

# **Implementation of the Deflated Variants of the Conjugate Gradient Method**

## **Implementace deflated verzí sdružených gradientů**



# Zadání diplomové práce

Student: **Bc. Jakub Kružík**  
Studijní program: N2658 Výpočetní vědy  
Studijní obor: 2612T078 Výpočetní vědy  
Téma: **Implementace deflated verzí sdružených gradientů**  
**Implementation of the deflated variants of the conjugate gradient method**  
Jazyk vypracování: čeština

## Zásady pro vypracování:

Významného zlepšení rychlosti konvergence metody sdružených gradientů (CG) používaných k iteračnímu řešení soustavy lineárních rovnic se symetrickou pozitivně definitní maticí může být dosaženo rozšířením Krylovových podprostorů o vektory, které aproximují vlastní vektory odpovídající vlastním číslům blízkým nule [1] nebo definují seskupování uzlů do skupin např. podle doménové dekompozice [2], tzv. deflated verze CG (DCG). Experimenty s různými variantami DCG potvrzují výraznou akceleraci řešení v porovnání se standardní CG metodou nejen pro klasickou soustavu rovnic, ale zejména pro řešení soustav s více pravými stranami.

Student by se měl seznámit s variantami DCG včetně předpokládání, související Deflated Lanczosovou metodou, implementovat je (s využitím Matlabu, PETSc apod.) a provést jejich porovnání. Součástí práce by měla být diskuze nad jejich možnými paralelizacemi (výpočetní náročnost, množství komunikace, paměťové nároky).

## Seznam doporučené odborné literatury:

[1] Y. Saad, M. Yeung, J. Erhel, F. Guyomarc'h: A deflated version of the Conjugate Gradient Algorithm, Research report from Dep. of Computer Science and Engineering, University of Minnesota, 1998.

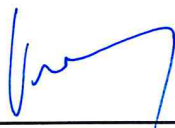
[2] R. Löhner, F. Mut, J. R. Cebal, R. Aubry and G. Houzeaux: Deflated preconditioned conjugate gradient solvers for the pressure-Poisson equation: Extensions and improvements, Int. J. Numer. Meth. Engng 2011; 87:2–14.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Ing. David Horák, Ph.D.**

Datum zadání: **31.10.2017**

Datum odevzdání: **21.5.2018**



doc. Mgr. Vít Vondrák, Ph.D.  
vedoucí katedry



Ing. Zdeňka Chmelíková, Ph.D.  
prorektorka pro studium



I hereby declare that this master's thesis was written by myself. I have referenced all the sources and publications I have drawn upon.

Ostrava, 21st May 2018

.....*Jakub Kravinský*.....



I would like to express my thanks to doc. Ing. David Horák, Ph.D., the supervisor of this thesis, for his invaluable help, ideas, advice and suggestions that significantly improved this work.

My thanks belong to the entire PERMON team for all their help, and for creating a great working environment.

I am also grateful to doc. Ing. Dalibor Lukáš, Ph.D. for providing me with a boundary element method benchmark.

Heartfelt thanks go to my family and friends for their continued support.

I would also like to thank my former employer IT4Innovations National Supercomputing Center, VSB – Technical University of Ostrava and my current employer Institute of Geonics of the Czech Academy of Sciences for supporting my work on this thesis and for providing me with the opportunities to learn and to work on exciting projects.

This work made use of the facilities of ARCHER, the UK's national high-performance computing service, provided by The Engineering and Physical Sciences Research Council (EPSRC), The Natural Environment Research Council (NERC), EPCC, Cray Inc. and The University of Edinburgh under the PRACE Distributed European Computing Initiative (DECI-14) project PERMON.





## Abstrakt

Sdružené gradienty jsou jednou z nejpoužívanějších metod pro řešení rozsáhlých soustav lineárních rovnic se symetrickou pozitivně-semidefinitní maticí. Jeden ze způsobů urychlení konvergence metody je deflace. Principem deflace je skrývání té části spektra matice, která způsobuje zpomalení konvergence. Tato diplomová práce se zabývá efektivní implementací různých deflated verzí sdružených gradientů. Velká pozornost je také věnována teorii a volbě deflačního prostoru. Možnosti implementace jsou demonstrovány na rozsáhlém množství příkladů.

**Klíčová slova:** deflace, předpodmínění projektorem, sdružené gradienty, deflatované sdružené gradienty, DCG, CG, waveletová komprese, multigrid, hrubý problém, Krylovův podprostor

## Abstract

The conjugate gradient algorithm is one of the most popular methods for the solution of large systems of linear equations with symmetric positive semi-definite matrix. One of the schemes accelerating the convergence of conjugate gradients is deflation which effectively hides parts of the matrix spectrum that slows down the convergence. This master's thesis deals with efficient parallel implementation of the deflated conjugate gradient method with various modifications. Detailed theoretical considerations and the crucial choice of the deflation space are also discussed. The implementation is showcased on a wide range of benchmarks.

**Keywords:** deflation, preconditioning by projector, conjugate gradient, deflated conjugate gradient, DCG, CG, wavelet compression, multigrid, coarse problem, Krylov subspace



# Contents

<b>1</b>	<b>Introduction</b>	<b>11</b>
<b>2</b>	<b>Conjugate Gradient Method</b>	<b>13</b>
2.1	Steepest Descent Method . . . . .	13
2.2	Minimize over Subspace . . . . .	16
2.3	Lanczos Method . . . . .	17
2.4	Using Lanczos Method to Solve Linear Systems . . . . .	18
2.4.1	Recursive Formulation . . . . .	19
2.4.2	Conjugacy of Gradients and Directions . . . . .	21
2.5	Conjugate Gradients Algorithm . . . . .	22
2.5.1	From CG back to Lanczos . . . . .	23
2.5.2	CG Convergence . . . . .	25
2.5.3	Preconditioned CG . . . . .	26
<b>3</b>	<b>Deflated Conjugate Gradients</b>	<b>29</b>
3.1	Deriving DCG . . . . .	29
3.1.1	Preconditioned DCG . . . . .	31
3.2	Preconditioning Effect of Deflation . . . . .	32
3.2.1	Shifting the Eigenvalues . . . . .	34
3.3	DCG Coarse Problem . . . . .	35
3.3.1	Required Accuracy for CP Solution . . . . .	35
3.3.2	Nested DCG . . . . .	36
<b>4</b>	<b>Deflation Spaces</b>	<b>37</b>
4.1	Eigenvectors . . . . .	37
4.2	Subdomain Aggregation . . . . .	37
4.3	Discrete Wavelet Compression . . . . .	38
4.4	Multigrid Prolongation and Restriction . . . . .	42
<b>5</b>	<b>Libraries and HPC environment</b>	<b>43</b>
5.1	PETSc . . . . .	43
5.2	PERMON . . . . .	43
5.3	SuperLU . . . . .	43
5.4	SLEPc . . . . .	43
5.5	MFEM . . . . .	44
5.6	ARCHER . . . . .	44
<b>6</b>	<b>Implementation</b>	<b>45</b>
6.1	Solver Settings . . . . .	45
6.1.1	Setting the Deflation Matrix . . . . .	45

6.2	Cost of Matrices Assembly and Operations . . . . .	46
<b>7</b>	<b>Numerical Experiments</b>	<b>51</b>
7.1	Benchmarks . . . . .	51
7.1.1	SuiteSparse Matrix Collection . . . . .	51
7.1.2	MFEM Example 1: Laplace Equation . . . . .	51
7.1.3	MFEM Example 2: Linear Elasticity . . . . .	52
7.1.4	2D BEM Laplace . . . . .	53
7.2	Results . . . . .	53
7.2.1	InitCG vs DCG . . . . .	54
7.2.2	Choice of Deflation Space . . . . .	54
7.2.3	Level of Deflation Matrices . . . . .	58
7.2.4	Potential of Adaptive Precision Nested DCG with CP Corrections . .	59
7.2.5	Parallel Scalability . . . . .	61
<b>8</b>	<b>Conclusion</b>	<b>67</b>
<b>A</b>	<b>Results for Matrices from SuiteSparse Matrix Collection</b>	<b>75</b>

## List of Symbols and Abbreviations

BEM	– Boundary Element Method
CG	– Conjugate Gradient
CP	– Coarse Problem
DCG	– Deflated Conjugate Gradient
DOF	– Degree of Freedom (unknown)
FWT	– Fast Wavelet Transform
HPC	– High Performance Computing
JD	– Jacobi-Davidson
LOBPCG	– Locally Optimal Block Preconditioned Conjugate Gradient
PCG	– Preconditioned Conjugate Gradient
PDCG	– Preconditioned Deflated Conjugate Gradient
SPD	– Symmetric Positive Definite
$\ \mathbf{x}\ _{\mathbf{A}}$	– Energy norm ( $\sqrt{\mathbf{x}^T \mathbf{A} \mathbf{x}}$ )
$\ \mathbf{x}\ $	– Euclidean norm ( $\sqrt{\mathbf{x}^T \mathbf{x}}$ )
$\nabla$	– Gradient
$\mathbf{I}$	– Identity matrix
$\mathbf{e}_k$	– $k$ th vector of the standard basis
$\mathcal{K}$	– Krylov subspace
$\mathbf{O}$	– Null matrix
$\mathbf{o}$	– Null vector
$\mathbb{R}^n$	– Real coordinate space with $n$ dimensions
$\sigma(\mathbf{A})$	– Spectrum (set of eigenvalues) of $\mathbf{A}$



## List of Figures

1	Quadratic form . . . . .	14
2	Steepest descent vs CG . . . . .	15
3	Wavelet decomposition . . . . .	40
4	Implemented scaling functions . . . . .	41
5	Laplace: solution . . . . .	52
6	Linear elasticity: geometry . . . . .	52
7	Linear elasticity: solution . . . . .	53
8	Comparison of deflation spaces: Laplace . . . . .	56
9	Comparison of deflation spaces: Elasticity . . . . .	57
10	Comparison of deflation spaces: BEM . . . . .	57
11	Effect of deflation levels . . . . .	59
12	Parallel scalability: Laplace small . . . . .	62
13	Parallel scalability: Laplace large . . . . .	63
14	Parallel scalability: Elasticity . . . . .	65





---

## List of Tables

1	Deflation matrices costs . . . . .	48
2	InitCG vs DCG: Laplace benchmark . . . . .	54
3	InitCG vs DCG: Elasticity benchmark . . . . .	54
4	Comparison of deflation spaces . . . . .	56
5	Comparison of wavelet-based deflation spaces . . . . .	57
6	Effect of deflation levels . . . . .	59
7	Adaptive nested DCG . . . . .	60
8	Parallel scalability: Laplace small . . . . .	62
9	Parallel scalability: Laplace large . . . . .	63
10	Parallel scalability: Elasticity . . . . .	64



---

## List of Algorithms

1	Steepest descent method . . . . .	15
2	Lanczos method . . . . .	18
3	Lanczos method for linear systems . . . . .	19
4	Recursive Lanczos method for linear systems . . . . .	21
5	CG . . . . .	24
6	PCG . . . . .	28
6	. . . . .	28
7	InitCG . . . . .	30
8	DCG . . . . .	31
8	. . . . .	31
9	PDCG . . . . .	31



# 1 Introduction

A large number of problems in engineering, physics, finance, etc. transform into some system of linear equations with a positive definite matrix. Moreover, with the continuing increase in the available computational resources the problem sizes also grow rapidly.

The conjugate gradient (CG) algorithm is one of the most widely used methods for the solution of such systems. This is due to several favourable factors like a small memory footprint or the ability to scale to a large number of computational cores very well. Another advantage is that it is generally implemented as an iterative method allowing for early termination if we are in a certain sense close to the solution. These and other factors make the method successful, especially for very large matrices.

For the last few decades, various schemes accelerating the convergence of CG and similar methods were and still are in the forefront of the development in the field of the numerical methods.

One of such schemes is deflation. The speed of the convergence of the CG method depends highly on the spectrum of the matrix comprising the linear system. Some parts of the spectrum can slow down the convergence. The idea behind the deflation is to hide (deflate) from the CG method the part of the spectrum that retards the convergence. This is achieved by splitting the problem into two parts. The first one is a directly obtained solution on the space representing the part of the spectrum with bad convergence and the second one is computed by the CG method operating only on the complement of the first space.

The aim of this thesis is to create an efficient implementation of the deflated conjugate gradient (DCG) method with various modifications. Moreover, we also discuss the theoretical considerations and demonstrate the efficiency by numerical experiments.

This thesis is divided into the following sections. First, we derive the CG method in Section 2. This is done through the derivation of the steepest descent method and the Lanczos method. We also discuss the convergence of CG and preconditioning.

In Section 3 we describe the DCG method and its variant InitCG. Then we discuss the convergence of the algorithm. We also investigate how to solve the coarse problem (CP) that is part of the DCG operator. A scheme of nesting DCGs for the CP solution is proposed.

Section 4 describes some choices of deflation spaces. These include deflation by eigenvectors, subdomain aggregation, discrete wavelet compression and finally the prolongation/restriction multigrid operators.

A brief overview of the libraries and the high performance computing (HPC) infrastructure used for the numerical experiments can be found in Section 5.

Section 6 discusses the implementation. Various options for the solver are described. Implementation choices about handling the matrices in the deflation operator are also discussed.

Finally, in Section 7 we show how DCG and its modification with various deflation spaces perform on a wide range of benchmarks.



## 2 Conjugate Gradient Method

Krylov subspace methods represent one of the most successful classes of methods for solving large and often sparse linear systems of equations as well as eigenvalue problems. In fact, the Krylov subspace methods were named as one of the 'Top 10 algorithms of the 20th century' [1, 2]. These methods are nowadays generally viewed as iterative. The CG method [3] developed independently by Hestenes and Stiefel and the method due to Lanczos [4, 5], both introduced in the early 1950s, were the first to spark the interest in the Krylov subspace iterative methods.

The following subsections introduce the CG method. We start with the steepest descent method. Then we derive a strategy based on special subspaces designed to improve upon the convergence of the steepest descent method. After that, we derive Lanczos tridiagonalization which generates the subspaces we need, and then we adapt it to solve linear systems. We then simplify the method to get the common formulation of CG. We also discuss the convergence of CG and finally, we derive the preconditioned version of CG.

For the whole thesis, we assume that we have the following system of linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \quad (1)$$

where  $\mathbf{A}$  is a symmetric positive definite (SPD) matrix and  $\mathbf{b}$  is the right-hand side. We denote by  $\mathbf{x}_*$  the solution of (1). The matrix and the vectors are assumed to be real and  $n$ -dimensional.

The solution of the linear system given by (1) is equivalent to a problem of an unconstrained quadratic minimization

$$\min_{\mathbf{x}} f(\mathbf{x}), \quad \text{where } f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} - \mathbf{x}^T \mathbf{b}. \quad (2)$$

Notice that from the necessary condition for extrema  $\mathbf{x}_*$  minimizes  $f(\mathbf{x})$  if

$$\nabla f(\mathbf{x}_*) = \mathbf{A}\mathbf{x}_* - \mathbf{b} = \mathbf{o}.$$

The quadratic form  $f(\mathbf{x})$  can be visualized for an SPD matrix  $\mathbf{A}$  of dimension  $n = 2$ , see Figure 1. It is helpful to think about finding the solution of the linear system (1) as finding the minimum of the elliptic paraboloid of appropriate dimension given by (2).

### 2.1 Steepest Descent Method

The steepest descent method is based on the line search procedure. Given  $\mathbf{x}_k$  approximating solution of (2), search direction  $\mathbf{v}_k$ , and step length  $\alpha_k$  the line search generates a new approximation  $\mathbf{x}_{k+1}$  by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{v}_k. \quad (3)$$

To ensure that  $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$  we need to choose appropriate search direction and step

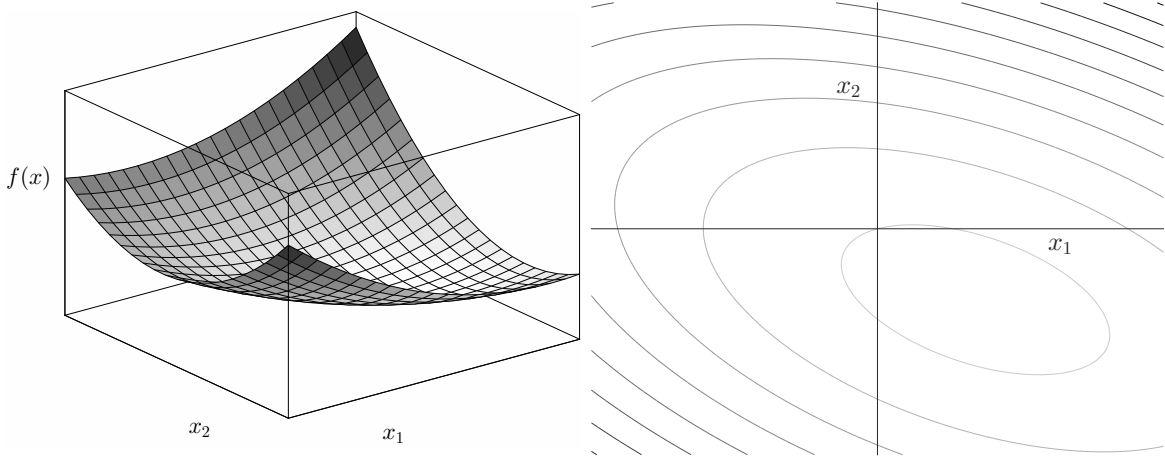


Figure 1: Surface and contour plot of  $f(\mathbf{x})$  for an SPD matrix of dimension  $n = 2$

length. Naturally, the search direction should be the negative gradient of  $f(\mathbf{x})$  because the gradient is the direction in which  $f(\mathbf{x})$  most rapidly increases. Note that the choice of the search direction suggests the reason for method's name. Therefore, we have

$$\mathbf{v}_k = -\nabla f(\mathbf{x}_k) = \mathbf{b} - \mathbf{A}\mathbf{x}_k.$$

Let us define a residual  $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$ . Note that  $\mathbf{v}_k = \mathbf{r}_k$ . Rewriting (3) using residuals as the descent directions yields

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{r}_k.$$

Scaling the previous equation by  $-\mathbf{A}$  and adding  $\mathbf{b}$  to both sides, we obtain the following recurrence for the residuals

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{r}_k.$$

Now, the only thing missing is the value of  $\alpha_k$ . Knowing the search direction, we just need to minimize  $f(\mathbf{x}_{k+1})$  with respect to the single variable  $\alpha_k$ . So again, using the necessary condition for extrema we have

$$\begin{aligned} \frac{d}{d\alpha_k} f(\mathbf{x}_{k+1}) &= \frac{d}{d\alpha_k} \left( \frac{1}{2} (\mathbf{x}_k + \alpha_k \mathbf{r}_k)^T \mathbf{A} (\mathbf{x}_k + \alpha_k \mathbf{r}_k) - (\mathbf{x}_k + \alpha_k \mathbf{r}_k)^T \mathbf{b} \right) \\ &= \mathbf{r}_k^T \nabla f(\mathbf{x}_{k+1}) = \mathbf{r}_k^T (-\mathbf{r}_{k+1}) = 0. \end{aligned} \quad (4)$$

We have found out that the descent direction  $\mathbf{r}_k$  is orthogonal to the gradient  $\nabla f(\mathbf{x}_{k+1}) = -\mathbf{r}_{k+1}$ . See Figure 2 for an illustration and notice that the orthogonality of the descent directions means that we might minimize in the same direction more than once. Using the



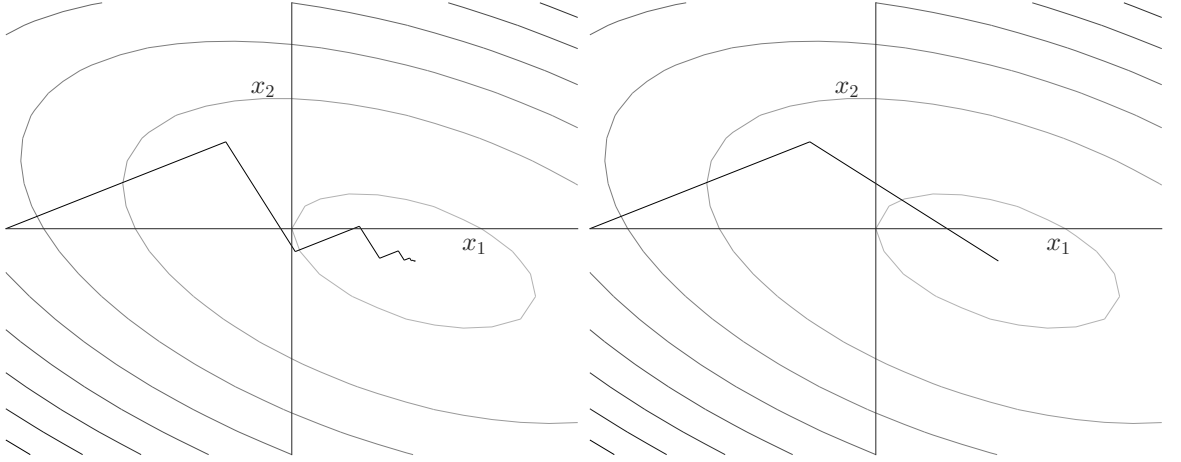


Figure 2: Contour plots of  $f(\mathbf{x})$  for an SPD matrix of dimension  $n = 2$  with plotted steps of the steepest descent method (left) and CG (right)

orthogonality of the residuals, we may find the value of  $\alpha_k$  as follows

$$\begin{aligned}
 \mathbf{r}_{k+1}^T \mathbf{r}_k &= 0 \\
 (\mathbf{b} - \mathbf{A}\mathbf{x}_{k+1})^T \mathbf{r}_k &= 0 \\
 (\mathbf{b} - \mathbf{A}(\mathbf{x}_k + \alpha_k \mathbf{r}_k))^T \mathbf{r}_k &= 0 \\
 (\mathbf{b} - \mathbf{A}\mathbf{x}_k)^T \mathbf{r}_k - \alpha_k (\mathbf{A}\mathbf{r}_k)^T \mathbf{r}_k &= 0 \\
 \mathbf{r}_k^T \mathbf{r}_k &= \alpha_k (\mathbf{A}\mathbf{r}_k)^T \mathbf{r}_k \\
 \alpha_k &= \frac{\mathbf{r}_k^T \mathbf{r}_k}{(\mathbf{A}\mathbf{r}_k)^T \mathbf{r}_k} = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_k^T \mathbf{A}\mathbf{r}_k}. \tag{5}
 \end{aligned}$$

When should we stop refining our approximation? Let us define the error  $\boldsymbol{\epsilon}_k = \mathbf{x}_k - \mathbf{x}_*$ . Naturally, we would like to stop the iterations when the error is sufficiently small. However, the solution  $\mathbf{x}_*$  is unknown. Luckily we can notice that  $\mathbf{r}_k = -\mathbf{A}\boldsymbol{\epsilon}_k$  and so we can use the norm of the residual as a stopping criterion.

We can sum up the previous observations in Algorithm 1.

---

**Algorithm 1:** Steepest descent method

---

**Input:**  $\mathbf{A}$ ,  $\mathbf{x}_0$ ,  $\mathbf{b}$

- 1  $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
- 2 **for**  $k = 0, \dots$ :
- 3    $\mathbf{s} = \mathbf{A}\mathbf{r}_k$
- 4    $\alpha_k = (\mathbf{r}_k^T \mathbf{r}_k) / (\mathbf{s}^T \mathbf{r}_k)$
- 5    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{r}_k$
- 6    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{s}$

**Output:**  $\mathbf{x}_k$

---

## 2.2 Minimize over Subspace

The problem with the steepest descent method is that in each step it minimizes  $f(\mathbf{x})$  only in a single direction. Therefore, it might minimize in the same direction more than once, because it can not use the information obtained by the previous steps. What we would prefer is a method that in a single iteration would find minimizer over some subspace and a way to extend this subspace for the next iteration. Formally, we would like to create a nested sequence of subspaces

$$\mathcal{S}_1 \subset \mathcal{S}_2 \subset \cdots \subset \mathbb{R}^n,$$

where  $\dim(\mathcal{S}_k) = k$ . Now, given an initial guess  $\mathbf{x}_0$  for each  $k$  we could solve

$$f(\mathbf{x}_k) = \min_{\mathbf{x}_0 + \mathcal{S}_k} f(\mathbf{x}).$$

Because the subspaces are expanding, we will obtain better and better approximation of the solution  $\mathbf{x}$ , and after  $n$  steps the exact solution. However, we would like to get a good approximation far sooner than after  $n$  steps. Therefore, we need to construct subspaces  $\mathcal{S}_k$  so that  $f(\mathbf{x})$  decreases quickly.

Since the subspaces are nested we know, that  $\mathcal{S}_{k+1}$  contains the previous minimizer  $\mathbf{x}_k$ . Moreover, we already know that the objective function  $f(\mathbf{x})$  decreases most rapidly in the direction of the negative gradient. Therefore, it seems reasonable to extend  $\mathcal{S}_k$  into  $\mathcal{S}_{k+1}$  by the gradient  $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$ . This choice makes the next approximation  $\mathbf{x}_{k+1}$  at least as good as the one that the steepest descent method would make.

