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 Notation

For linear algebra, similar notation is considered as in [5] and [4].

- 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_k is a matrix consisting of s vectors $[v_1, v_2, \ldots, v_s]$, $\underline{V}_k$ comprises vectors $[V_k, v_{s+1}]$. The underline means one more column at the end.
- If again, V_k is a matrix consisting of s vectors $[v_1, v_2, \ldots, v_s]$, $\acute{V}$ consists of vectors $[v_2, v_3, \ldots, v_s]$. 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.

Abstract

2 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 [3]

3 Related work

s-step methods, CA-ILU(0)

4 Computational kernels

In this thesis *computational kernels* define the parts of an algorithm with the highest costs. These costs include both arithmetic operations and communication. 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 6.2.1) has to balance between communication and redundant computations; CA-GMRES (section 6) incorporates additional techniques to deal with ill-conditioned basis vectors.

4.1 Matrix powers kernel (MPK)

[5] p.60
The Matrix Powers Kernel Power iteration, SpMV instead of MV, sparse matrix like a graph $\rightarrow$ spacial, temporal locality not as efficiently used as in dense MV. Avoid communication by sending / receiving all necessary values beforehand (look at reachability of graph(A)) and computing s basis vectors without further communication.

Although, the matrix powers kernel was not implemented in the context of this thesis it is still an essential kernel to avoid communication and therefore, will be briefly summarized here. The matrix powers kernel replaces the sparse matrix-vector products that generate the basis for the Krylov subspace $\mathcal{K}_m(A, v) = \left[v, Av, A^2v, \ldots, A^{m-1}v\right]$. One invocation of

the MPK produces the same amount of basis vectors as s sparse matrix-vector products. The data and the workload are partitioned among P processors, where each processor gets $A(\alpha,:)$ and $v_0(\alpha)$ with α being the subset assigned to the specific processor. by a factor $\Theta(s)$ and the matrix has to be loaded from slow to fast memory only once instead of s. than each processor is assigned a part α of the input vector v_0 and $A(\alpha,:)$.
To minimize communication in a parallel setting, the s monomial basis vectors of the Krylov subspace $[y, Ay, A2y, \ldots, A^s y]$ are computed with no communication using the so-called matrix powers kernel [13]. This requires ghosting and computing redundantly on each processor the data required for computing its part of the vectors with no communication. Note that throughout this paper we use the term ghosting to denote the storage of redundant data, of vectors or matrices, that do not belong to the processor's assigned domain or part, but are needed for future computations.
First, the data and the work is split between P processors. Each processor is assigned a part α of the input vector y_0 ($y_0(\alpha)$) and $A(\alpha,:)$, where $\alpha \subseteq V(G(A))$. Then, each processor has to compute the same part α of $y_1 = Ay_0$, $y_2 = Ay_1$, till $y_s = Ay_{s-1}$ without communicating with other processors. To do so, each processor fetches all the data needed from the neighboring processors, to compute its part α of the s vectors. Thus, to compute $y_s(\alpha)$, each processor should receive the missing data of $y_0(\eta_s)$ and $A(\eta_s,:)$ from its neighboring processors and store it redundantly, where $\eta_s = R(G(A), \alpha, s)$. Finally, each processor computes the set $R(G(A), \alpha, s-i)$ of the vectors y_i for $i = 1, 2, \ldots, s$ without any communication with the other processors.

4.2 Tall and skinny QR (TSQR)

Unconditionally stable like Householder QR but less communication

4.3 Block Classical Gram-Schmidt (BCGS)

Dense matrix-matrix multiplications → less communication (factor $\Theta(s)$ fewer messages) than unblocked CGS. Also, BCGS requires less communication that BMGS. No reorthogonalization because solving linear system with (std. GMRES uses unblocked MGS-Arnoldi) CA-GMRES uses TSQR which improves orthogonality of the block columns.

5 Arnoldi iteration

first presented in *REF[[W. E. Arnoldi, The principle of minimized iterations in the solution of the matrix eigenvalue problem, Q. Appl. Maths, 9 (1951), pp. 17–29.]]*

s steps of standard Arnoldi produce an $s+1 \times s$ upper Hessenberg Matrix $\underline{H}$ and $m \times s+1$ orthonormal vectors q_1, q_2, …, q_s, q_{s+1} that form a basis for the Krylov subspace $\mathcal{K}_{s+1}(A, r)$ such that $AQ = \underline{Q}\underline{H}$. There are many ways to orthogonalize successive basis vectors. Modified Gram-Schmidt (MGS) is often employed because it provides high

numerical stability, however, although not as stable, Classical Gram-Schmidt (CGS) is more suited for parallel implementations because it provides fewer synchronization points. introduction.. produces $AQ = QH$

5.1 Arnoldi(s)

[238] H. F. Walker, Implementation of the GMRES and Arnoldi methods using Householder transformations, Tech. Rep. UCRL-93589, Lawrence Livermore National Laboratory, Oct. 1985.
source above! At the heart of Arnoldi's method is a Gram-Schmidt process, and so we refer to the GMRES implementation of [8] as the Gram-Schmidt implementation. The basic form of Arnoldi's method given above employs the classical Gram-Schmidt process, which is numerically untrustworthy. Because of roundoff, there may be severe loss of orthogonality among the computed Vm'S. In practice, it is usual to implement Arnoldi's method using the modified Gram-Schmidt process (see Golub and Van Loan [4]). Mathematically, this is just a rearrangement of the classical process; computationally, it has superior properties. Even the modified Gram-Schmidt process can fail to perform well if the vectors on which it operates are not sufficiently independent. Indeed, if S (sl,... ,sin) is an n m matrix the columns of which are to be orthonormalized and if Q (ql,. qm) is the computed result of applying modified Gram-Schmidt to the columns of S using floating point arithmetic with unit rounding error u, then Bjorck [1] has shown that (1.6) QTQ I + E, [IEll2 where the condition number 2(S) is the ratio of the largest singular value of S to the smallest. It follows that at the mth step of Arnoldi's method using modified GramSchmidt, Vm+ may have a significantly nonzero conponent in the span of vl,. Vm if 2((vl,...,vm,Avm)) is large, i. e., if Avm is nearly in the span of vl,...,Vm. Shad [7, p. 214 has suggested that the Gram-Schmidt process in Arnoldi's method may be an important source of errors in the full and incomplete orthogonalization methods ([6],[7]), which are related to GMRES.

