Iterative and parallel methods for linear systems,

with applications in circuit simulation

Wim Bomhof

Iterative and parallel methods for linear systems, with applications in circuit simulation

Iteratieve en parallelle methoden voor lineaire stelsels, met toepassingen in circuitsimulatie

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de Rector Magnificus, Prof.dr. W.H. Gispen, ingevolge het besluit van het College van Promoties in het openbaar te verdedigen op woensdag 23 mei 2001 des ochtends te 10.30 uur

door

Christiaan Willem Bomhof

geboren op 1 juli 1969, te Vaassen

Promotor: Prof.dr. H.A. van der Vorst
Faculteit der Wiskunde en Informatica
Universiteit Utrecht

Dit onderzoek maakt deel uit van het ELSIM-project van het Platform HPCN en is een samenwerkingsverband tussen de Universiteit Utrecht en Philips Research.

Mathematics Subject Classification: 65F10, 65F05, 65F50, 65L10, 65Y05, 94C05.

Bomhof, Christiaan Willem
Iterative and parallel methods for linear systems, with applications in circuit simulation
Proefschrift Universiteit Utrecht – Met een samenvatting in het Nederlands.

ISBN 90-393-2708-4

Contents

Chapter 1

Introduction

Large sparse linear systems

$$Ax = b\,, \tag{1.1}$$

often arise in the simulation of physical and other phenomena. Solving these linear systems may be expensive in terms of CPU-time. Therefore, it is a good idea to develop efficient sequential and/or parallel methods for solving (1.1).

In this thesis we focus on numerical methods for solving three types of linear systems (1.1):

- A mixed direct/iterative parallel method for linear systems arising in transient analysis of circuits: Chapter 2 and 3.
- A parallelizable iterative method for p-cyclic linear systems, with applications in periodic steady state analysis of circuits: Chapter 4.
- An iterative method for linear systems of the form $P(B)y = c$, where $P(B)$ is a matrix polynomial in B: Chapter 5.

A large part of this thesis deals with linear systems arising in circuit simulation. However, the methods proposed in this thesis may be useful in other applications as well.

Circuits can be modelled with the differential algebraic equation (DAE)

$$\frac{d}{dt}q(x(t)) + j(x(t), t) = 0\,, \tag{1.2}$$

with $x \in \mathbb{R}^n$. The DAE character of (1.2) is reflected in the function q. For example, (1.2) is a DAE, but not an ODE, if $n = 2$ and q depends only on x_1. In Section 1.1 we consider the construction of this DAE briefly. Sections 1.2, 1.3, and 1.4, are an introduction to Chapter 2/3, Chapter 4, and Chapter 5, respectively.

In transient analysis (1.2) is solved on a certain time interval. The linear systems arising in transient analysis are derived in Section 1.2. We also give a brief overview of existing methods for solving these linear systems. In periodic steady state analysis, the function j in (1.2) is periodic in time t, and a periodic boundary constraint is added to (1.2). In Section 1.3 we consider the linear systems arising in periodic steady state analysis. Section 1.4 is an introduction to Chapter 5, where linear systems of the form

$P(B)y = c$ are considered. These linear systems are not directly related with circuit simulation, but applications arise in, for example, lattice quantum chromodynamics and Tikhonov-Phillips regularization [34], and in higher order implicit methods for initial value problems [37, Ch. 8].

1.1 Basic equations in circuit simulation

Circuits can be modelled with three basic ingredients (see for example [33], [64, Ch. 1]):

- Kirchhoff's current law (KCL): The algebraic sum of the currents into a node is zero.
- Kirchhoff's voltage law (KVL): The algebraic sum of the voltage drops around a closed loop is zero.
- Branch constitutive equations (BCE): these can be algebraic or differential equations. For example, a resistor can be modelled by Ohm's law: $i = v/R$, and a linear capacitor can be modelled by $i = \frac{d}{dt}(Cv)$.

We will illustrate this with an example.

Example We consider the circuit of Figure 1.1.

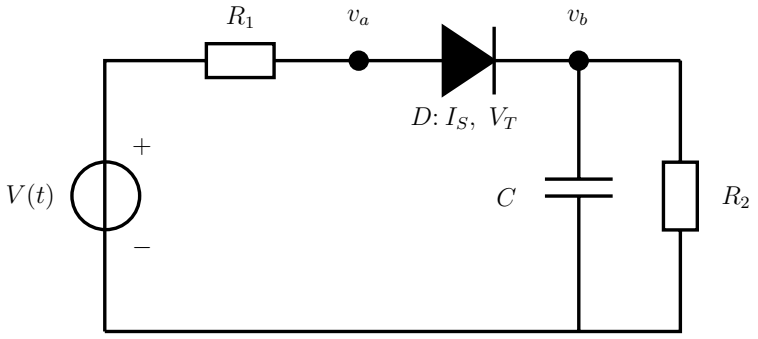


Figure 1.1: A simple circuit

This circuit can be modelled with $n = 6$ circuit unknowns: 4 branch currents I_{R_1}, I_{R_2}, I_D and I_C, and 2 node voltages v_a and v_b. The equations are

- KCL for node a: $I_{R_1} = I_D$.
- KCL for node b: $I_D = I_C + I_{R_2}$.
- BCE for resistor R_1: $I_{R_1} = (V(t) - v_a)/R_1$.

- BCE for diode D with parameters I_S and V_T (a simple diode model): $I_D = I_S(e^{(v_a - v_b)/V_T} - 1)$.
- BCE for capacitor C: $I_C = \frac{d}{dt}(Cv_b)$.
- BCE for resistor R_2: $I_{R_2} = v_b/R_2$.

We have used KVL implicitly by assigning the voltage potentials v_a and v_b to the nodes a and b, respectively. The above 6 equations form a differential algebraic equation (DAE) that describes the circuit behaviour. This DAE can be written in the general form (1.2). □

In general, circuits can be modelled with a DAE of the form (1.2). For example, resistors, (time dependent) current and voltage sources, and diodes, contribute to the function j. Capacitors and inductors contribute to the function q. A popular systematic method for generating this DAE from the KCLs, KVLs, and BCEs is modified nodal analysis [49], see also [64, Ch. 1], [33]. Other methods may lead to DAEs of larger or smaller dimension. The numerical properties, such as the index of the DAE, may also depend on the chosen modelling technique.

1.2 Chapter 2 and 3: linear systems arising in transient analysis

In transient analysis the DAE

$$\frac{d}{dt}q(x(t)) + j(x(t), t) = 0 \in \mathbb{R}^n \,, \tag{1.3a}$$

$$x(0) = x_0 \,, \tag{1.3b}$$

is solved for $0 < t \leq T$. We assume that the DAE has consistent initial conditions.

One (simple) method for solving (1.3) numerically is the backward Euler method. This method approximates the derivative of $q(x(t))$ with a finite difference:

$$\frac{q(x_k) - q(x_{k-1})}{t_k - t_{k-1}} + j(x_k, t_k) = 0 \,, \quad k \geq 1 \,, \tag{1.4}$$

here x_k is an approximation to $x(t_k)$ and $t_0 = 0 < \ldots < t_{k-1} < t_k < \ldots \leq T$. The algebraic equation (1.4) is not necessarily linear. Newton's method can be used to solve x_k from (1.4). Then for each Newton iteration a linear system of the form

$$Ax_k^{(i)} = b^{(i)} \,, \tag{1.5}$$

has to be solved, where A is the Jacobian associated with (1.4):

$$A = \frac{\alpha_0}{h}C + G \,, \quad \text{with} \quad C = \left.\frac{\partial q}{\partial x}\right|_{x = x_k^{(i-1)}} \,, \quad G = \left.\frac{\partial j}{\partial x}\right|_{x = x_k^{(i-1)},\, t = t_k} \,, \tag{1.6}$$

$\alpha_0 = 1$, and $h = t_k - t_{k-1}$. The backward Euler scheme of (1.4) uses a first order approximation for the derivative $\frac{d}{dt}q(x)$. Higher order schemes can be used as well; for instance

$$\frac{d}{dt}q(x_k) \approx \frac{\sum_{i=0}^{m} \alpha_i q(x_{k-i})}{t_k - t_{k-1}},$$

with parameters α_i and $m < 7$. This leads to the so called BDF (backward differentiation formula) methods. Variable order BDF methods with variable step sizes are often used in practice in transient analysis of circuits. The linear systems arising in BDF methods have a structure similar as (1.5), with (1.6), but with a different value of α_0. Numerical problems may arise in the BDF method if the DAE (1.3) is of higher index, see [15, Ch. 3].

A forward Euler scheme for (1.3) is obtained by replacing $j(x_k, t_k)$ by $j(x_{k-1}, t_{k-1})$ in (1.4). However, in this case the Jacobian $A = C/h$, is likely to be singular. Therefore, forward Euler and also other explicit schemes are not suitable for (1.3).

The matrices G and C can be computed by using stamps. Each circuit element, for example resistor, inductor, and voltage source, has a stamp associated with it. These stamps describe the contribution of the element to the G and/or C matrices. A stamp adds nonzeros to G and/or C at specific positions when the circuit element is processed. This process leads to sparse matrices G and C. The matrix A has some nice properties if j and q are computed with modified nodal analysis:

- A is symmetric if it contains only non-controlled elements [52, Sect. 4.4].
- A is symmetric positive semidefinite if it contains only non-controlled capacitors, resistors or current sources [52, Sect. 4.4].
- The average number of nonzeros per row of A is often small, say less than 15; see Section 2.2.

Controlled elements are elements whose parameters depend on other circuit variables that do not belong directly to that element, for example: a current source with a current defined by a voltage difference somewhere else in the circuit. These elements are used frequently to model transistors. General circuits lead to unsymmetric and indefinite A, but the nonzero pattern is often rather close to symmetric. In general, the matrix A is not diagonally dominant, although many rows and columns may have a relatively large diagonal entry. Therefore, some form of pivoting is necessary if Gaussian elimination is used for the solution of (1.5).

1.2.1 Solving $Ax = b$

Traditionally, the linear systems (1.5) arising in transient analysis of circuits are solved by sparse Gaussian elimination, see for example [57]. This approach is usually rather efficient because the amount of fill-in in the LU factorization is often small for these problems if a suitable ordering is applied to the linear system. This can be, for example, the minimum degree [62] ordering applied to $A + A^T$. State-of-the-art sparse LU methods, such as SuperLU [22] and UMFPACK [19], often work with dense submatrices in

order to enable the use of efficient BLAS 2 and BLAS 3 routines. This does not seem to be very useful for circuit simulation matrices, because the L and U factors of A are usually too sparse to benefit from this approach, see Section 2.8.2.

Iterative methods for $Ax = b$ are not very popular in circuit simulation. These methods are usually more expensive than direct methods in terms of CPU-time, because it is difficult to identify effective preconditioners for $Ax = b$, see [8], [65]. However, recent results [59] indicate that ILU-type preconditioned Krylov subspace methods may be cheaper than direct sparse methods in terms of floating-point operations.

Existing parallel LU codes do not perform well for circuit simulation matrices, see Section 2.1. In Chapter 2 we will propose a fast sequential and efficiently parallelizable solver for (1.5). This is done by a tailored combination of sparse direct and preconditioned iterative techniques.

The parallel solver presented in Chapter 2 assumes that (1.5) has the following form:

$$\begin{bmatrix} A_{00} & 0 & 0 & A_{0m} \\ 0 & \ddots & 0 & \vdots \\ 0 & 0 & A_{m-1m-1} & \vdots \\ A_{m0} & \dots & \dots & A_{mm} \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_0 \\ \vdots \\ \vdots \\ b_m \end{bmatrix} . \tag{1.7}$$

This form can always be realized by permuting the rows and columns of A in (1.5). Block Gaussian elimination of $x_0, \ldots, x_{m-1}$ in (1.7) leads to a relatively small, and relatively dense, reduced system

$$Sx_m = y_m , \tag{1.8}$$

with the Schur complement $S = A_{mm} - \sum_{i=0}^{m-1} A_{mi} A_{ii}^{-1} A_{im}$. The unknowns $x_0, \ldots, x_{m-1}$ can be computed efficiently with a backward solve, after x_m has been solved from (1.8). The entire process is efficiently parallelizable, except for the solution of the reduced system (1.8). Solving the reduced system (1.8) with direct elimination may cost up to 80 percent of the total flops for solving (1.7), for the linear systems that we consider in Section 2.2. However, the Schur complement is often too small for an efficient parallelization of a direct LU method. Therefore, the Schur complement forms a potential bottleneck for parallel computation.

In Chapter 2 the reduced system (1.8) is solved with preconditioned GMRES [76]. This iterative approach is only efficient if a suitable preconditioner is used. We propose to construct a preconditioner K by discarding entries of S that are relatively small with respect to the diagonal entries. The number of nonzeros of such a preconditioner K may be more than 10 times less than the number of nonzeros of S. This preconditioner is used to transform (1.8) into the preconditioned linear system

$$K^{-1} S x_m = K^{-1} y_m .$$

The preconditioned matrix $K^{-1}S$ usually has a nice clustering of eigenvalues, which results in fast GMRES convergence. The most expensive part of preconditioned GMRES is the well parallelizable matrix-vector product with S. Also matrix-vector products of the form $K^{-1}v$ have to be computed in preconditioned GMRES This can be done in a relatively inexpensive way by a direct LU approach. Numerical experiments show

that the iterative approach for the Schur complement leads to a fast sequential and well parallelizable method.

The costs for constructing the linear systems (1.5), in transient analysis of circuits, are not negligible compared with solving these linear systems. For some circuits it may be even more expensive to construct the linear systems than to solve them. Therefore, the construction of the linear systems (1.5) should also be parallelized for an efficient parallel circuit simulator. The construction is relatively easy to parallelize. We will not consider this further in this thesis.

In Chapter 3 we discuss a number of implementation issues related to the parallel solver of Chapter 2.

1.3 Chapter 4: linear systems arising in periodic steady state analysis

A circuit driven by periodic sources, each with the same period T, will usually converge to a periodic steady state. This periodic steady state is the solution of the DAE (1.3a), with a periodic boundary constraint:

$$\frac{d}{dt}q(x(t)) + j(x(t),t) = 0 \in \mathbb{R}^n\,, \tag{1.9a}$$

$$x(0) = x(T)\,. \tag{1.9b}$$

We make the following assumptions on (1.9): $j(x,t+T) = j(x,t)$ for all t, the period T is not an unknown, and all explicitly time dependent coefficients and sources are periodic with period T. These assumptions will exclude free running oscillators.

In principle, (1.9) can be solved by transient analysis on a long time interval, because transient analysis will usually converge to a periodic solution $x(t)$. However, for some circuits this convergence may be rather slow; it may take more than, say, 1000 periods. This leads to an excessive amount of CPU-time for the simulation process. Therefore, it may be more efficient to solve the periodic problem (1.9) with a method that exploits the periodic structure. This can be done with, for example, the (multiple) shooting method or the finite difference method, see [7], [52, Ch. 7], [83], [84]. Here we will describe the finite difference method based on backward Euler.

The periodic DAE (1.9) is discretized on the discrete time points

$$t_0 = 0 < t_1 < \ldots < t_{M-1} < t_M = T\,,$$

with backward Euler. With $x_k \approx x(t_k)$, a system of nM algebraic equations is obtained:

$$\frac{q(x_k) - q(x_{\tilde{k-1}})}{t_k - t_{k-1}} + j(x_k,t_k) = 0\,, \quad k = 1,\ldots,M\,, \tag{1.10}$$

with

$$\tilde{k-1} \equiv \begin{cases} k-1\,, & \text{if } k = 2,\ldots,M\,, \\ M\,, & \text{if } k = 1\,. \end{cases}$$

In (1.10) we have used the periodic boundary constraint (1.9b). Therefore x_0 arises not explicitly in (1.10). The system (1.10) can be solved with Newton's method. This leads to a block linear system of the form:

$$\begin{bmatrix} \frac{C_1}{h_1}+G_1 & 0 & 0 & -\alpha\frac{C_M}{h_1} \\ -\frac{C_1}{h_2} & \frac{C_2}{h_2}+G_2 & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -\frac{C_{M-1}}{h_M} & \frac{C_M}{h_M}+G_M \end{bmatrix} \begin{bmatrix} x_1^{(i)} \\ \vdots \\ \vdots \\ x_M^{(i)} \end{bmatrix} = \begin{bmatrix} c_1^{(i)} \\ \vdots \\ \vdots \\ c_M^{(i)} \end{bmatrix}, \tag{1.11}$$

with

$$C_k = \left.\frac{\partial q}{\partial x}\right|_{x=x_k^{(i-1)}}, \quad G_k = \left.\frac{\partial j}{\partial x}\right|_{x=x_k^{(i-1)},\, t=t_k},$$

$\alpha = 1$, $h_k = t_k - t_{k-1}$, and an obvious right-hand side. Similar to transient analysis, the C and G matrices appear again, see Section 1.2. The linear systems arising in multiple shooting methods have a similar structure as (1.11), although with a smaller M.

A discretization based on the θ-method [46, Sect. II.7] is also possible. Then $j(x_k, t_k)$ in (1.9) is replaced by $\theta j(x_k, t_k) + (1-\theta) j(x_{k-1}, t_{k-1})$. For $\theta = 1$ this is equivalent to backward Euler. The choice $\theta = 0.5$ is equivalent to the trapezoidal rule. This choice is not practical, because it can be shown that it leads to a singular system (1.11), if M is even, C and G are constant, the step size is constant, and if C is singular. In principle, higher order BDF schemes can be used to discretize $\frac{d}{dt}q(x(t))$, but this leads to more nonzero blocks in (1.11), making the solution method more expensive.

Another application of linear systems of the form (1.11) arises in periodic AC analysis. In periodic AC analysis the effect of a small noise source on the behaviour of the circuit is studied. We will now briefly discuss the linear systems arising in periodic AC analysis; we refer to [85] for more details. The noise source may be modelled with a perturbation $s(t)$ applied to the DAE (1.9a):

$$\frac{d}{dt}q(x(t)) + j(x(t), t) + s(t) = 0\,. \tag{1.12}$$

The noise function $s(t)$ is of the form $s(t) = Ue^{2\pi \mathbf{i} f_n t}$, with $\mathbf{i} = \sqrt{-1}$, and a constant vector $U \in \mathbb{R}^n$. Usually one is interested in the behaviour of the circuit for several noise frequencies f_n. The solution of (1.12) can be written as $x = x_{\text{pss}} + x_{\text{n}}$, where x_{pss} is the periodic steady state solution of (1.9). Linearizing (1.12) around x_{pss} leads to a linear DAE for x_{n}. Backward Euler discretization of this DAE results in a linear system of the form (1.11), with $\alpha = e^{-2\pi \mathbf{i} f_n T}$. Periodic AC analysis leads to a multiple right-hand side problem, because (1.9a) has often to be solved for several noise frequencies f_n.

1.3.1 Solving periodic steady state matrices

Direct methods for the solution of (1.11) are considered in, for example, [4] and [7]. Usually these methods are rather expensive: $\mathcal{O}(n^3 M)$ flops if the sparsity of C_k and G_k is not exploited. Iterative methods may be much more efficient than direct methods, even when the direct method exploits the sparsity of C_k and G_k, see Section 4.10.

Our iterative approach of Chapter 4 is based on a reduction of (1.11) to a smaller p-cyclic linear system

$$\begin{bmatrix} I & 0 & 0 & -B_1 \\ -B_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \vdots \\ \vdots \\ \hat{x}_p \end{bmatrix} = \begin{bmatrix} \hat{b}_1 \\ \vdots \\ \vdots \\ \hat{b}_p \end{bmatrix} , \tag{1.13}$$

with $p \leq M$. This reduced system can be obtained from (1.11) by block diagonal scaling followed by block Gaussian elimination. Existing iterative methods for p-cyclic linear systems are not efficient or not efficiently parallelizable, see Section 4.2. Our p-cyclic GMRES method is often slightly more expensive than existing methods, but in contrast to these methods the method is efficiently parallelizable on p processors, even if the p-cyclic matrix has a rather small block size.

The idea behind p-cyclic GMRES is to build p independent search spaces with $\hat{x}_i \in \mathcal{K}_i^{(m)}$, $i = 1, \ldots, p$ (which are not Krylov subspaces in general). The residual norm of (1.13) is minimized over these search spaces.

1.4 Chapter 5: linear systems of the form $P(B)y = c$

In Chapter 5 we propose a new iterative approach for solving linear systems of the form

$$P(B)y = c\,, \tag{1.14}$$

where $P(B)$ is a polynomial in B. Applications of $P(B)y = c$ arise in, for example, lattice quantum chromodynamics and Tikhonov-Phillips regularization [34]. Another application of $P(B)y = c$ arises in higher order implicit methods for initial value problems [37, Ch. 8]. Equation (1.14) is not directly related to circuit simulation. We consider it here because we found that it can be solved by similar techniques as used in Chapter 4.

In several existing approaches for (1.14), see for example [24], [34], and [88], the approximate solution $y^{(m)}$ of (1.14) is selected from the Krylov subspace

$$\mathcal{K}^m(B, c) = \text{span}(c, Bc, \ldots, B^{m-1}c)\,.$$

The approximate solution $y^{(m)}$ is chosen such that the residual $r^{(m)} = c - P(B)y^{(m)}$, is small in some sense. In [24], [88], this is achieved by taking (using MATLAB notation)

$$y^{(m)} = \|c\| V_*^{(m)} P(H_*^{(m)}(1{:}m, 1{:}m))^{-1} e_1\,.$$

Here the Hessenberg matrix $H_*^{(m)}$ and the matrix $V_*^{(m)}$ are defined by the standard Arnoldi reduction

$$BV_*^{(m)} = V_*^{(m+1)} H_*^{(m)}\,,$$

associated with the Krylov subspace $\mathcal{K}^m(B, c)$, see for example [73, Ch. 6].

Our method for (1.14) is based on an augmented block linear system for (1.14). Without loss of generality we assume that the matrix polynomial of degree p can be written as

$$P(B) = (B + \mu_p I) \cdot \ldots \cdot (B + \mu_1 I) + \rho I \; .$$

The augmented block linear system is defined by

$$\begin{bmatrix} I & 0 & 0 & B + \mu_1 I \\ -(B + \mu_2 I) & \ddots & 0 & 0 \\ 0 & \ddots & I & 0 \\ 0 & 0 & -(B + \mu_p I) & \rho I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ c \end{bmatrix} \; , \tag{1.15}$$

Block Gaussian elimination of $x_1, \ldots, x_{p-1}$ in (1.15) leads to $P(B)x_p = c$, hence $x_p = y$, and we can focus on solving (1.15) instead of $P(B)y = c$. The idea of the methods proposed in Chapter 5 is to take $x_i \in \mathcal{K}^m(B, c)$, $i = 1, \ldots, m$, where each x_i is chosen such that the residual norm of (1.15) is minimized. This idea leads to a method called P(GMRES) for general matrices B, and a method P(MINRES) for the symmetric case. For $P(B) = B$ these methods reduce to the well known methods GMRES [76] and MINRES [68]. In comparison with existing methods, the new approach may lead to a smaller residual norm $\|r^{(m)}\| = \|c - P(B)y^{(m)}\|$, for the same number of iterations m, for $p > 1$. In Chapter 5 this new approach is compared with existing Krylov subspace methods for $P(B)y = c$, and the advantages and disadvantages are illustrated with a few examples.

1.5 Additional remarks

Chapter 2 has been published in an almost identical form as:

> C.W. Bomhof, H.A. van der Vorst, *A parallel linear system solver for circuit simulation problems.* Numer. Linear Algebra Appl. 7 (2000), pp. 649-665.[1]

A small part of Chapter 4 appears in the paper:

> W. Bomhof, H.A. van der Vorst, *A parallelizable GMRES-type method for p-cyclic matrices, with applications in circuit simulation.* To appear in the proceedings of the SCEE-2000 Workshop, August 20-23, 2000, Warnemünde.

Unless stated otherwise, $\| \cdot \|$ denotes in this thesis the 2-norm of a matrix or vector, although some (in)equalities may be valid for other norms as well. The MATLAB colon notation is used frequently to refer to matrix entries. For example $A(3, 4 : 6) = [a_{34} \; a_{35} \; a_{36}]$.

[1]

Chapter 2

A parallel linear system solver for circuit simulation problems

Abstract: In this chapter we present a parallel mixed direct/iterative method for solving linear systems $Ax = b$ arising from circuit simulation. The systems are solved by a block LU factorization with an iterative method for the Schur complement. The Schur complement is usually a relatively small and rather dense matrix. Nevertheless, direct LU decomposition of the Schur complement takes too much time in order to achieve reasonable speedup results. Our iterative method for the Schur complement is often much faster than the direct LU approach. Moreover, the iterative method is better parallelizable. This results in a fast sequential and well parallelizable method.

Keywords: preconditioner, parallel iterative method, mixed direct/iterative method, sparse LU factorization, circuit simulation, iterative solution methods, Schur complement, GMRES

AMS subject classifications: 65F10, 65F05, 65F50, 65Y05, 94C05

2.1 Introduction

In circuit simulation, often a series of linear systems has to be solved. For example, in transient analysis, a DAE leads in each time-step to a system of nonlinear equations, usually solved with the Newton method. For a single Newton step a linear system

$$Ax = b \tag{2.1}$$

has to be solved. Most circuit simulators handle this problem using an LU factorization of A.

Iterative methods for linear systems have been less effective in circuit simulation. McQuain *et al.* [65] did experiments with several variants of ILU in combination with GMRES, BiCGSTAB and other methods. They conclude that none of these iterative methods have acceptable performance for circuit simulation problems. Efficient parallel ILU-schemes have been developed recently by, for instance, Hysom and Pothen [53],

and Karypis and Kumar [56]. Our purpose is to find iterative or hybrid approaches that compete with direct methods on sequential machines, and that are parallelizable. Recently, the gap between iterative and direct methods has been narrowed for circuit simulation problems, for instance by Lengowski [59]. She has successfully used CGS with an incomplete LU drop tolerance preconditioner. In this approach, very small drop tolerances and only a few CGS steps are used. Compared to the LU approach this saves up to about 50 percent floating-point operations (flops). Nguyen[1] has used a block Gauss-Seidel method with a dynamic partitioning. This works well, but only for some types of circuit simulation problems. Unpreconditioned Krylov methods converge too slowly in practice.

Existing parallel LU codes do not perform well for circuit simulation matrices. Our experiments with the shared memory SuperLU code by Demmel *et al.* [21] showed no speedup for relevant circuit simulation test problems (the matrices of Section 2.2). SuperLU uses the column elimination tree of A for coarse grain parallelization. The column elimination tree of A is the elimination tree of A^TA, and for circuit matrices A^TA is often a nearly dense matrix. In that case, the column elimination tree is almost a single chain and then parallelization on a coarse grain level is not successful. On a fine grain level, SuperLU parallelizes the computations by pipelining the computations of the dependent columns. This introduces some parallel overhead, and the poor speedup indicates that this overhead is too large.

The undocumented sparse LU code PSLDU of Rothberg [72], an SGI product, only available on SGI platforms, is also not very suitable for circuit simulation matrices. This code allows no pivoting (shared memory SuperLU has partial pivoting), in order to make parallelization easier. However, this is not really effective, the speedup results were unsatisfactory for our test matrices. Also, for our problems, some form of pivoting is necessary for numerical stability. In the experiments with PSLDU, we preordered A with a suitable row permutation in order to avoid too small pivots on the diagonal. This is not very realistic, because in practice a suitable permutation can only be determined during the LU factorization. However, we were mainly interested in the maximum attainable speedup for PSLDU.

Recently, progress on parallel sparse LU methods for distributed memory machines has been made by Li and Demmel [60], by Jiang *et al.* [54], and by Amestoy *et al.* [2], [3]. Li and Demmel use static pivoting instead of partial pivoting in their SuperLU code. This makes the sparse LU factorization more scalable on distributed memory machines. The S^+ code of Jiang *et al.* starts with a static symbolic factorization to predict the nonzero patterns of L and U. In the second step this symbolic factorization is used in order to make a scalable numerical factorization. For circuit simulation matrices this approach is not very successful, see our Section 2.8.2. The MUMPS code, developed by Amestoy *et al.*, uses an asynchronous multifrontal approach with dynamic scheduling of the computing tasks.

In this chapter, we will propose a block LU factorization with an iterative method for the Schur complement to solve linear systems $Ax = b$. An outline of the method is given in Section 2.3. The preconditioner for the Schur complement is described in

[1]Personal communication with T.P. Nguyen (Philips Electronics, Eindhoven)

Section 2.4. In the sequential case there is some freedom in choosing the number of unknowns for the iterative part of the method. With a suitable choice it is possible to optimize the method, see Section 2.5. Section 2.6 describes an algorithm to find a suitable parallel block partition of the matrix. Possible pivot problems are discussed in Section 2.7. The last section of this chapter describes the numerical experiments.

2.2 Properties of circuit matrices

Matrices from circuit simulation are usually very sparse, e.g. the average number of nonzeros per row is usually smaller than 15. The nonzero pattern is often nearly symmetric. Table 2.1 shows some characteristics of our test matrices, arising from actual circuit simulation. The matrices are taken at some Newton step during a transient simulation of a circuit. Modified nodal analysis [49], [64, Ch. 1] was used to model the circuits. The sparse tableau analysis method [64] for modelling circuits might lead to matrices that are less suitable for our method.

problem	n	nnz(A) $\times 10^3$	h	flops $\times 10^6$	nnz($L+U$) $\times 10^3$
`circuit_1`	2624	36	131	0.86	41
`circuit_2`	4510	21	95	0.51	32
`circuit_3`	12127	48	85	0.54	68
`circuit_4`	80209	308	308	15.28	461

Table 2.1: Characteristics of test matrices. The dimension of the problem is n. nnz(A) is the number of nonzeros of A. flops is the number of MATLAB flops for $[L, U] =$ `lu`$(A, 0.001)$ and $x = U\backslash(L\backslash b)$. h is the height of the elimination tree of $A + A^T$ (assume no exact numerical cancellation). The matrices are available from: `http://www.math.uu.nl/people/bomhof/`.

We will assume that the matrix A already has a fill reducing ordering. The minimum degree ordering [62] of $A + A^T$ is a good ordering for circuit matrices that do not already have a fill reducing ordering. The minimum degree ordering Q should be applied symmetrically to A, that is, the permuted matrix is $Q^T AQ$. Several other ordering heuristics for circuit matrices are considered in [71]. There it is shown that a good (minimum local fill type) ordering heuristic may on average save 30 percent of the floating-point operations for an LU factorization, in comparison with the minimum degree ordering. Our approach for solving $Ax = b$, to be presented in the next sections, may also benefit from these orderings.

The diagonal pivot is not always a suitable pivot in the LU factorization, and partial pivoting is necessary for numerical stability. In practice threshold pivoting [25, Sect. 7.8] with very small thresholds, say 0.001, works fine (see also Section 2.8 and [57]).

Our test problems are rather small. Solving the largest problem takes only a few seconds on an average workstation. Nevertheless, even for these problems a parallel

solver is useful because one may have to solve thousands of these systems for one circuit simulation.

The amount of fill-in in LU factorization is usually very small for circuit simulation problems. For our test problems $\mathrm{nnz}(L+U) < 1.6\mathrm{nnz}(A)$, where $\mathrm{nnz}(A)$ is the number of nonzeros of A. Furthermore, the number of floating-point operations per nonzero entry is small (between 11 and 50). This implies that an iterative method is only cheaper than a direct method, when the number of iterations is small. Note that one matrix-vector product costs $2\mathrm{nnz}(A)$ flops, so the third and the fifth column indicate how much room there is for an iterative method. The height h of the elimination tree gives a rough indication of the possibilities for parallel elimination of unknowns, see Section 2.6.

2.3 Outline of the parallel solver

The parallel algorithm is based on a doubly bordered block diagonal matrix partition:

$$\hat{A} = P^T A P = \begin{bmatrix} A_{00} & 0 & \dots & 0 & A_{0m} \\ 0 & A_{11} & \ddots & \vdots & A_{1m} \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \dots & 0 & A_{m-1m-1} & \vdots \\ A_{m0} & A_{m1} & \dots & \dots & A_{mm} \end{bmatrix}, \tag{2.2}$$

where P is a permutation matrix. The block matrix partition (2.2) may also arise in domain decomposition methods. The permuted linear system is $\hat{A}\hat{x} = \hat{b}$, with $\hat{x} = P^T x$ and $\hat{b} = P^T b$. For circuit matrices often $n_m \equiv \mathrm{size}(A_{mm}) \leq n/20$, where n is the dimension of the linear system $Ax = b$. The matrix $\hat{A}$ is suitable for the parallel block LU linear solver with m processors, presented in Algorithm 2.1.

This is a coarse grained parallel algorithm, which means that the negative influence of parallel overhead on the speedup will be relatively small. As a part of the system solve one has to solve the reduced system

$$S x_m = y_m \,,$$

with the Schur complement S. This has to be solved as fast as possible in order to achieve a reasonable speedup. Solving $Sx_m = y_m$ directly may cost up to 80 percent of the total flops for solving $Ax = b$. So, the Schur complement forms a bottleneck for parallel computation.

A dense direct (parallelizable) method for the Schur complement may be of interest, because S is rather dense and the Mflop rates for dense methods are much higher than for sparse direct methods. In the sequential case ($m = 1$), this would lead to a method more or less similar to the (sequential) unsymmetric LU method MA48 [26] of Duff and Reid. This direct method switches from sparse to dense if the density of the reduced matrix is more than, say, 50 percent. Our preconditioned iterative method, to be presented in the next section, usually needs much less flops than the $2/3n_m^3$ flops for a direct LU method. This leads to a fast iterative solver for the Schur complement, although the

Algorithm 2.1: The parallel linear solver

parallel_for $i = 0 : m-1$,
  Decompose A_{ii}: $L_{ii}U_{ii} = P_{ii}A_{ii}$
  $L_{mi} = A_{mi}{U_{ii}}^{-1}$
  $U_{im} = {L_{ii}}^{-1}P_{ii}A_{im}$
  $y_i = {L_{ii}}^{-1}P_{ii}b_i$
  $S^{(i)} = L_{mi}U_{im}$
  $z^{(i)} = L_{mi}y_i$
end
$S = A_{mm} - \sum_{i=0}^{m-1} S^{(i)}$
$y_m = b_m - \sum_{i=0}^{m-1} z^{(i)}$
Solve parallel: $Sx_m = y_m$
parallel_for $i = 0 : m-1$,
  $x_i = {U_{ii}}^{-1}(y_i - U_{im}x_m)$
end

Mflop rate is low. The most expensive part of the iterative method, the matrix-vector product, is also well parallelizable.

A nice property of Algorithm 2.1 is that in exact arithmetic the residual norm of $Ax = b$ is equal to the residual norm of $Sx_m = y_m$, if $Sx_m = y_m$ is solved iteratively and $x_i = {U_{ii}}^{-1}(y_i - U_{im}x_m)$. This is easy to show.

In Section 2.6 we describe how to identify a permutation matrix P, which permutes the matrix A into the block form (2.2). Note that a circuit simulator can use the same permutation for P each Newton step because the sparsity pattern of A does not change. So, the costs of constructing a suitable P can be amortized over the Newton steps.

2.4 The preconditioner for the Schur complement

Unpreconditioned iterative methods for solving the reduced system $Sx_m = y_m$ converge slowly for circuit simulation problems. Preconditioning is a prerequisite for such problems, but it is difficult to identify effective preconditioners for the Schur complement S. The diagonal elements of S are often relatively large, but zeros on the diagonal may occur, so that simple diagonal scaling is not robust. The ratio of the smallest and the largest eigenvalues of S may be large, 10^6 or more, and S is likely to be indefinite, that is, it has both eigenvalues with positive and negative real part. This makes the problem relatively difficult for iterative solution methods.

Our preconditioner C is based on discarding small elements of the (explicitly computed) Schur complement S. The elements that are larger than a relative threshold define the preconditioner:

$$c_{ij} = \begin{cases} s_{ij} & \text{if} \quad |s_{ij}| > t|s_{ii}| \quad \text{or} \quad |s_{ij}| > t|s_{jj}| \\ 0 & \text{elsewhere,} \end{cases} \tag{2.3}$$

t is a parameter: $0 \leq t < 1$. $t = 0.02$ is often a good choice. Note that the preconditioner C will be symmetric if S is symmetric. The number of nonzeros of C can be much smaller than the number of nonzeros of S, see Figures 2.1 and 2.2. Construction of the preconditioner C costs only $3\text{nnz}(S)$ flops and is easy to parallelize.

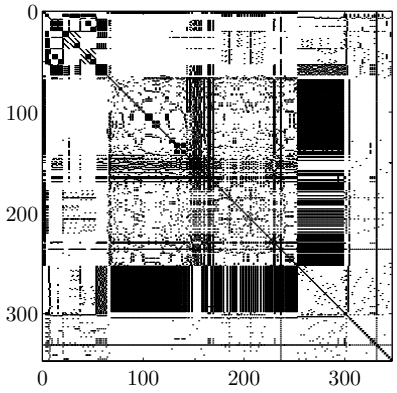


Figure 2.1: Nonzero pattern of a Schur complement S of problem `circuit_4`, $\text{nnz}(S) = 26675$, $n_m = \text{size}(S) = 346$.

For the preconditioner action, it is necessary to solve systems of the form $Cz = r$. These systems can be solved efficiently by sparse direct techniques. A sparse LU decomposition of C will be too expensive, because of a large amount of fill-in. However, the amount of fill-in will be small after reordering the matrix with the minimum degree algorithm [62]. For example, the matrix of Figure 2.2 has 1181 nonzeros and the sparse LU factorization of the reordered matrix has 1206 nonzeros, a fill-in of only 25 elements.

The minimum degree algorithm defines a permutation matrix P, so that we actually deal with D:

$$D = P^T C P \,.$$

The sparse LU factorization is performed with partial pivoting. This leads again to a permutation matrix, Q:

$$LU = QD = QP^T CP \quad \text{or} \quad C = PQ^T LUP^T \,.$$

Solving the system $Cz = r$ is straight forward.

Other orderings P for the preconditioner C are possible as well. For example MATLAB's `colperm` permutation is a very effective ordering for circuit simulation problems. In this ordering the number of nonzeros per column of D is nondecreasing. This ordering is very cheap to generate, because we only have to sort the number of nonzeros per column. This can be done in only $\mathcal{O}(n_m)$ ($n_m \equiv \text{size}(S)$) operations, which is important because the ordering algorithm is part of the sequential bottleneck of the parallel program. This `colperm` ordering sometimes gives poor results for non-circuit simulation problems. For instance, for discretized PDE problems the reverse Chuthill Mckee

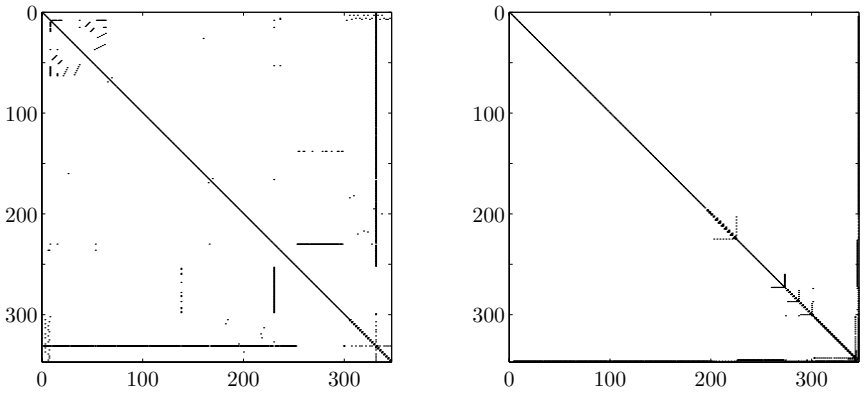


Figure 2.2: Left: nonzero pattern of the preconditioner C of the S from Figure 2.1, with $t = 0.02$, $\text{nnz}(C) = 1181$. Right: Matrix D, this is C after the minimum degree ordering.

ordering [41] turns out to be a successful alternative ordering for C (as became evident from our numerical experiments, not reported here).

In Section 2.8, we show that this preconditioner is very effective. The convergence of GMRES [76] is not sensitive to the threshold parameter t. The matrix-vector product Sv is the most expensive part of the GMRES process. For sufficiently large S, it is sensible to parallelize this matrix-vector product in order to increase the scalability of the parallel solver.

For the comparison of the direct and the iterative approach for solving $Sx_m = y_m$, we will treat S as a dense matrix. Solving the system with a dense direct LU solver costs $\approx 2/3{n_m}^3$ flops. Now, assume that after k_{G} GMRES steps the residual is sufficiently small and assume that the matrix-vector product flops of the GMRES process are the CPU-determining costs. Then the GMRES process costs $k_{\text{G}} \cdot 2{n_m}^2$ flops. Under these assumptions, the iterative approach is cheaper from about $n_m > 3k_{\text{G}}$. In terms of CPU-time this may be different, because the direct flops are likely to be faster than the iterative flops.

2.5 Switching from direct to iterative

In this section we will propose a criterion for the switch from the direct to the iterative part. With this criterion we try to minimize the computational costs of the method.

Solving $Sx_m = y_m$ with a preconditioned iterative method is much more efficient than with a direct method if S is relatively small (but not too small) and dense. The iterative method performs badly, compared with the sparse direct method, if S is large and sparse. Therefore, at some stage of the sparse LU factorization of A, it is more efficient to switch to the iterative approach. In the following we will try to find a criterion

for this. In deriving this criterion we will not try to be very precise, since only orders of magnitude matter. The aim of the derivation is mainly to give some insight in the differences in costs of the direct and the iterative part of the method. We also want to show that we have a reasonable criterion to choose between direct and iterative.

Now, suppose that there are $k+1$ unknowns left at a stage of the Gaussian elimination (GE) process, so the right lower submatrix in the GE process is an $(k+1) \times (k+1)$ matrix T:

$$T = \begin{bmatrix} c & d^T \\ e & V \end{bmatrix} ,$$

with V an $k \times k$ matrix. We will assume that d and e have the same sparsity pattern, because the sparsity pattern of A is nearly symmetric. So this is a reasonable approximation. T is a Schur complement and we can take it for system $Sx_m = y_m$, to be solved with an iterative solver. The costs of $k_{\rm G}$ matrix-vector products Sw are approximately

$$k_{\rm G} \, (\, 4 \, {\rm nnz}(e) + 2 \, {\rm nnz}(V) \,) \text{ flops.} \tag{2.4}$$

We can also decide to perform one more Gaussian elimination step. Then the Schur complement becomes

$$S = V - (1/c) \, ed^T \, ,$$

and $k_{\rm G}$ matrix-vector products plus the Gaussian elimination cost approximately

$$2 \, k_{\rm G} \, {\rm nnz}(V) + 2 \, {\rm nnz}(e)^2 \text{ flops} \ , \tag{2.5}$$

if we assume that ed^T introduces no extra fill-in in V. The amount in (2.4) is smaller than in (2.5) if

$$ {\rm nnz}(e) > 2 \, k_{\rm G} \, . \tag{2.6}$$

This is a motivation to use the number of nonzeros as a criterion for the choice between the direct and the iterative approach. However, we do not know $k_{\rm G}$ in advance. Moreover, we have neglected some numbers of flops, that are different for the k and $k+1$ case. For example, the flops for constructing the preconditioner, $O({\rm nnz}(e))$ flops in (2.5), the flops for the $\alpha x + y$ and $x^T y$ operations in GMRES and the flops for the forward and the backward solve of the preconditioner. Note also that the number of GMRES steps $k_{\rm G}$, required to reach a sufficiently small residual, increases slowly with increasing k. We conclude that criterion (2.6) is of limited use in practice. We use the criterion ${\rm nnz}(e) > q$, with a guessed parameter q instead of $2k_{\rm G}$. For the circuit simulation test problems $q = 40$ turns out to work well. Both CPU-time and the number of flops to solve the system $Ax = b$ are small for this q, see Section 2.8. The results are not sensitive to q.

The direct/iterative partition of the unknowns can be made before the actual block LU factorization takes place. It is based on the symbolic Cholesky factor L_C of $A + A^T$. In the algorithm c_i is used for the number the number of nonzeros in column i of L_C:

$$c_i = {\rm nnz}(L_C(:,i)) \, . \tag{2.7}$$

The nonzero structures of the L and U factors of $LU = A$ (without pivoting) are related to the nonzero structure of L_C: ${\rm struct}(L) \subseteq {\rm struct}(L_C)$ and ${\rm struct}(U) \subseteq {\rm struct}({L_C}^T)$.

The equal sign holds if struct(A) = struct(A^T). The nonzero pattern of A is nearly symmetric for circuit simulation matrices.

Algorithm 2.2 is the direct/iterative partitioning algorithm.

Algorithm 2.2: Direct/iterative partitioning

input: matrix A, parameter q
output: boolean vector *direct*

```
[L_C, parent] = symbolic_Cholesky(A + A^T)
direct(1 : n) = true
direct(root) = false
for i = 1 : n
   if (c_i ≥ q)
      j = i
      while (direct(j) = true)
         direct(j) = false
         j = parent(j)
      end
   end
end
```

The vector *parent* is a side product of the symbolic Cholesky algorithm. *parent*(i) is the parent of i in the elimination tree of $A + A^T$. *direct*(i) = *false* means that unknown i will be solved iteratively. Once the algorithm marks an unknown i 'iterative', all its ancestors are marked 'iterative'.

In the parallel case, one has to choose S at least so large that a load balanced distribution of the parallel tasks is possible, see next section.

2.6 Finding a suitable block partition

In this section we describe briefly how a permutation matrix P, which permutes the matrix A into form (2.2), can be found. We do not claim that our method to find P is the best one, but our approach is simple and works well, which is shown by the parallel experiments of Section 2.8.2.

We use the elimination tree of $A + A^T$ and the direct/iterative partition proposed in Section 2.5, to determine the permutation matrix P. The elimination tree breaks up into many (one or more) subtrees if we remove the unknowns of the Schur complement in the elimination tree. These subtrees are independent of each other in the Gaussian elimination process, see [61]. Grouping the subtrees in m groups (m is the number of processors), with approximately equal weights, gives the block matrix partition (2.2). A load balanced partition will not be possible if there are too large subtrees. In that case, one can easily split these subtrees into smaller ones, by moving more unknowns to the Schur complement. For some problems this results in a large Schur complement which

leads to modest parallel speed ups. With a small height h of the elimination tree it is usually possible to have both a load balanced partition and a small Schur complement. Circuit simulation matrices often have a small h relative to the number of unknowns n.