It follows that  $\mathcal{S}_1 = \text{span}\{\mathbf{g}_0\}$ . Now, as discussed above, we extend the space by  $\mathbf{g}_1$ , i.e.  $\mathcal{S}_2 = \text{span}\{\mathbf{g}_0, \mathbf{g}_1\}$ . It turns out we can rewrite this slightly because

$$\mathbf{g}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b} = \mathbf{A}\mathbf{x}_1 - \mathbf{A}\mathbf{x}_0 + \mathbf{g}_0 = \mathbf{A}(\mathbf{x}_0 + \alpha\mathbf{g}_0) - \mathbf{A}\mathbf{x}_0 + \mathbf{g}_0 \in \text{span}\{\mathbf{g}_0, \mathbf{A}\mathbf{g}_0\},$$

where  $\alpha \in \mathbb{R}$ . We used the fact that  $\mathbf{x}_1$  is minimizer on  $\mathbf{x}_0 + \text{span}\{\mathbf{g}_0\} = \mathbf{x}_0 + \alpha\mathbf{g}_0$ . Similarly, for the next space, we have

$$\begin{aligned} \mathbf{g}_2 &= \mathbf{A}\mathbf{x}_2 - \mathbf{b} = \mathbf{A}\mathbf{x}_2 - \mathbf{A}\mathbf{x}_1 + \mathbf{g}_1 \\ &= \mathbf{A}(\mathbf{x}_0 + \alpha\mathbf{g}_0 + \beta\mathbf{A}\mathbf{g}_0) - \mathbf{A}(\mathbf{x}_0 + \gamma\mathbf{g}_0) + \mathbf{g}_1 \in \text{span}\{\mathbf{g}_0, \mathbf{A}\mathbf{g}_0, \mathbf{A}^2\mathbf{g}_0\}, \end{aligned}$$

where  $\alpha, \beta, \gamma \in \mathbb{R}$ . By repeating this process we obtain that

$$\mathcal{S}_{k+1} = \text{span}\{\mathbf{g}_0, \mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_k\} = \text{span}\{\mathbf{g}_0, \mathbf{A}\mathbf{g}_0, \mathbf{A}^2\mathbf{g}_0, \dots, \mathbf{A}^k\mathbf{g}_0\} = \mathcal{K}_{k+1}(\mathbf{A}, \mathbf{g}_0). \quad (6)$$

The space  $\mathcal{K}_{k+1}(\mathbf{A}, \mathbf{g}_0) = \text{span}\{\mathbf{g}_0, \mathbf{A}\mathbf{g}_0, \mathbf{A}^2\mathbf{g}_0, \dots, \mathbf{A}^k\mathbf{g}_0\}$  is called the Krylov subspace. Thus we found that we can hope for a faster convergence than that of the steepest descent method by minimizing the functional (2) in the Krylov subspaces.

### 2.3 Lanczos Method

Any symmetric matrix has a so called tridiagonal decomposition. More formally, we can find an orthonormal matrix  $\mathbf{Q} = (\mathbf{q}_1, \dots, \mathbf{q}_n)$  such that

$$\mathbf{Q}^T \mathbf{A} \mathbf{Q} = \mathbf{T}, \quad (7)$$

where  $\mathbf{T}$  is a tridiagonal matrix using e.g. Householder transformations [6]. The reductions to the tridiagonal matrices are very favourable for solving eigenvalue problems [7] as  $\mathbf{A}$  and  $\mathbf{T}$  are similar matrices, and therefore they have the same spectrum.

However, using something like the Householder transformations would be very expensive, so let us instead try to form matrix  $\mathbf{T}$  directly. Thanks to the symmetry of  $\mathbf{A}$  we can write  $\mathbf{T}$  as

$$\mathbf{T} = \begin{pmatrix} \gamma_1 & \delta_1 & & \cdots & 0 \\ \delta_1 & \gamma_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \delta_{n-1} \\ 0 & \cdots & & \delta_{n-1} & \gamma_n \end{pmatrix}.$$

Assume  $\mathbf{Q}$  is given, then premultiplying Equation (7) by  $\mathbf{Q}$  we have  $\mathbf{A} \mathbf{Q} = \mathbf{Q} \mathbf{T}$ . This can be equivalently rewritten for each column index  $k \in \{1, \dots, n-1\}$  as

$$\mathbf{A} \mathbf{q}_k = \delta_{k-1} \mathbf{q}_{k-1} + \gamma_k \mathbf{q}_k + \delta_k \mathbf{q}_{k+1}, \quad (8)$$

where  $\delta_0 \mathbf{q}_0 = 0$ . Premultiplying the previous equation by  $\mathbf{q}_k^T$  and using the orthonormality of  $\mathbf{Q}$  we obtain

$$\gamma_k = \mathbf{q}_k^T \mathbf{A} \mathbf{q}_k.$$

From Equation (8) we can also easily get that

$$\mathbf{q}_{k+1} = \frac{(\mathbf{A} - \gamma_k \mathbf{I}) \mathbf{q}_k - \delta_{k-1} \mathbf{q}_{k-1}}{\delta_k} = \frac{\mathbf{r}_k}{\delta_k},$$

and since  $\mathbf{q}_{k+1}$  is supposed to be orthonormal, we have

$$\delta_k = \pm \|\mathbf{r}_k\|.$$

Without the loss of generality we can choose  $\delta_k = \|\mathbf{r}_k\|$ .

We sum up our observations into Algorithm 2. Notice that given  $\widetilde{\mathbf{q}}_1$  the outlined procedure generates a sequence of orthonormal  $\mathbf{q}_k$  called the Lanczos vectors such that  $\mathbf{q}_k \in \mathcal{K}_k(\mathbf{A}, \widetilde{\mathbf{q}}_1)$ . Also note that the iteration can breakdown before  $k = n$ . If  $\mathbf{r}_k = \mathbf{o}$  then from (8)

$$\mathbf{A} \mathbf{q}_k = \mathbf{Q}_k \mathbf{T}_k,$$

and so  $\mathbf{Q}_k$  range is invariant for  $\mathbf{A}$ . Which is a good news if we try to find the solution of (1) in the subspace spanned by the columns of  $\mathbf{Q}_k$ .

---

**Algorithm 2:** Lanczos method

---

**Input:**  $\mathbf{A}, \widetilde{\mathbf{q}}_1$   
 1  $\mathbf{q}_0 = \mathbf{o}$   
 2  $\mathbf{r}_0 = \widetilde{\mathbf{q}}_1$   
 3  $\delta_0 = \|\mathbf{r}_0\|$   
 4  $\mathbf{q}_1 = \mathbf{r}_0/\delta_0$   
 5  $k = 1$   
 6 **while**  $\delta_{k-1} \neq 0$ :  
 7      $\mathbf{s} = \mathbf{A}\mathbf{q}_k$   
 8      $\gamma_k = \mathbf{q}_k^T \mathbf{s}$   
 9      $\mathbf{r}_k = \mathbf{s} - \gamma_k \mathbf{q}_k - \delta_{k-1} \mathbf{q}_{k-1}$   
 10     $\delta_k = \|\mathbf{r}_k\|$   
 11     $\mathbf{q}_{k+1} = \mathbf{r}_k/\delta_k$   
 12     $k = k + 1$   
**Output:**  $\mathbf{Q}_k, T_k$

---

## 2.4 Using Lanczos Method to Solve Linear Systems

We outlined in Section 2.2 a way to improve the convergence of the steepest descent method. The key ingredient was that we are minimizing  $f(x)$  over subspace  $\mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{g}_0)$ . In the previous section we discovered that this subspace is generated by the Lanczos procedure, i.e. we can minimize over the subspace

$$\mathbf{x}_0 + \mathcal{K}(\mathbf{A}, \mathbf{g}_0, k) = \mathbf{x}_0 + \text{span}\{\mathbf{q}_1, \dots, \mathbf{q}_k\} = \mathbf{x}_0 + \{\alpha_1 \mathbf{q}_1 + \dots + \alpha_k \mathbf{q}_k : \alpha_k \in \mathbb{R}\}$$

assuming  $\mathbf{q}_1 = \mathbf{r}_0/\delta_0$  where  $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$  and  $\delta_0 = \|\mathbf{r}_0\|$ .

Now, by setting  $\mathbf{Q}_k = (\mathbf{q}_1, \dots, \mathbf{q}_k)$  we can rewrite the problem as finding  $\mathbf{y}_k \in \mathbb{R}^n$  that minimizes

$$\begin{aligned} f(\mathbf{x}_0 + \mathbf{Q}_k \mathbf{y}_k) &= \frac{1}{2} (\mathbf{x}_0 + \mathbf{Q}_k \mathbf{y}_k)^T \mathbf{A} (\mathbf{x}_0 + \mathbf{Q}_k \mathbf{y}_k) - (\mathbf{x}_0 + \mathbf{Q}_k \mathbf{y}_k)^T \mathbf{b} \\ &= \frac{1}{2} (\mathbf{Q}_k \mathbf{y}_k)^T \mathbf{A} (\mathbf{Q}_k \mathbf{y}_k) + (\mathbf{Q}_k \mathbf{y}_k)^T \mathbf{A} \mathbf{x}_0 - (\mathbf{Q}_k \mathbf{y}_k)^T \mathbf{b} + \frac{1}{2} \mathbf{x}_0^T \mathbf{A} \mathbf{x}_0 - \mathbf{x}_0^T \mathbf{b} \\ &= \frac{1}{2} \mathbf{y}_k^T \mathbf{Q}_k^T \mathbf{A} \mathbf{Q}_k \mathbf{y}_k + \mathbf{y}_k^T \mathbf{Q}_k^T (\mathbf{A} \mathbf{x}_0 - \mathbf{b}) + f(\mathbf{x}_0) \\ &= \frac{1}{2} \mathbf{y}_k^T (\mathbf{Q}_k^T \mathbf{A} \mathbf{Q}_k) \mathbf{y}_k - \mathbf{y}_k^T (\mathbf{Q}_k^T \mathbf{r}_0) + f(\mathbf{x}_0). \end{aligned}$$

Therefore, in each iteration our algorithm will generate a new approximation

$$\mathbf{x}_k = \mathbf{x}_0 + \mathbf{Q}_k \mathbf{y}_k, \tag{9}$$

where  $\mathbf{y}_k$  is obtained as a solution of

$$\begin{aligned} \mathbf{Q}_k^T \mathbf{A} \mathbf{Q}_k \mathbf{y}_k &= \mathbf{Q}_k^T \mathbf{r}_0 \\ \mathbf{T}_k \mathbf{y}_k &= \mathbf{Q}_k^T \mathbf{r}_0 \\ \mathbf{T}_k \mathbf{y}_k &= \delta_0 \mathbf{Q}_k^T \mathbf{q}_1 \\ \mathbf{T}_k \mathbf{y}_k &= \delta_0 \mathbf{e}_1. \end{aligned} \tag{10}$$

Plugging in these observations into Algorithm 2 we obtain Algorithm 3.

---

**Algorithm 3:** Lanczos method for linear systems

---

**Input:**  $\mathbf{A}$ ,  $\mathbf{x}_0$ ,  $\mathbf{b}_0$

```

1  $\mathbf{q}_0 = \mathbf{o}$ 
2  $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ 
3  $\delta_0 = \|\mathbf{r}_0\|$ 
4  $\mathbf{q}_1 = \mathbf{r}_0/\delta_0$ 
5  $k = 1$ 
6 while  $\delta_{k-1} \neq 0$ :
7    $\mathbf{s} = \mathbf{A}\mathbf{q}_k$ 
8    $\gamma_k = \mathbf{q}_k^T \mathbf{s}$ 
9    $\mathbf{T}_k \mathbf{y}_k = \delta_0 \mathbf{e}_1$ 
10   $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{Q}_k \mathbf{y}_k$ 
11   $\mathbf{r}_k = \mathbf{s} - \gamma_k \mathbf{q}_k - \delta_{k-1} \mathbf{q}_{k-1}$ 
12   $\delta_k = \|\mathbf{r}_k\|$ 
13   $\mathbf{q}_{k+1} = \mathbf{r}_k/\delta_k$ 
14   $k = k + 1$ 
Output:  $\mathbf{x}_k$ 
```

---

However, the formulation has several drawbacks. The first one, is that in the  $k$ th iteration we have to do a matrix vector multiplication with  $n \times k$  (dense) matrix. Moreover, to obtain this matrix we have to store all previous Lanczos vectors. The second drawback is the need to solve the tridiagonal system in order to obtain  $\mathbf{y}_k$ .

#### 2.4.1 Recursive Formulation

Fortunately, the problems outlined above can be circumvented. Let us start with the solution of the tridiagonal system (10). Because  $\mathbf{T}$  is similar to  $\mathbf{A}$  it is SPD and therefore it has an  $\mathbf{LDL}^T$  factorization [6], where

$$\mathbf{L}_k = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_1 & 1 & & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & l_{k-1} & 1 \end{pmatrix}, \quad \mathbf{D}_k = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & d_k \end{pmatrix}.$$

Thanks to the tridiagonality of  $\mathbf{T}$ ,  $\mathbf{L}_k$  is only bidiagonal. By setting

$$\mathbf{T}_k = \mathbf{L}_k \mathbf{D}_k \mathbf{L}_k^T,$$

and comparing the coefficients we have

$$\begin{aligned} d_1 &= \gamma_1, \\ l_i &= \delta_i / d_i, \\ d_{i+1} &= \gamma_{i+1} - l_i \delta_i, \quad i \in \{1, \dots, k-1\}. \end{aligned}$$

Now we can rewrite (10) as

$$\begin{aligned} \mathbf{L}_k \mathbf{D}_k \mathbf{L}_k^T \mathbf{y}_k &= \delta_0 \mathbf{e}_1 \\ \mathbf{L}_k \mathbf{D}_k \mathbf{u}_k &= \delta_0 \mathbf{e}_1, \end{aligned} \tag{11}$$

where  $\mathbf{u}_k = \mathbf{L}_k^T \mathbf{y}_k$ . By taking a closer look at (11)

$$\begin{pmatrix} d_1 & 0 & \cdots & 0 \\ d_1 l_1 & d_2 & & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & d_{k-1} l_{k-1} & d_k \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_k \end{pmatrix}_k = \begin{pmatrix} \delta_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

we can easily evaluate the solution

$$u_k = \begin{cases} \delta_0 / d_1 & \text{if } k = 1 \\ -d_{k-1} l_{k-1} u_{k-1} / d_k & \text{if } k > 1. \end{cases}$$

It turns out that we actually do not need to compute  $\mathbf{y}_k$ . If we take  $\mathbf{C}_k \in \mathbb{R}^{n \times k}$  satisfying

$$\mathbf{C}_k \mathbf{L}_k^T = \mathbf{Q}_k,$$

which in the expanded form reads as

$$\begin{pmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_k \end{pmatrix} \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_1 & 1 & & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & l_{k-1} & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_k \end{pmatrix},$$

then we immediately see that

$$\mathbf{c}_k = \begin{cases} \mathbf{q}_1 & \text{if } k = 1 \\ \mathbf{q}_k - l_{k-1}\mathbf{g}_{k-1} & \text{if } k > 1. \end{cases}$$

Using this and  $\mathbf{L}_k^T \mathbf{y}_k = \mathbf{u}_k$  we can rewrite (9) as

$$\mathbf{x}_k = \mathbf{x}_0 + \mathbf{Q}_k \mathbf{y}_k = \mathbf{x}_0 + \mathbf{C}_k \mathbf{L}_k^T \mathbf{y}_k = \mathbf{x}_0 + \mathbf{C}_k \mathbf{u}_k = \mathbf{x}_0 + \mathbf{C}_{k-1} \mathbf{u}_{k-1} + u_k \mathbf{c}_k = \mathbf{x}_{k-1} + u_k \mathbf{c}_k.$$

We have found a recursive formula for  $\mathbf{x}_k$ . Moreover both  $u_k$  and  $\mathbf{c}_k$  are also obtained recursively. This means that we do not have to store the Lanczos vectors or the tridiagonal matrix. By using the derived recursions we rewrite Algorithm 3 into recursive formulation of the Lanczos method for linear system (Algorithm 4).

---

**Algorithm 4:** Recursive Lanczos method for linear systems

---

**Input:**  $\mathbf{A}$ ,  $\mathbf{x}_0$ ,  $\mathbf{b}_0$

```

1  $\mathbf{q}_0 = \mathbf{o}$ 
2  $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ 
3  $\delta_0 = \|\mathbf{r}_0\|$ 
4  $\mathbf{q}_1 = \mathbf{r}_0/\delta_0$ 
5  $\mathbf{c}_1 = \mathbf{q}_1$ 
6  $k = 1$ 
7 while  $\delta_{k-1} \neq 0$ :
8    $\mathbf{s} = \mathbf{A}\mathbf{q}_k$ 
9    $\gamma_k = \mathbf{q}_k^T \mathbf{s}$ 
10  if  $k = 1$ :
11     $d_1 = \gamma_1$ 
12     $u_1 = \delta_0/d_1$ 
13  else:
14     $l_{k-1} = \delta_{k-1}/d_{k-1}$ 
15     $d_k = \gamma_k - \delta_{k-1}l_{k-1}$ 
16     $u_k = -\delta_{k-1}u_{k-1}/d_k$ 
17   $\mathbf{x}_k = \mathbf{x}_{k-1} + u_k \mathbf{c}_k$ 
18   $\mathbf{r}_k = \mathbf{s} - \gamma_k \mathbf{q}_k - \delta_{k-1} \mathbf{q}_{k-1}$ 
19   $\delta_k = \|\mathbf{r}_k\|$ 
20   $\mathbf{q}_{k+1} = \mathbf{r}_k/\delta_k$ 
21   $\mathbf{c}_{k+1} = \mathbf{q}_{k+1} - l_k \mathbf{c}_k$ 
22   $k = k + 1$ 
Output:  $\mathbf{x}_k$ 
```

---

### 2.4.2 Conjugacy of Gradients and Directions

We call two vectors conjugate if they are orthogonal ( $\mathbf{x}^T \mathbf{y} = 0$ ) and  $\mathbf{A}$ -conjugate (or  $\mathbf{A}$ -orthogonal) if they are orthogonal with respect to  $\mathbf{A}$ -based dot product ( $\mathbf{x}^T \mathbf{A} \mathbf{y} = 0$ ).

Let

$$\mathbf{g}_k = \mathbf{A}\mathbf{x}_k - \mathbf{b} = \nabla f(\mathbf{x}_k)$$

be the gradient in each iteration of Algorithm 4. Then we can rewrite it as

$$\mathbf{g}_k = \mathbf{A}(\mathbf{x}_0 + \mathbf{q}_k \mathbf{y}_k) - \mathbf{b} = -\mathbf{r}_0 + (\mathbf{Q}_k \mathbf{T}_k + \mathbf{r}_k \mathbf{e}_k^T) \mathbf{y}_k,$$

where we used  $\mathbf{A}\mathbf{Q}_k = \mathbf{Q}_k \mathbf{T}_k + \mathbf{r}_k \mathbf{e}_k^T$  that follows from (8) and an examination of the columns of  $\mathbf{Q}_k \mathbf{T}_k$ . Because  $\mathbf{Q}_k \mathbf{T}_k \mathbf{y}_k = \delta_0 \mathbf{Q}_k \mathbf{e}_1 = \mathbf{r}_0$  we have

$$\mathbf{g}_k = (\mathbf{e}_k^T \mathbf{y}_k) \mathbf{r}_k = (\mathbf{e}_k^T \mathbf{y}_k) \delta_k \mathbf{q}_{k+1}. \quad (12)$$

Since  $\mathbf{g}_k$  is a multiple of  $\mathbf{q}_{k+1}$  we have from mutual orthogonality of  $\mathbf{q}_k$  that  $\mathbf{g}_k$  are also mutually orthogonal. Moreover, from (6) it follows that  $\mathbf{g}_k$  is orthogonal to  $\mathcal{K}_k(\mathbf{A}, \mathbf{g}_0)$ .

We can also show that the search directions  $\mathbf{c}_k$  are mutually  $\mathbf{A}$ -orthogonal. We have

$$\mathbf{T}_k = \mathbf{Q}_k^T \mathbf{A} \mathbf{Q}_k = (\mathbf{C}_k \mathbf{L}_k^T)^T \mathbf{A} (\mathbf{C}_k \mathbf{L}_k^T) = \mathbf{L}_k (\mathbf{C}_k^T \mathbf{A} \mathbf{C}_k) \mathbf{L}_k^T,$$

and from uniqueness of the  $\mathbf{T}_k = \mathbf{L}_k \mathbf{D}_k \mathbf{L}_k^T$  factorization [6] it follows

$$\mathbf{D}_k = \mathbf{C}_k^T \mathbf{A} \mathbf{C}_k.$$

Since  $\mathbf{D}_k$  is diagonal the search directions  $\mathbf{c}_k$  are mutually  $\mathbf{A}$ -orthogonal.

## 2.5 Conjugate Gradients Algorithm

Let us redefine  $\mathbf{r}_k$  as a residual

$$\mathbf{r}_k = \mathbf{b}_k - \mathbf{A}\mathbf{x}_k = -\mathbf{g}_k.$$

We can make the last algorithm easier to read by considering the search direction

$$\mathbf{c}_{k+1} = \mathbf{q}_{k+1} - l_k \mathbf{c}_k.$$

From (12),  $\mathbf{q}_{k+1}$  is just some multiple of  $\mathbf{r}_k$ . This allows us to write the search direction (with numbering shifted so that the first search direction is  $\mathbf{p}_0$ ) as

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_{k+1} \mathbf{p}_k. \quad (13)$$

Premultiplying by  $(\mathbf{A}\mathbf{p}_k)^T$  it follows from the  $\mathbf{A}$ -conjugacy of the search directions that

$$\beta_{k+1} = -\frac{\mathbf{p}_k^T \mathbf{A} \mathbf{r}_{k+1}}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}. \quad (14)$$



If we instead premultiply by  $(\mathbf{A}\mathbf{p}_{k+1})^T$  and again use the  $\mathbf{A}$ -conjugacy of the search directions we get a handy identity

$$\mathbf{p}_{k+1}^T \mathbf{A}\mathbf{p}_{k+1} = \mathbf{p}_{k+1}^T \mathbf{A}\mathbf{r}_{k+1}.$$

Because  $\mathbf{p}_0 = \mathbf{c}_1$ ,  $\mathbf{p}_k$  is multiple of  $\mathbf{c}_{k+1}$ . Therefore, we need to update the step-length in the line-search

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k.$$

Scaling the equation by  $-\mathbf{A}$  and adding  $\mathbf{b}$  leads to a residual recurrence

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k. \quad (15)$$

Premultiplying by  $\mathbf{r}_k^T$  and using the orthogonality of gradients it follows that

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_k^T \mathbf{A}\mathbf{p}_k} = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k}.$$

We can further simplify the relation for  $\beta_k$ . Premultiplying the residual recurrence (15) by  $\mathbf{r}_{k+1}^T$  and using the orthogonality of residuals we get

$$\mathbf{r}_{k+1}^T \mathbf{r}_{k+1} = -\alpha_k \mathbf{r}_{k+1}^T \mathbf{A}\mathbf{p}_k.$$

Doing the same with  $\mathbf{r}_k^T$  yields

$$\mathbf{r}_k^T \mathbf{r}_k = \alpha_k \mathbf{r}_k^T \mathbf{A}\mathbf{p}_k = \alpha_k \mathbf{p}_k^T \mathbf{A}\mathbf{p}_k.$$

Substituting these into (14) allows us to compute  $\beta_{k+1}$  as

$$\beta_{k+1} = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}.$$

Rewriting Algorithm 4 using the derived formulas yields the CG method illustrated in Algorithm 5.

### 2.5.1 From CG back to Lanczos

The previous section derived CG from the Lanczos method for the solution of linear systems. It is clear that given same inputs the methods generate same approximation in each iteration. However, we did not establish direct relations between various coefficients and vectors that these methods generate. It would be useful if we could while performing solve with CG obtain the Lanczos vectors and the tridiagonal matrix. For example, it would enable us to fairly cheaply obtain the extremal eigenvalues of  $\mathbf{A}$ .

