STABLE LINEAR SOLVES WITH PRECONDITIONER UPDATES FOR MODEL REDUCTION

Ph.D. Thesis

By

NAVNEET PRATAP SINGH

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

NOVEMBER 2020

Stable Linear Solves with Preconditioner Updates for Model Reduction

A THESIS

submitted to the

INDIAN INSTITUTE OF TECHNOLOGY INDORE

in partial fulfillment of the requirements for

the award of the degree

of

DOCTOR OF PHILOSOPHY

by

NAVNEET PRATAP SINGH

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

NOVEMBER 2020

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled "**Stable Linear Solves with Preconditioner Updates for Model Reduction**" in the partial fulfillment of the requirements for the award of the degree of **Doctor of Philosophy** and submitted in the **Discipline of Computer Science and Engineering, Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from January 2014 to November 2020 under the supervision of Dr. Kapil Ahuja, Associate Professor, Indian Institute of Technology Indore, India.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

(30-Nov-2020)

Signature of the student with date
(NAVNEET PRATAP SINGH)

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

Signature of Thesis Supervisor with date
(Dr. KAPIL AHUJA)

NAVNEET PRATAP SINGH has successfully given his Ph.D. Oral Examination held on 24 March 2021.

(Dr. Sudeshna Chattopadhyay)	Madhu N. Belur, EE, IITB	
Signature of Chairperson (OEB)	Signature of External Examiner	Signature of Thesis Supervisor
Date: March 24, 2021	Date: 24th March 2021	Date:
Signature of PSPC Member #1	Signature of PSPC Member #2	Signature of Convener, DPGC
Date:	Date:	Date: 24/03/2021

Signature of Head of Discipline
Date:

ACKNOWLEDGEMENTS

I would like to express my deep sense of gratitude and indebtedness towards my supervisor **Dr. Kapil Ahuja**. Perseverance, exuberance, positive approaches are just some of the traits he has imprinted on my personality.

He steered me through this journey with his invaluable advice, positive criticism, stimulating discussions and consistent encouragement. If I will stand proud of my achievements, then undeniably he is the main creditor. With his enthusiasm, inspiration, and great efforts to explain things clearly and simply, he helped make mathematics fun for me. He gently corrected me at every stage: coming up with the relevant ideas, implementing them efficiently, presenting them with less ambiguity, and writing concisely. I would like to thank you for encouraging my research and allowing me to grow as a researcher. I could not have imagined having a better adviser and mentor for my Ph.D. study.

I am thankful to **Dr. Aruna Tiwari** and **Dr. Swadesh K. Sahoo**, my research progress committee members for taking out some valuable time to evaluate my progress all these years. Their valuable comments and suggestions helped me to improve my work at various stages.

My special thanks also goes to **Prof. Dr. Heike Fassbender**, who provided me an opportunity to join their team at Technische Universität Braunschweig, Germany, as an intern.

I would like to appreciate the fine company of my dearest colleagues and friends especially, Saumya Bhadauriya, Piyush Joshi, Sadaf Ali, Rajendra Choudhary, Rudesh Dwivedi, OM Prakesh Patel, Nikhil Tripathi, Mayank Swarnkar, Chandan Gautam, Ram Prakash Sharma, Aditya Shastri, Mayank Modak and Rakesh Sharma. I sincerely thanks **Rohit Agrawal** for the sleepless nights that we spent working together before deadlines.

I am also grateful to the institute staffs for their unfailing support and assistance and the Ministry of Human Resource Development for funding the PhD research.

I sincerely thanks **Mrs. Richa Ahuja** for her patience during the technical dis-

cussions I had with my supervisor. I would like to thanks **Aarav Ahuja**, for his patience and compromise as he didn't get enough time to play with his father (my supervisor).

I cannot express in words how grateful I am to my parents, who have been great source of inspiration over the years and never raised an eyebrow when I claimed my thesis would be finished 'in the next two months' for nearly one year. A special thanks to my siblings, uncles, thank you for your support and unwavering belief in me. I would like to thank my friends (Ritesh Raj, Vinay Tiwari, Ajay Singh, Aakhi M., Chainika Malhotra, Gokul Sharma) who have kept me on the track, added value to my thinking, taken up the slack I left in the wake of thesis anxiety and stress, buoyed me, and survived years of unfinished business with me.

Finally, I am thankful to all who directly or indirectly contributed, helped and supported me.

Navneet Pratap Singh

To my Parents

&

my supervisor

ABSTRACT

Simulation of large dynamical systems can be unmanageable due to high demands on computational resources. These large systems can be reduced into a smaller dimension by using Model Order Reduction (MOR) techniques. The reduced system has approximately the same characteristics as the original system but it requires significantly less computational effort in simulation. MOR can be done in many ways such as balanced truncation, Hankel approximations, and Krylov projection. Among these, the projection methods are quite popular, and hence, we focus on them.

We work with a wide array of MOR algorithms for reducing an extensive range of linear dynamical systems. That is, parametric/non-parametric as well as first-order and second-order.

In these MOR algorithms, sequences of very large and sparse linear systems arise during the model reduction process. Solving such linear systems is the main computational bottleneck in efficient scaling of these MOR algorithms for reducing extremely large dynamical systems. Preconditioned iterative methods are often used for solving such linear systems.

These iterative methods introduce errors because they solve the linear systems up to a certain tolerance. Hence, our *first* focus is to analyze the stability of MOR algorithms when using inexact linear solves. Further, in these MOR algorithms, the change from one linear system to the next is usually very small, and hence, the applied preconditioner could be reused, which is our *second* focus. Here, by using our techniques we demonstrate that a 1.2 million problem can be reduced in 3 days instead of earlier 8 days.

LIST OF PUBLICATIONS

Journal Papers

1. **Navneet Pratap Singh**, Kapil Ahuja, "Preconditioned Linear Solves for Parametric Model Order Reduction", *International Journal of Computer Mathematics*, Taylor & Francis, vol. 97, no. 7, pp. 1484–1502, 2020.

2. **Navneet Pratap Singh**, Kapil Ahuja, "Reusing Preconditioner in Moment Matching based Projective Model Order Reduction Algorithms", *IEEE Access*, vol. 8, pp. 133233–133247, 2020.

Technical Report

3. **Navneet Pratap Singh** and Kapil Ahuja. *Stability Analysis of Inexact Solves in Moment Matching based Model Reduction.* Preprint arXiv:1803.09283, 2018.

Conference Presentations

4. **Navneet Pratap Singh**, Kapil Ahuja, *Preconditioned Linear Solves for Parametric Model Order Reduction*, 11th Preconditioning Conference, University of Minnesota, July 02, 2019

5. **Navneet Pratap Singh**, Kapil Ahuja and Heike Fassbender, *Preconditioned Iterative Solves in Model Reduction of Second Order Linear Dynamical Systems*, Householder Symposium XX on Numerical Linear Algebra, USA, June 18-23, 2017

6. **Navneet Pratap Singh** and Kapil Ahuja, *Stability Analysis of Inexact Solves in Moment Matching based Model Reduction*, Reduced Basis Summer School, Hedersleben, Germany, October 4-7, 2016,

Contents

List of Figures

List of Tables

Table of Notations

Notation	**Definition**
$\mathbb{R}$	The set of real numbers
$\mathbb{C}$	The set of complex numbers
$\mathbb{N}$	The set of natural numbers
$\|\cdot\|_f$	The Frobenius norm
$\|\cdot\|$	The Euclidean norm for vectors and the induced spectral norm for matrices
$\|\cdot\|_{H_2}$	$H_2 - \mathbf{norm}$; detailed explanation on page 18
$\|\cdot\|_{H_\infty}$	$H_\infty - \mathbf{norm}$; detailed explanation on page 18
$\otimes$	The Kronecker product
$vec(\cdot)$	Vectorization of a matrix
I	The identity matrix
$H(s)$	Input dynamical system transfer function; detailed explanation on page 8
κ	Relative condition number for matrix operations; detailed explanation on page 28
$\kappa_B(H(s))$ & $\kappa_{\mathscr{C}}(H(s))$	Condition number for transfer function operations; detailed explanation on page 29
$\kappa(H(s))$	Condition number in our context; detailed explanation on page 29

Table of Abbreviations

Abbreviations	**Full form**	**Defining Location Page**
MOR	Model Order Reduction	2
SPD	Symmetric Positive Definite	6
AIRGA	Adaptive Iterative Rational Global Arnoldi	7
CG	Conjugate Gradient	20
FOM	Full Orthogonalization Method	20
SVD	Singular Value Decomposition	30
SISO	Single Input Single Output	32
SPAI	Sparse Approximate Inverse	33
ICHOL	Incomplete Cholesky Factorization	33
RPMOR	Robust Algorithm for Parametric Model Order Reduction	39
RCG	Recycling Conjugate Gradient	46
ILU	Incomplete LU Factorization	52
GMRES	Generalized Minimal Residual	61
MSPAI	Modified Sparse Approximate Inverse	61
GCRO	Generalized Conjugate Residual Orthogonal	77
SIMO	Single Input Multiple Output	78
spMV	Sparse Matrix-Vector Product	83
spMM	Sparse Matrix-Matrix Product	83
NMVs	Number of Matrix-Vector Products	84
PMOR-L	Data-Driven Parametrized Model Reduction algorithm in the Loewner Framework	93

Abbreviations	**Full form**	**Defining Location Page**
BIRKA	Bilinear Iterative Rational Krylov Algorithm	112
MIMO	Multiple Input Multiple Output	112
QB-IHOMM	Quadratic Bilinear-Implicit Higher Order Moment Matching	116

Chapter 1

Introduction

Dynamical systems arise in many areas of science and engineering. There are three factor that define a dynamical system; (i) linearity; (ii) parametrization; and (iii) order. Linear dynamical systems usually approximate the real-life phenomenas well, and hence, have been extensively studied. Thus, we focus on *linear* dynamical systems.

Whether a dynamical systems in parametrized or not; and the order of derivatives in the system are the other two characteristics defining a dynamical system. In this dissertation, we look at all such systems.

A parameterized second-order dynamical system is usually of the form [1][1]

$$\begin{aligned} M(p_1, p_2, \ldots, p_w)\ddot{x}(t) + D(p_1, p_2, \ldots, p_w)\dot{x}(t) + K(p_1, p_2, \ldots, p_w)x(t) = Bu(t), \\ y(t) = Cx(t), \end{aligned} \tag{1.1}$$

where $M(\cdot), D(\cdot), K(\cdot) \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{q\times n}$ and $(p_1, p_2, \ldots, p_w)$ are the parameters that are linearly embedded in the dynamical system matrices. Also, $x(t)\colon \mathbb{R} \to \mathbb{R}^n$ is the vector of all states, $u(t)\colon \mathbb{R} \to \mathbb{R}^m$ and $y(t)\colon \mathbb{R} \to \mathbb{R}^q$ are the inputs and the outputs of the system, respectively. If $M(\cdot) = 0$, then above equation can be written in the form of the parametric first-order dynamical system as

$$\begin{aligned} D(p_1, p_2, \ldots, p_w)\dot{x}(t) + K(p_1, p_2, \ldots, p_w)x(t) = Bu(t), \\ y(t) = Cx(t). \end{aligned} \tag{1.2}$$

[1] As earlier, focus is only on *linear* dynamical systems.

Table 1.1: Linear MOR Algorithms based upon Projection.

S. No.	**Category**	**First-order**	**Second-order**
1	Non-Parametric	Cell 1	Cell 2
		IRKA [4], $(Sy)^2$IRKA [5], MIRIAm [6]	SOR-IRKA [7], SO-IRKA [8], SOSPDR [9], AIRGA [10]
2	Parametric	Cell 3	Cell 4
		IPMOR [11], RPMOR [12], PBTMR [13]	S-RPMOR [1], IDPA [14], S-PBTMR [13]

If in (1.1) and (1.2) the system matrices are independent of the parameters, then they represent a non-parametric second-order and first-order dynamical system, respectively.

Simulation of large dynamical systems can be unmanageable due to high demands on computational resources. These large systems can be reduced into a smaller dimension by using Model Order Reduction (MOR) techniques [2, 3, 4, 5]. The reduced system has approximately the same characteristics as the original system but it requires significantly less computational effort in simulation. MOR can be done in many ways such as balanced truncation, Hankel approximations, and Krylov projection. Among these, the projection methods are quite popular, and hence, we focus on them.Table 1.1 summarizes most of the commonly used such algorithms.

In the mentioned MOR algorithms in Table 1.1, sequences of very large and sparse linear systems arise during the model reduction process. Solving such linear systems is the main computational bottleneck in efficient scaling of these MOR algorithms for reducing extremely large dynamical systems, which we discuss next.

1.1 Iterative Methods and Preconditioners

Direct methods, which are based upon different matrix factorizations, are commonly used for solving linear systems of equations [15]. Standard direct methods scale badly in-terms of the number of operations and the memory used (with respect to the increase in the size of the linear systems; as here). They typically perform dense linear algebra operations, and hence, are not an efficient choice when the linear system matrices are sparse (as here as well).

An alternative to this is to use sparse direct methods. These methods solve this scaling problem to a great extent such that linear systems of fairly large size could be efficiently solved[2]. However, sparse direct methods also become prohibitively expensive for extremely large sizes (hundreds of millions of equations to billions of equations).

In such cases, using iterative methods are usually the only viable option, which scale well both in time and memory. Although iterative methods are not as robust or reliable as direct methods, they are still preferred when scaling is a bigger issue. This is the case with MOR algorithms, and hence, we use them here.

Krylov subspace based methods are very popular class of iterative methods [18], which we focus on. If $Ax = b$ is the linear system to be solved, with $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, x_0 the initial solution and r_0 (where $r_0 = b - Ax_0$) the initial residual, then these methods find the solution in $\mathbb{K}_k(A,\ r_0) = span\{r_0,\ Ar_0,\ A^2r_0,\ \ldots,\ A^{k-1}r_0\}$, where $\mathbb{K}_k(\cdot,\ \cdot)$ represents the Krylov subspace.

Often iterative methods are slow or fail to converge, and hence, preconditioning is used to accelerate them. We expect that the preconditioned iterative solves would find a solution in less amount of time as compared to the unpreconditioned ones. For most of the input dynamical systems, the Krylov subspace methods fail to converge. Hence, we use a preconditioner. The goal is to find a preconditioner that is cheap to compute as well as apply.

If P is a non-singular matrix that approximates the inverse of A, then the precon-

[2]Often, they work well for linear systems arising from certain problem classes [15, 16, 17], for example, discretization of PDEs in two dimensions.

ditioned system becomes $AP\tilde{x} = b$ with $x = P\tilde{x}$. This is termed as right preconditioning. Similarly, left preconditioning can also be performed, where the preconditioner is present on the left side of the matrix [15][3]. If the linear system coefficient matrices are SPD, then both the types of preconditioning give the same results [15].

For our MOR algorithms under-consideration, the linear system coefficient matrices do not have any special structure. Hence, both these types of preconditioning work differently. In our experiments, we mostly use right preconditioning because it is fairly common [19, 20]. However, to demonstrate that our techniques are independent of the type of preconditioning, for some models, we experiment with left preconditioning in the side as well.

Preconditioned iterative methods are not exact because they solve linear systems upto a certain tolerance. This raises the question that if preconditioned iterative methods are used inside the MOR algorithms, then are these algorithms stable with respect to the error introduced by these methods. Hence, our *first* focus is to investigate the stability of MOR algorithms (with respect to use of iterative methods). This is briefly elaborated upon in Section 1.2.

Further, in these MOR algorithms, the change from one linear system to the next is usually very small, and hence, the applied preconditioner could be reused, which is our *second* focus. This aspect is expanded in Section 1.3.

1.2 Stability Analysis of MOR Algorithms

As mentioned earlier, we investigate the stability of MOR algorithms with respect to use of preconditioned iterative methods. This kind of analysis was first proposed in [21], where a popular MOR algorithm for *linear* non-parametric first-order dynamical systems was analyzed (corresponding to Cell 1 of Table 1.1). An extended stability analyses for commonly used MOR algorithms for *bilinear* dynamical systems (non-parametric and parametric; first-order) have recently been done in [22] and [23].

[3]If the preconditioner is present on both the sides of the coefficient matrix, then it is called split/ center preconditioning.

In this dissertation, we focus on stability analyses of MOR algorithms belonging to other Cells of Table 1.1 (i.e. 2, 3, and 4). That is, as earlier with focus on *linear* systems, we perform stability analyses of MOR algorithms for non-parametric second-order, parametric first-order, and parametric second-order dynamical systems.

Besides *comprehensively* analyzing a wide range of MOR algorithms, this dissertation has another very unique contribution that has not yet been looked at by any other past work. For example, in [21] the authors mention that their stability analysis for non-parametric first-order dynamical systems can be easily carried to non-parametric second-order systems. Besides the fact that the authors do not perform this analysis in-details, they also do not focus on how to satisfy the arising stability conditions.

In all our analyses, we show that satisfying the stability conditions requires *changing the underlying linear solvers*, and that too in an efficient way so as to not incur any extra cost.

In the current context, it is important to highlight the difference between our track of stability analyses and the one done in [24] as well. The authors in [24] first showed that the SOAR algorithm is unstable with respect to the machine precision errors (and not inexact solves of iterative methods, which is our focus). Then, they proposed a Two-level orthogonal Arnoldi (TOAR) algorithm that cures this instability of SOAR (we propose recycling variants of the underlying iterative methods for achieving stability).

1.3 Reuse of Preconditioners in MOR Algorithms

As earlier, in most of the MOR algorithms, the change from one linear system to the next is small. Using this fact, we propose a preconditioner reuse technique.

[25] and [26] first applied this technique in the quantum Monte Carlo context, where it is referred to as recycling preconditioners. For the case when the linear system coefficient matrices are perturbed by a varying constant times the identity matrix, efficient preconditioners have also been developed. These preconditioners are independent of the underlying application and are referred to as preconditioner updates (see

[27] for Symmetric Positive Definite (SPD) coefficient matrices and [28] for general coefficient matrices).

This approach has been used in the optimization context in [29], where it is again termed as preconditioner updates. In the MOR context, [7] and [30] have used this technique for IRKA, which is used for MOR of non-parametric first-order dynamical systems (belonging to Cell 1 of Table 1.1).

As for stability, here too, we apply the precondition reuse technique to MOR algorithms belonging to other Cells of Table 1.1(i.e. 2, 3, and 4). That is, as earlier with focus of *linear* dynamical systems by reusing preconditioners, we accelerate convergence of MOR algorithms for non-parametric second-order, parametric first-order, and parametric second-order dynamical systems.

Besides *comprehensively* applying this technique to a wide array of MOR algorithms, we demonstrate the usefulness of using it in *a real-life industrial setting*, which has not been done in any of the previous works. We demonstrate that by reusing preconditioners, a 1.2 million problem can be reduced in 3 days instead of 8 days earlier.

Next, we provide an outline of the Chapters in this dissertation.

1.4 Organization of the Dissertation

The dissertation consists of five more Chapters. The stability analyses of MOR algorithms for non-parametric second-order dynamical systems and parametric first plus second-order dynamical systems are discussed in Chapter 2 and Chapter 3, respectively. Chapter 4 and Chapter 5 apply the theory of reusing preconditioners to the algorithms from Chapter 2 and Chapter 3, respectively. We provide our conclusions and future work in Chapter 6.

Chapter 2

Stability Analysis of Non-parametric MOR

As discussed in the Introduction, here we focus on MOR of non-parametric second-order dynamical systems. Adaptive Iterative Rational Global Arnoldi (AIRGA) [10], is a popular MOR algorithm belonging to this category. This algorithm uses *Ritz-Galerkin projection* and is used for reducing such dynamical systems with *proportional damping*. These systems have the form

$$\begin{aligned} M\ddot{x}(t) + D\dot{x}(t) + Kx(t) &= Bu(t), \\ y(t) &= Cx(t) = \mathscr{C}^T x(t), \end{aligned} \tag{2.1}$$

where $M,\ D,\ K \in \mathbb{R}^{n\times n},\ B \in \mathbb{R}^{n\times m},\ \mathscr{C} \in \mathbb{R}^{n\times q}$, and $D = \alpha M + \beta K$. Here, $\alpha,\ \beta$ are some scalar values. Let $V \in \mathbb{R}^{n\times r}$ and its columns span an r-dimensional subspace ($r \ll n$). In principle, the Ritz-Galerkin projection method involves the steps below.

- Approximating the reduced state vector $\hat{x}(t)$ using V as $x(t) \approx V\hat{x}(t)$ leads to

$$\begin{aligned} MV\ddot{\hat{x}}(t) + DV\dot{\hat{x}}(t) + KV\hat{x}(t) - Bu(t) &= r(t), \\ \hat{y}(t) &= \mathscr{C}^T V\hat{x}(t), \end{aligned}$$

  where $r(t)$ is the residual after projection.

- Enforcing the residual $r(t)$ to be orthogonal to V or $V^T r(t) = 0$ leads to the reduced system given as follows:

$$\begin{aligned} V^T \left(MV\ddot{\hat{x}}(t) + DV\dot{\hat{x}}(t) + KV\hat{x}(t) - Bu(t) \right) &= 0, \\ \hat{y}(t) = \ \mathscr{C}^T V \hat{x}(t). & \end{aligned} \tag{2.2}$$

or

$$\begin{aligned} \hat{M}\ddot{\hat{x}}(t) + \hat{D}\dot{\hat{x}}(t) + \hat{K}\hat{x}(t) - \hat{B}u(t) &= 0, \\ \hat{y}(t) = \hat{\mathscr{C}}^T \hat{x}(t), & \end{aligned} \tag{2.3}$$

where

$$\hat{M} = V^T M V, \ \hat{D} = V^T D V, \ \hat{K} = V^T K V, \ \hat{B} = V^T B, \text{ and } \hat{\mathscr{C}}^T = \mathscr{C}^T V. \tag{2.4}$$

To compute this projection matrix V, AIRGA matches the moments of the original system transfer function and the reduced system transfer function.