We will describe the partition algorithm in more detail in Chapter 3. Other implementation issues will also be discussed there.

Note that we have found the permutation matrix P in an indirect way. We start with a fill reducing ordering, then we use the elimination tree to find the independent blocks. Better parallel orderings might be possible by directly partitioning the graph of the matrix A into m independent subgraphs instead of partitioning the elimination tree of A. There exist packages such as METIS [55] and Chaco [47] for this task. It is not clear whether these packages are suitable for circuit simulation problems or not. The aim of METIS is to find a load balanced distribution with a Schur complement which is as small as possible. This does not have to be optimal for our mixed direct/iterative method where we like to have dense rows and columns in the Schur complement. METIS can assign weights to vertices and edges of the graph. However, in our case the weights depend strongly on the elimination ordering which is not known in advance. For a large number of processors it is desirable to have the Schur complement as small as possible because the direct part of the method (Algorithm 2.1) parallelizes much better than the iterative part. In that case packages like METIS and Chaco are useful.

2.7 Pivoting

Threshold pivoting [25, Sect. 7.8] with a small threshold t is used in the local LU decomposition of

$$\begin{bmatrix} A_{ii} \\ A_{mi} \end{bmatrix}.$$

The diagonal block A_{ii} almost always contains a suitable pivot for circuit simulation problems. However, in very exceptional cases there might be a suitable pivot only in the A_{mi} block. Then A_{ii} is nearly singular. This is not allowed, because exchanging rows of $\hat{A}$ destroys the structure of the block matrix (2.2). This problem is solved by moving the trouble causing unknown to the iterative part of the solver. So, the size of A_{ii} is decreased by one and the size of S is increased by one. This will not happen very often for circuit simulation problems. In fact we implemented the method without this pivot strategy, but that did not cause any stability problems for the circuit simulation test problems. This pivot strategy is comparable to the delayed pivot strategy which Duff and Koster [27] use in the multifrontal code. However, they permute large entries of A to the diagonal before the actual LU factorization.

This strategy can be refined by using two pivot threshold parameters. One for pivoting inside the diagonal blocks and one much smaller threshold to detect if it is necessary to move an unknown to the Schur complement. This improves the stability of the local LU decomposition of the diagonal blocks.

Note that in the sequential case pivoting is not restricted to the diagonal block A_{00}, partial pivoting without any restrictions is possible.

Another possibility is to replace small diagonal pivots a_{ii} by a suitably large entry if there is no pivot inside the diagonal block. This is equivalent by making a rank one correction $\beta e_i e_i^T$ to the original matrix A. Afterwards, the local L and U factors together with the preconditioner C can be used as a preconditioner for the system $Ax = b$. A drawback of this approach is that the iterative process for $Ax = b$ is much more expensive than for $Sx_m = y_m$. This is due to the longer vector lengths ($n \gg n_m$) and due to the more expensive matrix-vector product. In [60] Li and Demmel do not use partial pivoting during the LU decomposition at all. Instead, they permute large entries to the diagonal before the actual LU factorization. During the LU factorization they replace small pivots by larger ones. Iterative refinement is applied afterwards. This results in a scalable parallel LU code.

In subsequent Newton steps of a circuit simulation it is often possible to use the same pivot sequence during a number of Newton steps, as long as the pivots satisfy the threshold pivot criterion. The local LU factorization can take much advantage of an unchanged pivot sequence, because of reduction of symbolic overhead which leads to a faster factorization. Note that the matrices in the local LU factorization are extremely sparse which implies that the symbolic part of the LU factorization is relatively expensive.

2.8 Numerical experiments

2.8.1 The preconditioner

In this subsection we report on the preconditioner results for the problems of Section 2.2. We enlarged this set of problems with a number of other circuit simulation and non-circuit simulation problems. The full set of test problems is given in Table 2.2.

The matrices `memplus`, `orsirr_1`, `watt__1` and `sherman3` are available from the Matrix Market [63]. Matrix `memplus` is a circuit simulation matrix. The `TIx` matrices were kindly provided by Kai Shen. He used these matrices to test a parallel distributed memory sparse LU code [54]. The `TIx` matrices are circuit simulation matrices resulting from a transient simulation. All the matrices from the Matrix Market and the `TIx` matrices were permuted symmetrically with MATLAB's minimum degree ordering `symmmd` [41] of $A + A^T$. The original ordering of `circuit_`x is already a fine ordering. The matrices lap_... are discretized Laplacian operators on the unit square. Matrix `lap_nd128` is on a 128×128 grid and ordered with nested dissection, `lap_nd256` is on a finer 256×256 grid and `lap_md128` is ordered with minimum degree. For some problems there was no right-hand side b available. In that case we used a vector of all ones for b.

Note that the number of flops for an LU factorization of a circuit simulation matrix is rather sensitive to the ordering that is used. For example Larimore [58] reports for `memplus` $5597.6 \cdot 10^6$, $698.5 \cdot 10^6$, and $30.4 \cdot 10^6$ flops, for an LU factorization of A with the orderings colamd, colmmd, and amdbar. With MATLAB's `symmmd` only $2.0 \cdot 10^6$ flops are needed.

Two parameters are fixed for each different problem; the pivot threshold for the local LU is 0.001 and the GMRES tolerance is 10^{-7}. For circuit simulation problems

problem	n	nnz(A) $\times 10^3$	h	flops $\times 10^6$	nnz($L+U$) $\times 10^3$
circuit_1	2624	36	131	0.86	41
circuit_2	4510	21	95	0.51	32
circuit_3	12127	48	85	0.54	68
circuit_4	80209	308	308	15.28	461
memplus	17758	99	137	2.27	122
TIa	3432	25	249	8.07	100
TIb	18510	145	579	45.30	458
TIc	1588	13	65	0.18	18
TId	6136	53	306	11.63	155
lap_md128	16129	80	533	57.24	705
lap_nd128	16129	80	368	67.86	902
lap_nd256	65025	324	750	589.61	4563
orsirr_1	1030	7	195	2.65	51
watt__1	1856	11	355	11.47	126
sherman3	5005	20	482	22.28	220
heat	6972	28	696	116.11	692

Table 2.2: Characteristics of test matrices, see Table 2.1 for explanation of the symbols.

we use a preconditioner threshold of $t = 0.02$, a value of $q = 40$ for the direct/iterative parameter, and the `colperm` fill reducing ordering for C. This ordering is nearly as good as the minimum degree ordering, but much faster to compute. For non-circuit simulation problems we use $t = 0.005$ and the minimum degree ordering on C. The parameter q is chosen problem dependent: $q = \max(60, 0.5 \max_j(\text{nnz}(L(:, j))$, with L the symbolic Cholesky factorization of $A + A^T$. It turns out that these q, t and fill reducing orderings lead to small CPU-times and a nearly optimal (with respect to different parameter choices) number of flops. The parameter q is for circuit simulation problems smaller than for other problems. This can be explained partly by the slower GMRES convergence of non-circuit simulation problems. Note that the results are not very sensitive to the value of the parameters q and t. The results are shown in Table 2.3.

Although we developed the method for circuit simulation problems, the methods also work reasonably well for some other problems. For problems `circuit_3` and `TIc` parameter q is too large to do anything iteratively. For all the other problems the GMRES convergence is very fast which leads to a significant reduction of flops compared with the direct method. For `circuit_4` and `memplus` the number of nonzeros of the preconditioner C is more than 20 times smaller than the number of nonzeros of S. Nevertheless, the iterative method converges very well. For `circuit_4`, `TIa`, `TIb` and `TId` solving the Schur complement directly costs more than 80 percent of the flops for solving $Ax = b$ directly. This part can be done much faster with our preconditioned iterative method. For these problems a fast Schur complement solver is a prerequisite to have reasonable speedup results. Note that we have $n_m > 3k_G$ for each problem , which

	parameters							flops	
problem	q	t	Ord	n_m	nnz(S) $\times 10^3$	nnz(C) $\times 10^3$	k_{G}	direct $\times 10^6$	hybrid $\times 10^6$
`circuit_1`	40	0.020	col	127	7.5	.7	6	0.86	0.47
`circuit_2`	40	0.020	col	57	2.5	.3	5	0.51	0.47
`circuit_3`	40	0.020	col	0	0	0	0	0.54	0.54
`circuit_4`	40	0.020	col	344	26.8	1.1	6	15.28	3.15
`memplus`	40	0.020	col	166	10.8	.4	5	2.27	0.94
`TIa`	40	0.020	col	348	22.2	2.4	5	8.07	0.93
`TIb`	40	0.020	col	1265	63.0	10.4	12	45.30	7.38
`TIc`	40	0.020	col	0	0	0	0	0.18	0.18
`TId`	40	0.020	col	434	28.2	2.6	7	11.63	1.58
`lap_md128`	112	0.005	mmd	1011	138.5	15.8	19	57.24	27.65
`lap_nd128`	95	0.005	mmd	1305	140.3	16.0	17	67.86	30.44
`lap_nd256`	191	0.005	mmd	2649	591.3	32.2	24	589.61	252.81
`orsirr_1`	60	0.005	mmd	245	19.4	1.4	14	2.65	1.43
`watt__1`	88	0.005	mmd	432	51.4	9.1	8	11.47	5.26
`sherman3`	95	0.005	mmd	627	83.9	13.6	11	22.28	15.64
`heat`	193	0.005	mmd	894	222.6	26.4	18	116.11	56.04

Table 2.3: Solving the Schur complement iteratively. Ord is the symmetric ordering applied to C. Minimum degree (MATLAB's `symmmd`) is mmd and `colperm` is col. n_m is the dimension of the Schur complement S. The number of GMRES steps is k_{G}. The (MATLAB) flop counts are the number of flops to solve $Ax = b$ by a direct LU method and by our hybrid (mixed direct/iterative) method.

indicates that it is attractive to use the iterative approach for the Schur complement, see Section 2.4.

For problem `TIb` the LU factorization of the permuted preconditioner D is rather expensive. Considerable savings (in the number of flops) are possible by discarding small elements during the LU factorization of D. This can be done as follows: Start with the LU factorization of D and stop when the columns of L become too dense. Take the Schur complement, discard small entries as described in Section 2.4, and reorder this approximate Schur complement. Proceed with the LU decomposition until completion. Now we have a sort of incomplete LU factorization of D which can be used instead of the exact L and U factors of D. Problem `TIb` can be solved with only $4.02 \cdot 10^6$ flops by using this approach. This is a reduction of 46 percent compared to our original mixed direct/iterative method. Most of the test problems reported here are too small to benefit from this approach.

Our experiments showed that the time step, that the circuit simulator selects in the circuit simulation process, has almost no influence on the GMRES convergence for $Sx_m = y_m$.

2.8.2 Sequential and parallel experiments

The parallel method was implemented with multiprocessing C compiler directives for SGI shared memory systems. The matrix-vector product of the GMRES method was parallelized. The other parts of the GMRES method are not parallelized. We will discuss implementation issues in Chapter 3. Note that a distributed memory implementation is possible as well because we have a coarse grained algorithm that does not need complicated communication.

The direct solver we used (for the direct part of our method) was our own implementation of GP-Mod. This is the sparse LU method of Gilbert and Peierls [40] extended by the symmetric reductions of Eisenstat and Liu [30], [31], see also [22]. The sparse sequential LU code MA48 [26] uses the same techniques. However, MA48 uses a direct dense solver (via BLAS routines) for the rather dense reduced system, in contrast to our iterative approach. Any sparse LU method for the Schur complement is allowed as a direct solver, including, for example, multifrontal methods. We choosed GP-Mod because it is relatively easy to implement. Moreover, in [22] it is reported that, for circuit simulation problem `memplus`, GP-Mod is faster than SuperLU and also faster than the multifrontal code UMFPACK [19].

The same parameters are used as in Section 2.8.1. But for the non-circuit simulation matrices Liu's minimum degree ordering [62] is applied to C instead of MATLAB's `symmmd`. The results on the SGI Power Challenge are shown in Table 2.4. Table 2.5 shows the parallel results for the SGI Origin 200. The sequential results in the column 'direct' were obtained by setting the direct/iterative parameter q to $n+1$. In this case the code never switches from direct to iterative. Hence, we have obtained a sparse direct solver. With the Power Challenge we could measure the wall clock time because we were allowed to run our processes with the highest possible priority. For the Origin 200 we do not report the wall clock time because there were other minor processes on the system, which made accurate wall clock timing unreliable. Therefore, we measured the CPU-time, which is close to the wall clock time if the system load is modest. Note that processes which are waiting for synchronization are still consuming full CPU-time, so it is fair to measure CPU-time.

Problems `circuit_3` and `TIc` have no iterative part in the sequential case. So they do not benefit from the mixed direct/iterative approach, as already noticed in Section 2.8.1. In the parallel case, both problems have a small Schur complement, so the work involved with the Schur complement is only a small fraction of the overall amount of work, and the method is still parallelizable in this case. Problem `circuit_3` has good speedup results, `TIc` is too small to have good parallel results.

From the results in Section 2.8.1 one might expect a large reduction in CPU-time for some circuit simulation problems. But, for example, for `circuit_4` the gain is less than a factor of two. This is easy to explain by an example: Suppose we solve a problem directly and 80 percent of the flops is in the Schur complement part of the computations. The other 20 percent are very sparse flops which are much slower. So, suppose that the 80 percent takes 0.5 seconds and that the 20 percent also takes 0.5 seconds. Now, suppose we can speedup the Schur complement part by a factor of 10 by using an iterative method. Then the number of flops reduces from 100 percent to $20+80/10 = 28$ percent.

problem	Time (sec.)			Speedup				
	SuperLU	direct	hybrid	2	4	6	8	10
circuit_1	0.085	0.052	0.051	1.66	2.25	3.83	3.76	5.20
circuit_2	0.082	0.035	0.040	1.88	2.58	3.58	3.29	3.88
circuit_3	0.334	0.084	0.085	1.99	3.94	5.45	5.91	6.98
circuit_4	2.367	1.130	0.709	1.99	3.60	5.94	7.14	9.27
memplus	0.440	0.191	0.168	1.92	3.68	4.69	6.18	8.00
lap_md128	1.470	2.096	1.544	1.88	3.34	4.55	4.13	5.65
lap_nd128	1.740	2.534	1.643	1.79	2.97	3.95	5.31	4.56
lap_nd256	11.364	28.401	11.387	1.49	2.56	3.18	4.72	5.34
orsirr_1	0.076	0.083	0.075	1.42	1.59	2.36	2.27	2.50
watt__1	0.241	0.331	0.354	1.76	1.44	1.75	2.04	1.74
sherman3	0.440	0.662	0.559	1.61	1.85	2.43	2.08	2.10
heat	2.639	5.758	2.671	1.64	1.86	1.65	1.31	1.00

Table 2.4: Parallel results on a SGI Power Challenge with 12 R10000 processors at 195 MHz. The time to solve the system by SuperLU [22], by our own direct solver (see above text) and by our hybrid solver is in columns 2, 3 and 4. The speedup results in the next columns are relative to the results in the hybrid column.

problem	Time (sec.)			Speedup		
	SuperLU	direct	hybrid	2	3	4
circuit_1	0.080	0.051	0.048	1.57	2.00	2.17
circuit_2	0.074	0.035	0.037	1.80	2.22	2.37
circuit_3	0.285	0.074	0.074	1.86	2.75	3.62
circuit_4	1.781	0.882	0.558	1.89	2.81	3.50
memplus	0.358	0.169	0.143	1.83	2.49	3.19
TIa	0.430	0.254	0.090	1.54	1.57	1.63
TIb	3.198	1.509	0.538	1.53	2.30	2.39
TIc	0.037	0.017	0.017	1.74	2.34	2.45
TId	0.564	0.376	0.151	1.60	1.69	1.87
lap_md128	1.289	1.783	1.298	1.77	2.28	2.68
lap_nd128	1.557	2.191	1.416	1.67	2.15	2.82
orsirr_1	0.074	0.087	0.074	1.35	1.58	1.81
watt__1	0.209	0.331	0.307	1.61	1.33	1.35
sherman3	0.402	0.641	0.489	1.55	1.72	1.74
heat	2.227	3.905	2.202	1.48	1.73	1.83

Table 2.5: Parallel results on a SGI Origin 200 with 4 R10000 processors at 180 MHz. See also Table 2.4 for explanation.

The CPU-time reduces only from 1 second to $0.5 + 0.5/10 = 0.55$ seconds. Moreover, the direct Schur complement flops are faster than the iterative ones and the iterative method introduces some extra overhead due to datastructures, non-flop operations for reordering the preconditioner C etc. In the worst case (`circuit_2`) there is even a small loss in CPU-time.

For reference the results of the SuperLU code [22] are also in the tables. For circuit simulation problems our direct method is faster than SuperLU. This is not remarkable, because of the sparsity of circuit simulation problems, see [22]. The difference in time between our direct implementation and the SuperLU code shows that our direct implementation is sufficiently efficient although the Mflop rate is low. For `circuit_4` we have only $15.39/1.1297 = 13.6$ Mflops which is much lower than the peak performance of 390 Mflops of the SGI R10000 processor. For the mixed direct/iterative solver the Mflops rate is even worse. This is normal for circuit simulation problems because of, among other things, the poor cache reuse and the large symbolic overhead for these extreme sparse problems.

For the non-circuit simulation problems SuperLU is always faster than our direct sparse LU implementation. However, the speed of our hybrid method is often comparable to the SuperLU method for these problems. This is remarkable because it was not the intention to solve these kind of problems with our method. SuperLU uses supernodes in order to perform most of the numerical computation in dense matrix kernels. In contrast, our method tries to keep everything sparse. The fast results of our method were obtained with a q parameter such that a relatively large part of the work is in the iterative part of the method. As a consequence the parallel speedup results are not exceptional, because the iterative part is less well parallelizable than the direct part of the method (only the matrix-vector product of the iterative method has been parallelized). For non-circuit simulation problems, there is often a significant amount of work in the other parts of the iterative method. However, for our test problems these parts are too small for parallelization.

For circuit simulation problems `TIa`, `TIb` and `TId` something similar occurs. The sequential timings for these problems are up to three times faster than the timings for the direct method and the parallel speedup results are a bit disappointing. This is due to the expensive Schur complement system solve. We may conclude that a good sequential performance often leads to less well parallel results. The best parallel results are obtained with the three circuit simulation matrices: `circuit_3`, `circuit_4` and `memplus`. These problems have a heigth h of the elimination tree which is small relative to the number of unknowns n, this is good for parallelization, see Section 2.6. The problems `circuit_1`, `circuit_2` are too small for reasonable speedups.

The overhead for determining the block partition (see Section 2.6) and for memory allocation is not included in the timings because we assumed that we are in a Newton process. So these actions have to be done only once and can be reused in the Newton process.

For all the problems reported here it was possible to find a suitable pivot inside the diagonal block. However, there exist problems for which this is not possible (for example, Matrix Market [63] problems `lnsp3937` and `e20r3000`). A solution to this problem is described in Section 2.7. We have not implemented this in the code. Therefore, we

had to choose a smaller LU pivot threshold in some cases. For problems `circuit_1` and `circuit_3` we used a pivot threshold of 0.0001 if there were 5 or more processors. For `TIb` we always used a value of 0.0001. These small pivot thresholds do not lead to stability problems for these problems.

For the CPU-time measurements of the sequential method we used the same parameters as in the previous subsection. The iterative part of the method is less well parallelizable than the direct part of the method. Therefore, we increased parameter q for some non-circuit simulation problems and for the problems `TIx` if there were 3 or more processors. This results in more direct work and in less iterative work. Moreover we increased the preconditioner threshold t in order to reduce the costs of the not parallelized preconditioner action and to increase the number of parallel matrix-vector products. Globally this leads to slightly faster parallel results.

Demmel, Gilbert and Li also report on parallel results for problems `memplus` and `sherman3` with their shared memory SuperLU code [21]. These results have been copied in Table 2.6. The speedup results for `memplus` are much worse than ours. For the non-circuit simulation problem `sherman3` our results are worse.

	Speedup	
problem	4 proc.	8 proc.
`memplus`	1.73 (3.68)	1.73 (6.18)
`sherman3`	2.36 (1.85)	2.78 (2.08)

Table 2.6: Parallel speedups of SuperLU on a SGI Power Challenge with 16 R8000 processors at 90 MHz, from [21]. Our results (from Table 2.4) are between parentheses.

Jiang, Richman, Shen, and Yang report 12.81 seconds for `TIb` on 8 processors of a 450 MHz Cray T3E [54] which is not very fast. This is caused by the unlucky combination of the minimum degree ordering and the 'S^+' factorisation method for circuit simulation problems, although a better ordering has not been identified yet. For a number of non-circuit simulation problems they have very nice results.

2.9 Concluding remarks

In this chapter we have proposed a preconditioned iterative method for the solution of a Schur complement system for circuit simulation problems. This leads to an efficient sequential method which is sometimes much faster than direct sparse LU factorization. Moreover, the method is often well parallelizable which is supported by our parallel experiments. Note that a good sequential performance does not automatically lead to good parallel results. The method is not restricted to circuit simulation problems, although the results for these problems are better than for most other problems.

For the problems considered here, the Schur complement was too small for a full parallelization of the iterative solver; we only parallelized the matrix-vector product.

Therefore, for large problems our method is not as scalable as a full parallel direct method may be. However, for large problems and a small number of processors (say $p \leq 8$) our method is still competitive compared to a full direct method, because our partly parallelized iterative method for the Schur complement can be as fast as a fully parallelized direct method.

The sequential bottleneck in the current implementation of our method is due to the sparse LU decomposition of the permuted preconditioner $D = P^T C P$, the forward and backward solves with the L and U factors of $LU = QD$, and the GMRES vector updates and inner products for the small system. For sufficiently large Schur complements efficient parallelization of these components is feasible, which will improve the parallel scalability for large linear systems $Ax = b$.

Chapter 3

Implementation of a parallel mixed direct/iterative linear system solver

Abstract: In this chapter we discuss some implementation details of the parallel mixed direct/iterative linear system solver of Chapter 2. This solver is based on block LU factorization with an iterative method for the Schur complement. We discuss an efficient shared memory implementation of the block LU factorization. Furthermore, the partitioning algorithm which partitions the linear system $Ax = b$ into load balanced blocks is described.

Keywords: parallel computing, sparse LU factorization, preconditioned iterative methods

AMS subject classifications: 65F10, 65F05, 65F50, 65Y05, 94C05

3.1 Introduction

In Chapter 2 we presented a parallel mixed direct/iterative method for solving linear systems $Ax = b$ arising from circuit simulation. In that chapter we stated that the method is efficiently parallelizable, which was justified by parallel experiments. Here we will discuss the actual parallel implementation in more detail. The block LU factorization part of the parallel solver will be considered in Section 3.2. We will try to give some insight into the communication patterns of the shared memory implementation in Section 3.3. In Section 3.4 we discuss the algorithm which partitions matrix A into block form (2.2).

This chapter can be seen as a supplement to Chapter 2 because it considers only topics related to the parallel solver of Chapter 2. We assume that the reader has read Chapter 2.

3.2 Parallel block LU factorization

The reduced system $Sx_m = y_m$ is constructed in Algorithm 2.1 of Chapter 2. Here we will rewrite this algorithm in a slightly different form that is more suitable for parallelization

and implementation. In order to do this we distribute the rows of A_{mm} and b_m over m matrices and m vectors with the wrap mapping defined by the diagonal matrix $Q^{(i)}$:

$$A_{mm}^{(i)} = Q^{(i)} A_{mm}, \quad b_m^{(i)} = Q^{(i)} b_m, \quad i = 1, \ldots, m-1\,,$$

with

$$(Q^{(i)})_{pq} = \begin{cases} 1 & \text{if } \; p = q \;\; \text{and} \;\; p \bmod m = i\,, \\ 0 & \text{elsewhere.} \end{cases}$$

Note that $\sum_{i=0}^{m-1} Q^{(i)} = I$ and $\sum_{i=0}^{m-1} A_{mm}^{(i)} = A_{mm}$. For example, a 7 by 7 sparse matrix A_{33} is split as follows:

$$A_{33} = \begin{bmatrix} \star & \star & & \star & & & \\ \star & \star & \star & & \star & & \\ & \star & \star & & & & \star \\ \star & & & \star & \star & & \\ & \star & & \star & \star & \star & \star \\ & & & & \star & \star & \star \\ & & \star & & \star & \star & \star \end{bmatrix} = A_{33}^{(0)} + A_{33}^{(1)} + A_{33}^{(2)} =$$

$$\begin{bmatrix} \star & \star & & \star & & & \\ \star & & & \star & \star & & \\ & & \star & & & \star & \star \\ & & \star & & & \star & \star \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{bmatrix} + \begin{bmatrix} \star & \star & \star & & \star & & \\ & & & \star & & \star & \star \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{bmatrix} + \begin{bmatrix} & & & & & & \\ & & & & & & \\ \star & \star & & & & \star & \\ & & & & & & \\ & & & & \star & \star & \star \\ & & & & & & \\ & & & & & & \end{bmatrix},$$

if zero-based indexing is used for the matrix entries. Now we can write the construction of the reduced system, see Algorithm 2.1 on page 15, in a slightly different form given in Algorithm 3.1.

Algorithm 3.1: Construction of $Sx_m = y_m$

for $i = 0 : m-1$,
  Decompose A_{ii}: $L_{ii}U_{ii} = P_{ii}A_{ii}$
  $L_{mi} = A_{mi}{U_{ii}}^{-1}$
  $U_{im} = {L_{ii}}^{-1}P_{ii}A_{im}$
  $y_i = {L_{ii}}^{-1}P_{ii}b_i$
  $\bar{S}^{(i)} = A_{mm}^{(i)} - L_{mi}U_{im}$
  $\bar{z}^{(i)} = b_m^{(i)} - L_{mi}y_i$
end
$S = \sum_{i=0}^{m-1} \bar{S}^{(i)}$
$y_m = \sum_{i=0}^{m-1} \bar{z}^{(i)}$

In this formulation $\bar{S}^{(i)}$ can be seen as the Schur complement of A_{ii} in

$$\begin{bmatrix} A_{ii} & A_{im} \\ A_{mi} & A_{mm}^{(i)} \end{bmatrix}. \tag{3.1}$$

The matrices defined in Algorithm 3.1 arise naturally in the partial LU factorization (with restricted pivoting) of (3.1):

$$\begin{bmatrix} L_{ii} & 0 \\ L_{mi} & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \bar{S}^{(i)} \end{bmatrix} \begin{bmatrix} U_{ii} & U_{im} \\ 0 & I \end{bmatrix} = \begin{bmatrix} P_{ii} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{ii} & A_{im} \\ A_{mi} & A_{mm}^{(i)} \end{bmatrix}. \tag{3.2}$$

The vectors $\bar{z}^{(i)}$ and y_i follow from

$$\begin{bmatrix} y_i \\ \bar{z}^{(i)} \end{bmatrix} = \begin{bmatrix} L_{ii} & 0 \\ L_{mi} & I \end{bmatrix}^{-1} \begin{bmatrix} P_{ii} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} b_i \\ b_m^{(i)} \end{bmatrix} . \tag{3.3}$$

The outer product LU factorization method [42, Algorithm 3.2.3] leads to (3.2) if index k of the outer loop runs from 1 to $n_i = \text{size}(A_{ii})$ instead of $n_i + n_m = \text{size}(A_{ii}) + \text{size}(A_{mm})$, and with pivoting restricted to the A_{ii} block.

For sparse matrices, a left-looking column oriented LU factorization[1] method may be preferred above an outer product LU factorization method, because such a method allows for more efficient sparse matrix data structures [40], [22]. Algorithm 3.2 computes the partial LU factorization 3.2 in a column oriented way. For the ease of presentation we

Algorithm 3.2: Column oriented partial LU factorization without pivoting

Input: A in the form (3.1)
Output: L, S, and U such that $LSU = A$ in the form of equation (3.2), with $P_{ii} = I$.

$n_i = \text{size}(A_{ii})$
$n_m = \text{size}(A_{mm})$
$n = n_i + n_m$
$L = I_n$
$S = \begin{bmatrix} I_{n_i} & 0 \\ 0 & 0_{n_m} \end{bmatrix}$
$U = 0_n$
for $j = 1 : n_i$
  $v = L^{-1}A(:, j)$
  $U(1{:}j, j) = v(1{:}j)$
  $L(j + 1{:}n, j) = v(j + 1{:}n)/v(j)$
end
for $j = n_i + 1 : n$
  $v = L^{-1}A(:, j)$
  $U(1{:}n_i, j) = v(1{:}n_i)$
  $S(n_i + 1{:}n, j) = v(n_i + 1{:}n)$
end
$U(n_i + 1{:}n, n_i + 1{:}n) = I_{n_m}$

have not included pivoting in Algorithm 3.2. The diagonal entries of L are not explicitly stored in practice, because they all have a value equal to 1. Our sparse implementation of Algorithm 3.2, with restricted pivoting, is based on the sparse partial pivoting idea's of Gilbert and Peierls [40], extended with the symmetric reductions of Eisenstat and Liu [30], [31]. The sparse sequential LU code MA48 [26] also uses these techniques, but uses a direct dense solver (via BLAS routines) for the rather dense reduced matrix.

[1] Column oriented LU factorization is also known as 'gaxpy' or 'j k i' Gaussian elimination [42, Ch. 3].

3.3 Parallelization of the solver

In this section we will try to give some insight into the actual parallel implementation and the communication patterns of the solver. We assume that the solver is implemented on a shared memory parallel computer with m processors, see Figure 3.1.

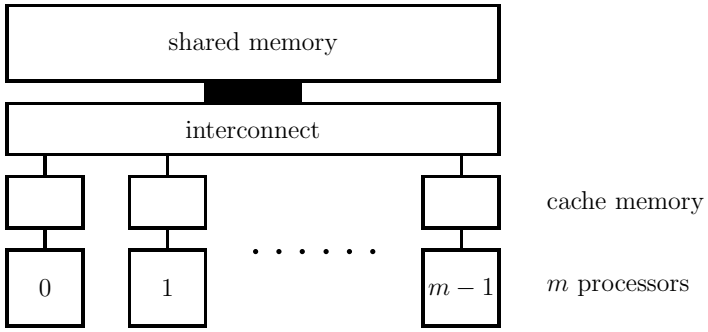


Figure 3.1: A shared memory parallel computer with m processors.

Processor i can communicate with processor j, $i \neq j$ by writing data to the shared memory, followed by a barrier synchronization command. This barrier synchronization command stops a processor until all the other processors have reached a barrier synchronization command. After such a synchronization processor j reads the data it requires from the appropriate shared memory locations. In Algorithm 3.3 this is indicated by the *get* command. Obviously, communication between different processors is more expensive than reading data from local cache memory.

We used the multiprocessing C compiler directives for SGI shared memory systems to parallelize the solver. With this parallel programming model there is no need to write or read data explicitly to or from the shared memory, because there is one global address space for the shared memory. This makes parallel programming relatively simple. Nevertheless, in Algorithm 3.3 we used the *get* statement in order to make the communication visible for the reader.

The solver can be implemented equally well on a distributed memory computer, because the communication patterns are rather simple. In that case explicit send and/or receive commands are necessary in the computer code.

Algorithm 3.3 is a more detailed description of the parallelized linear solver than Algorithm 2.1. In the following we will give a brief explanation of the algorithm. Between line $^{(b)}$ and $^{(d)}$ the Schur complement S and the preconditioner C (see Section 2.4) are constructed in parallel. Between line $^{(e)}$ and $^{(h)}$ the reduced system $Sx_m = y_m$ is solved with preconditioned GMRES, where only the matrix-vector product is parallelized. At line $^{(f)}$ a permutation for C is computed by processor 0, in order to make an efficient LU factorization possible, see Section 2.4. At line $^{(g)}$ processor 0 solves $Sx_m = y_m$ with preconditioned GMRES. Processor 0 sends a message to the other processors i if it needs

Algorithm 3.3: parallel linear solver with communication

Barrier synchronization is denoted by a thick line: ——————.
Comments are in *italic*.

begin_parallel

(a) i=my_proc_number
compute a partial LU decomposition, see equation (3.2) and (3.3):
(b) $[U_{ii}, U_{im}, \bar{S}^{(i)}, \bar{z}^{(i)}, y_i] = \text{partial\_LU}(A_{ii}, A_{im}, A_{mi}, A_{mm}^{(i)}, b_i, b_m^{(i)})$
get Schur complement contributions from other processors:
(c) for $j = 0 : m-1$
  get $Q^{(i)}\bar{S}^{(j)}$ from processor j
end
add Schur complement contributions:
$Q^{(i)}S = \sum_j (Q^{(i)}\bar{S}^{(j)})$
get diagonal entries of Schur complement from other processors:
for $j = 0 : m-1$
  get $\text{diag}(Q^{(j)}S)$ from processor j
end
compute contributions to the preconditioner C, see formula (2.3):
(d) $Q^{(i)}C = \text{precon}(Q^{(i)}S, \text{diag}(S), t)$
(e) if $i = 0$
  collect the preconditioner in processor 0 and compute y_m
  for $j = 0 : m-1$
    get $Q^{(j)}C$ from processor j
    get $\bar{z}^{(i)}$ from processor j
  end
  $y_m = \sum_{j=0}^{m-1} \bar{z}^{(j)}$
  compute fill reducing ordering for C:
(f) $P_C = \text{colperm}(C)$
  make an LU factorization of the permuted C:
  $[L_D, U_D, P_D] = \text{LU}({P_C}^T C P_C)$
  solve $Sx_m = y_m$ with GMRES on processor 0:
(g) $x_m = \text{GMRES}(Q^{(0)}S, y_m, L_D, U_D, P_D, P_C)$
else
  compute matrix $\times$ vector: $Q^{(i)}Sv$, if GMRES on processor 0 requests for it:
  $w = \text{matvec}(Q^{(i)}S, v)$
(h) end
get x_m from processor 0
backward solve for x_i:
(i) $x_i = {U_{ii}}^{-1}(y_i - U_{im}x_m)$

end_parallel

a matrix-vector product $Q^{(i)}Sv$. This can be implemented with two synchronization points per iteration. Note that it is also possible to run GMRES redundantly on each processor. Then only one synchronization per matrix-vector product is needed, instead of two.

We see in Algorithm 3.3 that the communication patterns between the different processors are rather simple: the number of synchonizations and the number of *get* commands are low. In the *for* loop at line $^{(c)}$ in Algorithm 3.3 a relatively large amount of data is moved from other processors to processor i, compared with other *get* commands in Algorithm 3.3.

Note that the first and last synchronizations in Algorithm 3.3 are not necessary. However in a parallel circuit simulator these synchronizations are needed in order to separate the construction of the linear system from the solution part, see Algorithm 3.4.

Algorithm 3.4: Parallel circuit simulation

```
begin_parallel
   while the simulation is not finished
      compute the coefficients of Ax = b (equation (2.1)) (in parallel)
      solve the linear system, see Algorithm 3.3 from line (a) to (i)
   end
end_parallel
```

3.4 The partitioning algorithm

In this section we will describe the partitioning algorithm, which permutes (2.1) into the block linear system (2.2), in more detail than in Section 2.6. Our approach is rather simple. Advanced algorithms for scheduling the parallel Cholesky factorization are proposed in, for example, [39] and [69]. Variants of these algorithms can also be used for our partitioning problem. The partitioning algorithm is described in Algorithm 3.5. In the following, we will clarify some parts of Algorithm 3.5.

Algorithm 3.5 starts by identifying the independent subtrees of the elimination tree, see Section 2.6. These independent subtrees can be identified as follows: The elimination tree has nodes $1, \ldots, n$ and *root*. We will try to find an ordered list $F = (f_1, \ldots, f_{n_T})$ of nodes f_i with $direct(f_i) = true$ and $direct(parent(f_i)) = false$, see also Algorithm 2.2, Section 2.5. These nodes are the roots of the subtrees $T[f_i]$. In the Gaussian elimination process the unknowns associated with the subtree $T[f_i]$ are independent of the unknowns associated with $T[f_j]$, if $i \neq j$ [61]. The nodes $F = (f_1, \ldots, f_{n_T})$ can be found by making a preorder traversal [1, Ch. 2] through the elimination tree. Each time when a node is visited with $direct(f_i) = true$ and $direct(parent(f_i)) = false$, this node f_i is added to the list F.

Algorithm 3.5: The partitioning algorithm

identify the root nodes $F = (f_1, \ldots, f_{n_T})$ of the independent...
  subtrees $T[f_1], \ldots, T[f_{n_T}]$
compute the weights $w_1, \ldots, w_n$
repeat
  assign the subtrees using formula (3.7) to the processors i
  compute the total weight per processor
  if the imbalance H (see formula (3.8)) is too large
    identify the heaviest loaded processor j
    choose the best subtree of processor j to split into smaller subtrees
  end
until imbalance H is small

Example Algorithm 2.2 (Section 2.5) partitions the elimination tree in a direct and an iterative part, see the example in Figure 3.2. In this example there are three independent subtrees: $T[5]$, $T[8]$, and $T[12]$. □

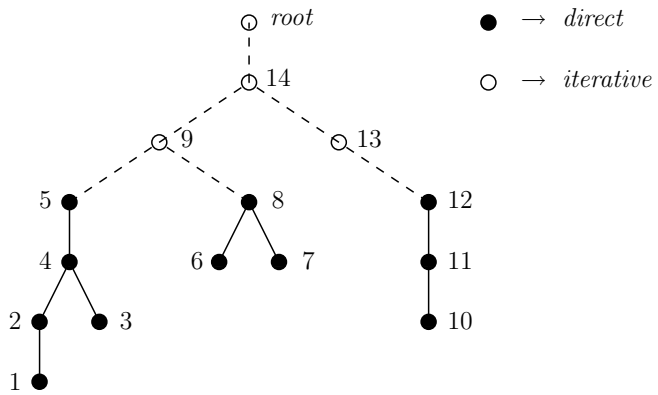


Figure 3.2: The subtrees $T[5]$, $T[8]$, and $T[12]$, are independent subtrees in the elimination tree.

The Gaussian elimination process is parallelized by distributing the subtrees $T[f_j]$, $j = 1, \ldots, n_T$ and their associated unknowns over the m processors. The subtrees are distributed such that each processor gets approximately the same amount of work. In the following we will explain how to do this. The amount of work for eliminating unknown i is modelled with the cost function

$$t_i = c_i^2 + \alpha c_i + \beta \ ,$$

with parameters α and β and the c_i of (2.7). The c_i^2 is proportional to the number

of flops needed to eliminate unknown i if we assume that the matrix has a symmetric structure. The $\beta + \alpha c_i$ accounts for the symbolic overhead that is introduced by each unknown and by each nonzero of L and U. In practice the partitioning results are not very sensitive to the values of parameters α and β. Gaussian elimination of all the unknowns in a subtree $T[i]$, $i = 1, \ldots, n, root$, with root i costs

$$w_i = t_i + \sum_{j \in \text{descendants}(i)} t_j \tag{3.4a}$$

$$= t_i + \sum_{\{j \mid \text{parent}(j)=i\}} w_j \ . \tag{3.4b}$$

All the weights w_i, $i = 1, \ldots, n, root$ can be efficiently computed in a recursive way by (3.4b) during one postorder tree traversal [1, Ch. 2] through the elimination tree.

Now each subtree $T[f_j]$, $j = 1, \ldots, n_T$ has a weight w_{f_j}, which is a measure for the costs of elimination of the unknowns associated with $T[f_j]$. The total weight is

$$W_{\text{tot}} = \sum_{j=1}^{n_T} w_{f_j} \ .$$

The independent subtrees are distributed over the m processors by cutting the ordered list $F = (f_1, \ldots, f_{n_T})$ into m approximately equally weighted sublists $(F_0, \ldots, F_{m-1})$:

$$F = (f_1, \ldots, f_{n_T}) = \tag{3.5}$$

$$(F_0, \ldots, F_{m-1}) = (\, (f_{\underline{k}_0}, \ldots, f_{\overline{k}_0}), (f_{\underline{k}_1}, \ldots, f_{\overline{k}_1}), \ldots, (f_{\underline{k}_{m-1}}, \ldots, f_{\overline{k}_{m-1}}) \,) \ . \tag{3.6}$$

The subtrees $T[f_k]$ are mapped to the m processors in a linear way:

$$f_k \in F_i \ \text{ if } \ \left\lfloor \ m \left(\sum_{j=1}^{k-1} w_{f_j} + w_{f_k}/2 \right) / W_{\text{tot}} \ \right\rfloor = i \ . \tag{3.7}$$

The function $\lfloor \cdot \rfloor$ rounds to the previous nearest integer.

The total weight for processor i is

$$W_i = \sum_{k=\underline{k}_i}^{\overline{k}_i} w_{f_k} \ .$$

We define the imbalance H of the distribution as

$$H = \max_{i=0,\ldots,m-1} W_i/(W_{\text{tot}}/m) - 1 \ . \tag{3.8}$$

If the imbalance is large, say $H > 0.02$, and if processor i is the heaviest loaded processor then the subtree $T[f_{\underline{k}_i}]$ or $T[f_{\overline{k}_i}]$ is split into smaller subtrees. If the weight $w_{f_{\overline{k}_i}}$ 'fits better' on processor i than the weight $w_{f_{\underline{k}_i}}$, then $T[f_{\underline{k}_i}]$ is split, otherwise $T[f_{\overline{k}_i}]$ is split. This choice is based on the following criterion: if

$$\left| \, m \left(\sum_{j=1}^{\underline{k}_j - 1} w_{f_j} \right) / W_{\text{tot}} - i \, \right| > \left| \, m \left(\sum_{j=1}^{\overline{k}_j} w_{f_j} \right) / W_{\text{tot}} - i \, \right| ,$$

then $T[f_{\underline{k}_j}]$ is split, otherwise $T[f_{\overline{k}_j}]$ is split. A subtree $T[i]$ is split into smaller subtrees by removing nodes at the root of the subtree until the subtrees breaks into two or more subtrees.

A better load balance, with the same list of subtrees $T[f_1], \ldots, T[f_{n_T}]$, is possible if more general subtree distributions are allowed than (3.5). For example, we may distribute $f_i,\ i = 1, \ldots, n_T$ over m subsets G_j, $j = 0, \ldots, m-1$, such that

$$\max_j \sum_{f_i \in G_j} w_{f_i} \tag{3.9}$$

is minimized approximately. This is essentially the strategy used in [39]. The next example shows that this may lead to a better load balanced distribution.

Example We consider a problem with 5 independent subtrees $T[f_1], \ldots, T[f_5]$ with weights $(w_{f_1}, \ldots, w_{f_5}) = (10, 20, 6, 6, 6)$. The distribution of formula (3.5) leads to $F_0 = (f_1, f_2)$ and $F_1 = (f_3, f_4, f_5)$, with $W_0 = 30$, $W_1 = 18$, and $H = 30/24 - 1 = 0.25$. An optimal distribution for (3.9) is $G_0 = \{f_1, f_3, f_4\}$ and $G_1 = \{f_2, f_5\}$, with $w_1 + w_3 + w_4 = 22$ and $w_2 + w_5 = 26$. This leads to a much smaller imbalance: $H = 26/24 - 1 = 0.083$. □

For circuit simulation problems there is often a large number of relatively light subtrees. In that case it is easy to find a distribution with a small imbalance, and the repeat-until loop of Algorithm 3.5 is finished after a very modest number of iterations, for instance m. For example, for problem `circuit_1` in Chapter 2 this loop is never repeated and the imbalance is rather small: $H = 0.0096$. For that problem the approach in (3.9) leads to an imbalance $H = 0.00036$. Although Algorithm 3.5 is certainly not optimal, it works sufficiently well in practice, for circuit simulation problems.

Chapter 4

A parallelizable GMRES-type method for p-cyclic matrices

Abstract: In this chapter we present a GMRES-type method for the solution of linear systems with a p-cyclic coefficient matrix. The method has similarities with existing GMRES approaches for p-cyclic matrices, but in contrast to these methods the method is efficiently parallelizable, even if the p-cyclic matrix has a rather small block size. Moreover the method has good stability properties. However, the serial costs of the method may be somewhat higher. Numerical experiments demonstrate the effectiveness of the method.

Keywords: parallel iterative method, p-cyclic matrix, Krylov subspace methods, GMRES, periodic steady state

AMS subject classifications: 65F10, 65L10, 65Y05.

4.1 Introduction

Linear systems of the form

$$\begin{bmatrix} F_1 & 0 & 0 & -E_M \\ -E_1 & F_2 & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -E_{M-1} & F_M \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ \vdots \\ c_M \end{bmatrix} \tag{4.1}$$

often arise in the simulation of periodic phenomena. The E_i and F_i are sparse n by n matrices. Block E_M couples the end of the periodic interval with the start. We will assume that the diagonal blocks F_i are nonsingular; the off-diagonal blocks E_i may be singular. Moreover, we assume that (4.1) is nonsingular. The matrix in (4.1) is called an M-cyclic matrix.

If one y_i is known, the others can be computed by the forward recursion

$$y_j = \begin{cases} {F_j}^{-1}(c_j + E_{j-1}y_{j-1}) & \text{if } j = i+1, \ldots, M\,, \\ {F_j}^{-1}(c_j + E_M y_M) & \text{if } j = 1\,, \\ {F_j}^{-1}(c_j + E_{j-1}y_{j-1}) & \text{if } j = 2, \ldots, i-1\,. \end{cases} \tag{4.2}$$

A backward recursion is in general not possible since E_i may be singular.