To derive (13) we have used the fact that  $\mathbf{q}_k$  is a multiple of  $\mathbf{r}_{k-1}$ . Since  $\mathbf{q}_k$  is normalized

**Algorithm 5:** CG

---

**Input:**  $A, x_0, b$

1  $r_0 = b - Ax_0$

2  $p_0 = r_0$

3 **for**  $k = 0, \dots$ :

4     $s = Ap_k$

5     $\alpha_k = (r_k^T r_k) / (s^T p_k)$

6     $x_{k+1} = x_k + \alpha_k p_k$

7     $r_{k+1} = r_k - \alpha_k s$

8     $\beta_{k+1} = (r_{k+1}^T r_{k+1}) / (r_k^T r_k)$

9     $p_{k+1} = r_{k+1} + \beta_{k+1} p_k$

**Output:**  $x_k$

---

it follows that

$$q_k = \pm \frac{1}{\|r_{k-1}\|} r_{k-1}. \quad (16)$$

Defining

$$R_k = (r_0, \dots, r_{k-1}), \quad P_k = (p_0, \dots, p_{k-1}), \quad B_k = \begin{pmatrix} 1 & -\beta_1 & 0 & \dots & 0 \\ 0 & 1 & -\beta_2 & & \vdots \\ & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -\beta_{k-1} \\ 0 & \dots & & 0 & 1 \end{pmatrix},$$

and by using (13) and  $p_0 = r_0$  we have

$$R_k = P_k B_k.$$

Moreover, thanks to mutual  $A$ -orthogonality of  $p_k$  it follows that

$$R_k^T A R_k = B_k^T P_k^T A P_k B_k = B_k^T \text{diag} (p_0^T A p_0 \dots p_{k-1}^T A p_{k-1}) B_k.$$

Setting

$$K_k = \text{diag} \left( \frac{1}{\|r_0\|}, \dots, \frac{1}{\|r_{k-1}\|} \right),$$

and using (16) we can write the Lanczos tridiagonal matrix as

$$T_k = Q_k^T A Q_k = K_k^T R_k^T A R_k K_k = K_k^T B_k^T \text{diag} (p_0^T A p_0 \dots p_{k-1}^T A p_{k-1}) B_k K_k.$$

By carrying out the multiplication on the right-hand side and comparing the result with the

coefficients of  $\mathbf{T}_k$ , it follows

$$\begin{aligned}\gamma_k &= \frac{1}{\alpha_k} + \frac{\beta_k}{\alpha_{k-1}}, \quad \beta_1 = 0, \quad \alpha_0 = 1, \\ \delta_k &= \frac{\sqrt{\beta_{k+1}}}{\alpha_k}.\end{aligned}$$

### 2.5.2 CG Convergence

We have constructed the CG algorithm so that in every iteration it minimizes  $f(\mathbf{x})$  over the Krylov subspace  $\mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ . With the error of approximation defined as  $\boldsymbol{\epsilon}_k = \mathbf{x}_k - \mathbf{x}_*$  we have

$$\begin{aligned}f(\mathbf{x}_k) &= \frac{1}{2} \mathbf{x}_k^T \mathbf{A} \mathbf{x}_k - \mathbf{x}_k^T \mathbf{b} = \frac{1}{2} (\mathbf{x}_k + \mathbf{x}_* - \mathbf{x}_*)^T \mathbf{A} (\mathbf{x}_k + \mathbf{x}_* - \mathbf{x}_*) - \mathbf{x}_k^T \mathbf{b} \\ &= \frac{1}{2} \boldsymbol{\epsilon}_k^T \mathbf{A} \boldsymbol{\epsilon}_k + \frac{1}{2} \boldsymbol{\epsilon}_k^T \mathbf{A} \mathbf{x}_* + \frac{1}{2} \mathbf{x}_*^T \mathbf{A} \mathbf{x}_k - \mathbf{x}_k^T \mathbf{b} = \frac{1}{2} \boldsymbol{\epsilon}_k^T \mathbf{A} \boldsymbol{\epsilon}_k - \frac{1}{2} \mathbf{x}_*^T \mathbf{A} \mathbf{x}_*,\end{aligned}$$

and since  $f(\mathbf{x}_*) = -\mathbf{x}_*^T \mathbf{A} \mathbf{x}_* / 2$  the equation can be rewritten as

$$f(\mathbf{x}_k) = \frac{1}{2} \|\boldsymbol{\epsilon}_k\|_{\mathbf{A}}^2 + f(\mathbf{x}_*).$$

In other words, each iteration of CG minimizes the error in  $\mathbf{A}$ -norm over the subspace  $\mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ . Since

$$\boldsymbol{\epsilon}_k = \mathbf{x}_k - \mathbf{x}_* \in -\mathbf{x}_* + \mathbf{x}_0 + \text{span}\{\mathbf{r}_0, \dots, \mathbf{A}^{k-1} \mathbf{r}_0\} = \boldsymbol{\epsilon}_0 + \text{span}\{\mathbf{A} \boldsymbol{\epsilon}_0, \dots, \mathbf{A}^k \boldsymbol{\epsilon}_0\},$$

the error term can be written as a linear combination

$$\boldsymbol{\epsilon}_k = \left( \mathbf{I} + \sum_{i=1}^k \phi_i \mathbf{A}^i \right) \boldsymbol{\epsilon}_0,$$

where  $\phi_i \in \mathbb{R}$  are coefficients chosen by CG so that  $\|\boldsymbol{\epsilon}_k\|_{\mathbf{A}}$  is minimized. Moreover, using a polynomial  $P_k(\mathbf{A})$  of degree  $k$  that satisfies  $P(\mathbf{O}) = \mathbf{I}$ , we can rewrite the previous equation as

$$\boldsymbol{\epsilon}_k = P_k(\mathbf{A}) \boldsymbol{\epsilon}_0. \tag{17}$$

This presents another way of thinking about CG – in  $k$ th iteration it finds a polynomial  $P_k(\mathbf{A})$  that minimizes the error term given by (17) in  $\mathbf{A}$ -norm, i.e.

$$\|\boldsymbol{\epsilon}_k\|_{\mathbf{A}} = \min_{P_k} \|P_k(\mathbf{A}) \boldsymbol{\epsilon}_0\|_{\mathbf{A}}.$$

Given the  $\lambda_{\min} = \lambda_1, \dots, \lambda_n = \lambda_{\max}$  eigenvalues of  $\mathbf{A}$  and their corresponding normalized eigenvectors  $\mathbf{v}_1, \dots, \mathbf{v}_n$ , the error term  $\boldsymbol{\epsilon}_0$  can be expressed as a linear combination

$$\boldsymbol{\epsilon}_0 = \sum_{i=1}^n \xi_i \mathbf{v}_i,$$

which allows us to write (17) as

$$\boldsymbol{\epsilon}_k = P_k(\mathbf{A}) \sum_{i=1}^n \xi_i \mathbf{v}_i = \sum_{i=1}^n \xi_i P_k(\lambda_i) \mathbf{v}_i.$$

Note that given eigenvector  $\mathbf{v}$  and its associated eigenvalue  $\lambda$  of  $\mathbf{A}$ , we have  $P(\mathbf{A})\mathbf{v} = P(\lambda)\mathbf{v}$ . Therefore we will use the same notation for  $P_k$  regardless whether the argument is a matrix or scalar. If we use a scalar as an argument we will expect the result to be scalar as well. Therefore, it follows that  $P_k(0) = 1$ . Using the orthonormality of  $\mathbf{v}_i$  we can express the square of  $\mathbf{A}$ -norm of the error as

$$\begin{aligned} \|\boldsymbol{\epsilon}_k\|_{\mathbf{A}}^2 &= \min_{P_k} \sum_{i=1}^n \xi_i^2 (P_k(\lambda_i))^2 \lambda_i \leq \min_{P_k} \max_{\lambda \in \sigma(\mathbf{A})} (P_k(\lambda))^2 \sum_{i=1}^n \xi_i^2 \lambda_i \\ &= \min_{P_k} \max_{\lambda \in \sigma(\mathbf{A})} (P_k(\lambda))^2 \|\boldsymbol{\epsilon}_0\|_{\mathbf{A}}^2. \end{aligned} \quad (18)$$

This result gives us some insight into what is and is not a favourable spectrum of  $\mathbf{A}$ . Assuming our initial guess  $\mathbf{x}_0$  was not a solution, then the error is zero if  $P_k$  is zero for each distinct eigenvalue. If  $\mathbf{A}$  was only positive semi-definite and the linear system was consistent (right-hand side  $\mathbf{b}$  was in the range of  $\mathbf{A}$ ) we can still use CG for the solution of such system [8]. Moreover, the zero eigenvalues are ignored, and therefore null space of  $\mathbf{A}$  is ignored as well. Tight clusters of eigenvalues or eigenvalues with high multiplicity are (essentially) reduced by the same  $P_k$ , which saves a number of iterations. The eigenvalues close to the zero are problematic because of the restriction  $P_k(0) = 1$  it is hard to find a polynomial that is small in these small eigenvalues.

It follows from (18) that the relative error in  $\mathbf{A}$ -norm can be bounded by

$$\frac{\|\boldsymbol{\epsilon}_k\|_{\mathbf{A}}}{\|\boldsymbol{\epsilon}_0\|_{\mathbf{A}}} \leq \min_{P_k} \max_{\lambda} |P_k(\lambda)| \leq 2 \left( \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k,$$

where  $\kappa = \lambda_{\max}/\lambda_{\min}$  is called the condition number of  $\mathbf{A}$ . The last inequality is derived by bounding the value of  $\max_{\lambda} |P_k(\lambda)|$  by the  $k$ th scaled and shifted Chebyshev polynomial on the  $[\lambda_{\min}, \lambda_{\max}]$  interval, see e.g. [9, 10] for a proof.

### 2.5.3 Preconditioned CG

As was shown in the previous section, the speed of convergence depends on the spectral properties of  $\mathbf{A}$ . To improve the convergence of the CG method we can precondition the

system of linear equations [10]. A preconditioner for CG is the inverse of an SPD matrix  $\mathbf{M}$  that is applied to both sides of the system (1), i.e.

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}.$$

Note that if  $\mathbf{M}^{-1} = \mathbf{A}^{-1}$  then  $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$  and we have found the solution. Finding the inverse of  $\mathbf{A}$  is however costly. Therefore, we try to find some matrix (often some approximation of  $\mathbf{A}^{-1}$ ) that is easy to compute as well as easy to apply, while significantly improving the spectral properties of the preconditioned system  $\mathbf{M}^{-1}\mathbf{A}$ . Overviews of the most widely used preconditioners can be found, e.g. in [6, 9, 11].

The problem with preconditioner  $\mathbf{M}^{-1}$  is that  $\mathbf{M}^{-1}\mathbf{A}$  is not generally an SPD matrix. Fortunately, this problem can be avoided because for every square SPD matrix  $\mathbf{M}$  there exists an SPD matrix  $\mathbf{C}$  such that  $\mathbf{M} = \mathbf{C}\mathbf{C}$  [6]. Moreover,  $\mathbf{C}^{-1}\mathbf{A}\mathbf{C}^{-1}$  is SPD and has the same eigenvalues as  $\mathbf{M}^{-1}\mathbf{A}$ .

We can transform the system (1) into

$$\begin{aligned} \mathbf{C}^{-1}\mathbf{A}\mathbf{C}^{-1}\tilde{\mathbf{x}} &= \mathbf{C}^{-1}\mathbf{b}, \quad \tilde{\mathbf{x}} = \mathbf{C}\mathbf{x}, \\ \tilde{\mathbf{A}}\tilde{\mathbf{x}} &= \tilde{\mathbf{b}}, \end{aligned}$$

use the CG method to find  $\tilde{\mathbf{x}}$ , and then from the equation above we get  $\mathbf{x}$ . However, we would have to factorize the preconditioning matrix  $\mathbf{M}$  to obtain its square root  $\mathbf{C}$ . Luckily, it turns out that this is not necessary. Let us write the update formulas for the modified system

$$\begin{aligned} \alpha_k &= \left( \tilde{\mathbf{r}}_k^T \tilde{\mathbf{r}}_k \right) / \left( \tilde{\mathbf{p}}_k^T \tilde{\mathbf{A}} \tilde{\mathbf{p}}_k \right), \\ \tilde{\mathbf{x}}_{k+1} &= \tilde{\mathbf{x}}_k + \alpha_k \tilde{\mathbf{p}}_k, \\ \tilde{\mathbf{r}}_{k+1} &= \tilde{\mathbf{r}}_k - \alpha_k \tilde{\mathbf{A}} \tilde{\mathbf{p}}_k, \\ \beta_{k+1} &= \left( \tilde{\mathbf{r}}_{k+1}^T \tilde{\mathbf{r}}_{k+1} \right) / \left( \tilde{\mathbf{r}}_k^T \tilde{\mathbf{r}}_k \right), \\ \tilde{\mathbf{p}}_{k+1} &= \tilde{\mathbf{r}}_{k+1} + \beta_{k+1} \tilde{\mathbf{p}}_k. \end{aligned}$$

Using  $\mathbf{x}_k = \mathbf{C}^{-1}\tilde{\mathbf{x}}_k$  and definition of  $\tilde{\mathbf{b}}$ , we can rewrite the residual as

$$\tilde{\mathbf{r}}_k = \tilde{\mathbf{b}} - \tilde{\mathbf{A}}\tilde{\mathbf{x}}_k = \mathbf{C}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}_k) = \mathbf{C}^{-1}\mathbf{r}_k.$$

Substituting this into our update formulas and using the definition of  $\tilde{\mathbf{A}}$  we get

$$\begin{aligned}\alpha_k &= \left( \mathbf{r}_k^T \mathbf{M}^{-1} \mathbf{r}_k \right) / \left( \left( \mathbf{C}^{-1} \tilde{\mathbf{p}}_k \right)^T \mathbf{A} \left( \mathbf{C}^{-1} \tilde{\mathbf{p}}_k \right) \right), \\ \mathbf{C} \mathbf{x}_{k+1} &= \mathbf{C} \mathbf{x}_k + \alpha_k \tilde{\mathbf{p}}_k, \\ \mathbf{C}^{-1} \mathbf{r}_{k+1} &= \mathbf{C}^{-1} \mathbf{r}_k - \alpha_k \mathbf{C}^{-1} \mathbf{A} \mathbf{C}^{-1} \tilde{\mathbf{p}}_k, \\ \beta_{k+1} &= \left( \mathbf{r}_{k+1}^T \mathbf{M}^{-1} \mathbf{r}_{k+1} \right) / \left( \mathbf{r}_k^T \mathbf{M}^{-1} \mathbf{r}_k \right), \\ \tilde{\mathbf{p}}_{k+1} &= \mathbf{C}^{-1} \mathbf{r}_{k+1} + \beta_{k+1} \tilde{\mathbf{p}}_k.\end{aligned}$$

Finally, substituting  $\mathbf{z}_k = \mathbf{M}^{-1} \mathbf{r}_k$  and  $\mathbf{p}_k = \mathbf{C}^{-1} \tilde{\mathbf{p}}_k$  into the derived formulas yields the preconditioned conjugate gradient (PCG) method illustrated in Algorithm 6.

Algorithm 6: PCG	Algorithm: CG
<b>Input:</b> $\mathbf{A}$ , $\mathbf{M}^{-1}$ , $\mathbf{x}_0$ , $\mathbf{b}$	<b>Input:</b> $\mathbf{A}$ , $\mathbf{x}_0$ , $\mathbf{b}$
1 $\mathbf{r}_0 = \mathbf{b} - \mathbf{A} \mathbf{x}_0$	1 $\mathbf{r}_0 = \mathbf{b} - \mathbf{A} \mathbf{x}_0$
2 $\mathbf{z}_0 = \mathbf{M}^{-1} \mathbf{r}_0$	2
3 $\mathbf{p}_0 = \mathbf{z}_0$	3 $\mathbf{p}_0 = \mathbf{r}_0$
4 <b>for</b> $k = 0, \dots$ :	4 <b>for</b> $k = 0, \dots$ :
5 $\mathbf{s} = \mathbf{A} \mathbf{p}_k$	5 $\mathbf{s} = \mathbf{A} \mathbf{p}_k$
6 $\alpha_k = \left( \mathbf{r}_k^T \mathbf{z}_k \right) / \left( \mathbf{s}^T \mathbf{p}_k \right)$	6 $\alpha_k = \left( \mathbf{r}_k^T \mathbf{r}_k \right) / \left( \mathbf{s}^T \mathbf{p}_k \right)$
7 $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$	7 $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
8 $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{s}$	8 $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{s}$
9 $\mathbf{z}_{k+1} = \mathbf{M}^{-1} \mathbf{r}_{k+1}$	9
10 $\beta_{k+1} = \left( \mathbf{r}_{k+1}^T \mathbf{z}_{k+1} \right) / \left( \mathbf{r}_k^T \mathbf{z}_k \right)$	10 $\beta_{k+1} = \left( \mathbf{r}_{k+1}^T \mathbf{r}_{k+1} \right) / \left( \mathbf{r}_k^T \mathbf{r}_k \right)$
11 $\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_{k+1} \mathbf{p}_k$	11 $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_{k+1} \mathbf{p}_k$
<b>Output:</b> $\mathbf{x}_k$	<b>Output:</b> $\mathbf{x}_k$

### 3 Deflated Conjugate Gradients

Deflation for CG, also known as CG with preconditioning by projectors, was introduced independently in [12–14]. As we saw in the previous section, CG finds in the  $k$ th iteration a minimizer of  $f(\mathbf{x})$  over the Krylov subspace  $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ . The basic idea of DCG is to enrich this Krylov subspace by some subspace  $\mathcal{W}$ . If the subspace is properly chosen it follows that since every iteration finds a minimizer over a larger subspace, we can hope for faster convergence.

#### 3.1 Deriving DCG

Let us define the deflation matrix as

$$\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_m) \in \mathbb{R}^{n \times m}, m < n$$

Assuming that  $\mathbf{W}$  is a full rank matrix and  $\mathcal{W}$  is a subspace spanned by columns of  $\mathbf{W}$ , we can denote a projector

$$\mathbf{P} = \mathbf{I} - \mathbf{W} (\mathbf{W}^T \mathbf{A} \mathbf{W})^{-1} \mathbf{W}^T \mathbf{A} = \mathbf{I} - \mathbf{Q} \mathbf{A}$$

onto an  $\mathbf{A}$ -conjugate complement of  $\mathcal{W}$ .

We can split the solution into the solution on the deflation space  $\mathcal{W}$  and the solution on the  $\mathbf{A}$ -conjugate complement of  $\mathcal{W}$ . As we discussed in Section 2.4.2 the  $k$ th residual has to be orthogonal to the  $k$ th subspace that CG minimizes over. This leads to the restriction that the first residual is orthogonal to  $\mathbf{W}$ , i.e.

$$\mathbf{W}^T \mathbf{r}_0 = 0.$$

Given an arbitrary initial guess  $\mathbf{x}_{-1}$  and defining the residual  $\mathbf{r}_{-1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{-1}$  we can choose  $\mathbf{x}_0$  to be

$$\mathbf{x}_0 = \mathbf{x}_{-1} + \mathbf{W} (\mathbf{W}^T \mathbf{A} \mathbf{W})^{-1} \mathbf{W}^T \mathbf{r}_{-1}. \quad (19)$$

Multiplying from left by  $\mathbf{W}^T \mathbf{A}$  gives

$$\begin{aligned} \mathbf{W}^T \mathbf{A} \mathbf{x}_0 &= \mathbf{W}^T \mathbf{A} \mathbf{x}_{-1} + \mathbf{W}^T (\mathbf{b} - \mathbf{A} \mathbf{x}_{-1}) \\ \mathbf{W}^T \mathbf{A} \mathbf{x}_0 &= \mathbf{W}^T \mathbf{A} \mathbf{b} \end{aligned} \quad (20)$$

$$\mathbf{o} = \mathbf{W}^T \mathbf{A} \mathbf{b} - \mathbf{W}^T \mathbf{A} \mathbf{x}_0 = \mathbf{W}^T \mathbf{r}_0. \quad (21)$$

From (20) it follows that  $\mathbf{x}_0$  is the exact solution of (1) in  $\mathcal{W}$  and therefore (Equation (21))  $\mathbf{r}_0$  is orthogonal to  $\mathcal{W}$ . If we use  $\mathbf{x}_0$  as the initial guess for CG, we obtain the InitCG method [15] illustrated in Algorithm 7.

If the columns of  $\mathbf{W}$  are exact eigenvectors then in exact arithmetic  $\mathcal{W}$  is orthogonal to  $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$  because the residuals will not have any components in the direction of the

**Algorithm 7:** InitCG

---

**Input:**  $\mathbf{A}$ ,  $\mathbf{x}_{-1}$ ,  $\mathbf{b}$ ,  $\mathbf{W}$

```

1  $\mathbf{r}_{-1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{-1}$ 
2  $\mathbf{x}_0 = \mathbf{x}_{-1} + \mathbf{Q}\mathbf{r}_{-1}$ 
3  $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ 
4  $\mathbf{p}_0 = \mathbf{r}_0$ 
5 for  $k = 0, \dots$ :
6    $\mathbf{s} = \mathbf{A}\mathbf{p}_k$ 
7    $\alpha_k = (\mathbf{r}_k^T \mathbf{r}_k) / (\mathbf{s}^T \mathbf{p}_k)$ 
8    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ 
9    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{s}$ 
10   $\beta_{k+1} = (\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}) / (\mathbf{r}_k^T \mathbf{r}_k)$ 
11   $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_{k+1} \mathbf{p}_k$ 
Output:  $\mathbf{x}_k$ 

```

---

eigenvectors spanning  $\mathcal{W}$ . However if  $\mathbf{W}$  does not consist of the exact eigenvectors or the computations are done in finite precision, this relation does not hold, and we need to use some sort of correction.

Our first problem is that  $\mathbf{p}_0 = \mathbf{r}_0$  is not necessarily  $\mathbf{A}$ -orthogonal to  $\mathcal{W}$ . If this is the case, then  $\mathbf{x}_1$  has components in  $\mathcal{W}$ . We resolve this by setting

$$\mathbf{p}_0 = \mathbf{P}\mathbf{r}_0.$$

Similarly, since  $\mathbf{r}_{k+1} = \mathbf{r}_0 - \mathbf{A}(\alpha_0 \mathbf{p}_0 + \dots + \alpha_k \mathbf{p}_k)$  we have to use the same trick as above, so that the update formula for descent direction becomes

$$\mathbf{p}_{k+1} = \mathbf{P}\mathbf{r}_{k+1} + \beta_{k+1} \mathbf{p}_k.$$

Effectively, we are making the search direction  $\mathbf{A}$ -conjugate to  $\mathcal{W}$  by projecting the components in  $\mathcal{W}$  out of the residual. This ensures that CG is not searching in  $\mathcal{W}$  but only in its  $\mathbf{A}$ -conjugate complement. Thus we have achieved the required splitting of the solution. The coefficient  $\beta_{k+1}$  is chosen so that it orthogonalizes  $\mathbf{r}_{k+1}$  against the previous direction, so should it change because we are orthogonalizing  $\mathbf{P}\mathbf{r}_{k+1}$  instead? No because, following the derivation of (14) and using that  $\mathbf{p}_k$  is  $\mathbf{A}$ -orthogonal to  $\mathcal{W}$  we have

$$\beta_{k+1} = -\frac{\mathbf{p}_k^T \mathbf{A}\mathbf{P}\mathbf{r}_{k+1}}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} = -\frac{\mathbf{p}_k^T \mathbf{A} \left( \mathbf{r}_{k+1} - \mathbf{W} \left( \mathbf{W}^T \mathbf{A}\mathbf{W} \right)^{-1} \mathbf{W}^T \mathbf{A}\mathbf{r}_{k+1} \right)}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} = -\frac{\mathbf{p}_k^T \mathbf{A}\mathbf{r}_{k+1}}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k}.$$

From derivation  $\alpha_k$  it follows that this coefficient does not have to be changed either.

Modifying Algorithm 7 so that we explicitly  $\mathbf{A}$ -orthogonalize the search directions with respect to  $\mathcal{W}$  gives us the deflated conjugate gradient (DCG) method as shown in Algorithm 8.



Algorithm 8: DCG	Algorithm: CG
<b>Input:</b> $A, x_{-1}, b, W$ <b>1</b> $P = I - QA$ <b>2</b> $r_{-1} = b - Ax_{-1}$ <b>3</b> $x_0 = x_{-1} + Qr_{-1}$ <b>4</b> $r_0 = b - Ax_0$ <b>5</b> $p_0 = Pr_0$ <b>6</b> <b>for</b> $k = 0, \dots$ : <b>7</b> $s = Ap_k$ <b>8</b> $\alpha_k = (r_k^T r_k) / (s^T p_k)$ <b>9</b> $x_{k+1} = x_k + \alpha_k p_k$ <b>10</b> $r_{k+1} = r_k - \alpha_k s$ <b>11</b> $\beta_{k+1} = (r_{k+1}^T r_{k+1}) / (r_k^T r_k)$ <b>12</b> $p_{k+1} = Pr_{k+1} + \beta_{k+1} p_k$ <b>Output:</b> $x_k$	<b>Input:</b> $A, x_0, b$ <b>1</b> $r_0 = b - Ax_0$ <b>2</b> $p_0 = r_0$ <b>3</b> <b>for</b> $k = 0, \dots$ : <b>4</b> $s = Ap_k$ <b>5</b> $\alpha_k = (r_k^T r_k) / (s^T p_k)$ <b>6</b> $x_{k+1} = x_k + \alpha_k p_k$ <b>7</b> $r_{k+1} = r_k - \alpha_k s$ <b>8</b> $\beta_{k+1} = (r_{k+1}^T r_{k+1}) / (r_k^T r_k)$ <b>9</b> $p_{k+1} = r_{k+1} + \beta_{k+1} p_k$ <b>Output:</b> $x_k$

### 3.1.1 Preconditioned DCG

We can also derive a preconditioned version of the previous algorithm. The derivation is done in the same way as in Section 2.5.3. Carrying this out yields preconditioned DCG (PDCG) illustrated in Algorithm 9.