The transfer function of (2.1) is given by

$$H(s) = \mathscr{C}^T \left(s^2 M + sD + K \right)^{-1} B = \mathscr{C}^T X(s),$$

where $X(s) = \left(s^2 M + sD + K \right)^{-1} B$. The power series expansion of $X(s)$ around an expansion point $s_0 \in \mathbb{R}$ is given by (see, e.g., [31])

$$X(s) = \sum_{j=0}^{\infty} X^{(j)}(s_0) \left(s - s_0 \right)^j, \tag{2.5}$$

where,

$$\begin{aligned} X^{(0)}(s_0) &= \left(s_0^2 M + s_0 D + K \right)^{-1} B, \\ X^{(1)}(s_0) &= \left(s_0^2 M + s_0 D + K \right)^{-1} \left(-\left(2s_0 M + D \right) \right) X^{(0)}(s_0), \qquad \text{and} \\ X^{(j)}(s_0) &= \left(s_0^2 M + s_0 D + K \right)^{-1} \left[-\left(2s_0 M + D \right) X^{(j-1)}(s_0) - M X^{(j-2)}(s_0) \right], \end{aligned} \tag{2.6}$$

for $j = 2, 3, \ldots$. Here, $X^{(j)}(s_0)$ is called the j^{th}-order system moment at s_0.

Similarly, the transfer function of the reduced system (2.3) is given by

$$\hat{H}(s) = \hat{\mathscr{C}}^T \hat{X}(s),$$

where $\hat{X}(s) = \left(s^2\hat{M} + s\hat{D} + \hat{K}\right)^{-1}\hat{B}$. The power series expansion of $\hat{X}(s)$ around an expansion point $s_0 \in \mathbb{R}$ is given by

$$\hat{X}(s) = \sum_{j=0}^{\infty} \hat{X}^{(j)}(s_0)\,(s - s_0)^j\,. \tag{2.7}$$

The j^{th}-order system moment $\hat{X}^{(j)}(s_0)$ is defined analogously to $X^{(j)}(s_0)$ in (2.6).

The goal of moment-matching approach is to find a reduced system such that the first few moments of (2.5) and (2.7) are matched, that is, $X^{(j)}(s_0) = \hat{X}^{(j)}(s_0)$ for $j = 0, 1, 2, \ldots, t$ for some t. This can be achieved by the observation below. With

$$\begin{aligned}
P_1 &= -\left(s_0^2M + s_0D + K\right)^{-1}(2s_0M + D)\,,\\
P_2 &= -\left(s_0^2M + s_0D + K\right)^{-1}M,\\
Q &= \left(s_0^2M + s_0D + K\right)^{-1}B,
\end{aligned}$$

we have from (2.6)

$$\begin{aligned}
X^{(0)}(s_0) &= Q,\\
X^{(1)}(s_0) &= P_1X^{(0)}(s_0)\,, \qquad \text{and}\\
X^{(j)}(s_0) &= P_1X^{(j-1)}(s_0) + P_2X^{(j-2)}(s_0)
\end{aligned}$$

for $j \geq 2$. As already observed in [9], these moments are just the blocks of the second-order Krylov subspace

$$\mathbb{G}^j(P_1,\ P_2,\ Q) = \text{span}\{Q,\ \mathfrak{S}_1(P_1,\ P_2)\,Q,\ \mathfrak{S}_2(P_1,\ P_2)\,Q,\ \ldots,\ \mathfrak{S}_j(P_1,\ P_2)\,Q\},$$

where $\mathfrak{S}_j(P_1,\ P_2) = P_1\mathfrak{S}_{j-1}(P_1,\ P_2) + P_2\mathfrak{S}_{j-2}(P_1,\ P_2)$ for $j > 2$, with $\mathfrak{S}_1(P_1,\ P_2) = P_1$ and $\mathfrak{S}_2(P_1,\ P_2) = P_1^2 + P_2$.

For the special case of proportionally damped second-order linear systems, it has been observed in [32] that with $\mathscr{A} = (s_0^2M + s_0D + K)$

$$\begin{aligned}
\mathbb{G}^j(P_1,\ P_2,\ Q) &= \mathbb{G}^j\left(-\mathscr{A}^{-1}(2s_0M + D)\,,\ -\mathscr{A}^{-1}M,\ \mathscr{A}^{-1}B\right),\\
&= \mathbb{G}^j\left(-\mathscr{A}^{-1}((2s_0 + \alpha)M + \beta K)\,,\ -\mathscr{A}^{-1}M,\ \mathscr{A}^{-1}B\right),\\
&= \mathbb{K}^j\left(-\mathscr{A}^{-1}M,\ \mathscr{A}^{-1}B\right) = \mathbb{K}^j(P_2,\ Q)\,,
\end{aligned}$$

where $\mathbb{K}^j\left(P_2,\ Q\right)$ is the standard block Krylov subspace

$$\mathbb{K}^j\left(P_2, Q\right) = \text{span}\{Q,\ P_2Q,\ P_2^2Q,\ \ldots,\ P_2^{j-1}Q\}.$$

The reduced order system (2.3), which matches the first $\lceil r/m \rceil$ moments of the original system (2.1) can be obtained by projecting (2.1) with $\Pi = VV^T$ with an orthonormal matrix $V \in \mathbb{R}^{n\times r}$ whose columns span $\mathbb{K}^j(P_2, Q)$.

Standard efficient methods to compute the desired orthogonal basis of $\mathbb{K}^j\left(P_2,\ Q\right)$ are, e.g., the block or the global Arnoldi algorithm [18, 33, 34]. The AIRGA algorithm generates V by a global Arnoldi method. Its relevant parts are given in Algorithm 1. Unlike as discussed above, the AIRGA algorithm uses not just one expansion point, but a set of ℓ expansion points. This ensures a better reduced system in the entire frequency domain of interest. The method is adaptive, i.e. it automatically chooses the number of moments to be matched at each expansion point s_i. This is controlled by the inner `while` loop starting at line 9. The variable j stores the total number of moments matched. The upper bound on max value of j or J is $\lceil r_{\max}/m \rceil$, where $r_{\max}$ is the maximum dimension to which we want to reduce the state variable (input from the user), and m is the dimension of the input. For a thorough discussion on how to determine convergence, to choose the expansion points in the inner loop as well as a new set of expansion points in the outer loop, see [10].

Next, we discuss the stability analysis of using inexact linear solves in AIRGA.

2.1 Backward Stability Analysis

Let V be calculated exactly, and f be the functional representation of the exact MOR algorithm (that uses V during reduction process). Similarly, let $\widetilde{V}$ be calculated inexactly (i.e., by a Krylov subspace solver), and $\widetilde{f}$ be the functional representation of the inexact MOR algorithm (that uses $\widetilde{V}$ during reduction process). Then, from the backward stability definition, a MOR algorithm is backward stable with respect

to inexact linear solves if [35]

$$\widetilde{f}(x) = f(\widetilde{x}) \quad \text{for some } \widetilde{x} \text{ with} \tag{2.8}$$

$$\frac{\|x - \widetilde{x}\|_{H_2 \text{ or } H_\infty}}{\|x\|_{H_2 \text{ or } H_\infty}} = \mathcal{O}(\|Z\|), \tag{2.9}$$

where $\widetilde{x}$ is the perturbed full model corresponding to the error in the linear solves for $\widetilde{V}$ in the inexact MOR algorithm. This perturbation is denoted by Z. Further, H_2 and H_∞ denote the standard functional norms.

Here, the function f maps $H(s)$ to $\hat{H}(s)$ or $f(H(s)) = \hat{H}(s)$. This is represented by AIRGA when a direct solver for solving the linear systems at lines 5 and 14 is employed (see Algorithm 1). This is called the exact AIRGA algorithm.

The function $\tilde{f}$ maps the transfer function $H(s)$ of the original system to the transfer function of the reduced system employing an iterative solver in order to solve the linear systems at lines 5 and 14 in AIRGA (instead of a direct solver; again see Algorithm 1). This is denoted by $\tilde{f}(H(s)) = \tilde{\hat{H}}(s)$ and is called the inexact AIRGA algorithm.

For our discussion, we are only interested in one outer iteration step. The matrix $V = [V_1, V_2, \ldots, V_J]$ is generated and the reduced system is computed with V as in (2.4) (lines 26-28). This immediately gives $f(H(s))$ and $\tilde{f}(H(s))$ when using of direct solver and iterative solver, respectively. Further, we need to assume that the choice of the expansion points is the same no matter whether iterative solves or a direct solve is used.

Next, we analyze (2.8) and (2.9) separately in the below two subsections.

2.1.1 First Condition of Stability

Consider the linear systems for $X^{(0)}(s_i) \in \mathbb{R}^{n \times m}$ at line 5

$$\left(s_i^2 M + s_i D + K\right) X^{(0)}(s_i) = B,$$

where $s_i \in S = \{s_1,\ s_2,\ \ldots,\ s_\ell\}$. We denote the inexactly computed solution for $X^{(0)}(s_i)$ by $\tilde{X}^{(0)}(s_i)$. Let the associated residual be $\eta_{0i} \in \mathbb{R}^{n \times m}$ for $i = 1,\ \ldots,\ \ell$.

Algorithm 1 Adaptive Iterative Rational Global Arnoldi Algorithm [10]

Input: $\{M, D, K, B, \mathscr{C}$; error tolerances $(\epsilon_1,\ \epsilon_2)$; $r_{\max}$; initial set of expansion points $S = \{s_1, \ldots, s_\ell\}\}$

while the H_2 error between two consecutive reduced systems is greater than ϵ_1 **do**

  for each $s_i \in S$ **do**

    $X^{(-1)}(s_i) = 0$, $h_\pi^{(-1)} = 0$

    $X^{(0)}(s_i) = \left(s_i^2 M + s_i D + K\right)^{-1} B$, $h_\pi^{(0)} = 1$

    Also, get a good basis (vectors orthogonal to machine precision and normalized to length 1) of $X^{(0)}(s_i)$ via a QR decomposition

  end for

  j = 1

  while the H_2 error between two intermediate reduced systems is greater than ϵ_2 AND $j < \lceil r_{\max}/m \rceil$ **do**

    Choose an expansion point $\sigma_j \in S$; $\sigma_j = \text{argmax}_{s_i} \|h_\pi^{(j-1)} \mathscr{C}^T X^{(j-1)}(s_i)\|_f$

    $V_j = X^{(j-1)}(\sigma_j)/\|X^{(j-1)}(\sigma_j)\|_f$

    for $i = 1,\ \ldots,\ \ell$ **do**

      if $(s_i == \sigma_j)$ **then**

        $X^{(j)}(s_i) = -\left(s_i^2 M + s_i D + K\right)^{-1} M V_j$, $h_\pi^{(j)} = h_\pi^{(j-1)} \|X^{(j-1)}(s_i)\|_f$

      else

        $X^{(j)}(s_i) = X^{(j-1)}(s_i)$, $h_\pi^{(j)} = h_\pi^{(j-1)}$

      end if

      for $t = 1,\ 2,\ \ldots,\ j$ **do**

        $\gamma_{t,j}(s_i) = \text{trace}(V_t^H \cdot X^{(j)}(s_i))\ X^{(j)}(s_i) = X^{(j)}(s_i) - \gamma_{t,j}(s_i) V_t$

      end for

    end for

    j = j+1

    Compute temporary reduced system matrices using the intermediate V and (2.4)

  end while

  Set $J = j$ and pick $\sigma_J \in S$

  $V_J = X^{(J-1)}(\sigma_J)/||X^{(J-1)}(\sigma_J)||_f$ and $V = [V_1,\ V_2,\ \ldots,\ V_J]$

  Also, get a good basis of V via a QR decomposition

  Compute the reduced order system matrices $\hat{M}$, $\hat{D}$ and $\hat{K}$ with V as in (2.4)

  Choose new set of expansion points $S = \{s_1, \ldots, s_\ell\}$ using eigenvalues of the reduced system

end while

Compute the reduced order system matrices $\hat{B}$, and $\hat{\mathscr{C}}$ with V as in (2.4)

Then, the above equation is equivalent to

$$\left(s_i^2 M + s_i D + K\right) \tilde{X}^{(0)}\left(s_i\right) = B + \eta_{0i}. \tag{2.10}$$

All $\tilde{X}^{(0)}(s_i)$ are used at line 10 for picking the best expansion point for this first step, which is denoted by σ_1 with $\eta_{(0)}$ has the corresponding residual. Next, in Algorithm 1 at line 11, at the first iteration of the `while` loop (i.e. j=1), $\tilde{V}_1$ is computed as (as above, here $\tilde{}$ is added because of the inexactness)

$$\tilde{V}_1 = \tilde{X}^{(0)}\left(\sigma_1\right) / \|\tilde{X}^{(0)}\left(\sigma_1\right)\|_f, \tag{2.11}$$

Further, at line 14 in Algorithm 1 the inexact solve gives

$$\left(\sigma_1^2 M + \sigma_1 D + K\right) \tilde{X}^{(1)}\left(\sigma_1\right) = -M\tilde{V}_1 + \eta_1. \tag{2.12}$$

$\tilde{X}^{(1)}(s_i)$ will be equal to $\tilde{X}^{(0)}(s_i)$, $\forall s_i \in S = \{s_1, s_2, \ldots, s_\ell\} \backslash \{\sigma_1\}$. As above, all $\tilde{X}^{(1)}(s_i)$ are used at line 10 for picking the best expansion point at this second step, which is denoted by σ_2 with $\eta_{(1)}$ as the corresponding residual. Next, in Algorithm 1 at line 11 after one iteration of the `while loop` (i.e. j=2), $\tilde{V}_2$ is computed as

$$\tilde{V}_2 = \tilde{X}^{(1)}\left(\sigma_2\right) / \|\tilde{X}^{(1)}\left(\sigma_2\right)\|_f. \tag{2.13}$$

Further, at line 14 the inexact solve yields for $j = 2, \ldots, J-1$

$$\left(\sigma_j^2 M + \sigma_j D + K\right) \tilde{X}^{(j)}\left(\sigma_j\right) = -M\tilde{V}_j + \eta_j. \tag{2.14}$$

$\tilde{X}^{(j)}(s_i)$ will be equal to $\tilde{X}^{(j-1)}(s_i)$, $\forall s_i \in S = \{s_1, s_2, \ldots, s_\ell\} \backslash \{\sigma_j\}$. As done for first and second step, all $\tilde{X}^{(j-1)}(s_i)$ are used at line 10 for picking the best expansion point at the j^{th} step, which is denoted by σ_j with $\eta_{(J-1)}$ as the corresponding residual. Thus, in Algorithm 1 at line 11 for $j = 3, \ldots, J-1$ and at line 26 for $j = J$, $\tilde{V}_j$ is computed as

$$\tilde{V}_j = \tilde{X}^{(j-1)}\left(\sigma_j\right) / \|\tilde{X}^{(j-1)}\left(\sigma_j\right)\|_f, \tag{2.15}$$

Finally, $\tilde{V} = \left[\tilde{V}_1,\ \tilde{V}_2,\ \ldots,\ \tilde{V}_J\right]$ is set up and used to generate the reduced system (obtained by the inexact AIRGA algorithm),

$$\begin{aligned} &\tilde{\hat{M}} = \tilde{V}^T M \tilde{V},\ \tilde{\hat{D}} = \tilde{V}^T D \tilde{V},\ \tilde{\hat{K}} = \tilde{V}^T K \tilde{V}, \\ &\tilde{\hat{B}} = \tilde{V}^T B, \text{ and } \tilde{\hat{\mathscr{C}}}^T = \mathscr{C}^T \tilde{V}. \end{aligned} \tag{2.16}$$

This reduced order system is equivalent to $\tilde{f}\left(H(s)\right)$.

Now we have to find a perturbed original system $\tilde{H}\left(s\right)$, such that the exact AIRGA on it or $f(\tilde{H}(s))$, will give the reduced system as obtained $\big($by applying inexact AIRGA on the original full system or $\tilde{f}\left(H(s)\right)\big)$. That is, find $\tilde{H}(s)$ such that $\tilde{f}\left(H(s)\right) = f(\tilde{H}(s))$. This will satisfy the first stability condition (2.8).

Among the many systems $\tilde{H}\left(s\right)$ one can consider here, we concentrate on those that have a constant perturbation $Z \in \mathbb{R}^{n\times n}$ in K only. That is,

$$\tilde{K} = K + Z,\ \tilde{M} = M,\ \tilde{D} = D,\ \tilde{B} = B,\ \text{and}\ \tilde{\mathscr{C}} = \mathscr{C}.$$

Although in (2.10), only one linear system's data is used in deciding $\tilde{V}_1$, which is $\tilde{X}^{(0)}(\sigma_1)$. However, as mentioned earlier, all these linear solves $\tilde{X}^{(0)}(s_i)$ are used in picking σ_1. Then, for $\tilde{H}$ we have that instead of (2.10), $\tilde{X}^{(0)}(s_i)$ is the exact solution of

$$\left(s_i^2 M + s_i D + (K+Z)\right)\tilde{X}^{(0)}\left(s_i\right) = B, \tag{2.17}$$

for $i = 1, 2, \ldots, \ell$. Similarly, it follows that the linear systems (2.12) and (2.14) are solved exactly as

$$\left(\sigma_j^2 M + \sigma_j D + (K+Z)\right)\tilde{X}^{(j)}\left(\sigma_j\right) = -M\tilde{V}_j, \tag{2.18}$$

for $j = 1, \ldots, J-1$, where σ_j is the expansion point picked at the j^{th} step.

The final matrix $\tilde{V} = \left[\tilde{V}_1,\ \tilde{V}_2, \ldots,\ \tilde{V}_J\right]$ is exactly the same as before since

(a) $\tilde{X}^{(0)}\left(\sigma_1\right)$ in (2.17) is the same as that of (2.10) as well as $\tilde{X}^{(j)}\left(\sigma_j\right)$ in (2.18) is the same as that in (2.12), (2.14), and

(b) $\tilde{V}_j$ for $j = 1, \ldots, J$ are still given by (2.11), (2.13) and (2.15).

Thus, the reduced order system $\big($obtained by the exact AIRGA algorithm applied

to the perturbed system $\tilde{H}\big)$ is given by

$$\begin{aligned}
\hat{\tilde{M}} &= \tilde{V}^T \tilde{M} \tilde{V} = \tilde{V}^T M \tilde{V} = \tilde{\tilde{M}}, \\
\hat{\tilde{D}} &= \tilde{V}^T \tilde{D} \tilde{V} = \tilde{V}^T D \tilde{V} = \tilde{\tilde{D}}, \\
\hat{\tilde{K}} &= \tilde{V}^T \tilde{K} \tilde{V} = \tilde{V}^T (K + Z) \tilde{V} = \tilde{\tilde{K}} + \tilde{V}^T Z \tilde{V}, \\
\hat{\tilde{B}} &= \tilde{V}^T \tilde{B} = \tilde{V}^T B = \tilde{\tilde{B}}, \text{ and} \\
\hat{\tilde{\mathscr{C}}}^T &= \tilde{\mathscr{C}}^T \tilde{V} = \mathscr{C}^T \tilde{V} = \tilde{\tilde{\mathscr{C}}}^T.
\end{aligned} \tag{2.19}$$

This reduced order system is equivalent to $f\left(\tilde{H}(s)\right)$. Obviously, this is already almost the same as $\tilde{f}\left(H(s)\right)\Big($recall that our goal is to find $\tilde{H}\left(s\right)$ such that $\tilde{f}\left(H(s)\right) = f\left(\tilde{H}(s)\right)\Big)$. Thus, we need to find Z such that $\hat{\tilde{K}} = \tilde{\tilde{K}}$ or $\tilde{V}^T Z \tilde{V} = 0$.

If we look at the inexact solves in (2.10), (2.12) and (2.14), and the corresponding perturbed solves in (2.17) and (2.18), we find that both are equivalent and a total of $\ell + J - 1$ linear systems are solved. Since the dimension of $\tilde{V}$ is only J, we further work with only those linear systems that form our $\tilde{V}$ and ignore the remaining systems. Putting all these linear systems together we get

$$Z\ \mathbf{X} = \eta, \tag{2.20}$$

where $\mathbf{X}$ is formed by stacking the relevant block columns of $X^{(j)}(\sigma_j)$ or $\mathbf{X} = \Big[\tilde{X}^{(0)}\left(\sigma_1\right),\ \tilde{X}^{(1)}\left(\sigma_2\right),\ \ldots,\ \tilde{X}^{(J-1)}\left(\sigma_J\right)\Big]$; similarly, after stacking the relevant block columns of η_j together we get $\eta = \big[-\eta_{(0)},\ \ldots, -\eta_{(J-1)}\big]$.

In the above equation, we can replace $\mathbf{X}$ in-terms of $\tilde{V}$ by using (2.11), (2.13), and (2.15). That is, (2.20) can be rewritten as

$$Z\tilde{V}\mathscr{D}_X^{-1} = \eta \quad or \quad Z\tilde{V} = \eta\mathscr{D}_X, \tag{2.21}$$

where $\mathscr{D}_X = \begin{bmatrix} \frac{1}{\left\|\tilde{X}^{(0)}(\sigma_1)\right\|_f} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{\left\|\tilde{X}^{(1)}(\sigma_2)\right\|_f} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{\left\|\tilde{X}^{(J-1)}(\sigma_J)\right\|_f} \end{bmatrix}$.