5.1.1 The Monomial basis

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

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

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 $\sigma_1, \ldots, \sigma_s$. 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. In theory, the basis is still linearly independent in exact arithmetic. In machine

precision they become inevitably dependent at a certain point.

scaling the rapidly growing condition number and vector length → scale vector to lengh 1, but: → need a different basis.

5.1.2 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 be 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. [2] suggest that complex arithmetic can be avoided by setting

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

and

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

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. [5] observed that the runtime of the Newton basis was still close to the runtime of the Monomial basis.
[5] 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.

5.2 Arnoldi(s,t)

5.2.1 Introduction

5.2.2 Scaling the first basis vector

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

- 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)

5.2.3 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:

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

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 $s-1 \times s-1$.

5.2.4 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:

$$\underline{\mathfrak{B}}_k = \begin{pmatrix} \mathfrak{H}_{k-1} & 0_{sk,s} \\ h_{k-1} e_1 e_{sk}^T & \underline{B}_k \end{pmatrix} \tag{3}$$

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

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

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,

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

$$\underline{\mathfrak{H}}_{k-1,k} := -\mathfrak{H}_{k-1} \mathfrak{R}_{k-1,k} R_k^{-1} + \underline{\mathfrak{R}}_{k-1,k} \underline{B}_k R_k^{-1} \tag{4}$$

A different approach In order to get better Eigenvalue approximations Erhel [3] computed $2s$ Ritz values, picked s out of them and applied the Modified Leja Ordering. This improved the condition of the Newton basis and lead to (better) convergence. [5] 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. This might not always be the case. E.g., the $2s$ Ritz values could consist of complex conjugate pairs only. If s is odd, there is no way this would fit without splitting a complex conjugate pair. Remember that in order to avoid complex arithmetic the consecutive order of a complex conjugate pair must be preserved. A way to address this could be to incorporate all computed Ritz values, i.e. to apply $2s$ Ritz values over two outer iterations of Arnoldi(s,t). This slightly changes the way (4) is computed. 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 (3) by an additional entry above $\underline{B}_k$.

$$\underline{\mathfrak{B}}_k = \begin{pmatrix} \mathfrak{H}_{k-1} & -e_{sk}e_1^T\Im(\theta_s)^2 \\ h_{k-1}e_1e_{sk}^T & \underline{B}_k \end{pmatrix}. \tag{5}$$

(4) then changes to

$$\underline{\mathfrak{H}}_{k-1,k} := -\mathfrak{H}_{k-1}\mathfrak{R}_{k-1,k}R_k^{-1} + \underline{\mathfrak{R}}_{k-1,k}\underline{B}_kR_k^{-1} + e_{sk}e_sg \tag{6}$$

with

$$g := -\Im(\theta_s)^2e_se_1^TR_k^{-1}.$$

This approach was addressed only theoretically here and therefore, is subject to further investigation.

6 CA-GMRES

6.1 GMRES(m)

Algorithm 1 restarted 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 $y_k := \text{argmin}_y \|\underline{H}_k y - \beta e_1\|_2$
14: Set $x_0 := x_0 + Q_k y_k$
15: **end while**

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

$$z_k := \text{argmin}_z \|b - A(x_0 + z)\|_2 \tag{7}$$

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

$$\begin{aligned} \text{argmin}_y \|r_0 - \underline{Q}\underline{H}y\|_2 &= \text{argmin}_y \|\underline{Q}^T r_0 - \underline{H}y\|_2 \\ &= \text{argmin}_y \|\beta e_1 - \underline{H}y\|_2 \end{aligned} \tag{8}$$

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.

The CA-GMRES algorithm solves a different least-squares problem than (8):

$$\text{argmin}_y \left\|\beta e_1 - \underline{RB}R^{-1}y\right\|_2 \tag{9}$$

Algorithm 2 restarted Newton CA-GMRES

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 $\acute{V}_k$ where $v_{i+1} = (A - \theta_i I)v_i, i = sk + 1 : sk + s$
      $\underline{\acute{\mathfrak{R}}}_{k-1,k} := \underline{\mathfrak{Q}}_{k-1}^T \underline{\acute{V}}_k$
      $\underline{\acute{V}}_k' := \underline{\acute{V}}_k - \underline{\mathfrak{Q}}_{k-1} \underline{\acute{\mathfrak{R}}}_{k-1,k}$
      Compute QR factorization of $\underline{\acute{V}}_k' \to \underline{\acute{Q}}_k \underline{\acute{R}}_k$ 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 $H_k := R_k B_k R_k^{-1} + \tilde{\rho}_k^{-1} b_k z_k e_s^T - h_{k-1} e_1 e_{s(k-1)}^T \mathfrak{R}_{k-1,k} R_k^{-1}$
      Compute $h_k := \tilde{\rho}_k^{-1} \rho_k b_k$
      $\underline{\mathfrak{H}}_k := \begin{pmatrix} \mathfrak{H}_{k-1} & \underline{\mathfrak{H}}_{k-1,k} \\ h_0 e_1 e_{sk}^T & H_k \\ 0_{1,sk} & h_k e_s^T \end{pmatrix}$
      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

6.2 Preconditioning

Left, right, split, we consider left preconditioning ($M^{-1}Ax = M^{-1}b$) only. Scaling is a special type of preconditioning. [5] 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 [5] found that for practical problems, equilibration established to be quite effective and almost made the basis type irrelevant. We observed something similar after applying the ILU(0) preconditioner to the system.

6.2.1 CA-ILU(0) preconditioner

$M = LU$
Algorithm 2 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 [4] (can be very long or short, dependent on overall length)

6.3 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.