Block diagonal scaling of (4.1) with $M = \mathrm{diag}({F_1}^{-1}, \ldots, {F_M}^{-1})$ leads to a linear system that is easier to handle:

$$\begin{bmatrix} I & 0 & 0 & -C_1 \\ -C_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -C_M & I \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} c'_1 \\ \vdots \\ \vdots \\ c'_M \end{bmatrix}, \tag{4.3}$$

with

$$c'_i = {F_i}^{-1} c_i \,, \tag{4.4a}$$

$$C_i = \begin{cases} {F_i}^{-1} E_M & \text{if } i = 1 \,, \\ {F_i}^{-1} E_{i-1} & \text{if } i = 2, \ldots, M \,. \end{cases} \tag{4.4b}$$

The M-cyclic linear system (4.3) can be reduced by block Gaussian elimination to a p-cyclic one, with $p \leq M$. Note that block Gaussian elimination may be unstable since no pivoting is used. In order to eliminate blocks, we define a partition of the integers $1, \ldots, M$:

$$(1, 2, \ldots, M) = (\underline{q}_1, \ldots, \overline{q}_1, \ldots, \ldots, \underline{q}_p, \ldots, \overline{q}_p) \,. \tag{4.5}$$

Clearly $\underline{q}_i \leq \overline{q}_i$, for $i = 1, \ldots p$ and $\overline{q}_i + 1 = \underline{q}_{i+1}$, for $i = 1, \ldots, p-1$. A reduced system is obtained by eliminating the unknowns $y_{\underline{q}_1}, \ldots, y_{\overline{q}_1 - 1},\ y_{\underline{q}_2}, \ldots, y_{\overline{q}_2 - 1},\ \ldots,\ y_{\underline{q}_p}, \ldots, y_{\overline{q}_p - 1}$ from (4.3). Hence, the unknowns $y_{\overline{q}_1},\ y_{\overline{q}_2},\ \ldots,\ y_{\overline{q}_p}$ are not eliminated. The reduced system is

$$Ax = b \,, \text{ with}$$

$$A = \begin{cases} I - B_1 \,, & \text{if } p = 1 \,, \\ \begin{bmatrix} I & 0 & 0 & -B_1 \\ -B_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix}, & \text{if } p \geq 2 \,, \end{cases} \tag{4.6}$$

and

$$B_i = C_{\overline{q}_i} \cdot \ldots \cdot C_{\underline{q}_i},\ i = 1, \ldots, p \,, \tag{4.7}$$

$$x = \left[x_1^T, \ldots, \ldots, x_p^T\right]^T \,,\ b = \left[b_1^T, \ldots, \ldots, b_p^T\right]^T \,.$$

The vectors b_i are defined by the following recursion:

$$w_{\underline{q}_i} = {c'}_{\underline{q}_i} \,, \tag{4.8a}$$

$$w_k = {c'}_k + C_k w_{k-1},\ k = \underline{q}_i + 1, \ldots, \overline{q}_i \,, \tag{4.8b}$$

$$b_i = w_{\overline{q}_i} \,. \tag{4.8c}$$

The x_i and y_j are related via

$$y_{\overline{q}_i} = x_i, \text{ for } i = 1, \ldots, p \,.$$

The other y_j, $j = 1, \ldots, M$, can be computed by the forward recursion (4.2), after the p-cyclic linear system (4.6) has been solved.

There are several applications of p-cyclic matrices, or equivalently M-cyclic matrices. Nonlinear DAEs with periodic boundary conditions can be solved by, for instance, the multiple shooting method or the finite difference method, see [7] and Section 1.3. This leads to linear systems with an M-cyclic structure.

In [82] a parabolic equation with periodic boundary conditions leads to p-cyclic matrices. Another application with the p-cyclic linear system (4.6), although with non-constant block sizes, arises in Markov Chain models, see e.g. [66]. In Section 4.13 of this chapter the numerical methods are applied to p-cyclic matrices arising from periodic steady state simulation of circuits.

Direct methods for the solution of (4.1) are considered in [4] and [7]. These methods require usually $O(n^3M)$ operations (if the sparsity of the E_j and F_j blocks is not exploited) to solve the linear system, which is prohibitive if M or the block size n is large. Parallel direct methods are considered in [4], [6], [32], [92], [93]. The theory of the SOR iterative method for p-cyclic matrices is well developed, see e.g. [28], [44], [45], [87, Ch. 4]. A suitably ordered matrix with property A and with non vanishing diagonal elements is a 2-cyclic matrix of the form (4.1), with $M = 2$, [94, Ch. 2]. The Chebyshev method of Golub and Varga [43] for 2-cyclic matrices roughly halves the amount of work per iteration compared to the standard Chebyshev method applied to the system (4.6), with $p = 2$. Reid [70] has shown that similar gains can be achieved for the conjugate gradients method. A generalization of this result for p-cyclic matrices has been made by Hochbruck [50, Ch. 5]. This generalization is based on the QMR method.

In order to exploit the sparsity of E_i and F_i in (4.1), the matrices B_i are not formed explicitly in practice, when an iterative solver is applied to (4.6). Instead, sparse LU decompositions of the diagonal blocks F_i with partial pivoting, and the formulas (4.4b) and (4.7) are used to compute a matrix-vector product $B_i v$. Therefore, the matrix-vector products of an iterative method for (4.6) are relatively expensive compared with the vector updates, inner products, and other operations of the method.

In Section 4.2 we will discuss existing minimal residual approaches for p-cyclic linear systems. Usually these methods are either not efficiently parallelizable or converge too slowly for the type of problems considered here. Section 4.3 presents a GMRES-type method that is efficiently parallelizable, but the serial costs may be somewhat higher. This method will be called p-cyclic GMRES. In Section 4.5 the convergence of p-cyclic GMRES is analysed for a special case. In Sections 4.6 to 4.9 a few related topics are considered. Costs and efficient parallelization are discussed in Sections 4.10 and 4.11. The chapter finishes with a few numerical experiments.

Independently and simultaneously with our work, Dekker has developed the P-GMRES method [20]. For the special case $p = 2$, our method is equivalent to P-GMRES. For $p = 1$ our method is equivalent to standard GMRES [76].

4.2 Existing minimal residual approaches for p-cyclic matrices

In this section we will discuss minimal residual methods for the iterative solution of the p-cyclic linear system (4.6).

4.2.1 GMRES

The p-cyclic linear system (4.6) can be solved by applying GMRES method [76] to it. Another possibility is to eliminate $x_1, \ldots, x_{p-1}$ from (4.6) using block Gaussian elimination and to apply GMRES to the reduced linear system

$$(I - B_p \cdot \ldots \cdot B_1) x_p = \hat{b}_p \,, \tag{4.9}$$

where $\hat{b}_p$ is defined by the recurrence relation

$$\hat{b}_1 = b_1 \,, \tag{4.10a}$$

$$\hat{b}_j = b_j + B_j \hat{b}_{j-1} \,, \quad \text{for } j = 2, \ldots, p \,. \tag{4.10b}$$

Note that the reduced linear system (4.9) is equal to the linear system obtained by block Gaussian elimination of $y_1, \ldots, y_{M-1}$ in (4.3). Hence, (4.9) is also equal to the p-cyclic linear system (4.6), with $p = 1$. The matrix $I - B_p \cdot \ldots \cdot B_1$ is not explicitly formed in practice, since only matrix-vector products $(I - B_p \cdot \ldots \cdot B_1)v$ are needed by GMRES.

After m GMRES iterations the approximate solution is $x_p^{(m)}$ and the residual is

$$r_p^{(m)} = \hat{b}_p - (I - B_p \cdot \ldots \cdot B_1) x_p^{(m)} \,.$$

The other unknowns $x_1^{(m)}, \ldots, x_{p-1}^{(m)}$ follow from (4.6) by a simple forward recursion:

$$x_i^{(m)} = \begin{cases} b_1 + B_1 x_p^{(m)} \,, & \text{if } i = 1 \,, \\ b_i + B_i x_{i-1}^{(m)} \,, & \text{if } i = 2, \ldots, M \,. \end{cases}$$

Simple calculations show that the residual of the large system is

$$b - A x^{(m)} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hat{b}_p - (I - B_p \cdot \ldots \cdot B_1) x_p^{(m)} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ r_p^{(m)} \end{bmatrix} \,.$$

Hence, the norm of the residual of the p-cyclic linear system (4.6) is equal to the norm of the residual of the reduced system (4.9): $\|b - A x^{(m)}\| = \|r_p^{(m)}\|$.

Unfortunately, GMRES usually converges much worse for (4.6) than for the reduced system (4.9), if $p \neq 1$. The difference in convergence can be explained by the differences between the eigenvalue distribution in the complex plane of the matrix A in (4.6), and the matrix $I - B_p \cdot \ldots \cdot B_1$, as we will see now.

Varga [87, page 107] has shown that the eigenvalues of

$$\widetilde{B} = \begin{bmatrix} 0 & 0 & 0 & B_1 \\ B_2 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & B_p & 0 \end{bmatrix}$$

and

$$\mathcal{L}_1 = \begin{bmatrix} I & 0 & 0 & 0 \\ -B_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix}^{-1} \begin{bmatrix} & B_1 \\ 0 & 0 \\ & 0 \end{bmatrix} = \begin{bmatrix} 0 & B_1 \\ & B_2 B_1 \\ & \vdots \\ & B_p \cdot \ldots \cdot B_2 B_1 \end{bmatrix}, \qquad (4.11)$$

are related to each other: If μ is an eigenvalue of $\widetilde{B}$, then μ^p is an eigenvalue of $\mathcal{L}_1$. If λ is a nonzero eigenvalue of the matrix $\mathcal{L}_1$ and $\mu^p = \lambda$, then μ is an eigenvalue of $\widetilde{B}$. From (4.11) we see a simple relation between the eigenvalues of $\mathcal{L}_1$ and $B_p \cdot \ldots \cdot B_1$: If λ is a nonzero eigenvalue of $\mathcal{L}_1$, then λ is an eigenvalue of $B_p \cdot \ldots \cdot B_1$. If λ is an eigenvalue of $B_p \cdot \ldots \cdot B_1$, then λ is an eigenvalue of $\mathcal{L}_1$.

Now it is obvious how the eigenvalues of $A = I - \widetilde{B}$ are related to the eigenvalues of $I - B_p \cdot \ldots \cdot B_1$. This is illustrated in Figure 4.1 that shows the eigenvalues of a p-cyclic matrix obtained from the periodic steady state simulation of a specific circuit. Clearly, the reduced system has a more favourable eigenvalue distribution for GMRES, because of better clustering of the eigenvalues. In this example the matrix A has block size 22. The condition number of A in the 2-norm is $\kappa(A) \approx 56$, $\kappa(I - B_p \cdot \ldots \cdot B_1) \approx 10$.

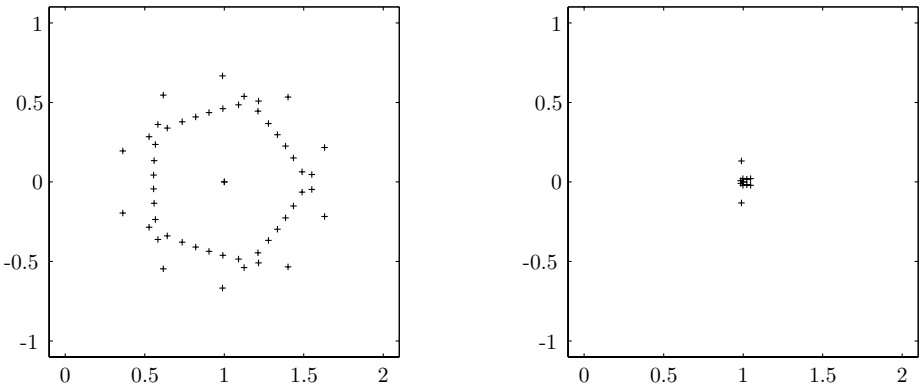


Figure 4.1: Left: eigenvalues of a 5-cyclic matrix A. Right: eigenvalues of the reduced matrix $I - B_5 B_4 B_3 B_2 B_1$.

Note that reducing the linear system $Ax = b$ to $(I - B_p \cdot \ldots \cdot B_1)x_p = \hat{b}_p$ may cause stability problems if there are blocks B_i with large $||B_i||$. This will be discussed in Section 4.13.3.

4.2.2 GMRES for a p-cyclic system with a special initial guess

With a special initial guess considerable savings are possible when applying GMRES to the p-cyclic linear system (4.6). For the initial guess $x^{(0)}$ we take

$$x^{(0)} = \begin{bmatrix} \hat{b}_1 \\ \vdots \\ \hat{b}_{p-1} \\ 0 \end{bmatrix},$$

where $\hat{b}_i$ is defined in (4.10). Now the initial residual is

$$r^{(0)} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hat{b}_p \end{bmatrix}.$$

Since we have an initial guess $x^{(0)}$ we actually deal with the system

$$\begin{bmatrix} I & 0 & 0 & -B_1 \\ -B_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \vdots \\ \vdots \\ \hat{x}_p \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hat{b}_p \end{bmatrix}, \tag{4.12}$$

or $A\hat{x} = r^{(0)}$ and the solution of (4.6) is $x = x^{(0)} + \hat{x}$.

The Krylov subspace associated with (4.12) has a special structure:

$$\mathcal{K}^{pm}(A, r^{(0)}) = \text{span}\Bigg(\begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hat{b}_p \end{bmatrix}, \begin{bmatrix} B_1\hat{b}_p \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ B_2B_1\hat{b}_p \\ 0 \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hat{B}\hat{b}_p \end{bmatrix}, \begin{bmatrix} B_1\hat{B}\hat{b}_p \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ldots, \ldots, \begin{bmatrix} 0 \\ 0 \\ B_{p-1} \cdot \ldots \cdot B_1\hat{B}^{m-1}\hat{b}_p \\ 0 \end{bmatrix} \Bigg), \tag{4.13}$$

with $\hat{B} = B_p \cdot \ldots \cdot B_1$. Now p GMRES iterations are approximately as expensive as one GMRES iteration for the p-cyclic linear system (4.6), if this structure is exploited.

This method for solving $Ax = b$ will be called $x^{(0)}$-GMRES, in the remainder of this chapter. The idea behind $x^{(0)}$-GMRES is widely known, for example, it has been used by Reid [70] for the conjugate gradients method with $p = 2$. Hochbruck [50, Ch. 5] has exploited a similar idea for the QMR method with $p \geq 2$. For this method both the initial residual $r^{(0)}$ and the initial 'shadow residual' $s^{(0)}$ (the vector $s^{(0)}$ is used for generating $\mathcal{K}^{pm}(A^T, s^{(0)})$) have only one nonzero block.

Now we will compare $x^{(0)}$-GMRES for (4.6) with GMRES applied to the reduced system (4.9) for $p \geq 2$. The $x^{(0)}$-GMRES method minimizes the residual norm over a Krylov subspace:

$$\hat{x}^{(pm)} = \underset{\hat{x} \in \mathcal{K}^{pm}(A, r^{(0)})}{\operatorname{argmin}} \|b - A\hat{x}\| .$$

A suboptimal solution for $A\hat{x} = r^{(0)}$ is given by $\hat{\hat{x}}^{(pm)} = N\hat{x}_p^{(pm)}$, with

$$N = \begin{bmatrix} & B_1 & \\ & \vdots & \\ B_{p-1} \cdot \ldots \cdot B_2 B_1 & & \\ & I & \end{bmatrix},$$

and

$$\begin{aligned} \hat{x}_p^{(pm)} &\in \operatorname{span}(\hat{b}_p,\ \hat{B}\hat{b}_p,\ \ldots,\ \hat{B}^{m-1}\hat{b}_p) \\ &= \mathcal{K}^m(I - B_p \cdot \ldots \cdot B_1, \hat{b}_p) . \end{aligned} \tag{4.14}$$

Note that the Krylov subspace (4.14) is equal to the Krylov subspace associated with the reduced system (4.9). The suboptimal solution $\hat{\hat{x}}^{(pm)}$ leads to a residual $\hat{\hat{r}}_{x^{(0)}}^{(pm)}$ with $p-1$ zero blocks:

$$\hat{\hat{r}}_{x^{(0)}}^{(pm)} = r^{(0)} - AN\hat{x}_p^{(pm)} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hat{b}_p - (I - \hat{B})\hat{x}_p^{(pm)} \end{bmatrix} .$$

The norm of the residual after pm iterations of $x^{(0)}$-GMRES is

$$\begin{aligned} \|r_{x^{(0)}}^{(pm)}\| &= \min_{\hat{x} \in \mathcal{K}^{pm}(A, r^{(0)})} \|r^{(0)} - A\hat{x}\| && \text{(4.15a)} \\ &\leq \min_{\hat{\hat{x}}_p \in \mathcal{K}^m(I - B_p \cdot \ldots \cdot B_1, \hat{b}_p)} \|r^{(0)} - AN\hat{\hat{x}}_p\| && \text{(4.15b)} \\ &= \min_{x_p \in \mathcal{K}^m(I - B_p \cdot \ldots \cdot B_1, \hat{b}_p)} \|\hat{b}_p - (I - B_p \cdot \ldots \cdot B_1)x_p\| && \text{(4.15c)} \\ &= \|r_{\text{red}}^{(m)}\| . && \text{(4.15d)} \end{aligned}$$

Here $r_{\text{red}}^{(m)}$ is the residual of GMRES applied to the reduced system (4.9), after m iterations. Inequality (4.15b) follows from the suboptimality of $\hat{\hat{x}}^{(pm)}$.

In the following, we will derive an upper bound for $\|r_{\text{red}}^{(m)}\|$ in terms of $r_{x^{(0)}}^{(pm)}$. After pm iterations of $x^{(0)}$-GMRES, the residual is

$$r_{x^{(0)}}^{(pm)} = \begin{bmatrix} r_1^{(pm)} \\ \vdots \\ \vdots \\ r_p^{(pm)} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hat{b}_p \end{bmatrix} - \begin{bmatrix} I & 0 & 0 & -B_1 \\ -B_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix} \begin{bmatrix} \hat{x}_1^{(pm)} \\ \vdots \\ \vdots \\ \hat{x}_p^{(pm)} \end{bmatrix} . \tag{4.16}$$

Premultiplying (4.16) with the 1 by p block matrix M, defined by

$$M = [B_p \cdot \ldots \cdot B_2 \quad B_p \cdot \ldots \cdot B_3 \quad \ldots \quad B_p \quad I] \ ,$$

gives

$$\begin{aligned} M r^{(pm)}_{x^{(0)}} &= \hat{b}_p - [\ 0 \quad \ldots \quad 0 \quad I - B_p \cdot \ldots \cdot B_1\ ]\, \hat{x}^{(pm)} \\ &= \hat{b}_p - (I - B_p \cdot \ldots \cdot B_1)\hat{x}^{(pm)}_p \ , \end{aligned}$$

and this will be used in formula (4.17b). The $\hat{x}^{(pm)}_p$ found by $x^{(0)}$-GMRES, see formula (4.16), is a suboptimal solution for the reduced linear system (4.9):

$$\begin{aligned} \|r^{(m)}_{\text{red}}\| &= \min_{x_p \in \mathcal{K}^m(I - B_p \cdot \ldots \cdot B_1, \hat{b}_p)} \|\hat{b}_p - (I - B_p \cdot \ldots \cdot B_1)x_p\| && \text{(4.17a)} \\ &\leq \|\hat{b}_p - (I - B_p \cdot \ldots \cdot B_1)\hat{x}^{(pm)}_p\| && \text{(4.17b)} \\ &= \|M r^{(pm)}_{x^{(0)}}\| && \text{(4.17c)} \\ &\leq \|M\|\ \|r^{(pm)}_{x^{(0)}}\| \ . && \text{(4.17d)} \end{aligned}$$

Combining (4.15) and (4.17) shows that asymptotically there is no big advantage of $x^{(0)}$-GMRES over GMRES applied to the reduced system (4.9), if $\|M\|$ is sufficiently small:

$$\|r^{(pm)}_{x^{(0)}}\| \leq \|r^{(m)}_{\text{red}}\| \leq \|M\|\ \|r^{(pm)}_{x^{(0)}}\| \ .$$

The value of $\|M\|$ is often small in practical applications. If ρ is defined by $\rho = \max_{2 \leq i \leq p} \|B_p \cdot \ldots \cdot B_i\|$, then it is easy to show that $\|M\| \leq \rho\sqrt{p}$.

Numerical experiments (not reported) indicate that in inexact arithmetic there seems to be no significant advantage of $x^{(0)}$-GMRES over GMRES applied to the reduced system (4.9), see also the numerical experiment in Section 4.13.3. Note that this is different in the QMR case, at least for the two numerical examples considered in [50, Ch. 5]. These examples deal with the computation of a unique (up to a scaling factor) non trivial solution of a singular linear system $Ax = 0$. For both examples QMR applied to the reduced system has a stagnation phase in the convergence, while '$x^{(0)}$-QMR' has not.

Note that $x^{(0)}$-GMRES computes $x_1, \ldots, x_p$, while GMRES applied to the reduced system (4.9) needs a forward recursion in order to compute $x_1, \ldots, x_{p-1}$ after x_p has been solved from (4.9). However, the orthogonalization costs of $x^{(0)}$-GMRES are approximately p times higher, in comparison with GMRES applied to (4.9). The costs of both methods are discussed in Section 4.10.

4.3 p-cyclic GMRES

The $x^{(0)}$-GMRES method and GMRES applied to the reduced system (4.9) offer good convergence if both $||B_i||$ is sufficiently small and B_i has most of its of eigenvalues close to zero. However, in practice the matrix-vector products $B_i v$ are often hard to parallelize since B_i is usually not explicitly known, but defined in a sequential way by (4.4b) and

(4.7). In practice E_i and F_i are often too small for parallelization on a matrix level. In contrast to these methods, applying GMRES directly to the p-cyclic linear system (4.6) is efficiently parallelizable because the p matrix-vector products of the form $B_i v_i$, $i = 1, \ldots, p$, are independent of each other. Unfortunately GMRES converges relatively slowly for (4.6), and even in a parallel environment there is no gain for (4.6), if $p \neq 1$.

The p-cyclic GMRES method, to be presented in this section, is efficiently parallelizable. Moreover it suffers less from instability problems if $||B_i||$ is large. The convergence of p-cyclic GMRES is often slightly poorer than GMRES applied to the reduced linear system (4.9). The p-cyclic GMRES method is somewhat in the same way related to GMRES on a 1 by 1 block reduced system (see previous section) as the multiple shooting method for boundary value problems is related to the single shooting method [7]. Note that the multiple shooting method has better parallelization and stability properties than the single shooting method.

The GMRES method applied to the p-cyclic linear system (4.6) finds an approximate solution $x^{(m)}$ in the Krylov subspace

$$\mathcal{K}^m = \text{span}(\begin{bmatrix} w_1^{(1)} \\ \vdots \\ w_p^{(1)} \end{bmatrix}, \begin{bmatrix} w_1^{(2)} \\ \vdots \\ w_p^{(2)} \end{bmatrix}, \ldots, \begin{bmatrix} w_1^{(m)} \\ \vdots \\ w_p^{(m)} \end{bmatrix}) , \tag{4.18}$$

with

$$x^{(m)} = \underset{x \in \mathcal{K}^m}{\text{argmin}} \|b - Ax\| .$$

The $w_i^{(j)}$ are recursively defined by

$$w_i^{(j)} = \begin{cases} b_i & \text{if } j = 1,\ i = 1, \ldots, p , \\ B_i w_{\widetilde{i-1}}^{(j-1)} & \text{if } j = 2, \ldots, m ,\ i = 1, \ldots, p . \end{cases} \tag{4.19}$$

Here we used the notion $\widetilde{-}1$. The definition of $\widetilde{-}1$ within the context of p-cyclic matrices is:

$$i\widetilde{-}1 \equiv \begin{cases} i - 1 , & \text{if } i = 2, \ldots, p , \\ p , & \text{if } i = 1 . \end{cases}$$

For example, with $p = 4$ and $m = 4$, the Krylov subspace is

$$\mathcal{K}^4 = \text{span}(\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}, \begin{bmatrix} B_1 b_4 \\ B_2 b_1 \\ B_3 b_2 \\ B_4 b_3 \end{bmatrix}, \begin{bmatrix} B_1 B_4 b_3 \\ B_2 B_1 b_4 \\ B_3 B_2 b_1 \\ B_4 B_3 b_2 \end{bmatrix}, \begin{bmatrix} B_1 B_4 B_3 b_2 \\ B_2 B_1 B_4 b_3 \\ B_3 B_2 B_1 b_4 \\ B_4 B_3 B_2 b_1 \end{bmatrix}) .$$

The idea of the p-cyclic GMRES method is to decouple the Krylov subspace (4.18) into p independent subspaces $\mathcal{K}_i^m$, each with dimension less than or equal to m:

$$\mathcal{K}_i^m = \text{span}(w_i^{(1)}, w_i^{(2)}, \ldots, w_i^{(m)}) ,\ i = 1, \ldots, p .$$

We seek an approximate solution $x^{(m)}$ of the p-cyclic linear system (4.6) in the search space

$$\hat{\mathcal{K}}^m = \{x^{(m)} = \begin{bmatrix} x_1^{(m)} \\ \vdots \\ x_p^{(m)} \end{bmatrix} | x_i^{(m)} \in \mathcal{K}_i^m\} .$$

For example, if $p = 4$, then

$$x_2^{(4)} \in \mathcal{K}_2^4 = \text{span}(b_2, B_2b_1, B_2B_1b_4, B_2B_1B_4b_3) .$$

The approximate solution $x^{(m)} \in \hat{\mathcal{K}}^m$ is chosen such that the norm of the residual $\|b - Ax^{(m)}\|$ is minimal. In the final form of the p-cyclic GMRES method we will alter this definition of $x^{(m)}$ slightly, see Section 4.3.2.

It is easily seen that

$$\mathcal{K}^m \subset \hat{\mathcal{K}}^m ,$$

and we have $\dim(\hat{\mathcal{K}}^m) \leq mp$. Usually the dimension of $\hat{\mathcal{K}}^m$ is p times larger than the dimension of $\mathcal{K}^m$ and therefore we hope that p-cyclic GMRES achieves a much faster convergence than GMRES does. The following relation holds for the norm of the residual:

$$\|r^{(m),\, p\text{-cyclic GMRES}}\| \leq \|r^{(m),\, \text{GMRES}}\| , \tag{4.20}$$

since $\mathcal{K}^m \subset \hat{\mathcal{K}}^m$.

4.3.1 Orthogonal basis for $\hat{\mathcal{K}}^m$

An orthogonal basis for $\hat{\mathcal{K}}^m$ is desirable, for a practical and accurate method. Fortunately, we only need an orthogonal basis for each $\mathcal{K}_i^m$, $i = 1, \ldots, p$, which immediately leads to an orthogonal basis for $\hat{\mathcal{K}}^m$. Orthogonalization of the $w_i^{(j)}$ vectors defined in (4.19), leads to an orthogonal basis for $\hat{\mathcal{K}}^m$:

$$s_i^{(j)} = \begin{cases} b_i^{(1)} & \text{if } j = 1,\ i = 1, \ldots, p , \\ B_i v_{i-1}^{(j-1)} & \text{if } j = 2, \ldots, m ,\ i = 1, \ldots, p , \end{cases} \tag{4.21a}$$

$$\hat{s}_i^{(j)} = \text{ orthogonalize } s_i^{(j)} \text{ with respect to } \{v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(j-1)}\} , \tag{4.21b}$$

$$v_i^{(j)} = \hat{s}_i^{(j)} / \|\hat{s}_i^{(j)}\| . \tag{4.21c}$$

Step 1 and step 2 of Algorithm 4.1 construct an orthogonal basis in an Arnoldi-like way. Therefore, we will call these two steps p-cyclic Arnoldi. For $p = 1$, step 1 and 2 of Algorithm 4.1 reduce to the standard Arnoldi method. In step 3 the approximate solution with minimal residual norm is computed. For $p = 1$, Algorithm 4.1 reduces to standard GMRES with Householder orthogonalization, if the least-squares problem is replaced by $\min_{y_1} \|(\widetilde{I} - H_1^{(m)})y_1 - \delta_1 e_1\|$.

Classical or modified Gram-Schmidt is less suitable for orthogonalization of the vector $B_i\hat{v}_i^{(j)} = B_i v_{i-1}^{(j)}$ with respect to $v_i^{(1)}$, $\ldots$, $v_i^{(j)}$, since $B_i v_{i-1}^{(j)}$ may be exactly or nearly in $\text{span}(v_i^{(1)}, \ldots, v_i^{(j)})$. This leads to an exact break down of the Gram-Schmidt process,

Algorithm 4.1: p-cyclic GMRES

1. Initialization:
   Choose a random nonzero vector w
   for $i = 1 : p$
      Define $H_i^{(m)} = \text{zeros}(m+1, m)$
      if $\|b_i\| \neq 0$
         $v_i^{(1)} = b_i$
      else (in practice usually $\|b_i\| \neq 0$)
         $v_i^{(1)} = w$
      end
      $[v_i^{(1)}, \text{dummy}, u_i^{(1)}] = \text{house\_orth}(v_i^{(1)}, [\,])$
      $U_i^{(1)} = u_i^{(1)}$
      $V_i^{(1)} = v_i^{(1)}$
      $\delta_i = {v_i^{(1)}}^T b_i$
   end
2. Iterate:
   for $j = 1 : m$
      $$\begin{bmatrix} \hat{v}_1^{(j)} \\ \hat{v}_2^{(j)} \\ \vdots \\ \hat{v}_p^{(j)} \end{bmatrix} = \begin{bmatrix} v_p^{(j)} \\ v_1^{(j)} \\ \vdots \\ v_{p-1}^{(j)} \end{bmatrix} \qquad (*)$$
      for $i = 1 : p$ (parallelizable for loop)
         $[v_i^{(j+1)}, H_i^{(m)}(1{:}j+1, j), u_i^{(j+1)}] = \text{house\_orth}(B_i \hat{v}_i^{(j)}, U_i^{(j)})$
         $U_i^{(j+1)} = [U_i^{(j)} \;\; u_i^{(j+1)}]$
         $V_i^{(j+1)} = [V_i^{(j)} \;\; v_i^{(j+1)}]$
      end
   end
3. Finish:
   Solve the least-squares problem:
   $$\min_{y_1,\ldots,y_p} \left\| \begin{bmatrix} \widetilde{I} & 0 & 0 & -H_1^{(m)} \\ -H_2^{(m)} & \widetilde{I} & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -H_p^{(m)} & \widetilde{I} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ \vdots \\ y_p \end{bmatrix} - \begin{bmatrix} \delta_1 e_1 \\ \vdots \\ \vdots \\ \delta_p e_1 \end{bmatrix} \right\|$$
   $$x^{(m)} = \begin{bmatrix} V_1^{(m)} y_1 \\ \vdots \\ V_p^{(m)} y_p \end{bmatrix}$$

or to instabilities in the Gram-Schmidt process. We may expect more stability problems for p-cyclic GMRES than for standard GMRES, if classical or modified Gram-Schmidt is used for orthogonalization. This can be explained as follows: A (near) break down in the Gram-Schmidt process is often no problem for standard GMRES since it has (nearly) converged in that case. However, this is not necessarily true for p-cyclic GMRES, as we will see in an example on page 54.

Instead of Gram-Schmidt orthogonalization we use Householder orthogonalization , see e.g. [42, Ch. 5], which is known to be more stable, but twice as expensive. Other stable orthogonalization methods, which are twice expensive as standard Gram-Schmidt methods, are iterated classical or modified Gram-Schmidt. For parallelization of the orthogonalization process iterated classical Gram-Schmidt might be preferred above Householder orthogonalization. However, we aim at parallelizability of Algorithm 4.1 at the block level.

The function

$$[v_i^{(j+1)}, H_i^{(m)}(1{:}j+1, j), u_i^{(j+1)}] = \text{house\_orth}(B_i v_{i-1}^{(j)}, U_i^{(j)})$$

of Algorithm 4.1, computes $v_i^{(j+1)}$ and $H_i^{(m)}(1{:}j+1, j)$ such that

$$B_i v_{i-1}^{(j)} = [V_i^{(j)} \;\; v_i^{(j+1)}] H_i^{(m)}(1{:}j+1, j)\,, \quad \text{and} \tag{4.22a}$$

$$I = [V_i^{(j)} \;\; v_i^{(j+1)}]^T [V_i^{(j)} \;\; v_i^{(j+1)}]\,. \tag{4.22b}$$

The Householder vectors are stored in $U_i^{(j+1)}$. The Householder method leads also to expansion of $V_i^{(j)}$ in the special case that

$$B_i v_{i-1}^{(j)} \in \text{span}(v_i^{(1)}, \ \ldots, \ v_i^{(j)}) \;\; \text{or} \; B_i v_{i-1}^{(j)} = 0\,.$$

In these cases $\{H_i^{(m)}\}_{j+1,j} = 0$ or $H_i^{(m)}(1{:}j+1, j) = 0$, respectively, but the Householder vector $u_i^{(j+1)}$ is still well defined and $v_i^{(j+1)}$ is still orthogonal to $V_i^{(j)}$. However, then

$$\text{span}(v_i^{(1)}, \ \ldots, \ v_i^{(j)}, \ B_i v_{i-1}^{(j)}) \neq \text{span}(v_i^{(1)}, \ \ldots, \ v_i^{(j)})\,.$$

This is different from a Gram-Schmidt method that would break down in this situation because, in the final step of the Gram-Schmidt process, there would be a normalization of a zero vector. It is possible to solve this problem by replacing this zero vector by an arbitrary vector which is orthogonal to the previous vectors, but the Householder approach is a more neat solution.

In an actual implementation of Algorithm 4.1 it is not necessary to store both $V_i^{(j)}$ and $U_i^{(j)}$. In Algorithm 4.1 the column vectors $V_i^{(j)}$ are needed only once in step 3, in order to compute $x_i^{(m)}$. In the same way as in standard GMRES with Householder orthogonalization, see [73, Ch. 6] and [91], $x_i^{(m)}$ can be computed by using $U_i^{(j)}$ instead of $V_i^{(j)}$. So, $V_i^{(j)}$ need not be stored and the vector storage requirements are mainly determined by the matrix with Householder vectors $U_i^{(j)}$.

Equation (4.22a) can be written in matrix form:

$$B_i V_{\widetilde{i-1}}^{(m)} = V_i^{(m+1)} H_i^{(m)} \,. \tag{4.23}$$

Here $H_i^{(m)}$ is a $m+1$ by m Hessenberg matrix. This leads to the following block matrix relation:

$$\begin{bmatrix} I & 0 & 0 & -B_1 \\ -B_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix} \begin{bmatrix} V_1^{(m)} & 0 & \dots & 0 \\ 0 & V_2^{(m)} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & V_p^{(m)} \end{bmatrix} = \begin{bmatrix} V_1^{(m+1)} & 0 & \dots & 0 \\ 0 & V_2^{(m+1)} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & V_p^{(m+1)} \end{bmatrix} \begin{bmatrix} \widetilde{I} & 0 & 0 & -H_1^{(m)} \\ -H_2^{(m)} & \widetilde{I} & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -H_p^{(m)} & \widetilde{I} \end{bmatrix} , \tag{4.24a}$$

or more compactly

$$A V^{(m)} = V^{(m+1)} H^{(m)} \,, \tag{4.24b}$$

with the $m+1$ by m matrix defined by

$$\widetilde{I} \equiv \begin{bmatrix} & I & \\ 0 & \dots & 0 \end{bmatrix} . \tag{4.25}$$

The subspace $\hat{\mathcal{K}}^m$ is not necessarily equal to the range$(V^{(m)})$:

Theorem 4.1 *The matrix $V^{(m)}$ generated by Algorithm 4.1 satisfies the following properties:*

1. $\hat{\mathcal{K}}^m \subset \text{range}(V^{(m)})$.
2. $\hat{\mathcal{K}}^m = \text{range}(V^{(m)})$ *if and only if* $\{H_i^{(m)}\}_{j+1,j} \neq 0,\ j = 1, \dots, m-1,\ i = 1, \dots, p$ *and* $b_i \neq 0,\ i = 1, \dots, p$.

Proof Part 1 will be proved by induction on k, $k = 1, \dots, m-1$. For $k = 1$, we have

$$\mathcal{K}_i^1 = \text{span}(b_i) = \text{range}(V_i^{(1)}),\ i = 1, \dots, p \,,$$

if $b_i \neq 0$, and $\mathcal{K}_i^1 = \{0\} \subset \text{range}(V_i^{(1)})$ if $b_i = 0$. Now, suppose

$$\mathcal{K}_i^k = \text{span}(w_i^{(1)}, \dots, w_i^{(k)}) \subset \text{range}(V_i^{(k)}) = \text{span}(v_i^{(1)}, \dots, v_i^{(k)}) \,, \tag{4.26}$$

for $i = 1, \dots, p$. We will prove that

$$\mathcal{K}_i^{k+1} \subset \text{range}(V_i^{(k+1)}) \,.$$

From (4.26) it follows that

$$w_i^{(k+1)} = B_i w_{\widetilde{i-1}}^{(k)} \in \text{span}(B_i v_{\widetilde{i-1}}^{(1)}, \dots, B_i v_{\widetilde{i-1}}^{(k)}) \,.$$

Each vector $B_i v_{i-1}^{(j)}, j = 1, \ldots, k$ is in range$(V_i^{(j+1)})$ by construction (cf. (4.22a)), so

$$\text{span}(B_i v_{i-1}^{(1)}, \ldots, B_i v_{i-1}^{(k)}) \subset \text{range}(V_i^{(k+1)}) \,,$$

and $w_i^{(k+1)} \in \text{range}(V_i^{(k+1)})$. We conclude that $w_i^{(j)} \in \text{range}(V_i^{(j)})$, for $j = 1, \ldots, k+1$, and

$$\mathcal{K}_i^{k+1} = \text{span}(w_i^{(1)}, \ldots, w_i^{(k+1)}) \subset \text{range}(V_i^{(k+1)}) \,,$$

which completes the induction step. So $\mathcal{K}_i^m \subset \text{range}(V_i^{(m)})$ and therefore we have that $\hat{\mathcal{K}}^m \subset \text{range}(V^{(m)})$.

For part 2, take $\hat{\mathcal{K}}^m = \text{range}(V^{(m)})$ and suppose that there is an i with $b_i = 0$. Then $w_i^{(1)} = 0$, but $v_i^{(1)} \neq 0$, so $\dim(\mathcal{K}_i^m) \leq m - 1 < \dim(\text{range}(V_i^{(m)})) = m)$. This implies that $\mathcal{K}_i^m \neq \text{range}(V_i^{(m)})$ and therefore $\hat{\mathcal{K}}^m \neq \text{range}(V^{(m)})$, which is a contradiction, so $b_i \neq 0$ for $i = 1, \ldots, p$. Now suppose that $\hat{\mathcal{K}}^m = \text{range}(V^{(m)})$ and that there are i and k such that $\{H_i^{(m)}\}_{k+1,k} = 0$. We already have (see part 1)

$$w_i^{(k+1)} \in \text{span}(B_i v_{i-1}^{(1)}, \ldots, B_i v_{i-1}^{(k)}) \,.$$

Again the vectors $B_i v_{i-1}^{(j)}, j = 1, \ldots, k-1$ are in the range$(V_i^{(j+1)})$, but $B_i v_{i-1}^{(k)} \in \text{range}(V_i^{(k)})$ since $\{H_i^{(m)}\}_{k+1,k} = 0$, see (4.22a). Hence, $\text{span}(B_i v_{i-1}^{(1)}, \ldots, B_i v_{i-1}^{(k)}) \subset \text{range}(V_i^{(k)})$, which implies that $w_i^{(k+1)} \in \text{range}(V_i^{(k)})$ and $\mathcal{K}_i^{k+1} \subset \text{range}(V_i^{(k)})$. The $k+1$ vectors $w_i^{(1)}, \ldots, w_i^{(k+1)}$ are not linearly independent because they span $\mathcal{K}_i^{k+1}$, which is a subspace of the k dimensional subspace range$(V_i^{(k)})$. Obviously, $\mathcal{K}_i^m \neq \text{range}(V_i^{(m)})$ and therefore $\hat{\mathcal{K}}^m \neq \text{range}(V^{(m)})$, and we conclude that $\{H_i^{(m)}\}_{k+1,k} \neq 0$ if $\hat{\mathcal{K}}^m = \text{range}(V^{(m)})$.

Finally we will prove that $\hat{\mathcal{K}}^m = \text{range}(V^{(m)})$ if $\{H_i^{(m)}\}_{j+1,j} \neq 0$, $j = 1, \ldots, m-1$ and $b_i \neq 0$. Because of part 1 we only have to show that range$(V^{(m)}) \subset \hat{\mathcal{K}}^m$, and this is equivalent to range$(V_i^{(m)}) \subset \mathcal{K}_i^m$. This can be shown by induction on k, $k = 1, \ldots, m-1$. Clearly range$(V_i^{(1)}) \subset \mathcal{K}_i^1$. Suppose that range$(V_i^{(k)}) \subset \mathcal{K}_i^k$, then from (4.22a) it follows that

$$v_i^{(k+1)} = \frac{B_i v_{i-1}^{(k)} - V_i^{(k)} H_i^{(m)}(1{:}k, k)}{\{H_i^{(m)}\}_{k+1,k}} \,. \tag{4.27}$$

Here $V_i^{(k)} H_i^{(m)}(1{:}k, k) \in \mathcal{K}_i^k$; moreover

$$v_{i-1}^{(k)} \in \mathcal{K}_{i-1}^k = \text{span}(w_{i-1}^{(1)}, \ldots, w_{i-1}^{(k)}) \,.$$

This leads to

$$B_i v_{i-1}^{(k)} \in \text{span}(B_i w_{i-1}^{(1)}, \ldots, B_i w_{i-1}^{(k)}) = \text{span}(w_i^{(2)}, \ldots, w_i^{(k+1)}) \subset \mathcal{K}_i^{k+1} \,,$$

and from (4.27) it follows that $v_i^{(k+1)} \in \mathcal{K}_i^{k+1}$. Now it is easily seen that range$(V_i^{(k+1)}) \subset \mathcal{K}_i^{k+1}$, and we conclude that $\hat{\mathcal{K}}^m = \text{range}(V^{(m)})$. □

4.3.2 Computing the approximate solution

In step 3 of Algorithm 4.1, the approximate solution of $Ax = b$ is computed in a similar way as in GMRES. So, we seek an approximate solution $x^{(m)}$ that solves the least-squares problem

$$\min_{x^{(m)} \in \text{range}(V^{(m)})} \|b - Ax^{(m)}\| .$$

Since $x^{(m)} \in \text{range}(V^{(m)})$, we may substitute

$$x^{(m)} = V^{(m)} y .$$

This simplifies the least-squares problem:

$$\|b - Ax^{(m)}\| = \|b - AV^{(m)} y\| \tag{4.28a}$$

$$= \|b - V^{(m+1)} H^{(m)} y\| \equiv J(y) , \tag{4.28b}$$

here we have used relation (4.24b). By construction (Algorithm 4.1) we have that $b_i = \delta_i v_i^{(1)}$, $i = 1, \ldots, p$, with $\delta_i = {v_i^{(1)}}^T b_i$, and b_i is orthogonal to $v_i^{(j)}$ for $j \geq 2$. Hence b_i can be written as

$$b_i = V_i^{(m+1)} \delta_i e_1 ,$$

with $e_1 = [1, 0, \ldots, 0]^T$. This leads to the following expression for b:

$$b = \begin{bmatrix} V_1^{(m+1)} & & \\ & \ddots & \\ & & V_p^{(m+1)} \end{bmatrix} \begin{bmatrix} \delta_1 e_1 \\ \vdots \\ \delta_p e_1 \end{bmatrix} = V^{(m+1)} d , \text{ with } d = \begin{bmatrix} \delta_1 e_1 \\ \vdots \\ \delta_p e_1 \end{bmatrix}$$

Now the function $J(y)$ in (4.28b), which has to be minimized, becomes

$$J(y) = \|V^{(m+1)} (\, d \, - \, H^{(m)} y \,)\| \tag{4.29a}$$

$$= \|d \, - \, H^{(m)} y\| . \tag{4.29b}$$

Here we have used that $V^{(m+1)}$ is an orthogonal matrix, because the matrices $V_i^{(m+1)}$, $i = 1, \ldots, p$ are orthogonal. The above results are exploited in step 3 of Algorithm 4.1, with y replaced by $y = \left[y_1^T, \ldots, y_p^T\right]^T$. Methods to solve this least-squares problem of size $mp \times (m+1)p$, will be discussed in Section 4.4.

For $p = 1$ the linear system $(I - B_1)x_1 = b_1$ reduces to the least-squares problem $\min_y \|e_1 - (\widetilde{I} - H_1)y\|$. In this case p-cyclic GMRES is nearly equivalent to GMRES.