Multiplying $\tilde{V}^T$ from the left side of (2.21), we get

$$\tilde{V}^T Z \tilde{V} = \tilde{V}^T \eta \mathcal{D}_X. \tag{2.22}$$

Assume that we are using a Ritz-Galerkin based iterative solver. Here, the solution space of the linear systems is orthogonal to the corresponding residuals, i.e. $\tilde{V}_1 \perp \eta_{(0)},\ \tilde{V}_2 \perp \eta_{(1)},\ \ldots$,and $\tilde{V}_J \perp \eta_{(J-1)}$ [36]. Hence,

$$\begin{aligned} \tilde{V}^T \eta &= - \begin{bmatrix} \tilde{V}_1^T \\ \tilde{V}_2^T \\ \vdots \\ \tilde{V}_{J-1}^T \\ \tilde{V}_J^T \end{bmatrix} \begin{bmatrix} \eta_{(0)} & \eta_{(1)} & \ldots & \eta_{(J-1)} \end{bmatrix}, \\ &= - \begin{bmatrix} 0 & \tilde{V}_1^T \eta_{(1)} & \ldots & \tilde{V}_1^T \eta_{(J-2)} & \tilde{V}_1^T \eta_{(J-1)} \\ \tilde{V}_2^T \eta_{(0)} & 0 & \ldots & \tilde{V}_2^T \eta_{(J-2)} & \tilde{V}_2^T \eta_{(J-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{V}_{J-1}^T \eta_{(0)} & \tilde{V}_{J-1}^T \eta_{(1)} & \ldots & 0 & \tilde{V}_{J-1}^T \eta_{(J-1)} \\ \tilde{V}_J^T \eta_{(0)} & \tilde{V}_J^T \eta_{(1)} & \ldots & \tilde{V}_J^T \eta_{(J-2)} & 0 \end{bmatrix}. \end{aligned} \tag{2.23}$$

Our goal here is to make the right hand side of the above equation equal to zero. The upper triangular part of the above matrix is zero if we have the following orthogonalities:

$$\begin{aligned} \begin{bmatrix} \tilde{V}_1 \end{bmatrix} &\perp \eta_{(1)}, \\ \begin{bmatrix} \tilde{V}_1 & \tilde{V}_2 \end{bmatrix} &\perp \eta_{(2)}, \\ &\vdots \\ \begin{bmatrix} \tilde{V}_1 & \tilde{V}_2 & \tilde{V}_3 & \ldots & \tilde{V}_{J-2} \end{bmatrix} &\perp \eta_{(J-2)}, \\ \begin{bmatrix} \tilde{V}_1 & \tilde{V}_2 & \tilde{V}_3 & \ldots \tilde{V}_{J-2} & \tilde{V}_{J-1} \end{bmatrix} &\perp \eta_{(J-1)}. \end{aligned} \tag{2.24}$$

Similarly, for the lower triangular part of the above matrix to be zero we need the

following orthogonalities:

$$\begin{aligned}
\tilde{V}_2 &\perp \left[\eta_{(0)}\right], \\
\tilde{V}_3 &\perp \left[\eta_{(0)}\ \eta_{(1)}\right], \\
&\vdots \\
\tilde{V}_{J-1} &\perp \left[\eta_{(0)}\ \eta_{(1)}\ \ldots\ \eta_{(J-3)}\right], \\
\tilde{V}_J &\perp \left[\eta_{(0)}\ \eta_{(1)}\ \ldots\ \eta_{(J-3)}\ \eta_{(J-2)}\right].
\end{aligned} \tag{2.25}$$

At the first glance, there seem to be two problems in achieving the above discussed orthogonalities in an iterative solver. One is the amount of code changes to be done. The other is the extra cost associated at every iterative step of the solver, which may undermine the benefit of using an iterative solver itself. In Section 2.2.2, we show that both these issues can be easily resolved by using a recycling variant of the underlying iterative solver (briefly summarized below).

While solving a sequence of linear systems, if the consecutive systems do not change much, then some information can be reused from solving one linear system to solving the next. In the context of Krylov based iterative linear solvers, this information is in the form of the generated Krylov subspace. The process of reusing Krylov subspaces from one linear system to the next is termed as "recycling" [37, 38, 39, 40].

A subset of $\tilde{V}$'s and η's of (2.24) and (2.25) can be used to span a recycle space, leading to almost no code changes in the recycling variant of the underlying iterative solver. In some cases, this choice of the recycle space can actually accelerate the convergence of the next linear system in the sequence. In case when this recycle space deteriorates the convergence of the next linear system, this behaviour is bounded. In the numerical experiments section (Section 2.4), we support both these conjectures (acceleration and deterioration of the convergence of iterative linear solvers) with multiple examples.

Therefore, after applying (2.23), (2.24) and (2.25) to (2.22), we get $\tilde{V}^T Z \tilde{V} = 0$. Thus, $\hat{\tilde{K}} = \tilde{\hat{K}}$ or

$$\tilde{f}\left(H\left(s\right)\right) = f\left(\tilde{H}\left(s\right)\right) = \tilde{\hat{H}}\left(s\right),$$

where $H(s) = \mathscr{C}^T (s^2M + sD + K)^{-1} B$,
$\tilde{H}(s) = \mathscr{C}^T (s^2M + sD + (K + Z))^{-1} B$, and
$\hat{\tilde{H}}(s) = \hat{\mathscr{C}}^T \left(s^2\hat{\tilde{M}} + s\hat{\tilde{D}} + \hat{\tilde{K}}\right)^{-1} \hat{\tilde{B}} = \hat{\mathscr{C}}^T \left(s^2\hat{\tilde{M}} + s\hat{\tilde{D}} + \hat{\tilde{K}}\right)^{-1} \hat{\tilde{B}}$. Thus, we satisfy the first condition of stability.

2.1.2 Second Condition for Stability

According to the second condition of stability, given in (2.9), the difference between the unperturbed (original) full system and the perturbed full system should be of the order of the perturbation [35]. These errors are measured in the commonly used norms as below.

$$H_2 - \textbf{norm} \qquad \|H - G\|_{H_2} = \frac{1}{2\pi} \int_{-\infty}^{\infty} \|H(\imath\omega) - G(\imath\omega)\|_f d\omega,$$

$$H_\infty - \textbf{norm} \qquad \|H - G\|_{H_\infty} = \max_{\omega \in \mathbb{R}} \|H(\imath\omega) - G(\imath\omega)\|_2,$$

where the transfer functions H and G belong to systems with the same input and output dimension. Theorem 4.3 from [21] helps in giving the desired result.

Theorem 2.1 *If* $\|Z\|_2 < \frac{1}{\|A(s)^{-1}\|_{H_\infty}}$ *then*

$$\|H(s) - \tilde{H}(s)\|_{H_2} \leq \frac{\|A(s)^{-1}B\|_{H_\infty} \|\mathscr{C}^T A(s)^{-1}\|_{H_2}}{1 - \|A(s)^{-1}\|_{H_\infty} \|Z\|_2} \|Z\|_2, \tag{2.26}$$

where $A(s) = (s^2M + sD + K)$.

Proof: see Appendix A.

If $\|Z\|_2 < 1$ and $\|A(s)^{-1}\|_{H_\infty} < 1$, then we have $\|A(s)^{-1}\|_{H_\infty} \|Z\|_2 < 1$, and hence,

$$\frac{1}{1 - \|A(s)^{-1}\|_{H_\infty} \|Z\|_2} < \frac{1}{1 - \|A(s)^{-1}\|_{H_\infty}}. \tag{2.27}$$

Substituting (2.27) in (2.26) we get

$$\begin{aligned} \frac{\|H(s) - \tilde{H}(s)\|_{H_2}}{\|H(s)\|_{H_2}} &\leq \frac{\|A(s)^{-1}B\|_{H_\infty} \|\mathscr{C}^T A(s)^{-1}\|_{H_2}}{\|H(s)\|_{H_2}} \cdot \frac{1}{1 - \|A(s)^{-1}\|_{H_\infty}} \cdot \|Z\|_2 \\ &= \mathcal{O}(\|Z\|_2). \end{aligned} \tag{2.28}$$

This satisfies the second condition of stability.

The next theorem summarizes our complete stability analysis

Theorem 2.2 *If the linear systems arising in the AIRGA algorithm are solved by*

(a) Ritz-Galerkin based solver (i.e. the residual is orthogonal to the generated Krylov subspace),

(b) the extra orthogonalities given by (2.24) and (2.25) are satisfied by such a solver,

(c) $A(s)$ *as defined in Theorem 2.1 is invertible and* $\|A(s)^{-1}\|_{H_\infty} < 1$*,*

(d) Z *is given by (2.20) exists and* $\|Z\|_2 < 1$*,*

then the AIRGA algorithm is backward stable with respect to the inexact linear solves.

If we look at Algorithm 1, besides the linear solves at lines 5 and 14, we are also concerned about the construction of V from X (since V gives us our reduced system). There are three places in code where X is modified further to obtain V rather than just normalizing X to V (on lines 11 and 26). First is at line 6, where a QR decomposition of X is done. Second, an Arnoldi iteration on X is done at lines 18–20. Finally, the QR decomposition of V is done at line 27. All these code changes are nothing but an attempt to get a good basis (vectors orthogonal to machine precision and normalized to length 1) of X and V, which have negligible effect on our analysis. Hence, for ease of exposition, we ignore them.

Next, we discuss how to satisfy the backward stability conditions given by Theorem 2.2

2.2 Satisfying Backward Stability Conditions

In this section, we analyze the hypothesis of Theorem 2.2 as so to achieve a backward stable AIRGA, we mostly focus on conditions (a) and (b) and not (c) and (d). Condition (c) cannot be worked upon much because it is dependent on the expansion points and the input dynamical system. Condition (d) does not create much challenges because, as discussed in the next section, perturbation is directly proportional to the residuals, which can be iteratively controlled.

From condition (a) of Theorem 2.2 we know that we need to use a Ritz-Galerkin based method for solving the underlying linear systems in AIRGA. The Conjugate Gradient (CG) method is one of the most popular solver of such a type. The CG method is mainly used for solving Symmetric Positive Definite (SPD) linear systems. For solving non-symmetric linear systems, Full Orthogonalization Method (FOM) [33, 41] is the one that is based upon the Ritz-Galerkin theory.

In this work, we focus on the CG method, and hence, in the results section, we take models that lead to SPD linear systems in the AIRGA algorithm. FOM method can be similarly used.

Next, we first discuss how to change the theory of the CG method such that condition (b) of Theorem 2.2 or the extra orthogonalities, (2.24)-(2.25), are satisfied (in Section 2.2.1). Further, we describe how the recommended changes can be easily implemented (in Section 2.2.2).

2.2.1 Achieving Extra Orthogonalities

The CG method consists of two components. One is the Lanczos algorithm that gives a good basis of the generated Krylov subspace. Such a basis has vectors orthogonal to machine precision and normalized to length 1. The other is the Ritz-Galerkin projection to obtain solution estimates from this subspace. The orthogonalities in (2.25) can be achieved by modifying the Lanczos algorithm (discussed in Section 2.2.1.1), and those in (2.24) can be achieved by changing the Ritz-Galerkin projection (discussed in Section 2.2.1.2).

2.2.1.1 Adapting the Lanczos Process

Assume we are trying to solve the linear system of the form

$$Ax = b, \tag{2.29}$$

where $A \in \mathbb{C}^{n\times n}$ and $b \in \mathbb{C}^n$. Let x_0 be the initial solution vector with $r_0 = b - Ax_0$ as the corresponding residual. The Lanczos algorithm computes a good basis of the

generated Krylov subspace involving A and r_0 as [18]

$$\begin{aligned} w_{k+1} \ &\in \ \mathbb{K}^k\left(A, r_0\right) \equiv \text{span}\{r_0, Ar_0, A^2r_0, \cdots, A^{k-1}r_0\} \\ &s.t. \qquad w_{k+1} \perp \begin{bmatrix} w_1 & w_2 & \cdots & w_k \end{bmatrix}, \end{aligned} \tag{2.30}$$

where w_{k+1} is the Lanczos vector at the $(k+1)^{th}$ iterative step and $w_1 = r_0/\|r_0\|$[1]. Now, assume we are carrying some residual vector $\tilde{r}$ from another linear system, which we need to make orthogonal to the final solution of (2.29). Then, the Lanczos algorithm above would consist of the following procedure:

$$\begin{aligned} &w_{k+1} \in \mathbb{K}^k\left(A, r_0\right) \\ s.t. \qquad &w_{k+1} \perp \begin{bmatrix} w_1 & w_2 & \cdots w_k & \underbrace{\tilde{r}} \end{bmatrix}. \end{aligned}$$

Recall from the previous sections that in AIRGA, the first set of linear systems to be solved iteratively are given by (2.10). As mentioned in the paragraph between (2.10)–(2.11), the expansion point chosen is σ_1, and hence, in the linear system playing a role in our stability analysis is

$$\left(\sigma_1^2 M + \sigma_1 D + K\right) \tilde{X}^{(0)}\left(\sigma_1\right) = B + \eta_{(0)}. \tag{2.31}$$

Next, we need to iteratively solve (2.12), i.e.

$$\left(\sigma_1^2 M + \sigma_1 D + K\right) \tilde{X}^{(1)}\left(\sigma_1\right) = -M\tilde{V}_1 + \eta_1. \tag{2.32}$$

Here, we need a good basis (vectors orthogonal to machine precision and normalized to length 1) of the Krylov subspace involving the coefficient matrix $\mathscr{K}_1 = (\sigma_1^2 M + \sigma_1 D + K)$ and $(\eta_1)_0$, which is the initial residual of (2.32). Hence, the Lanczos algorithm here would consist of the following procedure:

$$\begin{aligned} &(w_1)_{k+1} \in \mathbb{K}^k\left(\mathscr{K}_1, (\eta_1)_0\right) \\ &s.t. \qquad (w_1)_{k+1} \perp \left[(w_1)_1 \ (w_1)_2 \ \cdots \ (w_1)_k\right], \end{aligned} \tag{2.33}$$

[1]Here, the first equation of (2.30) is implemented using

$$w_{k+1} = Aw_k - c_1 w_1 - c_2 w_2 - \ldots - c_{k-1} w_{k-1} - c_k w_k.$$

Finally, the second equation of (2.30) gives us $c_1, c_2, \ldots, c_k$. For a complete derivation of this, see chapter 5 of [36].

where $(w_1)_{k+1}$ is the Lanczos vector at the $(k+1)^{th}$ iterative step and $(w_1)_1 = (\eta_1)_0 / \| (\eta_1)_0 \|$. At this stage it is not clear if the solution of (2.32), i.e. $\tilde{X}^{(1)}(\sigma_1)$, would be used to form $\tilde{V}_2$ or not (see the discussion between (2.12)–(2.13); equivalently lines 10–11 of Algorithm 1). However, to avoid repeating solving (2.32) in case its solution is used to form $\tilde{V}_2$, we adapt the Lanczos procedure given by (2.33) as

$$\begin{aligned} & (w_1)_{k+1} \in \mathbb{K}^k \left(\mathscr{K}_1, (\eta_1)_0 \right) \\ s.t. \qquad & (w_1)_{k+1} \perp \left[(w_1)_1 \ (w_1)_2 \ \cdots \ (w_1)_k \ \underbrace{\eta_{(0)}} \right], \end{aligned} \tag{2.34}$$

where $\eta_{(0)}$ is the final residual obtained after solving (2.31) iteratively.

Next, the expansion point σ_2 is chosen (as above, see paragraph between (2.12)–(2.13) or lines 10–11 of Algorithm 1). If σ_2 turns to be equal to σ_1 (i.e. $\sigma_2 = \sigma_1$), then $\tilde{V}_2 = \tilde{X}^{(1)}(\sigma_1)/\|\tilde{X}^{(1)}(\sigma_1)\|_f$, and we would be satisfied the *first* orthogonality of (2.25), i.e. $\tilde{V}_2 \perp [\eta_{(0)}]$.

If σ_2 turns to be not equal to σ_1 (say $\sigma_2 = s_i \neq \sigma_1$), then $\tilde{V}_2 = \tilde{X}^{(1)}(s_i)/\|\tilde{X}^{(1)}(s_i)\|_f = \tilde{X}^{(0)}(s_i)/\|\tilde{X}^{(0)}(s_i)\|_f$ with $\tilde{X}^{(0)}(s_i)$ given by (2.10) or

$$\left(s_i^2 M + s_i D + K \right) \tilde{X}^{(0)} (s_i) = B + \eta_{0i}, \tag{2.35}$$

which we would have already solved once. Hence, to satisfy the *first* orthogonality of (2.25) or $\tilde{V}_2 \perp [\eta_{(0)}]$, we would need to resolve (2.10) or (2.35) by adapting its Lanczos process as given in (2.34). That is, carry extra $\eta_{(0)}$ in its Krylov subspace.

Next, we need to iteratively solve (2.14) for $j = 2$, i.e.

$$\left(\sigma_2^2 M + \sigma_2 D + K \right) \tilde{X}^{(2)} (\sigma_2) = -M \tilde{V}_2 + \eta_2. \tag{2.36}$$

Here, we need a good basis (vectors orthogonal to machine precision and normalized to length 1) of the Krylov subspace involving the coefficient matrix $\mathscr{K}_2 = (\sigma_2^2 M + \sigma_2 D + K)$ and $(\eta_2)_0$, which is the initial residual of (2.36). Hence, the Lanczos algorithm here would consist of the following procedure:

$$\begin{aligned} & (w_2)_{k+1} \in \mathbb{K}^k \left(\mathscr{K}_2, (\eta_2)_0 \right) \\ s.t. \qquad & (w_2)_{k+1} \perp \left[(w_2)_1 \ (w_2)_2 \ \cdots \ (w_2)_k \right], \end{aligned} \tag{2.37}$$

where $(w_2)_{k+1}$ is the Lanczos vector at the $(k+1)^{th}$ iterative step and $(w_2)_1 = (\eta_2)_0 / \| (\eta_2)_0 \|$.

As earlier, at this stage it is not clear if the solution of (2.36), i.e. $\tilde{X}^{(2)}(\sigma_2)$, would be used to form $\tilde{V}_3$ or not (see the discussion between (2.14)–(2.15); equivalently lines 10–11 of Algorithm 1). However to avoid repeating solving (2.36) in case its solution is used to form $\tilde{V}_3$, we adapt the Lanczos procedure given by (2.37) as

$$\begin{aligned} & & (w_2)_{k+1} \in \mathbb{K}^k \left(\mathcal{H}_2, (\eta_2)_0\right) \\ s.t. \qquad & (w_2)_{k+1} \perp \left[(w_2)_1 \ (w_2)_2 \ \cdots \ (w_2)_k \ \underbrace{\eta_{(0)} \ \eta_{(1)}} \right], \end{aligned} \tag{2.38}$$

where $\eta_{(1)}$ is the final residual obtained after solving (2.32) iteratively.

Next, the expansion point σ_3 is chosen (as above, see paragraph between (2.14)–(2.15) or equivalently lines 10–11 of Algorithm 1). If σ_3 turns to be equal to σ_2 (i.e. $\sigma_3 = \sigma_2$), then $\tilde{V}_3 = \tilde{X}^{(2)}(\sigma_2)/\|\tilde{X}^{(2)}(\sigma_2)\|_f$, and we would have satisfied the *second* set of orthogonalities of (2.25), i.e. $\tilde{V}_3 = [\eta_{(0)} \ \eta_{(1)}]$.

If σ_3 turns to be not equal to σ_2 (i.e. $\sigma_3 \neq \sigma_2$), then it may be equal to σ_1 or some other s_i. In case $\sigma_3 = \sigma_1 \neq \sigma_2$, than $\tilde{V}_3 = \tilde{X}^{(2)}(\sigma_1)/\|\tilde{X}^{(2)}(\sigma_1)\|_f = \tilde{X}^{(1)}(\sigma_1)/\|\tilde{X}^{(1)}(\sigma_1)\|_f$ with $\tilde{X}^{(1)}(\sigma_1)$ given by (2.12) or (2.32), which we would have already solved once. Hence, to satisfy the *second* set of orthogonalities of (2.25) or $\tilde{V}_3 \perp [\eta_{(0)} \ \eta_{(1)}]$, we would need to resolve (2.12) or (2.32) by adapting its Lanczos process as given in (2.38). That is, carry extra $\eta_{(0)}$ and $\eta_{(1)}$ in its Krylov subspace.

In case $\sigma_3 = s_i$ with $s_i \neq \sigma_1$, and $s_i \neq \sigma_2$, then $\tilde{V}_3 = \tilde{X}^{(2)}(s_i)/\|\tilde{X}^{(2)}(s_i)\|_f = \tilde{X}^{(1)}(s_i)/\|\tilde{X}^{(1)}(s_i)\|_f = \tilde{X}^{(0)}(s_i)/\|\tilde{X}^{(0)}(s_i)\|_f$ with $\tilde{X}^{(0)}(s_i)$ again given by (2.10) or

$$\left(s_i^2 M + s_i D + K\right) \tilde{X}^{(0)}(s_i) = B + \eta_{0i}, \tag{2.39}$$

which we would have already solved once. Hence, to satisfy the *second* set of orthogonalities of (2.25) or $\tilde{V}_3 \perp [\eta_{(0)} \ \eta_{(1)}]$, we would need to resolve (2.10) or (2.39) by adapting its Lanczos process as given in (2.38). That is, carry extra $\eta_{(0)}$ and $\eta_{(1)}$ in its Krylov subspace.

We need to repeat a similar procedure for (2.14) for all $j = 3, \ldots, J-1$.

2.2.1.2 Adapting the Ritz-Galerkin Projection

Recall that if we were trying to solve the linear system given in (2.29) by the CG method, then (2.30) gives a good basis (vectors orthogonal to machine precision and normalized to length 1) of the generated Krylov subspace. The solution update at the k^{th} iterative step is given as [18]

$$x_k = x_0 + \zeta_k, \tag{2.40}$$

where $\zeta_k = W_k y_k$ and $W_k = [w_1 \; w_2 \; \ldots \; w_k]$ with the columns of this matrix given by (2.30). In the CG method, this y_k is defined by a Ritz-Galerkin projection

$$r_k \perp W_k, \tag{2.41}$$

where $r_k = b - A(x_0 + \zeta_k) = b - A(x_0 + W_k y_k) = r_0 - AW_k y_k$. Now, assume we are carrying some solution vector $\tilde{x}$ from another linear system, which we need to make orthogonal to the final residual of (2.29). Then, the Ritz-Galerkin projection as above would consists of the following procedure:

$$r_k \perp \Big[W_k \quad \underbrace{\tilde{x}} \Big]. \tag{2.42}$$

Let us now look at the second linear system to solve in the AIRGA algorithm, i.e. (2.32). For this, a good basis of the generated Krylov subspace is given by (2.33). To find the solution vector here, the Ritz-Galerkin projection is defined as

$$(\eta_1)_k \perp (W_1)_k \,, \tag{2.43}$$

where $(\eta_1)_k$ is the residual of (2.32) at the k^{th} iterative step and $(W_1)_k = \big[(w_1)_1 \; (w_1)_2 \; \cdots \; (w_1)_k \big]$ with the columns of this matrix given by (2.33). Note that, as earlier, η_1 is the final residual of (2.32) (at convergence of CG).

As earlier, at this stage it is not clear if the residual of (2.32), i.e. η_1 would be the residual we care, about, i.e. $\eta_{(1)}$. These two residuals map to the fact whether $\tilde{X}^{(1)}(\sigma_1)$ would be used to form $\tilde{V}_2$ or not (again see the discussion between (2.12)–(2.13); equivalently lines 10–11 of Algorithm 1). However, to avoid repeating solving

(2.32) in case its solution is used to form $\tilde{V}_2$, we adapt the projection given by (2.43) as

$$(\eta_1)_k \perp \left[(W_1)_k \underbrace{\tilde{V}_1}\right], \tag{2.44}$$

where $\tilde{V}_1$ is given by (2.11).

Next, the expansion point σ_2 is chosen (as above, see paragraph between (2.12)–(2.13); equivalently lines 10–11 of Algorithm 1). If σ_2 turns to be equal to σ_1 (i.e. $\sigma_2 = \sigma_1$), then $\tilde{V}_2 = \tilde{X}^{(1)}(\sigma_1)/\|\tilde{X}^{(1)}(\sigma_1)\|_f$, and we would be satisfied the *first* orthogonality of (2.24), i.e. $\tilde{V}_1 \perp [\eta_{(1)}]$.

