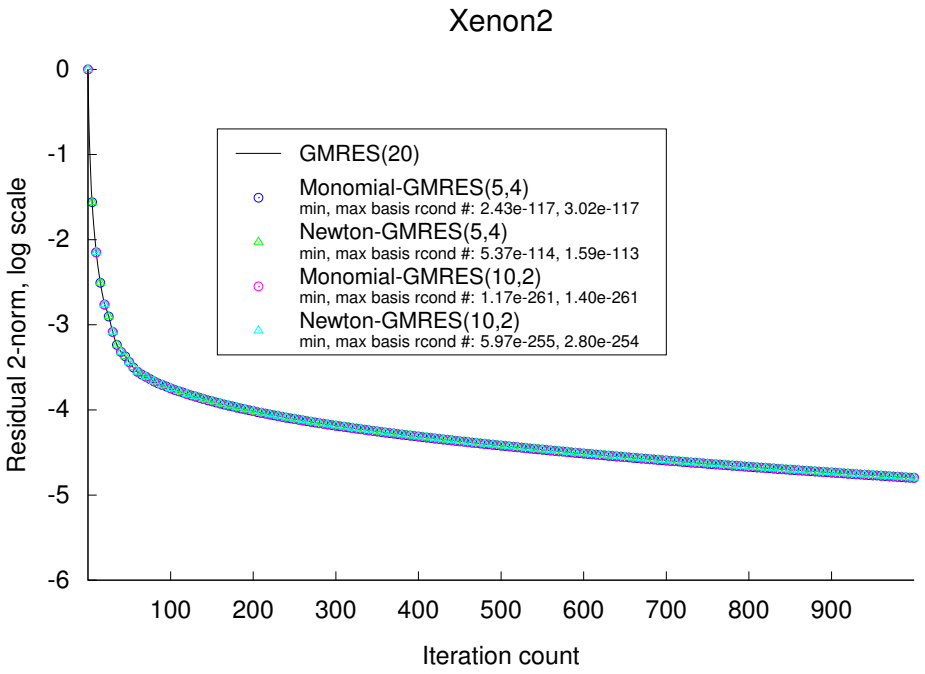

Figure 15: ●●●

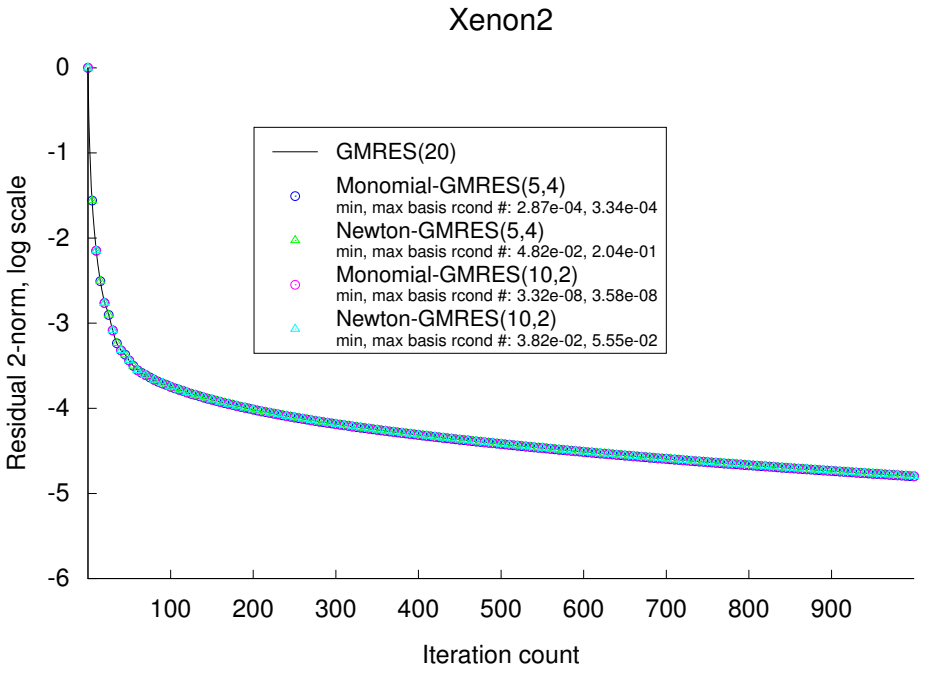

Figure 16: ●●●

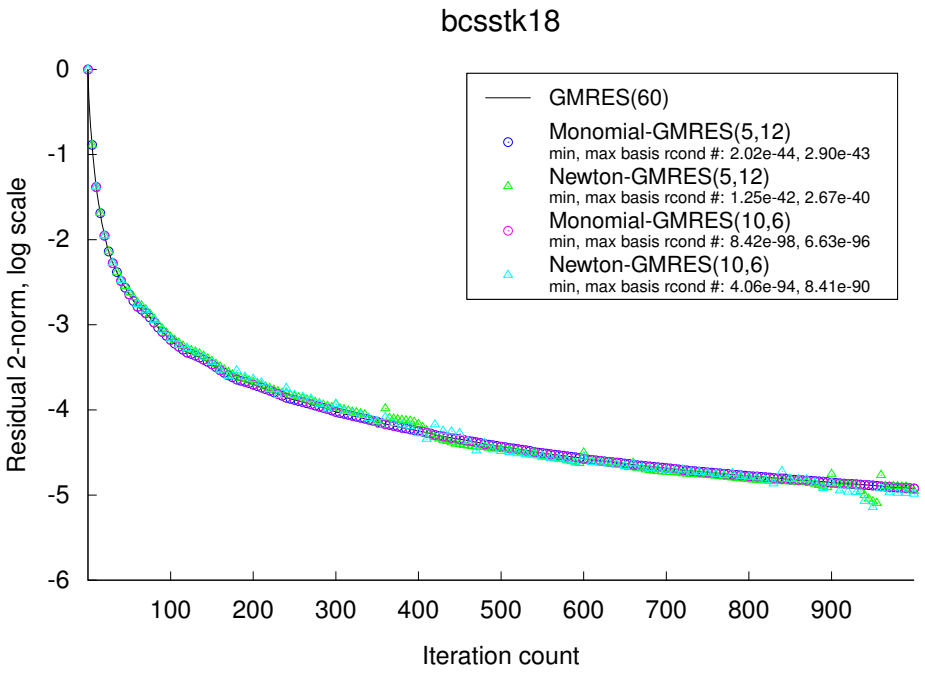

Figure 17: ●●●

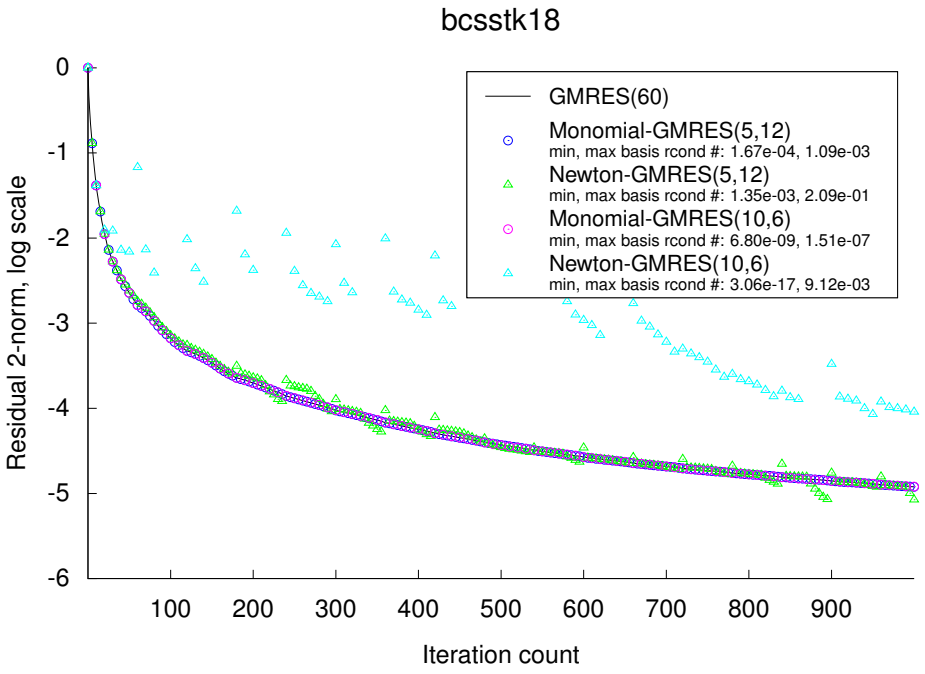

Figure 18: ●●●

8.5 Performance experiments

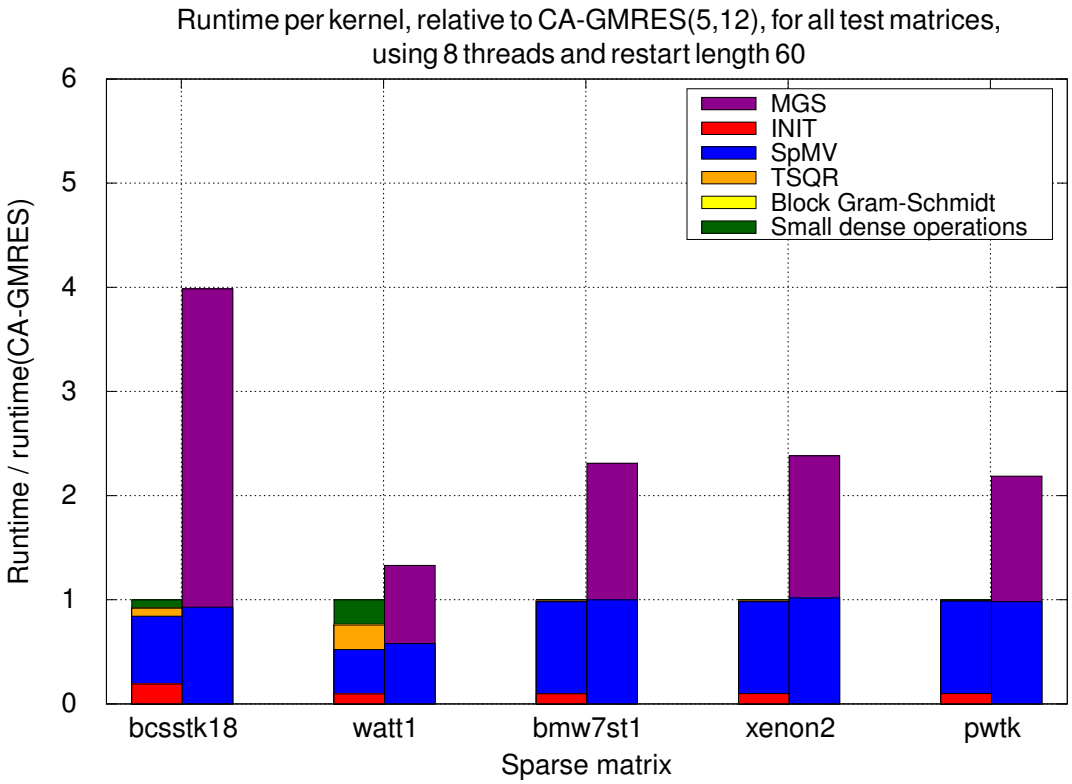

Figure 19: left CA-GMRES, right GMRES

8.5.1 Summary

Conclusion: the MPK is an important kernel and should have been implemented, also restarting with s steps of std. GMRES is not optimal and leaves room for optimization.

9 Conclusion

Krylov subspace methods (summarize this section briefly in introduction)

[2] p.191

- short description:
  Krylov subspace definition: $\mathcal{K}_k(A, r_0) = span\left\{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\right\}$

All iterative methods build and enhance a KSP with every iteration.