Algorithm 9: PDCG
<b>Input:</b> $A, x_{-1}, b, W$ <b>1</b> $P = I - QA$ <b>2</b> $r_{-1} = b - Ax_{-1}$ <b>3</b> $x_0 = x_{-1} + Qr_{-1}$ <b>4</b> $r_0 = b - Ax_0$ <b>5</b> $z_0 = M^{-1}r_0$ <b>6</b> $p_0 = Pz_0$ <b>7</b> <b>for</b> $k = 0, \dots$ : <b>8</b> $s = Ap_k$ <b>9</b> $\alpha_k = (r_k^T z_k) / (s^T p_k)$ <b>10</b> $x_{k+1} = x_k + \alpha_k p_k$ <b>11</b> $r_{k+1} = r_k - \alpha_k s$ <b>12</b> $z_{k+1} = M^{-1}r_{k+1}$ <b>13</b> $\beta_{k+1} = (r_{k+1}^T z_{k+1}) / (r_k^T z_k)$ <b>14</b> $p_{k+1} = Pz_{k+1} + \beta_{k+1} p_k$ <b>Output:</b> $x_k$

### 3.2 Preconditioning Effect of Deflation

We derived the DCG method so that it splits the solution into two parts where the first part is given by (19). Therefore, we have

$$\mathbf{x} = \mathbf{x}_0 + \mathbf{u}, \quad (22)$$

where

$$\begin{aligned} \mathbf{u} &= \mathbf{x} - \mathbf{x}_0 = \mathbf{x} - \mathbf{x}_{-1} - \mathbf{W} \left( \mathbf{W}^T \mathbf{A} \mathbf{W} \right)^{-1} \mathbf{W}^T \mathbf{r}_{-1} \\ &= \mathbf{x} - \mathbf{x}_{-1} - \mathbf{W} \left( \mathbf{W}^T \mathbf{A} \mathbf{W} \right)^{-1} \mathbf{W}^T (\mathbf{b} - \mathbf{A} \mathbf{x}_{-1}) \\ &= \mathbf{x} - \mathbf{x}_{-1} - \mathbf{W} \left( \mathbf{W}^T \mathbf{A} \mathbf{W} \right)^{-1} \mathbf{W}^T \mathbf{A} (\mathbf{x} - \mathbf{x}_{-1}) \\ &= \mathbf{P} (\mathbf{x} - \mathbf{x}_{-1}). \end{aligned}$$

Premultiplying the above equation by  $\mathbf{P}$  and using the defining property of projectors ( $\mathbf{P}\mathbf{P} = \mathbf{P}$ ) yields

$$\mathbf{P}\mathbf{u} = \mathbf{P}\mathbf{P} (\mathbf{x} - \mathbf{x}_{-1}) = \mathbf{P} (\mathbf{x} - \mathbf{x}_{-1}) = \mathbf{u}. \quad (23)$$

Since  $\mathbf{x}_0$  in (22) is known we can rewrite the system (1) as

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{A}(\mathbf{x}_0 + \mathbf{u}) &= \mathbf{b} \\ \mathbf{A}\mathbf{u} &= \mathbf{b} - \mathbf{A}\mathbf{x}_0 = \mathbf{r}_0. \end{aligned} \quad (24)$$

By (23),  $\mathbf{u}$  is a solution of the deflated system

$$\mathbf{A}\mathbf{P}\mathbf{y} = \mathbf{r}_0.$$

Conversely, given a solution  $\mathbf{y}$ , we can set  $\mathbf{u} = \mathbf{P}\mathbf{y}$ , that solves (24) and at the same time obeys (23). Therefore, the solution of (1) can take the form

$$\mathbf{x} = \mathbf{x}_0 + \mathbf{P}\mathbf{y}.$$

To recover the solution  $\mathbf{x}$ , we premultiply the equation by  $\mathbf{P}\mathbf{A}$  which yields

$$\mathbf{P}\mathbf{A}\mathbf{x} = \mathbf{P}\mathbf{A}\mathbf{x}_0 + \mathbf{P}\mathbf{A}\mathbf{P}\mathbf{y} = \mathbf{P}\mathbf{A}\mathbf{x}_0 + \mathbf{P}\mathbf{r}_0 = \mathbf{P}\mathbf{A}\mathbf{x}_0 + \mathbf{P}(\mathbf{b} - \mathbf{A}\mathbf{x}_0) = \mathbf{P}\mathbf{b}. \quad (25)$$

On the other hand, premultiplying by  $\mathbf{P}^T \mathbf{A}$  gives

$$\mathbf{P}^T \mathbf{A}\mathbf{x} = \mathbf{P}^T \mathbf{A}\mathbf{x}_0 + \mathbf{P}^T \mathbf{A}\mathbf{P}\mathbf{y} = \mathbf{P}^T \mathbf{A}\mathbf{x}_0 + \mathbf{P}^T \mathbf{r}_0 = \mathbf{P}^T \mathbf{A}\mathbf{x}_0 + \mathbf{P}^T (\mathbf{b} - \mathbf{A}\mathbf{x}_0) = \mathbf{P}^T \mathbf{b}. \quad (26)$$

This suggests that the projectors  $\mathbf{P}$  and  $\mathbf{P}^T$  can be thought of as "preconditioners".

Moreover, applying PCG described in Algorithm 6 to solve (1) with the preconditioner  $M^{-1} = PP^T$ , i.e. solving

$$PP^T Ax = PP^T b \quad (27)$$

using the initial guess defined by (19) is equivalent to DCG. By induction, it is obvious that  $W^T r_k = 0$ , therefore  $P^T r_k = r_k$ . From this it follows that the update formulas for the search directions  $p_k$  are the same. By the same argument, we have

$$r_k^T z_k = r_k^T PP^T r_k = r_k^T r_k,$$

and therefore the coefficients  $\alpha_k$  and  $\beta_k$  are also equivalent.

Additionally, it follows by a straightforward comparison and with some rewriting (carried out e.g. in [16]) that the DCG method is equivalent to the standard CG method applied to the linear system

$$P^T APx = P^T b. \quad (28)$$

The equivalence to CG and PCG described above suggests that the rate of convergence of DCG depends on the spectrum of  $P^T AP$  or equivalently of  $PP^T A$ . In fact we will show that the spectra of all of the linear system operators defined in (25), (26), (27) and (28) are equivalent. We have

$$AP = A(I - QA) = A - AQA = (I - AQ)A = P^T A.$$

Since  $\sigma(AP) = \sigma(PA)$  we conclude that  $\sigma(PA) = \sigma(P^T A)$ . Premultiplying the previous equation by  $P^T$  it follows that

$$P^T AP = P^T P^T A = P^T A.$$

If we instead postmultiply the equation by  $P$ , we have

$$APP = AP = P^T AP.$$

Therefore, it was shown that

$$\sigma(PA) = \sigma(P^T A) = \sigma(PP^T A) = \sigma(P^T AP).$$

Now, assume that the columns of the deflation matrix  $W$  are exact eigenvectors of  $A$ . Then immediately

$$P^T AW = AW - AW(W^T AW)^{-1}W^T AW = O = \text{diag}(0, \dots, 0)W,$$

i.e. the columns of  $W$  are eigenvectors of  $P^T A$  belonging to  $\lambda = 0$  eigenvalues. Moreover, if  $\lambda$  and  $v$  is an eigenpair of  $A$  but  $v$  is not a column of  $W$  then thanks to the symmetry of  $A$

we have  $\mathbf{W}^T \mathbf{v} = \mathbf{o}$  and also  $\mathbf{W}^T \mathbf{A} \mathbf{v} = \mathbf{o}$ , therefore

$$\mathbf{P}^T \mathbf{A} \mathbf{v} = \mathbf{A} \mathbf{v} - \mathbf{A} \mathbf{W} \left( \mathbf{W}^T \mathbf{A} \mathbf{W} \right)^{-1} \mathbf{W}^T \mathbf{A} \mathbf{v} = \mathbf{A} \mathbf{v} = \lambda \mathbf{v}.$$

In other words DCG operator has the same spectrum as  $\mathbf{A}$  except that the eigenvalue belonging to eigenvector comprising  $\mathbf{W}$  are shifted to zero. As was mentioned in Section 2.5.2, the CG method ignores the space spanned by the null space of the operator. This allows us to consider the effective condition number

$$\mathcal{K}_{eff} = \frac{\lambda_{max}}{\lambda_{min}},$$

where  $\lambda_{max}$  and  $\lambda_{min}$  is respectively the maximal and the minimal non-zero eigenvalue of  $\mathbf{P}^T \mathbf{A}$  or one of the spectrally equivalent operators.

### 3.2.1 Shifting the Eigenvalues

We saw that if  $\mathbf{W}$  consists of the exact eigenvectors of  $\mathbf{A}$ , then the associated eigenvalues are shifted to zero. However, if the deflated eigenvectors are only approximate, then the associated eigenvalues might not be zeroed out completely but be just very small instead. The eigenvalues close to zero can significantly slow down the convergence as was discussed in Section 2.5.2.

It was suggested in [17] that we can add a correction factor  $\mathbf{Q}$  to projector  $\mathbf{P}$  leading to a so-called projector with coarse problem correction

$$\mathbf{P}_s = \mathbf{P} + \mathbf{Q}.$$

We can straightforwardly replace  $\mathbf{P}$  in DCG with  $\mathbf{P}_s$ . More care has to be taken for PDCG, see [17] for the resulting algorithm. Similarly to above we have

$$\begin{aligned} \mathbf{P}_s^T \mathbf{A} \mathbf{W} &= \mathbf{A} \mathbf{W} - \mathbf{A} \mathbf{W} \left( \mathbf{W}^T \mathbf{A} \mathbf{W} \right)^{-1} \mathbf{W}^T \mathbf{A} \mathbf{W} + \mathbf{W} \left( \mathbf{W}^T \mathbf{A} \mathbf{W} \right)^{-1} \mathbf{W}^T \mathbf{A} \mathbf{W} \\ &= \mathbf{W} = \text{diag}(1, \dots, 1) \mathbf{W}, \end{aligned}$$

and since  $\mathbf{Q}$  is orthogonal to eigenvectors not in  $\mathbf{W}$ , we can show in the same way as above that the rest of the eigenvalues are not changed. Therefore, using  $\mathbf{P}_s$  leads to the operators having the same spectrum as  $\mathbf{A}$  except that the eigenvalues belonging to the columns of  $\mathbf{W}$  are shifted to one.

Since

$$\mathbf{P}_s = \mathbf{I} - \mathbf{Q} \mathbf{A} + \mathbf{Q} = \mathbf{I} - \mathbf{Q} (\mathbf{A} - \mathbf{I}),$$

the cost of applying  $\mathbf{P}_s$  compared to  $\mathbf{P}$  is one more vector-vector addition.

We note, however, that this approach can be problematic because in some case it can create an unfavourable spectrum. For example, let the extremal eigenvalues of  $\mathbf{A}$  be  $\lambda_{min} =$

$10^2$  and  $\lambda_{max} = 10^4$  and assume that the eigenvector belonging to  $\lambda_{max}$  is not a column in  $\mathbf{W}$ , then this approach creates a new isolated eigenvalue  $\lambda = 1$  and the effective condition number of DCG is hundred times worse than that of  $\mathbf{A}$ . Therefore we suggest multiplying the correction factor  $\mathbf{Q}$  in  $\mathbf{P}_s$  by a constant  $C$  that does not create the unfavourable spectrum. A good choice is an eigenvalue belonging to an eigenvector not in  $\mathbf{W}$  as the deflated eigenvalues will coalesce into the chosen non-deflated eigenvalue. If the eigenvectors in  $\mathbf{W}$  are inexact, then the deflated eigenvalues will create eigenvalue cluster near  $C$ , and we can still expect a good convergence.

The numerical experiments in [17] showed that  $\mathbf{P}_s$  also has a stabilization effect when the projections  $\mathbf{P}$  and especially the application of the inverse in  $\mathbf{P}$  are computed with low accuracy.

### 3.3 DCG Coarse Problem

The inverse in the projector  $\mathbf{P}$  is called coarse problem (CP) while  $\mathbf{W}^T \mathbf{A} \mathbf{W}$  is called coarse problem matrix. We will employ a direct solver for solving CP.

Assuming a row-wise distribution of matrices, the rows of the CP matrix are distributed among the same number of cores we are using to solve the linear system. However, the dimension of CP is smaller than the dimension of  $\mathbf{A}$ , quite often significantly (even just a few rows). If we tried to solve this problem by a fully parallel approach, the cost of communication, as well as the required time, could be extremely high. To solve this problem, we employ the same strategy that was successfully used for the solution of the FETI method CP [18–21]. We create MPI sub-communicators, and then we copy the whole matrix into each sub-communicator distributing it over the available ranks in each sub-communicator. This allows us to factorize the CP matrix redundantly on each sub-communicator. The forward and backward solves are done by scattering the whole input vector into each sub-communicator leading also to redundant solves on sub-communicators. Apart from scattering the input vector into sub-communicator the communication in the solves is restricted to the sub-communicators. Therefore, the number of sub-communicators effectively controls the level of parallelization of the CP solution. It was pointed out in [18] that assuming the computational cores are assigned to the sub-communicators contiguously then the sub-communicator approach exploits data locality and can be thought of as a communication avoiding technique.

#### 3.3.1 Required Accuracy for CP Solution

While we remarked that CP would be solved by a direct solver and therefore with full machine precision, it will prove useful to have a look at the accuracy level that is actually needed for the CP solution.

Given the relative tolerance  $\epsilon$  for the outer iteration, the numerical experiments in [22] showed that to get the same convergence as that obtained by using a direct solver, the

stopping criterion of the inner solver (CG) used for solving CP has the form

$$\|\mathbf{r}_k^{inner}\| \leq c\epsilon \|\mathbf{b}^{inner}\|, \quad 0 < c \leq 1.$$

It was shown in [23], using the theory developed for the inexact Krylov subspace methods [24, 25], that this accuracy is needed only in the first few iterations of the outer solver and can be relaxed as we are getting closer to the solution of the original system. Their stopping criterion has the form of

$$\|\mathbf{r}_k^{inner}\| \leq \frac{c\epsilon}{\|\mathbf{r}_i^{outer}\|} \|\mathbf{b}^{inner}\|, \quad c > 0.$$

The constant  $c$  is guaranteed to exist. However, we do not know the upper bound - the value that would lead to maximal stopping criteria relaxation while keeping the number of iterations required by the outer solver to converge same as when CP is solved directly. We call DCG that uses this stopping criterion adaptive precision DCG.

### 3.3.2 Nested DCG

Given a hierarchy of the deflation matrices  $\mathbf{W}_k$ ,  $k \in \{1, \dots, n\}$  such that

$$\mathbf{W}_1^T \mathbf{A} \mathbf{W}_1 \tag{29}$$

is a coarse problem matrix, and

$$\mathbf{W}_2^T \mathbf{W}_1^T \mathbf{A} \mathbf{W}_1 \mathbf{W}_2$$

is even coarser. We assume that this hierarchy continues until the coarsest problem matrix reads

$$\mathbf{W}_n^T \cdots \mathbf{W}_2^T \mathbf{W}_1^T \mathbf{A} \mathbf{W}_1 \mathbf{W}_2 \cdots \mathbf{W}_n.$$

Then we can use DCG to solve (1) with  $\mathbf{W}_1$  being the deflation matrix. If  $\mathbf{W}_1$  is large, then it would be very costly to factorize (29). Instead, we can again use DCG but this time to solve (29) where the deflation matrix will be  $\mathbf{W}_2$ . We can nest DCG solvers for CP until the coarsest level with the coarsest problem matrix which is hopefully small enough that we can solve it easily with a direct method.

To nest DCG solvers for the CP solution was suggested in [26]. Using nested DCG with the stabilising effect provided by the shifted projector  $\mathbf{P}_s$  and with the adaptive precision could lead to significant speed-up of the application of deflation thanks to the cheap CP solution on the coarsest level and the reduction of the number of iteration required by the inner solvers.

## 4 Deflation Spaces

The choice of the deflation space is crucial because if we choose the space properly, DCG can be significantly faster.

There are several factors that need to be considered. The deflation matrix should be readily obtainable. The CP should be easily solvable. And our deflation space should significantly improve time to solution.

### 4.1 Eigenvectors

We already saw in Section 3.2 that if we create the deflation space from eigenvectors, we effectively hide (deflate) the associated eigenvalues from the CG method.

Obviously, we should deflate the eigenvectors that are slowing down the convergence the most. As discussed in Section 2.5.2 these are quite often the eigenvectors belonging to the smallest eigenvalues [8, 16]. In general, the eigenvectors belonging to extremal eigenvalues are a good choice [27] as it can lead to a significant decrease of the effective condition number.

Note that if the eigenvalues we try to deflate are in a cluster and we do not deflate the whole cluster we will not see essentially any improvement in the convergence rate.

The good thing about the eigenvector-based deflation is that in general, we can see significant improvements in the convergence with relatively small deflation matrices. Therefore the CP solution is cheap.

We also note, that if the columns of  $\mathbf{W}$  have associated eigenvalues  $\lambda_i, \lambda_k, \dots$  we can essentially for free obtain  $\mathbf{A}\mathbf{W}$  or  $\mathbf{W}^T\mathbf{A}$  by just scaling the columns of  $\mathbf{W}$  or rows of  $\mathbf{W}^T$  respectively by the appropriate eigenvalues, i.e.

$$\mathbf{A}\mathbf{W} = \mathbf{W} \text{diag}(\lambda_i, \lambda_k, \dots) \quad \text{or} \quad \mathbf{W}^T\mathbf{A} = \text{diag}(\lambda_i, \lambda_k, \dots)\mathbf{W}^T. \quad (30)$$

However, the problem is how to obtain the eigenvectors. In general, finding the approximate eigenvectors is very costly. In our experience Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) [28, 29] is fast if we have a good preconditioner. However, the solution of the linear system (1) with this good preconditioner is even faster, and there is essentially no need to use DCG. Since we quite often do not have a good preconditioner we generally used the Jacobi-Davidson (JD) method on subspaces as described in [30]. This method is fairly robust, but it is also quite slow.

If we are solving the same linear system with multiple right-hand sides, we can use the equivalence of CG with the Lanczos method (see Section 2.5.1) to obtain eigenvectors approximation at the end of CG/DCG solves [16]. A significant improvement upon the idea is the eigCG algorithm [31].

### 4.2 Subdomain Aggregation

The subdomain aggregation was introduced in [12] and successfully applied in e.g. [32–34]

The idea is that if we solve a problem using a finite element method, we can split the computational domain into a number of non-overlapping subdomains. Then each subdomain contributes a single vector to  $\mathbf{W}$  that contains ones on the indices of the grid points belonging to the subdomain and zeros otherwise. Note that the CP matrix then aggregates the components of  $\mathbf{A}$  on subdomains.

It was pointed out in [33] that the subdomain aggregation often approximate the small eigenvalues of  $\mathbf{A}$ .

The choice of subdomains should take into account irregularities in the computational domain, e.g. jumps in coefficients [34].

### 4.3 Discrete Wavelet Compression

It was observed in [26] that since the CP matrix represents a coarse grid approximation of  $\mathbf{A}$  we can create the CP matrix by a discrete wavelet compression.

The idea of a discrete wavelet compression using the fast wavelet transform (FWT) [35–37] can be described as follows. Assume that the input we want to compress is a matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ , and that number of rows  $n$  is divisible by 2. First, we create an orthonormal projector onto a scaling subspace with the decomposition low-pass filter coefficients  $h_1, \dots, h_k$  in each row shifted by two positions against the previous row, i.e.

$$\mathbf{H}_{1,n} = \begin{pmatrix} h_1 & h_2 & h_3 & \dots & 0 & \dots & 0 & 0 \\ 0 & 0 & h_1 & h_2 & \dots & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ h_{k-1} & h_k & 0 & 0 & 0 & \dots & h_{k-3} & h_{k-2} \end{pmatrix} \in \mathbb{R}^{\frac{n}{2} \times n}.$$

Notice the filter is wrapped around when it overflows the matrix dimension. Similarly, we can create projector onto a wavelet subspace with the same structure but with the decomposition high-pass filter  $g_1, \dots, g_k$  in each row, i.e.

$$\mathbf{G}_{1,n} = \begin{pmatrix} g_1 & g_2 & g_3 & \dots & 0 & \dots & 0 & 0 \\ 0 & 0 & g_1 & g_2 & \dots & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ g_{k-1} & g_k & 0 & 0 & 0 & \dots & g_{k-3} & g_{k-2} \end{pmatrix} \in \mathbb{R}^{\frac{n}{2} \times n}.$$

Note that the number of columns of each projector is the same as the number of rows of the matrix that we want to compress, while the number of rows is just a half. First index describes the projector level, and will be explained later. The second one tells us the number of rows in the input matrix. Finally, we can create a transformation matrix

$$\mathbf{M}_{1,n} = \begin{pmatrix} \mathbf{H}_{1,n} \\ \mathbf{G}_{1,n} \end{pmatrix} \in \mathbb{R}^{n \times n}.$$



Applying transformation matrix  $\mathbf{M}_{1,n}$  from left and its transpose from right on the input matrix, we obtain

$$\mathbf{M}_{1,n} \mathbf{A} \mathbf{M}_{1,n}^T = \begin{pmatrix} \mathbf{H}_{1,n} \mathbf{A} \mathbf{H}_{1,n}^T & \mathbf{H}_{1,n} \mathbf{A} \mathbf{G}_{1,n}^T \\ \mathbf{G}_{1,n} \mathbf{A} \mathbf{H}_{1,n}^T & \mathbf{G}_{1,n} \mathbf{A} \mathbf{G}_{1,n}^T \end{pmatrix} \in \mathbb{R}^{n \times n}.$$

The resulting matrix contains all information of the input. However, the first block of the resulting matrix contains the most useful information (so-called trends), while the other blocks contains just fine details of the input. This is because the application of  $\mathbf{H}$  cuts of the high frequencies, while  $\mathbf{G}$  cuts of the low frequencies of the signal it is applied on. Therefore, assuming that  $n/2$  is divisible by 2, we can create another transformation matrix  $\mathbf{M}_{1,n/2}$ , and apply it in the same way as before, but now just on the first block

$$\begin{aligned} & \mathbf{M}_{1,n/2} \mathbf{H}_{1,n} \mathbf{A} \mathbf{H}_{1,n}^T \mathbf{M}_{1,n/2}^T = \\ & \begin{pmatrix} \mathbf{H}_{1,n/2} \mathbf{H}_{1,n} \mathbf{A} \mathbf{H}_{1,n}^T \mathbf{H}_{1,n/2}^T & \mathbf{H}_{1,n/2} \mathbf{H}_{1,n} \mathbf{A} \mathbf{H}_{1,n}^T \mathbf{G}_{1,n/2}^T \\ \mathbf{G}_{1,n/2} \mathbf{H}_{1,n} \mathbf{A} \mathbf{H}_{1,n}^T \mathbf{H}_{1,n/2}^T & \mathbf{G}_{1,n/2} \mathbf{H}_{1,n} \mathbf{A} \mathbf{H}_{1,n}^T \mathbf{G}_{1,n/2}^T \end{pmatrix} \in \mathbb{R}^{\frac{n}{2} \times \frac{n}{2}}. \end{aligned}$$

Again, the first block contains the most useful information and now its dimension is a quarter of the original. We can obtain this block without creating the matrices  $\mathbf{G}_{1,*}$  and  $\mathbf{M}_{1,*}$ . Moreover, we can directly assemble the product of  $\mathbf{H}_{1,n/2} \mathbf{H}_{1,n}$  as a second level projection matrix

$$\mathbf{H}_{2,n} = \mathbf{H}_{1,n/2} \mathbf{H}_{1,n}.$$

A two-level wavelet decomposition using Haar wavelet is illustrated in Figure 3. Notice that most of the information is contained in the upper left part of the image. If we zeroed out all of the resulting image except the upper left 1/16th, and then we reconstructed the picture [35], the resulting image would still look reasonably well. This is the basic idea behind using wavelets for data compression.

Assuming that  $n$  is divisible by  $2^m$  we can create up to  $m$  level projection matrix  $\mathbf{H}_{m,n}$  in the same way.

On the other hand, if  $n$  is not divisible by  $2^m$  and we would still like to use  $m$  level projection matrix, we have to employ some form of extension of wavelet transforms for arbitrary lengths of input data, see, e.g. [40]. We implemented the truncated and extended filter matrices. The truncation leads to errors on the boundaries while extension leads to redundancy. In our use case, we did not observe any significant difference between the two. To construct these wavelet projections, let  $N$  be the number of the filter coefficients, the number of rows for the truncated matrix is  $r = \lceil n/2 \rceil$  and it is  $r = \lfloor (n + N - 1) / 2 \rfloor$  for the extended matrix. Let  $k = N - 2$ . Now we can create a larger variant of the deflation



Figure 3: Example of a two-level decomposition (right) [38] using Haar wavelet of Lenna test picture (left) [39]

projection matrix as

$$\widetilde{\mathbf{H}} = \begin{pmatrix} h_1 & h_2 & h_3 & \dots & 0 & \dots & 0 & 0 \\ 0 & 0 & h_1 & h_2 & \dots & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \end{pmatrix} \in \mathbb{R}^{r \times 2r+k}.$$

Now set  $c = k + 1$  for the extended matrix and  $c = k/2 + 1$  for the truncated matrix. Then our filter matrix  $\mathbf{H}_{1,n} \in \mathbb{R}^{r \times n}$  is obtained by taking all columns from  $c$ th column up to  $(c + n)$ th column from the previous matrix  $\widetilde{\mathbf{H}}$ .

If the columns of  $\mathbf{W}$  are eigenvectors belonging to the smallest eigenvalues, then the DCG coarse problem consists of the lowest frequency components of  $\mathbf{A}$ . Similarly, our input matrix transformed by  $\mathbf{H}_{m,n}$  projection, contains the lowest frequency components of  $\mathbf{A}$ . Then in a sense, we can set

$$\mathbf{W}^T \mathbf{A} \mathbf{W} = \mathbf{H}_{m,n} \mathbf{A} \mathbf{H}_{m,n}^T,$$

and therefore we can choose our deflation space as a transpose of the  $m$  level projection matrix onto the scaling subspace, i.e.  $\mathbf{W} = \mathbf{H}_{m,n}^T$ . Note that this choice allows us to use the nested DCG.

We implemented a general function to assemble these scaling matrices with arbitrary filter coefficients. Our implementation can currently generate the matrices with Haar, 4, 8, and 16 coefficients Daubechies (db4,db8,db16), Biorthogonal 2.2, and discrete Meyer (FIR approximation) filters. An illustration of the implemented scaling functions can be found in Figure 4.