If σ_2 turns to be not equal to σ_1 (say $\sigma_2 = s_i \neq \sigma_1$), then $\tilde{V}_2 = \tilde{X}^{(1)}(s_i)/\|\tilde{X}^{(1)}(s_i)\|_f = \tilde{X}^{(0)}(s_i)/\|\tilde{X}^{(0)}(s_i)\|_f$ with $\tilde{X}^{(0)}(s_i)$ given by (2.10) or

$$\left(s_i^2 M + s_i D + K\right) \tilde{X}^{(0)}(s_i) = B + \eta_{0i}, \tag{2.45}$$

which we would have already solved. Hence, to satisfy the *first* orthogonality of (2.24) or $\tilde{V}_1 \perp [\eta_{(1)}]$, we would need to resolve (2.45) by adapting its projection as given in (2.44). That is, carrying extra $\tilde{V}_1$ in its Krylov subspace.

Similarly, all the other orthogonalities of (2.24) can be achieved. As mentioned earlier, the use of recycling variant of CG helps us avoid the cumbersome code changes, and this discussed next.

2.2.2 Implementation

Developing the CG algorithm that is based upon the adapted Lanczos process and the adapted Ritz-Galerkin projection is doable. However, developing its efficient implementation involving standard two/ three term recurrences is non-trivial. Also, as the sequence number of the linear system increase (i.e. j gets larger), the number of orthogonalizations to be done also increase linearly.

As briefly discussed in Section 2.1.1, using a recycling CG (RCG) [42, 43] helps alleviate both these problems. Hence, in this subsection we *first* discuss the idea behind RCG. *Second* we describe how to use RCG so as to easily achieve the earlier described

extra orthogonalities. We do this with no code changes to the existing algorithm. Here, we also discuss the extra computational cost of such an implementation.

Assume that we want to solve the linear system in (2.29). Also assume that the recycle space is in the form of span {U}, where columns of $U \in \mathbb{R}^{n\times k}$ are linearly independent. If x_{-1} is the initial guess for (2.29) and $r_{-1} = b - Ax_{-1}$ is the corresponding residual, then the projected initial guess x_0 is defined as [42, 43]

$$x_0 = x_{-1} + U\left(U^T AU\right)^{-1} U^T r_{-1},$$

with the corresponding residual $r_0 = b - Ax_0$.

At the k^{th} iterative step, the Lanczos process involves [44]

$$\begin{aligned} w_{k+1} \in \mathbb{K}^k(A, U, r_0) \equiv \text{span}\{U, r_0, Ar_0, A^2r_0, \cdots, A^{k-1}r_0\} \\ s.t. \qquad w_{k+1} \perp [U\ w_1\ w_2\ \cdots w_k], \end{aligned}$$

where w_{k+1}, as earlier, is the $(k+1)^{th}$ Lanczos vector and $w_1 = r_0/\|r_0\|$. The Ritz-Galerkin projection here is as follows:

$$r_k \perp \mathbb{K}^k(A, U, r_0).$$

The final solution update and the residual recurrences take the following form:

$$\begin{aligned} x_{k+1} &= x_k + \alpha_k p_k, \\ r_{k+1} &= r_k + \alpha_k A p_k, \end{aligned}$$

where

$$\begin{aligned} p_k &= \beta_{k-1} p_{k-1} + \left(I - U(U^T AU)^{-1}(AU)^T\right) r_k, \\ \alpha_k &= \left(r_k^T r_k\right) / \left(p_k^T A p_k\right), \\ \beta_{k-1} &= \left(r_k^T r_k\right) / \left(r_{k-1}^T r_{k-1}\right). \end{aligned}$$

Next, we discuss how to use the above machinery for our requirements. Consider solving the linear system given by (2.32), originally (2.12). For adapting the Lanczos process in Section 2.2.1.1, while solving this linear system, we need to achieve the extra orthogonality in (2.34). Similarly, for adapting the Ritz-Galerkin projection in Section

2.2.1.2, while solving this linear system, we need to achieve the extra orthogonality in (2.44). Both these orthogonalities can be achieved if we take

$$U = \left[\eta_{(0)}\ \tilde{V}_1\right] \tag{2.46}$$

in RCG.

By defining U as above, $\eta_{(0)}$ and $\tilde{V}_1$ are added in the Krylov search space, which is not needed in the adapted Lanczos process. Also, we are doing extra work here since $\eta_{(0)}$ orthogonality is needed only for Lanczos (not for Ritz-Galerkin), and $\tilde{V}_1$ is needed for Ritz-Galerkin (not for Lanczos).

These facts are true but besides the benefit of ease of implementation, this choice of space often leads to acceleration of the system. We support this with experiments in the next section. A theoretical study of this choice of space is the part of future work.

Also, to satisfy all the other orthogonalities of the previous subsection, equivalent of U (as in (2.46)) can be defined. Since we are usually more concerned about the accuracy of the obtained reduced dynamical systems, we investigate this apects next.

2.3 Accuracy of the Reduced Systems

Using Theorem 15.1 of [35] we know that if the AIRGA algorithm is backward stable, then the relative accuracy of the reduced system obtained by using the inexact AIRGA algorithm, as compared to using the exact AIRGA algorithm, is given as follows:

$$\frac{\|\hat{H}(s) - \tilde{\hat{H}}(s)\|_{H_2}}{\|\hat{H}(s)\|_{H_2}} = \mathcal{O}\left(\kappa\left(H(s)\right) \cdot \|Z\|_2\right), \tag{2.47}$$

where $\kappa\left(H(s)\right)$ is the condition number of $H(s)$ (discussed below), and Z is the perturbation in $H(s)$. As earlier, $\hat{H}(s)$ is the reduced system obtained by using the exact AIRGA algorithm and $\tilde{\hat{H}}(s)$ is the reduced system obtained by using the inexact AIRGA algorithm. We are looking at reduced systems obtained at line 28 of Algorithm 1. That is, after each step of the outer `while` loop (line 2). Thus, accuracy

of the reduced system is dependent on the conditioning of the problem as well as the perturbation. Next, we look at both these quantities separately.

2.3.1 Conditioning Expression

Here, we first introduce condition number concept as well-understood and widely used for matrix operations. Then, we discuss the use of condition number with respect to the operations on transfer functions of dynamical systems. Finally, we expand upon condition number in our context.

Condition number for matrix operations

The relative condition number $\kappa = \kappa(x)$ for a problem given as a function $f : \mathbb{X} \to \mathbb{Y}$, where both $\mathbb{X}$ and $\mathbb{Y}$ are normed vector spaces and $x \in \mathbb{X}$, is defined by [35]

$$\kappa = \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \left(\frac{\|\delta f\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|} \right),$$

or again assuming δx and δf are infinitesimal,

$$\kappa = \sup_{\delta x} \left(\frac{\|\delta f\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|} \right).$$

For example, the condition number of matrix-vector multiplication between $A \in \mathbb{C}^{m \times n}$ and $x \in \mathbb{C}^n$ is given as

$$\kappa \leq \|A\| \|A^{-1}\|.$$

Condition number for transfer function operations

If we perturb the input dynamical system matrices B and $\mathscr{C}$ by δB and $\delta \mathscr{C}$, respectively, then the perturbed dynamical systems are defined as follows [21]:

$$H_{\delta B}(s) = \mathscr{C}^T A(s)^{-1}(B + \delta B) \text{ and } H_{\delta \mathscr{C}}(s) = (\mathscr{C} + \delta \mathscr{C})^T A(s)^{-1} B,$$

where $A(s) = (s^2M + sD + K)$. In this context, the condition numbers of the transfer function response at $s = \sigma$ given by

$$\kappa_B(H(\sigma)) = \frac{\|\mathscr{C}^T A(\sigma)^{-1}\|\|B\|}{\|H(\sigma)\|} \text{ and}$$
$$\kappa_{\mathscr{C}}(H(\sigma)) = \frac{\|\mathscr{C}^T\|\|A(\sigma)^{-1}B\|}{\|H(\sigma)\|}$$

measure the relative sensitivity of the dynamical system with respect to the perturbations in B and $\mathscr{C}$, respectively.

Condition number in our context

We want to compute conditioning of our system with respect to performing the inexact linear solves on lines 5 and 14 of Algorithm 1. Since for backward stability we equate the reduced system obtained by performing the inexact AIRGA algorithm on the unperturbed (original) full system ($H(s)$) and performing the exact AIRGA algorithm on the perturbed full system $\left(\tilde{H}(s)\right)$, these inexact linear solves are captured by $\tilde{H}(s)$. Thus, the conditioning of the input dynamical system with respect to computing the H_2-norm of the error system $H(s) - \tilde{H}(s)$ will give us a *good approximation* to the conditioning of the input dynamical system that we want to access $\Big($with respect to computing the $H_2 -$ norm of $\hat{H}(s) - \hat{\tilde{H}}(s)\Big)$. Similar behaviour has been observed for linear first-order dynamical systems (see Theorem 3.1 and 3.3 in [21]) and bilinear first-order dynamical systems [22].

Recall, the condition number by definition means the relative change in the output $\Big($for us this is $\|H(s) - \tilde{H}(s)\|_{H_2}/\|H(s)\|_{H_2}\Big)$ with respect to the relative change in the input (for us this is $\|Z\|_2/\|K\|_2$ since we are perturbing the K matrix) [22]. Hence, from (2.28) we have

$$\frac{\|H(s) - \tilde{H}(s)\|_{H_2}}{\|H(s)\|_{H_2}} \leq \frac{\|A(s)^{-1}B\|_{H_\infty}\|\mathscr{C}^T A(s)^{-1}\|_{H_2}}{\|H(s)\|_{H_2}} \cdot \frac{\|K\|_2}{1 - \|A(s)^{-1}\|_{H_\infty}} \cdot \frac{\|Z\|_2}{\|K\|_2}, \quad (2.48)$$

where it is assumed that $\|Z\|_2 < 1$ and $\|A(s)^{-1}\|_{H_\infty} < 1$. Hence, the above inequality is equivalent to

$$\frac{\|H(s) - \tilde{H}(s)\|_{H_2}}{\|H(s)\|_{H_2}} \leq \kappa\left(H(s)\right) \cdot \frac{\|Z\|_2}{\|K\|_2}, \quad (2.49)$$

where,

$$\kappa\left(H(s)\right) = \frac{\|A(s)^{-1}B\|_{H_\infty}\|\mathscr{C}^T A(s)^{-1}\|_{H_2}}{\|H(s)\|_{H_2}} \cdot \frac{\|K\|_2}{1 - \|A(s)^{-1}\|_{H_\infty}}. \tag{2.50}$$

In the numerical experiments section (Section 2.4), for the first example taken, we show that this condition number is fairly small, whereas, for the second one it is large. In other words, the first problem is well conditioned and the second problem is ill-conditioned with respect to the H_2-norm of the error system $H(s) - \tilde{H}(s)$[2]. Note that $\|Z\|_2 < 1$ and $\|A(s)^{-1}\|_{H_\infty} < 1$, as assumed here, come from the assumptions for backward stability of the AIRGA algorithm (see Theorem 2.2), and hence, we do not need any extra assumptions.

2.3.2 Computation of Perturbation

Recall (2.20), which has the form

$$Z\mathbf{X} = \eta. \tag{2.51}$$

Here, $Z \in \mathbb{R}^{n\times n}, \mathbf{X} \in \mathbb{R}^{n\times mJ}$, and $\eta \in \mathbb{R}^{n\times mJ}$. Also note that we are solving for Z. As discussed in Introduction, the upper bound for J is $\lceil r_{\max}/m \rceil$, and hence, $mJ \leq r_{\max}$. Using the fact that $r_{\max} \ll n$, we have $mJ < n$. Hence, we have an underdetermined system of equations, which will have more than one solution. For such a system, Singular Value Decomposition (SVD) helps provide one solution [45]. This SVD for $\mathbf{X}$ is given as follows:

$$\mathbf{X} = \mathbb{U}\Sigma\mathbb{V}^T,$$

where $\mathbb{U} \in \mathbb{R}^{n\times n}, \mathbb{V} \in \mathbb{R}^{mJ\times mJ}$ are unitary matrices (i.e. $\mathbb{U}\mathbb{U}^T = I$ and $\mathbb{V}\mathbb{V}^T = I$) and $\Sigma \in \mathbb{R}^{n\times mJ}$ is a diagonal matrix comprising of singular values of $\mathbf{X}$. Let $r_n = rank(\mathbf{X})$, then $\Sigma = \text{diag}\left(\varsigma_1, \ldots, \varsigma_{r_n}, 0, \ldots, 0\right)$. Partitioning $\mathbb{U}$ as $[\mathbb{U}_1\ \mathbb{U}_2]$ and $\mathbb{V}$ as $[\mathbb{V}_1\ \mathbb{V}_2]$, where $\mathbb{U}_1$, $\mathbb{V}_1$ have r_n columns; U_2, V_2 have the remaining columns of U, V, respectively; and

[2]This ill-conditioning of the second problem does not effect our main conjecture. We discuss this aspect in-detail later.

$r_n \leq mJ$, we get

$$\mathbf{X} = \begin{bmatrix} \mathbb{U}_1 & \mathbb{U}_2 \end{bmatrix} \begin{bmatrix} \Sigma_{r_n} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbb{V}_1 & \mathbb{V}_2 \end{bmatrix}^T, \tag{2.52}$$

where $\Sigma_{r_n} = \text{diag}\,(\varsigma_1, \ldots \varsigma_{r_n})$. By using (2.52) and definition of Moore-Penrose Pseudoinverse ([46]; page 423) we have

$$\mathbf{X}^\dagger = \begin{bmatrix} \mathbb{V}_1 & \mathbb{V}_2 \end{bmatrix} \begin{bmatrix} \Sigma_{r_n}^{-1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbb{U}_1^T \\ \mathbb{U}_2^T \end{bmatrix}.$$

Substituting the above expression in (2.51), we have[3]

$$Z = \eta \mathbb{V}_1 \Sigma_{r_n}^{-1} \mathbb{U}_1^T. \tag{2.53}$$

Next, we relate the perturbation Z and the cumulative residual η.

$$\begin{aligned} \|Z\|_2 \leq \|Z\|_f &\leq \|\eta \cdot \mathbb{V}_1 \Sigma_{r_n}^{-1} \mathbb{U}_1^T\|_f \leq \|\eta\|_f \|\mathbb{V}_1 \Sigma_{r_n}^{-1} \mathbb{U}_1^T\|_f, \\ &\leq \left(\| - \eta_{(0)}\|_f + \cdots + \| - \eta_{(J-1)}\|_f\right) \left(\|\mathbb{V}_1 \Sigma_{r_n}^{-1} \mathbb{U}_1^T\|_f\right). \end{aligned} \tag{2.54}$$

In the above equation, $-\eta_{(0)}, \ldots, -\eta_{(J-1)}$ represent the residuals obtained while solving the linear systems arising in the model reduction process. These residuals will reduce if we solve such linear systems more accurately. The second term $\|\mathbb{V}_1 \Sigma_{r_n}^{-1} \mathbb{U}_1^T\|_f$ is usually more dependent on the selection of the expansion points (s_i), and less on the accuracy to which we solve the linear systems [21]. We support this argument with numerical experiments.

To summarize from (2.47) we know, $\|\hat{H}(s) - \tilde{\hat{H}}(s)\|_{H_2}$ is proportional to $\kappa\,(H(s))$ and $\|Z\|_2$. The problem is usually well conditioned and $\|Z\|_2$ is directly proportional to the cumulative residual norm $\|\eta\|_f$ (as in (2.54)). Thus, assuming backward stability conditions hold (as discussed in the previous section), as we iteratively solve the linear systems arising in the AIRGA algorithm more accurately (i.e. reduce the stopping tolerance of the linear solver), we should get a more accurate reduced system. This is very useful in deciding when to stop the linear solver. If we need a very accurate reduced system, then we need to iterate more in the linear solver, else we can stop earlier.

[3]If the system $Z\mathbf{X} = \eta$ is in-consistent, then this is the least squares solution.

2.4 Numerical Experiments

As motivated in Section 2.2, for stability we focus on the CG method for solving the linear systems arising in the AIRGA algorithm. Also, as discussed earlier, CG is optimal for SPD linear systems. Thus, we need to ensure that the coefficient matrices of all the linear systems to be solved are SPD.

The coefficient matrices are of the form $s_i^2 M + s_i D + K$. To achieve that these matrices are SPD at start we do as below.

(a) We take input models that have M, D and K matrices as SPD. We use the one dimensional beam model (size $10,000$) [32] and the Gyroscope model (size $17,361$) [47] that have such matrices and are commonly used (discussed in the next two subsections). These models are of the form [32, 47]

$$\begin{aligned} M\ddot{x}(t) + D\dot{x}(t) + Kx(t) &= Fu(t), \\ y(t) &= C_p x(t), \end{aligned} \tag{2.55}$$

where $M,\ D,\ K \in \mathbb{R}^{n\times n}$ are the mass, the damping and the stiffness matrices, respectively, $F \in \mathbb{R}^{n\times 1}$ and $C_p \in \mathbb{R}^{1\times n}$. These models are Single Input Single Output (SISO), and have proportional damping, i.e. $D = \alpha M + \beta K$, where the damping coefficients α and β belong to $(0, 1)$.

(b) We take the input expansion points (s_i) to be real and positive.

Next, we discuss how to ensure that the linear system matrices are SPD after the first AIRGA iteration (i.e. after start). After the first AIRGA iteration, the expansion points are chosen from the eigenvalues of the quadratic eigenvalue problems of the form $\lambda^2 \hat{M} + \lambda \hat{D} + \hat{K}$. For both our models, these eigenvalues turn out to be complex (case 3.8 of Table 1.1 in [48]). Thus, we get complex expansion points. Execution of the AIRGA algorithm as well as the accuracy of the reduced system does not get affected if one uses real expansion points or complex expansion points. Since real expansion points here are positive too (again because of case 3.8 of Table 1.1 in [48]), using them ensures that our coefficient matrices, $s_i^2 M + s_i D + K$, are SPD at all the AIRGA iterations. Hence, we use real expansion points.

In Algorithm 1, at line 2, the overall iteration (`while-loop`) terminates when the

change in the reduced system (computed as the H_2-error between the reduced systems of two consecutive AIRGA iterations) is less than a certain tolerance. We take this tolerance to be 10^{-04} based on values in [10]. There is one more stopping criteria in this algorithm at line 9. This checks the H_2-error between two temporary reduced systems. We take this tolerance to be 10^{-06} based upon values in [10].

As also motivated in Section 2.2, to ensure that the extra orthogonalities for a stable AIRGA algorithm are satisfied, we use RCG instead of CG. As earlier, we refer to this as the inexact AIRGA algorithm. Preconditioning has to be employed when iterative methods fail or have a very slow convergence. Here, for the first model, we observe that the unpreconditioned RCG method has slow convergence whereas in the second model it fails to converge. Thus, we use a preconditioner. Since Sparse Approximate Inverse (SPAI) [19] and Incomplete Cholesky Factorization (ICHOL) [49, 18] are the most general types of preconditioners, we can use any of these preconditioners with RCG. Here, we use the standard SPAI (with stopping tolerance of 10^{-04}) for the first model and the standard ICHOL (with drop tolerance of 10^{-04}) for the second model. For comparison, we solve all linear systems by a direct method as well. As earlier, we refer to this as the exact AIRGA algorithm. For certain types of analyses, we compare CG and RCG behaviours too.

We implement our codes in MATLAB (2016b), and test on a machine with the following configuration: Intel Xeon(R) CPU E5-1620 V3 @ 3.50 GHz., frequency 1200 MHz., 8 CPU and 64 GB RAM.

2.4.1 One Dimensional Beam Model

As discussed earlier, we do experiments on a system of size $10,000$. Damping coefficients α and β both are taken as 0.05 [32] we take three expansion points as $s_1 = 0.3142, s_2 = 0.6283$ and $s_3 = 0.9425$ (based upon experience). The maximum dimension to which we want to reduce the system (r_{max}) is taken as 3 based upon experience. Thus, in the AIRGA algorithm, we have to solve linear systems of size $10,000 \times 10,000$. While using RCG for solving these linear systems, we use two different stopping tolerances 10^{-02} and 10^{-08}. Ideally, as discussed earlier, we should

obtain a more accurate reduced system for the smaller stopping tolerance.

First, let us look at the assumptions for backward stability of the AIRGA algorithm (see Theorem 2.2). While referring to this theorem, we have already satisfied the conditions (a) and (b) by using CG and RCG, respectively. At all AIRGA iterations, σ_1 picked is s_1, σ_2 picked is s_2, and σ_3 picked is s_3. Thus, at all AIRGA iterations, we solve the linear systems in the following order to match our theory proposed in subsections 2.2.1.1 and 2.2.1.2:

- (2.10) (including (2.31))– Three systems corresponding to three expansion points
- (2.12) (or (2.32))– One system
- (2.10) (or (2.35))– One system *resolve*
- (2.14) (or (2.36))– One system
- (2.10) (or (2.39))– One system *resolve*

Hence, at all AIRGA iterations instead of solving five linear systems, we solve seven linear systems. This is acceptable because this gives us a stable MOR algorithm. Sometimes, use of recycle space accelarates the convergence of all linear systems in-turn offsetting this extra cost. We demonstrate this behaviour in the next example.

Next, we analyze the assumptions (c) and (d) of Theorem 2.2. For all expansion points, $A(s)$ is invertible and $\|A(s)^{-1}\|_{H_\infty}$ is less than one. E.g., for the initial set of expansion points, $\|A(s)^{-1}\|_{H_\infty}$ is 2.68×10^{-02}. Finally, $\|Z\|_2$, at the end of the first AIRGA iteration, for the RCG stopping tolerance of 10^{-02} and 10^{-08} is 0.28 and 0.06, respectively, both of which are also less than one [4].These values are less than one at the end of all the other AIRGA iterations as well. The condition number for our problem, as defined in (2.50), is 8.63×10^{-02}. This shows that the one dimensional beam model is well-conditioned. As earlier, we still use the SPAI preconditioner for better accelaration.

The accuracy results are given in Fig. 2.1. We use the following settings: expansion points $s_i = 2\pi f$, where frequency f vector consists of equally spaced twenty

[4]Our $\mathbf{X}$ while solving (2.20) using (2.53) is full rank i.e. 8 here.