The following theorem is useful in order to solve the least-squares problem:

Theorem 4.2 *The matrix $H^{(m)}$, defined in (4.24a) and (4.24b), has full rank.*

Proof Suppose that $H^{(m)}$ does not have full rank, then there is a $y \neq 0$ such that $H^{(m)} y = 0$. In that case $AV^{(m)} y = V^{(m+1)} H^{(m)} y = 0$. The vector $w = V^{(m)} y$ is nonzero since $V^{(m)}$ is of full rank ($V^{(m)}$ is orthogonal). But $Aw = 0$, and this is a contradiction because A is nonsingular, so $H^{(m)}$ has full rank. □

Theorem 4.2 implies that the least-squares problem has a unique solution. An immediate consequence is that p-cyclic GMRES cannot break down if $m \leq n$. So, even after convergence, p-cyclic GMRES does not break down. The full rank property of $H^{(m)}$ leads to the following theorem:

Theorem 4.3 *The p-cyclic GMRES algorithm has converged if $\{H_i^{(m)}\}_{m+1,m} = 0$ for all $i \in \{1, \ldots, p\}$.*

Proof The rows $m+1$, $2(m+1)$, $\ldots$, $p(m+1)$ of $H^{(m)}$ are zero if $\{H_i^{(m)}\}_{m+1,m} = 0$ for all $i \in \{1, \ldots, p\}$. The $m+1$-st entry of the vector $\delta_i e_1$ is also zero. Therefore, the solution of the least-squares problem (4.29) does not change if we delete the (zero) rows $m+1$, $2(m+1)$, $\ldots$, $p(m+1)$. Now the least-squares problem is reduced to a linear system of full rank which has a unique solution since $H^{(m)}$ has full rank. So, the minimum of $J(y)$ in (4.29) is zero and the p-cyclic GMRES algorithm has converged. □

The converse of Theorem 4.3 is not true in general. This is shown by the next example:

Example In this example 2-cyclic GMRES converges in two steps ($m = 2$), but nevertheless $\{H_1^{(m)}\}_{m+1,m} \neq 0$:

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 & 1 & 3 \\ 0 & 1 & 0 & 1 & 3 & 0 \\ 0 & 0 & 1 & 0 & 4 & 1 \\ 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

The solution of the nonsingular linear system $Ax = b$ is $x = 1/3\,[1, -1, 0, 1, 0, 0]^T$. Algorithm 4.1 generates the matrices

$$V_1^{(3)} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \; V_2^{(3)} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; H_1^{(2)} = \begin{bmatrix} -2 & 1 \\ -1 & 3 \\ 0 & 4 \end{bmatrix}, \; H_2^{(2)} = \begin{bmatrix} -2 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

There is convergence after two steps since

$$x \in \text{range}(V^{(2)}) = \text{range} \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

After one step $\{H_1^{(1)}\}_{2,1} = 0$, but this is not sufficient for convergence. Note that $\{H_1^{(1)}\}_{2,1} = 0$ does not imply that one block has converged: After one iteration both block 1 and 2 of $Ax = b$ have a nonzero residual. □

Theorem 4.3 can be used to show that p-cyclic GMRES converges fast if the matrices B_i have low rank:

Theorem 4.4 *Let* $\mathrm{rank}(B_i) \leq k$ *and* $\{H_i^{(k)}\}_{j+1,j} \neq 0$ *for* $i = 1, \ldots, p$ *and* $j = 1, \ldots, k$, *then p-cyclic GMRES converges in at most* $k+1$ *iterations.*

Proof Premultiplying (4.23) with ${V_i^{(m+1)}}^T$ and substituting $m = k+1$ yields

$$H_i^{(k+1)} = {V_i^{(k+2)}}^T B_i V_{\tilde{i-1}}^{(k+1)} .$$

For general matrices M and N with matching row and column dimensions, the relations $\mathrm{rank}(MN) \leq \mathrm{rank}(M)$ and $\mathrm{rank}(MN) \leq \mathrm{rank}(N)$ hold. So, for the matrix $H_i^{(k+1)}$ we have

$$\mathrm{rank}(H_i^{(k+1)}) = \mathrm{rank}({V_i^{(k+2)}}^T B_i V_{\tilde{i-1}}^{(k+1)}) \leq \mathrm{rank}(B_i) . \tag{4.30}$$

In the $k+1$-st iteration of p-cyclic GMRES $\{H_i^{(k+1)}\}_{k+2,k+1}$ has to be zero, otherwise $\mathrm{rank}(H_i^{(k+1)}) = k+1$, (this is easily seen because $H_i^{(k+1)}$ is a Hessenberg matrix) which is not possible because of (4.30). Now we use Theorem 4.3 to show that p-cyclic GMRES has converged. □

Krylov subspace methods, like BiCGSTAB, GMRES, and QMR, offer the possibility of taking advantage of a suitably chosen initial guess. Of course, we can do the same for the methods discussed in this section and in Section 4.2: Suppose the initial guess is $\bar{x}$ and the correction is x^c, then solve $Ax^c = \bar{b} \equiv b - A\bar{x}$ with p-cyclic GMRES or a method of Section 4.2 by replacing b by $\bar{b}$ in all formulas. The solution of the p-cyclic linear system (4.6) is $x = x^c + \bar{x}$.

4.3.3 The periodic Schur form

In this section we will discuss the periodic Schur form because of its relation to p-cyclic matrices and p-cyclic Arnoldi. The standard Arnoldi iteration [74, Ch. 6], [89, Ch. 7], and variants of it, e.g., [80], are popular methods for computing a partial Schur form of a general square matrix B: $BQ_k = Q_k R_k$, here Q_k is an unitary n by k matrix and R_k is an upper triangular k by k matrix. The columns of Q_k are the Schur vectors and the diagonal elements of R_k are eigenvalues of B.

Bojanczyk, Golub, and Van Dooren introduced the periodic Schur decomposition in [12], see also [81]. The periodic Schur decomposition can be formulated as follows:

Let the matrices B_i, $i = 1, \ldots, p$ be given. Then there exist unitary matrices Q_i and upper triangular matrices R_i, $i = 1, \ldots, p$, such that

$$\begin{bmatrix} I & 0 & 0 & -B_1 \\ -B_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix} \begin{bmatrix} Q_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & Q_p \end{bmatrix} = \begin{bmatrix} Q_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & Q_p \end{bmatrix} \begin{bmatrix} I & 0 & 0 & -R_1 \\ -R_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -R_p & I \end{bmatrix} , \tag{4.31}$$

moreover $Q_p^H B_p \cdot \ldots \cdot B_1 Q_p = R_p \cdot \ldots \cdot R_1$ is upper triangular and the Q_i can be chosen so that the diagonal elements (eigenvalues) of $R_p \cdot \ldots \cdot R_1$ appear in any desired order.

A real matrix variant of the periodic Schur form is also possible. Then R_i is block upper triangular with 1 by 1 or 2 by 2 blocks on the diagonal. In the relation (4.31) we recognize a more or less similar structure to (4.24a). The periodic QR algorithm [12] is a direct method for computing this periodic Schur decomposition. A partial periodic Schur form of order k can be defined as (4.31) by replacing the following matrices: $Q_i \leftarrow Q_i(:, 1{:}k)$, $R_i \leftarrow R_i(1{:}k, 1{:}k)$ and for the right-hand side of (4.31) $I \leftarrow I_k$. This partial periodic Schur form may be of interest for identifying the slow converging modes or the unstable modes of a periodic equation or for computing a few eigenvalues of the p-cyclic matrix.

A (possibly implicitly) restarted p-cyclic Arnoldi method might be useful for computing a partial periodic Schur form of a p-cyclic matrix. This should be done in a similar way as variants of Arnoldi's method compute a standard partial Schur form. Note that such a p-cyclic Arnoldi method will have nice parallelization properties.

4.4 Solving the reduced least-squares problem

In step 3 of Algorithm 4.1, the reduced least-squares problem

$$\min_y \|d - H^{(m)} y\| \tag{4.32}$$

has to be solved. There are several ways for doing this and in this subsection we will discuss some of these. For each approach we display the approximate costs for solving (4.32), and discuss other properties. These approximate costs are based on the assumption that p is relatively small compared with m, say $p < m$. We will see that the approach of Section 4.4.3 is preferable if $p^2 \ll 2n$. The other approaches may be of interest if p is relatively large compared with m and n. The FOM approach of (4.4.1) is mainly of theoretical interest because it generalizes the relation between standard GMRES and FOM.

4.4.1 p-cyclic FOM

In the standard GMRES method, an $m+1$ by m least-squares problem

$$\min_y \|\beta e_1 - G_m y\|$$

is solved, with G_m a Hessenberg matrix. An approximate solution to this problem can be found by dropping the last row of G_m and the last entry of e_1. Then the least-squares problem reduces to a linear system and the resulting method is called FOM (full orthogonalization method) [76]. In the following we will see that we can apply a similar trick to (4.32). This gives a relatively inexpensive approximate solution of (4.32).

The FOM approximation for (4.32) is obtained by neglecting the matrix entries $\{H_i^{(m)}\}_{m+1,m}$, $i = 1, \ldots, p$, of $H^{(m)}$. So, we delete the rows $m+1, 2(m+1), \ldots, p(m+1)$

of (4.32) and an approximate solution y^{F} of (4.32) is found by solving

$$\begin{bmatrix} I & 0 & 0 & -H_1^{(m,\square)} \\ -H_2^{(m,\square)} & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -H_p^{(m,\square)} & I \end{bmatrix} y^{\mathrm{F}} = \begin{bmatrix} \delta_1 e_1 \\ \vdots \\ \delta_p e_1 \end{bmatrix}, \tag{4.33}$$

where $H_i^{(m,\square)} = H_i^{(m)}(1{:}m, 1{:}m)$, and e_1 is the unit vector $[1, 0, \ldots, 0]^T$ of suitable dimension. The p-cyclic FOM solution $x^{\mathrm{F},(m)}$ is defined by

$$x^{\mathrm{F},(m)} = V^{(m)} y^{\mathrm{F}} . \tag{4.34}$$

Applying block Gaussian elimination to (4.33) leads to a reduced linear system

$$(I - H_p^{(m,\square)} \cdot \ldots \cdot H_1^{(m,\square)}) y_p^{\mathrm{F}} = \mathrm{rhs} ,$$

and y^{F} can be computed by a forward recursion. Solving (4.33) by block Gaussian elimination costs approximately

$$\frac{1}{3}pm^3 + \frac{1}{2}p^2m^2 \ \text{ flops} .$$

The elimination procedure may not be stable if one or more $H_i^{(m,\square)}$, $i = 1, \ldots, p$, has large singular values, see for instance [7]. In that case, Gaussian elimination with a suitable pivot strategy can be used for (4.33), but this may be more expensive. Another disadvantage is that (4.33) may be singular.

Instead of interpreting (4.33) as an approximation to (4.32) there is another way of deriving (4.33): We want to solve $Ax = b$ approximately, with $x^{(m)} \in \mathrm{range}(V^{(m)})$, so we write $x^{(m)} = V^{(m)}y$. The residual is

$$r^{(m)} = b - Ax^{(m)} = b - AV^{(m)}y.$$

In the Ritz-Galerkin approach the residual $r^{(m)}$ is orthogonal to the search space: $b - AV^{(m)}y \perp \mathrm{range}(V^{(m)})$. This leads to ${V^{(m)}}^T(b - AV^{(m)}y) = 0$, or

$${V^{(m)}}^T A V^{(m)} y = {V^{(m)}}^T b , \tag{4.35}$$

and this equation is precisely the same as (4.33).

In Section 4.4.4 we will derive some relations between the convergence of p-cyclic GMRES and the convergence of p-cyclic FOM

4.4.2 Structured QR factorization

A common way to solve the least-squares problem (4.32) is by QR factorization of $H^{(m)}$: $Q^{(m)}R^{(m)} = H^{(m)}$, with an upper triangular $R^{(m)} \in \mathbb{R}^{mp \times (m+1)p}$ and an orthogonal $Q^{(m)} \in \mathbb{R}^{(m+1)p \times (m+1)p}$, and (using MATLAB notation)

$$y = R^{(m)}(1{:}mp, 1{:}mp)^{-1}\{{Q^{(m)}}^T d\}(1{:}mp) .$$

A QR factorization can be constructed by successively applying Householder reflections Q_k, $k = 1, \ldots, pm$ to $H^{(m)}$ and d in order to transform $H^{(m)}$ to an upper triangular form, see [42, Ch. 5]. Fortunately, the Householder vectors v_k, which define the reflections $Q_k = I - 2v_k v_k^T$, are sparse (or more precisely, structured) vectors, because of the special structure of (4.32). The QR factorization for the structured matrix $H^{(m)}$ costs approximately

$$8pm^3 + 4p^2m^2 \quad \text{flops}\,, \tag{4.36}$$

if the special structure of $H^{(m)}$ is exploited. This is much more expensive than the block Gaussian elimination approach in FOM. However, the QR approach gives the exact least-squares solution and the QR approach may be more stable. Figure 4.2 shows the nonzero pattern of a matrix $H^{(6)}$, with $p = 4$, and the nonzero pattern of $R^{(6)}$ from the QR factorization of $H^{(6)}$.

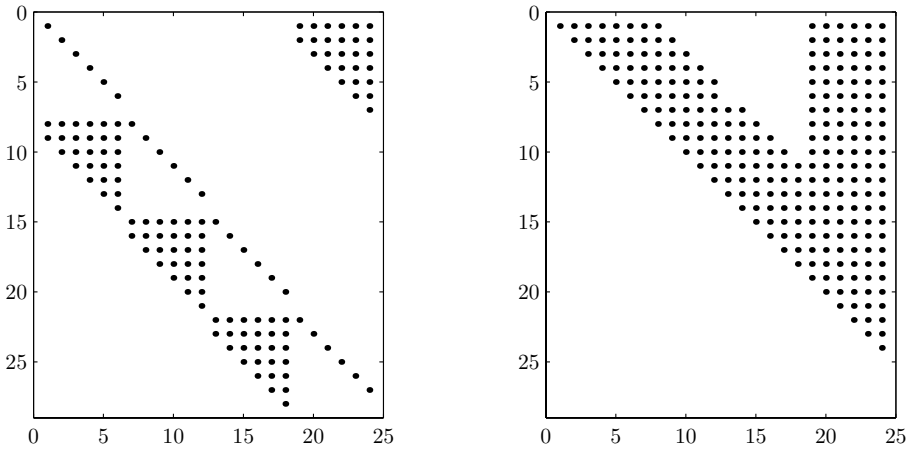


Figure 4.2: Left: nonzero pattern of $H^{(6)}$, with $p = 4$. Right: nonzero pattern of $R^{(6)}$, $Q^{(6)}R^{(6)} = H^{(6)}$.

In practice m is not known beforehand, and m will be increased until the least-squares problem (4.32) can be solved with sufficiently small residual norm. The standard GMRES method for linear systems updates the QR factorization of the Hessenberg matrix each iteration. This enables a cheap computation of the residual norm of the linear system $Ax = b$ without actually computing $\|b - Ax\|$. In p-cyclic GMRES this is slightly more complicated. Updating the QR factorization of $H^{(m)}$, after m has been increased by one, is possible, but the accumulated costs for the QR factorization of $H^{(m)}$ would be much higher than $8pm^3 + 4p^2m^2$ flops: $\mathcal{O}(p^2m^3)$ flops. However, after reordering $H^{(m)}$, a cheaper incremental QR factorization is possible, assuming that $m \gg p$. We will discuss this in Section 4.4.3.

In the remainder of this section we will propose a heuristic for an inexpensive estimation of the residual norm of (4.32). A stopping criterion can be based on this heuristic,

which makes the QR factorization of this section slightly more useful for practical applications.

Suppose that $B_i = B$, $i = 1, \ldots, p$, then block Gaussian elimination of $x_1, \ldots, x_{p-1}$ in the p-cyclic linear system (4.6) leads to $(I - B^p)x_p = \hat{b}_p$. This can also be written as $(I - \nu_1 B) \cdot (I - \nu_2 B) \cdot \ldots \cdot (I - \nu_p B)x_p = \hat{b}_p$, with $\nu_i^p = 1$. This system can be solved by solving successively $(I - \nu_1 B)w_1 = \hat{b}_p$, $(I - \nu_2 B)w_2 = w_1$, etcetera. Solving $(I - \nu_i B)w_i = w_{i-1}$ with standard GMRES gives a relative residual norm of $\min_y \|(\widetilde{I} - \nu_i G_m)y - e_1\|$, where G_m is the Hessenberg matrix defined by the Arnoldi process for the Krylov subspace $\mathcal{K}(B, w_{i-1})$.

This suggests that the value of $\min_y \|(\widetilde{I} - \alpha H_i^{(m)})y - e_1\|$ tells something about the convergence of p-cyclic GMRES. We propose the following heuristic for the residual norm of (4.6):

$$\begin{aligned} \|r^{(m)}\| &= \|b - Ax^{(m)}\| \\ &\approx \chi_m \equiv \|b\| \sum_i \max_{\alpha \in \{-1,1\}} \min_y \|(\widetilde{I} - \alpha H_i^{(m)})y - e_1\| \,. \end{aligned} \tag{4.37}$$

We compute the maximum for two values of α, with $|\alpha| = 1$. Other values of α, with $|\alpha| = 1$, are possible, but this does not seem to improve the estimation in practice, because $\min_y \|(\widetilde{I} - \alpha H_i^{(m)})y - e_1\|$ often has a maximum at $\alpha = 1$ or $\alpha = -1$. The value of $\min_y \|(\widetilde{I} - \alpha H_i^{(m)})y - e_1\|$ is easy to compute with a QR factorization of $\widetilde{I} - \alpha H_i^{(m)}$. Therefore, the cost of computing (4.37) is relatively small: $\mathcal{O}(pm^2)$ flops.

For the circuit simulation problems considered in Section 4.13.1, with $p \leq 8$, we found that roughly $0.1\chi_{\lceil 1.2m \rceil} \leq \|r^{(m)}\| \leq 10\chi_{\lfloor 0.8m \rfloor}$, with χ_m defined in (4.37). The functions $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ round to the next and previous nearest integer, respectively.

4.4.3 Block QR factorization

In this section we consider the solution of a reordered form of the least-squares problem (4.32). From equation (4.24b): $AV^{(m)} = V^{(m+1)}H^{(m)}$ it follows that $H^{(m)} = {V^{(m+1)}}^T A V^{(m)}$. We will define the reordered matrix $\hat{H}^{(m)}$ by

$$\hat{H}^{(m)} = {\hat{V}^{(m+1)}}^T A \hat{V}^{(m)} \,, \tag{4.38}$$

where $\hat{V}^{(j)}$ is a column permutation of $V^{(j)}$ according to the order in which these columns are constructed by Algorithm 4.1. So, the reordered matrix $\hat{V}^{(j)}$ is:

$$\hat{V}^{(j)} = \begin{bmatrix} v_1^{(1)} & 0 & \ldots & 0 & v_1^{(2)} & 0 & \ldots & 0 & \ldots & v_1^{(j)} & 0 & \ldots & 0 \\ 0 & v_2^{(1)} & & 0 & 0 & v_2^{(2)} & & 0 & \ldots & 0 & v_2^{(j)} & & 0 \\ \vdots & & \ddots & & \vdots & & \ddots & & \ldots & \vdots & & \ddots & \\ 0 & 0 & & v_p^{(1)} & 0 & 0 & & v_p^{(2)} & \ldots & 0 & 0 & & v_p^{(j)} \end{bmatrix} . \tag{4.39}$$

Figure 4.3 shows the nonzero pattern of a matrix $\hat{H}^{(6)}$, with $p = 4$ and the nonzero pattern of $R^{(6)}$ from the QR factorization of $\hat{H}^{(6)}$. The matrix $\hat{H}^{(j+1)}$ can be obtained

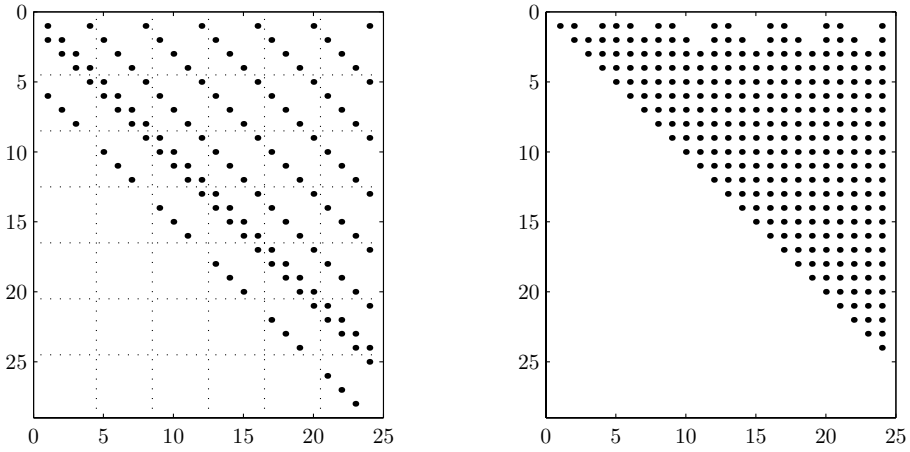


Figure 4.3: Left: nonzero pattern of the reordered matrix $\hat{H}^{(6)}$, with $p = 4$. Right: nonzero pattern of $R^{(6)}$, $Q^{(6)}R^{(6)} = \hat{H}^{(6)}$.

in step $j+1$ of p-cyclic GMRES by appending p new columns at the right-hand side of $\hat{H}^{(j)}$.

In the standard GMRES method, a sequence of Givens rotations is computed to eliminate the subdiagonal entries of the Hessenberg matrix. Now observe that $\hat{H}^{(m)}$ has a block Hessenberg form, and instead of a sequence of Givens rotations we can apply a sequence of $2p$ by $2p$ orthogonal transformations to the block rows of $\hat{H}^{(m)}$ to transform $\hat{H}^{(m)}$ to an upper triangular matrix.

The reordered least-squares problem is

$$\min_{\hat{y}} \|\hat{d} - \hat{H}^{(m)}\hat{y}\| . \tag{4.40}$$

Here $\hat{d}$ is the vector $[\delta_1, \delta_2, \ldots, \delta_p, 0, 0, \ldots, 0]^T$ of length $p(m+1)$. The p-cyclic GMRES solution is $x^{(m)} = \hat{V}^{(m)}\hat{y}$, where $\hat{y}$ minimizes (4.40). A block partition of (4.40) with p by p blocks is defined by:

$$\hat{H}^{(m)} = \begin{bmatrix} H_{11} & H_{12} & \ldots & H_{1m} \\ H_{21} & H_{22} & & \vdots \\ & H_{32} & & \vdots \\ & & \ddots & \vdots \\ & & & H_{m+1,m} \end{bmatrix} \text{ and } \hat{d} = \begin{bmatrix} d_1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix} .$$

Algorithm 4.2 solves the least-squares problem (4.40). The $m+1$ by $m+1$ block matrix Ω_j transforms the subdiagonal blocks $H_{j+1,j}$ to zero and transforms the diagonal blocks H_{jj} to upper triangular matrices: a QR factorization of $\hat{H}^{(m)}$ is constructed. The right-hand side $\hat{d}$ is also premultiplied by the orthogonal matrix Ω_j.

Algorithm 4.2: Solve the least-squares problem (4.40)

1. Initialization:
   Set $H^{[0]} = \hat{H}^{(m)}$ and $d^{[0]} = \hat{d}$
2. Loop:
   for $j = 1 : m$
   Make a QR factorization:
   $Q_j \, R_j = \begin{bmatrix} H_{jj}^{[j-1]} \\ H_{j+1,j} \end{bmatrix}$ with a $2p$ by $2p$ matrix Q_j
   Define:
   $\Omega_j = \begin{bmatrix} I_{p(j-1)} & & \\ & Q_j & \\ & & I_{p(m-j)} \end{bmatrix}$
   Transform $H^{[j-1]}$ and $d^{[j-1]}$:
   $H^{[j]} = \Omega_j^T H^{[j-1]}$
   $d^{[j]} = \Omega_j^T d^{[j-1]}$
   end
3. Finish:
   $\hat{y} = \{H^{[m]}(1{:}mp, 1{:}mp)\}^{-1} \, d^{[m]}(1{:}mp)$

Define the orthogonal matrix ${Q^{(m)}}^T = \Omega_m^T \cdot \ldots \cdot \Omega_1^T$, then at the end of step 2 we have that

$$ {Q^{(m)}}^T \hat{H}^{(m)} = H^{[m]} = \begin{bmatrix} H_{11}^{[m]} & H_{12}^{[m]} & \ldots & H_{1m}^{[m]} \\ 0 & H_{22}^{[m]} & & \vdots \\ & 0 & & \vdots \\ & & \ddots & H_{mm}^{[m]} \\ & & & 0 \end{bmatrix} \text{ and } {Q^{(m)}}^T \hat{d} = d^{[m]} = \begin{bmatrix} d_1^{[m]} \\ \vdots \\ \vdots \\ \vdots \\ d_{m+1}^{[m]} \end{bmatrix}, $$

where $H^{[m]}$ is an upper triangular matrix. So (4.40) transforms to

$$ \min_{\hat{y}} \| d^{[m]} \; - \; H^{[m]} \hat{y} \| \, , \tag{4.41} $$

which is solved in step 3 of Algorithm 4.2. The minimum of (4.41) is equal to $\| d_{m+1}^{[m]} \|$.

In summary we have that

$$ \min_{x^{(m)} \in V^{(m)}} \| b - A x^{(m)} \| = \min_{\hat{y}} \| \hat{d} \; - \; \hat{H}^{(m)} \hat{y} \| \tag{4.42a} $$

$$ = \min_{\hat{y}} \| d^{[m]} \; - \; H^{[m]} \hat{y} \| = \| d_{m+1}^{[m]} \| \, . \tag{4.42b} $$

In an actual implementation of p-cyclic GMRES the computations of Algorithm 4.2 can be rearranged such that $d^{[j]}$ and the QR factorization of $\hat{H}^{(j)}$ are updated at each iteration. For example, in iteration j, $\Omega_1, \ldots, \Omega_{j-1}$ are applied to the j-th block column

of $\hat{H}^{(m)}$. Then the QR factorization $Q_j R_j$ is computed in order to transform the j-th column to upper triangular form. The right-hand side is also transformed with Q_j^T. The norm of the residual $\|b - Ax^{(j)}\|$ can be monitored by computing $\|d_{j+1}^{[j]}\|$ at each iteration. The Matlab code of P(GMRES), see Appendix A and Chapter 5, shows how to update a QR factorization of a block upper Hessenberg matrix $\hat{H}^{(m)}$.

In an efficient implementation of Algorithm 4.2, Q_j is not computed explicitly, but is stored as a sequence of (structured) Householder transformations that exploit the structure of $\hat{H}^{(m)}$.

The matrix $H^{(m)}$ has lower bandwidth $p+m$, while $\hat{H}^{(m)}$ has lower bandwidth $p+1$, so for $\hat{H}^{(m)}$ fewer lower triangular entries have to be eliminated in the QR factorization. However, the R factor of $\hat{H}^{(m)}$ has much more fill-in, which is shown by Figure 4.3. Therefore, the QR factorization of $\hat{H}^{(m)}$ may be more expensive than the QR factorization of $H^{(m)}$, depending on p and m. The QR factorization of $\hat{H}^{(m)}$ costs approximately

$$2p^3m^2 \quad \text{flops}\,. \tag{4.43}$$

The costs (4.43) are more expensive than the approach of the previous Section (4.36) if (to first order) $p^2 > 4m$.

Note that the Householder orthogonalization in Algorithm 4.1 costs approximately

$$4pm^2n \quad \text{flops}\,.$$

So the costs (4.43) for the least-squares system are negligible compared with the orthogonalization if $p^2 \ll 2n$.

4.4.4 Relations between p-cyclic GMRES and p-cyclic FOM

Relationships between the convergence of standard GMRES and the convergence of standard FOM have been derived by (among others) Cullum and Greenbaum [18] and Brown [16]. Deriving generalizations of these relationships for the p-cyclic case is not always possible as we will see here.

The p-cyclic FOM solution is defined in (4.33) and (4.34). In order to derive a relationship between p-cyclic GMRES and p-cyclic FOM, the reordering of Section 4.4.3 turns out to be convenient. The reordered reduced FOM system is given by

$$\hat{H}^{(m,\square)}\hat{y}^{\mathrm{F}} = \hat{d}(1{:}mp)\,, \tag{4.44}$$

where $\hat{H}^{(m,\square)} = \hat{H}^{(m)}(1{:}m, 1{:}m)$. The FOM solution is given by $x^{\mathrm{F},(m)} = \hat{V}^{(m)}\hat{y}^{\mathrm{F}}$.

After step $m-1$ and step m of Algorithm 4.2 we have that

$$H^{[m-1]} = \begin{bmatrix} \ddots & \vdots \\ 0 & H_{mm}^{[m-1]} \\ 0 & H_{m+1,m} \end{bmatrix}, \quad d^{[m-1]} = \begin{bmatrix} \vdots \\ d_m^{[m-1]} \\ 0 \end{bmatrix},$$

and

$$H^{[m]} = \begin{bmatrix} \ddots & \vdots \\ 0 & H_{mm}^{[m]} \\ 0 & 0 \end{bmatrix}, \quad d^{[m]} = \begin{bmatrix} \vdots \\ d_m^{[m]} \\ d_{m+1}^{[m]} \end{bmatrix},$$

respectively. Step $m-1$ and step m are related by

$$\begin{bmatrix} C_a & S_b \\ S_a & C_b \end{bmatrix} \begin{bmatrix} H_{mm}^{[m-1]} \\ H_{m+1,m} \end{bmatrix} = \begin{bmatrix} H_{mm}^{[m]} \\ 0 \end{bmatrix}, \tag{4.45}$$

and

$$\begin{bmatrix} C_a & S_b \\ S_a & C_b \end{bmatrix} \begin{bmatrix} d_m^{[m-1]} \\ 0 \end{bmatrix} = \begin{bmatrix} d_m^{[m]} \\ d_{m+1}^{[m]} \end{bmatrix}, \tag{4.46}$$

where C_a, S_a, C_b, S_b are defined by a block partition of Q_j^T:

$$Q_j^T = \begin{bmatrix} C_a & S_b \\ S_a & C_b \end{bmatrix}.$$

The p by p matrices C_a, S_a, C_b, S_b are generalizations of the sine s and the cosine c arising in standard GMRES. The orthogonal transformations $Q_1^T, \ldots, Q_{m-1}^T$, have put the linear system (4.44) into upper block triangular form. Therefore, the m-th block of $\hat{y}^{\mathrm{F},(m)}$ is given by

$$\hat{y}_m^{\mathrm{F},(m)} = {H_{mm}^{[m-1]}}^{-1} d_m^{[m-1]}. \tag{4.47}$$

The residual norm of the FOM solution is

$$\begin{aligned} \|r^{\mathrm{F},(m)}\| &= \|\hat{d} - \hat{H}^{(m)} \hat{y}^{\mathrm{F},(m)}\| && \text{(4.48a)} \\ &= \|H_{m+1,m} {H_{mm}^{[m-1]}}^{-1} d_m^{[m-1]}\| && \text{(4.48b)} \\ &= \|C_b^{-1} S_a d_m^{[m-1]}\|. && \text{(4.48c)} \end{aligned}$$

The last equality follows from the last block row of (4.45). The p-cyclic GMRES solution has residual norm

$$\|r^{\mathrm{G},(m)}\| = \|d_{m+1}^{[m]}\| = \|S_a d_m^{[m-1]}\|. \tag{4.49}$$

Formulas (4.48) and (4.49) lead to the inequality:

$$\|r^{\mathrm{G},(m)}\| / \|C_b\| \leq \|r^{\mathrm{F},(m)}\| \leq \|r^{\mathrm{G},(m)}\| \, \|C_b^{-1}\|.$$

For $p = 1$, this simplifies to $\|r^{\mathrm{G},(m)}\| / |c| = \|r^{\mathrm{F},(m)}\|$, a well known result [16], [18]. Note that we have assumed that both $H_{mm}^{[m-1]}$ and C_b are nonsingular.

The next theorem gives another relation between p-cyclic FOM and p-cyclic GMRES.

Theorem 4.5 *If the residual norm of p-cyclic GMRES stagnates, that is $\|r^{\mathrm{G},(m)}\| = \|r^{\mathrm{G},(m-1)}\| \neq 0$, then the linear system (4.44) is singular, and therefore the p-cyclic FOM solution is not defined.*

Proof The residual norm of p-cyclic GMRES at iteration $m-1$ or m is equal to the minimum value of a least-squares problem:

$$\min_{\hat{y} \in \mathbb{R}^{pm}} \|\hat{d} - \hat{H}^{(m)} \hat{y}\| = \min_{\hat{y} \in \mathbb{R}^{p(m-1)}} \|\hat{d} - \hat{H}^{(m-1)} \hat{y}\|.$$

It follows that

$$\|d_{m+1}^{[m]}\| = \min_{\hat{y}\in\mathbb{R}^{pm}} \|\hat{d} - \hat{H}^{(m)}\hat{y}\| = \min_{\hat{y}\in\mathbb{R}^{p(m-1)}} \|\hat{d} - \hat{H}^{(m-1)}\hat{y}\| = \|d_m^{[m-1]}\| \,. \tag{4.50}$$

The matrix Q_j is orthogonal, so from (4.46) it follows that

$$\| \begin{bmatrix} d_m^{[m-1]} \\ 0 \end{bmatrix} \| = \| \begin{bmatrix} d_m^{[m]} \\ d_{m+1}^{[m]} \end{bmatrix} \| \,. \tag{4.51}$$

Combining (4.50) and (4.51) leads to $\|d_m^{[m]}\| = 0$, and therefore $d_m^{[m]} = 0$. From the equality $C_a d_m^{[m-1]} = d_m^{[m]} = 0$ we see that C_a is singular (note that $d_m^{[m-1]} \neq 0$, otherwise p-cyclic GMRES has converged after $m-1$ iterations). The next equation shows that this implies that $H_{mm}^{[m-1]}$ is singular too.

$$\begin{aligned} C_a^T H_{mm}^{[m]} &= C_a^T H_{mm}^{[m]} + S_a^T 0 \\ &= C_a^T C_a H_{mm}^{[m-1]} + C_a^T S_b H_{m+1,m} + S_a^T S_a H_{mm}^{[m-1]} + S_a^T C_b H_{m+1,m} \\ &= H_{mm}^{[m-1]} \,. \end{aligned}$$

Here we used formula (4.45) and the orthogonality of Q_j^T: $C_a^T C_a + S_a^T S_a = I$ and $C_a^T S_b + S_a^T C_b = 0$. From (4.47) we see that $\hat{y}^{\mathrm{F}}$ is not defined if $H_{mm}^{[m-1]}$ is singular. We conclude that the p-cyclic FOM solution is not defined if p-cyclic GMRES stagnates. □

Brown [16] has proved that the solution of standard FOM is not defined if and only if standard GMRES stagnates. In the p-cyclic case this is different because the converse of Theorem 4.5 is not true in general, as can be shown easily by construction.

4.5 Convergence analysis for a special case

It is difficult to analyse the convergence of p-cyclic GMRES, with $p > 1$, for the general case of the p-cyclic linear system (4.6), repeated here for convenience:

$$\begin{bmatrix} I & 0 & 0 & -B_1 \\ -B_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \vdots \\ \vdots \\ \hat{x}_p \end{bmatrix} = \begin{bmatrix} \hat{b}_1 \\ \vdots \\ \vdots \\ \hat{b}_p \end{bmatrix} .$$

However, p-cyclic GMRES converges at least as fast as standard GMRES, see (4.20). In this section we will analyse the convergence for a special case:

$$B_i = B = \text{constant} \,,$$

with B diagonalizable, in combination with the initial guess of $x^{(0)}$-GMRES, see Section 4.2.2. The linear system is:

$$\begin{bmatrix} I & 0 & 0 & -B \\ -B & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hat{b}_p \end{bmatrix} , \tag{4.52}$$

or more compactly $Ax = \hat{b}$. In Algorithm 4.1 we will choose the vector w equal to $\hat{b}_p$. This simplifies the p-cyclic GMRES process:

$$V_i^{(m)} = V_*^{(m)}, \quad H_i^{(m)} = H_*^{(m)}, \quad \text{for } i = 1, \ldots, p\,.$$

The following relation holds for $V_*^{(m)}$ and $H_*^{(m)}$: $BV_*^{(m)}{=}V_*^{(m+1)}H_*^{(m)}$. This is the standard Arnoldi relation associated with the Krylov subspace $\mathcal{K}(B, \hat{b}_p)$.

We will assume that $\{H_*^{(m)}\}_{j+1,j} \neq 0$, $j = 1, \ldots, m{-}1$, otherwise the iterative process has converged because of Theorem 4.3. This assumption implies that $\text{range}(V_*^{(m)}) = \mathcal{K}(B, \hat{b}_p)$. The approximate solution $x^{(m)}$ of p-cyclic GMRES satisfies

$$\begin{aligned} x_i^{(m)} \in \text{range}(V_*^{(m)}) &= \mathcal{K}(B, \hat{b}_p) \\ &= \{\, v \mid v = P^{m-1}(B)\hat{b}_p,\ P^{m-1} \in \mathbb{P}^{m-1} \}\,, \end{aligned}$$

where $\mathbb{P}^k$ is the set of polynomials with degree less than or equal to k.

After m iterations the residual norm of p-cyclic GMRES is

$$\|r^{(m)}\| = \min_{x_i \in \mathcal{K}_*^{(m)}} \|\hat{b} - Ax\|\,.$$

We will find an upper bound for $\|r^{(m)}\|$ if we select a suboptimal solution $x^{(m)}$:

$$x^{(m)} = \begin{bmatrix} x_1^{(m)} \\ \vdots \\ x_{p-1}^{(m)} \\ x_p^{(m)} \end{bmatrix} = \begin{bmatrix} BP^{m-p}(B)\hat{b}_p \\ \vdots \\ B^{p-1}P^{m-p}(B)\hat{b}_p \\ P^{m-p}(B)\hat{b}_p \end{bmatrix}\,,$$

(assuming that $p \leq m$). For this approximate solution $x^{(m)}$ the residual is

$$r^{(m)} = \hat{b} - Ax^{(m)} = \begin{bmatrix} r_1^{(m)} \\ \vdots \\ r_{p-1}^{(m)} \\ r_p^{(m)} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hat{b}_p - (I - B^p)P^{m-p}(B)\hat{b}_p \end{bmatrix}\,.$$

We have assumed that B is diagonalizable: $B{=}WDW^{-1}$, with $D{=}\text{diag}(\lambda_1, \ldots, \lambda_n)$. This leads to an upper bound for $\|r^{(m)}\|/\|r^{(0)}\|$:

$$\begin{aligned} \frac{\|r^{(m)}\|}{\|r^{(0)}\|} = \frac{\|\hat{b}_p - (I - B^p)P^{m-p}(B)\hat{b}_p\|}{\|\hat{b}_p\|} &\leq \|I - (I - B^p)P^{m-p}(B)\| \\ &= \|W(I - (I - D^p)P^{m-p}(D))W^{-1}\| \\ &\leq \kappa(W)\|I + (D^p - I)P^{m-p}(D)\| \\ &= \kappa(W)\epsilon^{(m)}\,, \end{aligned}$$

where $\kappa(W)$ is defined by: $\kappa(W) = \|W\|\,\|W^{-1}\|$, and $\epsilon^{(m)}$ is defined by:

$$\epsilon^{(m)} = \min_{P^{m-p} \in \mathbb{P}^{m-p}} \max_{i=1,\ldots,n} |1 + (\lambda_i^p - 1)P^{m-p}(\lambda_i)|\,. \tag{4.53}$$

If B is such that the eigenvalues $\lambda_i, i = 1, \ldots, n$ are real, and $0 \leq \lambda_i \leq \beta < 1, i = 1, \ldots, n$, then we can use Chebyshev polynomials in order to derive an upper bound for $\epsilon^{(m)}$.

$$\epsilon^{(m)} = \min_{P^{m-p} \in \mathbb{P}^{m-p}} \max_{i=1,\ldots,n} |1 + (\lambda_i^p - 1)P^{m-p}(\lambda_i)| \tag{4.54a}$$

$$\leq \min_{P \in \mathbb{P}^{m-p}} \max_{z \in [0,\beta]} |1 + (z^p - 1)P(z)| \,. \tag{4.54b}$$

In Section 4.6.1 we will show that there is an important class of problems for which $0 \leq \lambda_i \leq \beta < 1$.

The minimization problem (4.54b) can be reformulated as: find a polynomial $\widetilde{P}(z)$ of degree $m - p$ such that

$$|\widetilde{P}(z) - \frac{1}{1 - z^p}|$$

is small in some sense, for $z \in [0, \beta]$. This problem can be solved approximately with Chebyshev approximation theory, see for example [17, Ch. 3], which gives a $\widetilde{P}$ that is a linear combination of $m - p$ shifted and scaled Chebyshev polynomials. However, we will focus on finding a polynomial

$$q(z) = 1 + (z^p - 1)P(z)$$

of degree m which is small on $[0, \beta]$ and

$$q(z_i) = 1 \,,$$

for the roots z_i of $z^p - 1 = 0$:

$$z_i = \cos(\frac{2\pi i}{p}) + \mathbf{i} \sin(\frac{2\pi i}{p}), \;\; i = 0, \ldots, p-1 \,, \tag{4.55}$$

where $\mathbf{i} = \sqrt{-1}$. More precisely, the minimization problem (4.54b) is equivalent to

$$\min_{\substack{q \in \mathbb{P}^m \\ q(z_i)=1, i=0,\ldots,p-1}} \max_{z \in [0,\beta]} |q(z)| \,.$$

It is well known that for $p = 1$ the minimum of problem (4.54b) is reached by the scaled and shifted Chebyshev polynomial

$$s_m(z) = \frac{T_m(\frac{2z}{\beta} - 1)}{T_m(\frac{2 \cdot 1}{\beta} - 1)} \,. \tag{4.56}$$

For $p > 1$, the polynomial s_m satisfies $s_m(z_0)$=1, but in general it is not true that $s_m(z_i) = 1, i = 1, \ldots, p-1$. For a polynomial q which is small on $[0, \beta]$ and $q(z_i) = 1$ for all $i = 0, \ldots, p-1$, it is natural to take a linear combination of p scaled and shifted Chebyshev polynomials:

$$q(z) = \sum_{j=0}^{p-1} s_{m-j}(z) d_j \,. \tag{4.57}$$

In general this linear combination will not give us the optimal solution of (4.54b), but the suboptimal solution is rather close to the optimal solution, as we will see later. The condition $q(z_i) = 1$ gives a linear system of p equations:

$$\begin{bmatrix} s_m(z_0) & \dots & s_{m-p+1}(z_0) \\ \vdots & & \vdots \\ s_m(z_{p-1}) & \dots & s_{m-p+1}(z_{p-1}) \end{bmatrix} \begin{bmatrix} d_0 \\ \vdots \\ d_{p-1} \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} . \tag{4.58}$$

The linear system (4.58) has complex coefficients, but surprisingly the solution $d = [d_0, \dots, d_{p-1}]^T$ is real. This can be shown by taking the complex conjugate (denoted by a $^-$) of both the left-hand side and the right-hand side of (4.58):

$$\begin{bmatrix} s_m(\bar{z}_0) & \dots & s_{m-p+1}(\bar{z}_0) \\ \vdots & & \vdots \\ s_m(\bar{z}_{p-1}) & \dots & s_{m-p+1}(\bar{z}_{p-1}) \end{bmatrix} \begin{bmatrix} \bar{d}_0 \\ \vdots \\ \bar{d}_{p-1} \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} . \tag{4.59}$$

Then note that the roots z_i of $z^p = 1$ are related to each other: if $z_i \notin \mathbb{R}$ then $\bar{z}_i = z_{p-i}$. So a simple row permutation of the linear system (4.58) leads to (4.59) with $\bar{d}$ replaced by d. This shows that $\bar{d} = d$, so that d is real. Here we have assumed that the linear system has a unique solution. Our numerical experiments (not reported) indicate that this is a reasonable assumption.

We have computed the solution d of (4.58) for a number of combinations of p, m, and b, and we always found $d_j \geq 0$, $j = 0, \dots, p-1$. This leads to the following conjecture:

Conjecture 4.6 *The linear system (4.58), with z_i and $s_k(z)$ defined by (4.55) and (4.56) respectively, $0 < \beta < 1$, and with $p \leq m$, is nonsingular and has a positive solution: $d_j \geq 0$, $j = 0, \dots, p-1$.*

For the special case $p = 2$, we will prove Conjecture 4.6. With $p = 2$ and $s_m(z_0) = s_{m-1}(z_0) = 1$ (note that $z_0 = 1$) formula (4.58) simplifies to

$$\begin{bmatrix} 1 & 1 \\ s_m(-1) & s_{m-1}(-1) \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} .$$

The solution of this linear system is

$$d_0 = \frac{s_{m-1}(-1) - 1}{s_{m-1}(-1) - s_m(-1)}, \quad d_1 = \frac{s_m(-1) - 1}{s_m(-1) - s_{m-1}(-1)} .$$

For m even we know that $s_m(-1) \geq 1$ and $s_{m-1}(-1) \leq 0$, and we find $0 \leq d_1 \leq 1$. With $d_0 + d_1 = 1$ it follows that $0 \leq d_0 \leq 1$. Similarly we can prove that $d_i \geq 0$ for odd m, this proves the conjecture for the special case $p = 2$.

Now assume that Conjecture 4.6 holds for all $p \geq 1$ and observe that the first row of (4.58) is $d_0 + \ldots + d_{p-1} = 1$. We conclude that the linear combination (4.57) is a convex combination. The following relation holds if $z \in [0, \beta]$:

$$\begin{aligned} |q(z)| &= |d_0 s_m(z) + \ldots + d_{p-1} s_{m-p+1}(z)| \\ &\leq |d_0 s_m(z)| + \ldots + |d_{p-1} s_{m-p+1}(z)| \\ &\leq d_0 s_m(\beta) + \ldots + d_{p-1} s_{m-p+1}(\beta) \;=\; q(\beta) \\ &\leq d_0 s_{m-p+1}(\beta) + \ldots + d_{p-1} s_{m-p+1}(\beta) \;=\; s_{m-p+1}(\beta) \end{aligned}$$

We obtain the following upper bound for the relative residual norm if we combine the previous results.

$$\frac{\|r^{(m)}\|}{\|r^{(0)}\|} \leq \kappa(W) q(\beta) \leq \kappa(W) s_{m-p+1}(\beta) = \frac{\kappa(W)}{T_{m-p+1}(2/\beta - 1)} \tag{4.60}$$

In Section 4.13.2 we will show experimentally that this upper bound is rather sharp.

We already stated that the linear combination q of scaled and shifted Chebyshev polynomials is suboptimal for the minimization problem (4.54b). The next inequalities give some insight in the degree of optimality of q:

$$s_m(\beta) = \min_{\substack{t \in \mathbb{P}^m \\ t(1)=1}} \max_{z \in [0,\beta]} |t(z)| \leq \min_{\substack{t \in \mathbb{P}^m \\ t(z_i)=1,\, i=0,\ldots,p-1}} \max_{z \in [0,\beta]} |t(z)| \tag{4.61a}$$

$$\leq \max_{z \in [0,\beta]} |q(z)| \tag{4.61b}$$

$$= q(\beta) \leq s_{m-p+1}(\beta) \tag{4.61c}$$

In Figure 4.4, the polynomial q is plotted for $m = 16$, $p = 4$, $\beta = 0.95$, and $\kappa(W) = 1$. We see that the absolute value of the local extrema of q are close to $q(\beta)$, which indicates that the polynomial $q(z)$ is near optimal.