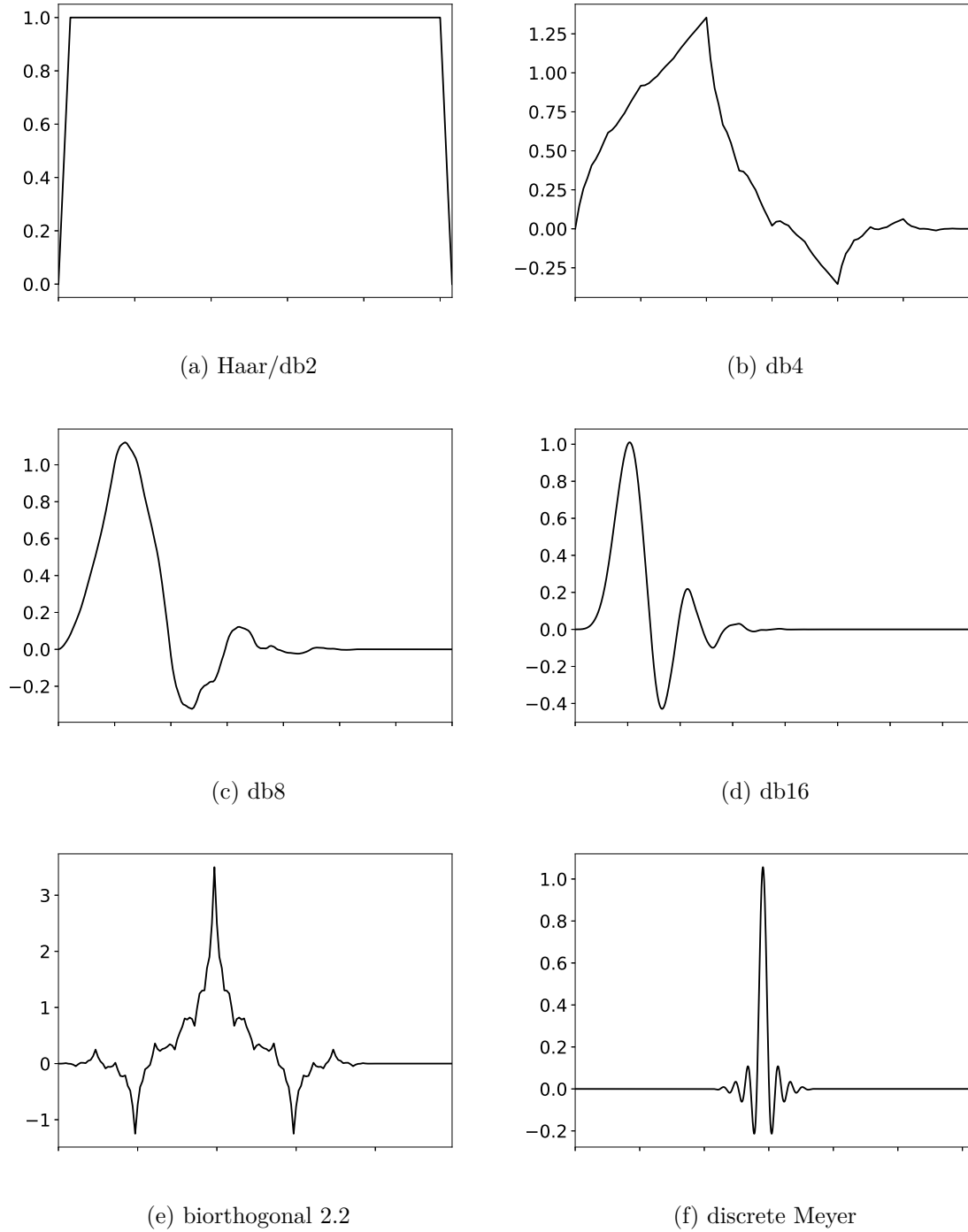


Figure 4: Implemented scaling functions

Haar wavelet is especially interesting as it has only two coefficients, both of them equal to  $1/\sqrt{2}$ . Given the level  $m$ , we can observe that the compressed operator effectively aggregates  $2^m$  entries in the input matrix. Therefore, it can be viewed as an algebraic subdomain aggregation.

Note that the presented discrete wavelet compression was obviously designed for dense matrices. We can expect that the behaviour of the wavelet compression will depend highly on the way the filters fit the sparsity pattern of the input matrix.

#### 4.4 Multigrid Prolongation and Restriction

In the previous section, we noted that the CP matrix consists of the lowest frequency components of  $\mathbf{A}$  or equivalently it contains the more coarse view of the initial matrix. The same CP matrix arises in multigrid [11, 41]. In fact, the similarity of DCG and two-grid iteration was noted in [17, 23, 42].

Therefore, we can set  $\mathbf{W}$  to be the multigrid prolongation operator making  $\mathbf{W}^T$  the restriction operator. Moreover, our wavelet-based deflation matrices are also used as algebraic multigrid operators [43]. Therefore we conclude that essentially any geometric or algebraic prolongation matrices from multigrid might be used as a deflation matrix.

Similarly to the wavelet-based deflation, we can create a hierarchy of prolongation matrices. We reuse the notation of the previous section and refer to the individual prolongation matrices by their level in the hierarchy. Note that this choice of the deflation matrix also allows for nesting DCGs.

Note that the resulting operator is essentially a multigrid operator without smoothers.

## 5 Libraries and HPC environment

This section provides a brief overview of the numerical libraries and HPC computational environment used to obtain the numerical results.

### 5.1 PETSc

The Portable, Extensible Toolkit for Scientific Computation (PETSc) [44–46] is a suite of data structures and routines providing building blocks for the implementation of scientific applications suitable for parallel computers. It is written in C and uses the MPI standard for parallelization. The parallelization comes mostly from a row-wise distribution of the matrices over a number of computational cores. The building blocks that PETSc provides are for example parallel vectors and matrices or a number of parallel solvers for linear, non-linear or optimization problems. PETSc also has many nice features like automatic profiling or a possibility to change applications behaviour by command line options.

Our implementation of DCG method described in the next section is built as a PETSc linear solver.

The newest version of PETSc (3.9.0) was used for all the numerical experiments.

### 5.2 PERMON

PERMON (Parallel, Efficient, Robust, Modular, Object-oriented, Numerical) [47] is a scalable software toolbox for solution of quadratic programming (QP) problems. It is based on PETSc, and it follows its highly successful design. The main module PermonQP includes data structures, transformations, algorithms, and supporting functions for QP. Other modules include PermonFLLOP implementing FETI type domain decomposition and PermonSVM implementing linear support vector machines.

Our DCG implementation is part of PERMON.

### 5.3 SuperLU

It was mentioned in Section 3.3 that we employ a direct solver for the CP solution. In our experience, a supernodal LU factorization approach is generally the fastest, and it scales reasonably well. If the CP is small, we use a sequential solve provided by SuperLU [48]. If on the other hand, CP is larger we employ several cores for its solution using SuperLU\_DIST [49].

### 5.4 SLEPc

SLEPc [7, 50], the Scalable Library for Eigenvalue Problem Computations, is a PETSc-based software library for the solution of large-scale sparse eigenvalue problems on parallel computers.

We used it for the computation of the eigenvector deflation spaces using either the JD method on subspaces [30] or LOBPCG [28] implemented in BLOPEX [29].

The version 3.9.0. was used.

## 5.5 MFEM

MFEM [51] is a free, lightweight, scalable C++ library for finite element methods. It supports wide variety of finite element spaces, various discretization techniques, etc. It employs HYPRE [52] for most of the linear algebra routines and ParMETIS/METIS 5 [53] for sub-domain partitioning.

Several examples in the numerical experiments were constructed using the library.

We used the newest version 3.3.2.

## 5.6 ARCHER

The numerical tests were run on the ARCHER supercomputer [54]. ARCHER is a Cray XC30 based supercomputer that consists of 4920 compute nodes. With each compute node containing two 2.7 GHz, 12-core Intel E5-2697 v2 (Ivy Bridge) processors, the total number of cores available on ARCHER is 118,080. Each compute node has at least 64 GB of memory. Compute nodes are interconnected by the Aries interconnect using a Dragonfly topology. According to the current (November 2017) TOP500 list [55], the ARCHER is the 79th most powerful supercomputer with Rmax of 1642.5 TFlop/s in the Linpack benchmark.

For each library, we used the following modules. As a compiler, the cce/8.6.5 (Cray Compiler Environment) was used. An MPI library was provided by the cray-mpich/7.7.0 module. The module libsci/17.12.1 was used as an implementation of the BLAS, LAPACK and ScaLAPACK routines. The Cray Third Party Scientific Libraries (TPSL) collection version 17.11.1 provided SuperLU 5.2.1, SuperLU\_DIST 5.1.3, HYPRE 2.11.2 and METIS 5.1.0.

## 6 Implementation

In this section, we describe the implementation of the DCG method. The method was implemented as a linear solver (KSP) in PETSc with the KSP type being KSPDCG. The developed implementation is available in the PERMON toolbox on its GitHub page [56].

The implementation uses the linear algebra building blocks (`Mat`, `Vec`, `IS`, etc. routines) provided by PETSc and some useful utilities provided by PERMON.

### 6.1 Solver Settings

Since KSPDCG implements the KSP interface the common options are available, e.g., `-ksp_monitor` command line option for monitoring the residuals. All the available parameters can be obtained by setting KSPDCG as the solver and running with `-help`. We will describe some of these command line options available for our implementation.

KSPDCG implements at the same time (preconditioned) InitCG and PDCG (DCG is obtained by setting PCNONE as the preconditioner). By default PDCG is used. To use InitCG the user can request it with the `-ksp_dcg_initcg` option.

To use the correction described in Section 3.2.1 which is by default not employed, one can set the `-ksp_dcg_correct` option.

The adaptive precision shown in Section 3.3.1 is also not used by default. User can request it by setting `-ksp_dcg_adaptive`, the constant  $c$  defaults to one and can be changed by `-ksp_dcg_adaptive_const`.

To control the maximal number of nested DCGs (Section 3.3.2), user can set `-ksp_dcg_max_nested_level`. If the hierarchy of the deflation operators contains more than this number of matrices then the the smallest matrices are multiplied with each other to create a single grouped operator. The default value is zero, i.e. no nesting of DCGs is done.

#### 6.1.1 Setting the Deflation Matrix

User can set the deflation matrix or its transpose with the `KSPDCGSetDeflation()` function. If the matrix set is an implicit product matrix (`MATPROD` that is equivalent to multiplicative `MATCOMPOSITE`) then the nested DCG can be used.

If the deflation matrix is not set then the transpose of the 1-level Haar deflation matrix is computed. A computation of a different basis can be requested by `-ksp_dcg_compute_space`. The possible options for wavelet based deflation matrices are:

- `haar` – Haar deflation matrix with  $\lceil n/2 \rceil$  rows where the filter is cut off on the last row of the deflation matrix, if it does not fit
- `jackethaar` – same as `haar` except that if the filter on the last row does not fit the last two rows are constructed as a 2-point Jacket-Haar, see [57]
- `db2` – 2 Daubechies coefficients (Haar)

- **db4** – 4 Daubechies coefficients
- **db8** – 8 Daubechies coefficients
- **db16** – 16 Daubechies coefficients
- **biorth22** – Biorthogonal 2.2 (6 coefficients)
- **meyer** – Discrete Meyer (FIR Approximation) (62 coefficients)

The number of the deflation levels defaults to one but can be changed by `-ksp_dcg_compute_space_size`. We always directly assemble **haar** and **jackethaar** spaces. The others are assembled on their individual levels, but the whole operator is kept as an implicit product matrix. This allows us to use them in the nested DCG scheme. By default these filters are truncated, by setting the option `-ksp_dcg_space_extended` the filters are extended instead.

Another deflation matrix that is possible to compute is the subdomain aggregation. It is obtained by using the **aggregation** argument of the option. This assumes that each core owns the whole subdomain. This is, e.g. the case when a domain is partitioned by METIS into subdomains assigned to different cores, and then each core computes "locally" its own part of the linear system.

If SLEPc is available, then we can also specify **slepc** as the argument of the option. This creates a SLEPc solver for the eigenvalue problems (EPS) and tries to compute the number of eigenvectors specified by `-ksp_dcg_compute_space_size` associated with the smallest eigenvalues. The options for the solver can be changed using the standard prefix `-eps_`. The argument **slepc-cheap** works in the same way but it also directly assemble **AW** by scaling the obtained eigenvectors by their eigenvalues as was shown in (30).

## 6.2 Cost of Matrices Assembly and Operations

For a good parallel implementation, it is necessary to ascertain whether some matrix operators should be assembled and how costly are their applications. For example, **AW** has significantly fewer columns than **A** so that  $\mathbf{W}^T \mathbf{A}$  applied by matrixTranspose-vector multiplication could be faster than matrix-vector multiplication by **A** followed by matrix-vector multiplication by  $\mathbf{W}^T$ . On the other hand, assembled **AW** can be expected to be denser and therefore, matrix-vector operations with this operator might require more communication than applying the operators one after the other. Of course, the above discussion applies only to large and sparse deflation matrices, like those created by wavelet compression described in Section 4.3.

Several numerical experiments were performed to show the costs of various operations involving deflation matrices. Time in seconds needed by the operations was used as a metric because it best captures the costs of both computation and communication. While these tests were done only on a single benchmark, they should represent the broad trends with reasonable accuracy. These experiments were run on Archer (see Section 5.6). Linear elasticity



benchmark, described in Section 7.1.3, with db4 wavelet-based deflation (Section 4.3) with 1, 3, 5, and 7 wavelet compression levels, was chosen as a problem with non-trivial linear system matrix. The results are reported for a medium sized problem in Table 1. The number of cores was chosen so that there were, respectively, at least 20,000, 40,000, and 80,000 degrees of freedom (DOFs) per core.

A number of operations was tested. These included matrix-vector operations reported in the upper half of the table. The operations with operator MATRIX are denoted as "Mv(MATRIX)" and "MTv(MATRIX)" for matrix-vector (`MatMult()` in PETSc) and transpose matrix-vector (`MatTransposeMult()`) product, respectively. `MATRIXt` represents an explicit transpose of MATRIX. The prefix "i" symbolises that the operation is implicit, i.e. the operator MATRIX is not explicitly assembled. Note that times for the matrix-vector operations are multiplied by  $10^3$ . Similarly, in the lower part of the table, purely matrix operations are reported. First, the sum of the times needed for individual assembly of all of the deflation matrices as either implicit  $\mathbf{W}$  or  $\mathbf{W}^T$  is shown. Then we have the cost of the explicit assembly of  $\mathbf{W}$  or  $\mathbf{W}^T$ . Note, that there is no appreciable difference between the assembly from local matrices or their transpose. For completeness, the cost of the explicit transpose of the deflation matrix and its transpose is also shown. Next section of the table illustrates the cost of explicitly forming the  $\mathbf{W}\mathbf{A}$  and  $\mathbf{W}^T\mathbf{A}$  with (transpose) matrix-matrix products (`Mat(Transpose)MatMult()`). The last section of the tables represents various ways to assemble CP. Not yet described operations are "MMM" representing `MatMatMatMult()` function and "PTAP" that corresponds to `MatPtAP()` which should be optimal for the assembly of CP.

Based on the reported results, it seems that the application of  $\mathbf{W}$  and  $\mathbf{W}^T$  should be done implicitly. It does not make much of a difference, whether we store the individual wavelet-based matrices or their transpose. Considering the application of  $\mathbf{W}^T\mathbf{A}$ , it is again faster to multiply individually by  $\mathbf{A}$  and then by  $\mathbf{W}^T$  than to use the assembled operator. In the previous cases, the advantage of the implicit operators lies in their sparsity (less communication is required). On the other hand, it is definitely worth to assemble  $\mathbf{W}^T\mathbf{A}\mathbf{W}$  because, while denser, the dimension is significantly reduced. If CP is solved with a direct solver, it has to be assembled anyway. If on the other hand, we use an iterative solver, we do not have to assemble CP. However, for the worst case (1 level wavelet compression) it starts to pay off to assemble the operator after less than 300 applications. With increasing number of the compression levels, it starts to pay off much sooner (generally tens of applications).

The actual assembly of both individual and global deflation matrices is extremely cheap. The assembly of the individual wavelet-based deflation matrices in the transposed form has the advantage of being purely local.

With the previous discussion of the results in mind, the actual implementation computes wavelet-based individual deflation matrices as their transpose, creating implicit  $\mathbf{W}^T$ . From these individual matrices,  $\mathbf{W}$  is explicitly assembled. For the action of  $\mathbf{W}$  and its transpose, the implicitly represented  $\mathbf{W}^T$  is used. CP is directly formed by `MatPtAP()`, the memory taken by explicitly formed  $\mathbf{W}$  is then freed, leaving more space for possibly solving CP by a

Cores	2,558							1,279							639						
Deflation lvl.	1	3	5	7	1	3	5	7	1	3	5	7	1	3	5	7					
iMv(W)	0.3916	0.5584	1.2752	0.8579	0.9831	2.4702	2.0027	2.3616	2.1713	4.4017	5.0102	4.4949									
iMTv(Wt)	0.4100	0.5997	1.3185	0.8552	1.0581	2.5269	1.7582	1.8550	2.4382	4.0346	3.5657	3.5415									
Mv(W)	1.4352	1.2357	1.5621	2.4653	2.2599	1.9288	1.9210	5.1121	9.1056	7.7666	6.5115	6.5573									
MTv(Wt)	1.3831	1.1781	1.6072	2.4348	2.3339	1.9825	2.4401	5.5185	9.0067	7.7150	6.6660	8.8667									
iMv(Wt)	0.4544	0.5883	1.3857	0.7754	0.9871	2.6629	1.4815	1.6146	2.5424	3.4210	3.5115	3.4936									
iMTv(W)	0.4188	0.5591	1.3815	0.7801	0.9403	2.6087	1.5202	1.4554	2.3930	3.2335	3.7325	3.4385									
Mv(Wt)	4.0796	4.1424	3.8020	3.9230	4.3512	4.1788	3.1849	5.5428	11.9400	15.2360	15.1900	9.0805									
MTv(W)	3.5960	3.7861	2.9852	3.0245	2.4067	2.2822	2.4062	4.7297	8.9755	7.7082	6.6378	8.9818									
MTv(AW)	12.1250	11.7540	11.0120	9.4175	25.5820	24.3000	22.4010	19.4030	52.4960	50.3380	44.0570	36.5380									
Mv(WtA)	12.4700	12.1690	9.8372	9.4815	27.7540	22.4230	20.7330	19.0310	58.0600	50.2760	45.4230	40.8960									
Mv(A)	5.5373	5.4425	5.3949	5.5367	11.3700	11.1090	11.2000	11.1390	25.9410	24.7640	24.7170	24.6620									
Mv(WtAW)	8.3512	4.3328	1.6100	0.3220	16.4270	8.1494	3.0492	0.7407	32.2210	15.7280	5.7741	1.7728									
iW sum	0.0285	0.0384	0.0697	0.0540	0.0360	0.0770	0.0516	0.0585	0.0508	0.0698	0.0685	0.0720									
iWt sum	0.0125	0.0203	0.0430	0.0288	0.0161	0.0512	0.0296	0.0306	0.0311	0.0385	0.0515	0.0485									
W assemb.	0.0000	0.0861	0.1943	0.1080	0.0000	0.2341	0.1240	0.1092	0.0000	0.2354	0.2186	0.2170									
Wt assemb.	0.0000	0.0723	0.1312	0.1563	0.0000	0.2608	0.3124	0.3947	0.0000	0.8800	1.1360	1.2917									
Trasnp(W)	0.0337	0.1074	0.0632	0.0577	0.0478	0.0742	0.0696	0.0684	0.0573	0.0814	0.0858	0.0858									
Trasnp(Wt)	0.0316	0.0194	0.0234	0.0160	0.0424	0.0367	0.0274	0.0259	0.0561	0.0521	0.0563	0.0494									
MM(A,W)	0.8408	0.8014	0.3654	0.3148	1.8112	1.6082	0.7377	0.6316	3.7726	1.6490	1.3707	1.2366									
MTM(W,A)	1.1158	1.2547	1.6365	3.1685	1.9332	2.2403	2.9207	6.4156	3.1401	3.7599	5.2783	12.5240									
MM(Wt,A)	0.6307	0.7349	0.9705	1.9415	1.3131	1.4058	1.9175	4.0770	2.9614	2.7859	3.6983	8.2849									
MMM(Wt,A,W)	1.4835	1.4360	1.1929	1.0328	3.0295	2.8075	2.3248	2.1120	6.3686	5.1301	4.5752	4.2520									
PTAP(A,W)	1.4987	0.9073	0.3899	0.2414	2.2781	1.7612	0.7602	0.4808	4.4569	2.6712	1.4810	0.9881									
MM(WtA,w)	1.5242	2.7982	0.4832	0.1744	3.0403	1.7598	0.9837	0.4048	6.1739	3.3202	1.9737	0.7983									
MM(Wt,AW)	0.5860	0.5328	0.2310	0.1485	1.1228	0.6984	0.4514	0.3115	2.3712	1.2034	0.8811	0.6302									
MTM(AW,W)	2.7271	3.2013	2.6935	1.4623	4.8334	6.4712	6.4511	4.2479	9.3536	13.1960	14.4400	11.4220									

Table 1: Times in seconds of matrix assembly and operations involving deflation matrices for linear elasticity benchmark with 51,171,075 DOFs. **Vector operation times are scaled by  $10^3$**  and represents the average over 100 operations.

direct method. The same implementation is used for other large and sparse deflation matrices.

The situation is different for a dense deflation matrix with relatively few columns (like those obtained from eigenvectors based deflation). In this case, in order to assemble  $\mathbf{W}^T \mathbf{A} \mathbf{W}$  we first assemble  $\mathbf{A} \mathbf{W}$  and then using the `MatTransposeMatMult()` operation, we obtain the CP matrix. The advantage of this approach is that the cost of applying  $\mathbf{W}^T$  using  $\mathbf{W}$  is the same as applying  $\mathbf{W}^T \mathbf{A}$  using already assembled  $\mathbf{A} \mathbf{W}$ , i.e. we save the cost of the matrix-vector multiplication by  $\mathbf{A}$  in every application of the deflation projector. Note, that for eigenvectors based deflation we can assemble  $\mathbf{A} \mathbf{W}$  cheaply thanks to Equation (30).



## 7 Numerical Experiments

We tested the implementation on several benchmarks described in Section 7.1. In Section 7.2 we compare InitCG and DCG convergence and investigate the choice of deflation space, the effect of chosen levels for wavelet-based and multigrid-based deflation spaces, the nested DCG with and without corrections, and finally we showcase the scalability of DCG. All tests were done on the ARCHER supercomputer (Section 5.6).

### 7.1 Benchmarks

This section introduces the benchmarks used to evaluate selected options of the implementation. Our aim was to provide a wide variety of benchmarks so that the various aspects of the method can be appropriately tested. We made the benchmarks available on GitHub [58] in order to make our results reproducible.

#### 7.1.1 SuiteSparse Matrix Collection

The SuiteSparse Matrix Collection (formerly known as the University of Florida Sparse Matrix Collection), is a large and actively growing set of sparse matrices that arise in real applications [59, 60].

The collection provides matrices from a wide range of domains that include structural engineering, computational fluid dynamics, circuit simulation, power networks, financial modelling, etc. Moreover, the benchmarks are repeatable and quite often comparable. These are the reasons why it is widely used for tests of various linear solvers.

If the benchmark included a right-hand side it was used, otherwise we used a constant vector with the Euclidean norm of 1.

#### 7.1.2 MFEM Example 1: Laplace Equation

This benchmark is based on Example 1 included in MFEM (Section 5.5). It solves the Laplace equation

$$-\Delta \mathbf{u} = 1$$

with homogeneous Dirichlet Boundary condition. We used a square with a disc-shaped hole in the middle discretized with triangular elements as an input mesh. Figure 5 illustrates the solution.

To understand how we obtain some of the deflation spaces, let us briefly describe a common MFEM workflow. First, a small sequential mesh is loaded, and we do several sequential mesh refinements. After that, we partition the mesh using METIS among available cores. Then we generally refine the mesh further in parallel. With each parallel refinement we are able to generate a geometric multigrid prolongation operator facilitating transfers between the coarser and the refined grid. After we refined the mesh enough, we describe a weak form including boundary conditions of our problem and set the appropriate finite

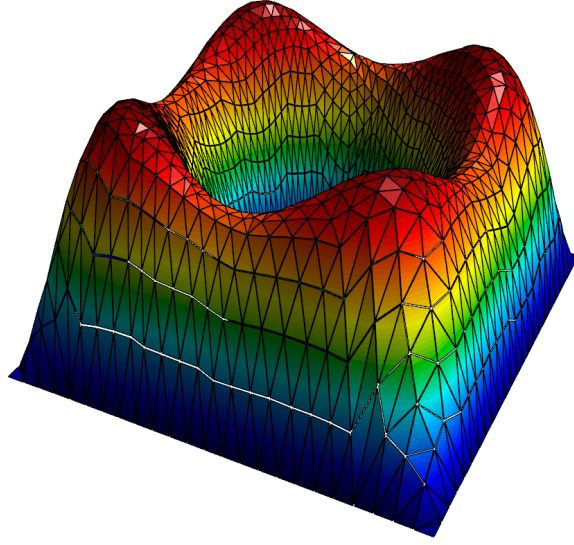


Figure 5: Laplace: solution

element collection and space. Then we can finally assemble the system matrix and right-hand side in parallel and mostly locally. Note that the subdomain aggregation used in the numerical experiments works by aggregating the unknowns/DOFs on the subdomains defined by METIS partitioning of the domain.

### 7.1.3 MFEM Example 2: Linear Elasticity

The benchmark is adapted version of MFEM Example 2 that solves a linear elasticity problem describing multi-material cantilever beam with the weak form of

$$-div(\sigma(\mathbf{u})) = 0,$$

where

$$\sigma(\mathbf{u}) = \lambda div(\mathbf{u})\mathbf{I} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$$

is the stress tensor corresponding to the displacement field  $\mathbf{u}$ , and  $\lambda$  and  $\mu$  are the material Lamé constants. One side of the beam is fixed the other is pulled down by a constant force. The geometry is illustrated by Figure 6. We used hexahedral finite elements for discretization. The solution is depicted in Figure 7.