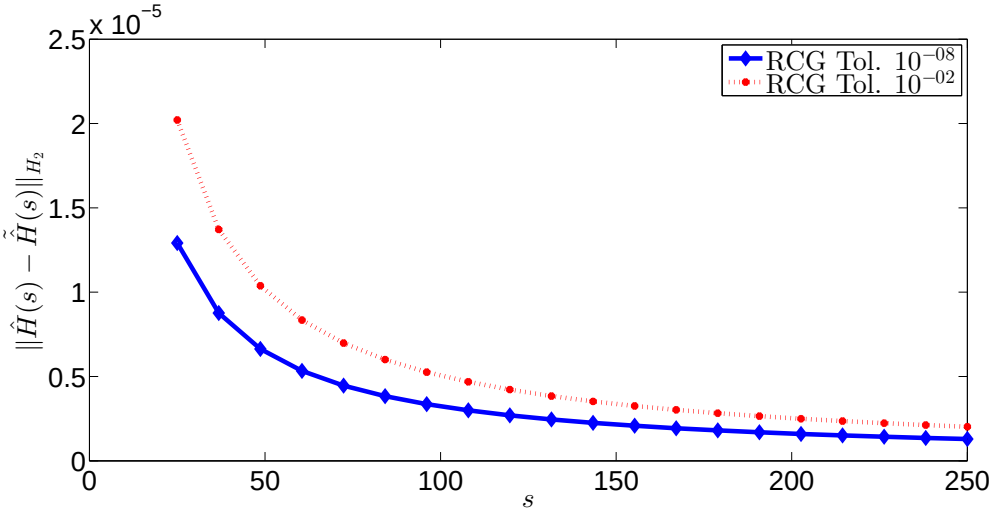

Figure 2.1: Accuracy of the reduced system plotted at each AIRGA iteration for two different stopping tolerances in RCG; one dimensional beam model.

points between 25 and 250. In Fig. 2.1, we have the accuracy of the reduced system $\left(\|\hat{H}(s) - \tilde{\hat{H}}(s)\|_{H_2}\right)$ on the y-axis and the AIRGA iterations on the x-axis. Here, the dotted line corresponds to the RCG stopping tolerance of 10^{-02} while the solid line corresponds to the RCG stopping tolerance of 10^{-08}. From Fig. 2.1, it is evident that we get a more accurate reduced system as we solve the linear systems more accurately (dotted line is above the solid one at all the AIRGA iterations).

In Table 2.1, we give the accuracy results corresponding to each AIRGA iterations. The AIRGA algorithm gets more consistent as it converges to ideal expansion points. Hence, the accuracy of the reduced system for the RCG stopping tolerance of 10^{-08} is visibly better than the accuracy of the reduced system for the RCG stopping tolerance of 10^{-02}.

2.4.2 Gyroscope Model

As mentioned earlier, we do another experiment on a system of size $17,361$. Damping coefficients α and β are taken as 0.2 and 1.34×10^{-04}, respectively [47]. Here, again, we take three expansion points as $s_1 = 6.2832, s_2 = 317.3009$ and $s_3 = 628.3185$. The dimension to which we want to reduce the system (r_{max}) is taken as 12 based upon

Table 2.1: Accuracy of the reduced system at each AIRGA iteration for the two different stopping tolerances in RCG; one dimensional beam model.

AIRGA	$\|\|\hat{H} - \tilde{\hat{H}}\|\|_{H_2}$	
Iteration	RCG stopping tolerance of 10^{-02}	RCG stopping tolerance of 10^{-08}
1	8.34×10^{-05}	2.99×10^{-06}
2	6.98×10^{-05}	2.69×10^{-06}
3	1.37×10^{-05}	2.45×10^{-06}
4	1.03×10^{-05}	2.25×10^{-06}

similar values in [47]. Here, in the AIRGA algorithm we have to solve the linear systems of size $17,361 \times 17,361$. Again, we use RCG for solving these linear systems. To demonstrate our main result, we ideally want the stopping tolerances to be six orders of magnitude different from each other. E.g., 10^{-02} and 10^{-08} in the previous problem. Here, we are unable to solve the linear systems for tolerances less than 10^{-10}. As for the higher tolerance, if we go beyond 10^{-08}, then the AIRGA algorithm's convergence varies (differing iteration counts for convergence). Thus, we cannot compare results of the two cases. Hence, we use stopping tolerances of 10^{-08} and 10^{-10}. As discussed earlier, we should obtain a more accurate reduced system for the smaller stopping tolerance.

Similar to the previous experiment, here also we look at the remaining assumptions for backward stability of the AIRGA algorithm (see Theorem 2.2). While referring to this theorem, we have already satisfied the conditions (a) and (b) by using CG and RCG, respectively. When applying theorem of Section 2.1 here (to satisfy (b)), for simplicity, we do not perform the required resolves. The results below demonstrates that this approximation does not have any effect on our intended behaviour. We still get a more accurate reduced system as we solve the linear systems more accurately. Here, we also show that use of recycle space accelerate the convergence of the linear systems.

Next, again, we analyze the assumptions (c) and (d) of Theorem 2.2. For all expansion points, $A(s)$ is invertible and $\|A(s)^{-1}\|_{H_\infty}$ is less than one. E.g., for the

initial set of expansion points, $\|A(s)^{-1}\|_{H_\infty}$ is 6.46×10^{-01}. Finally, $\|Z\|_2$, at the end of the first AIRGA iteration, for the RCG stopping tolerance of 10^{-08} and 10^{-10} is 8.6×10^{-01} and 3.3×10^{-01}, respectively, both of which are also less than one [5]. These values are less than one at the end of all the other AIRGA iterations as well.

The condition number for this problem, as defined in (2.50), is 5.15×10^{09}. This shows that the Gyroscope model is ill-conditioned. As earlier, we use the basic ICHOL preconditioner, which helps to reduce the amount of ill-conditioning but does not completely eliminated it. If needed, we can use a more advanced preconditioner.

Accuracy of the reduced system is proportional to the condition number $\kappa\left(H(s)\right)$ and the perturbation $\|Z\|$ (see (2.47)). Since, the condition number here still remain high, we get a less accurate reduced system. However, this is still a good problem for us since we want to demonstrate that the reduction in perturbation (linked to linear solver stopping tolerance) improves accuracy. High condition number spoils the accuracy equally for both the RCG stopping tolerances (10^{-08} and 10^{-10}). The accuracy results are given in Table 2.2. It is again evident that we get a more accurate reduced system as we solve the linear systems more accurately.

For this model, we observe that the number of iterations required for convergence of RCG is less than that of CG, both of which are given in Table 2.3. We see a savings of about 10% in the average linear solver iterations. The corresponding computation times are given in Table 2.4. The savings in iterations translate to about 5% savings in time.

Here, we do some other analysis corresponding to (2.54), i.e. relation between the perturbation and the stopping tolerance. From Table 2.5, we demonstrate that $\|\mathbb{V}_1\Sigma_{r_n}^{-1}\mathbb{U}_1^T\|_f$ is less sensitive to the accuracy to which we solve the linear systems (as we reduce the stopping tolerance of RCG from 10^{-08} to 10^{-10}, $\|\mathbb{V}_1\Sigma_{r_n}^{-1}\mathbb{U}_1^T\|_f$ stays almost the same).

[5] Our $\mathbf{X}$ while solving (2.20) using (2.53) is rank deficient (10 instead of 12) but that does not affect our computations.

Table 2.2: Accuracy of the reduced system at each AIRGA iteration for the two different stopping tolerances in RCG; Gyroscope Model.

AIRGA Iteration	$\|\|\hat{H} - \tilde{\hat{H}}\|\|_{H_2}$	
	RCG stopping tolerance 10^{-08}	RCG stopping tolerance 10^{-10}
1	1.55×10^{-03}	8.66×10^{-04}
2	3.63×10^{-05}	3.14×10^{-05}

Table 2.3: Convergence analysis of CG and RCG at two different stopping tolerances; Gyroscope Model.

AIRGA Iteration	Stopping tolerance 10^{-08}		Stopping tolerance 10^{-10}	
	Avg. CG Itr.	Avg. RCG Itr.	Avg. CG Itr.	Avg. RCG Itr.
1	216	207	244	224
2	202	180	228	206
Total	**418**	**387**	**472**	**430**

Table 2.4: Computation time of CG and RCG at two different stopping tolerances; Gyroscope Model.

AIRGA Iteration	Stopping tolerance 10^{-08}		Stopping tolerance 10^{-10}	
	CG time (secs.)	RCG time (secs.)	CG time (secs.)	RCG time (secs.)
1	2.35	2.20	2.49	2.41
2	2.04	1.95	2.33	2.23
Total	**4.39**	**4.15**	**4.82**	**4.64**

Table 2.5: The perturbation expression quantities for RCG at two different stopping tolerances; Gyroscope Model.

AIRGA Iteration	RCG Stopping tolerance 10^{-08}		RCG Stopping tolerance 10^{-10}	
	$\|\eta\|_f$	$\|\mathbb{V}_1\Sigma_{r_n}^{-1}\mathbb{U}_1^T\|_f$	$\|\eta\|_f$	$\|\mathbb{V}_1\Sigma_{r_n}^{-1}\mathbb{U}_1^T\|_f$
1	2.5×10^{-09}	1.34×10^{09}	2.9×10^{-10}	1.33×10^{09}
2	2.6×10^{-09}	1.98×10^{11}	2.6×10^{-10}	1.98×10^{11}

Chapter 3

Stability Analysis of Parametric MOR

Robust Parametric Model Order Reduction [1] is a Ritz-Galerkin projection based algorithm for MOR of parametric first-order and second-order dynamical systems (see Cell 3 and Cell 4 of Table 1.1). Recall (1.1), which can be rewritten for the first-order systems as

$$\begin{aligned} D(p_1, p_2, \ldots, p_w)\dot{x}(t) + K(p_1, p_2, \ldots, p_w)x(t) &= Bu(t), \\ y(t) &= Cx(t). \end{aligned} \tag{3.1}$$

Let $V \in \mathbb{R}^{n\times r}$ be a projection matrix determined by RPMOR. Using $x(t) \approx V\hat{x}(t)$ in (3.1), we obtain the following system:

$$\begin{aligned} D(p_1, p_2, \ldots, p_w)V\dot{\hat{x}}(t) + K(p_1, p_2, \ldots, p_w)V\hat{x}(t) - Bu(t) &= r(t), \\ \hat{y}(t) &= CV\hat{x}(t), \end{aligned}$$

where $r(t)$ is the residual after projection. Applying the Galerkin approach by multiplying V^T in the first equation above we get

$$\begin{aligned} V^T\Big(D(p_1, p_2, \ldots, p_w)V\dot{\hat{x}}(t) + K(p_1, p_2, \ldots, p_w)V\hat{x}(t) - Bu(t)\Big) &= 0, \\ \hat{y}(t) &= CV\hat{x}(t) \end{aligned}$$

or

$$\begin{aligned} \hat{D}(p_1, p_2, \ldots, p_w)\dot{\hat{x}}(t) + \hat{K}(p_1, p_2, \ldots, p_w)\hat{x}(t) - \hat{B}u(t) = 0, \\ \hat{y}(t) = \hat{C}\hat{x}(t), \end{aligned} \tag{3.2}$$

where $\hat{D}(\cdot),\ \hat{K}(\cdot)\ \in \mathbb{R}^{r\times r}, \hat{B}\ \in \mathbb{R}^{r\times m}, \hat{C}\ \in \mathbb{R}^{q\times r}$, $r \ll n$, and $\hat{y}(t)$ is a good approximation to $y(t)$ for a broad range of inputs [21].

The projection matrix V can be determined by many ways. One common way is by moment matching [50, 51, 52], which is discussed next. The parametric system (3.1) can also be stated in the frequency domain (e.g. after Laplace transformation) as [53]

$$\begin{aligned} \left(sD(p_1, p_2, \ldots, p_w) + K(p_1, p_2, \ldots, p_w)\right) x = Bu(s), \\ y = Cx, \end{aligned} \tag{3.3}$$

where s is the frequency and is considered as a new parameter. The above equation can be rewritten as

$$\begin{aligned} \mathbb{A}\left(s, p_1, p_2, \ldots, p_w\right) x = Bu(s), \\ y = Cx, \end{aligned} \tag{3.4}$$

where $\mathbb{A}\left(s, p_1, p_2, \ldots, p_w\right) \in \mathbb{R}^{n\times n}$ is the new parametrized matrix. The above equation is a frequency domain representation of not just first-order dynamical systems but systems of arbitrary differentiation order. As above, for first-order systems $\mathbb{A}(s, p_1, p_2, \ldots, p_w) = sD(p_1, p_2, \ldots, p_w) + K(p_1, p_2, \ldots, p_w)$. For second-order systems $\mathbb{A}(s, p_1, p_2, \ldots, p_w) = s^2M(p_1, p_2, \ldots, p_w) + sD(p_1, p_2, \ldots, p_w) + K(p_1, p_2, \ldots, p_w)$ and so on.

Next, the system in (3.4) is transformed to an affine form as below, which is assumed to exist by the RPMOR algorithm.

$$\begin{aligned} \left(\mathbb{A}_0 + \grave{s}_1\mathbb{A}_1 + \ldots + \grave{s}_w\mathbb{A}_w + \grave{s}_{w+1}\mathbb{A}_{w+1}\right) x = Bu(s), \\ y = Cx, \end{aligned} \tag{3.5}$$

where $\mathbb{A}_0, \mathbb{A}_1, \ldots, \mathbb{A}_{w+1} \in \mathbb{R}^{n\times n}$ and are referred to as dynamical system sub-matrices for rest of this work. The new parameters $\grave{s}_j$ (for $j = 1, \ldots, w+1$) are some functions (polynomial, rational, etc.) of the earlier parameters $(s, p_1, p_2, \ldots, p_w)$.

Next, the state x in (3.5) is computed at the initial set of expansion points $(\grave{s}_1^1, \ldots, \grave{s}_w^1, \grave{s}_{w+1}^1)$ as[1]

$$x = \left[I - (\sigma_1 M_1 + \ldots + \sigma_w M_w + \sigma_{w+1} M_{w+1})\right]^{-1} \left(\mathscr{A}(1)\right)^{-1} Bu(s), \tag{3.6}$$

where $\sigma_j = \grave{s}_j - \grave{s}_j^1, M_j = -\left(\mathscr{A}(1)\right)^{-1} \mathbb{A}_j$ for $j = 1, 2, \ldots, w+1$, and

$$\mathscr{A}(1) = \mathbb{A}_0 + \grave{s}_1^1 \mathbb{A}_1 + \grave{s}_2^1 \mathbb{A}_2 + \ldots + \grave{s}_{w+1}^1 \mathbb{A}_{w+1}. \tag{3.7}$$

A linear system with this matrix is to be efficiently solved and is the focus of our work. Applying Taylor series expansion on (3.6) we get

$$\begin{aligned} x &= \sum_{h=0}^{\infty} \left[\sigma_1 M_1 + \ldots + \sigma_w M_w + \sigma_{w+1} M_{w+1}\right]^h \tilde{B} u(s), \\ &= \sum_{h=0}^{\infty} \mathfrak{x}^{(h)}(\sigma_1, \ldots, \sigma_w, \sigma_{w+1}) u(s), \end{aligned} \tag{3.8}$$

where

$$\begin{aligned} \tilde{B} &= \left(\mathscr{A}(1)\right)^{-1} B, \\ \mathfrak{x}^{(0)}(\sigma_1, \ldots, \sigma_w, \sigma_{w+1}) &= \tilde{B}, \\ \mathfrak{x}^{(1)}(\sigma_1, \ldots, \sigma_w, \sigma_{w+1}) &= \left[\sigma_1 M_1 + \ldots + \sigma_w M_w + \sigma_{w+1} M_{w+1}\right] \mathfrak{x}^{(0)}(\sigma_1, \ldots, \sigma_w, \sigma_{w+1}), \\ &\vdots \\ \mathfrak{x}^{(h)}(\sigma_1, \ldots, \sigma_w, \sigma_{w+1}) &= \left[\sigma_1 M_1 + \ldots + \sigma_w M_w + \sigma_{w+1} M_{w+1}\right] \mathfrak{x}^{(h-1)}(\sigma_1, \ldots, \sigma_w, \sigma_{w+1}). \end{aligned}$$

Here, $\mathfrak{x}^{(h)}(\sigma_1, \ldots, \sigma_w, \sigma_{w+1})$ is called the h^{th}-order system moment at $(\sigma_1, \ldots, \sigma_w, \sigma_{w+1})$. Similarly for the reduced system (3.2), the state variable can be written as

$$\hat{x} = \sum_{h=0}^{\infty} \hat{\mathfrak{x}}^{(h)}(\sigma_1, \ldots, \sigma_w, \sigma_{w+1}) u(s). \tag{3.9}$$

In the reduced system, the h^{th}-order system moment $\hat{\mathfrak{x}}^{(h)}(\sigma_1, \ldots, \sigma_w, \sigma_{w+1})$ is defined similar to $\mathfrak{x}^{(h)}(\sigma_1, \ldots, \sigma_w, \sigma_{w+1})$. The goal of moment matching approach is to find a reduced system such that the first few moments of (3.8) and (3.9) are matched. This provides the projection matrix V. The columns of V are given

[1]From here onwards, we represent a set of parameters as a set of expansion points.

by $\quad span\left\{\mathfrak{x}^{(0)}(\sigma_1,\ldots,\sigma_w,\sigma_{w+1}),\ \mathfrak{x}^{(1)}(\sigma_1,\ldots,\sigma_w,\sigma_{w+1}),\ \ldots,\ \mathfrak{x}^{(h)}(\sigma_1,\ldots,\sigma_w,\sigma_{w+1})\right\}$, where $h \in \mathcal{N}$ (the set of natural numbers).

After obtaining the first few columns of V corresponding to this initial set of expansion points, the above process is repeated with a new set of expansion points $\left(\grave{s}_1^2,\ldots,\grave{s}_w^2,\grave{s}_{w+1}^2\right)$. Thus, corresponding to (3.7), we obtain a new matrix

$$\mathscr{A}(2) = \mathbb{A}_0 + \grave{s}_1^2\mathbb{A}_1 + \grave{s}_2^2\mathbb{A}_2 + \ldots + \grave{s}_{w+1}^2\mathbb{A}_{w+1}, \tag{3.10}$$

and again, a linear system with this matrix is to be efficiently solved. A similar process is performed for the subsequent sets of expansion points $(\grave{s}_1^i,\ldots,\grave{s}_w^i,\ \grave{s}_{w+1}^i)$ for $i = 3,4,\ldots,\mathfrak{z}$, and corresponding to (3.7) and (3.10), we obtain a set of matrices

$$\mathscr{A}(i) = \mathbb{A}_0 + \grave{s}_1^i\mathbb{A}_1 + \grave{s}_2^i\mathbb{A}_2 + \ldots + \grave{s}_{w+1}^i\mathbb{A}_{w+1}. \tag{3.11}$$

Corresponding to each of these matrices, we need to solve a linear system efficiently. When matching first three moments these linear systems have the following form:

Zeroth-Order Moment Matching

$$\mathscr{A}(i)x^{(i)}(1) = B,$$

First-Order Moment Matching

$$\mathscr{A}(i)\left[x^{(i)}(2)\ \ \cdots\ \ x^{(i)}((w+1)+1)\right] = \left[\mathbb{A}_1x^{(i)}(1)\ \ \cdots\ \ \mathbb{A}_{w+1}x^{(i)}(1)\right],$$

Second-Order Moment Matching

$$\begin{aligned}
\mathscr{A}(i)\left[x^{(i)}\left((w+1)+2\right)\ \ \cdots\ \ x^{(i)}\left(2(w+1)+1\right)\right] &= \mathbb{A}_1\left[x^{(i)}(2)\ \ \cdots\ \ x^{(i)}((w+1)+1)\right],\\
&\vdots\\
\mathscr{A}(i)\left[x^{(i)}\left((w+1)(w+1)+2\right)\ \ \cdots\ \ x^{(i)}\left((w+2)(w+1)+1\right)\right] &= \mathbb{A}_{w+1}\left[x^{(i)}\left(2\right)\ \ \cdots\ \ x^{(i)}\left((w+1)+1\right)\right].
\end{aligned} \tag{3.12}$$

where B is given in (3.1); $\mathbb{A}_1,\mathbb{A}_2,\ldots,\mathbb{A}_{w+1}$ are the dynamical system sub-matrices given in (3.5); and $\mathscr{A}(1)$, $\mathscr{A}(2)$, $\ldots$, $\mathscr{A}(i)$ for $i = 1,\ldots,\mathfrak{z}$ are given by (3.7), (3.10) & (3.11). Note that matching higher moments, would result in more number of linear systems to be solved for each set of expansion points (i.e. for every i), where the matrix $\mathscr{A}(i)$ remains the same (since i is the same) and the right-hand sides change (corresponding to the increase in h). We list RPMOR in Algorithm 2.

The algorithm iterates with the first index value of i $\left(i = 1\right.$ or the initial set of expansion points $\left(\grave{s}_1^1,\ \ldots,\ \grave{s}_w^1,\ \grave{s}_{w+1}^1\right)$; see (3.7)$\left.\right)$, and then it restarts for the remaining index values of i $\left(i = 2,\ 3,\ \ldots,\ \mathfrak{z}\right.$ or the subsequent sets of expansion points

Algorithm 2 RPMOR [51]

Input: $\mathscr{A}(i) = \mathbb{A}_0 + \grave{s}_1^i\mathbb{A}_1 + \grave{s}_2^i\mathbb{A}_2 + \ldots + \grave{s}_{w+1}^i\mathbb{A}_{w+1}$; set of expansion points $\left(\grave{s}_1^i, \ldots, \grave{s}_w^i, \grave{s}_{w+1}^i\right)$, for $i = 1, 2, \ldots, \grave{\jmath}$; B, C, h, and ϵ.

Initialize $a_1 = 0,\ a_2 = 0,\ sum = 0$
Compute $x^{(i)}(1) = \mathscr{A}(i)^{-1}B$
Compute the first column in $V:\ v_1 = x^{(i)}(1)/\|x^{(i)}(1)\|$
$sum = 1$
for $\dot{\imath} = 1,\ 2,\ \ldots,\ h$ **do**
  $a_2 = sum$
  for $j = 1,\ 2,\ \ldots,\ w$ **do**
    if $a_1 = a_2$ **then**
      stop
    else
      for $\dot{\jmath} = a_1 + 1,\ \ \ldots,\ a_2$ **do**
        $col = sum + 1$
        $x^{(i)}(col) = \mathscr{A}(i)^{-1}\mathbb{A}_j x^{(i)}(\dot{\jmath})$
        for $k = 1,\ 2, \ldots,\ col - 1$ **do**
          $\hbar = v_k^T x^{(i)}(col)$
          $x^{(i)}(col) = x^{(i)}(col) - \hbar v_k$
        end for
        if $\|x^{(i)}(col)\| > \epsilon$ **then**
          $v_{col} = x^{(i)}(col)/\|x^{(i)}(col)\|_2$
          $sum = col$
        end if
      end for
    end if
  end for
  $a_1 = a_2$
end for
Orthogonalize the columns in V by modified Gram-Schmidt (MGS) w.r.t. ϵ

$\big(\grave{s}^i_1,\ \ldots,\ \grave{s}^i_w,\ \grave{s}^i_{w+1}\big)$; see (3.10) & (3.11)$\big)$. Hence, in the next two sections, we perform stability analysis of the RPMOR with one index value of i.