6.4 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

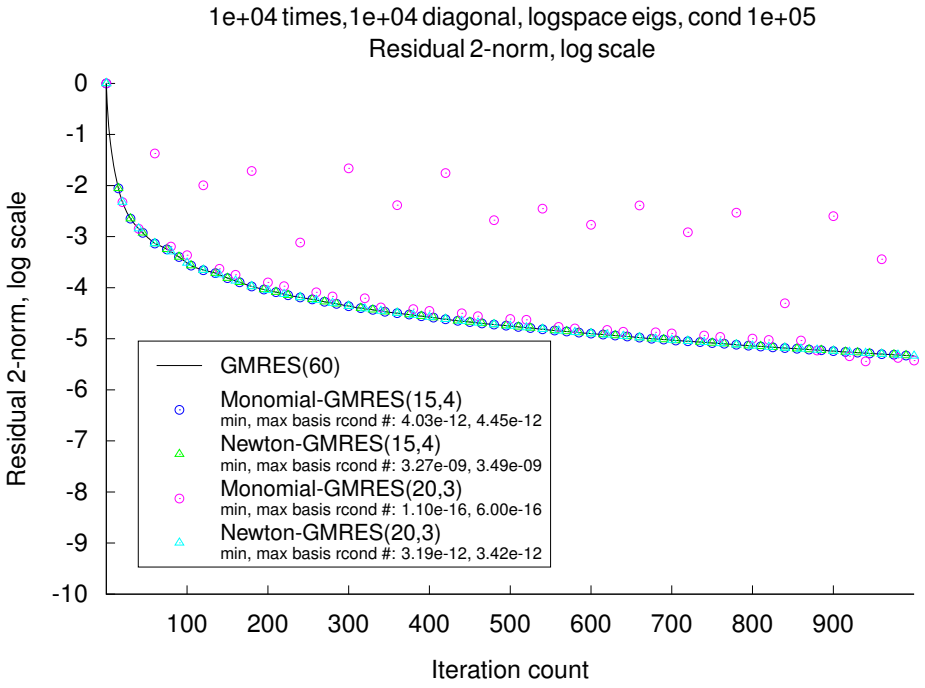

Figure 1: ●●●

• 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 [5].

6.5 Numerical experiments

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).