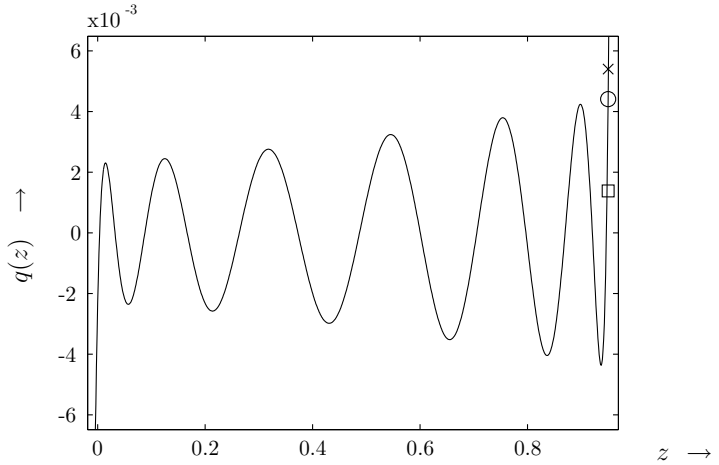


Figure 4.4: Polynomial q for $m = 16$, $p = 4$, and $\beta = 0.95$. Some values of q and s in formula (4.61) are marked in the figure: $\times$: $(e, s_{m-p+1}(\beta)) = (0.95, 5.4 \cdot 10^{-3})$, $\circ$: $(e, q(\beta)) = (0.95, 4.4 \cdot 10^{-3})$, $\square$: $(e, s_m(\beta)) = (0.95, 1.4 \cdot 10^{-3})$.

Now we will compare the upper bound (4.60) for the parallel case with the sequential case for this specific example ($m = 16$, $p = 4$, $\beta = 0.95$, and $\kappa(W) = 1$). Suppose that both in the parallel case and in the sequential case the number of matrix-vector products per processor is 16. The relative residual norm (4.60) is

$$\|r^{(m)}_{\text{par}}\| / \|r^{(0)}_{\text{par}}\| \leq (T_{13}(2/\beta - 1))^{-1} = 5.4 \cdot 10^{-3}$$

for the parallel case. For the sequential case we have to consider 4 iterations applied to the reduced linear system $(I - B^4)x_4 = \hat{b}_4$. The relative residual norm is

$$\|r_{\text{seq}}^{(m)}\|/\|r_{\text{seq}}^{(0)}\| \leq (T_4(2/\beta^4 - 1))^{-1} = 0.050\,,$$

because the eigenvalues of B^4 are in the interval $[0, \beta^4]$, see also formula (4.56). We see that the parallel case has a more favourable upper bound.

4.6 Examples

4.6.1 A linear differential equation: $\dot{y}(t) = Gy(t) + b(t),\ y(0) = y(T)$

In this section we consider the linear periodic steady state problem

$$\dot{y}(t) = Gy(t) + b(t),\ y(0) = y(T),\ t \in [0, T]\,, \tag{4.62}$$

with $T > 0$. The solution of

$$\dot{\hat{y}}(t) = G\hat{y}(t) + b(t),\ \hat{y}(0) = 0,\ b(t+T) = b(t)\,, \tag{4.63}$$

converges to this periodic steady state under certain conditions. Some simple calculations show that for $t = kT$ the solution of (4.63) is

$$\hat{y}(kT) = (I - (e^{GT})^k)(I - e^{GT})^{-1} e^{GT} \int_0^T e^{-Gt} b(t) dt\,. \tag{4.64}$$

Suppose that G is diagonalizable: $G = WDW^{-1}$, with $D = \text{diag}(\lambda_1, \ldots, \lambda_n)$, then

$$(e^{GT})^k = W(e^{DT})^k W^{-1} = W\text{diag}(e^{\lambda_1 kT}, \ldots, e^{\lambda_n kT})W^{-1}\,.$$

It follows that $z_k(t) = \hat{y}(kT + t)$, $t \in [0, T]$, $k = 0, 1, \ldots$, converges to the solution $y(t)$ of (4.62) for $k \to \infty$, if the spectrum of G satisfies

$$\lambda(G) \subset \{z |\ \text{real}(z) < 0\}\,.$$

However, the convergence may be very slow if there is an eigenvalue λ_i with $T\text{real}(\lambda_i) \approx 0$. Solving (4.62) numerically with a time stepping method and $t \to \infty$ is very expensive in that case, compared with a direct discretization of (4.62). From (4.64) we see that (4.63) may diverge for $k \to \infty$, if G has one or more eigenvalues with positive real part. Note that a periodic solution of (4.62) may still exist in this case, but this solution is unstable.

In the remainder of this example, we solve (4.62) by using p-cyclic matrices. The solution of

$$\dot{y}(t) = Gy(t) + b(t),\ y(t_i) = y_i$$

is

$$y(t) = e^{G(t-t_i)} y_i + e^{G(t-t_i)} \int_{t_i}^t e^{-G(t'-t_i)} b(t') dt'\,.$$

A partition of the interval $[0,T]$ of equation (4.62) into p subintervals $[t_i, t_{i+1}]$, $i = 0,\ldots,p-1$, with $0 = t_0 \leq \ldots \leq t_p = T$, leads to the p-cyclic linear system

$$\begin{bmatrix} I & 0 & 0 & -e^{G(t_1-t_0)} \\ -e^{G(t_2-t_1)} & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -e^{G(t_p-t_{p-1})} & I \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ \vdots \\ y_p \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ \vdots \\ b_p \end{bmatrix}, \tag{4.65}$$

with $y_i = y(t_i)$, $i = 1,\ldots,p$, and

$$b_i = e^{G(t_i-t_{i-1})} \int_{t_{i-1}}^{t_i} e^{-G(t-t_{i-1})} b(t) dt \,.$$

Note that the e^{GT} matrix appears again if $y_1,\ldots,y_{p-1}$ are eliminated from (4.65) by block Gaussian elimination:

$$(I - e^{GT}) y_p = \hat{b}_p \,.$$

A one step discretization method for (4.62) may lead to a similar p-cyclic system as (4.65), see Section 4.1. However, then $B_i = e^{G(t_i-t_{i-1})} + \mathcal{O}(h^k)$ instead of $B_i = e^{G(t_i-t_{i-1})}$, where k is the global order of discretization and h is the step size. Moreover, the b_i of the p-cyclic linear system (4.6) is not equal to the b_i of (4.65), because of discretization errors. We may expect approximately the same convergence of p-cyclic GMRES for the exact problem (4.65) and the discretized problem, if the step sizes are sufficiently small.

Now, assume that G is diagonalizable with real eigenvalues λ_i such that $\lambda_1 \leq \ldots \leq \lambda_n < 0$ (for example: G is a symmetric negative definite matrix) and assume that $t_i = Ti/p$. Then the B_i in the p-cyclic linear system (4.6) is $B_i = B = e^{GT/p}$, which has eigenvalues $0 < e^{\lambda_1 T/p} \leq \ldots \leq e^{\lambda_n T/p} \leq e < 1$. The convergence results of Section 4.5 can be used for this problem. Note that (4.64) (and therefore (4.63)) may converge very slowly if $e^{\lambda_n T/p} \approx 1$ and $e^{\lambda_{n-1} T/p} \ll 1$. The p-cyclic GMRES method converges well in this case, because it is easy to find a polynomial such that $\epsilon^{(m)}$ (see formula (4.53)) is small. This indicates that computing the periodic steady state of a circuit, as explained in Section 1.3, may be much more efficient than computing a transient analysis solution which has converged to a periodic steady state.

4.6.2 Tridiagonalizing a matrix

The p-cyclic GMRES idea presented in Section 4.3 can be used to tridiagonalize a general, not necessarily square, matrix. This might be useful in solving linear systems $By = c$, or sparse least-squares problems. We will derive a method which has similarities with the LSQR method of Paige and Saunders [67]. We have not found practical applications where this new method performs better than existing methods. The aim of this example is to show that a general unsymmetric matrix can be tridiagonalized by an orthogonal basis of a non-Krylov subspace.

Let us consider the damped least-squares problem

$$\min \left\| \begin{bmatrix} B \\ \lambda I \end{bmatrix} y - \begin{bmatrix} c \\ 0 \end{bmatrix} \right\|, \tag{4.66}$$

where λ is a parameter. The least-squares problem $\min_y \|By - c\|$ may have a non unique solution, for example if B has more columns than rows. With a $\lambda > 0$, the least-squares problem (4.66) is guaranteed to have a unique solution. The least-squares problem (4.66) can be transformed into an augmented linear system:

$$\begin{bmatrix} -\lambda^2 I & B^T \\ B & I \end{bmatrix} \begin{bmatrix} y \\ r \end{bmatrix} = \begin{bmatrix} 0 \\ c \end{bmatrix} . \tag{4.67}$$

The 'y'-part of the solution of (4.67) solves the least-squares problem (4.66). Note that the linear system (4.67) is nonsingular for any B if $\lambda > 0$.

The LSQR method method is based on the symmetric Lanczos algorithm applied to the symmetric system (4.67), in combination with a (stable) QR factorization of the reduced linear system. Similar to $x^{(0)}$-GMRES, the Krylov subspace has a special structure, because the right-hand side of (4.67) has a zero and a nonzero block, see expression (4.13). This structure is exploited by LSQR. The LSQR method uses short term recurrences to build the Krylov subspace, just like the symmetric Lanczos process.

Here we will apply p-cyclic Arnoldi (this is step one and two of Algorithm 4.1) to (4.67), so $v_1^{(1)} = c/\|c\|$ and $v_2^{(1)}$ a random nonzero vector of unit length. The p-cyclic Arnoldi algorithm reduces the matrix in (4.67) to a small matrix:

$$\begin{bmatrix} -\lambda^2 I & B^T \\ B & I \end{bmatrix} \begin{bmatrix} V_1^{(m)} & 0 \\ 0 & V_2^{(m)} \end{bmatrix} = \begin{bmatrix} V_1^{(m+1)} & 0 \\ 0 & V_2^{(m+1)} \end{bmatrix} \begin{bmatrix} -\lambda^2 \widetilde{I} & H_1^{(m)} \\ H_2^{(m)} & \widetilde{I} \end{bmatrix} . \tag{4.68}$$

With $\bar{H}_i^{(m)} = H_i^{(m)}(1{:}m, 1{:}m)$ we have

$$\begin{bmatrix} V_1^{(m)} & 0 \\ 0 & V_2^{(m)} \end{bmatrix} \begin{bmatrix} -\lambda^2 I & B^T \\ B & I \end{bmatrix} \begin{bmatrix} V_1^{(m)} & 0 \\ 0 & V_2^{(m)} \end{bmatrix} = \begin{bmatrix} -\lambda^2 I & \bar{H}_1^{(m)} \\ \bar{H}_2^{(m)} & I \end{bmatrix} . \tag{4.69}$$

The left-hand side of (4.69) is a symmetric matrix, so that $\bar{H}_1^{(m)^T} = \bar{H}_2^{(m)}$. Since both $\bar{H}_1^{(m)}$ and $\bar{H}_2^{(m)}$ are Hessenberg matrices, they have to be tridiagonal matrices and $H_1^{(m)}$ and $H_2^{(m)}$ are tridiagonal too. In Section 4.4.3 we reordered the columns of $V^{(m)}$ in order to have a small lower bandwidth of $\hat{H}^{(m)}$: see expression (4.39) and Figure 4.3. Here $H^{(m)}$ is a symmetric matrix, so $\hat{H}^{(m)}$ is symmetric, and it is easy to see that $\hat{H}^{(m)}$ is a block tridiagonal matrix with 2 by 2 blocks, for instance

$$\hat{H}^{(4)} = \begin{bmatrix} \star & \star & & \star & & & & \\ \star & \star & \star & & & & & \\ & \star & \star & \star & & \star & & \\ \star & & \star & \star & \star & & & \\ & & & \star & \star & \star & & \star \\ & & \star & & \star & \star & \star & \\ & & & & \star & \star & \star & \\ & & & \star & & \star & \star & \\ & & & & & \star & \star & \end{bmatrix} .$$

A QR factorization of $\hat{H}^{(m)}$ is useful in order to solve the reduced least-squares problem (4.40). This QR factorization can be made by applying 4 by 4 orthogonal matrices to selected rows of $\hat{H}^{(m)}$, in the same way as in Section 4.4.3. The 2 by 2 block structure of $\hat{H}^{(m)}$ implies that $B_i \hat{v}_i^{(j)}$ is orthogonal to $v_i^{(k)}$, $k = 1, \ldots j-2$. So we only

have to orthogonalize $B_i\hat{v}_i^{(j)}$ with respect to $v_i^{(j)}$ and $v_i^{(j-1)}$. In Algorithm 4.1 we have used Householder orthogonalization, but (iterated) modified Gram-Schmidt orthogonalization is more appropriate here. Householder orthogonalization forms the vector $(I-2u_i^{(j)}u_i^{(j)^T})\cdot\ldots\cdot(I-2u_i^{(1)}u_i^{(1)^T})B_i\hat{v}_i^{(j)}$ and we do not see if this leads to short recurrences. The method can be implemented with short-term recurrences if Gram-Schmidt orthogonalization is used (just as MINRES [68] can be viewed as a short-term recurrence variant of GMRES in the symmetric case). Using these short-term recurrences, it is possible to implement the method with a fixed amount of work and storage per iteration.

Now we return to (4.68) and observe that we have tridiagonalized B:

$$BV_1^{(m)} = V_2^{(m+1)}H_2^{(m)} .$$

This can be used to solve the least-squares problem (4.66) with $\lambda = 0$:

$$\min_y \|By - c\| .$$

With $y^{(m)} \in \text{range}(V_1^{(m)})$ we have:

$$\begin{aligned}\min_{y^{(m)}} \|By^{(m)} - c\| &= \min_z \|BV_1^{(m)}z - c\| \\ &= \min_z \|V_2^{(m+1)}H_2^{(m)}z - V_2^{(m+1)}\|c\|e_1\| \\ &= \min_z \|H_2^{(m)}z - \|c\|e_1\| .\end{aligned}$$

A QR factorization $H_2^{(m)} = QR$ is easily obtained by applying m Givens rotations to $H_2^{(m)}$, here R is an upper tridiagonal matrix. This leads to

$$z = R(1{:}m, 1{:}m)^{-1}\{Q^Te_1\}(1{:}m)\|c\|$$

and

$$y^{(m)} = V_1^{(m)}z = V_1^{(m)}R(1{:}m, 1{:}m)^{-1}\{Q^Te_1\}(1{:}m)\|c\| .$$

In an actual implementation of the method we apply only one Givens rotation per iteration and update $\{Q^Te_1\}(1{:}m)\|c\|$ immediately. Similar to GMRES and other Krylov minimum residual methods, the norm of the residual is equal to

$$\min_{y^{(m)}\in\text{range}(V_1^{(m)})} \|By^{(m)} - c\| = \{Q^Te_1\|c\|\}_{m+1} .$$

Since R is upper tridiagonal, we can also update $V_1^{(m)}R(1{:}m, 1{:}m)^{-1}$ each iteration with only one new column, which can be computed by a short-term recurrence. This leads to a method which uses two three-term recurrences for the construction of the search space for $\min_y \|By - c\|$. We will call the method tri_LSQR because of the tridiagonalization and the similarities with LSQR.

Now we consider the case where B is symmetric. Assume that $\{H_i^{(m)}\}_{j+1,j} \neq 0$ for $i = 1, 2$ and $j = 1, \ldots, m-1$, then

$$y^{(m)} \in \text{range}(V_1^{(m)}) = \text{span}\underbrace{(w, B^T c, B^T B w, B^T B B^T c, B^T B B^T B w, \ldots)}_{m \text{ vectors}},$$

where w is an arbitrary vector (we avoid using $\mathcal{K}_1^m$ since $b_2 = 0$). If B is symmetric and we choose $w = c$, then

$$y \in \text{range}(V_1^{(m)}) = \text{span}(c, Bc, B^2 c, \ldots, B^{m-1} c),$$

and this is the m dimensional Krylov subspace $\mathcal{K}^m(B, c)$. So we expect a convergence similar to MINRES [68] in this case.

For matrices B which are close to symmetric in some sense, it is natural to choose $w = c$ and hope that the iteration process converges nearly as fast as in the symmetric case. This would be nice since we have a short-term recurrence method for unsymmetric problems. Unfortunately, numerical experiments, not presented here, show that a small perturbation of a symmetric problem often leads to a much slower convergence.

We will finish this section with a numerical experiment that shows that tri_LSQR may be less efficient than other Krylov subspace solvers. Recall that the aim of this section is to show that a general unsymmetric matrix can be tridiagonalized by an orthogonal basis of a non-Krylov subspace. We consider the linear system `circuit_1` of Chapter 2. This linear system is not symmetric, although $A - A^T$ has quite a lot of relatively small nonzero entries (in absolute value). In order to make the linear system more suitable for iterative solvers we scaled the matrix: $\hat{A} = DAD$, with $D = \text{diag}(d_1, \ldots, d_n)$, and $d_i = (\|A(:, i)\|_1 \, \|A(i, :)\|_1)^{-1/4}$. Figure 4.5 shows the convergence of tri_LSQR for the scaled linear system. We see that tri_LSQR does less well than the other methods.

4.7 Block methods

Several block methods are more or less related to p-cyclic GMRES or $x^{(0)}$-GMRES. In this section we will discuss this in more detail.

4.7.1 Block Jacobi

We will show that p-cyclic GMRES can be seen as an (optimal) accelerated block Jacobi process. Take, for example, $p = 4$, then the Jacobi iteration is:

$$x^{(m+1)} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & B_1 \\ B_2 & 0 & 0 & 0 \\ 0 & B_3 & 0 & 0 \\ 0 & 0 & B_4 & 0 \end{bmatrix} x^{(m)}.$$

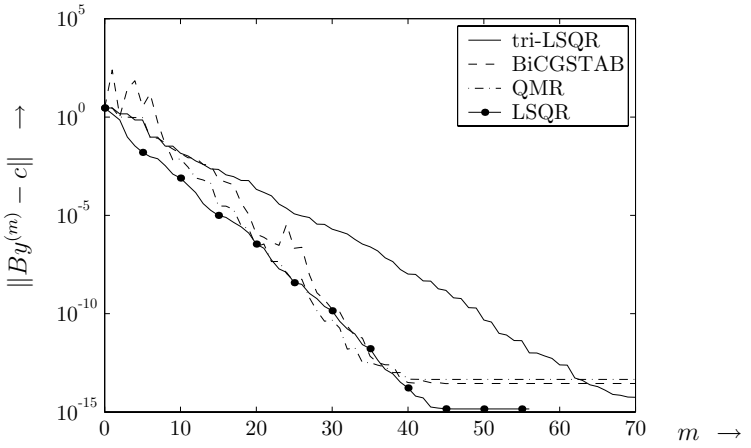


Figure 4.5: The convergence of problem `circuit_1`, scaled. The iteration count is m, and one iteration involves two matrix-vector products.

With an initial guess $x^{(0)} = 0$, the following iterates are generated:

$$x^{(1)} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}, x^{(2)} = \begin{bmatrix} b_1 + B_1 b_4 \\ b_2 + B_2 b_1 \\ b_3 + B_3 b_2 \\ b_4 + B_4 b_3 \end{bmatrix}, x^{(3)} = \begin{bmatrix} b_1 + B_1(b_4 + B_4 b_3) \\ b_2 + B_2(b_1 + B_1 b_4) \\ b_3 + B_3(b_2 + B_2 b_1) \\ b_4 + B_4(b_3 + B_3 b_2) \end{bmatrix}, \ldots .$$

Obviously, the space spanned by the iterates $x_i^{(j)}$ satisfies

$$\mathrm{span}(x_i^{(1)}, \ldots, x_i^{(m)}) = \mathcal{K}_i^m ,$$

and the p-cyclic GMRES method takes a linear combination of $x_i^{(1)}, \ldots, x_i^{(m)}$, such that $\|Ax - b\|$ is minimized. This shows that p-cyclic GMRES is an accelerated block Jacobi process. Note that this conclusion is valid for general p, but for ease of presentation, we restricted ourselves to $p = 4$.

4.7.2 Block Gauss-Seidel

We will illustrate the relationship between $x^{(0)}$-GMRES and the block Gauss-Seidel method by an example for the special case $p = 3$. The block Gauss-Seidel iterates are defined by

$$\begin{bmatrix} I & 0 & 0 \\ -B_2 & I & 0 \\ 0 & -B_3 & I \end{bmatrix} x^{(m+1)} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 & B_1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} x^{(m)} ,$$

with, e.g., $x^{(0)} = 0$. In an explicit form this becomes

$$x^{(m+1)} = \begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \hat{b}_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 & B_1 \\ 0 & 0 & B_2B_1 \\ 0 & 0 & B_3B_2B_1 \end{bmatrix} x^{(m)} ,$$

where $\hat{b}_i$ is defined in (4.10). This leads to

$$x^{(1)} = \begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \hat{b}_3 \end{bmatrix}, x^{(2)} = \begin{bmatrix} \hat{b}_1 + B_1\hat{b}_3 \\ \hat{b}_2 + B_2B_1\hat{b}_3 \\ \hat{b}_3 + \hat{B}_3B_2B_1\hat{b}_3 \end{bmatrix}, x^{(3)} = \begin{bmatrix} \hat{b}_1 + B_1(\hat{b}_3 + B_3B_2B_1\hat{b}_3) \\ \hat{b}_2 + B_2B_1(\hat{b}_3 + B_3B_2B_1\hat{b}_3) \\ \hat{b}_3 + B_3B_2B_1(\hat{b}_3 + B_3B_2B_1\hat{b}_3)) \end{bmatrix}, \dots \tag{4.70}$$

$x^{(0)}$-GMRES finds the m-th iterate in the shifted subspace 'initial guess+Krylov subspace'

$$\begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \\ 0 \end{bmatrix} + \mathcal{K}^m(\begin{bmatrix} I & 0 & -B_1 \\ -B_2 & I & 0 \\ 0 & -B_3 & I \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ \hat{b}_3 \end{bmatrix}) = \begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \\ 0 \end{bmatrix} + \operatorname{span}(\begin{bmatrix} 0 \\ 0 \\ \hat{b}_3 \end{bmatrix}, \begin{bmatrix} B_1\hat{b}_3 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ B_2B_1\hat{b}_3 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ B_3B_2B_1\hat{b}_3 \end{bmatrix}, \begin{bmatrix} B_1B_3B_2B_1\hat{b}_3 \\ 0 \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} \star \\ \star \\ \star \end{bmatrix}) .$$

Comparing this subspace with (4.70), we conclude that $x^{(0)}$-GMRES is an accelerated block Gauss-Seidel process.

4.7.3 Block GMRES

We consider the equation

$$\begin{bmatrix} I & 0 & 0 & -B_1 \\ -B_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix} [x_{(1)}, \dots, x_{(p)}] = \begin{bmatrix} b_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & \dots & 0 \\ 0 & 0 & b_p \end{bmatrix} . \tag{4.71}$$

Postmultiplying (4.71) by $[1, \dots, 1]$ leads to

$$\begin{bmatrix} I & 0 & 0 & -B_1 \\ -B_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix} (x_{(1)} + \dots + x_{(p)}) = \begin{bmatrix} b_1 \\ \vdots \\ \vdots \\ b_p \end{bmatrix} ,$$

so we can solve the p-cyclic linear system (4.6) by solving (4.71) and computing $x = x_{(1)} + \dots + x_{(p)}$ afterwards. It is natural to solve (4.71) with a block GMRES method [78]. The search space has a special structure since each right-hand side has only one nonzero block, this is similar to $x^{(0)}$-GMRES. Exploiting this structure leads to a method similar to p-cyclic GMRES.

4.8 Preconditioning p-cyclic linear systems

Suitable preconditioners have not been identified yet for the p-cyclic linear system (4.6). In this section we discuss some difficulties in preconditioning (4.6).

Krylov subspace methods for general linear systems $By = c$ are often used in combination with a preconditioner M. With left preconditioning (for example) a linear system $M^{-1}By = M^{-1}c$ is solved by the Krylov subspace method, instead of $By = c$. Here M is an approximation to B such that, for example $\|M - B\|$ is small or $M^{-1}B \approx I$, and $Mv = w$ is easy to solve. The aim is to transform $By = c$ into a linear system which is easier to solve (faster convergence) by a Krylov subspace method.

One problem of preconditioning (4.6) is that the matrices B_i are often not explicitly available, see Section 4.1, but that does not stop us investigating some preconditioning ideas. For a p-cyclic problem $Ax = b$, it is natural to have a p-cyclic preconditioner M such that $Mv = w$ is easy to solve. For example, for $p = 3$ the preconditioned system is

$$M^{-1}Ax = \begin{bmatrix} I & 0 & -D_1 \\ -D_2 & I & 0 \\ 0 & D_3 & I \end{bmatrix}^{-1} \begin{bmatrix} I & 0 & -B_1 \\ -B_2 & I & 0 \\ 0 & B_3 & I \end{bmatrix} x = \begin{bmatrix} I & 0 & -D_1 \\ -D_2 & I & 0 \\ 0 & D_3 & I \end{bmatrix}^{-1} b\,.$$

Unfortunately, $M^{-1}A$ does not have a p-cyclic structure in general. So methods that exploit the p-cyclic structure of A (the methods of Sections 4.2 and 4.3) cannot be used here, and we have to use general methods such as GMRES and BiCGSTAB for the preconditioned system. We conclude that the idea of using a p-cyclic preconditioner has no practical advantages. Hence, we can use equally well general preconditioners for $Ax = b$, such as ILU, sparse approximate inverses etc. However, our aim is to develop methods that exploit the p-cyclic structure of the problem, so we will not discuss this further.

Up to now we have not found good preconditioners that preserve the p-cyclic structure. The preconditioned system $(MAM^{-1})(Mx) = Mb$, with a block diagonal matrix $M = \mathrm{diag}(D_1, \ldots, D_p)$, has a p-cyclic structure, but $\hat{\mathcal{K}}^m(MAM^{-1}, Mb) = M\hat{\mathcal{K}}^m(A, b)$, so we expect the same convergence behaviour for both linear systems. Another attempt for preconditioning $Ax = b$ can be made by a preconditioner M, with M^{-1} a p-cyclic matrix, or the transpose of a p-cyclic matrix. In these cases the p-cyclic structure is lost too.

Now we consider preconditioning $Ax = b$ with the incomplete block factorization of the p-cyclic matrix A:

$$M = LU = \begin{bmatrix} I & 0 & 0 & 0 \\ -B_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix} \begin{bmatrix} I & 0 & 0 & -B_1 \\ 0 & I & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & I \end{bmatrix} = \begin{bmatrix} I & 0 & 0 & -B_1 \\ -B_2 & I & 0 & B_2B_1 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix}$$

The product LU differs from A only outside the block pattern of A, so we have an incomplete block factorization of A. It is possible to compute the matrix of the preconditioned

system $M^{-1}Ax = U^{-1}L^{-1}Ax = U^{-1}L^{-1}b$ explicitly:

$$L^{-1}A = \begin{bmatrix} I & 0 & 0 & -B_1 \\ 0 & I & 0 & -B_2B_1 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & I - B_p \cdot \ldots \cdot B_1 \end{bmatrix} \tag{4.72}$$

and

$$U^{-1}L^{-1}A = \begin{bmatrix} I & 0 & 0 & -B_1B_p \cdot \ldots \cdot B_1 \\ 0 & I & 0 & -B_2B_1 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & I - B_p \cdot \ldots \cdot B_1 \end{bmatrix}$$

The matrix $I - B_p \cdot \ldots \cdot B_1$ also appears in the reduced linear system (4.9) of Section 4.2: $(I - B_p \cdot \ldots \cdot B_1)x_p = \hat{b}_p$, and we do not expect that GMRES converges better for $U^{-1}L^{-1}Ax = U^{-1}L^{-1}b$ than for (4.9). Moreover, the GMRES inner products and vector updates are more expensive for $U^{-1}L^{-1}Ax = U^{-1}L^{-1}b$. An advantage of the preconditioned case is that $x_1, \ldots, x_p$ are computed instead of only x_p. But note that $x_1, \ldots x_{p-1}$ can be computed relative inexpensively by a forward recursion after x_p has been solved from (4.9). We conclude that solving $Ax = b$ with GMRES, preconditioned by this incomplete block factorization, is not preferable over applying GMRES to the reduced system (4.9).

Telichevesky *et al.* [83] propose preconditioning (4.1) with the block lower triangular part of (4.1) (diagonal blocks included). It is easily seen that this leads to a matrix of the form (4.72), with $p = M$. For this approach the same conclusion as for incomplete block factorization holds.

4.9 Multiple right-hand sides

In periodic AC analysis equations of the form (4.73) have to be solved for the frequencies f_k, $k = 1, \ldots, n_\mathrm{f}$, see Section 1.3, [85].

$$\begin{bmatrix} F_1 & 0 & 0 & -e^{-2\pi \mathbf{i} f_k T} E_M \\ -E_1 & F_2 & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -E_{M-1} & F_M \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} \gamma_{k,1} U \\ \vdots \\ \vdots \\ \gamma_{k,M} U \end{bmatrix}, \tag{4.73}$$

with $\gamma_{k,i} = -e^{2\pi \mathbf{i} f_k t_i}$, $i = 1, \ldots, M$. The matrices E_i and F_i, $i = 1, \ldots, M$, do not depend on k. In this section we will discuss a few ideas for solving (4.73).

Block diagonal scaling of (4.73) leads to the M-cyclic linear system

$$\begin{bmatrix} I & 0 & 0 & -e^{-2\pi \mathbf{i} f_k T} C_1 \\ -C_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -C_M & I \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} \gamma_{k,1} F_1^{-1} U \\ \vdots \\ \vdots \\ \gamma_{k,M} F_M^{-1} U \end{bmatrix} \tag{4.74}$$

Block Gaussian elimination of $y_1, \ldots, y_{M-1}$ leads to

$$(I - e^{-2\pi \mathbf{i} f_k T} B_1) x_1 = b_{k,1} \,, \tag{4.75}$$

where the right-hand side depends on f_k, see also Section 4.1. An efficient direct approach for (4.75) can be based on the Schur decomposition of B_1: $B_1 = QRQ^H$, with an unitary matrix Q and an upper triangular matrix R. With this Schur decomposition (4.75) transforms to $Q(I - e^{-2\pi \mathbf{i} f_k T} R) Q^H x_1 = b_{k,1}$, which is easy to solve for different values of f_k. Computing the Schur decomposition costs $\mathcal{O}(n^3)$ flops. See Section 4.10 for the costs of Block Gaussian elimination.

Telichevesky *et al.* [85] propose solving (4.75) with GCR [29]. Vuik [90] has developed a similar approach for solving linear systems with multiple right-hand sides. With GCR it is rather simple to reuse the search space, which is built for solving (4.75) with right-hand sides $b_{1,1}, \ldots, b_{k-1,1}$, for solving $(I - e^{-2\pi \mathbf{i} f_k T} B_1) x_1 = b_{k,1}$, $k = 2, \ldots, n_{\mathrm{f}}$. This leads to an efficient method for solving the multiple right-hand side problem (4.75). A disadvantage of this approach is that it is not efficiently parallelizable because of the serial nature of the matrix-vector product, see formula (4.7). This is similar to GMRES applied to the reduced system (4.9).

In the remainder of this section we will discuss parallelizable iterative approaches for (4.74). For parallelizability a reduction of (4.74) to a p-cyclic linear system

$$\begin{bmatrix} I & 0 & 0 & -e^{-2\pi \mathbf{i} f_k T} B_1 \\ -B_2 & I & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -B_p & I \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} b_{k,1} \\ \vdots \\ \vdots \\ b_{k,p} \end{bmatrix} , \tag{4.76}$$

with $1 < p \ll M$, seems to be more useful than the reduction of (4.74) to (4.75). Parallelism might be obtained by combining the p-cyclic GMRES idea with the GCR idea. However deriving such a GCR-type method for p-cyclic linear systems seems to be less straightforward than deriving p-cyclic GMRES. We will not discuss this promising approach further here.

Another approach for solving (4.74) with multiple right hand sides can be obtained by taking $p = M$, as we will see now. The subspace $\hat{\mathcal{K}}^m$ generated by p-cyclic GMRES, with $p = M$, is independent of the values of θ and f_k, if $\gamma_{k,i} \neq 0$. Therefore, all the linear systems for different values of f_k can be solved simultaneously by p-cyclic GMRES. Only the least-squares problem (4.32) differs for different values of f_k, because of the $e^{-2\pi \mathbf{i} f_k T}$ term. For an accurate DAE discretization, the value of M $(= p)$ should be sufficiently large. However, in practice we often see that p-cyclic GMRES converges poorly for large p. For example, for an average problem (4.73) arising in circuit simulation with a modest M, say $M = 100$, about n p-cyclic GMRES iterations may be needed for a sufficiently small residual, while p-cyclic GMRES applied to the reduced system (4.76), with a small p, say $p \leq 8$, may converge very well. In this case $(M = p)$ the cost of orthogonalization and the least-squares problem are very large and p-cyclic GMRES seems to have no advantages over a direct method for (4.73).

For standard linear systems with multiple right-hand sides, block generalizations of standard GMRES can be used, see (among others) [78]. A block generalization of p-cyclic GMRES is also possible.

4.10 Costs for solving the linear system

In this section, we compare the costs of several methods for the M-cyclic linear system (4.1). The costs of the minimal residual methods of Sections 4.2 and 4.3 are presented in Table 4.1. In this table, $K_2 = \text{nnz}(E_{j-1}) + \text{nnz}(L_j) + \text{nnz}(U_j)$ is the number of nonzeros of E_{j-1}, L_j, and U_j, where $L_j U_j = F_j$ is an LU factorization of F_j. Computing a LU factorization $L_j U_j = F_j$ costs K_1 flops. For the circuit simulation problems considered in Section 4.13.1 these costs are relatively small (one LU factorization costs much less than $\mathcal{O}(n^3)$), but not negligible, see also Section 4.13.1. We assume that K_1 and K_2 are independent of j. The costs for solving the p-cyclic system (4.6), instead of (4.1), can be obtained by substituting $M = p$, $K_1 = 0$, and $K_2 = \text{nnz}(B_i)$.

The costs of the matrix-vector products needs some more explanation. These costs consist of:

I. The preprocessing phase: recursions (4.8) and, or (4.10).

II. The iterative process.

III. The post processing phase: compute the missing y_j with recursion (4.2).

Obviously, for GMRES applied to the reduced linear system (4.9) and $x^{(0)}$-GMRES it is possible to combine the recursions (4.8) and (4.10). Note that (4.9) is equal to the $p = 1$ case of the p-cyclic linear system (4.6). This preprocessing phase costs $M - 1$ matrix-vector products of the form $F_j^{-1} E_{j-1} v$. The preprocessing phase of p-cyclic GMRES corresponds only to recursion (4.8), and costs $M - p$ matrix-vector products. For all the methods, the iteration process costs mM matrix-vector products. The post processing phase of p-cyclic GMRES and $x^{(0)}$-GMRES costs $M - p$ matrix-vector products, GMRES applied to (4.9) needs $M - 1$ matrix-vector products.

A direct method for (4.1) is obtained if the reduced system (4.9) is solved by LU factorization, with partial pivoting, instead of GMRES. In this case it is necessary to compute the matrix in (4.9) explicitly. This can be done by computing n matrix-vector products with the standard unit vectors $e_1, \ldots, e_n$. In some cases considerable savings are possible if zero columns of E_M are exploited. Only n_E matrix-vector products (with e_i) are needed to compute the matrix in (4.9) explicitly, if E_M has n_E nonzero columns. Solving (4.9) via LU factorization takes $2/3 n_E^3$ flops. The pre and post processing phase require $M - 1$ matrix-vector products each. The total computational costs for this direct method are

$$M K_1 + 2(n_E M + 2(M-1)) K_2 + 2/3 n_E^3 \text{ flops}. \tag{4.77}$$

This direct approach is relatively expensive compared with the iterative approach because in practice usually $m \ll n_E$. Other direct approaches for (4.1), see for example

Method	[a] GMRES to (4.9)	[b] $x^{(0)}$-GMRES	[c] p-cyclic GMRES
Number of iterations	m	pm	m
Storage requirements (number of nonzeros)			
Arnoldi vectors	mn	pmn	[d] pmn
Nonzeros $\hat{H}^{(m)}$, $\hat{R}^{(m)}$	$m^2/2$	$p^2m^2/2$	$p^2m^2/2$
Computational costs (flops)			
LU factorizations of F_j	MK_1	MK_1	MK_1
matrix-vector products	$2(mM + 2M - 2))K_2$	$2(mM+2M-1-p)K_2$	$2(mM + 2M - 2p)K_2$
Orthogonalization costs	[e] $2m^2n$	[e] $2pm^2n$	[f] $4pm^2n$
QR factorization of $\hat{H}^{(m)}$	[g] $3m^2$	$3p^2m^2$	[h] $2p^3m^2$

Table 4.1: Storage requirements and computational costs of three methods for the periodic steady state problem (4.1). Usually only the leading order terms are displayed, under the assumption that $p \ll m$. [a] See Section 4.2.1. [b] See Section 4.2.2. [c] See Section 4.3. [d] This is $2pmn$ if both $U_i^{(j)}$ and $V_i^{(j)}$ are stored, which is not necessary. [e] Modified Gram-Schmidt orthogonalization. [f] Householder orthogonalization. [g] This is negligible compared with the orthogonalization costs. [h] The approach of Section 4.4.3.

[4], which do not exploit the sparsity of the E_i and F_i matrices require often $\mathcal{O}(Mn^3)$, which may be much more than (4.77).

In Section 4.13.1 we compare the computational costs for several methods and two different linear systems.

4.11 Parallelization of p-cyclic GMRES

In this section we consider the parallel solution of (4.1) by applying p-cyclic GMRES to the reduced p-cyclic linear system (4.6). In order to make an efficient parallelization on a p processor distributed memory computer possible, we assume the following distribution of the M-cyclic system (4.3): processor i contains the matrices $C_{\underline{q}_i}, \ldots, C_{\overline{q}_i}$ and the vectors $c'_{\underline{q}_i}, \ldots, c'_{\overline{q}_i}$. Moreover we assume that $\underline{q}_i - \overline{q}_i$ is constant; this leads to an optimal load balance of the parallel computer. The distribution of c_i, E_i, and F_i in the p-cyclic linear system (4.6) follows from formula (4.4). In p-cyclic GMRES processor i stores b_i and $U_i^{(j+1)}$ and computes the matrix-vector multiplication with B_i.

Another type of parallelism can be obtained by parallelizing the matrix-vector products with E_i and F_i. However, in practice the block size n is often too small for efficient parallelization. Therefore we aim at parallelizability at a block level.

We will consider the parallelization of the following steps, which are needed to solve (4.1).

I. LU factorizations of the diagonal blocks F_i

II. Reducing (4.1) to a p-cyclic linear system (4.6) with recurrence (4.8).

III. Algorithm 4.1, initialization.

IV. Algorithm 4.1, m iterations of p-cyclic GMRES, with an update to the QR factorization of $\hat{H}^{(m)}$ at each iteration, see Section 4.4.3.

V. Solve the transformed least-squares problem (4.41).

VI. Algorithm 4.1, compute the approximate solution $x^{(m)}$.

VII. Recursion (4.2) for computing the missing y_j.

Step I, II, and III are trivially parallelizable without any communication.

In step IV each processor $i\widetilde{-}1$ sends an n-vector to processor i for the operation (*) in Algorithm 4.1: $\hat{v}_i^{(j)} = v_{i\widetilde{-}1}^{(j)}$. Furthermore, communication is needed in step IV in order to update the QR factorization of $\hat{H}^{(m)}$. Two different approaches for this QR factorization are:

A. Compute the same QR factorization of $\hat{H}^{(m)}$ on each processor.

B. Parallelize the QR factorization of $\hat{H}^{(m)}$.

In approach A, solving the least-squares problem costs $2p^3m^2$ flops per processor. Orthogonalization of the $v_i^{(j)}$ vectors costs $4m^2n$ flops per processor Therefore, the computational costs of the QR factorization are negligible if $p^3 \ll 2n$. In approach A, $1/2pm^2$ elements are broadcast by each processor. The costs of this broadcast are negligible with sending m times an n-vector $v_{i\widetilde{-}1}^{(j)}$ from processor $i\widetilde{-}1$ to processor i if $pm \ll 2n$. Approach B, may be of interest if the computational costs and/or communication costs are not negligible.

We will briefly discuss an efficient parallel implementation of Approach B. In this implementation each processor has its own copy of the right-hand side $\hat{d}$. It is not necessary to redistribute the entries of $\hat{H}^{(m)}$ over the processors. In iteration j, each processor broadcasts its own portion of $H_{jj}^{[j-1]}$ and $H_{j+1,j}$ to each other processor. No other communication is necessary for the parallel QR factorization of $\hat{H}^{(m)}$. One synchronization step per iteration is needed, but this synchronization step can be combined with the synchronization which is needed for sending the n-vector $v_{i\widetilde{-}1}^{(j)}$ from processor $i\widetilde{-}1$ to processor i. Each processor can compute its own copy of the orthogonal transformation Q_j because each processor has a copy of $H_{jj}^{[j-1]}$ and $H_{j+1,j}$. Hence, each processor can monitor the convergence of p-cyclic GMRES with $\|d_{j+1}^{[j]}\|$. The amount of communication per processor per iteration is approximately p^2 elements, because $H_{j+1,j}$ has only one nonzero per column. The total amount of communication per processor for m iterations is p^2m. Note that each processor has a copy of the (triangular) diagonal blocks of $H^{[m]}$. The other nonzeros of $H^{[m]}$ have a column wise distribution. Therefore step V requires another p^2m elements to be sent per processor plus m synchronization steps.

No communication is required for step VI. In step VII processor i needs $x_{i\widetilde{-}1}^{(m)}$ in order to compute $y_{\underline{q}_i}, \ldots, y_{\overline{q}_i-1}$.

Note that the communication for the least-squares problem is global, all the other communication is local: from processor $i\widetilde{-}1$ to processor i. For some parallel computer architectures local communication is less expensive than global communication.

The communication overhead of p-cyclic GMRES is usually much smaller than the communication overhead of parallelized standard GMRES. Parallelized standard GMRES with modified Gram-Schmidt orthogonalization needs j synchronization steps in iteration j in order to compute the inner products. The p-cyclic GMRES method (with approach A for the least-squares problem) needs only one synchronization step per iteration.

In order to get some insight into the costs of the parallel case compared with the costs of the sequential case we use a simple cost model, similar to a BSP [86] cost model proposed in [10]. In the cost model, the costs of sending k elements from one processor to another processor, and the costs of performing kg floating-point operations, are equal. The costs of barrier synchronization of the processors are modelled with l flops. For example, for a Cray T3E computer $g \approx 2$ and $l \approx 350$, if $p = 4$, see [48]. With this cost model, the parallel costs of solving (4.1) with approach B for the least-squares system are approximately

$$C_{m,p} = \frac{MK_1 + 2(mM + 2M - 2p)K_2}{p} + 4m^2n + 2p^2m^2 + (2p^2m + mn)g + 2ml\,, \quad (4.78)$$

if $p \geq 2$. These costs should be compared to the costs of solving (4.1) by applying GMRES to the reduced system (4.9), see Table 4.1.

$$C_{m,1} = MK_1 + 2(mM + 2M - 2)K_2 + 2m^2n \quad (4.79)$$

The theoretical speedup of the parallel algorithm is given by

$$S_p = C_{m_1,1}/C_{m_p,p}\,, \quad (4.80)$$

where m_p is the number of p-cyclic GMRES iterations and m_1 is the number of GMRES iterations on the reduced system (4.9). In practice m_p often needs to be slightly larger than m_1 for a sufficiently small residual norm, see Section 4.13.1.

For linear systems (4.1) arising in discretizations of periodic differential equations, the convergence of p-cyclic GMRES is mainly determined by the properties of the underlying differential equation, see also Section 4.6.1. The number of discretization points M has only a very small influence on the convergence of p-cyclic GMRES. This is confirmed by a few (not reported) numerical experiments. A large M leads to an accurate solution of the differential equation. For $M \to \infty$ the theoretical speedup (4.80) is given by

$$S_{M\to\infty,p} = p\frac{m_1 + 2}{m_p + 2}\,,$$

if costs for the LU factorizations of diagonal blocks F_j are neglected. The speedup is larger if these costs are included.

4.12 Cache efficient sequential p-cyclic GMRES

In this section we will show that the p-cyclic GMRES idea can be used to derive a cache efficient sequential method for p-cyclic matrices. With this cache efficient method higher Mflop rates are possible on cache based computer systems, in comparison with p-cyclic GMRES, $x^{(0)}$-GMRES, and GMRES applied to the reduced system (4.9).

Formula (4.21) indicates that $v_i^{(j)}$ depends only on $v_{i-1}^{(j-1)}$ and $\{v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(j-1)}\}$. This allows a different order for the computation of the $v_i^{(j)}$ vectors than that of Algorithm 4.1. This new ordering is presented in Figure 4.6 for the special case $p = 3$. The generalization for other p is straightforward.

Similar to (4.39) we can define a matrix $\widetilde{V}^{(j)}$, for example

$$\widetilde{V}^{(3)} = \begin{bmatrix} v_1^{(1)} & 0 & 0 & 0 & 0 & 0 & v_1^{(2)} & v_1^{(3)} & v_1^{(4)} & 0 & 0 & 0 \\ 0 & v_2^{(1)} & v_2^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & v_2^{(3)} & v_2^{(4)} & v_2^{(5)} \\ 0 & 0 & 0 & v_3^{(1)} & v_3^{(2)} & v_3^{(3)} & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \quad (4.81)$$

1:	$v_1^{(1)}$	**4:**	$v_1^{(2)}$	$v_1^{(3)}$	$v_1^{(4)}$	**7:**	$v_1^{(5)}$	$v_1^{(6)}$	$v_1^{(7)}$	$\cdots$
2:	$v_2^{(1)}$	$v_2^{(2)}$	**5:**	$v_2^{(3)}$	$v_2^{(4)}$	$v_2^{(5)}$	$\cdots$			
3:	$v_3^{(1)}$	$v_3^{(2)}$	$v_3^{(3)}$	**6:**	$v_3^{(4)}$	$v_3^{(5)}$	$v_3^{(6)}$	$\cdots$		

Figure 4.6: Cache efficient computational sequence for the computation of $v_i^{(j)}$. The numbers $1, \ldots, 7$ indicate the order of computation.