3.1 Satisfying First Condition of Backward Stability

Consider the first linear system of (3.12)(which is at line 2 of Algorithm 2)

$$\mathscr{A}(i)x^{(1)}(1) = B.$$

We denote the inexactly computed solution as $\tilde{x}^{(1)}(1)$. Also, let the associated residual be η_1. Then, the above equation is equivalent to

$$\mathscr{A}(i)\tilde{x}^{(1)}(1) = B + \eta_1. \tag{3.13}$$

Further, solving the second linear system from (3.12)$\big($which is at line 13 of Algorithm 2$\big)$ inexactly yields

$$\mathscr{A}(i)\tilde{x}^{(1)}(2) = \mathbb{A}_1\tilde{x}^{(1)}(1) + \eta_2. \tag{3.14}$$

Thus, solving the last linear system from (3.12)$\big($which is at line 13 of Algorithm 2$\big)$ inexactly gives

$$\mathscr{A}(i)\tilde{x}^{(1)}\left((w+2)(w+1)+1\right) = \mathbb{A}_{(w+1)}\tilde{x}^{(1)}\left((w+1)+1\right) + \eta_{((w+2)(w+1)+1)}, \tag{3.15}$$

where, as earlier, w denotes the number of parameters.

The projection matrix $\tilde{V}$ is obtained by stacking all the normalized $\tilde{x}^{(1)}(j)$, with $j = 1,\ 2,\ \ldots,\ ((w+2)(w+1)+1)$, as columns vectors. We obtain the reduced system as

$$\begin{aligned}
\tilde{\tilde{D}}(\cdot) &= \tilde{V}^T D(\cdot)\tilde{V},\\
\tilde{\tilde{K}}(\cdot) &= \tilde{V}^T K(\cdot)\tilde{V},\\
\tilde{\tilde{B}}(\cdot) &= \tilde{V}^T B,\\
\tilde{\tilde{C}}(\cdot) &= C\tilde{V}.
\end{aligned}$$

Next, the exact solutions of (3.12) (which is at lines 2 and 13 of Algorithm 2) on a perturbed model are given as follows:

$$(\mathscr{A}(i) + Z)\,\tilde{x}^{(1)}(1) = B, \tag{3.16}$$

$$(\mathscr{A}(i) + Z)\tilde{x}^{(1)}(2) = \mathbb{A}_1\tilde{x}^{(1)}(1), \tag{3.17}$$

$$(\mathscr{A}(i) + Z)\tilde{x}^{(1)}\left((w+2)(w+1)+1\right) = \mathbb{A}_{(w+1)}\tilde{x}^{(1)}\left((w+1)+1\right), \tag{3.18}$$

where Z is the constant perturbation matrix. If we map this perturbation to the original dynamical system matrices, we find that Z most easily combines with $\mathbb{A}_0$ in (3.5) leading to addition of Z with $K(\cdot)$ in (3.3) and (3.1). Thus, we have the following perturbed matrices: $\tilde{D}(\cdot) = D(\cdot), \tilde{K}(\cdot) = K(\cdot) + Z, \tilde{B} = B(\cdot)$ and $\tilde{C}(\cdot) = C(\cdot)$.

The projection matrix $\tilde{V}$ is obtained in a manner similar to above. The reduced system is now given by

$$\begin{aligned}
\hat{\tilde{D}}(\cdot) = \tilde{V}^T\tilde{D}(\cdot)\tilde{V} = \tilde{V}^T D(\cdot)\tilde{V} = \hat{\tilde{D}}(\cdot),\\
\hat{\tilde{K}}(\cdot) = \tilde{V}^T\tilde{K}(\cdot)\tilde{V} = \tilde{V}^T\left(K(\cdot) + Z\right)\tilde{V} = \tilde{\hat{K}}(\cdot) + \tilde{V}^T Z\tilde{V},\\
\hat{\tilde{B}}(\cdot) = \tilde{V}^T B,\\
\hat{\tilde{C}}(\cdot) = C\tilde{V}.
\end{aligned} \tag{3.19}$$

From line 12 of Algorithm 2, we know $((w+2)(w+1)+1)$ can be denoted as *col*. Further, comparing (3.13)–(3.15) with (3.16)–(3.18) we get

$$Z\ \mathbf{X} = \eta, \tag{3.20}$$

where $\mathbf{X} = \left[\tilde{x}^{(1)}(1),\ \tilde{x}^{(1)}(2), \ldots,\ \tilde{x}^{(1)}(col)\right]$ and $\eta = [-\eta_1,\ -\eta_2, \ldots,\ -\eta_{col}]$.

In the above equation, we replace $\mathbf{X}$ in-terms of $\tilde{V}$ (recall $\tilde{V} = \mathbf{X}\mathscr{D}_X$). That is,

$$Z\tilde{V}\mathscr{D}_X^{-1} = \mathsf{\eta} \quad or \quad Z\tilde{V} = \mathsf{\eta}\mathscr{D}_X, \tag{3.21}$$

where $\mathscr{D}_X = \begin{bmatrix} \frac{1}{\left\|\tilde{x}^{(1)}(1)\right\|_f} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{\left\|\tilde{x}^{(1)}(2)\right\|_f} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{\left\|\tilde{x}^{(1)}(col)\right\|_f} \end{bmatrix}$.

Multiplying $\tilde{V}^T$ from the left side of (3.21), we get

$$\tilde{V}^T Z \tilde{V} = \tilde{V}^T \eta \mathcal{D}_X. \tag{3.22}$$

Theorem 3.1 *Let the inexact linear solves in RPMOR algorithm ((3.13)–(3.15)) be solved while satisfying*

$$\tilde{V}^T \eta = - \begin{bmatrix} \tilde{V}_1^T \\ \tilde{V}_2^T \\ \vdots \\ \tilde{V}_{col}^T \end{bmatrix} \begin{bmatrix} \eta_1 & \eta_2 & \dots & \eta_{col} \end{bmatrix} = 0. \tag{3.23}$$

Then, the reduced system obtained by using inexact solves in the original system is same as the reduced system obtained by using exact solves in a perturbed system or the reduced system matrices on the both sides of the equality in (3.19) *are the same. Hence, RPMOR satisfies the first condition of backward stability with respect to these inexact linear solves, i.e. (2.8).*

From the above theorem, we infer that the underlying iterative solver should *firstly* be based upon a Ritz-Galerkin framework to achieve $\tilde{V}_1 \perp \eta_1$, $\tilde{V}_2 \perp \eta_2$, $\dots$, and $\tilde{V}_{col} \perp \eta_{col}$ [36]. Since Conjugate Gradient (i.e. CG) is one such algorithm [35], we propose its use in RPMOR. *Secondly*, this particular solver should also satisfy the remaining orthogonalities of (3.23). These orthogonalities can be easily satisfied by using a recycling variant of the underlying iterative solver, and hence, we propose the use of Recycling Conjugate Gradient (i.e. RCG [42]). This aspect has been already discussed in-detail in Chapter 2 for the non-parametric case, and the similar technique easily carries over to this parametric case.

3.2 Satisfying Second Condition of Backward Stability

For satisfying the second condition of stability in RPMOR, we use Theorem 4.3 from [21]. This translates to the result below.

Lemma 3.1 *If* $\|Z\|_2 < \frac{1}{\left\|\mathscr{A}(i)^{-1}\right\|_{H_\infty}}$ *then*

$$\left\|H\left(\grave{s}_1^i,\ldots,\grave{s}_{w+1}^i\right)-\widetilde{H}\left(\grave{s}_1^i,\ldots,\grave{s}_{w+1}^i\right)\right\|_{H_2} \leq \frac{\left\|\mathscr{A}\left(i\right)^{-1}B\right\|_{H_\infty}\left\|C\mathscr{A}\left(i\right)^{-1}\right\|_{H_2}}{1-\left\|\mathscr{A}\left(i\right)^{-1}\right\|_{H_\infty}\|Z\|_2}\|Z\|_2\,, \tag{3.24}$$

where $\mathscr{A}(i)=\mathbb{A}_0+\grave{s}_1^i\mathbb{A}_1+\ldots+\grave{s}_{w+1}^i\mathbb{A}_{w+1}$,
$H\left(\cdot\right)=C\mathscr{A}(i)^{-1}B$,
and $\tilde{H}\left(\cdot\right)=C\left(\mathscr{A}(i)+Z\right)^{-1}B$.

Proof: see in Appendix A.

Theorem 3.2 *If* $\|Z\|_2<1$ *and* $\|\mathscr{A}(i)^{-1}\|_{H_\infty}<1$, *then*

$$\|H(\cdot)-\tilde{H}(\cdot)\|_{H_2}=\mathcal{O}(\|Z\|_2). \tag{3.25}$$

Hence, RPMOR satisfies the second condition of backward stability with respect to the inexact linear solves, i.e. (2.9).

Next, we support our stability analysis theory with experiments.

3.3 Numerical Experiments

We perform preliminary experiment on the FOM model [54]. This model consists of a parametric linear dynamical system of size $n=1006$, as

$$\begin{aligned}\left(sD-A\left(\mathfrak{p}\right)\right)x&=Bu\left(s\right),\\ y&=Cx,\end{aligned} \tag{3.26}$$

where $D = I_{n\times n}$, $A(p) = diag\big(\mathbb{A}_1(p),\ \mathbb{A}_2,\ \mathbb{A}_3, \mathbb{A}_4\big)$, and
$B^T = C = \Big[\underbrace{10\ \ldots\ 10}_{6}\ \ \underbrace{1\ \ldots\ 1}_{1000}\Big]$ with

$$\mathbb{A}_1(p) = \begin{bmatrix} 1 & p \\ p & 1 \end{bmatrix}, \quad \mathbb{A}_2 = \begin{bmatrix} 1 & 200 \\ 200 & 1 \end{bmatrix}, \quad \mathbb{A}_3 = \begin{bmatrix} 1 & 400 \\ 400 & 1 \end{bmatrix}, \quad \text{and}$$

$$\mathbb{A}_4 = diag\,(1,\ \ldots, 1000)\,.$$

We use RPMOR settings as below. In the above equation, there is frequency variable s and parameters p (or p_j with $j = 1, \ldots, w$, and $w = 2$; $p_j \in [99.99,\ 100]$), all of which together form a set of expansion points.

The number of moments matched are three (i.e. up to $h = 2$), and we use four sets of expansion points ($i = 1, 2, \ldots, \mathfrak{z}$ with $\mathfrak{z} = 4$) as follows based upon values in [51]: $[0.1\pi, p_1]$, $[0.2\pi, p_2]$, $[0.3\pi, p_1]$, and $[0.45\pi, p_2]$. Since we have four sets of expansion points, RPMOR takes four iterative steps. For the settings above, the size of the reduced system r comes out to be 8.

This leads to solving linear systems of size 1006×1006. As earlier, here also, for solving the linear systems while computing V by a direct method (exact RPMOR), we use a backslash in Matlab. As discussed in Section 3.1, for stability we use RCG with two stopping tolerance 10^{-06} and 10^{-10}.

Preconditioning has to be employed when iterative methods fail or have a very slow convergence. Here, for the given model, we observe that the unpreconditioned RCG method has fails to converge. Thus, we use the standard Sparse Approximate Inverse (SPAI) [19] preconditioner (with stopping tolerance of 10^{-04}).

We implement our codes in MATLAB (2016b), and test on a machine with the following configuration: Intel Xeon(R) CPU E5-1620 V3 @ 3.50 GHz., frequency 1200 MHz., 8 CPU and 64 GB RAM.

Ideally, as discussed earlier, we should obtain a more accurate reduced system for smaller stopping tolerance. We use the following settings in (3.26): expansion points $s_i = 2\pi\sqrt{-1}f$, where frequency f vector consists of equally spaced twenty points between 25 and 250; parameters p_j consist of equally spaced twenty points between 100 and 200. The accuracy result is given in Figure 3.1. Here, we have accuracy of the

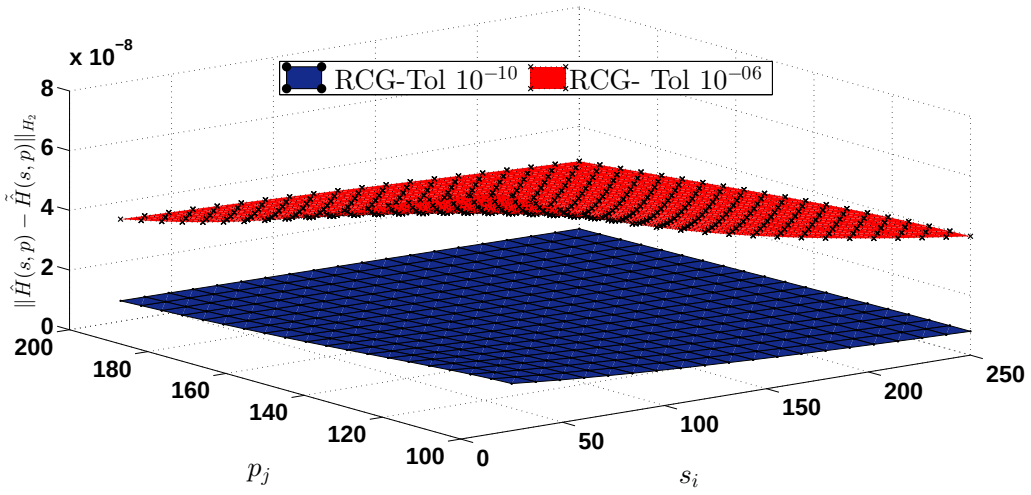

Figure 3.1: Accuracy of the reduced system plotted with respect to expansion points and parameters for the two different stopping tolerances in RCG; FOM model.

reduced system $\left(\left\|\hat{H}\left(s,\ \boldsymbol{p}\right) - \widetilde{\hat{H}}\left(s,\ \boldsymbol{p}\right)\right\|_{H_2}\right)$ on the z-axis, expansion points (i.e. s_i) on x-axis, and parameters (i.e. p_j) on the y-axis. From Figure 3.1, it is evident that we get a more accurate reduced system as we solve the linear systems more accurately (the blue surface is below to the red surface).

Chapter 4

Reusing Preconditioners for Non-parametric MOR

As mentioned in the Introduction, the focus of this Chapter (i.e. 4) and the next Chapter (i.e. 5) is on reusing preconditioners in the sequence of linear systems arising in different MOR algorithms (from Table 1.1). As earlier, here, we focus on the non-parametric case (Cells 1 and 2 of Table 1.1) and in the next Chapter we focus on the parametric case (Cells 3 and 4 of Table 1.1).

As also discussed before, application of preconditioner reuse to IRKA [4](the most popular MOR algorithm for non-parametric first-order dynamical systems; belonging to Cell 1) has been already extensively studied [7, 30]. Hence, in this Chapter we focus on AIRGA (the common MOR algorithm for non-parametric second-order dynamical systems). Although the focus here is on one algorithm, our proposed preconditioner reuse theory is developed to work for all algorithms (non-parametric and parametric; first-order and second-order). Next, we revisit theory of preconditioning, and specifically the Sparse Approximate Inverse (SPAI) preconditioners, which we use.

4.1 General Preconditioning and SPAI

Preconditioning is of two kinds (implicit and explicit), and we focus on the latter [55]. In case of implicit preconditioners, application of preconditioning requires

solving linear systems. For example, in factorization based preconditioning $A \approx LU$, where L and U are sparse triangular matrices approximating exact L and U factors. Here, application of the preconditioner requires only forward and backward solves. This is usually referred to as incomplete LU factorization (ILU) based preconditioner. Variations of ILU that exploit certain matrix constructs can also be developed. For example, ILU based upon Schur's complement [56]. Further, ILQ, SSOR and ADI are other kinds of preconditioning that fall under the implicit category [55].

Although implicit preconditioners have been used extensively for a very long time, they have their own drawbacks. For example, ILU based preconditioners do not be scale well when the system size becomes very large (computation time becomes prohibitively expensive). This is because, forward and backward solves in such preconditioners are inherently sequential and cannot be easily parallelized. Besides this, the breakdown in the factorization process because of the zero pivoting carries over from the full factorization case to this incomplete factorization case.

Explicit preconditioning is one where directly the inverse of the coefficient matrix is approximated or $P \approx A^{-1}$. Hence, applying the preconditioner just involves performing matrix-vector products [20]. Sparse approximate inverse (SPAI) are the most commonly used explicit preconditioners, which we use and are discussed in-detail later in this section.

Variations of approximate inverse preconditioners also exist. One example, as we have seen in the case of implicit preconditioning, is the Schur's complement based approximate inverse preconditioner [20]. Another example is where the approximate inverse preconditioner is constructed by using a high-order convergent scheme that relies on matrix-matrix multiplications [57, 58].

Hybrid of implicit and explicit preconditioning is also common. Here, combinations of factorizations and approximate inverses are used to compute a preconditioner. An example of this is given in [59], where for a SPD matrix, Cholesky factorization is first performed. This in-turn is used to obtain a more efficient approximate inverse preconditioner. Another example is where the approximate inverse of the coefficient matrix is used to compute an approximation to matrix's Schur's complement. This is

then used to build an ILU preconditioner [56].

Now, we give the details of SPAI. For constructing a preconditioner P corresponding to a coefficient matrix A, we focus on methods for finding approximate inverse of A by minimizing the Frobenius norm of the residual matrix $I - AP$. This minimization problem can be rewritten as [19]

$$\min_P \|I - AP\|_f^2. \tag{4.1}$$

Here, the columns of residual matrix $I - AP$ can be computed independently, which is an important property that can be exploited. Hence, the solution of (4.1) can be separated into n independent least square problems as

$$\begin{aligned} &\min_P \sum_{i=1}^{n} \|(I - AP)e^i\|_2^2, \ \textit{or} \\ &\min_{p^i} \|e^i - Ap^i\|_2^2, \quad \text{for } i = 1,\ 2,\ \ldots,\ n, \end{aligned} \tag{4.2}$$

where e^i and p^i are the i-th column of I and P, respectively. The above minimization problem can be implemented in parallel and one can efficiently obtain the explicit approximate inverse P of A.

Usually A is sparse. In this case, we can solve a more efficient version of the optimization problem given in (4.2). Here, first, a good sparsity pattern of P is assumed (usually the Identity matrix). As the solutions of the least squares problems are iteratively computed, this sparsity pattern is updated. One common updating strategy adaptively exploits the number of non-zeros arising in the resulting residuals ($r^i = e^i - Ap^i$), which requires solving 1D minimization problems [20]. A more sophisticated updating strategy uses a multivariate minimization [60]. Second, now since both A and P are sparse, we solve much smaller least squares problems, and all matrix-vector products are done in a *sparse-mode* (operations involving a sparse-matrix and a sparse-vector).

As earlier, in most of the listed MOR algorithms (Table 1.1), the change from one linear system to the next is usually very small, and hence, the applied preconditioner could be reused. Next, we propose our theory of reusing preconditioners in the MOR context.

4.2 General Theory of Reusing of Preconditioners

In general, the linear systems of equations generated in MOR algorithms have the following form:

$$\begin{aligned} A_1 X_1 &= B_1, \\ A_2 X_2 &= B_2, \\ &\vdots \\ A_\ell X_\ell &= B_\ell, \end{aligned} \tag{4.3}$$

where $A_i \in \mathbb{R}^{n \times n}$, $X_i \in \mathbb{R}^{n \times m}$, and $B_i \in \mathbb{R}^{n \times m}$; for $i = 1,\ 2,\ \ldots,\ \ell$.

Let P_1 be a good preconditioner for A_1 that is computed by the theory discussed in the above section ((4.1)–(4.2)) or

$$\min_{P_1} \|I - A_1 P_1\|_f^2.$$

Now, we need to find a good preconditioner P_2 corresponding to A_2. Using the standard SPAI theory, this means solving

$$\min_{P_2} \|I - A_2 P_2\|_f^2. \tag{4.4}$$

If we are able to enforce $A_1 P_1 = A_2 P_2$, then P_2 will be an equally good preconditioner for A_2 as much as P_1 is a good preconditioner for A_1 (since the Spectrum of $A_2 P_2$ would be same as that of $A_1 P_1$, on which convergence of any Krylov subspace method depends). Since P_2 is unknown here, we have a degree of freedom in choosing how to form it. Without loss of generality, we assume that $P_2 = Q_2 P_1$, where Q_2 is an unknown matrix. Here, we need to enforce $A_1 P_1 = A_2 Q_2 P_1$. Thus, instead of solving the minimization problem (4.4), we can solve

$$\min_{Q_2} \|A_1 - A_2 Q_2\|_f^2. \tag{4.5}$$

Note that P_2 here is never explicitly formed by multiplying two matrices Q_2 and P_1. Rather, always a matrix-vector product is done to apply the preconditioner.

Let ζ be the vector during the Krylov subspace method iteration that has to be multiplied with the preconditioned matrix $A_2 P_2$. Then, we do as follows:

$$\tilde{\zeta} = A_2 \left(Q_2 \left(P_1 \zeta \right) \right). \tag{4.6}$$

Table 4.1: Reusing preconditioner approaches.

First approach	**Second approach**
• $A_1P_1 = A_2P_2$ • If $P_2 = Q_2P_1$, then $A_1P_1 = A_2Q_2P_1$ • $\min_{Q_2} \|A_1 - A_2Q_2\|_f^2$	Same as the *first* approach
• $A_1P_1 = A_3P_3$ • If $P_3 = Q_3P_1$, then $A_1P_1 = A_3Q_3P_1$ • $\min_{Q_3} \|A_1 - A_3Q_3\|_f^2$	• $A_2P_2 = A_3P_3$ • If $P_3 = Q_3P_2$, then $A_2P_2 = A_3Q_3P_2$ • $\min_{Q_3} \|A_2 - A_3Q_3\|_f^2$
$\vdots$	$\vdots$
• $A_1P_1 = A_iP_i$ • If $P_i = Q_iP_1$, then $A_1P_1 = A_iQ_iP_1$ • $\min_{Q_i} \|A_1 - A_iQ_i\|_f^2$	• $A_{i-1}P_{i-1} = A_iP_i$ • If $P_i = Q_iP_{i-1}$, then $A_{i-1}P_{i-1} = A_iQ_iP_{i-1}$ • $\min_{Q_i} \|A_{i-1} - A_iQ_i\|_f^2$

If A_1 is close to A_2 in some matrix norm as compared to closeness between I and A_2, then (4.5) would be an easier minimization problem to solve than (4.4).