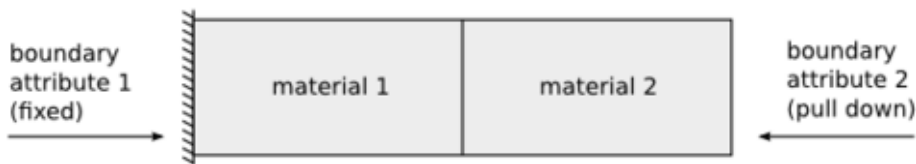


Figure 6: Linear elasticity: geometry [51]

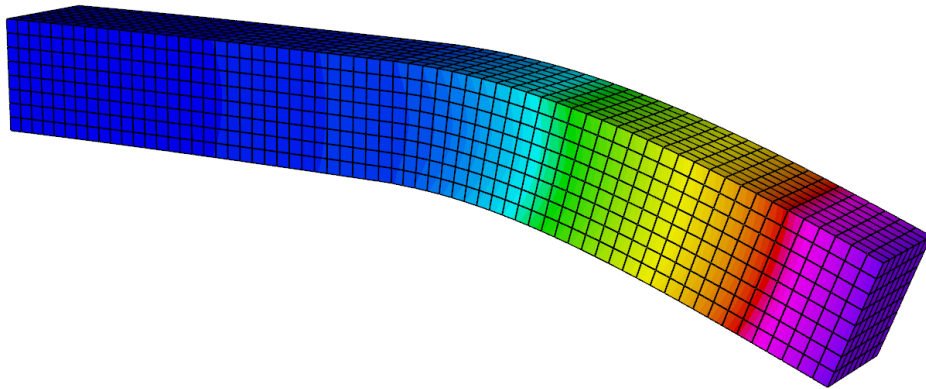


Figure 7: Linear elasticity: solution

#### 7.1.4 2D BEM Laplace

The last benchmark, kindly provided by Dalibor Lukáš [61], represents the boundary element method (BEM) discretization of the Laplace equation on an L-shaped domain (a square with one quarter missing). The right-hand-side is a constant vector with the Euclidean norm of 1.

The benchmark was chosen because it generates a dense matrix.

## 7.2 Results

This section investigates various aspects of the implementation. For brevity, most of these aspects are reported only for the Laplace benchmark. All times are reported in seconds.

Unless stated otherwise, the solver settings for the tests are as follows. The initial guess was a null vector. Maximum number of iterations was set to 30,000, The stopping criterion is given by relative residual  $\|\mathbf{r}_i\| / \|\mathbf{r}_0\| < \epsilon$  for CG and  $\|\mathbf{r}_i\| / \|\mathbf{r}_{-1}\| < \epsilon$  for DCG, ensuring a fair comparison. The tolerance is usually  $\epsilon = 10^{-6}$ .

The eigenvectors are computed by the JD method implemented in SLEPc (Section 5.4) with maximum subspace dimension (ncv) set to 400, convergence tolerance of  $\epsilon \cdot 10^{-2}$  and the maximum number of iterations set to 10,000. We always compute the eigenvectors belonging to the smallest eigenvalues.

Throughout the section we use several abbreviations:

- DTOL – solver is diverging; residual norm increased by more than  $10^4$  from its initial value
- DSF – direct solver failed (this is an old problem with direct solvers compiled with the Cray compiler, a factorization can fail depending on the number of cores and sub-communicators); happily, this did not occur often
- EDC – eigensolver did not converge or provided less than the number of requested eigenvectors
- ITS – solver reached the maximum number of iterations without converging

- NA – the estimated conditioned number is not available (CG did not converge), or deflation space is not available/applicable

We also abbreviate names of the deflation spaces as follows. Space "None" represents CG. We use "eigN" where N is some number to represent a deflation space comprising of N eigenvectors. The prolongation multigrid operators are denoted by "mg". The subdomain aggregation space is abbreviated "agg". The wavelet-based deflation uses the abbreviations described in Section 6.1.1.

### 7.2.1 InitCG vs DCG

In this section, we briefly compare the convergence of InitCG and DCG. We can expect that for eigenvector deflation we will obtain the same convergence if our space is a good approximation of the exact eigenvector space. We present the results for the deflation space with 5 and 40 eigenvectors, as well as for 1-level db2 as a representative of the rest of the available spaces that we observed share similar behaviour. The results are reported for the Laplace benchmark with 79,616 DOFs in Table 2 and for the linear elasticity benchmark with 316,928 DOFs in Table 3. We also varied the eigensolver relative tolerance. The value 0 in InitCG row means that DCG was used.

Space	none	db2	db2	eig5	eig5	eig5	eig40	eig40	eig40
InitCG	NA	0	1	0	1	1	0	1	1
Eig. Tol.	NA	NA	NA	1e-2	1e-2	1e-4	1e-2	1e-2	1e-4
Iters.	858	124	568	609	640	609	272	492	277

Table 2: InitCG(1) vs DCG(0): Laplace benchmark

Space	none	db2	db2	eig5	eig5	eig5	eig5	eig40	eig40	eig40	eig40
InitCG	NA	0	1	0	1	0	1	0	1	0	1
Eig. Tol.	NA	NA	NA	1e-2	1e-2	1e-8	1e-8	1e-2	1e-2	1e-8	1e-8
Iters.	2103	847	4176	2442	2919	1645	1645	2224	757	723	724

Table 3: InitCG(1) vs DCG(0): Linear elasticity benchmark

Interestingly, DCG in the Laplace benchmark is not very sensitive to the eigensolver tolerance while this is certainly not true in the case of the elasticity benchmark. On the other hand, supporting well the theory, InitCG performs comparably to DCG only if the eigenvectors are computed with a high enough precision.

As for the wavelet-based db2 deflation, we can observe that it performs poorly in InitCG.

### 7.2.2 Choice of Deflation Space

As was mentioned in Section 4, the choice of the deflation space is a crucial factor in the convergence of DCG. As such we tested the method on all available benchmarks.



In these tests, we used for the wavelet-based deflation and multigrid prolongation only a single level.

First, we tested DCG on 236 matrices that, at the time of writing, represents all the available SPD matrices in the SuiteSparse collection. The results are reported in Appendix A. The table includes id and name of the matrix in the collection, problem dimension, estimated condition number, and the number of iterations needed by CG and DCG with various deflation spaces to converge. Only eigenvector and wavelet-based deflation are available for these matrices.

For each matrix, if CG did not converge, we reduced the relative tolerance to  $\epsilon = 10^{-4}$ . Such cases are highlighted in the results by a light grey background. There were 47 matrices that did not converge with either CG or DCG even with the reduced precision. Their list is available at the top of the results.

In the following, we analyse the results and highlight the behaviour on some of the matrices. We refer to the matrices by their id:name (short description) trio. CG converged in 170 cases. On average (counting only cases when CG also converged), db2 brought a speed-up of iteration by a factor of 4.5 and converged for 177 matrices while eig40 with the total of 162 converged cases averaged the iterations speed-up of about 4.0 without counting an outlier. The maximum factor was observed in 1899:tmt\_sym (electromagnetics problem) for both eig40 and db2, where CG took 3,042 iterations to converge while db2 took 27 iterations (112 times speed-up) and eig40 took only a single iteration. Interestingly, there were other (non-trivial) benchmarks where, in this case, db2 needed just a single iteration as well. These were 1331:Muu (structural problem) and 2259:thermomech\_dM (deformation of a steel cylinder) which both needed about 50 iterations for CG to converge.

There were some matrices for which we could see that increasing the wavelet filter size decreases the number of iteration, i.e. CP better approximates the original matrix. We observed this behaviour on, e.g. 1883:ecology2 (using electrical network theory to model animal movement and gene flow) that needed 5,393 CG iterations while DCG with db2 required only 226 iterations and was further reduced to 193 iterations for the largest, discrete Meyer, filter. But in general, the results show that db2 often outperforms other wavelets or is just slightly slower. However, since db2 is more sparse and therefore cheaper to work with than the other wavelet filters, we generally give preference to it.

Let us draw the attention to some matrices where DCG performed exceptionally well. Eigenvector deflation was very successful for 1435:gyro (model reduction problem) achieving convergence in 1,680 iterations while CG needed 28,927. CG applied to 69:bcsttm19 (structural problem, part of a suspension bridge) took modest 473, but DCG with db2 solved the same problem in just 23 iterations. Discrete Meyer wavelet-based deflation reduced for 427:ex3 (computational fluid dynamics) the number of iterations from 10,073 achieved by CG to only 402.

There were also several matrices that did not converge with standard CG but converged with DCG. The most striking example of this is 2664:bundle\_adj (sparse bundle adjustment

Benchmark	DOFs	none	eig5	eig40	agg	mg	db2
Laplace	79,616	858	609	272	545	22	124
Laplace	316,928	1,781	1,247	560	1093	21	193
Laplace	1,264,640	3,680	2,548	1,148	2,221	21	278
Elasticity	15,795	1,130	853	390	2,825	249	763
Elasticity	111,843	2,103	1,645	724	5,097	245	847
Elasticity	839,619	3,962	3,200	1415	9,811	243	902
BEM	200	24	25	26	NA	NA	5
BEM	2,000	57	54	59	NA	NA	4
BEM	20,000	112	113	116	NA	NA	2

Table 4: Number of iterations for various deflation spaces and benchmarks

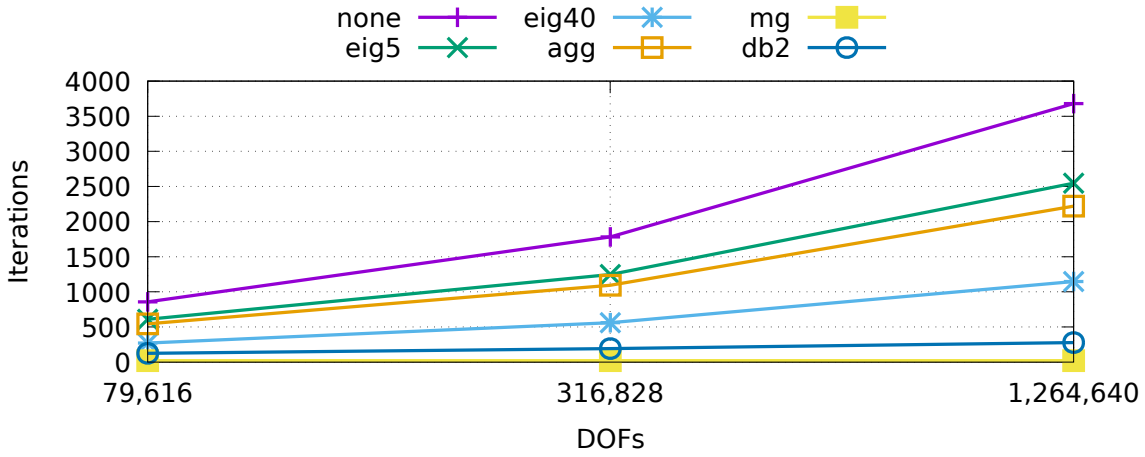


Figure 8: Number of iterations for various deflation spaces using Laplace benchmark

problem) where CG did not converge in 30,000 iterations, but DCG with db8 took only 32 iterations to converge.

The results for the Laplace, linear elasticity and BEM benchmarks (computed with 24 cores) are reported in Table 4 with visualization provided in Figures 8 to 10. These results include from wavelet-based deflations only db2. In Table 5 we also compare the efficiency of different wavelet scaling functions.

Let us analyse the results of the sparse (Laplace and Elasticity) benchmarks first. We specifically emphasised that 24 cores were used for this test because it determines the number of subdomains for the aggregation deflation space. A rule of thumb when computing in parallel with sparse matrices is that we should have at least 20,000 rows per core [62] to have enough work for each core so that the communication does not outweigh the computation. A large amount of rows per core is unfavourable to the subdomain aggregation based on METIS partitioning because the aggregated space is probably going to be too coarse to capture the essence of the fine grid well enough. We can see a slightly faster convergence of the subdomain aggregation than when using the eig5 deflation space on the Laplace benchmark. On the other

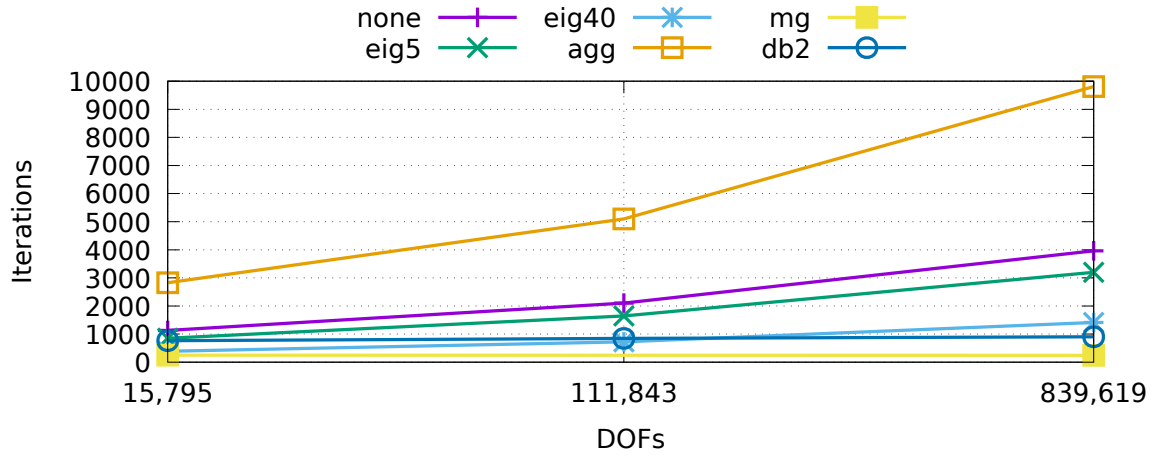


Figure 9: Number of iterations for various deflation spaces using linear elasticity benchmark

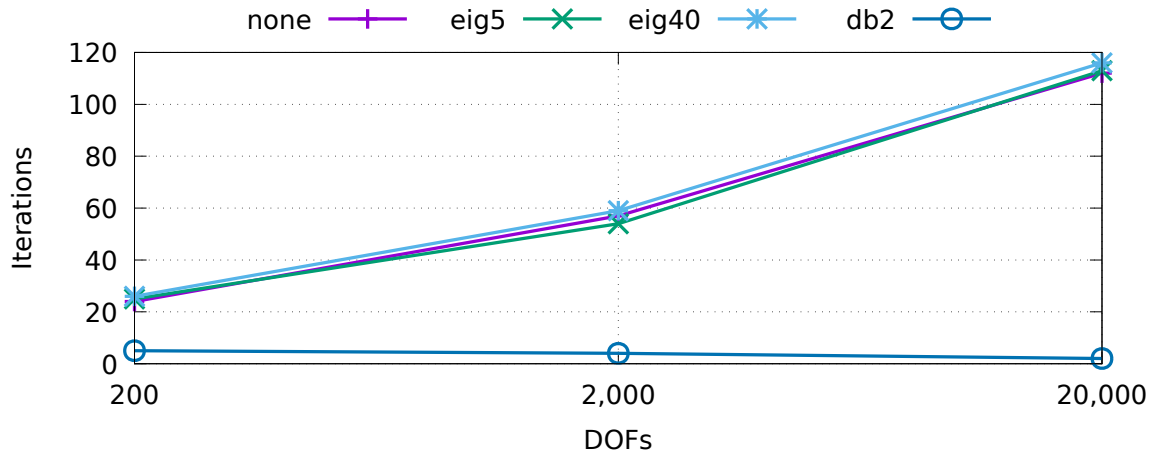


Figure 10: Number of iterations for various deflation spaces using BEM benchmark

Benchmark	DOFs	none	db2	db4	db8	db16	biorth	meyer
Laplace	79,616	858	124	110	105	104	114	101
Laplace	316,928	1,781	193	161	160	152	168	150
Laplace	1,264,640	3,680	278	240	231	227	249	221
Elasticity	15,795	1,130	763	764	737	741	747	742
Elasticity	111,843	2,103	847	802	790	780	817	782
Elasticity	839,619	3,962	902	872	840	841	896	DFS
BEM	200	24	5	6	6	7	6	6
BEM	2,000	57	4	4	5	6	4	4
BEM	20,000	112	2	2	3	4	2	2

Table 5: Number of iterations for various wavelet-based deflation spaces and benchmarks

hand, the subdomain aggregations perform very poorly on the linear elasticity benchmark. However, this is probably more due to the inability of a simple subdomain aggregation to capture the rigid body modes rotations of the beam rather than due to the small number of subdomains.

The eigenvectors in both sparse benchmarks perform reasonably well. The problem is that the number of iterations with increasing problem size increases with roughly the same speed as is increasing the number of iterations needed by CG. Meaning that we would ideally increase the number of eigenvectors used for deflation when increasing the problem size. However, the drawback is that not only we would need to compute more eigenvectors, but the cost of the eigensolvers grows with the size of the system.

The wavelet-based deflation does not have this problem. The number of iterations grows when the problem size increases but the growth is fairly modest. Overall, it performs very well.

Multigrid prolongation operator performs the best. We can see the multigrid property of keeping the number of iterations constant. However, the problem with both multigrid and wavelet-based deflation is that the cost of CP is increasing with increasing problem size.

Finally, we turn our attention to the BEM benchmark. Deflating eigenvectors does not seem to have any effect. This is probably due to not hiding the whole cluster of small eigenvalues with the chosen size of the deflation space.

On the other hand, wavelet-based deflation performs on this benchmark exceptionally well. In fact, we can see that the number of iterations is actually slightly decreasing with increasing problem size.

### 7.2.3 Level of Deflation Matrices

We would expect to see a deterioration of convergence when we increase the number of levels of multigrid prolongation and wavelet-based deflation matrices because the CP matrix becomes a worse approximation of the original operator. This behaviour is illustrated in Table 6 and Figure 11 using db2 deflation space for the Laplace benchmark with 20,197,376 DOFs computed on 120 cores. The results are reported with a near-optimal redundancy parameter (number of sub-communicators used for CP solution).

Increasing the level of deflation matrices decreases the CP size making the factorization and the subsequent solves faster. The drawback is the increase in the number of iterations. The presented results suggest that we need to balance these two considerations to obtain the fastest time to the solution.

Level	Iterations	Time	Time per Iteration	Redundancy	CP size
none	15,574	174.3	0.0112	none	none
13	5,374	122.1	0.0227	5	2,466
12	5,302	124.4	0.0235	5	4,931
11	5,214	125.9	0.0241	5	9,862
10	4,996	131.8	0.0264	5	19,724
9	4,686	143.1	0.0305	2	39,448
8	4,094	135.1	0.0330	2	78,896
7	3,491	147.3	0.0422	1	157,792
6	2,828	174.3	0.0616	1	315,584

Table 6: The effect of increasing levels of the deflation matrices illustrated on 20,197,376 DOFs Laplace Benchmark with db2 deflation computed on 120 cores

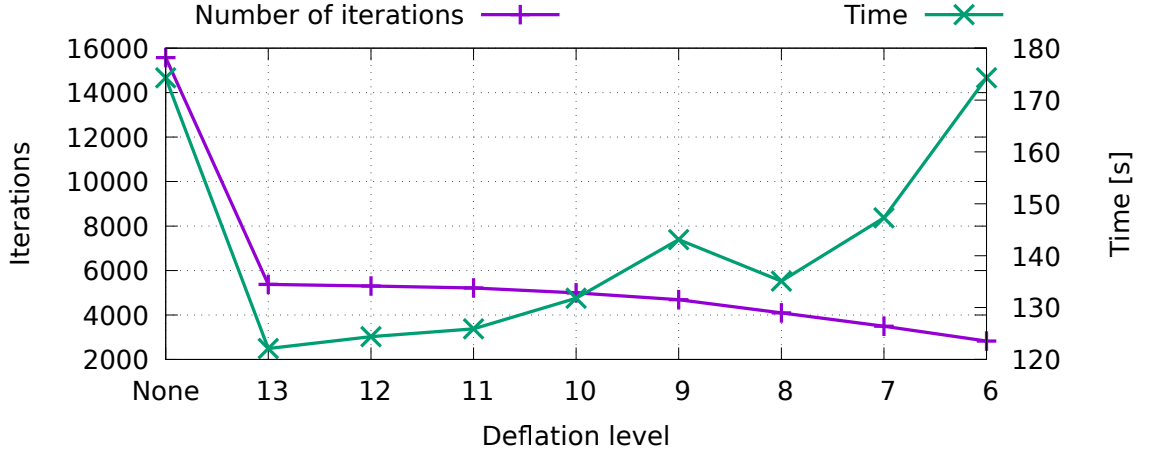


Figure 11: The effect of increasing levels of the deflation matrices illustrated on 20,197,376 DOFs Laplace Benchmark with db2 deflation computed on 120 cores

#### 7.2.4 Potential of Adaptive Precision Nested DCG with CP Corrections

While doing tests with nested deflation matrices, we discovered that using the CP corrections allows us to decrease the required accuracy of the nested DCGs massively. Our results are reported for 4-level multigrid deflation space on the Laplace benchmark with 79,616 DOFs computed on 24 cores in Table 7.

Note that in this case, the nested DCG scheme is not faster than using 4-level multigrid prolongation directly (in this case DCG converges with 156 iterations in 2.15 s). For comparison, a single level multigrid deflation took 21 iterations in 6.98 s.

However, the results are still very interesting. If we did not use the adaptive precision, then, since we use multigrid prolongation matrices, we would have about 21 iterations in each nested solve, i.e. a total of  $21^2$ ,  $21^3$  and  $21^4$  for respectively levels 1, 2 and 3. However, thanks to the adaptive precision, we average, with constants equal to 1, respectively less than 12, 8 and 6 iterations per outer iteration for the consecutive levels.

If we set the adaptive precision constants above 1 and do not use the CP corrections,

Adaptive constant			Iterations				Time
lvl1	lvl2	lvl3	DCG	lvl1	lvl2	lvl3	
1	1	1	21	240	1,857	11,008	21.96
8	5	10	21	305	912	1,420	4.45
8	5	50	21	1,069	731	323	3.69
8	5	75	21	1,557	732	203	4.29
8	5	100	21	2,244	775	139	5.32
8	5	200	21	3,604	901	45	7.49
8	12	75	22	1,633	583	117	4.22
8	12	100	22	2,081	648	70	4.98
8	15	75	407	1,552	576	100	6.05
8	15	100	460	2,040	580	58	7.06

Table 7: Illustrating the effect of adaptive precision nested DCG with various adaptive precision constants on the Laplace benchmark with 79,616 DOFs computed on 24 cores using mg deflation

we would not have guaranteed number of iterations to be the same as in the case when direct solver is used. Using the CP corrections, we can significantly reduce the accuracy while keeping the convergence of the outermost DCG constant. Moreover, we can reduce the number of innermost iterations significantly. The innermost iterations are relatively expensive due to the cost of the triangular solves needed by CP. However, while we can keep constant the number of iterations of the outermost DCG and reduce the number of iterations on the last and the second to last levels, the number of iterations on the first nested level is increasing.

If we reduce the accuracy too much, we see an increase in the number of iterations even on the outermost level. The problem is that at present we do not have a clear idea how to choose the adaptive precision constants depending on the benchmark, number of levels, etc.

However, looking at the results, we see that from the point of the convergence of the outermost DCG it is not necessary to go to the coarsest levels in each iteration. We can perhaps expand upon the idea. Recall that from our derivation of the method we projected the current residual so that the new descent direction is  $\mathbf{A}$ -orthogonal to the deflation space. This suggests that perhaps we should look at some measure of  $\mathbf{A}$ -orthogonality of the residual. Since we have to compute

$$\mathbf{W}^T \mathbf{A} \mathbf{r}_k$$

before we solve CP, we could look at the norm of this expression because if it is small, we know that the residual is almost  $\mathbf{A}$ -orthogonal to the deflation space and therefore there is no need to carry out the projection. Unfortunately, we so far do not know what can be considered as a small enough value of the norm.

### 7.2.5 Parallel Scalability

This section presents the parallel scalability of our implementation. We used the Laplace and linear elasticity benchmarks with the number of cores chosen so that the number of rows is respectively about 20, 40, 80 and 160 thousand per core. Optimal settings for deflation level and redundancy are reported.

The results for the Laplace benchmark with 20,197,376 DOFs can be found in Table 8 and Figure 12.

As we mentioned earlier, computation of eigenvectors is very expensive. We used the fact that DCG in the Laplace benchmark is not sensitive to the accuracy of the approximated eigenvectors (see Table 2) by setting eigensolver tolerance to  $10^{-2}$ . However, even then the computation of 40 eigenvectors on this larger benchmark was too costly making the solution time almost 20 times slower than that of CG.

The presented results show that in this case CG almost achieves strong scaling. Using mg deflation space yields the best performance (about 10 to 25 times faster than CG). However, it does not scale very well because to achieve optimal ratio of computation and communication in CP we are forced to keep about 1000 rows of the CP matrix per core regardless of the number of cores. Making the cost of the CP matrix factorization essentially constant.

While we are forced to treat CP in the same way also for db2 deflation, DCG with this space achieves super-linear scaling. However, the db2 deflation is...well, let us say slightly cheating. As we mentioned in Section 7.1.2, METIS partitions the input mesh into subdomains, and we then assemble the linear system matrix mostly locally (and using the subdomain local numbering of DOFs). If we increase the number of cores, we increase the number of subdomains which leads to a different permutation of the resulting matrix. In the case of the Laplace benchmark, this permutation is favourable to db2 in the sense that the number of iterations decreases with increasing number of cores (see Table 8). The speed-up for db2 over CG was up to 3.

We also tried to increase the size of the Laplace benchmark, but in order to achieve a reasonable convergence of CG, we had to drop the relative tolerance to  $\epsilon = 10^{-4}$ . We also took the opportunity to showcase the possibility to couple preconditioning and deflation. In this case, we employed a simple Jacobi preconditioner. The results for the Laplace benchmark with 80,764,928 DOFs are reported in Table 9 and Figure 13.

The scalability results are similar to the previous example. The effect of the Jacobi preconditioner has a comparable efficiency in both CG and DCG. The mg deflation again performs the best with speed-up over CG between 11 and 32. The wavelet-based deflation db2 achieved speed-up between 2 and 4.

The last scalability test was done on the linear elasticity benchmark with 51,171,075 DOFs. The results can be found in Table 10 and Figure 14.