$$\mathbf{r}_k = \mathbf{r}_0 + \sum_{j=1}^{k} c_j A^j \mathbf{r}_0 \quad \rightarrow \quad \mathbf{x}_k = \mathbf{x}_0 + \sum_{j=0}^{k-1} c_{j+1} A^j \mathbf{r}_0$$

what is good for the power method, is bad here, bc. vectors are in theory linearly independent but too close to parallel → in machine arithmetic they become linearly dependent. Need new basis ...

- Arnoldi [2] p.192

$$AQ_k = Q_{k+1} H_{k+1,k}$$

- Summary: ([2] p. 192)
  - 1. construct an orthogonal basis for the Krylov subspace;
  - 2. define an optimality property;
  - 3. use an effective preconditioner.

[2] p.184

- short description: most stable and prominent iterative method, for sym pos def matrices only.
- Algorithm description (just the basics)
  - $\|x - x_k\|_A = \text{argmin}_{y \in \mathcal{K}_k(A,r_0)} \|x - y\|_A$
    After k-steps, x_k minimizes in the KSP the A-norm x - y (only if A is SPD, or else it's not a norm)
  - follows basic concept: $\mathbf{x}_{new} = x_{old} + constant \cdot searchdirection$ (better version *steepest descent*)
  - $\mathbf{r}_k$ is multiple of q_{k+1} (q from Arnoldi; q is not directly used in CG)
    → (1) orthogonal residuals $\mathbf{r}_i^T \mathbf{r}_k = 0, \quad i < k$
    → (2) $(x_i - x_{i-1})^T A (x_k - x_{k-1}) = 0 \quad \rightarrow \quad \Delta\mathbf{x}_i^T A \Delta\mathbf{x}_k = 0, \quad i < k$
    in other words: the corrections in $\mathbf{x}$ are orthogonal in the A-inner product, hence the term 'conjugate' in CG. The term 'gradients' comes from minimizing the energy equation/quadratic form:

$$E(x) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - b^T\mathbf{x} \rightarrow min$$

    Set the derivative (gradient) to zero, i.e. $E'(x) = 0$:

$$A\mathbf{x} - b = 0$$

    and we're back to the original problem. So minimizing energies and solving the linear equation $A\mathbf{x} = b$ are basically the same.