Next, we apply a similar argument for finding a good preconditioner P_3 corresponding to A_3. We can obtain P_3 by enforcing either $A_1P_1 = A_3P_3$ or $A_2P_2 = A_3P_3$. For these two cases, P_3 would be as effective preconditioner for A_3 as P_1 is for A_1 or P_2 is for A_2, respectively. Here too, we have a degree of freedom in computing P_3. If we have $P_3 = Q_3P_1$ or $P_3 = Q_3P_2$ for these two cases, respectively[1], then we need to solve either of the following minimization problems:

$$\min_{Q_3} \|A_1 - A_3Q_3\|_f^2 \qquad \textit{or} \qquad \min_{Q_3} \|A_2 - A_3Q_3\|_f^2. \tag{4.7}$$

Similarly, for all the remaining linear system matrices in the sequence, $A_4, A_5, \ldots, A_i$, we have two choices as (4.7) in obtaining the corresponding good preconditioners, $P_4, P_5, \ldots, P_i$. All this is summarized in Table 4.1.

We propose with evidence the following two results:

[1]As mentioned after (4.5), we do not explicitly form the preconditioner by multiplying two matrices. Rather, the preconditioner is applied via a matrix-vector product.

(a) In the non-parametric case, the second approach is more suited. This is because in this case the minimization problem of the second approach is much easier to solve as compared to the first approach (attributed to rapidly changing expansion/ interpolation points, and in-turn, rapidly changing matrices). The computation of P_i from P_{i-1} in this case (rather than P_1 as above) does have the drawback of accumulated approximation errors, however, solving the minimization problem efficiently is a bigger bottleneck for scaling to large problems.

(b) In the parametric case, the first approach is more beneficial. This is because, in this case although the two approaches have a similarly hard minimization problem (attributed to slowly varying parameters, and in-turn, slowly changing matrices), the computation of P_i from P_1 in the first approach leads to a preconditioner with less approximation errors, and hence, a one which is more accurate.

Since the focus here is on the non-parametric case (Cell 2 of Table 1.1), we use the second approach, and this is further discussed in the next Section. Details of parametric case (Cell 3 and Cell 4 of Table 1.1) are given in Chapter 5 using the first approach.

4.3 Application of Reusing Preconditioner

If we closely observe Algorithm 1 (AIRGA), which belongs to Cell 2 of Table 1.1, linear systems are solved at lines 5 and 14. To solve these system, we can chose any solver from a large pool of available Krylov subspace methods. For example, GMRES [61], BI-CGSTAB [62], IDR(s)[63], etc. Since GMRES is the most popular one among these, we use it inside AIRGA in our result section.

If we relook at linear systems at lines 5 and 14 in Algorithm 1, we realize that they have more characteristics. These linear systems can be very easily transformed into general shifted linear systems of the form $\varsigma\mathscr{D}+\mathscr{K}$ (see Section 3 of [64]). Therefore, this property can be exploited in solving these sets of linear systems simultaneously [65, 66], which is part of our future work[2].

[2]If the linear system coefficient matrices have special properties, then more efficiency can be

Delving further into the complexity of such linear systems, we observe that the matrices change with the index of outer `while` loop (line 2) as well as with the index of the `for` loop corresponding to the expansion points (line 3). Hence, we denote such matrices not only with a subscript as in previous subsection but also with a superscript. That is, $A_i^{(z)} = \left(s_i^{(z)}\right)^2 M + s_i^{(z)} D + K,$ where $z = 1, \ldots, \mathfrak{z}$ (until covergence) and $i = 1, \ldots, \ell$. As the matrix $A_i^{(z)}$ changes with respect to two different indices, we can reuse preconditioners in many ways. However, here we use the second approach as discussed in the previous subsection. This approach adapted for AIRGA is diagrammatically represented in Figure 4.1.

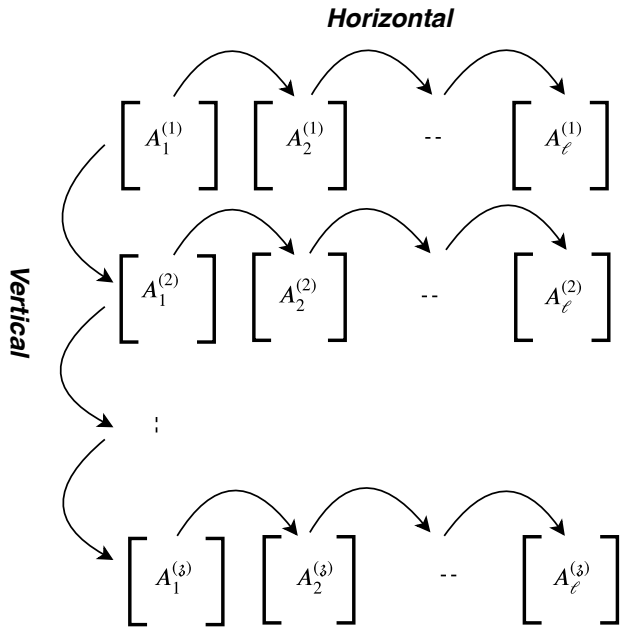

Figure 4.1: Reusing preconditioners in AIRGA.

Computation of preconditioners is done only at line 5 because at line 14, matrices do not change, only the right-hand sides do. Hence, we only focus on reusing preconditioners for line 5.

Next, we show how the new preconditioners are computed for both, the horizontal direction and the vertical direction. While looking at the horizontal route, let,

incorporated. For example, if the coefficient matrices $(\varsigma\mathfrak{D} + \mathfrak{K})$ have $\mathfrak{D}$, $\mathfrak{K}$ as real and ς as complex, then we can reduce the number of linear systems that are required to be solved. For more details, see Section 1 of [67].

$$A_{i-1}^{(z)}P_{i-1}^{(z)} = \overbrace{A_{i-1}^{(z)}\left(I+\left(\left(s_i^{(z)}\right)^2-\left(s_{i-1}^{(z)}\right)^2\right)\left(A_{i-1}^{(z)}\right)^{-1}M+\left(s_i^{(z)}-s_{i-1}^{(z)}\right)\left(A_{i-1}^{(z)}\right)^{-1}D\right)}^{A_i^{(z)}}P_i^{(z)},$$

$$= A_{i-1}^{(z)}\left(I+\left(\left(s_i^{(z)}\right)^2-\left(s_{i-1}^{(z)}\right)^2\right)\left(A_{i-1}^{(z)}\right)^{-1}M+\left(s_i^{(z)}-s_{i-1}^{(z)}\right)\left(A_{i-1}^{(z)}\right)^{-1}D\right)$$

$$\cdot \underbrace{\underbrace{\left(I+\left(\left(s_i^{(z)}\right)^2-\left(s_{i-1}^{(z)}\right)^2\right)\left(A_{i-1}^{(z)}\right)^{-1}M+\left(s_i^{(z)}-s_{i-1}^{(z)}\right)\left(A_{i-1}^{(z)}\right)^{-1}D\right)^{-1}}_{Q_i^{(z)}}P_{i-1}^{(z)}}_{P_i^{(z)}}.$$

Figure 4.2: Expressing one linear system matrix in terms of the other.

$A_{i-1}^{(z)} = \left(s_{i-1}^{(z)}\right)^2 M + s_{i-1}^{(z)}D + K$ and
$A_i^{(z)} = \left(s_i^{(z)}\right)^2 M + s_i^{(z)}D + K$ be the two coefficient matrices for different expansion points $s_{i-1}^{(z)}$ and $s_i^{(z)}$, respectively, with $i = 2, \ldots, \ell$. Using the above theory, we enforce $A_{i-1}^{(z)}P_{i-1}^{(z)} = A_i^{(z)}P_i^{(z)}$ in Figure 4.2. Thus, we eventually enforce $A_{i-1}^{(z)}P_{i-1}^{(z)} = A_i^{(z)}Q_i^{(z)}P_{i-1}^{(z)}$ and solve the minimization problem

$$\min_{Q_i^{(z)}} \|A_{i-1}^{(z)} - A_i^{(z)}Q_i^{(z)}\|_f^2.$$

This gives us the new preconditioner $P_i^{(z)} = Q_i^{(z)}P_{i-1}^{(z)}$. This minimization is again performed for n independent least square problems as in (4.2). Similar steps are followed for reusing preconditioners along the rest of the horizontal directions, i.e. for all $z = 1, \ldots, \mathfrak{z}$.

Now, applying this technique for the vertical direction, we have for $z = 2, \ldots, \mathfrak{z}$

$$A_1^{(z-1)}P_1^{(z-1)} = A_1^{(z)}P_1^{(z)}.$$

Following the steps as for the horizontal direction, here, we solve the minimization problem

$$\min_{Q_1^{(z)}} \|A_1^{(z-1)} - A_1^{(z)}Q_1^{(z)}\|_f^2.$$

This gives us the new preconditioner $P_1^{(z)} = Q_1^{(z)}P_1^{(z-1)}$. Again, this is solved as n independent least square problems as in (4.2).

AIRGA with an efficient implementation of the above discussed theory of reusing preconditioners is given in Algorithm 3. If we closely look at line 5 of Algorithm 1, the solution vector is denoted by $X^{(0)}(s_i)$, where the superscript "0" refers to the index of the inner `while` loop (line 9). We do not bother about this index because, as earlier, matrix does not change inside this inner loop. Rather, we need to capture the change because of the outer `while` loop indexed with z. Hence, we denote the solution vector as $\mathfrak{X}^{(z)}(s_i)$ in Algorithm 3 (lines 8, 11, 19 & 22). It is important to emphasize again that preconditioners are never computed explicitly. Rather, they are obtained using matrix-vector products (see line numbers 11, 19 & 22 of Algorithm 3).

Since shift-invariant preconditioners have been proposed for the general shifted linear systems [65, 68], our this reuse SPAI technique can be coupled with these preconditioners for further efficiency. We plan to look at this aspect as part of our future work. Next, we support our theory with multiple numerical experiments.

4.4 Numerical Experiments

For supporting our proposed preconditioned iterative solver theory using AIRGA [10], we perform experiments on two models. The first is a macroscopic equations of motion model (i.e. academic disk brake M_0) [69], and is discussed in Section 4.4.1. The second is also a similar model, however, this is a real-life industrial problem (i.e. industrial disk brake M_1) [69]. The experiments on this model are discussed in Section 4.4.2. These models are described by the following set of equations [69]:

$$\begin{aligned} M_\Omega \ddot{x}(t) &= -D_\Omega \dot{x}(t) - K_\Omega x(t) + Fu(t), \\ y(t) &= C^T x(t), \end{aligned} \tag{4.8}$$

where $M_\Omega = M,\ K_\Omega = K_E + K_R + \Omega^2 K_G,\ D_\Omega = \alpha M_\Omega + \beta K_\Omega$ (case of proportionally damped system; as needed for AIRGA) with commonly used parameter values as $\Omega = 2\pi,\ \alpha = 5 \times 10^{-02}$, and $\beta = 5 \times 10^{-06}$. Further, $F \in \mathbb{R}^n$ and $C^T \in \mathbb{R}^n$ are taken as $[1\ 0\ \cdots\ 0]^T$, which is the most frequently used choice. We take four expansion points linearly spaced between 1 and 500 based upon experience.

Algorithm 3 : AIRGA with reuse of SPAI preconditioner

$z = 1$

while the H_2 error between two consecutive reduced systems is greater than the ϵ_1 **do**

  if $z == 1$ **then**

    for $i = 1, \ldots, \ell$ **do**

      $A_i^{(1)} = \left(\left(s_i^{(1)}\right)^2 M + \left(s_i^{(1)}\right) D + K\right)$

      if $i == 1$ **then**

        Compute initial $P_1^{(1)}$ by solving $\min\limits_{P_1^{(1)}} \|I - A_1^{(1)} P_1^{(1)}\|_f^2$

        Solve $A_1^{(1)} P_1^{(1)} \mathfrak{X}^{(1)}(s_1) = F$

      else

        Compute $Q_i^{(1)}$ by solving $\min\limits_{Q_i^{(1)}} \|A_{i-1}^{(1)} - A_i^{(1)} Q_i^{(1)}\|_f^2$

        Solve $A_i^{(1)} [Q_i^{(1)} \ \cdots \ Q_2^{(1)} P_1^{(1)}] \mathfrak{X}^{(1)}(s_i) = F$

      end if

    end for

  else

    for $i = 1, \ldots, \ell$ **do**

      $A_i^{(z)} = \left(\left(s_i^{(z)}\right)^2 M + \left(s_i^{(z)}\right) D + K\right)$

      if $i == 1$ **then**

        Compute $Q_1^{(z)}$ by solving $\min\limits_{Q_1^{(z)}} \|A_1^{(z-1)} - A_1^{(z)} Q_1^{(z)}\|_f^2$

        Solve $A_1^{(z)} \left[Q_1^{(z)} \ \ldots \ Q_1^{(2)} \ P_1^{(1)}\right] \mathfrak{X}^{(z)}(s_1) = F$

      else

        Compute $Q_i^{(z)}$ by solving $\min\limits_{Q_i^{(z)}} \|A_{i-1}^{(z)} - A_i^{(z)} Q_i^{(z)}\|_f^2$

        Solve $A_i^{(z)} \left[\underbrace{Q_i^{(z)} \ \cdots \ Q_2^{(z)}} \ \underbrace{Q_1^{(z)} \ \ldots \ Q_1^{(2)}} \ P_1^{(1)}\right] \mathfrak{X}^{(z)}(s_i) = F$

      end if

    end for

  end if

  "All the given set of expansion points (i.e. $s_1, \ s_2, \ \ldots, \ s_\ell$) are updated"

  $z = z + 1$

end while

Although our purpose is to just reuse SPAI in AIRGA (Algorithm 3), we also execute original SPAI in AIRGA (Algorithm 1) for comparison. In Algorithms 1 and 3, at line 2 the overall iteration (`while-loop`) terminates when the change in the reduced model (computed as H_2-error between the reduced models at two consecutive AIRGA iterations) is less than a certain tolerance. We take this tolerance as 10^{-04} based upon the values in [10]. There is one more stopping criteria in Algorithms 1 at line 9 (also in Algorithm 3 but not listed here). This checks the H_2-error between two temporary reduced models. We take this tolerance as 10^{-06}, again based upon the values in [10]. Since this is an adaptive algorithm, the optimal size of the reduced model is determined by the algorithm itself, and is denoted by r.

The linear systems that arise here have non-symmetric matrices. There are many iterative methods available for solving such linear systems. We use the Generalized Minimal Residual (GMRES) method [61] because it is very popular. The stopping tolerance in GMRES is taken as 10^{-06}, which is a common standard. As mentioned in Introduction, for both the given models, we observe that unpreconditioned GMRES fails to converge. Hence, we use the SPAI preconditioner as described above (without and with reuse).

As mentioned earlier, without loss of generality, we perform right preconditioning. To demonstrate the effectiveness of our theory for all types of preconditioning, for the academic disk model, we give data corresponding to left preconditioning as well.

We use Modified Sparse Approximate Inverse (MSPAI 1.0) proposed in [20] as our preconditioner. This is because MSPAI uses a linear algebra library for solving sparse least square problems that arise here. We use standard initial settings of MSPAI $\big($i.e. tolerance (ep) of 10^{-04} and cache size (cs) of 80$\big)$.

We perform our numerical experiments on a machine with the following configuration: Intel Xeon (R) CPU E5-1620 V3 @ 3.50 GHz., frequency 1200 MHz., 8 CPU and 64 GB RAM. All the codes are written in MATLAB (2016b) (including AIRGA, GMRES) except SPAI and reusable SPAI. MATLAB is used because of ease of rapid prototyping. Computing SPAI and reusable SPAI in MATLAB is expensive, therefore, we use C++ version of these (SPAI is from MSPAI and reusable SPAI is written by

us). MSPAI further uses BLAS, LAPACK and ATLAS libraries.

It is important to emphasize that we do not integrate our MATLAB code base with the C++ based preconditioner. This is because integrating the two is complicated and is not needed here as well.

We compute SPAI and reusable SPAI in-parallel, separately, and save them on the hard-disk in the standard .mtx files [20]. When we run our MATLAB code base, then these files are read from the hard-disk into the main memory and converted into .mat files for further processing.

4.4.1 Academic Disk Brake Model

This model is of size $4,669$. Based upon experience, the maximum reduced system size (r_{max}) is taken as 20. As mentioned earlier, however, due to the adaptive nature of AIRGA, we obtain a reduced system of size $r = 13$. For this model, AIRGA takes two outer iterations (line 2 of Algorithms 1 and 3) to converge (i.e. $\mathfrak{z} = 2$).

Reusing the SPAI preconditioner is beneficial when the values of $\|I - A_i^{(z)}\|_f / \|I\|_f$ is large, and the values of $\|A_{i-1}^{(z)} - A_i^{(z)}\|_f / \|A_{i-1}^{(z)}\|_f$ and $\|A_1^{(z-1)} - A_1^{(z)}\|_f / \|A_1^{(z-1)}\|_f$ are small, which is true in this case (see Table 4.2). In this table, columns 1 and 2 list the AIRGA iterations and the four expansion points, respectively. The above three quantities are listed in columns 3, 4 and 5, respectively. For the first AIRGA iteration and the first expansion point, SPAI preconditioner cannot be reused because there is no earlier preconditioner (mentioned as *NA* in table). From the second expansion point (and the first AIRGA iteration), we perform horizontal reuse of preconditioner (see Figure 4.1). This is the same for the second AIRGA iteration as well. Vertical reuse of preconditioner is done only for the first expansion point (and the second AIRGA iteration; again see Figure 4.1).

In Table 4.3, we compare the SPAI and the reusable SPAI timings. As for Table 4.2, here columns 1 and 2 list the AIRGA iterations and the four expansion points, respectively. SPAI and reusable SPAI computation times are given in columns 3 and 4, respectively. At the first AIRGA iteration and the first expansion point, both SPAI and reusable SPAI take the same computation time. This is because, as above, reusing

Table 4.2: SPAI and reusable SPAI analysis for the academic disk brake model.

AIRGA Itr.†	Exp. Pts.‡	SPAI Case	Reusable SPAI Case	
		Standard $\frac{\|I - A_i^{(z)}\|_f}{\|I\|_f}$	Horizontal $\frac{\|A_{i-1}^{(z)} - A_i^{(z)}\|_f}{\|A_{i-1}^{(z)}\|_f}$	Vertical $\frac{\|A_1^{(z-1)} - A_1^{(z)}\|_f}{\|A_1^{(z-1)}\|_f}$
1	1	3.77×10^{06}	*NA*	*NA*
	2	4.36×10^{06}	0.1569	*NA*
	3	4.95×10^{06}	0.3139	
	4	5.54×10^{06}	0.4708	
2	1	7.63×10^{06}	*NA*	0.9996
	2	4.06×10^{06}	0.0180	*NA*
	3	1.62×10^{06}	20.3431	
	4	3.82×10^{06}	0.4985	

† AIRGA Iterations.

‡ Expansion Points.

of SPAI preconditioner is not applicable here. From the second expansion point of the first AIRGA iteration, we see substantial savings because of the reuse of the SPAI preconditioner (approximately 68%).

Before presenting GMRES data, we would like to discuss improvements in the condition numbers of the coefficient matrices because of the preconditioning. This data is given in Table 4.4. As evident, preconditioning does substantially improve the quality of the coefficient matrices.

Table 4.5 provides the iteration count and the computation time of GMRES. Here, we only provide GMRES execution details since the computation time of preconditioner has been discussed above. In this table, column 1 lists the AIRGA iterations. The number of linear solves and average GMRES iterations per linear solve are given in columns 2 and 3, respectively. Finally, columns 4 and 5 list the computation times of GMRES when using SPAI and reusable SPAI, respectively. We notice from this table that solving linear systems by GMRES with SPAI takes less computation time as compared to solving them by GMRES with reusable SPAI. This is because when we reuse the SPAI preconditioner in GMRES, additional matrix-vector products are

Table 4.3: SPAI and reusable SPAI computation time for the academic disk brake model.

AIRGA Iterations (z)	Expansion Points (s_i)	SPAI (Seconds)	Reusable SPAI (Seconds)
1	1	174	174
	2	164	10
	3	165	16
	4	165	20
2	1	165	64
	2	165	10
	3	165	108
	4	158	20
Total	**8**	**1321**	**422**

Table 4.4: Condition numbers of the coefficient matrices before and after application of SPAI[¶] for the academic disk brake model.

AIRGA Iterations (z)	Expansion Points (s_i)	Before Preconditioner	After Preconditioner
1	1	6.4×10^{06}	3.3×10^{03}
	2	4.6×10^{06}	2.9×10^{03}
	3	2.7×10^{06}	2.2×10^{03}
	4	1.8×10^{06}	1.6×10^{03}
2	1	2.0×10^{06}	1.3×10^{03}
	2	3.6×10^{06}	2.6×10^{03}
	3	5.8×10^{06}	5.1×10^{03}
	4	7.0×10^{06}	3.2×10^{03}

[¶] SPAI and Reusable SPAI improve the condition number almost equally.

Table 4.5: GMRES computation time for the academic disk brake model.

AIRGA Iterations (z)	No. of Linear Solves	GMRES Iterations per Linear Solve	GMRES Time when Using SPAI (Seconds)	GMRES Time when Using Reusable SPAI (Seconds)
1	10	271	7.98	8.62
2	13	270	8.12	8.93
Total	**23**	$10 \times 271 + 13 \times 270$ $= \mathbf{6220}$	$10 \times 7.98 + 13 \times 8.12$ $= \mathbf{185}$	$10 \times 8.62 + 13 \times 8.93$ $= \mathbf{202}$

Table 4.6: GMRES with SPAI and reusable SPAI computation time for the academic disk brake model.

AIRGA Iterations (z)	GMRES Plus SPAI Time (Seconds)	GMRES Plus Reusable SPAI Time (Seconds)
1	748	306
2	759	318
Total	**1507**	**624**

performed, however, this extra cost is almost negligible when compared to the savings in the preconditioner computation time for the latter case (as evident in Table 4.2 above; also see total GMRES and preconditioner time below).