In this case, the db2 deflation did not work in the sense that it did not reduce the number of iterations enough to offset the cost of the CP factorization and solves. Overall it was

Cores	Space	Deflation lvl.	Redundancy	Iterations	Time
120	none	none	none	15,574	172.47
240	none	none	none	15,575	77.98
504	none	none	none	15,575	42.90
1,008	none	none	none	15,574	25.79
120	db2	13	5	5,374	121.30
240	db2	13	20	4,090	40.24
504	db2	13	42	2,986	16.09
1,008	db2	12	42	2,277	8.41
120	mg	4	1	154	6.92
240	mg	4	3	154	4.34
504	mg	5	21	319	3.32
1,008	mg	5	42	319	2.37
120	eig40	NA	120	5,403	3330.16

Table 8: Parallel scalability for 20,197,376 DOFs Laplace benchmark

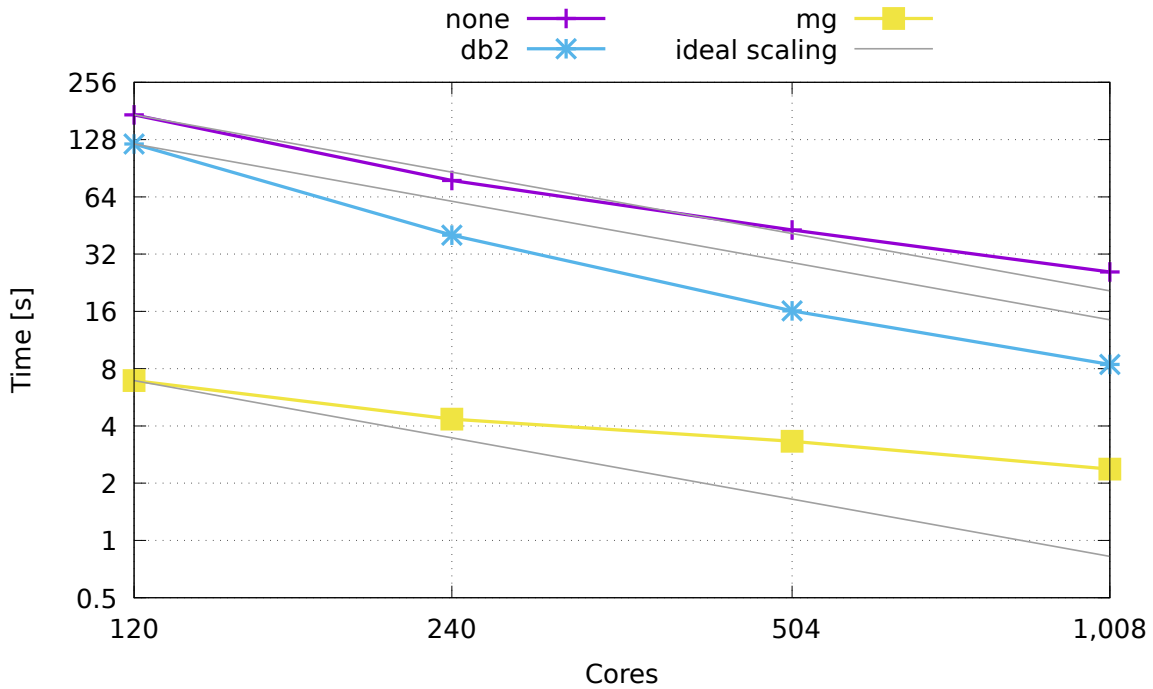


Figure 12: Parallel scalability for 20,197,376 DOFs Laplace benchmark



Cores	Space	Precond.	Deflation lvl.	Redundancy	Iterations	Time
504	none	none	none	none	22,549	254.08
1,008	none	none	none	none	22,549	118.00
2,016	none	none	none	none	22,549	72.91
4,032	none	none	none	none	22,549	48.88
504	none	Jacobi	none	none	19,592	227.33
1,008	none	Jacobi	none	none	19,592	102.08
2,016	none	Jacobi	none	none	19,592	60.87
4,032	none	Jacobi	none	none	19,592	39.31
504	db2	none	13	21	4,363	104.87
1,008	db2	none	13	42	3,182	35.82
2,016	db2	none	13	84	2,574	18.30
4,032	db2	none	13	168	2,187	13.03
504	db2	Jacobi	13	21	4,055	98.75
1,008	db2	Jacobi	13	42	2,847	33.09
2,016	db2	Jacobi	13	84	2,282	16.38
4,032	db2	Jacobi	13	168	1,936	11.75
504	mg	none	5	3	203	7.89
1,008	mg	none	5	12	203	5.40
2,016	mg	none	5	28	203	4.55
4,032	mg	none	5	42	203	4.43

Table 9: Parallel scalability for 80,764,928 DOFs Laplace benchmark

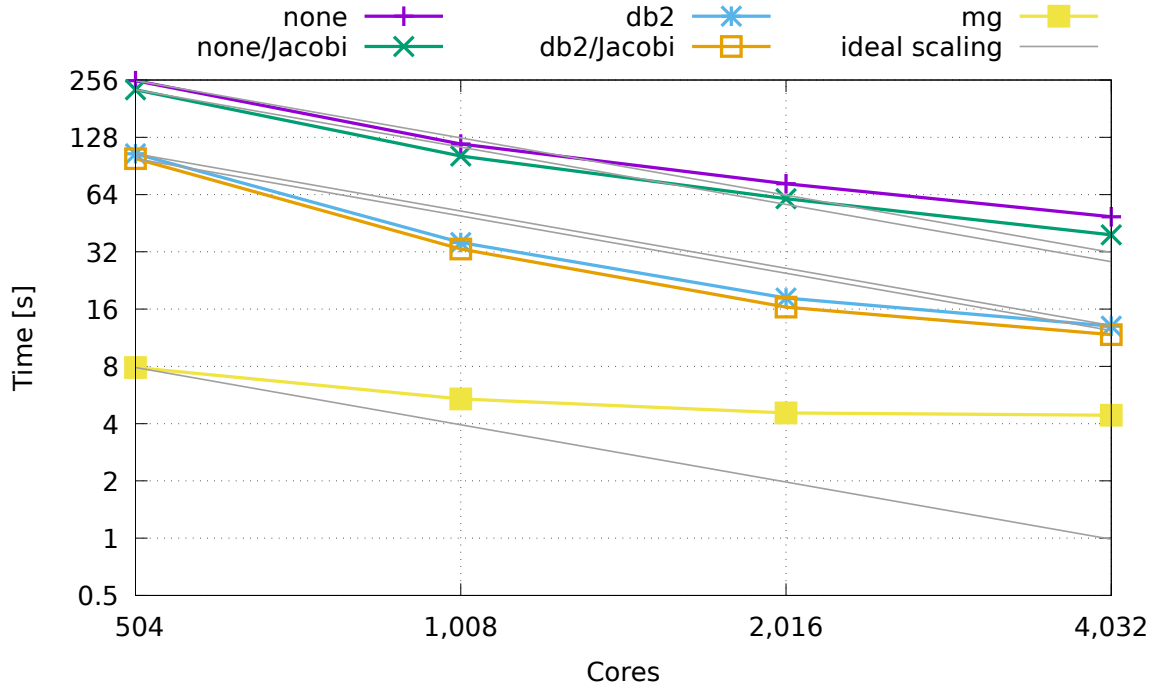


Figure 13: Parallel scalability for 80,764,928 DOFs Laplace benchmark

Cores	Space	Deflation lvl.	Redundancy	Iterations	Time
300	none	none	none	15,165	883.17
600	none	none	none	15,165	384.13
1,200	none	none	none	15,165	195.57
2,400	none	none	none	15,165	111.86
300	mg	3	1	581	103.04
600	mg	3	2	581	61.77
1,200	mg	3	5	581	42.52
2,400	mg	4	80	1,107	34.58

Table 10: Parallel scalability for 20,197,376 DOFs linear elasticity benchmark

about three times slower than CG. Again, we believe that this is because the wavelet-based deflation space does not capture the rigid body modes rotations.

The multigrid-based deflation performed reasonably well but not as good as in the case of the Laplace benchmark. The achieved speed-up was between 3 and 9.

The scalability is similar to the Laplace benchmark. The CG method achieves ideal scaling while, due to the CP solution, the mg deflation scalability is relatively poor.

Note that since a substantial cost of DCG lies in the CP matrix factorization, the scaling would significantly improve if our problem involved solving for a number of right-hand sides with the same system matrix.

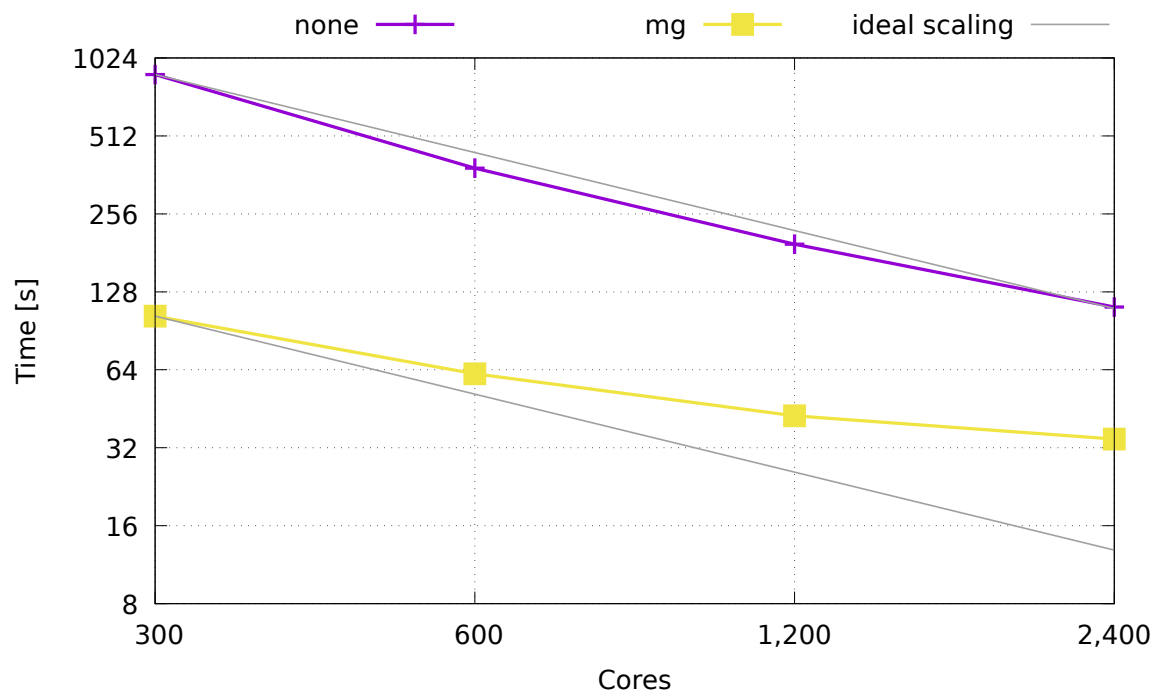


Figure 14: Parallel scalability for 51,171,075 DOFs linear elasticity benchmark



## 8 Conclusion

This thesis dealt with implementation of the deflated conjugate gradient method and its various modifications. An efficient parallel implementation was created on top of the PETSc framework for scientific computing and is now publicly available as part of the open-source PERMON toolbox for quadratic programming.

The thesis also carefully reviews the theoretical aspects of the methods. Some of the theory is described in the first sections. These sections aimed to provide an insightful view of how and why the considered methods work.

As far as I know, the thesis contains several new ideas and observations. These include improvements to the CP correction introduced in Section 3.2.1, and the introduction of the nested DCG and the possibility to combine this with CP corrections and adaptive precision.

I also extended the wavelet-based deflation (introduced by me and my advisor in [26]) to be able to handle matrices with dimensions not divisible by  $2^m$  and my experiments included not only Haar but also several other scaling filters.

While thinking about the wavelet-based deflation, it occurred to me that the approach is similar to the algebraic multigrid. The similarity of DCG to two-grid was known for a while, but there does not appear to be any numerical experiments. This is probably because the CP solution would be too expensive, but our exposure to the wavelet-based deflation presents an idea to multiply the prolongation matrices to obtain a prolongation from the coarsest grid to the finest grid.

Both the quality of the implementation and some of the theoretical considerations have been backed up by the numerical experiments carried out on a large number of benchmarks.

A thorough investigation of choice of the deflation space was carried out from which can be deduced a general order of precedence for the various deflation spaces as multigrid prolongation, wavelet-based, eigenvectors and METIS subdomain aggregation. This assumes that eigenvectors have to be computed and, as in our experiments, this computation is costly.

The adaptive precision nested DCG with the CP corrections showed an exciting potential and led to a suggestion to cheaply check whether the projection (or more precisely CP solution) in the computation of the next search direction has to be carried out.

The parallel scalability experiments up to 4,032 cores were also carried out. While, due to the costs of the CP solution, the scalability is not great, the results proved that DCG is able to achieve large speed-ups (up to 32) over CG even on reasonably large benchmarks.

In future work, I would like to investigate the hinted potential of the adaptive precision nested DCG with the CP corrections. Also, a comparison of DCG with multigrid preconditioned CG could be very interesting.

Overall, I very much enjoyed my investigation of the CG and DCG methods, and I would like to keep doing research in the area of the Krylov subspace methods.

Jakub Kružík



## References

- [1] B. A. Cipra, “The best of the 20th century: Editors name top 10 algorithms”, *SIAM news*, vol. 33, no. 4, pp. 1–2,
- [2] J. Dongarra and F. Sullivan, “Guest editors introduction to the top 10 algorithms”, *Computing in Science Engineering*, vol. 2, no. 1, pp. 22–23, Jan. 2000, ISSN: 1521-9615. DOI: [10.1109/MCISE.2000.814652](https://doi.org/10.1109/MCISE.2000.814652).
- [3] M. R. Hestenes and E. Stiefel, “Methods of conjugate gradients for solving linear systems”, *Journal of research of the National Bureau of Standards*, vol. 49, pp. 409–436, 1952.
- [4] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.
- [5] ———, “Solution of systems of linear equations by minimized iterations”, *J. Res. Nat. Bur. Standards*, vol. 49, no. 1, pp. 33–53, 1952.
- [6] G. H. Golub and C. F. van Loan, *Matrix Computations*, Fourth. JHU Press, 2013, ISBN: 1421407949 9781421407944.
- [7] J. E. Roman, C. Campos, E. Romero and A. Tomas, “SLEPc users manual”, D. Sistemes Informàtics i Computació, Universitat Politècnica de València, Tech. Rep. DSIC-II/24/02 - Revision 3.9, 2018.
- [8] A. van der Sluis and H. A. van der Vorst, “The rate of convergence of conjugate gradients”, *Numerische Mathematik*, vol. 48, no. 5, pp. 543–560, Sep. 1986, ISSN: 0945-3245. DOI: [10.1007/BF01389450](https://doi.org/10.1007/BF01389450).
- [9] A. Greenbaum, *Iterative Methods for Solving Linear Systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997, ISBN: 0-89871-396-X.
- [10] J. R. Shewchuk, “An introduction to the conjugate gradient method without the agonizing pain”, Pittsburgh, PA, USA, Tech. Rep., 1994.
- [11] Y. Saad, *Iterative methods for sparse linear systems*, Second. SIAM, 2003, ISBN: 0898715342.
- [12] R. A. Nicolaides, “Deflation of conjugate gradients with applications to boundary value problems”, *SIAM Journal on Numerical Analysis*, vol. 24, no. 2, pp. 355–365, 1987. DOI: [10.1137/0724027](https://doi.org/10.1137/0724027).
- [13] G. I. Marchuk and Y. A. Kuznetsov, “Theory and applications of the generalized conjugate gradient method”, *Advances in Mathematics. Supplementary Studies*, vol. 10, pp. 153–167, 1986.
- [14] Z. Dostal, “Conjugate gradient method with preconditioning by projector”, *International Journal of Computer Mathematics*, vol. 23, no. 3-4, pp. 315–323, 1988. DOI: [10.1080/00207168808803625](https://doi.org/10.1080/00207168808803625).

- [15] J. Erhel and F. Guyomarc'h, "An augmented conjugate gradient method for solving consecutive symmetric positive definite linear systems", *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1279–1299, 2000.
- [16] Y. Saad, M. Yeung, J. Erhel and F. Guyomarc'h, "A deflated version of the conjugate gradient algorithm", *SIAM Journal on Scientific Computing*, vol. 21, no. 5, pp. 1909–1926, 2000. DOI: [10.1137/S1064829598339761](https://doi.org/10.1137/S1064829598339761).
- [17] J. M. Tang, R. Nabben, C. Vuik and Y. A. Erlangga, "Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods", *Journal of scientific computing*, vol. 39, no. 3, pp. 340–370, 2009.
- [18] J. Kruzik, "Parallelizations of TFETI-1 coarse problem", Bachelor's Thesis, VSB - Technical University of Ostrava, 2016.
- [19] A. Vašatová, J. Tomčala, R. Sojka, M. Pecha, J. Kružík, D. Horák, V. Hapla and M. Čermák, "Parallel strategies for solving the FETI coarse problem in the PERMON toolbox", *Programs and Algorithms of Numerical Mathematics*, pp. 154–163, 2017.
- [20] V. Hapla, D. Horak and M. Merta, "Use of direct solvers in tfeti massively parallel implementation", in *Applied Parallel and Scientific Computing: 11th International Conference, PARA 2012, Helsinki, Finland, June 10-13, 2012, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 192–205, ISBN: 978-3-642-36803-5. DOI: [10.1007/978-3-642-36803-5\\_14](https://doi.org/10.1007/978-3-642-36803-5_14).
- [21] V. Hapla and D. Horak, "TFETI coarse space projectors parallelization strategies", in *Parallel Processing and Applied Mathematics - 9th International Conference, PPAM 2011, Torun, Poland, September 11-14, 2011. Revised Selected Papers, Part I*, 2011, pp. 152–162. DOI: [10.1007/978-3-642-31464-3\\_16](https://doi.org/10.1007/978-3-642-31464-3_16).
- [22] R. Nabben and C. Vuik, "A comparison of deflation and the balancing preconditioner", *SIAM Journal on Scientific Computing*, vol. 27, no. 5, pp. 1742–1759, 2006.
- [23] K. Kahl and H. Rittich, "The Deflated Conjugate Gradient Method: Convergence, Perturbation and Accuracy", *ArXiv e-prints*, Sep. 2012. arXiv: [1209.1963](https://arxiv.org/abs/1209.1963) [math.NA].
- [24] V. Simoncini and D. B. Szyld, "Theory of inexact krylov subspace methods and applications to scientific computing", *SIAM Journal on Scientific Computing*, vol. 25, no. 2, pp. 454–477, 2003.
- [25] J. Van Den Eshof and G. L. Sleijpen, "Inexact krylov subspace methods for linear systems", *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 1, pp. 125–153, 2004.
- [26] J. Kruzik and D. Horak, "Wavelet based deflation of conjugate gradient method", in *Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, Civil-Comp Press, Stirlingshire, UK, 2017. DOI: [10.4203/ccp.111.9](https://doi.org/10.4203/ccp.111.9).



- 
- [27] L. Y. Kolotilina, “Preconditioning of systems of linear algebraic equations by means of twofold deflation. i”, *Theory. J. Math. Sci.*, vol. 89, pp. 1652–1689, 1998.
  - [28] A. V. Knyazev, “Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method”, *SIAM journal on scientific computing*, vol. 23, no. 2, pp. 517–541, 2001.
  - [29] A. V. Knyazev, M. E. Argentati, I. Lashuk and E. E. Ovtchinnikov, “Block locally optimal preconditioned eigenvalue solvers (blopex) in hypre and petsc”, *SIAM Journal on Scientific Computing*, vol. 29, no. 5, pp. 2224–2239, 2007.
  - [30] E. Romero and J. E. Roman, “A parallel implementation of Davidson methods for large-scale eigenvalue problems in SLEPc”, *ACM Trans. Math. Software*, vol. 40, no. 2, 13:1–13:29, 2014.
  - [31] A. Stathopoulos and K. Orginos, “Computing and deflating eigenvalues while solving multiple right-hand side linear systems with an application to quantum chromodynamics”, *SIAM Journal on Scientific Computing*, vol. 32, no. 1, pp. 439–462, 2010. DOI: [10.1137/080725532](https://doi.org/10.1137/080725532).
  - [32] R. Blaheta, “Multilevel iterative methods and deflation”, in *ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics*, Delft University of Technology, 2006.
  - [33] S. P. MacLachlan, J. M. Tang and C. Vuik, “Fast and robust solvers for pressure-correction in bubbly flow problems”, *Journal of Computational Physics*, vol. 227, no. 23, pp. 9742–9761, 2008.
  - [34] J. van der Linden, T. Jönsthövel, A. Lukyanov and C. Vuik, “The parallel subdomain-levelset deflation method in reservoir simulation”, *Journal of Computational Physics*, vol. 304, pp. 340–358, 2016, ISSN: 0021-9991. DOI: [10.1016/j.jcp.2015.10.016](https://doi.org/10.1016/j.jcp.2015.10.016).
  - [35] G. Strang and T. Nguyen, *Wavelets and Filter Banks*, 2nd. Wellesley College, 1996, ISBN: 9780961408879, 0961408871.
  - [36] G. Bachmann, L. Narici and E. Beckenstein, *Fourier and Wavelet Analysis*, ser. Universitext. Springer New York, 2002, ISBN: 9780387988993.
  - [37] D. Walnut, *An Introduction to Wavelet Analysis*, ser. Applied and Numerical Harmonic Analysis. Birkhäuser Boston, 2002, ISBN: 9780817639624.
  - [38] D. Hooker and A. Sawchuk. (1973). Test image "lenna", [Online]. Available: [https://en.wikipedia.org/wiki/File:Lenna\\_\(test\\_image\).png](https://en.wikipedia.org/wiki/File:Lenna_(test_image).png) (visited on 10/05/2018).
  - [39] Lourakis. (2016). Wavelet decomposition of "lenna" image, [Online]. Available: [https://commons.wikimedia.org/wiki/File:Lenna\\_Haar\\_Decomposition\\_2\\_iterations.png](https://commons.wikimedia.org/wiki/File:Lenna_Haar_Decomposition_2_iterations.png) (visited on 10/05/2018).

- 
- [40] C. Taswell and K. C. McGill, “Algorithm 735: Wavelet transform algorithms for finite-duration discrete-time signals”, *ACM Trans. Math. Softw.*, vol. 20, no. 3, pp. 398–412, Sep. 1994, ISSN: 0098-3500. DOI: [10.1145/192115.192156](https://doi.org/10.1145/192115.192156).
  - [41] W. Hackbusch, *Multi-Grid Methods and Applications*. Jan. 1985, vol. 4, ISBN: 3-540-12761-5.
  - [42] J. M. Tang, S. P. MacLachlan, R. Nabben and C. Vuik, “A comparison of two-level preconditioners based on multigrid and deflation”, *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 4, pp. 1715–1739, 2010.
  - [43] W. L. Briggs and V. E. Henson, “Wavelets and multigrid”, *SIAM Journal on Scientific Computing*, vol. 14, no. 2, pp. 506–510, 1993.
  - [44] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, D. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. Smith, S. Zampini, H. Zhang and H. Zhang, *PETSc Web page*, 2018. [Online]. Available: <http://www.mcs.anl.gov/petsc>.
  - [45] —, “PETSc users manual”, Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.9, 2018. [Online]. Available: <http://www.mcs.anl.gov/petsc>.
  - [46] S. Balay, W. D. Gropp, L. C. McInnes and B. F. Smith, “Efficient management of parallelism in object oriented numerical software libraries”, in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset and H. P. Langtangen, Eds., Birkhäuser Press, 1997, pp. 163–202.
  - [47] PERMON web page, [Online]. Available: <http://permon.vsb.cz/> (visited on 16/05/2018).
  - [48] X. S. Li, “An overview of SuperLU: Algorithms, implementation, and user interface”, vol. 31, no. 3, pp. 302–325, Sep. 2005.
  - [49] X. S. Li and J. W. Demmel, “SuperLU\_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems”, *ACM Trans. Mathematical Software*, vol. 29, no. 2, pp. 110–140, Jun. 2003.
  - [50] V. Hernandez, J. E. Roman and V. Vidal, “SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems”, *ACM Trans. Math. Software*, vol. 31, no. 3, pp. 351–362, 2005.
  - [51] *MFEM: Modular finite element methods*, [mfem.org](http://mfem.org).
  - [52] R. D. Falgout and U. M. Yang, “Hypre: A library of high performance preconditioners”, in *International Conference on Computational Science*, Springer, 2002, pp. 632–641.
  - [53] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs”, *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.
  - [54] ARCHER web page, [Online]. Available: <http://www.archer.ac.uk/> (visited on 09/05/2018).

- 
- [55] TOP500 November 2017 list, [Online]. Available: <https://www.top500.org/list/2017/11/> (visited on 09/05/2018).
  - [56] J. Kruzik. PERMON DCG implementation, [Online]. Available: <https://github.com/permon/permon/tree/jakub/feature-dcg> (visited on 16/05/2018).
  - [57] G. Liu, D. Luo, G. Lv, Y. Guo and M. Lee, “Arbitrary-length jacket-haar transforms”, in *International Conference on Algorithms and Architectures for Parallel Processing*, Springer, 2015, pp. 330–343.
  - [58] J. Kruzik. PERMON DCG examples, [Online]. Available: <https://github.com/jkruzik/PermonDCGexamples> (visited on 16/05/2018).
  - [59] T. A. Davis and Y. Hu, “The university of florida sparse matrix collection”, *ACM Trans. Math. Softw.*, vol. 38, no. 1, 1:1–1:25, Dec. 2011, ISSN: 0098-3500. DOI: [10.1145/2049662.2049663](https://doi.org/10.1145/2049662.2049663).
  - [60] T. Davis and Y. Hu. SuiteSparse Matrix Collection web page, [Online]. Available: <https://sparse.tamu.edu/> (visited on 11/05/2018).
  - [61] D. Lukas. Dalibor Lukas homepage, [Online]. Available: <https://homel.vsb.cz/~luk76/> (visited on 14/05/2018).
  - [62] PETSc FAQ, [Online]. Available: <http://www.mcs.anl.gov/petsc/documentation/faq.html> (visited on 17/05/2018).