The approximate solution of $Ax = b$ is chosen such that $x^{(m)} \in \widetilde{V}^{(m)}$. In example (4.81) this leads to a 4 dimensional search space for x_1, a 5 dimensional search space for x_2, and a 3 dimensional search space for x_3. This shows that the cache efficient method is not a special case of p-cyclic GMRES, although there are a lot of similarities. Similar to p-cyclic GMRES, minimizing $\|b - Ax\|$ with $x \in \widetilde{V}^{(m)}$, leads to a least-squares problem $\min_y \|\widetilde{H}^{(m)}y - \widetilde{d}\|$. The structure of $\widetilde{H}^{(m)}$ is shown in Figure 4.7. A QR factorization of $\widetilde{H}^{(m)}$, which is useful to solve $\min_y \|\widetilde{H}^{(m)}y - \widetilde{d}\|$, can be computed efficiently since $\widetilde{H}^{(m)}$ has a lower bandwidth of only p.

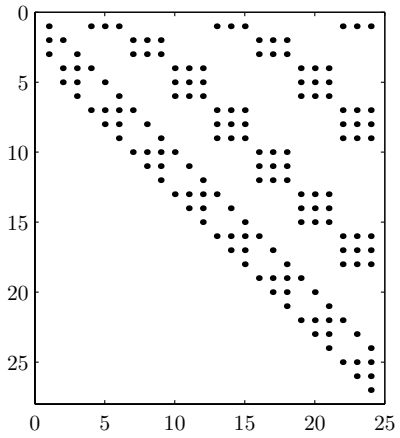


Figure 4.7: Nonzero pattern of $\widetilde{H}^{(6)}$, with $p = 3$.

In order to compute, for example, $v_2^{(3)}$, $v_2^{(4)}$ and $v_2^{(5)}$, the matrix-vector products $B_2v_1^{(2)}$, $B_2v_1^{(3)}$ and $B_2v_1^{(4)}$ are needed. Computing

$$B_2 \left[v_1^{(2)}\, v_1^{(3)}\, v_1^{(4)} \right]$$

can be done in a cache efficient way: each entry of B_2 that is read into cache memory is used $p = 3$ times instead of one time for the methods discussed in the previous sections. The same cache advantage appears if B_i is not explicitly available, but defined as a product of matrices E_j and $F_j^{-1} = U_i^{-1}L_i^{-1}$, as in Section 4.1. Some experiments

on a Sun Ultra 5 computer (270 MHz) showed that computing $B_2 \left[v_1^{(2)}\ v_1^{(3)}\ v_1^{(4)}\ v_1^{(5)} \right]$ (for the p=4 case) at once can be up to twice as fast as computing $B_2 v_1^{(2)}$, $B_2 v_1^{(3)}$, $B_2 v_1^{(4)}$ and $B_2 v_1^{(5)}$ separately. However, the results strongly depend on the computer architecture, the compiler optimization options, and the way in which the sparse matrix times dense vector/vectors operations are coded. Therefore, we will not present more detailed numerical experiments here.

In Section 4.13 we observe that usually the convergence of p-cyclic GMRES slows down with increasing p. On the contrary, the cache advantage increases with increasing p. So, there is a problem dependent pay off between these two effects. For some problems, p has to be sufficiently large because of stability reasons, see Section 4.13.3. The cache efficient approach is always advantageous in this situation, compared to Algorithm 4.1.

4.13 Numerical experiments

4.13.1 Periodic steady state problems arising in circuit simulation

In this section we demonstrate the effectiveness of the p-cyclic GMRES method for a number of (relatively small) circuit simulation test problems. These problems of the form (4.1) arise in periodic steady state analysis of a circuit, see Section 1.3

Table 4.2 shows the problem sizes of the linear systems (4.1). Transient analysis of $\frac{d}{dt}q(x(t)) + j(x(t), t) = 0$ converges very slowly to the periodic solution for these problems. The matrices are taken from the first Newton step of the nonlinear system solution process.

				iterations				
problem	M	n	Mn	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
pss_1	64	22	1408	5	5	6	7	8
pss_2	128	22	2816	9	10	11	11	11
pss_3	64	54	3456	12	12	12	12	12
pss_4	32	904	28928	1	2	2	4	6
pss_5	128	424	54272	34	36	43	52	67

Table 4.2: Convergence of p-cyclic GMRES for periodic steady state test problems. M is the number of time points, n is the number of unknowns per time point. Mn is the dimension of the linear system. The number of iterations needed to satisfy the stopping criterion (4.82) is displayed in the last 5 columns, for different values of p.

Section 4.1 describes how to reduce (4.1) to a p-cyclic system. The integer partition (4.5) is chosen such that $\underline{q}_i - \overline{q}_i = \text{constant}$. This is possible since both p and M are

powers of 2 here. Note that in a parallel environment it is desirable to have $\underline{q}_i - \overline{q}_i =$ constant, for a good load balance. We apply p-cyclic GMRES for the reduced p-cyclic system. The stopping criterion is based on the residual of (4.1), which makes a fair comparison possible between the different values of p. So, at each iteration the recursion (4.2) is used to compute the missing blocks y_i and then the residual of (4.1) is computed. Note that in practice a stopping criterion based on $\|d_{j+1}^{[j]}\|$ is more efficient, see Section 4.4.3. Let $r_{\text{pss}}^{(m)}$ denote the residual of (4.1) after m iterations of p-cyclic GMRES. The stopping criterion is relative to the norm of the right-hand side $c = \left[c_1^T, \ldots, \ldots, c_M^T\right]^T$ from (4.1):

$$\frac{\|r_{\text{pss}}^{(m)}\|}{\|c\|} < 10^{-10}\,. \tag{4.82}$$

The convergence results of p-cyclic GMRES are in Table 4.2. The convergence of problem `pss_5` is plotted in Figure 4.8. We see that usually the speed of convergence decreases

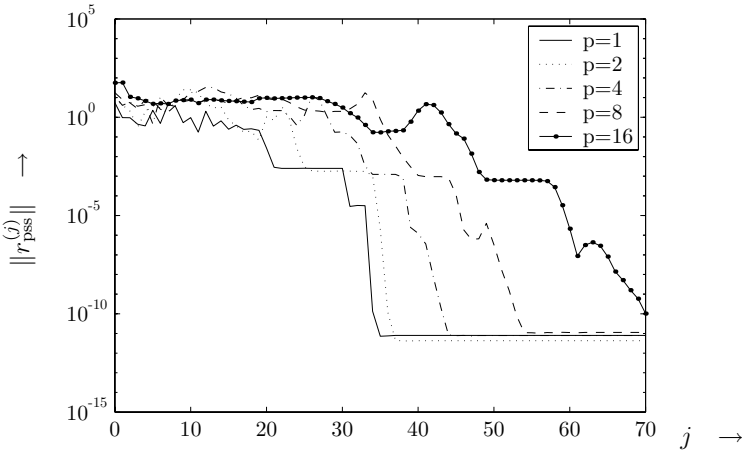


Figure 4.8: Convergence of p-cyclic GMRES for `pss_5`.

with increasing p, but it decreases only slowly relative to the increase in p. In Figure 4.8 we see that the convergence of $\|r_{\text{pss}}^{(j)}\|$ is not monotonic. The p-cyclic GMRES method applied to the p-cyclic linear system (4.6) converges monotonically, but this does not imply that the residual of (4.1) converges monotonically.

The convergence of p-cyclic GMRES depends on the time dependent circuit components. For example, a circuit with only k capacitors and no other time dependent components will usually converge in at most $k+1$ iterations. This can be seen as follows: Discretization of the periodic DAE with backward Euler leads to matrices E_i with $\text{rank}(E_i) \leq k$. The result follows from Theorem 4.4.

Very small capacitors, in comparison with associated resistors in the circuit, will play only a role on a time scale much smaller than the period T. These components will only have a small influence on the convergence. In practice, a circuit often has only

a few slow components and many fast components. This leads to a fast convergence of the iterative methods considered here.

In order to get more insight into the computational costs for solving (4.1) and the parallelizability of p-cyclic GMRES, we compare the costs, derived in Sections 4.10 and 4.11, for the specific cases `pss_4` and `pss_5`. The results are in Table 4.3. The costs of the direct method, see Section 4.10 formula (4.77), are $51.4 \cdot 10^6$ flops for `pss_4` ($n_E = 256$) and $281.2 \cdot 10^6$ flops for `pss_5` ($n_E = 366$). The iterative methods of Table 4.3 are much cheaper.

Method	GMRES to (4.9)		$x^{(0)}$-GMRES		p-cyclic GMRES	
problem	`pss_4`	`pss_5`	`pss_4`	`pss_5`	`pss_4`	`pss_5`
m	1	34	1	34	2	43
Computational costs (10^6 flops)						
LU of F_j,	0.415	2.96	0.415	2.96	0.415	2.96
Matrix $\times$ vector	0.454	24.02	0.439	24.00	0.579	29.99
Orthogonalization	0.002	0.98	0.007	3.92	0.058	12.54
QR of $\hat{H}^{(m)}$	0.000	0.00	0.000	0.06	0.001	0.24
Total	0.870	27.96	0.861	30.94	1.052	45.73
Communication costs (10^6 flops)						
$(2p^2m+mn)g$, see (4.78)					0.004	0.04
$2ml$ (synchronization)					0.001	0.03
Parallel costs (10^6 flops)	$p = 1$				$p = 4$	
Total/p					0.263	11.43
$C_{m,p}$, see (4.79), (4.78)	0.870	27.96			0.268	11.50
Parallel speedup						
S_p, see (4.80)					3.24	2.43

Table 4.3: Estimated computational costs and speedup for periodic steady state problems `pss_4` and `pss_5`, see Table 4.2. The column p-cyclic GMRES is based on 4 processors: $p = 4$.

The communication overhead of p-cyclic GMRES has only a small influence on the estimated speedup. The non optimal estimated speedup of `pss_4` and `pss_5` is mainly caused by the increased number of iterations, and by the increased orthogonalization costs, in comparison with the sequential case. Note that for more accurate discretizations (M large) the orthogonalization costs are relatively smaller.

4.13.2 Convergence upper bound

In this section we will compare the convergence upper bound derived in Section 4.5 with the actual convergence of a test problem.

We consider the problem (4.52) with $p = 4$, $n = 100$,

$$B = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)\,,$$

$\lambda_i = 2/5(1 + \cos(\,(i-1)/(n-1)\,\pi))$ and $\hat{b}_p$ a random vector with entries between -1 and 1, scaled to unit norm. Matrix B has eigenvalues between 0 and $4/5 = \beta$, with a relatively dense spectrum near 0 and $4/5$. This makes the minimization problem (4.53) relatively difficult. We will refer to this problem as problem `mod_0`. For this problem the convergence bound (4.60) is $\|r^{(j)}\| \leq 1/T_{j-p+1}(3/2)$.

The convergence of p-cyclic GMRES applied to `mod_0` and this upper bound are plotted in Figure 4.9. The upper bound has a delay of about 4 ($= p$) iterations compared to the actual convergence. Apparently some bounds used in Section 4.5 are not very sharp. The residual norm reduction factor per iteration is predicted quite well by the upper bound. The convergence is much worse if the right-hand side of (4.52) does not

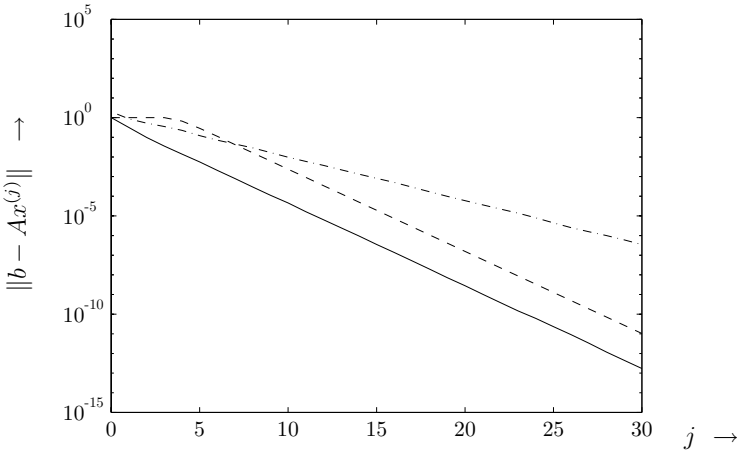


Figure 4.9: Convergence of test problems `mod_0` (———), `mod_1` (- · - · - · -), and upper bound (4.60) for `mod_0` (- - - - -).

have $p - 1$ zero blocks. This is shown in Figure 4.9 by the modified problem `mod_1`, which has a different right-hand side: b_i, $i = 1, \ldots, 4$ is a random vector with entries between -1 and 1, scaled to unit norm.

4.13.3 An unstable problem

In this section we will demonstrate that the p-cyclic GMRES method is very suitable for unstable problems. Free running oscillators, for example, may lead to linear systems which are locally unstable. We define the problem `unstable`; this is the linear system (4.1) with $M = 16$, $n = 100$, $F_i = I$, $E_i = Qe^D Q^T$ (e^D is the matrix exponential of D) $D = \mathrm{diag}(2, -0.04, -0.06, -0.08, \ldots, -1.98, -2)$, $c_i = [1, 0, \ldots, 0]^T/i$, and $Q = I - 2[1, \ldots, 1]^T[1, \ldots, 1]/n$ is an orthogonal matrix. The diagonal matrix D is transformed

with the orthogonal matrix Q in order to have an average rounding error behaviour. This linear system may arise in an exact discretization of $\dot{y} = QDQ^T y + b(t)$, $x(0) = x(16)$ for some particular $b(t)$. Note that a transient analysis with $t \to \infty$ does not converge to the periodic solution, but diverges. Therefore this problem is called an unstable problem.

E_i has one relatively large eigenvalue e^2 with eigenvector $q_1 = Q(:, 1)$ (the first column of Q). This leads to vectors with a very large component in the direction of q_1 in recursions (4.8) and (4.2), if p is small. Moreover, B_i (in the p-cyclic linear system (4.6)) multiplied by an arbitrary vector has a very large component in the q_1 direction if p is small. These large components in the q_1 direction cause a large relative rounding error (in inexact arithmetic) in the other directions, which leads to an inaccurate solution of (4.1). For large p these instability problems are less serious, because the recursions are much shorter. In this case the instability problems are transfered to the reduced least-squares problem (4.32) which is solved in a stable way by a QR factorization. In Figure 4.10, we see that for small p it is not possible to obtain an accurate solution of (4.1).

For comparison we mention the residual norm of two direct methods here: Block Gaussian elimination of $y_1, \ldots, y_{M-1}$ in (4.3) with LU factorization (partial pivoting) of the reduced system leads to a residual norm of $3.2 \cdot 10^{-2}$ for (4.1). An LU factorization of (4.1) with partial pivoting gives a residual norm of $9.3 \cdot 10^{-15}$. This is almost as accurate as p-cyclic GMRES, with $p = 16$, which stagnates at a residual norm of $1.6 \cdot 10^{-15}$. For $p = 1$ the residual of p-cyclic GMRES stagnates at $1.2 \cdot 10^{-1}$. The residual of $x^{(0)}$-GMRES (with modified Gram-Schmidt two times) applied directly to (4.1) stagnates at $4.7 \cdot 10^{-2}$.

It is not remarkable that $x^{(0)}$-GMRES and p-cyclic GMRES with $p = 1$ lead to a residual norm of the same order. For both methods the initial residual norm is $\|\hat{b}_p\| = \|r^{(0)}\| = 1.1 \cdot 10^{13}$. The numerical experiments were performed on a Sun Ultra 5 with a relative machine precision of approximately 10^{-16}. Hence we may not expect that an iterative method reduces the initial residual by more than a factor of 10^{16}. This explains the inaccurate results of $x^{(0)}$-GMRES and p-cyclic GMRES with $p = 1$.

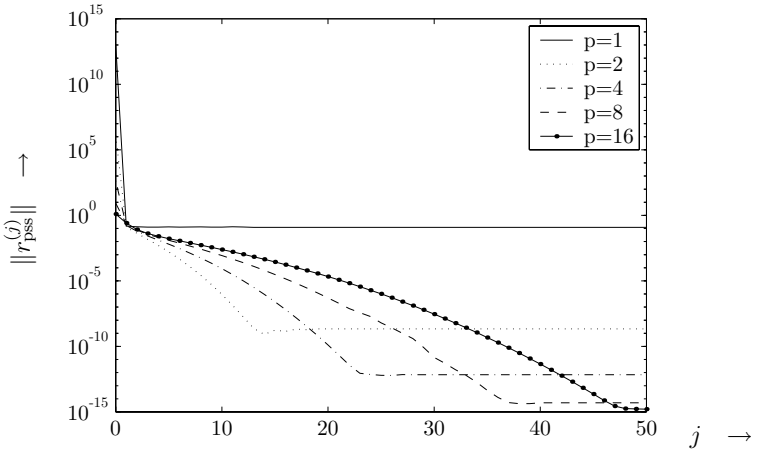


Figure 4.10: Convergence of problem `unstable`. $r_{\rm pss}^{(j)}$ is the residual of (4.1) after j iterations of p-cyclic GMRES

Chapter 5

New GMRES/MINRES-type methods for $P(B)y = c$

Abstract: We propose a new approach for solving linear systems of the form $P(B)y = c$. $P(B)$ is a matrix polynomial in B. The approximate solution $y^{(m)}$ of $P(B)y = c$ is in the Krylov subspace $\mathcal{K}^m(B, c) = \text{span}(c, Bc, \ldots, B^{m-1}c)$. The new approach is based on an augmented block linear system, with blocks of the form $B + \mu_i I$. For unsymmetric B this leads to a method called P(GMRES). We will derive a short recurrence method called $P(\text{MINRES})_{\text{S}}$ for the case where B is a symmetric matrix. If $P(B) = B$, these methods are equivalent to GMRES and MINRES, respectively. The new approach is compared with existing Krylov subspace methods for $P(B)y = c$, and the advantages and disadvantages are illustrated with a few examples.

Keywords: Krylov subspace methods, GMRES and MINRES methods, matrix polynomial, rational matrix function

AMS subject classifications: 65F10

5.1 Introduction

Krylov subspace methods have become very popular over the last few decades for solving linear systems $By = c$. In this chapter we consider generalizations of standard Krylov subspace methods to linear systems of the form

$$P(B)y = c\,. \tag{5.1}$$

Here $P(B)$ is a matrix polynomial in B of degree p and the linear system (5.1) is of dimension n.

To our knowledge there are currently no applications of $P(B)y = c$ in circuit simulation. The linear system $P(B)y = c$ is considered here because an idea more or less similar to the p-cyclic GMRES idea of Chapter 4 is applicable for $P(B)y = c$, as we will see in this chapter. Applications of $P(B)y = c$ arise in, for example, lattice quantum chromodynamics and Tikhonov-Phillips regularization [34]. Linear systems of the form

$P(B)y = c$ may also arise in higher order implicit methods for initial value problems [37, Ch. 8].

A Krylov subspace method for equation (5.1) computes an approximation $y^{(m')} \in \mathcal{K}^{m'}(P(B), c) = \text{span}(c, P(B)c, \ldots, P(B)^{m'-1}c)$, such that $r^{(m')} = c - P(B)y^{(m')}$ is small in some sense. In [24], [34], and [88], it is argued that it may be more efficient (at least in terms of matrix-vector products) to solve (5.1) by building a Krylov subspace $\mathcal{K}^m(B, c) = \text{span}(c, Bc, \ldots, B^{m-1}c)$, with $m \leq pm'$, and choosing $y^{(m)} \in \mathcal{K}(B, c)$ such that $r^{(m)} = c - P(B)y^{(m)}$ is small.

The SLDM method proposed by van der Vorst [88], and later refined by Druskin and Knizhnerman [24], exploit this idea for solving $P(B)y = c$. These methods construct the polynomial for the reduced matrix $H_*^{(m,\square)}$. The SLDM approximation $y^{(m)}$ for (5.1) is:

$$y^{(m)} = \|c\| V_*^{(m)} P(H_*^{(m,\square)})^{-1} e_1 \,. \tag{5.2}$$

Here $H_*^{(m,\square)}$ is defined by $H_*^{(m,\square)} = H_*^{(m)}(1{:}m, 1{:}m)$ (using MATLAB notation), and $V_*^{(m)}$ and the Hessenberg matrix $H_*^{(m)}$ are defined by the standard Arnoldi reduction

$$BV_*^{(m)} = V_*^{(m+1)} H_*^{(m)} \,, \tag{5.3}$$

associated with the Krylov subspace

$$\mathcal{K}^m(B, c) = \text{span}(c, Bc, \ldots, B^{m-1}c) \,,$$

see, e.g., [73, Ch. 6]. The columns $v_*^{(1)}, \ldots, v_*^{(m)}$ of $V_*^{(m)}$ span the Krylov subspace $\mathcal{K}^m(B, c)$. The Arnoldi reduction (5.3) associates the large linear system (5.1) with a relatively small (usually $m \ll n$) linear system: $P(H_*^{(m,\square)})^{-1} e_1$.

The approximate solution $y^{(m)}$ of the SLDM method is not defined if $P(H_*^{(m,\square)})$ is singular. Such a break down may happen even if $P(B)$ and B are both symmetric positive definite matrices. For example

$$B = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \; P(B) = (B - 2I)^2, \text{ and } V_*^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

then $H_*^{(1,\square)} = {V_*^{(1)}}^T B V_*^{(1)} = 2$ and $P(H_*^{(1,\square)}) = 0$. The matrices

$$P(B) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and B are symmetric positive definite. We will prove the next theorem which states that the SLDM method cannot break down before convergence, if certain conditions are satisfied.

Theorem 5.1 *Let B be a real symmetric matrix with minimum eigenvalue $\lambda_{\min}$ and maximum eigenvalue $\lambda_{\max}$, and let P be a polynomial with $P(z) > 0$ on the interval $z \in [\, \lambda_{\min}, \lambda_{\max} \,]$. Then $P(H_*^{(m,\square)})$ is nonsingular, which implies that SLDM cannot break down before convergence.*

Proof The matrix $H_*^{(m,\square)}$ is symmetric and therefore diagonalizable with an orthogonal matrix Q: $H_*^{(m,\square)} = QDQ^T$. The diagonal entries of $D = \mathrm{diag}(\theta_1, \ldots, \theta_m)$ are the Ritz values of B and they satisfy $\theta_i \in [\,\lambda_{\min}, \lambda_{\max}\,]$, for $i = 1, \ldots, m$, see, e.g., [42, Ch. 9]. The following relation holds for $P(H_*^{(m,\square)})$:

$$P(H_*^{(m,\square)}) = QP(D)Q^T = Q\,\mathrm{diag}(P(\theta_1), \ldots, P(\theta_m))\,Q^T\,.$$

The condition $P(z) > 0$ for $z \in [\lambda_{\min}, \lambda_{\max}]$ yields $P(\theta_i) > 0$, and this implies that $P(H_*^{(m,\square)})$ is nonsingular. □

Fiebach [34], [35] has proposed several 'pol' methods for $P(B)y = c$. These methods are different from the SLDM method. The method polGMRES [35] computes an approximate solution $y_{\mathrm{polG}}^{(m)}$ by solving the minimization problem

$$y_{\mathrm{polG}}^{(m)} = \underset{y \in \mathrm{range}(V_*^{(m)})}{\mathrm{argmin}} \|c - P(B)y\|\,. \tag{5.4}$$

The polMINRES method [34] is the short recurrence variant of polGMRES for symmetric matrices B. The polCG method [34] for symmetric positive definite matrices $P(B)$ computes an approximation $y_{\mathrm{polCG}}^{(m)}$ that minimizes the residual in the $P(B)^{-1}$ norm:

$$y_{\mathrm{polCG}}^{(m)} = \underset{y \in \mathrm{range}(V_*^{(m)})}{\mathrm{argmin}} \|c - P(B)y\|_{P(B)^{-1}}\,.$$

Our approach, to be presented in this chapter, is slightly different from all those mentioned before. It computes an approximate solution by minimizing the residual norm of an augmented linear system over a certain subspace. The solution method for the augmented linear system is related to the p-cyclic GMRES method, which was discussed in Chapter 4. The p-cyclic GMRES method uses the block matrix relation (4.24a) to reduce a p-cyclic linear system to a small least-squares problem. A more or less similar reduction is presented here for the augmented block linear system. In order to keep the presentation short, we will refer frequently to the results of Chapter 4. The augmented linear system idea will be introduced in Section 5.2. In Section 5.3, a number of residual norm inequalities are derived in order to be able to compare Fiebach's polGMRES method with our new P(GMRES) method. The P(GMRES) method simplifies for the symmetric case. A short recurrence variant of P(GMRES), called $P(\mathrm{MINRES})_{\mathrm{S}}$, is derived in Section 5.4. The storage requirements and the computational costs are discussed in Section 5.5.

The linear system $f(B)y = c$, with an analytic matrix function $f(B)$, is a generalization of $P(B)y = c$. In (among others) [38], [51], and [88], Krylov subspace methods are discussed for $f(B)y = c$, where f is the exponential operator: $f(B) = e^B$. This equation arises in the context of ordinary differential equations. In Section 5.7 we will show that the P(GMRES) idea can also be used for e^B. Other variants of the P(GMRES) idea are proposed in Section 5.6. One of these variants deals with the equation $R(B)y = c$, where $R(B)$ is a rational matrix function. The chapter is concluded with several numerical experiments.

In order to keep the presentation similar to Chapter 4 we use the notation $V_*^{(m)}$ and $H_*^{(m)}$ for the matrices generated by the Arnoldi process, see (5.3). The matrices $V_*^{(m)}$ and $H_*^{(m)}$ are strongly related to the $V_i^{(m)}$ and $H_i^{(m)}$, $i = 1, \ldots, p$, matrices of Chapter 4, as we will see. We assume that the Arnoldi process does not break down before the m-th iteration, so that $\mathcal{K}^m(B, c) = \text{range}(V_*^{(m)})$. This is not a restriction in practice, because the method proposed in Section 5.2 has converged in that case (this is called a lucky break down), as we will see there.

5.2 An augmented linear system for $P(B)y = c$

Without loss of generality we will assume that the coefficient for B^p in $P(B)$ is 1, so that the polynomial $P(B)$ of degree p can be written as

$$P(B) = (B + \mu_p I) \cdot \ldots \cdot (B + \mu_1 I) + \rho I \,. \tag{5.5}$$

The product $(B+\mu_i I)(B+\mu_j I)$ is commutative and therefore the ordering of the factors $B + \mu_i I$ can be chosen freely. With $\rho = 0$ we have a unique factorization, but $\rho \neq 0$ may be useful if one prefers to avoid complex μ_i. For example, if $p = 2$ we may write $B^2 + \gamma B + \delta I = (B + 0I)(B + \gamma I) + \delta I$. In order to solve $P(B)y = c$ we consider the augmented linear system

$$\begin{bmatrix} I & 0 & 0 & B + \mu_1 I \\ -(B + \mu_2 I) & \ddots & 0 & 0 \\ 0 & \ddots & I & 0 \\ 0 & 0 & -(B + \mu_p I) & \rho I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ c \end{bmatrix} , \tag{5.6}$$

or more compactly

$$Ax = b \,. \tag{5.7}$$

Block Gaussian elimination of $x_1, \ldots, x_{p-1}$ in (5.6) leads to $P(B)x_p = c$, hence $x_p = y$, and we can focus on solving (5.6) instead of $P(B)y = c$.

The linear system (5.6) has quite a similar structure to the p-cyclic linear system discussed in Chapter 4. Instead of choosing different search spaces for each x_i (see Chapter 4), it is natural to choose

$$x_i^{(m)} \in \mathcal{K}^m(B, c) = \text{range}(V_*^{(m)}) \,,$$

because, in contrast to Chapter 4 all the matrices involved are equal. Now x_i can be written as $x_i = V_*^{(m)} z_i$ and for x we have

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} V_*^{(m)} & & \\ & \ddots & \\ & & V_*^{(m)} \end{bmatrix} \begin{bmatrix} z_1 \\ \vdots \\ z_p \end{bmatrix} ,$$

or more compactly

$$x = V^{(m)} z \,.$$

An approximate solution for (5.7) is found by minimizing the norm of the residual of (5.7):

$$x^{(m)} = \underset{x\in\text{range}(V^{(m)})}{\text{argmin}} \|b - Ax\| \tag{5.8a}$$

$$= V^{(m)}\underset{z}{\text{argmin}}\|b - AV^{(m)}z\| \,. \tag{5.8b}$$

In order to solve (5.8b) an expression similar to (4.24a) is useful:

$$\begin{bmatrix} I & 0 & 0 & B+\mu_1 I \\ -(B+\mu_2 I) & \ddots & 0 & 0 \\ 0 & \ddots & I & 0 \\ 0 & 0 & -(B+\mu_p I) & \rho I \end{bmatrix} \begin{bmatrix} V_*^{(m)} & 0 & \dots & 0 \\ 0 & V_*^{(m)} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & V_*^{(m)} \end{bmatrix} =$$

$$\begin{bmatrix} V_*^{(m+1)} & 0 & \dots & 0 \\ 0 & V_*^{(m+1)} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & V_*^{(m+1)} \end{bmatrix} \begin{bmatrix} \tilde{I} & 0 & 0 & H_*^{(m)}+\mu_1\tilde{I} \\ -(H_*^{(m)}+\mu_2\tilde{I}) & \ddots & 0 & 0 \\ 0 & \ddots & \tilde{I} & 0 \\ 0 & 0 & -(H_*^{(m)}+\mu_p\tilde{I}) & \rho\tilde{I} \end{bmatrix}, \tag{5.9a}$$

or in compact notation

$$A\,V^{(m)} = V^{(m+1)}\,H^{(m)} \,. \tag{5.9b}$$

The matrix $\tilde{I}$ is defined in (4.25). Relation (5.9) immediately follows from the Arnoldi reduction (5.3) and $V^{(m)} = V^{(m+1)}\tilde{I}$.

Now the approximate solution can be computed as explained in Section 4.3.2. This leads to a small least-squares problem with solution $z^{(m)}$:

$$\begin{bmatrix} z_1^{(m)} \\ \vdots \\ \vdots \\ z_p^{(m)} \end{bmatrix} = \underset{z}{\text{argmin}}\| \begin{bmatrix} \tilde{I} & 0 & 0 & H_*^{(m)}+\mu_1\tilde{I} \\ -(H_*^{(m)}+\mu_2\tilde{I}) & \ddots & 0 & 0 \\ 0 & \ddots & \tilde{I} & 0 \\ 0 & 0 & -(H_*^{(m)}+\mu_p\tilde{I}) & \rho\tilde{I} \end{bmatrix} \begin{bmatrix} z_1 \\ \vdots \\ \vdots \\ z_p \end{bmatrix} - \begin{bmatrix} 0 \\ \vdots \\ 0 \\ e_1\|c\| \end{bmatrix} \| \,, \tag{5.10a}$$

also denoted by

$$z^{(m)} = \underset{z}{\text{argmin}}\|\tilde{d} \;-\; H^{(m)}z\| \,. \tag{5.10b}$$

The approximate solution $x^{(m)}$ of $Ax = b$ is $x^{(m)} = V^{(m)}z^{(m)}$, so $y^{(m)}$ is

$$y^{(m)} = x_p^{(m)} = V_*^{(m)}z_p^{(m)} \,. \tag{5.11}$$

Methods for the least-squares problem (5.10a) are discussed in Section 4.4. Here we will follow the approach of Section 4.4.3 (Algorithm 4.2), where $H^{(m)}$ is reordered in order to allow for an efficient QR factorization, if p is not too large.

Similar to the methods discussed in the introduction to this chapter, we have derived a method which solves $P(B)y = c$ by generating a Krylov subspace $\mathcal{K}^m(B, c)$ (but our method uses (5.6)). The method has a number of similarities with the standard

GMRES method [76] for $By = c$. Therefore, we will call our method P(GMRES). For the case $p = 1$, P(GMRES) is equivalent to standard GMRES. We will use the Arnoldi process with modified Gram-Schmidt orthogonalization to compute an orthogonal basis for $\mathcal{K}^m(B, c)$. Usually modified Gram-Schmidt orthogonalization will be sufficiently reliable to compute an orthogonal basis for $\mathcal{K}^m(B, c)$ in inexact arithmetic. Appendix A presents a MATLAB implementation of P(GMRES).

An advantage of P(GMRES) over SLDM is that it cannot break down before convergence, if $P(B)$ is nonsingular. This can be seen as follows: block Gaussian elimination of $x_1, \ldots, x_{p-1}$ in (5.6) gives $P(B)x_p = c$, so the matrix A in (5.7) is nonsingular if and only if $P(B)$ is nonsingular. We can use the same argument as in the proof of Theorem 4.2 (Section 4.3.2) to show that $H^{(m)}$ has full rank if A is nonsingular. Therefore, the least-squares problem (5.10) has a unique solution. The P(GMRES) method can only break down if the Arnoldi process with modified Gram-Schmidt orthogonalization breaks down. In this case $\{H^{(m)}\}_{m+1,m} = 0$ and analogous to Theorem 4.3, we may conclude that the solution has converged in this case: a lucky break down. In Section 4.3.2 we argued that p-cyclic GMRES cannot break down if $m \leq n$. It is different here because modified Gram-Schmidt orthogonalization is used instead of Householder orthogonalization.

The P(GMRES) method can easily be adapted so that (5.2) holds (although we do not recommend this if the conditions of Theorem 5.1 are not satisfied). This can be done by computing an approximate solution z^{F} of (5.10a) in a 'FOM'-like way, as discussed in Section 4.4.1, see formula (4.33). Block Gaussian elimination of $z_1^{\mathrm{F}}, \ldots z_{p-1}^{\mathrm{F}}$ in the FOM alternative of (5.10a) leads to (5.2): the SLDM method. This 'FOM'-system (4.33) can be derived by a Ritz-Galerkin approach, see formula (4.35). So, we observe that a Ritz-Galerkin approach may lead to SLDM. Note that Block Gaussian elimination of $z_1^{\mathrm{F}}, \ldots z_{p-1}^{\mathrm{F}}$ may be an unstable process, whereas the QR factorization of $H^{(m)}$ is a stable process.

In numerical experiment 3, Section 5.8, we show that (a symmetric variant of) P(GMRES) may give more accurate results than Fiebach's polMINRES method, if $P(B)$ is ill-conditioned.

5.2.1 Scaling of the block linear system

The P(GMRES) method is not invariant under scaling. We will illustrate the effects of scaling with an example. Consider the linear system $B^2y = c$, with a symmetric (nonsingular) matrix B. This problem can be solved by solving the augmented linear system

$$\begin{bmatrix} I & \alpha B \\ -\alpha B & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ \alpha^2 c \end{bmatrix}, \tag{5.12}$$

with a scaling parameter α. The effects of scaling parameters on augmented linear systems, associated with least-squares problems, are discussed in [5], [11], and [77]. In that case nonsquare matrices are involved with the augmented linear system. Our case is slightly different, because the blocks in (5.12) are square. Moreover, the nonzero

structure of the right-hand side differs.

The scaling parameter α affects the way in which the residuals $r_1^{(m)}$ and $r_2^{(m)}$ are weighted:

$$\frac{1}{\alpha^2}\begin{bmatrix} r_1^{(m)} \\ r_2^{(m)} \end{bmatrix} = \begin{bmatrix} 0 \\ c \end{bmatrix} - \begin{bmatrix} I/\alpha^2 & B/\alpha \\ -B/\alpha & 0 \end{bmatrix}\begin{bmatrix} x_1^{(m)} \\ x_2^{(m)} \end{bmatrix}.$$

Obviously, the approximate solution $x \in \text{range}(V^{(m)})$ with minimum residual norm depends on α. In Section 5.8, experiment 3, the influence of the scaling parameter α is illustrated by a numerical experiment.

The effect of rounding errors also depends on α, as we will now see. It is easy to show that the condition number κ of the augmented matrix is

$$\kappa(\begin{bmatrix} I & \alpha B \\ -\alpha B & 0 \end{bmatrix}) = \frac{1+\sqrt{1+4(\alpha\sigma_{\max}(B))^2}}{-1+\sqrt{1+4(\alpha\sigma_{\min}(B))^2}} \approx \begin{cases} \dfrac{\sigma_{\max}(B)}{\sigma_{\min}(B)} = \kappa(B) & \text{if } |\alpha| \text{ is large,} \\ \dfrac{1}{(\alpha\sigma_{\min}(B))^2} & \text{if } |\alpha| \text{ is small.} \end{cases}$$

The values $\sigma_{\min}(B)$ and $\sigma_{\max}(B)$ denote the smallest and largest singular value of the symmetric matrix B, respectively. The condition number of the augmented linear system (5.12) decreases monotonically, when α increases. The condition number may be smaller than the condition number of B^2, since the condition number of B^2 is $\kappa(B^2) = \kappa(B)^2$. So, a large α leads to a relatively small condition number. This suggests that we should choose a (very) large α in order to reduce the effects of rounding errors in the numerical solution process. However, this does not lead to an accurate solution of $B^2y = c$, because a large α effectively (if rounding errors play a role) decouples the first and the second block rows of (5.12). Obviously, this coupling is an essential part of the solution method. In numerical experiments (not reported) we observed that a very small or very large α may lead to a stagnation of the norm of the true residual $\|c - B^2x_p^{(m)}\|$ at a relatively high level. We advise scaling (5.12) so that $\sigma_{\min}(\alpha B) < 1 < \sigma_{\max}(\alpha B)$. A similar remark applies to the off-diagonal blocks $B + \mu_i I$ of the general case (5.6).

5.3 Convergence of minimal residual methods for $P(B)y = c$

A disadvantage of Fiebach's method polGMRES is that it has a startup phase of $p-1$ iterations. During the startup phase it is not possible to compute approximate solutions of (5.1). In contrast to this, P(GMRES) generates approximations starting with the first iteration. P(GMRES) may also be advantageous after the startup phase. This can be explained as follows. It costs $m-1$ matrix-vector products to generate the Krylov subspace $\mathcal{K}^m(B,c) = \text{span}(V_*^{(m)})$. In order to minimize $\|c - P(B)V_*^{(m)}z\|$ for z, polGMRES has to compute $P(B)V_*^{(m)}$ in some implicit way, this costs p extra matrix-vector products. So, $m+p-1$ matrix-vector products are needed by polGMRES if the solution is taken from the m dimensional Krylov subspace $\mathcal{K}^m(B,c)$. P(GMRES) also needs $m-1$ matrix-vector products to generate the Krylov subspace $\mathcal{K}^m(B,c)$. One

additional matrix-vector product is necessary in order to solve (5.8a). This leads to a total of m matrix-vector products.

The polGMRES method minimizes $\|c - P(B)y^{(m)}\text{polG}\|$ for $y^{(m)} \in \mathcal{K}^m(B, c)$. With the same amount of matrix-vector products P(GMRES) minimizes (5.8a) for $x_i \in \mathcal{K}^{m+p-1}(B, c)$. In practice we often see that the P(GMRES) residual $\|c - P(B)x_p^{(m+p-1)}\|$ is smaller than the polGMRES residual $\|c - P(B)y_{\text{polG}}^{(m)}\|$. Therefore, P(GMRES) may be slightly more efficient in terms of matrix-vector products. This potential advantage is nicely illustrated with the next example:

Example Consider the equation $B^2y = c$, with

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.75 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}, \quad c = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Suppose we want to solve this equation approximately by doing two matrix-vector products. For polGMRES this implies that we seek a solution in the subspace $\mathcal{K}^1(B, c) = \text{span}(c)$. Hence, $y_{\text{polG}}^{(1)} = \alpha c$ and the minimization problem is $\min_\alpha \|c - B^2\alpha c\|$. For the optimal value of α we obtain $\|r_{\text{polG}}^{(1)}\| = \|c - B^2\alpha_{\text{opt}}c\| \approx 0.786$.

The search space of P(GMRES) for x_1 and x_2 is $\mathcal{K}^2(B, c)$: $x_1^{(2)} = \alpha_1 c + \alpha_2 Bc$ and $x_2^{(2)} = \alpha_3 c + \alpha_4 Bc$. This leads to the minimization problem

$$\min_{\alpha_1,\alpha_2,\alpha_3,\alpha_4} \left\| \begin{bmatrix} 0 \\ c \end{bmatrix} - \begin{bmatrix} I & B \\ -B & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 c + \alpha_2 Bc \\ \alpha_3 c + \alpha_4 Bc \end{bmatrix} \right\|.$$

For the optimal values of α_i the norm of the residual is $\|r^{(2)}\| = \|c - B^2(\alpha_{3,\text{opt}}c + \alpha_{4,\text{opt}}Bc)\| \approx 0.266$. This is 3 times smaller than polGMRES' residual norm, for the same number of B evaluations. □

Now we will compare the residuals of polGMRES and P(GMRES). We define the 1 by p block matrix

$$M \equiv \left[\prod_{i=2}^{p}(B + \mu_i I) \quad \ldots \quad \prod_{i=p}^{p}(B + \mu_i I) \quad I \right].$$

Premultiplying the residual $b - Ax$ of (5.6) with M leads to

$$M(b - Ax) = c - P(B)x_p\,,$$

where we made use of the fact that products $(B+\mu_i I)(B+\mu_j I)$ are commutative. Recall that $x^{(m)}$ is defined by (5.8a) and $y_{\text{polG}}^{(m)}$ is defined by (5.4). The following relationship holds:

$$\|c - P(B)y_{\text{polG}}^{(m)}\| \le \|c - P(B)x_p^{(m)}\| \tag{5.13a}$$

$$\le \|M\|\,\|b - Ax^{(m)}\|\,, \tag{5.13b}$$

because $x_p^{(m)} \in \mathcal{K}^m(B,c)$ is a suboptimal solution of the minimization problem (5.4).

In order to derive other insightful inequalities for the residuals, we extend the polGMRES solution $y_{\text{polG}}^{(m)}$ to an approximate solution $x_{\text{polG}}^{(m)}$ of $Ax = b$:

$$x_{\text{polG}}^{(m)} = N y_{\text{polG}}^{(m)}, \quad \text{with} \quad N \equiv \begin{bmatrix} -\prod_{i=1}^{1}(B+\mu_i I) \\ \vdots \\ -\prod_{i=1}^{p-1}(B+\mu_i I) \\ I \end{bmatrix}.$$

The residual associated with $x_{\text{polG}}^{(m)}$ is

$$b - Ax_{\text{polG}}^{(m)} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ c - P(B)y_{\text{polG}}^{(m)} \end{bmatrix}.$$

From $y_{\text{polG}}^{(m)} \in \mathcal{K}^m(B,c)$ it follows that $x_{\text{polG}}^{(m)} \in \mathcal{K}^{m+p-1}(B,c)$. However, the approximate solution $x^{(m+p-1)} \in \mathcal{K}^{m+p-1}(B,c)$ of $Ax = b$ (defined by (5.8a-b)) has a smaller residual than $x_{\text{polG}}^{(m)}$:

$$\begin{aligned} \|b - Ax^{(m+p-1)}\| &\leq \|b - Ax_{\text{polG}}^{(m)}\| \\ &= \|c - P(B)y_{\text{polG}}^{(m)}\|. \end{aligned}$$

Together with (5.13) this leads to

$$\begin{aligned} \|b - Ax^{(m+p-1)}\| &\leq \|c - P(B)y_{\text{polG}}^{(m)}\| && (5.14) \\ &\leq \|c - P(B)x_p^{(m)}\| && (5.15) \\ &\leq \|M\|\,\|b - Ax^{(m)}\|. && (5.16) \end{aligned}$$

From (5.14) we conclude that polGMRES and P(GMRES) have asymptotically the same speed of convergence.

In the remainder of this section we will derive an upper bound for $\|b - Ax^{(m)}\|$ in terms of $P(z)$ and the eigenvalues of B. This analysis is adapted from the analysis of Section 4.5. We assume that B is diagonalizable: $B = W\text{diag}(\lambda_1, \ldots, \lambda_n)W^{-1}$, $D = \text{diag}(\lambda_1, \ldots, \lambda_n)$. Furthermore, we define $\kappa(W) = \|W\|\,\|W^{-1}\|$. The norm of the residual $r^{(m)} = b - Ax^{(m)}$ (with $x^{(m)}$ defined by (5.8a)) can be bounded by

$$\begin{aligned} \|r^{(m)}\| = \|b - Ax^{(m)}\| &\leq \|b - ANQ^{m-p}(B)c\| && (5.17a) \\ &= \|c - P(B)Q^{m-p}(B)c\| && (5.17b) \\ &\leq \|c\|\,\|I - P(B)Q^{m-p}(B)\| && (5.17c) \\ &\leq \|r^{(0)}\|\,\kappa(W)\,\|I - P(D)Q^{m-p}(D)\| && (5.17d) \\ &= \|r^{(0)}\|\,\kappa(W)\,\max_{i=1,\ldots,n}|1 - P(\lambda_i)Q^{m-p}(\lambda_i)|. && (5.17e) \end{aligned}$$

Here Q^{m-p} is an arbitrary matrix polynomial of maximum degree $m-p$. Therefore, $NQ^{m-p}(B)c \in \text{range}(V^{(m)})$. For an optimal polynomial Q^{m-p}, formula (5.17) leads to

$$\frac{\|r^{(m)}\|}{\|r^{(0)}\|} \leq \kappa(W) \min_{Q \in \mathbb{P}^{m-p}} \max_{i=1,\ldots,n} |1 - P(\lambda_i)Q(\lambda_i)| \,. \tag{5.18}$$

For polynomials of the form $P(B) = B^p - I$ and $\lambda_i \in [0, \beta]$ with $\beta < 1$, we can use the results of Section 4.5. For other polynomials P and/or eigenvalues λ_i of B, Conjecture 4.6 may not be valid and the suggested linear combination of Section 4.5:

$$q(z) = 1 - P(z)Q^{m-p}(z) = \sum_{j=0}^{p-1} s_{m-j}(z) d_j \,,$$

may be less suitable. In [34], Fiebach gives an upper bound for an optimization problem of the form (5.18), with $p = 2$, by using Chebyshev polynomials. This leads to a bound in terms of the roots of P and the smallest and largest eigenvalues of B.

For $P(B) = B$ (5.18) reduces to

$$\begin{aligned} \frac{\|r^{(m)}\|}{\|r^{(0)}\|} &\leq \kappa(W) \min_{Q \in \mathbb{P}^{m-1}} \max_{i=1,\ldots,n} |1 - \lambda_i Q(\lambda_i)| \\ &= \kappa(W) \min_{\widetilde{Q} \in \mathbb{P}^m, \widetilde{Q}(0)=1} \max_{i=1,\ldots,n} |\widetilde{Q}(\lambda_i)| \,, \end{aligned}$$

a well known GMRES convergence result [73, Ch. 6].