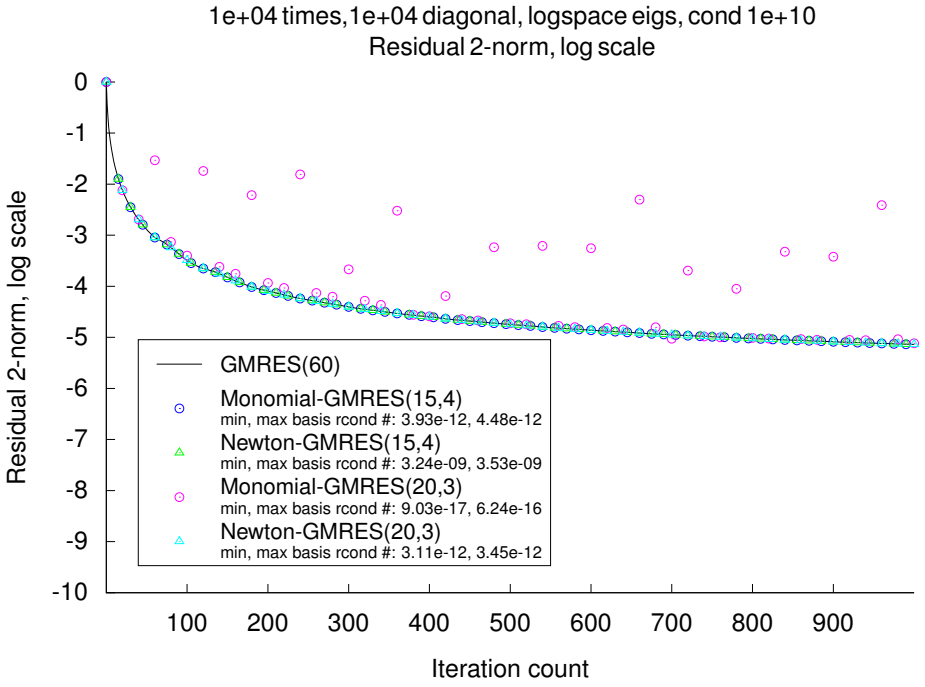

Figure 2: ●●●

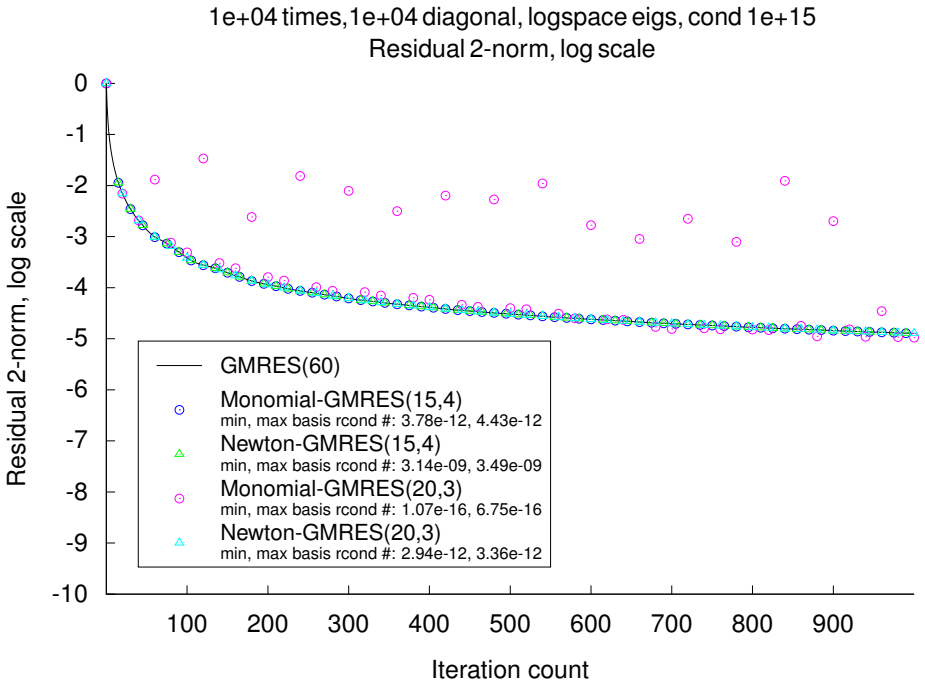

Figure 3: ●●●

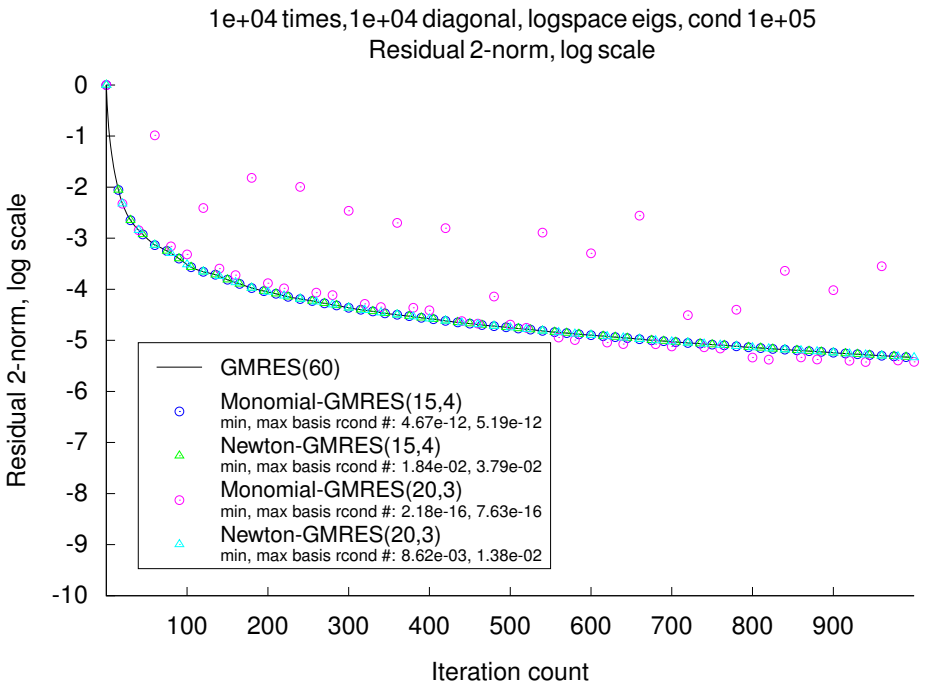

Figure 4: ●●●

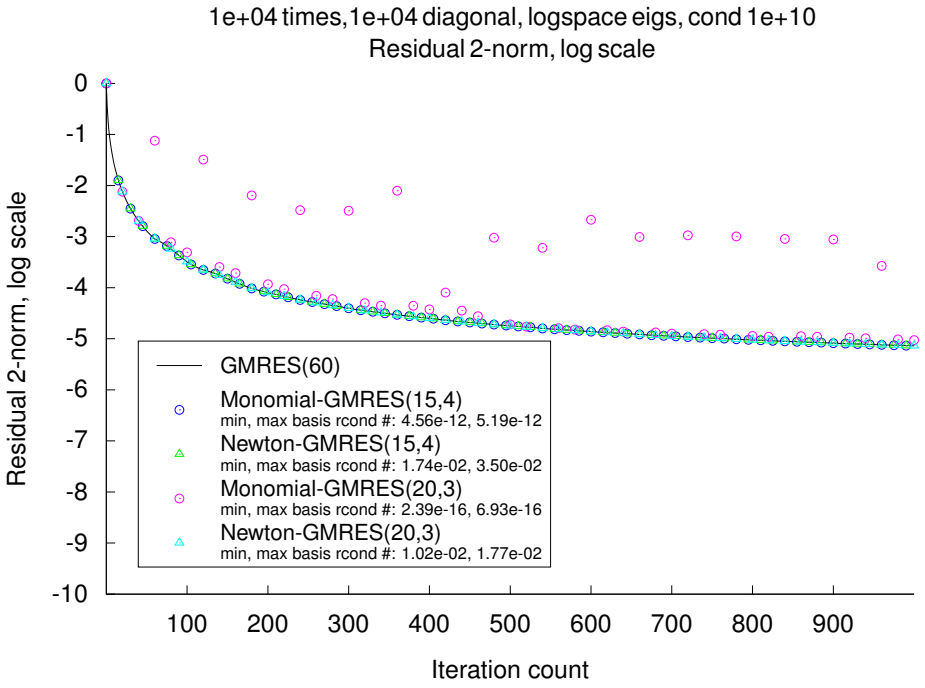

Figure 5: ●●●

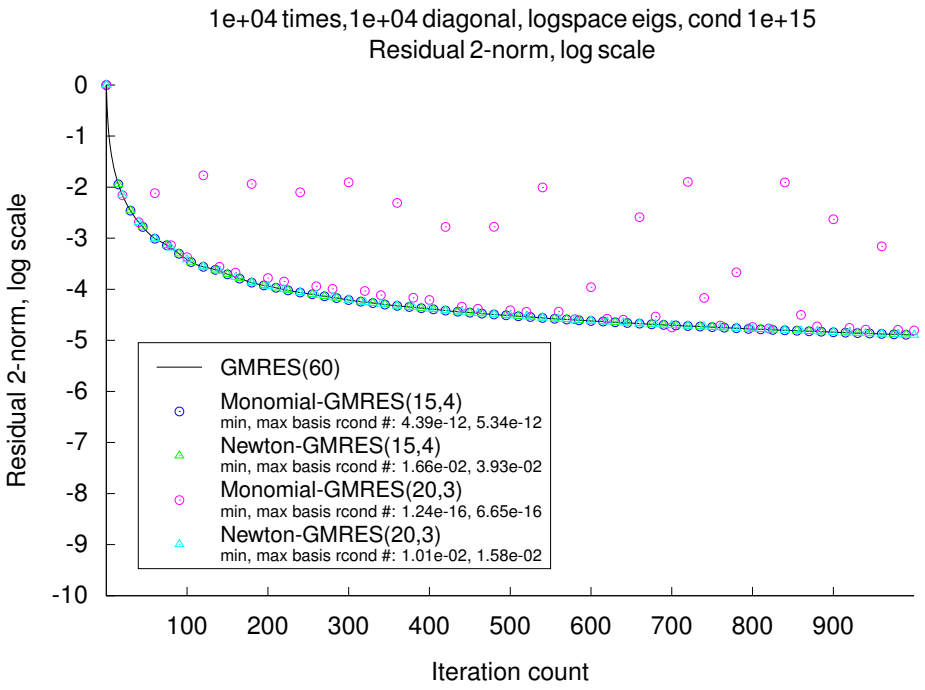

Figure 6: ●●●

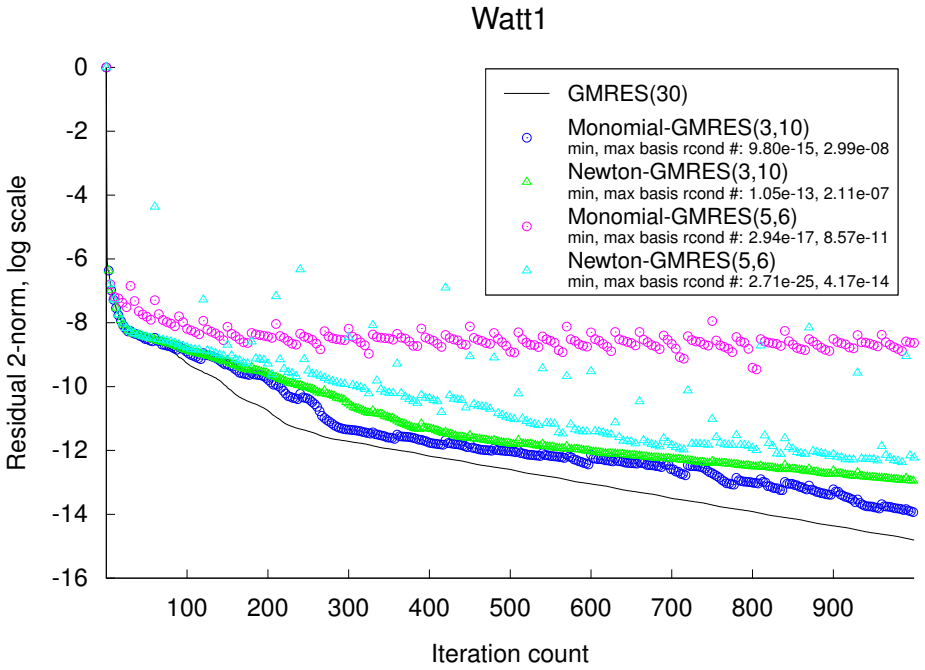

Figure 7: not scaled

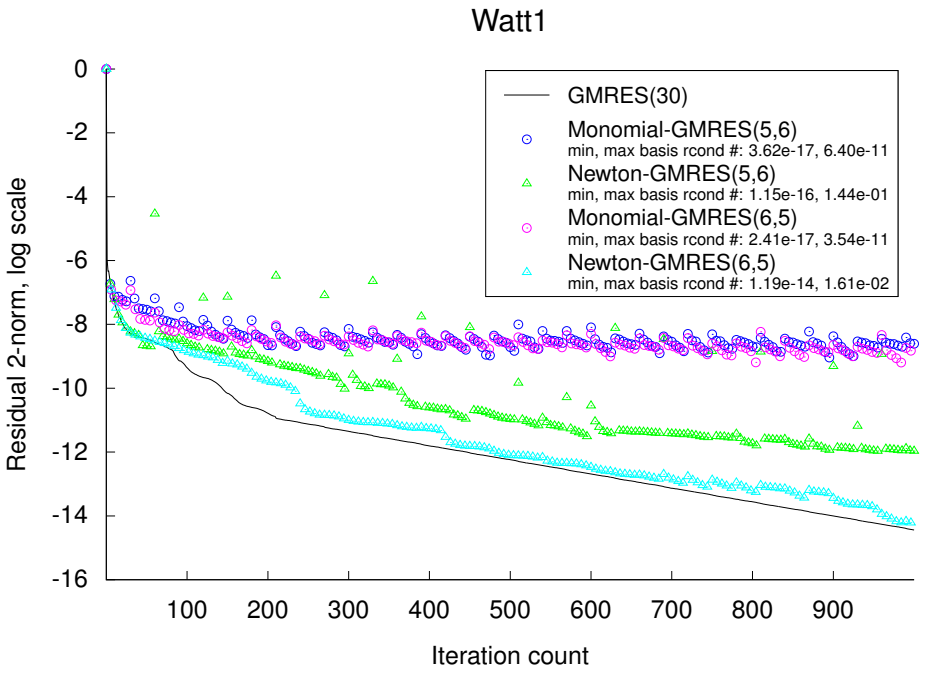

Figure 8: ●●●