- H is symmetric, and therefore tridiagonal. Arnoldi simplifies to Lanczos → short 'three-term' recurrences (only have to look at a few previous orthogonal vectors, not all of them).

- p.191 what about general matrices? Any non-singular matrix A can be transformed into SPD matrix via $A^TA \rightarrow$ bad condition number $\kappa(A)$.

  {The condition number of the matrix A^TA is the square of the condition number of A [...] A large condition number both increases the number of iterations required and limits the accuracy to which a solution can be obtained. [11] p. 89 (2.7.40)} consider Krylov subspace methods that are directly based on general matrix $A \rightarrow$ GMRES

GMRES (summarize this section briefly in introduction)

- short description: general form of MINRES (=like CG it is only for symmetric matrices, but must not be PD) MINRES minimizes $\|\mathbf{r}_k\|_2$ and CG minimizes energy norm of the residual $\|\mathbf{r}_k\|_{A^{-1}}$ or the energy norm of the error $\|\mathbf{x}^* - \mathbf{x}_k\|_A$ respectively. [2] (p. 198)

- algorithm description:

- $\|b - A\mathbf{x}_k\|_2 = argmin_{y \in \mathcal{K}_k(A,r_0)} \|b - Ay\|_2$
  After k-steps, $\mathbf{x}_k$ minimizes in the KSP the ℓ^2-norm $b - Ay$

- GMRES vs. CG:
  - CG forces the residual $\mathbf{r}_k$ to be orthogonal to the Krylov subspace $\mathcal{K}_k(A, r_0)$;
  - GMRES seeks the residual with minimum ℓ_2-norm within the Krylov subspace.

- Summary:
  The main components of a single iteration of GMRES are
  1. perform a step of the Arnoldi process;
  2. update the QR factorization of the updated upper Hessenberg matrix;
  3. solve the resulting least squares problem.

References

[1] Walter Edwin Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9(1):17 – 29, 1951.

[2] Uri M. Ascher and Chen Greif. *A First Course in Numerical Methods.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2011.

[3] Z. BAI, D. HU, and L. REICHEL. A newton basis gmres implementation. *IMA Journal of Numerical Analysis*, 14(4):563–581, 1994.

[4] J. Demmel, M. Hoemmen, Y. Hida, and E. Riedy. Nonnegative diagonals and high performance on low-profile matrices from householder qr. *SIAM Journal on Scientific Computing*, 31(4):2832–2841, 2009.

[5] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. Communication-optimal parallel and sequential qr and lu factorizations. *SIAM J. Sci. Comput.*, 34(1):206–239, February 2012.

[6] Jocelyne Erhel. A parallel gmres version for general sparse matrices. *Electronic Transactions on Numerical Analysis*, 3:160–176, 1995.

[7] A. Greenbaum, M. Rozloznik, M. Rozlo Zn Ik, and Z. Strakos. Numerical behaviour of the modified gram-schmidt gmres implementation, 1997.

[8] Laura Grigori and Sophie Moufawad. Communication avoiding ilu0 preconditioner. *SIAM Journal on Scientific Computing*, 37:C217–C246, 04 2015.

[9] Mark Hoemmen. *Communication-avoiding Krylov Subspace Methods*. PhD thesis, Berkeley, CA, USA, 2010. AAI3413388.

[10] D. Nuentsa Wakam and G.-A. Atenekeng Kahou. Parallel GMRES with a multiplicative schwarz preconditioner. *ARIMA*, 14:81–99, 2011.

[11] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

[12] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, July 1986.

[13] H. Walker. Implementation of the gmres method using householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 9(1):152–163, 1988.

List of Algorithms

Appendices

Appendix A

discussion of test matrices

Name	Dim	nnz
bmw	141K	3.7M
bcsstk18	12K	149K
pwtk	218K	3.7M
watt1	1856	11550
xenon2	157K	3.9M