## A Results for Matrices from SuiteSparse Matrix Collection

Matrices diverging on DTOL (id:name): 74:bcsstm24 , 228:plat1919 , 356:ct20stif , 358:msc01050 , 411:ex13 , 413:ex15 , 430:ex33 , 791:aft01 , 1253:bmw7st\_1, 1278:ship\_003 , 1425:bloweybq , 1437:LF10000

Matrices diverging on ITS (id:name): 40:bcsstk18 , 45:bcsstk23 , 46:bcsstk24 , 47:bcsstk25 , 341:bcsstk36 , 343:bcsstk38 , 362:msc23052 , 369:pwtk , 440:ex9 , 817:raefsky4 , 1269:m\_t1 , 1275:s3dkq4m2 , 1276:s3dkt3m2 , 1277:ship\_001 , 1279:shipsec1 , 1280:shipsec5 , 1281:shipsec8 , 1283:thread , 1287:vanbody , 1290:x104 , 1439:LFAT5000 , 1621:mhd1280b , 1623:mhd3200b , 1625:mhd4800b , 1644:msdoor , 1892:denormal , 2283:offshore , 2541:Serena , 2542:Emilia\_923 , 2543:Fault\_639 , 2545:Geo\_1438 , 2547:StocF-1465 , 2659:Bump\_2911 , 2660:Queen\_4147 , 2661:PFlow\_742

id	name	rows	cond	none	eig5	eig10	eig20	eig40	db2	db4	db8	db16	biorth	meyer
1	1138_bus	1,138	8.6E+06	2,130	1,171	1,038	699	523	416	314	320	295	311	279
2	494_bus	494	2.4E+06	1,171	710	550	377	233	278	217	225	252	222	210
3	662_bus	662	7.9E+05	531	287	210	159	128	211	199	198	207	202	187
4	685_bus	685	4.2E+05	519	255	187	133	99	129	114	128	122	116	109
23	bcsstk01	48	8.8E+05	137	87	50	26	7	59	62	55	69	66	62
24	bcsstk02	66	4.3E+03	44	33	21	17	12	37	38	39	38	38	39
25	bcsstk03	112	6.8E+06	578	414	306	178	64	279	198	269	279	211	200
26	bcsstk04	132	2.3E+06	535	302	173	124	90	238	204	174	217	250	222
27	bcsstk05	153	1.4E+04	262	129	105	69	37	98	106	101	102	109	104
28	bcsstk06	420	7.6E+06	3,936	3,011	2,197	1,028	417	1,062	906	960	1,226	884	1,051
29	bcsstk07	420	7.6E+06	3,936	3,011	2,197	1,028	417	1,057	904	960	1,226	884	1,052
30	bcsstk08	1,074	2.6E+07	6,400	5,734	4,819	3,356	2,740	467	369	362	431	375	270
31	bcsstk09	1,083	9.5E+03	194	138	101	74	53	102	115	114	128	113	122
32	bcsstk10	1,086	5.2E+05	3,914	3,013	1,865	883	617	1,257	925	982	983	849	1,051
33	bcsstk11	1,473	2.2E+08	24,899	14,252	10,001	5,819	3,018	13,654	8,680	8,427	7,551	9,618	7,687
34	bcsstk12	1,473	2.2E+08	24,899	14,252	10,001	5,819	3,018	13,680	8,680	8,415	7,551	9,618	7,692
35	bcsstk13	2,003	NA	its	its	its	its	its	its	its	27,669	22,501	its	25,753

id	name	rows	cond	none	eig5	eig10	eig20	eig40	db2	db4	db8	db16	biorth	meyer
36	bcsstk14	1,806	1.2E+10	13,768	13,782	13,827	13,817	8,974	6,563	9,046	8,263	its	8,477	8,090
37	bcsstk15	3,948	6.5E+09	21,776	21,781	EDC	EDC	EDC	3,791	7,044	7,154	7,386	5,824	5,872
38	bcsstk16	4,884	4.9E+09	463	EDC	EDC	EDC	EDC	235	511	1,080	1,190	581	801
39	bcsstk17	10,974	1.3E+10	21,810	21,775	21,829	21,875	21,989	its	its	its	its	its	its
41	bcsstk19	817	NA	its	EDC	EDC	EDC	EDC	8,325	8,768	9,016	11,560	9,154	9,594
42	bcsstk20	485	NA	dtol	EDC	EDC	EDC	EDC	2,503	dtol	dtol	dtol	dtol	dtol
43	bcsstk21	3,600	1.8E+07	10,619	6,060	4,389	3,103	1,899	4,861	2,844	1,781	1,706	5,928	4,605
44	bcsstk22	138	1.1E+05	336	191	157	114	74	132	124	136	130	129	126
48	bcsstk26	1,922	1.7E+08	26,958	17,080	14,815	11,856	8,983	3,721	3,951	2,587	3,214	3,648	2,859
49	bcsstk27	1,224	2.2E+04	909	895	863	860	817	434	359	381	306	411	331
50	bcsstk28	4,410	9.5E+08	13,777	9,095	5,406	2,634	1,477	4,238	3,515	3,910	3,689	3,383	3,743
57	bcsstm02	66	8.8E+00	12	12	11	9	6	7	17	15	15	16	16
60	bcsstm05	153	1.3E+01	17	18	18	15	13	12	17	19	19	18	19
61	bcsstm06	420	3.5E+06	119	121	121	83	73	101	179	239	302	226	307
62	bcsstm07	420	7.4E+03	278	279	254	252	212	166	126	121	137	113	142
63	bcsstm08	1,074	8.3E+06	160	124	123	101	96	57	171	159	170	205	174
64	bcsstm09	1,083	1.0E+04	2	NA	NA	NA	NA	2	4	12	13	7	7
66	bcsstm11	1,473	1.2E+05	25	26	27	22	22	25	55	57	59	56	72
67	bcsstm12	1,473	6.3E+05	2,787	2,788	2,597	2,290	2,076	1,539	959	989	873	1,138	955
69	bcsstm19	817	2.3E+05	473	469	438	427	425	23	39	33	41	45	35
70	bcsstm20	485	2.6E+05	281	263	265	264	264	41	48	47	49	49	41
71	bcsstm21	3,600	2.4E+01	3	4	4	4	4	3	6	13	14	12	19
72	bcsstm22	138	9.4E+02	50	36	29	27	27	34	52	49	45	50	48
73	bcsstm23	3,134	9.5E+08	5,608	2,594	2,137	1,812	1,567	2,991	4,475	4,425	4,749	5,423	5,059
75	bcsstm25	15,439	NA	its	its	its	its	its	23,754	its	23,650	25,624	its	18,536

id	name	rows	cond	none	eig5	eig10	eig20	eig40	db2	db4	db8	db16	biorth	meyer
76	bcsstm26	1,922	2.6E+05	2,103	1,842	1,487	1,323	1,016	226	726	886	837	689	761
159	gr_30_30	900	1.9E+02	34	29	25	20	15	8	11	10	11	11	10
206	lund_a	147	2.8E+06	342	182	121	75	45	193	208	205	250	212	222
207	lund_b	147	3.0E+04	393	243	157	110	62	130	134	134	147	131	127
217	nos1	237	2.0E+07	2,106	1,281	1,013	675	339	199	206	253	313	214	218
218	nos2	957	NA	its	20,160	15,073	7,258	3,716	884	901	1,232	1,555	946	1,067
219	nos3	960	3.8E+04	230	112	82	53	36	118	102	114	117	100	111
220	nos4	100	1.6E+03	76	41	29	18	12	42	41	44	45	41	44
221	nos5	468	1.1E+04	428	265	177	116	58	245	298	241	269	279	256
222	nos6	675	7.6E+06	996	988	1,008	982	997	353	1,028	1,738	1,682	960	1,734
223	nos7	729	2.4E+09	3,248	2,051	1,811	1,491	EDC	dtol	1,985	1,583	1,900	1,596	dtol
229	plat362	362	2.2E+11	6,729	EDC	EDC	EDC	EDC	its	its	its	its	its	its
315	mhdb416	416	4.0E+09	4,876	2,909	2,168	1,568	1,033	2,252	2,680	3,003	3,066	2,649	2,857
339	bcsstk34	588	2.8E+04	791	631	542	511	418	117	79	116	132	72	116
349	bcsstm39	46,772	8.3E+03	307	309	308	308	308	221	411	410	406	416	408
353	crystm01	4,875	2.3E+02	70	71	63	63	60	55	62	68	58	58	76
354	crystm02	13,965	2.5E+02	79	80	80	80	68	56	64	71	78	60	80
355	crystm03	24,696	2.6E+02	78	79	79	79	70	61	70	76	80	66	80
357	msc00726	726	4.2E+05	1,018	887	820	745	657	663	165	281	229	156	207
359	msc01440	1,440	1.4E+06	5,719	5,680	5,251	3,401	1,498	2,037	2,141	1,976	2,063	2,262	2,015
360	msc04515	4,515	2.3E+06	4,410	2,574	1,913	1,057	760	666	692	710	684	812	703
361	msc10848	10,848	NA	its	29,325	17,981	8,483	4,535	9,509	7,228	5,635	7,488	6,316	5,762
407	ex10	2,410	NA	its	EDC	EDC	EDC	EDC	272	320	267	938	315	dtol
408	ex10hs	2,548	NA	dtol	EDC	EDC	EDC	EDC	279	319	385	635	452	919
427	ex3	1,821	1.7E+10	10,073	6,511	5,111	4,052	2,927	683	420	608	415	641	402

id	name	rows	cond	none	eig5	eig10	eig20	eig40	db2	db4	db8	db16	biorth	meyer
436	ex5	27	6.6E+07	53	29	17	1	NA	29	28	28	21	30	29
752	finan512	74,752	2.9E+01	36	35	35	35	34	29	21	26	23	22	23
757	nasa1824	1,824	1.4E+06	2,544	1,967	1,630	1,175	967	1,006	834	632	678	753	664
758	nasa2146	2,146	NA	0	1	2	3	4	5	6	7	8	9	10
759	nasa2910	2,910	6.0E+06	3,453	2,713	1,953	1,648	1,257	1,358	1,216	905	976	1,092	987
760	nasa4704	4,704	3.1E+07	9,858	7,626	6,069	4,212	3,184	3,636	2,487	1,808	1,932	2,276	1,901
761	nasasrb	54,870	4.8E+07	15,080	9,037	9,014	9,016	6,093	8,839	6,104	12,427	9,500	9,330	9,975
804	cfd1	70,656	3.4E+05	1,622	888	636	569	454	786	822	837	832	817	837
805	cfd2	123,440	1.5E+06	5,935	4,031	2,929	2,185	1,705	5,874	5,152	5,287	4,979	5,387	5,191
813	olafu	16,146	NA	its	its	its	its	its	its	28,393	27,087	26,792	23,976	25,329
845	qa8fm	66,127	7.0E+01	50	51	51	48	47	38	38	38	38	38	37
868	bodyy4	17,546	7.3E+02	157	157	158	158	158	122	119	123	122	119	125
869	bodyy5	18,589	7.4E+03	492	493	493	495	492	365	378	379	388	372	393
870	bodyy6	19,366	7.3E+04	1,519	1,516	1,520	1,518	1,518	1,234	1,210	1,215	1,211	1,196	1,208
872	mesh1e1	48	5.2E+00	15	13	12	10	6	10	11	10	11	11	11
873	mesh1em1	48	1.9E+01	27	24	21	16	7	16	18	17	18	18	16
874	mesh1em6	48	5.9E+00	15	14	13	11	7	11	11	11	12	11	11
875	mesh2e1	306	2.8E+02	76	77	77	74	68	39	44	48	48	40	49
876	mesh2em5	306	2.4E+02	63	64	64	62	59	31	33	36	37	32	36
877	mesh3e1	289	8.8E+00	18	17	16	15	13	13	14	14	14	13	14
878	mesh3em5	289	5.0E+00	12	13	13	13	12	12	13	13	13	13	13
887	fv1	9,604	8.7E+00	19	20	20	19	18	8	8	8	8	8	7
888	fv2	9,801	8.7E+00	19	20	20	19	18	7	7	7	8	7	7
889	fv3	9,801	2.0E+03	112	94	82	63	49	11	10	10	10	10	9
924	Andrews	60,000	3.3E+16	135	EDC	EDC	EDC	EDC	dtol	dtol	dtol	dtol	dtol	dtol



id	name	rows	cond	none	eig5	eig10	eig20	eig40	db2	db4	db8	db16	biorth	meyer
936	nd3k	9,000	1.6E+07	6,018	2,281	1,094	748	515	4,472	4,813	5,051	5,236	4,524	5,144
937	nd6k	18,000	1.6E+07	6,577	2,732	1,452	1,019	757	5,647	6,109	6,327	6,589	5,833	6,643
938	nd12k	36,000	1.3E+07	7,612	2,914	1,665	1,290	978	5,603	5,972	6,339	6,458	5,660	6,533
939	nd24k	72,000	1.3E+07	8,335	3,072	2,041	1,439	1,066	6,057	6,352	6,802	6,961	5,987	7,007
942	af_shell3	504,855	3.5E+02	88	90	90	89	90	1,122	1,091	1,186	1,176	1,096	1,170
943	af_shell4	504,855	3.5E+02	82	84	84	84	84	1,120	1,090	1,186	1,176	1,097	1,173
946	af_shell7	504,855	3.4E+02	85	86	86	86	86	1,193	1,153	1,256	1,254	1,163	1,264
947	af_shell8	504,855	3.5E+02	80	81	81	81	81	1,197	1,156	1,262	1,261	1,167	1,278
1184	Pres_Poisson	14,822	2.0E+06	1,906	1,172	1,169	1,053	965	529	385	409	371	396	367
1202	gyro_k	17,361	1.1E+09	28,927	17,218	12,835	4,794	1,680	14,114	14,107	14,460	14,816	13,621	15,197
1203	gyro_m	17,361	2.5E+06	7,083	6,198	6,199	5,883	5,395	5,357	5,202	5,325	5,456	4,850	5,510
1205	t2dah_e	11,445	NA	its	its	its	26,423	17,341	its	its	its	its	its	its
1207	t2dal_e	4,257	3.8E+07	17,632	15,988	10,203	7,039	4,945	10,201	15,469	18,386	19,161	17,422	22,194
1211	t3dl_e	20,360	5.9E+03	262	263	256	256	256	342	522	512	515	533	524
1214	sts4098	4,098	2.1E+08	29,588	29,569	27,340	23,259	20,598	4,404	3,949	4,014	3,892	4,228	3,724
1252	audikw_1	943,695	NA	its	its	28,019	13,365	7,792	17,261	14,049	DFS	DFS	DFS	DFS
1254	bmwcra_1	148,770	3.8E+08	10,000	6,936	4,206	1,780	1,119	5,171	5,683	5,813	5,983	5,431	5,968
1257	crankseg_1	52,804	4.9E+07	3,262	1,907	1,402	1,179	1,021	1,041	1,040	1,163	1,156	982	DSF
1258	crankseg_2	63,838	4.8E+07	4,295	2,524	2,005	1,641	1,409	1,196	1,228	1,334	1,339	1,143	1,394
1266	hood	220,542	2.3E+08	24,903	22,640	22,668	19,161	16,954	7,557	27,871	26,273	its	29,187	its
1267	inline_1	503,712	NA	its	its	28,991	14,256	8,350	its	its	its	its	its	its
1268	ldoor	952,203	1.7E+08	25,533	22,520	19,809	17,602	15,574	11,754	13,871	DSF	DSF	DSF	DSF
1270	oilpan	73,752	NA	its	22,128	15,492	11,437	8,451	15,458	17,100	22,415	24,289	15,118	22,235
1288	wathen100	30,401	5.8E+03	256	145	129	113	93	98	79	105	101	76	97
1289	wathen120	36,441	2.6E+03	286	169	131	111	95	112	95	131	120	83	109

id	name	rows	cond	none	eig5	eig10	eig20	eig40	db2	db4	db8	db16	biorth	meyer
1310	cvxbqp1	50,000	2.2E+06	7,527	EDC	EDC	EDC	EDC	3,203	3,656	3,137	3,186	3,816	3,406
1311	gridgena	48,962	1.0E+00	1	2	2	1	1	1	1,739	1,797	1,958	1,723	1,714
1312	jnlbrng1	40,000	1.8E+02	85	84	83	81	77	32	41	34	39	40	37
1313	minsurfo	40,806	7.7E+01	49	52	52	52	51	19	28	22	25	28	30
1314	obstclae	40,000	1.0E+00	1	2	2	1	1	1	18	23	24	19	22
1315	torsion1	40,000	1.0E+00	1	2	2	1	1	1	18	23	24	19	22
1330	Kuu	7,102	1.6E+04	480	366	263	207	142	305	323	366	368	322	325
1331	Muu	7,102	7.6E+01	50	51	46	44	41	1	40	40	37	43	38
1347	bundle1	10,581	1.0E+03	155	156	156	155	155	87	56	37	40	51	36
1401	Chem97ZtZ	2,541	2.5E+02	87	88	88	88	88	45	47	45	45	47	45
1402	thermal1	82,654	3.2E+05	1,099	563	436	295	223	134	125	124	123	129	DSF
1403	thermal2	1,228,045	4.9E+06	3,627	1,746	1,534	1,035	751	598	DSF	DSF	DSF	590	DSF
1406	ted_B	10,605	1.9E+07	845	750	771	750	699	397	713	865	808	716	890
1409	ted_B_unscaled	10,605	1.3E+11	978	EDC	EDC	EDC	EDC	25	dtol	dtol	dtol	642	dtol
1412	G2_circuit	150,102	1.3E+07	8,918	5,661	4,093	2,956	2,324	900	2,186	1,835	2,415	2,256	DSF
1421	G3_circuit	1,585,478	1.5E+07	12,685	11,119	10,366	8,068	1,307	1,307	2,496	2,092	DSF	2,403	DSF
1422	apache1	80,800	3.0E+06	1,261	758	457	370	232	464	741	761	758	760	673
1423	apache2	715,176	3.1E+06	3,973	1,955	1,385	908	625	472	425	417	416	442	399
1435	gyro	17,361	1.1E+09	28,927	17,218	12,835	4,794	1,680	14,174	14,103	14,823	14,805	13,604	14,754
1438	LF10	18	3.9E+06	39	EDC	EDC	NA	NA	10	10	13	17	10	10
1440	LFAT5	14	1.4E+08	25	EDC	EDC	EDC	EDC	8	8	11	13	8	8
1453	bone010	986,703	4.7E+08	16,416	9,401	5,825	2,066	951	12,761	11,375	12,943	13,727	10,860	14,002
1454	boneS01	127,224	1.7E+07	2,355	1,024	697	534	406	1,329	1,358	1,453	1,498	1,282	1,536
1455	boneS10	914,898	NA	its	18,509	10,369	6,801	3,838	20,722	20,739	22,462	23,032	19,718	DSF
1506	Journals	124	9.8E+03	127	90	80	67	46	59	58	59	60	59	57

id	name	rows	cond	none	eig5	eig10	eig20	eig40	db2	db4	db8	db16	biorth	meyer
1580	af_0_k101	503,625	1.4E+08	28,753	20,496	13,578	7,218	4,294	16,672	15,689	15,914	15,222	16,348	15,800
1581	af_1_k101	503,625	1.4E+08	29,716	17,308	12,976	7,517	4,111	16,820	16,265	16,993	16,144	17,147	16,235
1582	af_2_k101	503,625	1.0E+06	1,661	1,628	1,616	1,611	1,406	1,549	1,505	1,037	864	1,549	1,516
1583	af_3_k101	503,625	1.1E+08	25,545	12,768	6,510	5,237	3,504	14,580	14,124	14,372	13,639	14,847	13,729
1584	af_4_k101	503,625	2.3E+08	29,171	9,985	9,079	5,249	3,699	16,432	15,931	16,207	15,894	16,809	16,223
1585	af_5_k101	503,625	2.3E+08	27,599	7,996	7,586	5,159	3,534	15,562	15,130	15,452	14,527	15,917	14,837
1605	s1rmq4m1	5,489	1.8E+06	6,011	4,426	3,976	3,171	2,471	2,592	2,398	1,859	2,601	2,718	2,662
1606	s2rmq4m1	5,489	1.8E+08	20,319	11,672	8,787	7,121	5,849	3,455	2,180	2,172	2,773	3,698	2,896
1607	s3rmq4m1	5,489	NA	its	19,085	15,378	11,630	9,797	3,890	7,245	3,413	4,790	5,612	3,896
1608	s1rmt3m1	5,489	2.5E+06	6,592	4,444	3,662	2,867	2,167	2,728	3,002	2,021	2,563	3,896	3,747
1609	s2rmt3m1	5,489	2.5E+08	28,826	16,691	11,845	9,889	7,847	4,149	3,166	2,600	3,200	5,146	4,034
1610	s3rmt3m1	5,489	NA	its	EDC	EDC	EDC	EDC	4,805	9,330	4,334	7,271	7,288	5,268
1611	s3rmt3m3	5,357	NA	its	its	its	its	28,828	its	14,571	7,884	8,334	12,664	9,833
1847	Dubcova1	16,129	1.0E+03	87	72	63	52	46	33	33	33	34	33	DSF
1848	Dubcova2	65,025	4.0E+03	157	132	114	89	69	37	37	36	36	37	36
1849	Dubcova3	146,689	4.0E+03	159	133	116	90	71	47	48	47	46	49	47
1850	BenElechi1	245,874	NA	its	20,313	11,838	5,747	3,224	22,030	22,212	20,059	20,753	21,631	17,958
1853	parabolic_fem	525,825	2.1E+05	1,487	880	646	487	387	185	127	143	118	145	127
1883	ecology2	999,999	6.5E+07	5,393	2,421	1,563	1,109	771	226	210	208	207	219	193
1899	tmt_sym	726,713	7.9E+08	3,042	810	385	238	1	27	43	44	47	47	48
1909	smt	25,710	1.6E+09	9,283	3,835	2,425	1,531	1,041	3,222	3,581	3,576	3,849	3,445	3,929
1911	plbuckle	1,282	1.3E+06	1,966	1,104	704	420	273	1,146	480	958	660	507	771
1912	cbuckle	13,681	1.7E+06	4,865	4,853	4,868	4,855	4,866	5,627	3,187	3,011	2,383	3,788	2,689
1919	2cubes_sphere	101,492	NA	19,480	21,485	EDC	EDC	EDC	24,961	its	its	its	its	its
1939	bibd_81_2	3,240	1.0E+00	1	2	2	2	2	1	2	2	2	2	2

id	name	rows	cond	none	eig5	eig10	eig20	eig40	db2	db4	db8	db16	biorth	meyer
2203	Trefethen_20b	19	3.0E+01	19	EDC	EDC	NA	NA	10	10	10	10	10	10
2204	Trefethen_20	20	6.3E+01	20	EDC	EDC	EDC	NA	11	11	11	11	11	11
2205	Trefethen_150	150	7.7E+02	95	54	37	25	16	56	57	60	67	56	57
2206	Trefethen_200b	199	5.2E+02	100	58	41	28	19	61	61	65	74	60	62
2207	Trefethen_200	200	1.1E+03	114	64	43	29	19	67	68	73	82	67	68
2208	Trefethen_300	300	1.8E+03	146	81	55	37	24	85	87	93	106	86	87
2209	Trefethen_500	500	3.2E+03	197	108	73	49	32	115	118	127	145	115	117
2210	Trefethen_700	700	4.7E+03	240	131	88	60	39	139	143	153	176	140	142
2211	Trefethen_2000	2,000	1.6E+04	435	235	157	106	69	251	259	279	320	253	257
2212	Trefethen_20000b	19,999	9.6E+04	1,338	738	508	352	236	805	816	852	1,026	779	819
2213	Trefethen_20000	20,000	2.0E+05	1,545	814	544	364	237	889	912	988	1,135	897	915
2257	thermomech_TC	102,158	6.5E+01	51	50	49	48	45	31	31	30	30	32	30
2258	thermomech_TK	102,158	3.0E+18	2,054	EDC	EDC	EDC	EDC	dtol	dtol	dtol	dtol	dtol	dtol
2259	thermomech_dM	204,316	6.5E+01	51	50	49	48	45	1	DSF	DSF	DSF	DSF	DSF
2261	shallow_water1	81,920	3.4E+00	10	11	11	11	11	8	9	9	9	8	9
2262	shallow_water2	81,920	1.0E+01	18	19	19	19	19	13	14	15	15	14	14
2373	pdb1HYS	36,417	3.5E+11	4,740	EDC	EDC	EDC	EDC	dtol	dtol	dtol	dtol	dtol	dtol
2374	consph	83,334	3.9E+06	13,112	10,974	9,641	7,958	6,383	7,046	7,441	8,566	9,176	6,929	9,895
2375	cant	62,451	2.6E+10	10,026	EDC	EDC	EDC	EDC	dtol	dtol	dtol	dtol	dtol	dtol
2544	Flan_1565	1,564,794	1.2E+08	17,399	11,901	9,027	5,638	3,506	10,884	14,163	13,854	15,997	13,082	16,602
2546	Hook_1498	1,498,023	3.6E+06	8,285	EDC	EDC	EDC	EDC	4,747	5,453	5,540	DFS	5,356	DSF
2664	bundle_adj	513,351	NA	its	EDC	EDC	EDC	EDC	1,066	33	32	73	DSF	66