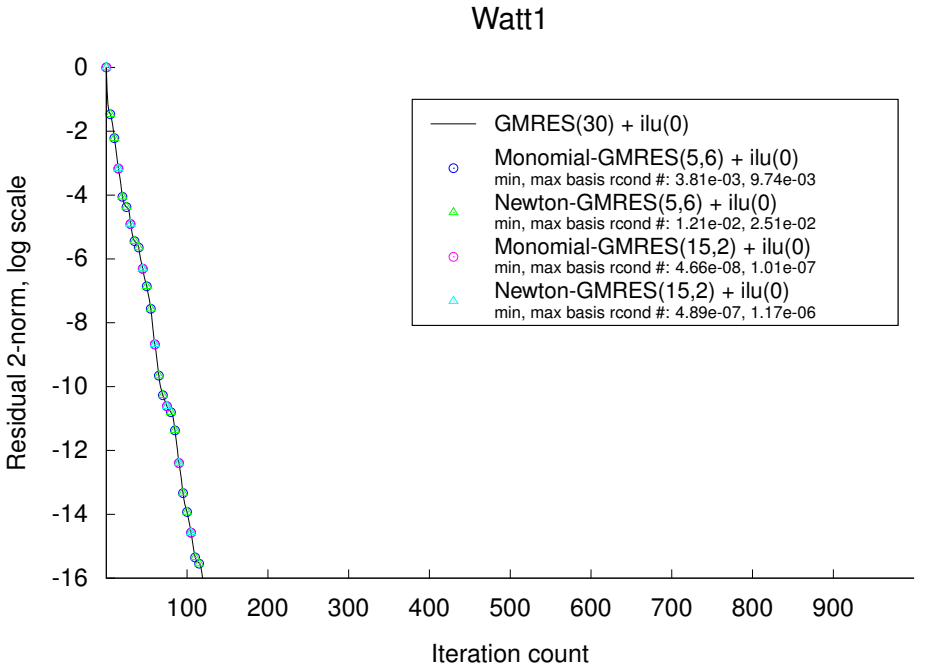

Figure 9: without basis vector scaling

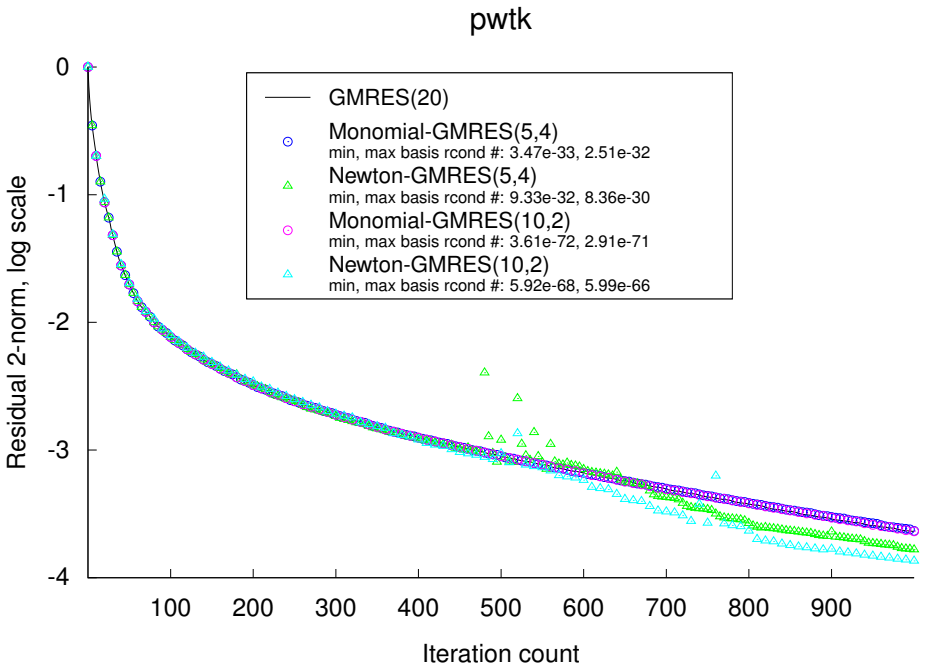

Figure 10: ●●●

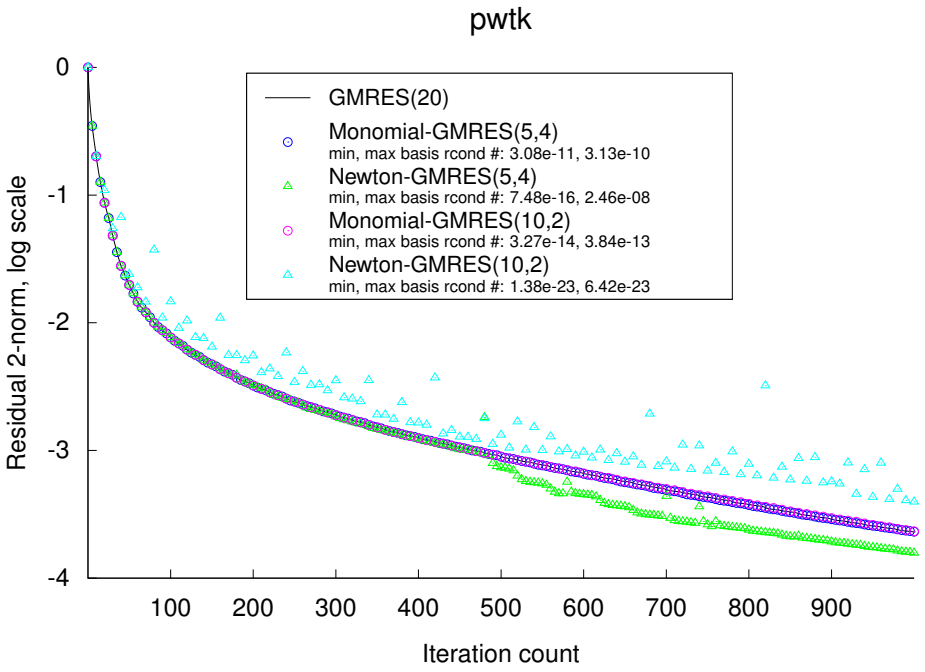

Figure 11: ●●●

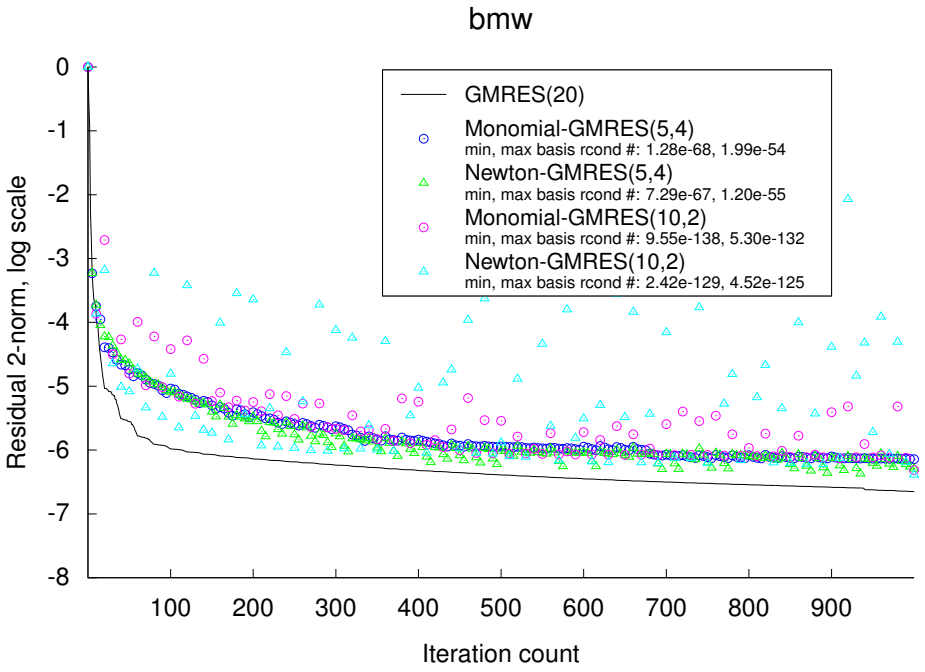

Figure 12: ●●●

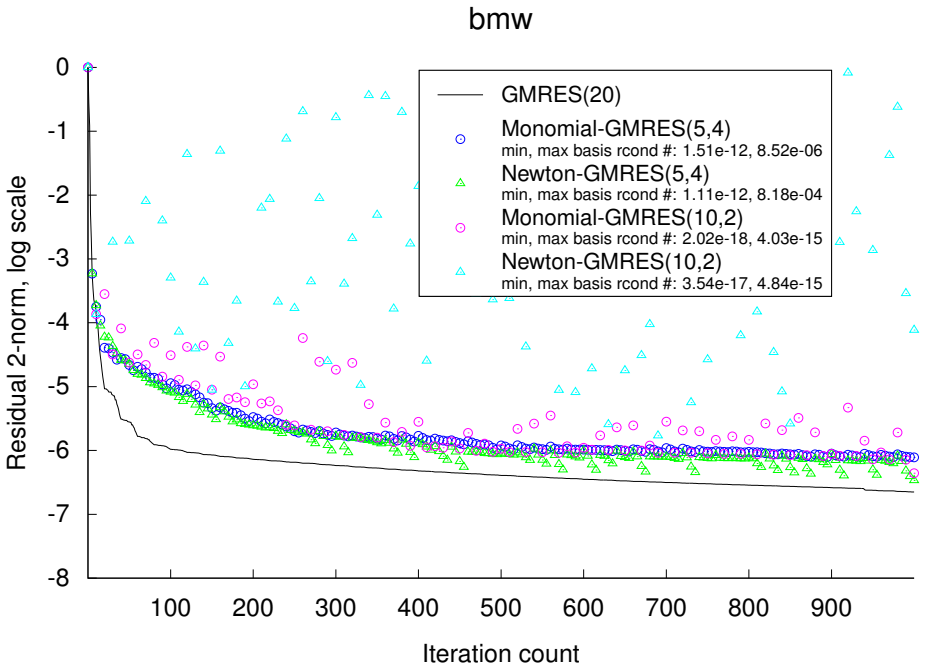

Figure 13: ●●●

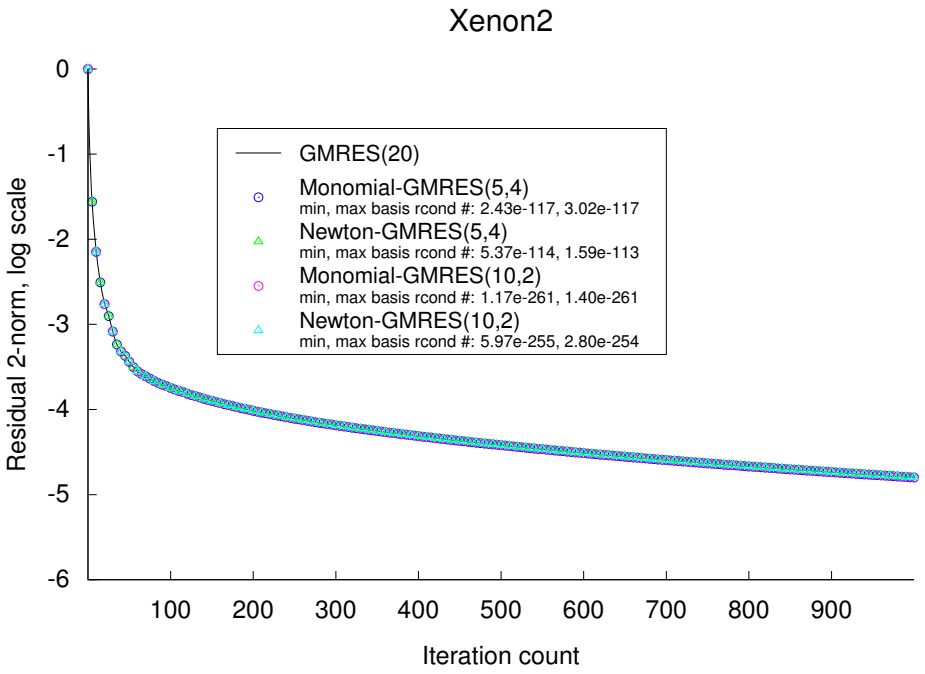

Figure 14: ●●●

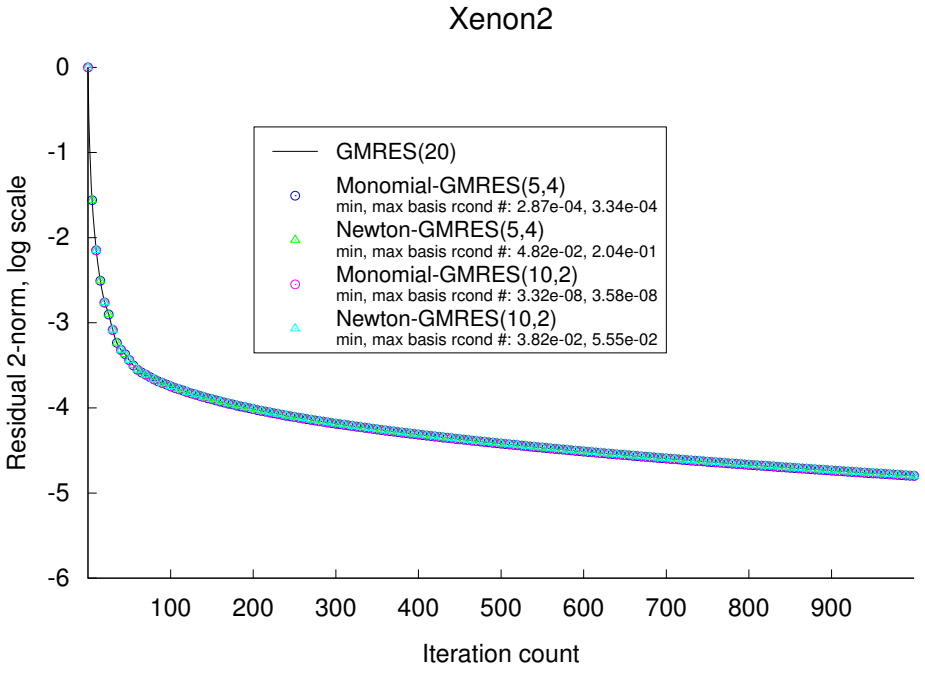