5.4 The symmetric case: P(MINRES)

For symmetric matrices B the standard Arnoldi reduction $BV_*^{(m)} = V_*^{(m+1)} H_*^{(m)}$ leads to a tridiagonal matrix $H_*^{(m)}$, where $H_*^{(m,\square)}$ is symmetric. The symmetric Lanczos algorithm exploits this symmetry in order to generate a $H_*^{(m)}$ and a $V_*^{(m+1)}$ that satisfy $BV_*^{(m)} = V_*^{(m+1)} H_*^{(m)}$. This leads to a large reduction of the number of inner products and vector updates, which can be exploited in a symmetric variant of P(GMRES). In the remainder of this section, we discuss how the symmetry can be exploited in the other parts of the P(GMRES) method. We will compare the computational costs of the unsymmetric and the symmetric methods for $P(B)y = c$ in Section 5.5. We refer to [73, Ch. 6] for a description of the Lanczos algorithm. In this chapter we assume that B is real. The generalization for the complex Hermitian case is straightforward.

In order to solve the least-squares problem (5.10), $H^{(m)}$ is reordered as described in Section 4.4.3. The reordered least-squares problem is denoted by

$$\min_{\hat{z}} \|\hat{d} \; - \; \hat{H}^{(m)} \hat{z}\| \,. \tag{5.19}$$

It is easily seen that the vector $\hat{d}$ is given by

$$\hat{d}_k = \begin{cases} {v_*^{(1)}}^T c \,, & \text{if } k = p \,, \\ 0 \,, & \text{if } k \neq p \,, \end{cases}$$

where $v_*^{(1)}$ is the first column vector of $V_*^{(m+1)}$. Figure 5.1 shows the nonzero pattern of a matrix $\hat{H}^{(m)}$ and its upper triangular factor of the QR factorization $\hat{H}^{(m)} = \hat{Q}^{(m)}\hat{R}^{(m)}$.

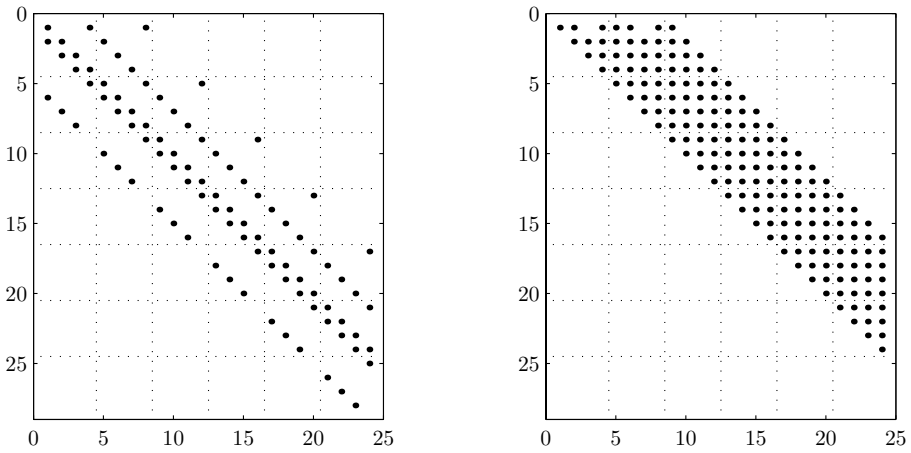


Figure 5.1: left: nonzero pattern of $\hat{H}^{(m)}$ in the least-squares problem (5.19) for $p = 4$ and $m = 6$. right: the nonzero pattern $\hat{R}^{(m)}$

The matrix $\hat{H}^{(m)}$ has a block tridiagonal structure. Therefore, the QR factorization of Section 4.4.3 simplifies because there is no need to apply the orthogonal transformation Ω_j^T to blocks H_{ij} which are zero. Only orthogonal transformations Ω_{j-2}^T, Ω_{j-1}^T and Ω_j^T need to be applied to the column $H_{1j}, \ldots, H_{j+1,j}$. This QR factorization $\hat{H}^{(m)} = \hat{Q}^{(m)}\hat{R}^{(m)}$ leads to a block upper tridiagonal $\hat{R}^{(m)}$ with bandwidth $2p + 1$.

Now we have derived an inexpensive symmetric variant of P(GMRES), which we will call $P(\text{MINRES})_{\text{L}}$. The $_{\text{L}}$ in stands for 'long', in contrast to a 'short' variant which we will discuss in the remainder of this section. MINRES refers to the standard MINRES method for indefinite symmetric linear systems $By = c$, by Paige and Saunders [68].

A disadvantage of $P(\text{MINRES})_{\text{L}}$ is that it is necessary to store the whole matrix $V_*^{(m)}$, because $y^{(m)}$ is computed after the reduced least-squares problem (5.19) is solved. In the following we will show that it is not necessary to store $V_*^{(m)}$ if the computations are rearranged. This leads to the method called $P(\text{MINRES})_{\text{S}}$. In the j-th iteration of $P(\text{MINRES})_{\text{S}}$ the approximate solution $y^{(j-1)}$ is updated to $y^{(j)}$ by using short-term recurrence relations.

We define the matrix $\hat{R}^{(j,\square)} = \hat{R}^{(j)}(1{:}pj, 1{:}pj)$. The approximate solution of (5.7) after j iterations is given by

$$x^{(j)} = \hat{V}^{(j)}\ \hat{R}^{(j,\square)^{-1}}\ \{\hat{Q}^{(j)^T}\tilde{d}\}(1{:}pj)\,.$$

However we are only interested in $y^{(j)} = x_p^{(j)}$, therefore

$$y^{(j)} = x_p^{(j)} = \hat{V}^{(j)}(1 + (p-1)j{:}pj, :)\ \hat{R}^{(j,\square)^{-1}}\ \{\hat{Q}^{(j)^T}\tilde{d}\}(1{:}pj)\,.$$

This can be computed with

$$y^{(j)} = \hat{V}^{(j)}(1 + (p-1)j{:}pj, :) \quad \left(\hat{R}^{(j,\square)^{-1}} \ \{\hat{Q}^{(j)^T} \tilde{d}\}(1{:}pj) \right) , \tag{5.20a}$$

or

$$y^{(j)} = \left(\hat{V}^{(j)}(1 + (p-1)j{:}pj, :) \ \hat{R}^{(j,\square)^{-1}} \right) \ \{\hat{Q}^{(j)^T} \tilde{d}\}(1{:}pj) \, . \tag{5.20b}$$

Formula (5.20a) leads to $P(\text{MINRES})_{\text{L}}$ and formula (5.20b) leads to $P(\text{MINRES})_{\text{S}}$. From the QR factorization of Section 4.4.3 it follows that the vector $\{\hat{Q}^{(j)^T} \tilde{d}\}(1{:}pj)$ grows at each iteration by p new entries, the other elements do not change:

$$\{\hat{Q}^{(j)^T} \tilde{d}\}(1{:}pj) = \begin{bmatrix} \{\hat{Q}^{(j-1)^T} \tilde{d}\}(1{:}p(j-1)) \\ \{\hat{Q}^{(j)^T} \tilde{d}\}(1 + p(j-1){:}pj) \end{bmatrix} . \tag{5.21}$$

Now we consider

$$W^{(j)} \equiv \hat{V}^{(j)}(1 + (p-1)j{:}pj, :) \ \hat{R}^{(j,\square)^{-1}} . \tag{5.22}$$

A recurrence relation for the columns of $W^{(j)}$ can easily be derived from

$$W^{(j)} \hat{R}^{(j,\square)} = \hat{V}^{(j)}(1 + (p-1)j{:}pj, :) \tag{5.23a}$$

$$= \big[\underbrace{0, \ldots, 0, v_*^{(1)}}_{p}, \ \ 0, \ldots, 0, v_*^{(2)}, \ \ \ldots, \ \ 0, \ldots, 0, v_*^{(j)} \big] , \tag{5.23b}$$

by comparing the columns of the left-hand side and the right-hand side in (5.23a). The recurrence relation defined by (5.23a) is a $2p+1$-term recurrence, because $\hat{R}^{(j,\square)}$ is an upper triangular matrix with bandwidth $2p+1$.

In order to see the similarities to MINRES [68], the recurrence can also be seen as a block three-term recurrence. In the j-th iteration only the columns $p(j-3)+1, \ldots, p(j-1)$ of $W^{(j-1)}$ are needed to compute the p new columns $p(j-1)+1, \ldots, pj$ of $W^{(j)}$. These p columns are appended to $W^{(j-1)}$:

$$W^{(j)} = \big[W^{(j-1)} \ \ W^{(j)}(:, p(j-1)+1{:}pj) \big] . \tag{5.24}$$

Formula (5.20b) can be simplified by using the relations (5.21), (5.22) and (5.24). This leads to

$$\begin{aligned} y^{(j)} &= W^{(j)} \ \{\hat{Q}^{(j)^T} \tilde{d}\}(1{:}pj) \\ &= \big[W^{(j-1)} \ \ W^{(j)}(:, p(j-1)+1{:}pj) \big] \begin{bmatrix} \{\hat{Q}^{(j-1)^T} \tilde{d}\}(1{:}p(j-1)) \\ \{\hat{Q}^{(j)^T} \tilde{d}\}(1 + p(j-1){:}pj) \end{bmatrix} \\ &= y^{(j-1)} + W^{(j)}(:, p(j-1)+1{:}pj) \{\hat{Q}^{(j)^T} \tilde{d}\}(1 + p(j-1){:}pj) \, . \end{aligned}$$

We see that only the last p columns of $W^{(j)}$ are needed in order to update the approximate solution from $y^{(j-1)}$ to $y^{(j)}$. This point of view helps to save storage. A MATLAB implementation of $P(\text{MINRES})_{\text{S}}$ is presented in Appendix A.

For $p = 1$ we have that $W^{(j)} = V_*^{(j)} R^{(j)^{-1}}$ and it is easily seen that $P(\text{MINRES})_\text{S}$ reduces to MINRES [68] in this case.

Unfortunately, $P(\text{MINRES})_\text{S}$ needs many (if p is large) more vector updates per iteration than $P(\text{MINRES})_\text{L}$, as we will see in Section 5.5. The main advantage of $P(\text{MINRES})_\text{S}$ over $P(\text{MINRES})_\text{L}$ is the reduced storage requirements (see also Section 5.5).

The rounding error behaviour of $P(\text{MINRES})_\text{L}$ and $P(\text{MINRES})_\text{S}$ are not the same. Sleijpen, van der Vorst, and Modersitzki [79] show that MINRES_L has a better rounding error behaviour than MINRES_S, for ill-conditioned indefinite symmetric linear systems $By = c$. (Obviously MINRES_S refers to the three-term recurrence method and MINRES_S, and MINRES_L refers to the variant which stores all the Lanczos vectors.) We may expect a similar behaviour for $P(\text{MINRES})_\text{S}$ and $P(\text{MINRES})_\text{L}$.

In inexact arithmetic, the orthogonality of the vectors generated by the Lanczos process is often lost due to rounding errors, as the iteration proceeds. Druskin, Greenbaum, and Knizhnerman [23] have shown that convergence of the SLDM method is still possible in spite of this loss of orthogonality. The numerical experiments of Section 5.8 indicate that this is also true for the $P(\text{MINRES})$ methods. A detailed analysis is beyond the scope of this thesis.

5.4.1 Lanczos two times: $P(\text{MINRES})_2$

The storage requirements of $P(\text{MINRES})_\text{L}$ can also be reduced by using an idea proposed by Bergamaschi and Vianello [9] for the SLDM method. The idea is to run the Lanczos process twice. The first run of the Lanczos process is used to compute the tridiagonal matrix $H_*^{(m)}$, and to compute a solution $\hat{z}^{(m)}$ of the least-squares problem (5.19). In the second run of the Lanczos process, $y^{(m)}$ is computed as a linear combination of Lanczos vectors: $y^{(m)} = V_*^{(m)} z_p^{(m)}$ (this is formula (5.11)), where $z_p^{(m)} = \hat{z}^{(m)}(p{:}p{:}pm)$. In this approach it is not necessary to store more than three Lanczos vectors, but twice as many matrix-vector products are needed in comparison with $P(\text{MINRES})_\text{L}$. The Lanczos process of the second run is slightly cheaper than the Lanczos process of the first run, because $H_*^{(m)}$ can be reused. This saves two inner products per iteration. The costs of $P(\text{MINRES})_2$ and other minimal residual methods for $P(B)y = c$ are considered in the next section.

5.5 Costs of minimal residual methods for $P(B)y = c$

In this section we consider the storage requirements and computational costs of the $P(\text{GMRES})$ method and the symmetric variants, polGMRES and polMINRES. These costs are summarized in Table 5.1.

The storage requirements and computational costs may depend slightly on implementation details. For the QR factorization of $\hat{H}^{(m)}$ we assumed that $\hat{H}^{(m)}$ is transformed to upper triangular form by sparse Householder transformations. In the MATLAB implementation of Appendix A the $2p$ by $2p$ orthogonal transformations (which transform

Method	P(GMRES)	P(MINRES)$_{\mathrm{L}}$	P(MINRES)$_{\mathrm{S}}$	P(MINRES)$_2$	polGMRES	polMINRES
Storage requirements (number of nonzeros)						
Arnoldi/Lanczos	$(m+1)n$	$(m+1)n$	$3n$	$3n$	$(m+1)n$	$3n$
Two vectors: y and c	$2n$	$2n$	$2n$	$2n$	$2n$	$2n$
W matrix [a]			[b] $3pn$			$(2p+1)n$
	+ ———	+ ———	+ ———	+ ———	+ ———	+ ———
Total	$(m+3)n$	$(m+3)n$	$(3p+5)n$	$5n$	$(m+3)n$	$(2p+6)n$
Nonzeros $\hat{H}^{(m)}$, $\hat{R}^{(m)}$	$p^2m^2/2$	$3p^2m$	$3p^2m$	$3p^2m$	$pm^2/2$	p^2m
Computational costs (flops)						
Matrix×vector	$2mK$	$2mK$	$2mK$	$4mK$	$2mK$	$2mK$
Arnoldi/Lanczos	$2m^2n$	$9mn$	$9mn$	$14mn$	$2m^2n$	$9mn$
Computing y [c]	$2mn$	$2mn$	$4p^2mn$	$2mn$	$2mn$	[d] $4pmn$
QR of $\hat{H}^{(m)}$	$2p^3m^2$	$8p^3m$	$8p^3m$	$8p^3m$	[e] $1/3pm^3$	[e] $15p^2m$

Table 5.1: Storage requirements and computational costs for m iterations of several minimal residual methods for $P(B)y = c$. The number of nonzeros of B is $K = \mathrm{nnz}(B)$. Often only the leading order terms are displayed, under the assumption that $p \ll m$. [a] The matrix W stores the vectors which are needed for the short recurrences of P(MINRES)$_{\mathrm{S}}$ and polMINRES. [b] It is possible to reduce the $3p$ to $2p+1$ if vectors which are not needed any more are removed immediately from memory. [c] For P(GMRES), P(MINRES)$_{\mathrm{L}}$, and polGMRES, y is computed as a linear combination of Lanczos or Arnoldi vectors. The P(MINRES)$_{\mathrm{L}}$ and polMINRES methods use short recurrences in order to compute y. [d] If short recurrences are used to compute y, other variants are also possible. [e] The flops for constructing $\hat{H}^{(m)}$ are included.

$\hat{H}^{(m)}$ to upper triangular form) are computed explicitly; this is more expensive but easier to program.

For Table 5.1 we assumed that the coefficients μ_i and ρ in (5.6) are real. Complex coefficients lead to higher costs. A reduction of the costs is possible if the coefficients μ_i occur in complex conjugate pairs, as we will see later in this section. We see that $P(\text{MINRES})_\text{S}$ is always more expensive than $P(\text{MINRES})_\text{L}$ in terms of flops. For large p, $P(\text{MINRES})_\text{S}$ is relatively very expensive and in that case $P(\text{MINRES})_\text{S}$ is only of interest if $3p+5$ vectors can be stored in computer memory, but $m+3$ vectors cannot. The $P(\text{MINRES})_2$ method is cheaper than $P(\text{MINRES})_\text{S}$ if one matrix-vector product is cheaper than $4p^2n$ flops. The $P(\text{MINRES})_2$ method always needs less storage than $P(\text{MINRES})_\text{S}$ and $P(\text{MINRES})_\text{L}$.

For standard GMRES, the costs of storing and factorizing the Hessenberg matrix are usually negligible. For large p and a relatively small n this is not necessarily true for P(GMRES), as we see in Table 5.1.

The costs of Fiebach's methods polGMRES and polMINRES increase less fast than that of P(GMRES) and $P(\text{MINRES})_\text{S}$, as p increases. This suggests that, for large p, it may be beneficial to mix Fiebach's approach with our approach, in order to mix the advantages and the disadvantages of both approaches. Suppose, for example, that p is an even number, then block Gaussian elimination of the odd numbered unknowns $x_1,\ x_3,\ \ldots,\ x_{p-1}$ in (5.6) leads to a $p/2$ by $p/2$ block linear system with blocks of the form $(B+\mu_i I)(B+\mu_{i+1}I) = B^2+(\mu_i+\mu_{i+1})B+\mu_i\mu_{i+1}I$, for $i = 1,\ 3,\ ,p-1$. For the blocks $B^2+(\mu_i+\mu_{i+1})B+\mu_i\mu_{i+1}I$ we can derive a relationship similar to (5.3):

$$\begin{aligned}(B^2+(\mu_i+\mu_{i+1})B+\mu_i\mu_{i+1}I)V_*^{(m)}&\\ = V_*^{(m+2)}(H_*^{(m+1)}H_*^{(m)}+(\mu_i+\mu_{i+1})I_{m+2,m+1}H_*^{(m)}+\mu_i\mu_{i+1}I_{m+2,m})&\,.\end{aligned}$$

The generalization of (5.9b) is $AV^{(m)} = V^{(m+2)}H^{(m)}$, where $H^{(m)}$ has $m+2$ by m blocks.

Other mixed variants of Fiebach's approach and our approach can be obtained by elimination of other combinations of unknowns from the set of unknowns $\{x_1,\ldots,x_{p-1}\}$. However, note that block Gaussian elimination may be unstable, see numerical experiments 3 and 5 in Section 5.8.

Elimination of the odd numbered unknowns has an attractive side effect if the μ_i occur in complex conjugate pairs: $\bar{\mu}_i = \mu_{i+1}$ for $i = 1,\ 3,\ ,p-1$. In that case $(B+\mu_i I)(B+\mu_{i+1}I) = B^2+(\mu_i+\bar{\mu}_i)B+\mu_i\bar{\mu}_i I$, $i = 1,\ 3,\ ,p-1$, is a polynomial with real coefficients. So, complex arithmetic can be avoided.

5.6 Other variants of P(GMRES)

In this section we propose some other ideas for solving $P(B)y = c$ or related problems.

5.6.1 The matrix polynomial $P(B) = \gamma_0 I+\gamma_1 B+\ldots+\gamma_p B^p$

Up until now we have assumed that $P(B)$ has the form $P(B) = (B+\mu_p I)\cdot\ldots\cdot(B+\mu_1 I)+\rho I$. It is always possible to write a polynomial $P(B) = \gamma_0 I+\gamma_1 B+\ldots+\gamma_{p-1}B^{p-1}+1B^p$ in

such a form, but this may lead to complex μ_i. Moreover, the problem of finding $\mu_1, \ldots, \mu_p$ for given scalars $\gamma_0, \ldots, \gamma_{p-1}$ and parameter ρ, may be ill-conditioned (although one may try to avoid ill-conditioning by choosing a good ρ). On the other hand, computing γ_i for a given polynomial $P(B) = (B + \mu_p I) \cdot \ldots \cdot (B + \mu_1 I) + \rho I$ may lead to very large γ_i if p is large and $|\mu_j|$ is modest. For example, $B = \text{diag}(9,\ 9.001,\ 9.002,\ \ldots,\ 9.1)$ and $P(B) = (B - 10I)^{40}$. These polynomials $P(B)$ with large γ_i are not suitable for the numerical method proposed in this section. We conclude that the best form for $P(B)$ is problem dependent.

Now we will propose a real arithmetic method for $P(B)y = c$, where $P(B)$ is an arbitrary polynomial of the form

$$P(B) = \gamma_0 I + \gamma_1 B + \ldots + \gamma_p B^p\,, \tag{5.25}$$

with $\gamma_i \in \mathbb{R}$ and $B \in \mathbb{R}^{n\times n}$.

We consider the linear system

$$\begin{bmatrix} I & 0 & 0 & \gamma_p B \\ -B & \ddots & 0 & \vdots \\ 0 & \ddots & I & \gamma_2 B \\ 0 & 0 & -B & \gamma_1 B + \gamma_0 I \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{p-1} \\ x_p \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ c \end{bmatrix}\,, \tag{5.26}$$

abbreviated by $Ax = b$. Block Gaussian elimination of $x_1, \ldots, x_{p-1}$ in (5.26) leads to $P(B)x_p = c$. Hence, the problem $P(B)y = c$ is solved by the block matrix equation (5.26). It is possible to derive other block matrix equations associated with $P(B)y = c$, but we restrict ourselves to (5.26).

Analogous to (5.9), an expression $AV^{(m)} = V^{(m+1)}\,H^{(m)}$, can be derived. The matrix $H^{(m)}$ can be obtained by replacing the block entries in the block matrix A of (5.26): $I \leftarrow \tilde{I}$, $B \leftarrow H_*^{(m)}$ and the square zero matrices 0 are replaced by another zero matrix 0 of suitable dimensions. Now the derivation of an iterative method which solves $P(B)y = c$ approximately (with P(B) of the form (5.25)), can be done in a similar way to P(GMRES). The reordering of $H^{(m)}$, described in Section 4.4.3, again leads to a block Hessenberg matrix $\hat{H}^{(m)}$. The costs of methods based on (5.26) are approximately equal to the costs presented in Section 5.5. Only lower order terms may differ, as we will see now for the symmetric case.

If B is symmetric, then $\hat{H}^{(m)}$ is a block tridiagonal matrix. The QR factorization of $\hat{H}^{(m)}$ leads to $\mathcal{O}(mp)$ more fill-in entries in the block upper tridiagonal matrix $\hat{R}^{(m)}$ than in Section 5.4. Therefore, the short recurrence method for (5.25) needs $\mathcal{O}(pm)$ more vector updates than the short recurrence method $P(\text{MINRES})_{\text{S}}$.

5.6.2 A QMR-type method for $P(B)y = c$

In this section we will show that it is possible to derive a QMR-type method for $P(B)y = c$, based on the unsymmetric Lanczos process. The QMR method [36] is a short recurrence Krylov subspace method for unsymmetric linear systems $By = c$. The method is based on the unsymmetric Lanczos process, see e.g., [73, Ch. 7]. The

unsymmetric Lanczos process constructs an $m+1$ by m tridiagonal matrix $T_*^{(m)}$ and matrices $V_*^{(j)} = [v_*^{(1)}, \ldots, v_*^{(j)}]$, with $v_*^{(1)} = \beta c$, such that relationship

$$BV_*^{(m)} = V_*^{(m+1)} T_*^{(m)} , \tag{5.27}$$

holds. The matrix $V_*^{(m+1)}$ is not orthogonal in general, but it is biorthogonal with respect to a matrix $W_*^{(m+1)}$: ${W_*^{(m+1)}}^T V_*^{(m+1)} = I$. We refer to [73, Ch. 7] for a description of the unsymmetric Lanczos process. Here we will only use the relationship (5.27).

With equation (5.27) we can derive a relationship analogous to (5.9). The differences from (5.9) are that $H_*^{(m)}$ is replaced by $T_*^{(m)}$: $AV^{(m)} = V^{(m+1)} T^{(m)}$, and that $V^{(j)}$ is not orthogonal. We seek an approximate solution of (5.7) of the form $x^{(m)} = V^{(m)} z$. The residual is

$$\begin{aligned} b - Ax^{(m)} &= b - AV^{(m)} z \\ &= V^{(m+1)} (\tilde{d} - T^{(m)} z) . \end{aligned}$$

The matrix $V^{(m+1)}$ is not orthogonal, so, in general $\|b - Ax^{(m)}\| \neq \|\tilde{d} - T^{(m)} z\|$, but a small $\|\tilde{d} - T^{(m)} z\|$ may lead to a small $\|b - Ax^{(m)}\|$. Hence, a suitable approximate solution of (5.7) is given by

$$x = V^{(m)} \operatorname*{argmin}_z \|\tilde{d} \; - \; T^{(m)} z\| . \tag{5.28}$$

The least-squares problem in formula (5.28) can be solved by a QR factorization of $T^{(m)}$. Reordering $T^{(m)}$ as described in Section 4.4.3 leads to a block tridiagonal matrix $\hat{T}^{(m)}$ that has the same nonzero structure as the matrix $\hat{H}^{(m)}$ used in P(MINRES), see Figure 5.1.

Similar to Section 5.4 it is possible to derive short recurrences in order to update $x^{(m)}$ at every iteration. This leads to a method that needs a fixed amount of memory and flops per iteration. Note that the memory requirements and the orthogonalization costs of methods based on the Arnoldi process increase in successive iterations.

We do not claim that the idea presented in this section leads to a reliable method for $P(B)y = c$. Our only aim was to show that it is in principle possible to combine the idea of Section 5.2 with the unsymmetric Lanczos process. Note that it is well known that the unsymmetric Lanczos process may suffer from (near) break downs, and so may this approach.

5.6.3 $R(B)y = c$ with a rational matrix function $R(B)$

The SLDM method can also be used for problems $f(B)y = c$, where f is an analytic function: $y^{(m)} = \|c\| V_*^{(m)} f(H_*^{(m,\square)})^{-1} e_1$. The P(GMRES) method cannot be generalized to general analytic matrix functions f, because the block linear system (5.6) has to be finite dimensional in practical applications. However, the linear system $f(B)y = c$ can be solved approximately with P(GMRES) or a P(GMRES)-like method by approximating the analytic function f by a polynomial or rational function of suitable order, for example a truncated Taylor series or a Padé approximation.

In order to keep the computational costs and the storage requirements of this approach low, a small p is desirable, see Section 5.5. Therefore, Chebyshev or rational Chebyshev approximations may be more suitable in practice. Padé and rational Chebyshev approximations lead to a rational matrix function $R(B)$.

We will briefly indicate in which sense $R(B)$ should approximate $f(B)$. Suppose that $f(B)y = c$ is solved approximately by $R(B)y = c$, then the error is

$$\begin{aligned} x_R - x_f &= R(B)^{-1}c - f(B)^{-1}c \\ &= (R(B)^{-1} - f(B)^{-1})c\,, \end{aligned}$$

and the residual is

$$\begin{aligned} r = c - f(B)x_R &= c - F(B)R(B)^{-1}c \\ &= (I - F(B)R(B)^{-1})c\,. \end{aligned}$$

Therefore, we suggest choosing the approximating rational function $R(z)$ so that

$$\max_{z\in\Omega} |R(z)^{-1} - f(z)^{-1}|$$

is small, or so that

$$\max_{z\in\Omega} |1 - f(z)/R(z)|$$

is small, where Ω denotes the field of values of C:

$$\Omega = \{z \in \mathbb{C} \mid z = \frac{x^H Ax}{x^H x}\,, x \in \mathbb{C}^n\}\,.$$

For the SLDM method this condition is reasonable, because the eigenvalues of $H_*^{(m,\square)}$ are in Ω and therefore $R(H_*^{(m,\square)}) \approx f(H_*^{(m,\square)})$. For P(GMRES) this condition is also reasonable, because SLDM and P(GMRES) are more or less related to each other, see the last paragraph of Section 5.2. In the next section, the error in the solution is analysed for the special case $f(B) = e^B$, where e^B is approximated by a truncated Taylor series.

In the remainder of this section, we will show that the augmented linear system idea, proposed in Sections 5.2 and 5.6.1, is also applicable for linear systems of the form $R(B)y = c$. We will assume that the rational matrix function $R(B)$ has the following form:

$$R(B) = P^p(B)\big(Q^{p-1}(B)\big)^{-1}\,, \tag{5.29}$$

with the polynomials $P^p(B)$ and $Q^{p-1}(B)$:

$$\begin{aligned} P^p(B) &= (\mu_{1p}I + \nu_{1p}B)(\mu_{21}I + \nu_{21}B)\cdot\ldots\cdot(\mu_{p,p-1}I + \nu_{p,p-1}B)\,, \\ Q^{p-1}(B) &= (\mu_{11}I + \nu_{11}B)\cdot\ldots\cdot(\mu_{p-1,p-1}I + \nu_{p-1,p-1}B)\,. \end{aligned}$$

The only restriction for the coefficients μ_{ij} and ν_{ij} is that $P^p(B)$ and $Q^{p-1}(B)$ should be nonsingular. In order to solve $R(B)y = c$ we introduce the block linear system

$$\begin{bmatrix} \mu_{11}I + \nu_{11}B & 0 & 0 & \mu_{1p}I + \nu_{1p}B \\ -(\mu_{21}I + \nu_{21}B) & \ddots & 0 & 0 \\ 0 & \ddots & \mu_{p-1,p-1}I + \nu_{p-1,p-1}B & 0 \\ 0 & 0 & -(\mu_{p,p-1}I + \nu_{p,p-1}B) & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ c \end{bmatrix}\,. \tag{5.30}$$

Block Gaussian elimination of $x_1, \ldots, x_{p-1}$ in (5.30) leads to $R(B)x_p = c$. Obviously, an iterative method can be derived for $R(B)y = c$, in a similar way to that in Sections 5.2 and 5.6.1.

The linear system $P^p(B)(Q^{p-1}(B))^{-1}y = c$ can also be solved by computing $\tilde{c} = Q^{p-1}(B)c$ (this costs $p-1$ matrix-vector products), followed by solving $P^p(B)y = \tilde{c}$ with P(GMRES). However, $P^p(B)(Q^{p-1}(B))^{-1}y = c$ may converge much faster than $P^p(B)y = \tilde{c}$, for example if $P^p(B)(Q^{p-1}(B))^{-1} \approx I$.

In the symmetric case, $R(B)y = c$ can be solved with short recurrences, analogous to Section 5.4.

5.7 Example: $e^B y = c$

In this example we show that the approach of the previous sections is applicable for the linear system $e^B y = c$. Krylov subspace methods for $e^B y = c$ are studied by (among others) [24], [38], [51], [75], and [88] Except for the corrected schemes in [75], all these methods are based on the SLDM approach:

$$y^{(m)} = \|c\| V_*^{(m)} (e^{H_*^{(m,\square)}})^{-1} e_1 \,.$$

Note that $e^{H_*^{(m,\square)}}$ is always nonsingular, so that this is not a reason to apply the idea of the previous sections to $e^B y = c$. However the case $e^B y = c$ nicely illustrates the possibilities of the P(GMRES) method and its variants.

The equation $e^B y = c$ is often associated with ordinary differential equations [38] and [51]. The relationship between $e^B y = c$ and ODEs can be illustrated by the ODE

$$u' = -Bu, \quad u(0) = u_0 \,. \tag{5.31}$$

The solution of (5.31) is $u(t) = e^{-Bt}u_0$ and the solution $u_1 = u(1)$ can be found by solving the linear equation $e^B u_1 = u_0$.

The method proposed in this example is based on the p by p block linear system

$$\begin{bmatrix} I & 0 & \ldots & \ldots & 0 & I/(p-1) \\ -B/(p-2) & \ddots & \ddots & & \vdots & \vdots \\ 0 & \ddots & I & \ddots & \vdots & \vdots \\ \vdots & \ddots & -B/2 & I & 0 & I/2 \\ \vdots & & \ddots & -B & I & I \\ 0 & \ldots & \ldots & 0 & -B & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ c \end{bmatrix} . \tag{5.32}$$

Block Gaussian elimination of $x_1, \ldots, x_{p-1}$ leads to $\widetilde{e^B} x_p = c$, where

$$\widetilde{e^B} \equiv I + B + \frac{B}{2!} + \ldots + \frac{B^{p-1}}{(p-1)!} \,,$$

is the $p-1$-st order truncated Taylor series of e^B. Analogous to Sections 5.2 and 5.6, it is obvious how to derive an iterative method for (5.32). Note that the block Gaussian

elimination process may be unstable if B has large eigenvalues. Nevertheless, x_p may still be a good approximation of $e^{-B}c$ in finite arithmetic, as we will see in experiment 5 of Section 5.8.

In order to analyse the method we will make the following assumption: B is a symmetric matrix with eigenvalues $\lambda_i \geq 0$. This is a reasonable assumption for the case where B is associated with an ordinary differential equation with non-increasing solution components. Moreover, by scaling the problem $e^B y = c$ it is always possible to obtain positive eigenvalues, because the scaling of $e^B y = c$ shifts the eigenvalues of B: $e^\sigma I = e^{\sigma I}$ and therefore $e^{\sigma I} e^B y = e^{\sigma I} c$ is equivalent to $e^{\sigma I + B} y = e^\sigma c$.

Now we will derive an upper bound for the norm of the error in $y^{(m)}$ in terms of the norm of the residual $r^{(m)} = b - Ax^{(m)}$. Note that $\|r^{(m)}\|$ can be computed very efficiently during the iteration process, see Section 4.4.3. In order to derive the upper bound we use the fact that the exact solution of $e^B y = c$ is $y = e^{-B}c$ and that the exact solution of $Ax = b$ is $x = A^{-1}b$, with $x_p = \widetilde{e^B}^{-1} c$. Recall that the approximate solution of $\widetilde{e^B} y = c$ at the m-th iteration is $y^{(m)} = x_p^{(m)}$. The norm of the error is bounded by

$$\|x_p^{(m)} - e^{-B}c\| = \|x_p^{(m)} - \widetilde{e^B}^{-1} c + (\widetilde{e^B}^{-1} - e^{-B})c\| \tag{5.33a}$$

$$\leq \|x_p^{(m)} - x_p\| + \|\widetilde{e^B}^{-1} - e^{-B}\| \, \|c\| \, . \tag{5.33b}$$

The vectors $x_p^{(m)}$ and x_p are one block of the larger vectors $x^{(m)}$ and x:

$$\begin{aligned} \|x_p^{(m)} - x_p\| &\leq \|x^{(m)} - x\| \\ &= \|A^{-1}(Ax^{(m)} - b)\| \\ &\leq \|A^{-1}\| \, \|r^{(m)}\| \, . \end{aligned}$$

Matrix B is diagonalizable with an orthogonal matrix Q_*: $B = Q_* \mathrm{diag}(\lambda_1, \ldots, \lambda_n) Q_*^T$, $D = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$. We define the block diagonal matrix $Q = \mathrm{diag}(Q_*, \ldots, Q_*)$. The matrix $Q^T A Q$ has n uncoupled p by p submatrices $M_{\lambda_1}, \ \ldots, \ M_{\lambda_n}$ with

$$M_\gamma = \begin{bmatrix} 1 & 0 & \ldots & 0 & 1/(p-1) \\ -\gamma/(p-2) & \ddots & \ddots & \vdots & \vdots \\ 0 & \ddots & 1 & 0 & 1/2 \\ \vdots & \ddots & -\gamma & 1 & 1 \\ 0 & \ldots & 0 & -\gamma & 1 \end{bmatrix} , \tag{5.34}$$

when a suitable reordering is applied to $Q^T A Q$. We use Q to simplify the norm of A^{-1}:

$$\begin{aligned} \|A^{-1}\| &= \|Q^T A^{-1} Q\| \\ &= \|(Q^T A Q)^{-1}\| \\ &= \max_{\lambda_i} \|M_{\lambda_i}^{-1}\| \\ &\leq \sup_{\gamma \geq 0} \|M_\gamma^{-1}\| \, . \end{aligned}$$

Numerical experiments with different values of γ and p lead to the following conjecture:

Conjecture 5.2 *The 2-norm of M_γ^{-1}, with M_γ defined by (5.34), is bounded:*

$$\sup_{\gamma \geq 0} \|M_\gamma^{-1}\| \leq p - 1 \,, \quad \textit{if} \;\; p \geq 3 \,.$$

A much weaker result, $\sup_{\gamma \geq 0} \|M_\gamma^{-1}\| \leq \sqrt{p} p^2$, for $p \geq 2$, can be proved by deriving upper and lower bounds for each vector entry s_i in $s = M_\gamma^{-1} t$, with $-1 \leq t_i \leq 1$.

Now we will simplify the term $\|\widetilde{e^B}^{-1} - e^{-B}\|$ in (5.33b).

$$\begin{aligned}
\|\widetilde{e^B}^{-1} - e^{-B}\| &= \| \left(\sum_{i=1}^{p-1} B^i/i! \right)^{-1} - e^{-B} \| \\
&= \| \left(\sum_{i=1}^{p-1} D^i/i! \right)^{-1} - e^{-D} \| \\
&= \max_j | \left(\sum_{i=1}^{p-1} \lambda_j^i/i! \right)^{-1} - e^{-\lambda_j} | \\
&\leq \max_{z \geq 0} |\widetilde{e^z}^{-1} - e^{-z}| \equiv \xi_p
\end{aligned}$$

The value of ξ_p indicates how accurate the approximation of e^{-z}, by the inverse of a truncated Taylor series, is. Computing ξ_p for different values of p is straightforward. For example, $\xi_{20} = 1.6 \cdot 10^{-7}$ and $\xi_{30} = 1.3 \cdot 10^{-10}$. We found that

$$\xi_p \leq 0.25 \cdot 0.50^p, \quad \text{for} \;\; 10 \leq p \leq 80 \,.$$

Note that a value ξ_p smaller than the machine precision is not of practical interest because of rounding errors. Therefore, only $p \leq 51$ is of interest when standard IEEE 754 double precision arithmetic is used (64 bits for one floating-point number).

Combining the previous results gives an upper bound for the error norm in terms of $\|r^{(m)}\|$:

$$\|y^{(m)} - e^{-B} c\| \leq (p-1)\|r^{(m)}\| + 0.25 \cdot 0.50^p \|c\| \,, \quad 10 \leq p \leq 80 \,, \tag{5.35}$$

assuming Conjecture 5.2. The error bound (5.35) may be useful for a stopping criterion of the iterative method. In Section 5.8 we will give an illustration of (5.35) by a numerical experiment.

Three obvious variants (see Sections 5.2 and 5.4) of the method discussed in this section are: e^{GMRES} for the unsymmetric case based on Arnoldi, $e^{\mathrm{MINRES}_{\mathrm{S}}}$ with short $(2p+1)$ term recurrences for the symmetric case, based on the symmetric Lanczos process, and $e^{\mathrm{MINRES}_{\mathrm{L}}}$ for the symmetric case without short recurrences, based on the symmetric Lanczos process. The computational costs and storage requirements of these methods are nearly equal to the corresponding methods mentioned in Section 5.5.

For unsymmetric B the eigenvalues are not necessarily on the positive real axis. Outside the positive real axis the truncated Taylor series may be less accurate, both in exact and inexact arithmetic. From calculus we know that for $p \to \infty$ the $p-1$-st order truncated Taylor series of e^z converges to e^z for all $z \in \mathbb{C}$. However, from numerical

mathematics we know that in inexact arithmetic this may not be true, for example, $1 + (-20) + (-20)^2/2! + (-20)^3/3! + \ldots$ does not converge to e^{-20}, due to cancellation. Here, the linear system (5.32) may be ill conditioned if B has eigenvalues far away from the positive real axis $[0, \infty[$. Numerical experiments (not reported) indicate that $\widetilde{e^z}^{-1}$ is a good approximation for e^{-z} in the complex plane around the positive real axis, if p is sufficiently large, even in inexact arithmetic. Recall that scaling the problem shifts the eigenvalues of B. This can be used to obtain a suitable eigenvalue distribution for B.

5.7.1 The function $\varphi(z) = (e^z - 1)/z$

The function $\phi(z) = (e^z - 1)/z$ arises in the context of ordinary differential equations. In [51] Hochbruck and Lubich propose an ODE solver of Rosenbrock-Wanner type, which is based on Krylov subspace approximations to $y = \varphi(\gamma h J)c$. Here J is the Jacobian of the ODE $x' = f(x)$, h is the step size, and γ is a parameter. In this section we will show that the method of Section 5.7 is also applicable here, but with a different right-hand side for (5.32).

Instead of $\gamma h J$, we will use $-B$. So, locally non-increasing solution components of $x' = f(x)$ are associated with positive eigenvalues of B. We are interested in computing $y = \varphi(-B)c$ or

$$y = (e^{-B} - I)(-B)^{-1}c\,. \tag{5.36}$$

The matrix function $\varphi(-B) = (e^{-B} - I)(-B)^{-1}$ is only defined for nonsingular B. The definition of $\varphi(-B)$ can be extended to singular B by the Taylor series $\varphi(-B) = \sum_{i\geq 0}(-B)^i/(i+1)!$. Premultiplying both hand sides of (5.36) by e^B leads to

$$e^B y = (e^B - I)B^{-1}c\,. \tag{5.37}$$

We already know how to handle the left-hand side of (5.37). The right-hand side is treated analogously:

$$\begin{bmatrix} I & 0 & \ldots & \ldots & 0 & I/(p-1) \\ -B/(p-2) & \ddots & \ddots & & \vdots & \vdots \\ 0 & \ddots & I & \ddots & \vdots & \vdots \\ \vdots & \ddots & -B/2 & I & 0 & I/2 \\ \vdots & & \ddots & -B & I & I \\ 0 & \ldots & \ldots & 0 & -B & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} 0 \\ c/((p-1)(p-2)) \\ c/((p-2)(p-3)) \\ \vdots \\ c/(2\cdot 1) \\ c \end{bmatrix}$$

Block Gaussian elimination of $x_1, \ldots, x_{p-1}$ leads to

$$\widetilde{e^B} x_p = (\widetilde{e^B} - I)B^{-1}c\,. \tag{5.38}$$

Hence, e^B in (5.37) is replaced by the $p-1$-st order Taylor approximation $\widetilde{e^B}$ and we conclude that x_p is an approximation of $y = \varphi(-B)c$.

Note that the same approximation for $y = \varphi(-B)c$ is given by $-x_{p-1}$ in equation (5.32), because block Gaussian elimination of $x_1, \ldots, x_{p-2}$, and x_p in (5.32), leads to (5.38).

5.8 Numerical experiments

In this section we present several numerical experiments to illustrate the possibilities of the approach proposed in the previous sections. For the numerical experiments we have only used diagonal matrices B. For the methods used in this section this is not restriction, because each symmetric problem can be transformed to a diagonal form which has the same convergence behaviour as the original problem. This is a well known property for Krylov subspace methods, and follows from the fact that symmetric matrices are diagonalizable with an orthogonal matrix. Note that the behaviour in inexact arithmetic may be slightly different for diagonal matrices [79]. An advantage of diagonal matrices is that the eigenpairs are explicitly known, which may be helpful in understanding the convergence behaviour. Moreover, the experiments are easy to reproduce by other researchers.

In the numerical experiments, we compare three different methods: SLDM, polMINRES and $P(\text{MINRES})_{\text{L}}$. For the problems considered here, $P(\text{GMRES})$ and $P(\text{MINRES})_{\text{S}}$ lead to practically the same results as $P(\text{MINRES})_{\text{L}}$. The polMINRES method uses short recurrences for the update of the approximate solution at every iteration. We used a simple variant of polMINRES which computes the approximate solution as a linear combination of Lanczos vectors at every iteration. The convergence results for several problems and several methods are plotted in Figures 5.2 to 5.7. These convergence plots have the number of matrix-vector products k on the horizontal axis and the norm of the error or residual on the vertical axis, see Table 5.2.

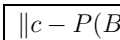
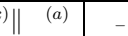
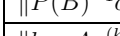
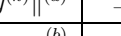
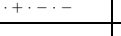
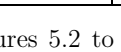
	SLDM	polMINRES	$P(\text{MINRES})_{\text{L}}$
$\|c - P(B)y^{(k)}\|$ $^{(a)}$			
$\|P(B)^{-1}c - y^{(k)}\|$ $^{(a)}$			
$\|b - Ax^{(k)}\|$ $^{(b)}$			

Table 5.2: Legend of the Figures 5.2 to 5.4 and Figure 5.7, see Table 5.3 for Figure 5.6. After k matrix-vector products the approximate solution of $P(B)y = c$ and $Ax = b$ is $y^{(k)}$ and $x^{(k)}$, respectively. $^{(a)}$ $\|c - P(B)y^{(k)}\|$ is the true residual norm and $\|P(B)^{-1}c - y^{(k)}\|$ is the true error norm. $^{(b)}$ The residual norm $\|b - Ax^{(k)}\|$ is computed as the norm of a recursively updated vector of dimension p, see formula (4.42).

Experiment 1 In this experiment we consider $B^2y = c$, where B is a scaled version of the matrix in example 4 of van der Vorst's paper on $f(A)x = b$ [88]. The scaled matrix is $B = 10D = 10\,\text{diag}(\lambda_1, \ldots, \lambda_{900})$, with $\lambda_1 = 0.034 < \lambda_2 < \ldots < \lambda_{900} = 1.2$. The right-hand side is $c = B^2[1, 1, \ldots, 1]^T$. The spectrum of D mimics roughly the spectrum of some incomplete Cholesky preconditioned matrix. We scaled the matrix in order to have a convergence phase where polMINRES is clearly better than $P(\text{MINRES})_{\text{L}}$. $P(\text{MINRES})_{\text{L}}$ uses the following parameters for the block linear system (5.6): $\mu_1 = 0$,

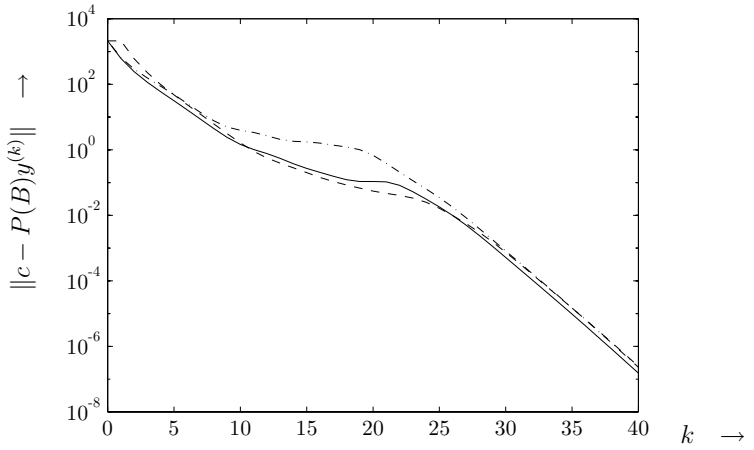


Figure 5.2: Experiment 1. Convergence plot; see Table 5.2 for legend.