Figure 15: ●●●

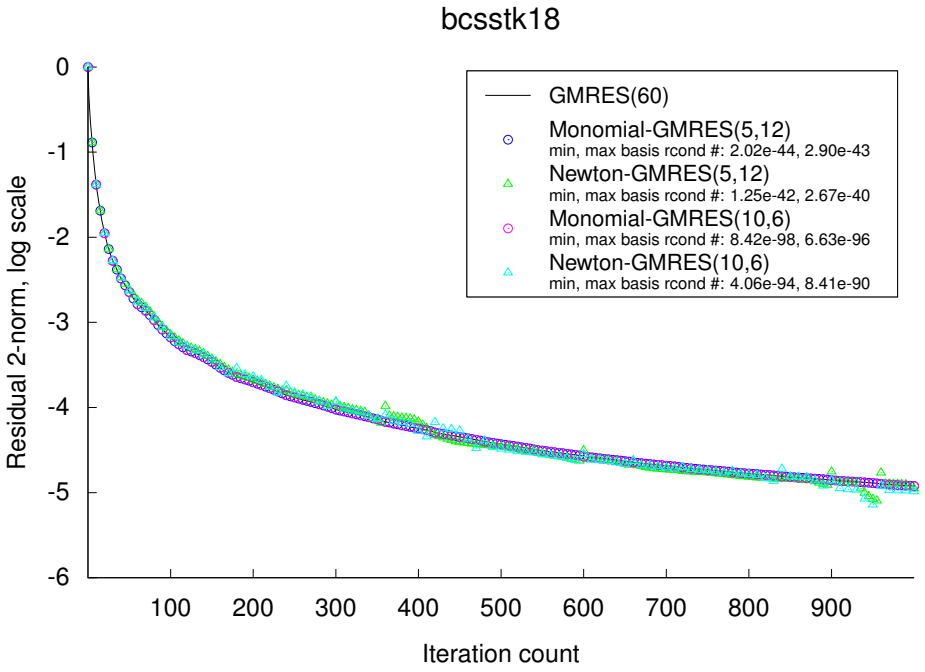

Figure 16: ●●●

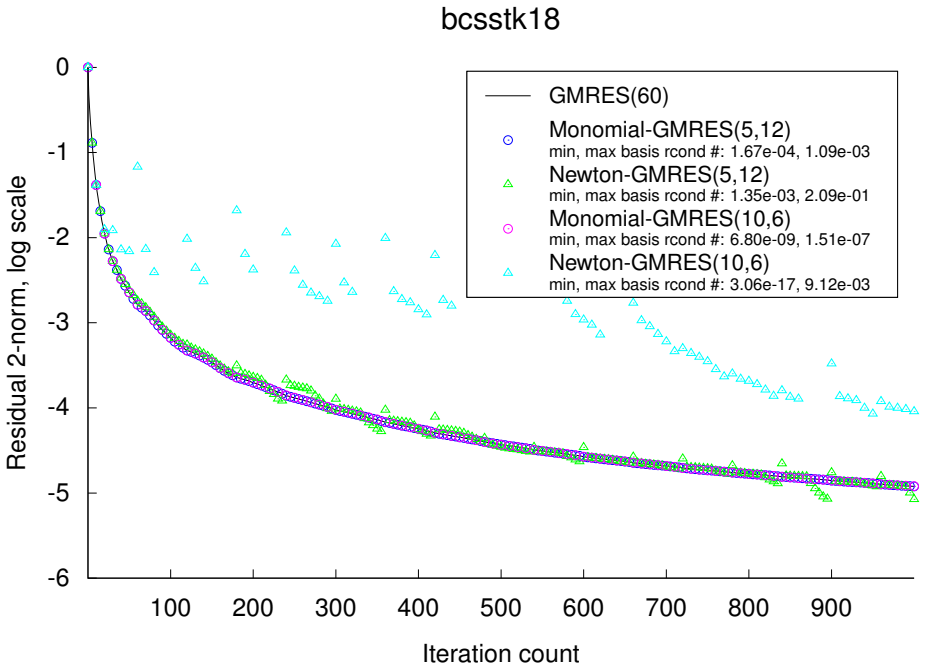

Figure 17: ●●●

6.6 Performance experiments

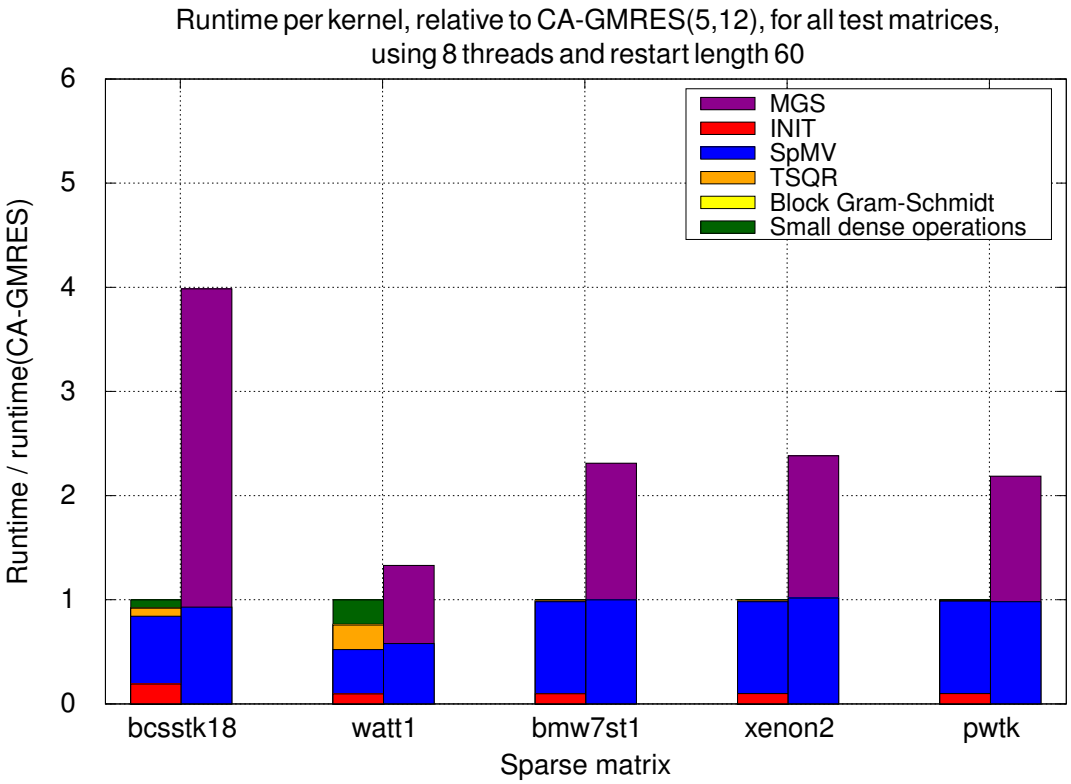

Figure 18: left CA-GMRES, right GMRES

6.6.1 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.

7 Conclusion

Krylov subspace methods (summarize this section briefly in introduction)

[1] 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 [1] p.192

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

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

[1] 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. [6] 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. [1] (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] Uri M. Ascher and Chen Greif. *A First Course in Numerical Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2011.

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

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

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

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

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

List of Algorithms

Appendices

Appendix A

discussion of test matrices