$\mu_2 = 0$ and $\rho = 0$. The convergence for the three methods is plotted in Figure 5.2. After 30 matrix-vector products $P(\text{MINRES})_{\text{L}}$ is slightly better than polMINRES.

Experiment 2 Again we consider $B^2y = c$, but now with $B = \text{diag}(1 + 1/1,\ 1 + 1/2,\ 1 + 1/3,\ \ldots,\ 1 + 1/99, 0.01)$ and $c = [1, 1, \ldots, 1, 0.0001]^T$. A near stagnation phase in the convergence of $P(\text{MINRES})_{\text{L}}$ and polMINRES is created by chosing the entries $B_{100,100}$ and c_{100} smaller than the others, see Figure 5.3.

For linear systems $By = c$, Brown [16] showed that FOM breaks down if GMRES stagnates: $y^{(j)} = y^{(j+1)}$. Furthermore, FOM has a relatively large residual if GMRES

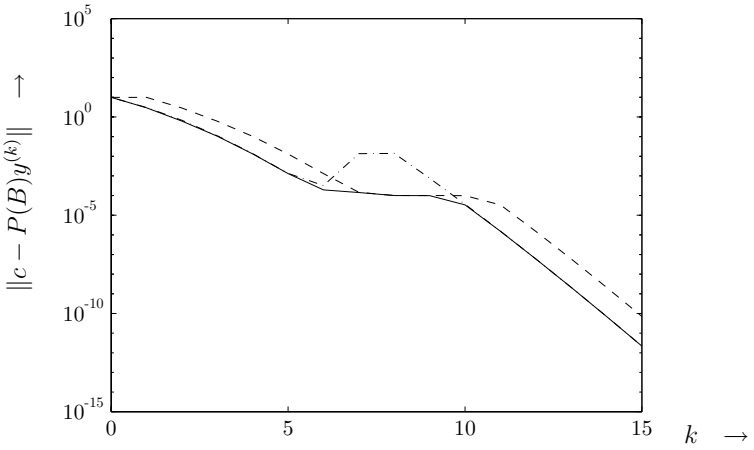


Figure 5.3: Experiment 2. Convergence plot; see Table 5.2 for legend.

nearly stagnates, see Cullum and Greenbaum [18]. We see a similar behaviour here: SLDM has a large residual if the minimal residual approach $P(\text{MINRES})_{\text{L}}$ nearly stagnates, see also Section 4.4.4 for an explanation of this phenomenon.

In terms of matrix-vector products $P(\text{MINRES})_{\text{L}}$ is the winner in this experiment. For a polynomial $P(B)$ of degree p, the maximal gain of $P(\text{MINRES})_{\text{L}}$ compared with polMINRES is $p-1$ matrix-vector products. This maximal gain of 1 matrix-vector product is asymptotically almost attained in this example, see Figure 5.3.

Experiment 3 In this experiment, we consider a problem for which Block Gaussian elimination of $x_1, \ldots, x_{p-1}$ in (5.6) is not a stable process. The problem is $B^4 y = c$, with $B = \text{diag}(0.5, 0.51, 0.52, \ldots, 1.99, 2, 500)$ and c a random vector, with entries between -1 and 1, scaled to unit norm. The parameters of (5.6) are $\mu_1 = \mu_2 = \mu_3 = \mu_4 = \rho = 0$.

The polMINRES method explicitly forms the matrix $H_*^{(m+3)} H_*^{(m+2)} H_*^{(m+1)} H_*^{(m)}$ for the solution of $B^4 y = c$. This matrix is ill-conditioned because B^4 is ill-conditioned: its condition number is $\kappa(B^4) = 10^{12}$. This leads to less accurate results, as we will see. $P(\text{MINRES})_{\text{L}}$ deals with the much better conditioned matrix $H^{(m)}$ of (5.9). The residual norms $\|c - P(B)y^{(k)}\|$ of polMINRES and $P(\text{MINRES})_{\text{L}}$ stagnate at approximately 10^{-6}, see Figure 5.5. However, the error norm of polMINRES stagnates, while $P(\text{MINRES})_{\text{L}}$ is still converging. The residual norm $\|b - Ax^{(k)}\|$ of (5.7) also converges to a small value.

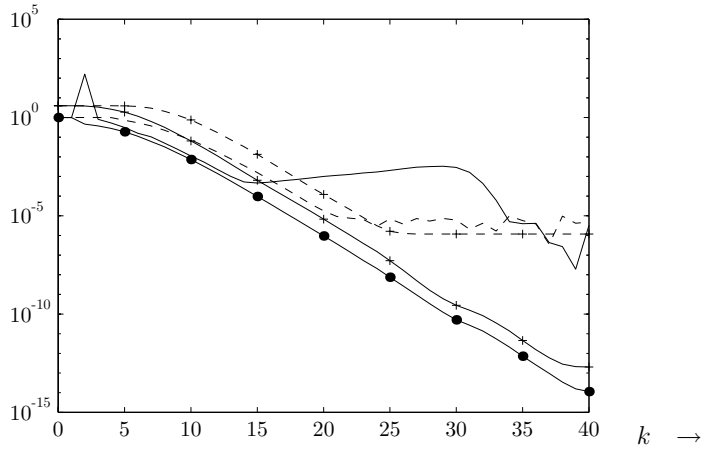


Figure 5.4: Experiment 3. Convergence of polMINRES and $P(\text{MINRES})_{\text{L}}$. Vertical axis: the norm of a residual or error, see Table 5.2

Experiment 4 This experiment shows that the convergence of $P(\text{MINRES})_{\text{L}}$ depends mildly on the scaling of the problem. We consider the problem $B^2 y = c$. The matrix B and vector c are defined in example 5 in [88]: $B = \text{diag}(\lambda_1, \ldots, \lambda_{900})$, with $\lambda_1 = 214.827 > \lambda_2 > \ldots > \lambda_{900} = 1.0$ and $c = B^2[1, 1, \ldots, 1]^T$.

For the linear system (5.12) we used two different scaling parameters α: $\alpha = 1/10$

and $\alpha = 1/100$. For $\alpha = 1/100$ the convergence is slightly smoother than for $\alpha = 1/10$, see Figure 5.5. But for $\alpha = 1/10$ the residual is often smaller. Note that the norm of the residual $\|b - Ax^{(k)}\|$ of the block linear system (5.7) is non-increasing. This does not imply that $\|c - P(B)y^{(k)}\|$ is non-increasing too, as we see in this experiment.

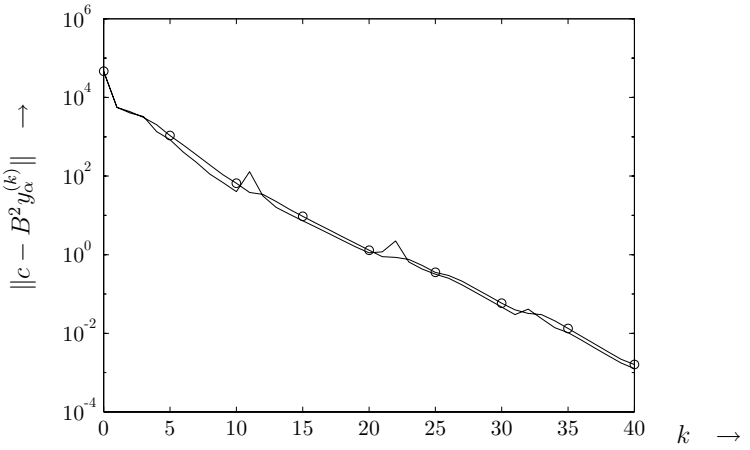


Figure 5.5: Experiment 4. Convergence of $P(\text{MINRES})_{\text{L}}$ for different scaling parameters. The solid line ——— is the residual for $\alpha = 1/10$. The circled line ——o—— is for $\alpha = 1/100$.

Experiment 5 We consider the linear system $e^B y = c$, with a diagonal matrix $B = 1/100\,\text{diag}(0,\ 4,\ 8,\ ,\ldots,\ 4000)$ and c a random vector, with entries between -1 and 1, scaled to unit norm. This is an example taken from [51]. The matrix e^B is very ill-conditioned: $\kappa(e^B) = 2.3 \cdot 10^{17}$. However, we will see that $y = e^{-B}c$ can be computed accurately by SLDM and the $e^{\text{MINRES}}{}_{\text{L}}$ method proposed in Section 5.7. This is not caused by the fact that B is a diagonal matrix, because SLDM and $e^{\text{MINRES}}{}_{\text{L}}$ work with a reduced matrix $H_*^{(m)}$ that is not a diagonal matrix. The method $e^{\text{MINRES}}{}_{\text{L}}$ approximates e^B implicitly with a polynomial $\widetilde{e^B}$ of degree $p-1$. In this experiment we used $p = 40$. This polynomial $\widetilde{e^B}$ is not suitable for polMINRES, because it is ill-conditioned. SLDM computes an approximate solution with $y^{(m)} = \|c\|V_*^{(m)}(e^{-H_*^{(m,\square)}})e_1$. In inexact arithmetic this is much more accurate than computing $y^{(m)} = \|c\|V_*^{(m)}(e^{H_*^{(m,\square)}})^{-1}e_1$, because $e^{H_*^{(m,\square)}}$ is ill-conditioned.

The convergence of SLDM and $e^{\text{MINRES}}{}_{\text{L}}$ is plotted in Figure 5.6. We see that the residual norm $\|c - e^B y^{(k)}\|$ is very large for this problem, but the true error $\|e^{-B}c - y^{(k)}\|$ converges rather well. Note that the error due to the Taylor approximation of e^B by $\widetilde{e^B}$ is included in the true error. The results of SLDM are slightly better than the results for $e^{\text{MINRES}}{}_{\text{L}}$. The upper bound (5.35) for $\|e^{-B}c - y^{(k)}\|$ is rather sharp for this problem.

	SLDM	$e^{\text{MINRES}_{\text{L}}}$
$\|c - e^B y^{(k)}\|$	– · – · – · – · –	————
$\|e^{-B}c - y^{(k)}\|$	– · – · + · – · –	——+——
(5.35) [(a)]		——×——

Table 5.3: Legend of Figure 5.6. $y^{(k)}$ is the approximate solution of $e^B y = c$ after k matrix-vector products. [(a)] This is the upper bound (5.35) for $\|y^{(m)} - e^{-B}c\|$.

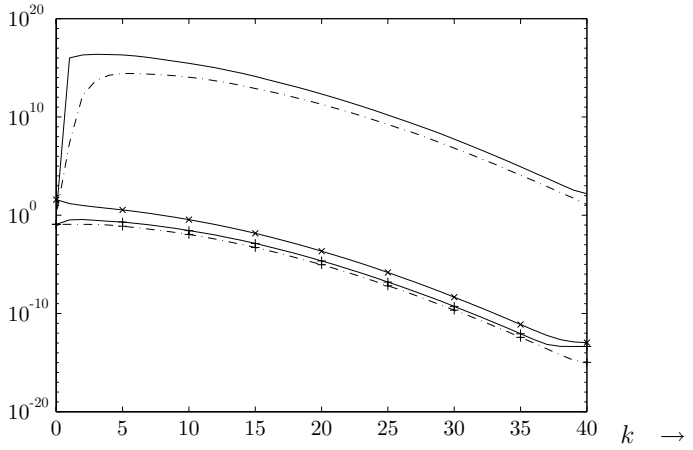


Figure 5.6: Experiment 5. Convergence of SLDM and $e^{\text{MINRES}_{\text{L}}}$. Vertical axis: the norm of a residual or error, see Table 5.3.

Experiment 6 This artificial experiment confirms that $P(\text{MINRES})_{\text{L}}$ is more suitable for solving $P(B)y = c$ than SLDM, if the factorized polynomial (5.5), with $\rho = 0$, has indefinite factors. The symmetric indefinite linear system is $B(B + 9/17I)y = c$, with

$$B = \text{diag}(n/(n+1),\ \ldots,\ 3/4,\ 2/3,\ 1/2,\ -1/2,\ -2/3,\ -3/4,\ \ldots,\ -n/(n+1))\,,$$

and $n = 50$. The right-hand side c is $c = [1, 1, \ldots, 1]^T$. The parameters of $P(\text{MINRES})_{\text{L}}$ for the block linear system (5.6) are: $\mu_1 = 0$, $\mu_2 = 9/17$ and $\rho = 0$. The matrix polynomial $P(B) = B(B + 9/17I)$ has one negative eigenvalue and $2n - 1$ positive eigenvalues. The conditions of Theorem 5.1 are not satisfied in this experiment. In exact arithmetic the solution of SLDM is not defined for each odd iteration. In inexact arithmetic this causes extremely large residual norms for the odd iterations, see Figure 5.7.

For any problem $P(B)y = c$, we may see locally a similar convergence behaviour for SLDM as in this experiment, if the factorized polynomial (5.5), with $\rho = 0$, has

indefinite factors. The SLDM solution is not defined if $H_*^{(m,\square)} - \mu_j$, $j = 1, \ldots, p$, is singular. This may happen frequently, depending on the right-hand side c, if (5.5) has indefinite factors.

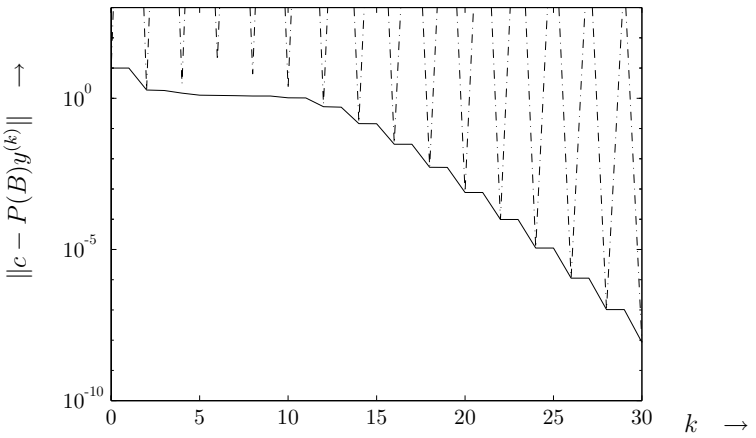


Figure 5.7: Experiment 6. Convergence of SLDM and $P(\text{MINRES})_{\text{L}}$ for an indefinite problem; see Table 5.2 for legend.

Appendix A

Matlab implementations of methods for $P(B)y = c$

In this appendix, we present MATLAB implementations of two methods proposed in Chapter 5: $P(\text{MINRES})_{\text{S}}$ and $P(\text{GMRES})$. Other variants, like $P(\text{MINRES})_{\text{L}}$ and $e^{\text{MINRES}}{}_{\text{L}}$, are easy to adapt from these two implementations.

The structure of the MATLAB implementation of $P(\text{MINRES})_{\text{S}}$ is displayed in Algorithm A.1.

Algorithm A.1: Structure of $P(\text{MINRES})_{\text{S}}$

Initialization
While not converged
  Compute a Lanczos vector
  Put the Lanczos coefficients in $\hat{H}^{(j)}$
  Update the QR factorization of $\hat{H}^{(j)}$
  Update the approximate solution $y^{(j)}$
End

The MATLAB code of $P(\text{MINRES})_{\text{S}}$ on page 120 is able to handle (complex) Hermitian matrices: $B^H = B \in \mathbb{C}^{n\times n}$. The vector c and the polynomial coefficients μ and ρ are also allowed to be complex. The matrix polynomial $P(B)$ does not need to be Hermitian.

The structure of the MATLAB implementation of $P(\text{GMRES})$ is displayed in Algorithm A.2. The inputs B, c, μ and ρ of the MATLAB $P(\text{GMRES})$ function on page 121 are allowed to be complex.

Algorithm A.2: Structure of P(GMRES)

Initialization
While not converged
  Compute an Arnoldi vector
  Put the Arnoldi coefficients in $\hat{H}^{(j)}$
  Update the QR factorization of $\hat{H}^{(j)}$
End
Compute the approximate solution $y^{(m)}$

Matlab code of P(MINRES)$_{\mathrm{S}}$

```
function [y,rnrm]=pminress(B,c,mu,rho,tol,m);
p=length(mu);
n=size(B,1);
W=zeros(n,3*p);
rnrm=zeros(m+1,1);
d=zeros(2*p,1);
d(p)=norm(c);
v=c/d(p);
rnrm(1)=abs(d(p));
j=1;
beta_old=0;
v_old=zeros(n,1);
y=zeros(n,1);
Q_old_old=eye(2*p);  Q_old=eye(2*p);
while (j<=m) & (rnrm(j)>tol),
  R=zeros(4*p,p);
  v_new=B*v-beta_old*v_old;
  alpha=v_new'*v;
  v_new=v_new-alpha*v;
  beta=norm(v_new);
  v_new=v_new/beta;
  H=[beta_old;alpha;beta];
  for i=1:p,
    if i==1,
      R(i+p:p:i+3*p,p)=H;
      R(i+2*p,p)=R(i+2*p,p)+mu(1);
    else
      R(i+p:p:i+3*p,i-1)=-H;
      R(i+2*p,i-1)=R(i+2*p,i-1)-mu(i);
    end;
    if i==p,
      R(i+2*p,i)=R(i+2*p,i)+rho;
```

```
    else
      R(i+2*p,i)=R(i+2*p,i)+1;
    end
  end;
  R(1:2*p,:)=Q_old_old'*R(1:2*p,:);
  R(1+p:3*p,:)=Q_old'*R(1+p:3*p,:);
  [Q,R(1+2*p:4*p,:)]=qr(R(1+2*p:4*p,:));
  d=Q'*d;
  Q_old_old=Q_old;  Q_old=Q;
  for i=1:p-1,
    W(:,2*p+i)=-(W(:,i:2*p-1+i)*R(i:2*p-1+i,i))/R(2*p+i,i);
  end;
  W(:,3*p)=(v-(W(:,p:3*p-1)*R(p:3*p-1,p)))/R(3*p,p);
  y=y+W(:,2*p+1:3*p)*d(1:p);
  rnrm(j+1)=norm(d(p+1:2*p));
  v_old=v;
  v=v_new;
  beta_old=beta;
  W(:,1:2*p)=W(:,p+1:3*p);
  d=[d(1+p:2*p) ; zeros(p,1)];
  j=j+1;
end;
```

Matlab code of P(GMRES)

```
function [y,rnrm]=pgmres(B,c,mu,rho,tol,m);
p=length(mu);
n=size(B,1);
Q=cell(m,1);
rnrm=zeros(m+1,1);
d=zeros((m+1)*p,1);
d(p)=norm(c);
v=c/d(p);
V=v;
rnrm(1)=abs(d(p));
j=1;
R=[ ];
while (j<=m) & (rnrm(j)>tol),
  Rc=zeros((m+1)*p,p);
  v=B*v;
  [v,H]=mgs(V,v);
  V=[V v];
  for i=1:p,
    if i==1,
      Rc(i:p:i+j*p,p)=H;
      Rc(i+(j-1)*p,p)=Rc(i+(j-1)*p,p)+mu(1);
```

```
    else
      Rc(i:p:i+j*p,i-1)=-H;
      Rc(i+(j-1)*p,i-1)=Rc(i+(j-1)*p,i-1)-mu(i);
    end;
    if i==p,
      Rc(i+(j-1)*p,i)=Rc(i+(j-1)*p,i)+rho;
    else
      Rc(i+(j-1)*p,i)=Rc(i+(j-1)*p,i)+1;
    end
  end;
  for k=1:j-1,
    Rc((k-1)*p+1:(k+1)*p,:)=Q{k}'*Rc((k-1)*p+1:(k+1)*p,:);
  end;
  [Q{j},Rc((j-1)*p+1:(j+1)*p,:)]=qr(Rc((j-1)*p+1:(j+1)*p,:));
  R=[R Rc(1:m*p,:)];
  d((j-1)*p+1:(j+1)*p)=Q{j}'*d((j-1)*p+1:(j+1)*p);
  rnrm(j+1)=norm(d(j*p+1:j*p+p));
  j=j+1;
end;
j=j-1;
z=R(1:j*p,1:j*p)\d(1:j*p);
y=V(:,1:j)*z(p:p:j*p);

function [q,y]=mgs(Q,q)
k=size(Q,2);
y=[];
for i=1:k,
  a=Q(:,i)'*q;
  q=q-Q(:,i)*a;
  y=[y;a];
end
a=norm(q);
q=q/a;
y=[y;a];
```

Bibliography

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, *The design and analysis of computer algorithms.* Addison-Wesley, Reading, MA, 1975.

[2] P.R. Amestoy, I.S. Duff and J.Y. L'Excellent, *Multifrontal parallel distributed symmetric and unsymmetric solvers.* Comput. Methods in Appl. Mech. Eng. 184, (2000), pp. 501-520.

[3] P.R. Amestoy, I.S. Duff, J.Y. L'Excellent, and J. Koster, *A fully asynchronous multifrontal solver using distributed dynamic scheduling.* Technical report RT/APO/99/2, ENSEEIHT-IRIT, Toulouse, France, 1999. Available at: `http://www.enseeiht.fr/apo/MUMPS/doc.html`

[4] P. Amodio, J.R. Cash, G. Roussos, R.W. Wright, G. Fairweather, I. Gladwell, G.L. Kraut, M. Paprzycki, *Almost block diagonal linear systems: sequential and parallel solution techniques, and applications.* Numer. Linear Algebra Appl. 7 (2000), pp. 275-317.

[5] M. Arioli, I.S. Duff, P.P.M de Rijk, *On the augmented system approach to sparse least-squares problems.* Numer. Math. 55 (1989), pp. 667-684.

[6] U.M. Ascher, S.Y.P. Chan, *On parallel methods for boundary value ODEs.* Computing 46 (1991), pp. 1-17.

[7] U.M. Ascher, R.M.M. Mattheij, R.D. Russell, *Numerical solution of boundary value problems for ordinary differential equations.* SIAM, Philadelphia, PA, 1995.

[8] M. Benzi, J.C. Haws, M. Tůma, *Preconditioning highly indefinite and nonsymmetric matrices.* SIAM J. Sci. Comput. 22 (2000), pp. 1333-1353

[9] L. Bergamaschi, M. Vianello, *Efficient computation of the exponential operator for large, sparse, symmetric matrices.* Numer. Linear Algebra Appl. 7 (2000), pp. 27-45.

[10] R.H. Bisseling, W.F. McColl, *Scientific computing on bulk synchronous parallel architectures.* Preprint 836, Dept. of Mathematics, Utrecht University, 1993.

[11] Å. Björck, *Iterative refinement of linear least squares solutions. I.* BIT 7 (1967), pp. 257-278.

[12] A. Bojanczyk, G. Golub, P. Van Dooren, *The periodic Schur decomposition. Algorithms and applications.* Proc. SPIE Conf., Vol. 1770, pp. 31-42, 1992.

[13] C.W. Bomhof, H.A. van der Vorst, *A parallel linear system solver for circuit simulation problems.* Numer. Linear Algebra Appl. 7 (2000), pp. 649-665.

[14] W. Bomhof, H.A. van der Vorst, *A parallelizable GMRES-type method for p-cyclic matrices, with applications in circuit simulation.* To appear in the proceedings of the SCEE-2000 Workshop, August 20-23, 2000, Warnemünde.

[15] K.E. Brenan, S.L. Campbell, L.R. Petzold, *Numerical solution of initial-value problems in differential-algebraic equations.* SIAM, Philadelphia, PA, 1996.

[16] P.N. Brown, *A theoretical comparison of the Arnoldi and GMRES algorithms.* SIAM J. Sci. Statist. Comput. 12 (1991), pp. 58-78.

[17] E.W. Cheney, *Introduction to approximation theory.* Reprint of the second (1982) edition. AMS Chelsea Publishing, Providence, RI, 1998.

[18] J. Cullum, A. Greenbaum, *Relations between Galerkin and norm-minimizing iterative methods for solving linear systems.* SIAM J. Matrix Anal. Appl. 17 (1996), pp. 223-247.

[19] T.A. Davis, I.S. Duff, *An unsymmetric-pattern multifrontal method for sparse LU factorization.* SIAM J. Matrix Anal. Appl. 18(1) (1997), pp. 140-158.

[20] K. Dekker, *Parallel GMRES and domain decomposition.* Report 00-05, Department of Applied Mathematical Analysis, Delft University of Technology, 2000. Available at: `ftp://ta.twi.tudelft.nl/pub/TWA_Reports/00-05.ps`

[21] J.W. Demmel, J.R. Gilbert. X.S. Li, *An asynchronous parallel supernodal algorithm for sparse Gaussian elimination.* SIAM J. Matrix Anal. Appl. 20(4) (1999), pp. 915-952.

[22] J.W. Demmel, S.C. Eisenstat, J.R. Gilbert, X.S. Li, J.W.H. Liu, *A supernodal approach to sparse partial pivoting.* SIAM J. Matrix Anal. Appl. 20(3) (1999), pp. 720-755.

[23] V. Druskin, A. Greenbaum, L. Knizhnerman, *Using nonorthogonal Lanczos vectors in the computation of matrix functions.* SIAM J. Sci. Comput. 19 (1998), pp. 38-54.

[24] V. Druskin, L. Knizhnerman, *Krylov subspace approximation of eigenpairs and matrix functions in exact and computer arithmetic.* Numer. Linear Algebra Appl. 2 (1995), pp. 205-217.

[25] I.S. Duff, A.M. Erisman, J.K. Reid, *Direct methods for sparse matrices.* Oxford University Press, Oxford, 1986.

[26] I.S. Duff, J.K. Reid, *The design of MA48: a code for the direct solution of sparse unsymmetric linear systems of equations.* ACM Trans. Math. Software 22 (1996), pp. 187-226.

[27] I.S. Duff, J. Koster, *The design and use of algorithms for permuting large entries to the diagonal of sparse matrices.* SIAM J. Matrix Anal. Appl. 20(4) (1999), pp. 889-901.

[28] M. Eiermann, W. Niethammer, A. Ruttan, *Optimal successive overrelaxation iterative methods for p-cyclic matrices.* Numer. Math. 57 (1990), pp. 593-606.

[29] S.C. Eisenstat, H.C. Elman, M.H. Schultz, *Variational iterative methods for nonsymmetric systems of linear equations.* SIAM J. Numer. Anal. 20 (1983), pp. 345-357.

[30] S.C. Eisenstat, J.W.H. Liu, *Exploiting structural symmetry in a sparse partial pivoting code.* SIAM J. Sci. Comput. 14(1) (1993), pp. 253-257.

[31] S.C. Eisenstat, J.W.H. Liu, *Exploiting structural symmetry in unsymmetric sparse symbolic factorization.* SIAM J. Matrix. Anal. Appl. 13(1) (1992), pp. 202-211.

[32] D.J. Evans, C.R. Wan, *Parallel direct solution for p-cyclic matrix systems.* Parallel Computing 19 (1993), pp. 79-93.

[33] P. Feldmann, R.W. Freund, *Numerical simulation of electronic circuits: state-of-the-art techniques and challenges.* Course notes, 1995 SIAM Annual Meeting, Available at: `http://cm.bell-labs.com/who/freund/index.html`

[34] P. Fiebach, *PolMINRES and PolCG for solving $P(A)x = b$.* preprint BUGHW-SC 99/1, University of Wuppertal, Department of Mathematics, 1999. Available at: `http://www.math.uni-wuppertal.de/org/SciComp/Preprints/SC9901.ps.gz`

[35] P. Fiebach, *Krylov-Verfahren zur Lösung von $P(A)x = b$.* PhD thesis, preprint BUGHW-SC 98/5, University of Wuppertal, Department of Mathematics, 1998. Available at: `http://www.math.uni-wuppertal.de/org/SciComp/Preprints/SC9805.ps.gz`

[36] R.W. Freund, N.M. Nachtigal, *QMR: a quasi-minimal residual method for non-Hermitian linear systems.* Numer. Math. 60(3) (1991), pp. 315-339.

[37] I. Fried, *Numerical solution of differential equations.* Academic Press, New York-London, 1979.

[38] E. Gallopoulos, Y. Saad, *Efficient solution of parabolic equations by Krylov approximation methods.* SIAM J. Sci. Statist. Comput. 13(5) (1992), pp. 1236-1264.

[39] G.A. Geist, E. Ng, *Task scheduling for parallel sparse Cholesky factorization.* Internat. J. Parallel Programming 18 (1989), pp. 291-314.

[40] J.R. Gilbert, T. Peierls, *Sparse partial pivoting in time proportional to arithmetic operations.* SIAM J. Sci. Comput. 9(5) (1988), pp. 862-874.

[41] J.R.Gilbert, C. Moler, R Schreiber, *Sparse matrices in Matlab: design and implementation.* SIAM J. Matrix Anal. Appl. 13(1) (1992), pp. 333-356.

[42] G.H. Golub, C.F. Van Loan, *Matrix computations.* Third edition, Johns Hopkins University Press, Baltimore, MD, 1996.

[43] G.H. Golub, R.S. Varga, *Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order Richardson iterative methods.* Numer. Math. 3 (1961), pp. 147-168.

[44] A. Hadjidimos, D. Noutsos, M. Tzoumas, *On the convergence domains of the p-cyclic SOR.* J. Comput. Appl. Math. 72 (1996), pp. 63-83.

[45] A. Hadjidimos, R.J. Plemmons, *Optimal p-cyclic SOR.* Numer. Math. 67 (1994), pp. 475-490.

[46] E. Hairer, S.P. Nørsett, G. Wanner, *Solving ordinary differential equations. I. Nonstiff problems.* Second edition. Springer-Verlag, Berlin, 1993.

[47] B. Hendrickson, R. Leland, *An improved spectral graph partitioning algorithm for mapping parallel computations.* SIAM J. Sci. Stat. Comput. 16(2) (1995), pp. 452-469.

[48] J.M.D. Hill, S.R. Donaldson, D.B. Skillicorn, *Portability of performance with the BSPlib communications library.* In Programming Models for Massively Parallel Computers, (MPPM'97), IEEE Computer Society Press, 1997.

[49] C.W. Ho, A.E. Ruehli, and P.A. Brennan, *The modified nodal approach to network analysis.* IEEE Trans. on Circuits and Systems 22 (1975), pp. 504-509.

[50] M. Hochbruck, *Lanczos- und Krylov-Verfahren für nicht-Hermitesche lineare Systeme.* PhD thesis, Fakultät für Mathematik, Universität Karlsruhe, 1992.

[51] M. Hochbruck, C. Lubich, *On Krylov subspace approximations to the matrix exponential operator.* SIAM J. Numer. Anal. 34(5) (1997), pp. 1911-1925.

[52] S.H.M.J. Houben *Algorithms for periodic steady state analysis on electric circuits.* Master's Thesis, Nat. Lab. Unclassified Report 804/99, Philips Electronics N.V., 1999.

[53] D. Hysom, A. Pothen *Efficient computation of ILU preconditioners.* Super Computing 1999 conference proceedings, Available at:
`http://www.sc99.org/proceedings/papers/hysom.pdf`

[54] B. Jiang, S. Richman, K. Shen, and T. Yang, *Efficient sparse LU factorization with lazy space allocation.* In SIAM 1999 Parallel Processing Conference on Scientific Computing.

[55] G. Karypis, V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs.* SIAM J. Sci. Comput. 20 (1999), pp. 359-392.

[56] G. Karypis, V. Kumar, *Parallel threshold-based ILU factorization.* Super Computing 1997 conference proceedings, Available at: `http://www.supercomp.org/sc97/proceedings/TECH/KARYPIS/INDEX.HTM`

[57] K. Kundert, *Sparse matrix techniques.* In: Circuit analysis, simulation and design, Albert Ruehli (Ed.), North-Holland, 1986.

[58] S. I. Larimore, *An approximate minimum degree column ordering algorithm.* MS Thesis, CISE Tech Report TR-98-016, University of Florida, 1998. Available at: `ftp://ftp.cise.ufl.edu/cis/tech-reports/tr98/tr98-016.ps`

[59] L. Lengowski, *CGS preconditioned with ILUT as a solver for circuit simulation.* Nat. Lab. Unclassified Report 828/98, Philips Electronics N.V., 1998.

[60] X.S. Li, J.W. Demmel, *Making sparse Gaussian elimination scalable by static pivoting.* Proceedings of Supercomputing 98 conference, November 7-13, 1998, in Orlando.

[61] J.W.H. Liu, *The role of elimination trees in sparse factorization.* SIAM J. Matrix Anal. Appl. 11(1) (1990), pp. 134-172.

[62] J.W.H. Liu, *Modification of the minimum degree algorithm by multiple elimination.* ACM Trans. Math. Software 11 (1985), pp. 141-153.

[63] Matrix Market, Collection of test matrices, available at: `http://math.nist.gov/MatrixMarket/index.html`.

[64] W.J. McCalla, *Fundamentals of computer-aided circuit simulation.* Kluwer Acad. Publ. Group, Dordrecht, the Netherlands, 1988.

[65] W.D. McQuain, C.J. Ribbens, L.T. Watson, R.C. Melville, *Preconditioned iterative methods for sparse linear algebra problems arising in circuit simulation.* Comput. Math. Appl. 27(8) (1994), pp. 25-45.

[66] C.D. Meyer, R.J. Plemmons (editors), *Linear algebra, Markov chains, and queueing models.* Springer-Verlag, New York, 1993.

[67] C.C. Paige, M.A. Saunders, *LSQR: an algorithm for sparse linear equations and sparse least squares.* ACM Trans. Math. Software 8 (1982), pp. 43-71.

[68] C.C. Paige, M.A. Saunders, *Solutions of sparse indefinite systems of linear equations.* SIAM J. Numer. Anal. 12 (1975), pp. 617-629.

[69] A. Pothen, C. Sun, *A mapping algorithm for parallel sparse Cholesky factorization.* SIAM J. Sci. Comput. 14 (1993), pp. 1253-1257.

[70] J.K. Reid, *The use of conjugate gradients for systems of linear equations possessing "Property A".* SIAM J. Numer. Anal. 9 (1972), pp. 325-332.

[71] G. Reißig, *On the performance of minimum degree and minimum local fill heuristics in circuit simulation.* Techn. Rep., Massachusetts Institute of Technology, Dept. Chem. Eng., 2001. Available at: `http://www.mit.edu/~gunther/`

[72] E. Rothberg, Silicon Graphics, Inc., man-page for PSLDU.

[73] Y. Saad, *Iterative methods for sparse linear systems.* PWS Publ. Comp., Boston, MA, 1996.

[74] Y. Saad, *Numerical methods for large eigenvalue problems.* Manchester University Press, New York, 1992.

[75] Y. Saad, *Analysis of some Krylov subspace approximations to the matrix exponential operator.* SIAM J. Numer. Anal. 29 (1992), pp. 209-228.

[76] Y. Saad, M. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems.* SIAM J. Sci. Statist. Comput. 7 (1986), pp. 856-869.

[77] M.A. Saunders, *Solution of sparse rectangular systems using LSQR and Craig.* BIT 35 (1995), pp. 588-604.

[78] V. Simoncini, E. Gallopoulos, *An iterative method for nonsymmetric systems with multiple right-hand sides.* SIAM J. Sci. Comput. 16 (1995), pp. 917-933.

[79] G.L.G. Sleijpen, H.A. van der Vorst, J. Modersitzki. *Differences in the effects of rounding errors in Krylov solvers for symmetric indefinite linear systems.* SIAM J. Matrix Anal. Appl. 22 (2000), pp. 726-751.

[80] D.C. Sorensen, *Implicitly restarted Arnoldi/Lanczos methods for large scale eigenvalue calculations.* In: Parallel numerical algorithms, pp. 119-165, Kluwer Acad. Publ., Dordrecht, 1997.

[81] J. Sreedhar, P. Van Dooren, *Pole placement via the periodic Schur decomposition.* Proc. Amer. Contr. Conf., San Francisco, June 2-4, pp. 1563-1567, 1993.

[82] G.J. Tee, *An application of p-cyclic matrices, for solving periodic parabolic problems.* Numer. Math. 6 (1964) 142-159.

[83] R. Telichevesky, K. Kundert, I. Elfadel, J. White, Fast simulation algorithms for RF circuits. In proc. IEEE 1996 Custom Integrated Circuits Conf., San Diego, pp. 437-444, 1996.

[84] R. Telichevesky, K.S. Kundert, and J.K. White, *Efficient steady-state analysis based on matrix-free krylov-subspace methods.* In proc. DAC'95, 1995.

[85] R. Telichevesky, K. Kundert, J. White, *Efficient AC and noise analysis of two-tone RF Circuits.* In proc. 33rd Design Automation Conference (DAC'96), Las Vegas, pp. 292-297, 1996.

[86] L.G. Valiant, *A bridging model for parallel computation.* Comm. Assoc. Comp. Mach. 33 (1990), pp. 103-111.

[87] R.S. Varga, *Matrix iterative analysis.* Prentice-Hall, Inc., Englewood Cliffs, N.J. 1962.

[88] H.A. van der Vorst, *An iterative solution method for solving $f(A)x = b$, using Krylov subspace information obtained for the symmetric positive definite matrix A.* J. Comput. Appl. Math. 18 (1987), pp. 249-263.

[89] H.A. van der Vorst, *Computational methods for large eigenvalue problems.* Submitted for publication with Elsevier - North Holland.

[90] C. Vuik, *Fast iterative solvers for the discretized incompressible Navier-Stokes equations.* Int. J. for Num. Meth. Fluids, 22 (1996), pp. 195-210.

[91] H.F. Walker, *Implementation of the GMRES method using Householder transformations.* SIAM J. Sci. Statist. Comput. 9 (1988), pp. 152-163.

[92] S.J. Wright, *Stable parallel algorithms for two-point boundary value problems.* SIAM J. Sci. Statist. Comput. 13 (1992), pp. 742-764.

[93] S.J. Wright, *Stable parallel elimination for boundary value ODEs.* Numer. Math. 67 (1994), pp. 521-535.

[94] D.M. Young, *Iterative solution of large linear systems.* Academic Press, New York-London, 1971.

Index

Samenvatting

Bij het ontwerp van elektronische schakelingen, die gebruikt worden in bijvoorbeeld CD-spelers en mobiele telefoons, maakt de ontwerper veelvuldig gebruik van circuitsimulatie. Bij circuitsimulatie wordt het gedrag van een schakeling (circuit) doorgerekend met een computer. Hierdoor wordt het maken van dure prototypes grotendeels overbodig. Ook zou zonder deze simulaties het ontwerpen van complexe geïntegreerde schakelingen, met vele duizenden transistoren, condensatoren, weerstanden en dergelijke, niet mogelijk zijn. Om snel een circuit te kunnen ontwerpen is het voor de ontwerper van belang dat de simulatie niet te veel (computer-)rekentijd kost. Met snellere (slimmere) rekenmethoden en ook met snellere computers, kan de rekentijd verkort worden.

Dit proefschrift gaat grotendeels over methoden die tot doel hebben de rekentijd voor het simuleren van een circuit korter te maken. De nieuwe methoden die we ontwikkeld hebben zouden echter ook nuttig kunnen zijn bij de simulatie van andere verschijnselen, zoals bijvoorbeeld vloeistofstromingen en chemische processen.

Bij het simuleren van circuits wordt de meeste rekentijd gebruikt voor het oplossen van grote stelsels lineaire algebraïsche vergelijkingen. Een stelsel van 2 vergelijkingen met 2 onbekenden, x en y, is bijvoorbeeld

$$\begin{cases} 3x + 5y &= 14 \\ 2x - 3y &= 3 \end{cases},$$

met als oplossing $x = 3$ en $y = 1$. Bij circuitsimulatie kunnen de stelsels zeer veel, bijvoorbeeld meer dan 50000, vergelijkingen hebben en evenveel onbekenden. Deze stelsels hebben dan wel een 'ijle' structuur. Dat wil zeggen dat er veel vergelijkingen zijn die slechts van een klein aantal onbekenden afhangen. Door op een slimme manier gebruik te maken van deze structuur kan er veel rekentijd bespaard worden. Na het inleidende eerste hoofdstuk beschrijven we in de hoofdstukken 2 en 3 een gecombineerde directe en iteratieve methode voor het oplossen van deze stelsels vergelijkingen.

Bij een directe methode worden onbekenden weggewerkt door een geschikt veelvoud van een vergelijking bij een andere vergelijking op te tellen. Op deze manier kan uiteindelijk de oplossing van het grote stelsel uitgerekend worden. Bij een iteratieve methode gebeurt ongeveer hetzelfde, maar de hoeveelheid rekenwerk wordt sterk beperkt door op geschikte plaatsen in het proces coëfficiënten te verwaarlozen. Het resultaat is dan wel een benadering van de oplossing in plaats van de exacte oplossing. Men tracht de fout in de oplossing te verkleinen door een correctie op de oplossing aan te brengen. Deze correctie wordt gevonden door een vergelijking voor de fout op te stellen en deze eveneens bij benadering op te lossen. Dit wordt herhaald totdat een voldoend nauwkeurige oplossing gevonden is.

In de praktijk maken circuitsimulatie-programma's vooral gebruik van directe methoden, omdat deze sneller bleken te zijn dan de tot nu toe bestaande iteratieve methoden. In hoofdstuk 2 laten we zien dat een gecombineerde directe en iteratieve methode wel drie keer sneller kan zijn dan een directe methode. Een prettige bijkomstigheid van deze aanpak is dat hij ook geschikt is voor 'parallelle' computers. Dat zijn computers waarin twee of meer processoren samenwerken. Met deze computers kan het rekenwerk verder versneld worden met een factor die kan oplopen tot het aantal processoren.

Hoofdstuk 4 gaat over het oplossen van lineaire stelsels vergelijkingen die optreden bij het simuleren van de periodieke stabiele toestand van een circuit. Het gaat hierbij om circuits waarvan alle spannings- en stroombronnen periodiek zijn in de tijd. Dit heeft tot gevolg dat alle spanningen en stromen in het circuit zich na een bepaalde periode herhalen. Simulatie van deze circuits geeft lineaire stelsels met een cyclische structuur. Bestaande methoden voor dit soort stelsels zijn niet zo goed geschikt voor parallelle computers. De methode die we in hoofdstuk 4 voorstellen is dat wel. De totale hoeveelheid rekenwerk is bij deze methode iets groter dan bij de bestaande methoden, maar dankzij het parallellisme kunnen de stelsels vergelijkingen op een parallelle computer toch beduidend sneller worden opgelost.

Hoofdstuk 5 gaat over een iteratieve methode voor lineaire stelsels vergelijkingen waarbij de coëfficiëntenmatrix een polynoom is van een andere matrix. Dit type lineaire stelsels komt onder andere voor bij een toepassing in de natuurkunde. Er zijn (nog) geen toepassingen in circuitsimulatie. Door de speciale structuur van het stelsel uit te buiten verkrijgen we een efficiënte methode. De nieuwe methode geeft vaak iets nauwkeuriger resultaten dan de bestaande methoden voor dit soort stelsels.

Dankwoord

Een aantal personen hebben bijgedragen aan de totstandkoming van dit proefschrift. In de eerste plaats is dat mijn promotor Henk van der Vorst, die ik wil bedanken voor de goede samenwerking de afgelopen vier jaar. Zijn enthousiasme, vragen en ideeën leidden ook tot enthousiasme, vragen en ideeën van mijn kant, zodat ik steeds weer een eind vooruit kon met mijn onderzoek. Ook wil ik Henk bedanken voor het precieze en heldere commentaar op stukken tekst, dat altijd gemakkelijk te verwerken was tot betere stukken tekst.

Jan ter Maten, Wil Schilders en Jaap Fijnvandraat (allen Philips Research), wil ik bedanken voor het aandragen van relevante soorten circuitsimulatie-problemen, het stellen van kritische vragen bij de zeswekelijkse HPCN-bijeenkomsten, en voor de practische hulp bij circuitsimulatie-zaken. Verder wil ik ook Menno Verbeek bedanken voor zijn inbreng tijdens en buiten de HPCN-besprekingen. De numerieke collega's in Utrecht bedank ik voor hun interesse in mijn resultaten en hun reacties daarop.

I would like to thank the members of the reading committee, prof. Iain Duff, prof. Bernd Fischer, prof. Marlis Hochbruck, and prof. Jan Verwer, for reading the manuscript. In particular, I am grateful to Iain Duff for his very detailed comments on the manuscript.

Roderik, Márcia en Ellen wil ik bedanken voor de gezellige werkomgeving.

Tenslotte bedank ik mijn vrouw Esther.

Curriculum vitae

Op 1 juli 1969 ben ik geboren te Vaassen. Van 1981 tot 1985 bezocht ik de Christelijke Technische School in Apeldoorn. Deze school werd afgesloten met een MAVO- en een LBO-diploma. Daarna volgde ik tot 1988 de MTS te Apeldoorn. Na het behalen van het propedeutisch examen van de studierichting elektrotechniek aan de HTS Zwolle, ging ik in 1989 Toegepaste Wiskunde studeren aan de Universiteit Twente. Tijdens deze studie heb ik stage gelopen bij het Koninklijke Shell Laboratorium in Amsterdam. Ook ben ik een aantal keer student-assistent geweest. Mijn afstudeerrichting was numerieke wiskunde bij prof. C.R. Traas. In 1996 werd het ingenieursdiploma bepaald, met lof. In hetzelfde jaar werd ik aangesteld als assistent in opleiding (AIO) bij de Vakgroep Wiskunde (nu Mathematisch Instituut) van de Universiteit Utrecht om promotieonderzoek te doen bij de onderzoeksgroep numerieke wiskunde van prof. H.A. van der Vorst. Hierbij werkte ik aan het onderdeel circuitsimulatie van het ELSIM-project van het platform HPCN. In dit project wordt samengewerkt tussen de Universiteit Utrecht en Philips Research Eindhoven. De resultaten van het onderzoek staan beschreven in dit proefschrift. Als AIO heb ik ook een aantal werkcolleges en practica gegeven.