As earlier, the data in Table 4.5 is corresponding to right preconditioning. In the case of left preconditioning we see only a modest change in the metrics under-consideration. That is, the total GMRES iterations, the total GMRES plus SPAI time, and the total GMRES plus reusable SPAI time are 6364, 190, and 204, respectively.

Table 4.6 gives the computation time of GMRES plus SPAI (column 2) and GMRES plus reusable SPAI (column 3) at each AIRGA iteration (column 1). As evident from this table, reusing the SPAI preconditioner leads to about 60% savings in total time required for solving all the linear systems.

4.4.2 Industrial Disk Brake Model

This model is of size 1.2 million. Based upon experience, the maximum reduced system size (r_{max}) is taken as 100. As mentioned earlier, however, due to the adaptive nature of AIRGA, we obtain a reduced system of size $r = 52$. For this model, AIRGA takes four outer iterations (line 2 of Algorithms 1 and 3) to converge (i.e. $\mathfrak{z} = 4$).

Again, reusing the SPAI preconditioner is beneficial when the value of $\|I - A_i^{(z)}\|_f / \|I\|_f$ is large, and the value of $\|A_{i-1}^{(z)} - A_i^{(z)}\|_f / \|A_{i-1}^{(z)}\|_f$ and $\|A_1^{(z-1)} - A_1^{(z)}\|_f / \|A_1^{(z-1)}\|_f$ are small, which is true in this case (see Table 4.7). The structure of this table is same as Table 4.2. As earlier, for the first AIRGA iteration and the first expansion point, SPAI preconditioner cannot be reused because there is no earlier preconditioner (mentioned as *NA* in table). From the second expansion point (and the first AIRGA iteration), we perform horizontal reuse of preconditioner (see Figure 4.1). This is the same for the second, the third and the fourth AIRGA iterations as well. Vertical reuse of preconditioner is done only for the first expansion point (and the second, the third, and the fourth AIRGA iterations; again see Figure 4.1).

In Table 4.8, we compare the SPAI and the reusable SPAI timings. The structure of this table is same as that of Table 4.3. As before, at the first AIRGA iteration and the first expansion point, both SPAI and reusable SPAI take the same computation time. This is because, as above, reusing of SPAI preconditioner is not applicable here. From the second expansion point of the first AIRGA iteration, we see substantial savings because of the reuse of the SPAI preconditioner (from 160 hours to 26 hrs 30 minutes; approximately 83%).

As in the case of the academic disk model, here too before presenting GMRES data, we would like to discuss improvements in the condition numbers of the coefficient matrix because of the preconditioning. This data is given in Table 4.9. As evident, preconditioning does substantially improve the quality of the coefficient matrices.

Table 4.10 provides the iteration count and the computation time of GMRES. Here, again we have only provided GMRES execution details since the computation time of the preconditioner has already been discussed above. The structure of this

Table 4.7: SPAI and reusable SPAI analysis for the industrial disk brake model.

AIRGA Iterations (z)	Expansion Points (s_i)	SPAI Case	Reusable SPAI Case	
		Standard	Horizontal	Vertical
		$\frac{\|I - A_i^{(z)}\|_f}{\|I\|_f}$	$\frac{\|A_{i-1}^{(z)} - A_i^{(z)}\|_f}{\|A_{i-1}^{(z)}\|_f}$	$\frac{\|A_1^{(z-1)} - A_1^{(z)}\|_f}{\|A_1^{(z-1)}\|_f}$
1	1	6.54×10^{08}	*NA*	*NA*
	2	6.54×10^{08}	3.74×10^{-05}	*NA*
	3	6.54×10^{08}	7.49×10^{-05}	
	4	6.54×10^{08}	1.12×10^{-04}	
2	1	1.31×10^{09}	*NA*	1.006
	2	6.65×10^{08}	0.49	*NA*
	3	6.53×10^{08}	0.50	
	4	6.56×10^{08}	0.49	
3	1	1.30×10^{09}	*NA*	1.009
	2	7.01×10^{08}	0.4658	*NA*
	3	6.53×10^{08}	0.5499	
	4	6.63×10^{08}	0.4940	
4	1	1.31×10^{09}	*NA*	1.0015
	2	6.86×10^{08}	0.4641	*NA*
	3	6.53×10^{08}	0.5002	
	4	6.56×10^{08}	0.4933	

Table 4.8: SPAI and reusable SPAI computation time for the industrial disk brake model.

AIRGA Iterations (z)	Expansion Points (s_i)	SPAI[§]	Reusing SPAI[§]
1	1	10 hrs	10 hrs
	2	10 hrs	1 hr
	3	10 hrs	1 hr
	4	10 hrs	1 hr
2	1	10 hrs	1 hr 30 mins
	2	10 hrs	1 hr
	3	10 hrs	1 hr
	4	10 hrs	1 hr
3	1	10 hrs	1 hr 30 mins
	2	10 hrs	1 hour
	3	10 hrs	1 hour
	4	10 hrs	1 hour
4	1	10 hrs	1 hr 30 mins
	2	10 hrs	1 hr
	3	10 hrs	1 hr
	4	10 hrs	1 hr
Total	**16**	**160 hrs**	**26 hrs 30 mins**

[§] All times given here differ in seconds (not evident because of the rounding to the nearest minute).

Table 4.9: Condition numbers of the coefficient matrices before and after application of SPAI for the industrial disk brake model.

AIRGA Iterations (z)	Expansion Points (s_i)	Before Preconditioner	After Preconditioner
1	1	1.4×10^{16}	1.4×10^{09}
	2	1.5×10^{16}	1.6×10^{09}
	3	1.6×10^{16}	2.5×10^{09}
	4	1.8×10^{16}	2.8×10^{09}
2	1	9.6×10^{16}	8.1×10^{09}
	2	3.4×10^{16}	2.4×10^{09}
	3	5.8×10^{16}	9.3×10^{09}
	4	2.2×10^{16}	3.4×10^{09}
3	1	7.5×10^{16}	1.9×10^{09}
	2	2.5×10^{16}	2.5×10^{09}
	3	5.0×10^{16}	6.1×10^{09}
	4	6.9×10^{16}	4.7×10^{09}
4	1	8.9×10^{16}	3.5×10^{09}
	2	9.5×10^{16}	8.3×10^{09}
	3	5.4×10^{16}	5.8×10^{09}
	4	1.3×10^{16}	5.4×10^{09}

Table 4.10: GMRES computation time for the industrial disk brake model.

AIRGA Iterations (z)	No. of Linear Solves	GMRES Iterations per Linear Solve	GMRES Time when Using SPAI (Minutes)	GMRES Time when Using Reusable SPAI (Minutes)
1	64	421	08	10
2	64	426	09	11
3	64	429	10	12
4	52	432	10	12
Total	**244**	$64 \times (421 + 426 + 429) + 52 \times 432 = \mathbf{104,128}$	$64 \times (08 + 09 + 10) + 52 \times 10 = \mathbf{2248}$	$64 \times (10 + 11 + 12) + 52 \times 12 = \mathbf{2736}$

table is same as that of Table 4.5. As earlier, we notice from this table that solving linear systems by GMRES with SPAI takes less computation time as compared to solving them by GMRES with reusable SPAI. This is again because of additional matrix-vector products in the reusable SPAI case. Here also, this extra cost is almost negligible when compared to the savings in the preconditioner computation time (as evident in Table 4.8; also see the total GMRES and preconditioner time below).

Table 4.11 gives the computation time of GMRES plus SPAI (column 2) and GMRES plus reusable SPAI (column 3) at each AIRGA iteration (column 1). As before, it is evident from this table, reusing the SPAI preconditioner leads to about 64% savings in total time (from 197 hours 28 minutes to 72 hours 06 minutes).

To demonstrate the quality of the reduced system, we plot the relative H_2 error between the transfer function of the original system and the reduced system with respect to the different expansion points (in Figure 4.3). The reduced system considered here is obtained by using GMRES with reusable SPAI. These expansion points, denoted by S, are computed as $2\pi f$, where the frequency variable f is linearly spaced between 1 and 500. As evident from this figure, the obtained reduced system is good (the error is very small). Further, we also observe from this figure that the reduced model is most accurate in 7–10 range of the expansion points. This is because the final expansion

Table 4.11: GMRES with SPAI and reusable SPAI computation time for the industrial disk brake model.

AIRGA Iterations (z)	GMRES plus SPAI Time	GMRES plus Reusable SPAI Time
1	48 hrs 32 mins	23 hrs 40 mins
2	49 hrs 36 mins	16 hrs 14 mins
3	50 hrs 40 mins	17 hrs 18 mins
4	48 hrs 40 mins	14 hrs 54 mins
Total	**197 hrs 28 mins**	**72 hrs 06 mins**

points, upon the convergence of AIRGA, lie in this range.

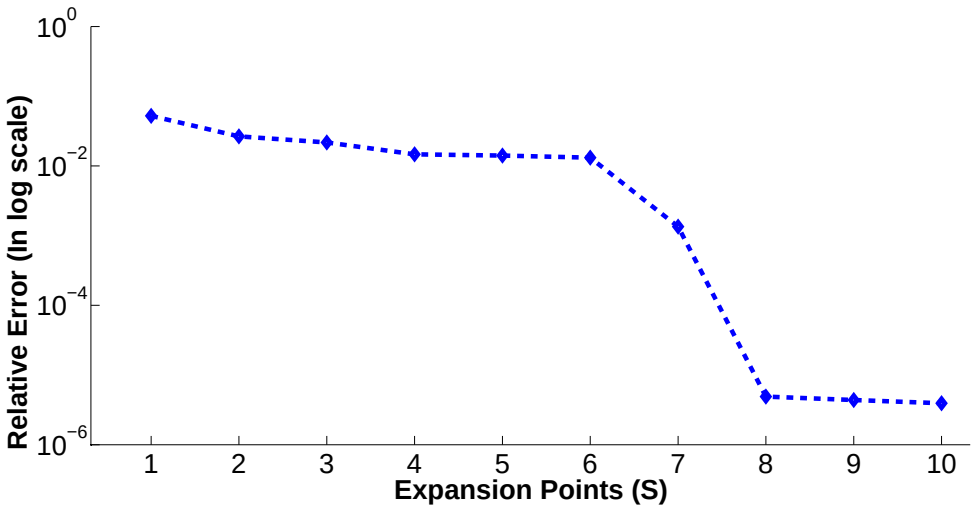

Figure 4.3: Relative error between the original and reduced system for the industrial disk brake model.

Chapter 5

Reuse of Preconditioners for Parametric MOR

Here, our preconditioner reuse focus is on MOR algorithms belonging to Cell 3 and 4 of Table 1.1. That is, those used for reducing parametric first-order and second-order dynamical systems, respectively. As done for stability analysis of parametric MOR algorithms in Chapter 3, here we pick RPMOR that works for both first-order and second-order systems. It is important to point out that our preconditioner reuse theory below does not bind to any order, with experiments done for both the parametric first-order and second-order dynamical systems.

5.1 Linear systems Arising in RPMOR

Recall from (3.12) that in RPMOR linear systems, when matching first three moments, have the following form:

Zeroth-Order Moment Matching ($h = 0$)

$$\mathscr{A}(i)x^{(i)}(1) = B,$$

First-Order Moment Matching ($h = 1$)

$$\mathscr{A}(i)\left[x^{(i)}(2) \quad \cdots \quad x^{(i)}((w+1)+1)\right] = \left[\mathbb{A}_1 x^{(i)}(1) \quad \cdots \quad \mathbb{A}_{w+1}x^{(i)}(1)\right],$$

Second-Order Moment Matching ($h = 2$)

$$\begin{aligned}
\mathscr{A}(i)\left[x^{(i)}\left((w+1)+2\right) \quad \cdots \quad x^{(i)}\left(2(w+1)+1\right)\right] &= \mathbb{A}_1\left[x^{(i)}(2) \quad \cdots \quad x^{(i)}((w+1)+1)\right], \\
&\vdots \\
\mathscr{A}(i)\left[x^{(i)}\left((w+1)(w+1)+2\right) \quad \cdots \quad x^{(i)}\left((w+2)(w+1)+1\right)\right] &= \mathbb{A}_{w+1}\left[x^{(i)}\left(2\right) \quad \cdots \quad x^{(i)}\left((w+1)+1\right)\right].
\end{aligned} \tag{5.1}$$

Note that matching higher moments, would result in more number of linear systems to be solved for each set of expansion points (i.e., for every i), where the matrix $\mathscr{A}(i)$ remains the same (since i is the same) and the right-hand sides change (corresponding to the increase in h). The preconditioned iterative solver theory derived below is more dependent on the changing matrices and less on the changing right-hand sides. Also in practice, usually up to second-order moments are matched [70, 51]. Hence, for ease of explanation, we do not discuss matching higher order moments.

In (5.1), for a particular i after the first equation (see line 2 of Algorithm 2), the right-hand side vectors of all subsequent sets of equations are available together (groups of equations corresponding to $h = 1$ and $h = 2$; see lines 5 and 13 of Algorithm 2). Hence, one can easily solve these linear systems simultaneously. For this, we can use a block version of the relevant iterative method [71, 72, 73].

This concept was introduced for the first time with Conjugate Gradient (CG) method [71]. A similar study with GMRES was proposed in [72]. Here, we give a brief overview of block iterative methods. There are three classes of block Krylov subspace methods; 1) classical block methods, 2) global block methods and, 3) loop-interchange block methods [74]. We use a classical block method because of its ease in usage. Let a linear system of equations with multiple right-hand sides be given as

$$\mathscr{A}\mathscr{X} = \mathscr{B},$$

where $\mathscr{A} \in \mathbb{R}^{n\times n}$, $\mathscr{B} \in \mathbb{R}^{n\times m}$, $m \ll n$. Given $\mathscr{X}_0$ and R_0 (i.e. $R_0 = \mathscr{B} - \mathscr{A}\mathscr{X}_0$) as the

initial solution and the initial residual, respectively, these methods build the block Krylov subspace $\mathbb{K}^j(\mathscr{A}, R_0) = \text{span}\{R_0, \mathscr{A}R_0, \mathscr{A}^2R_0, \ldots, \mathscr{A}^{j-1}R_0\}$, and find solution in it [71, 72]. Next, we apply our preconditioner reuse theory from Chapter 4 to RPMOR.

5.2 Theory of Reusing Preconditioners Applied to RPMOR

In general, the sequence of linear systems generated by the first equation of (5.1) (see line 2 of Algorithm 2) has the following form[1]:

$$\begin{aligned} \mathscr{A}(1)\mathscr{X}(1) &= \mathscr{B}(1), \\ \mathscr{A}(2)\mathscr{X}(2) &= \mathscr{B}(2), \\ &\vdots \\ \mathscr{A}(i)\mathscr{X}(i) &= \mathscr{B}(i), \end{aligned} \tag{5.2}$$

where $\mathscr{A}(1), \ldots, \mathscr{A}(i) \in \mathbb{R}^{n\times n}$ originally come form (3.7), (3.10) and (3.11); $\mathscr{B}(1), \ldots, \mathscr{B}(i) \in \mathbb{R}^{n\times m}$ all of which in this case are B from (1.1) and (3.1); and $\mathscr{X}(1) = x^{(1)}(1)$, $\mathscr{X}(2) = x^{(2)}(1)$, and $\mathscr{X}(i) = x^{(i)}(1)$ here.

This sequence of linear systems is exactly same as (4.3) except a slightly different notation, which we have avoided making uniform. This is because the structure of the underlying matrices in the non-parametric case (Chapter 4) and the parametric case (Chapter 5) are different. As mentioned in Chapter 4, first approach of Table 4.1 is more suitable here in the parametric case, which we rationalize elaborately below.

Here, $\mathscr{A}(1)$ is almost as close to $\mathscr{A}(i)$ as much as $\mathscr{A}(i-1)$ is close to $\mathscr{A}(i)$. For example, when applying RPMOR on commonly used models, we observe that $\|\mathscr{A}(1) - \mathscr{A}(3)\|$ is *almost of the same order of magnitude* as $\|\mathscr{A}(2) - \mathscr{A}(3)\|$ (in a relative sense).

[1]We look at the first equation because the matrix change happens here only, which is the case that requires computation of a new preconditioner. In the other equations of (5.1), the matrix remains the same and only the right-hand side changes. Thus, the preconditioner corresponding to the first equation is ideal.

[2]. The reason for this is, the relative change in the sets of parameters $\grave{s}_1^i, \grave{s}_2^i, \ldots, \grave{s}_{w+1}^i$ as the model reduction algorithm progresses ($i = 1, \ldots, \ell$) is often *many orders of magnitude smaller* when compared with the magnitude of elements in the unchanging dynamical system sub-matrices ($\mathbb{A}_0, \mathbb{A}_1, \ldots, \mathbb{A}_{w+1}$). Thus, the minimization problem of the first approach is almost of the same hardness as the minimization problem of the second approach.

The first approach has an added benefit that P_i here is more accurate since it is formed from P_1 ($P_i = Q_i P_1$), which is very accurate. On the contrary, in the second approach $P_i = Q_i P_{i-1},\ P_{i-1} = Q_{i-1} P_{i-2},\ \ldots,\ P_2 = Q_2 P_1$. Thus, approximation errors from P_2 to P_3 to P_{i-1} make P_i in the second approach less accurate as compared to the P_i in the first approach. Thus, for our implementation we use the first approach.

To summarize, when using basic SPAI for all $i = 2, 3, \ldots, \ell$, we need to solve $\min\limits_{P_i} \|I - \mathscr{A}(i)P_i\|_f^2$, which we first transform to $\min\limits_{P_i} \|\mathscr{A}(1)P_1 - \mathscr{A}(i)P_i\|_f^2$, and subsequently to $\min\limits_{Q_i} \|\mathscr{A}(1) - \mathscr{A}(i)Q_i\|_f^2$. This last formulation, as mentioned earlier, is usually much easier to solve. Next, we discuss the application of the first approach, as discussed above, to RPMOR.

From (5.1), recall (3.7) and (3.11), we know $\mathscr{A}(1) = \mathbb{A}_0 + \grave{s}_1^1\mathbb{A}_1 + \grave{s}_2^1\mathbb{A}_2 + \ldots + \grave{s}_{w+1}^1\mathbb{A}_{w+1}$ and $\mathscr{A}(i) = \mathbb{A}_0 + \grave{s}_1^i\mathbb{A}_1 + \grave{s}_2^i\mathbb{A}_2 + \ldots + \grave{s}_{w+1}^i\mathbb{A}_{w+1}$ are two coefficient matrices for different expansion points $\left(\grave{s}_1^1, \ldots, \grave{s}_{w+1}^1\right)$ and $\left(\grave{s}_1^i, \ldots, \grave{s}_{w+1}^i\right)$, respectively. Using the above theory, we express $\mathscr{A}(i)$ in terms of $\mathscr{A}(1)$ as follows:

$$\mathscr{A}(i) = \mathscr{A}(1)\Big(I + \left(\grave{s}_1^i - \grave{s}_1^1\right)\left(\mathscr{A}(1)\right)^{-1}\mathbb{A}_1 + \cdots + \left(\grave{s}_{w+1}^i - \grave{s}_{w+1}^1\right)\left(\mathscr{A}(1)\right)^{-1}\mathbb{A}_{w+1}\Big),$$

Now we enforce $\mathscr{A}(1)P_1 = \mathscr{A}(i)P_i$ or

$$\begin{aligned}\mathscr{A}(1)P_1 = \mathscr{A}(1)&\Big(I + \left(\grave{s}_1^i - \grave{s}_1^1\right)\left(\mathscr{A}(1)\right)^{-1}\mathbb{A}_1 + \cdots + \left(\grave{s}_{w+1}^i - \grave{s}_{w+1}^1\right)\left(\mathscr{A}(1)\right)^{-1}\mathbb{A}_{w+1}\Big)\cdot\\ &\Big(I + \left(\grave{s}_1^i - \grave{s}_1^1\right)\left(\mathscr{A}(1)\right)^{-1}\mathbb{A}_1 + \cdots + \left(\grave{s}_{w+1}^i - \grave{s}_{w+1}^1\right)\left(\mathscr{A}(1)\right)^{-1}\mathbb{A}_{w+1}\Big)^{-1} P_1,\\ &= \mathscr{A}(i)P_i,\end{aligned}$$

[2]Average values of $\|\mathscr{A}(1) - \mathscr{A}(3)\|/\|\mathscr{A}(1)\|$ for RPMOR applied to Micro-Gyroscope Model and Electro-Chemistry Model are 2.11×10^{-09} and 1.77×10^{-05}, respectively. Also, average values of $\|\mathscr{A}(2) - \mathscr{A}(3)\|/\|\mathscr{A}(2)\|$ for the two respective models are 1.07×10^{-09} and 7.66×10^{-06}, respectively.

where

$$P_i = \left(I + \left(\grave{s}_1^i - \grave{s}_1^1 \right) \left(\mathscr{A}(1) \right)^{-1} \mathbb{A}_1 + \cdots + \left(\grave{s}_{w+1}^i - \grave{s}_{w+1}^1 \right) \left(\mathscr{A}(1) \right)^{-1} \mathbb{A}_{w+1} \right)^{-1} P_1.$$

Let

$$Q_i = \left(I + \left(\grave{s}_1^i - \grave{s}_1^1 \right) \left(\mathscr{A}(1) \right)^{-1} \mathbb{A}_1 + \cdots + \left(\grave{s}_{w+1}^i - \grave{s}_{w+1}^1 \right) \left(\mathscr{A}(1) \right)^{-1} \mathbb{A}_{w+1} \right)^{-1},$$

then the above implies $\mathscr{A}(1)P_1 = \mathscr{A}(i)Q_iP_1$. Further, instead of solving this last equation leading to $P_i = Q_iP_1$, we solve a simpler problem

$$\min_{Q_i} ||\mathscr{A}(1) - \mathscr{A}(i)Q_i||_f^2 = \min_{(q_i)^{(\imath)}} \sum_{\imath=1}^{n} \left|\left| (a_1)^{(\imath)} - \mathscr{A}(i) \, (q_i)^{(\imath)} \right|\right|_2^2,$$

where $(a_1)^{(\imath)}$ and $(q_i)^{(\imath)}$ denote the $\imath^{th}$ columns of $\mathscr{A}(1)$ and Q_i, respectively.

Next, we demonstrate the effectiveness of our technique using multiple numerical examples.

5.3 Numerical Results

We demonstrate our proposed preconditioned iterative solver theory using RPMOR [51] on two models. The first is a Electro-Chemistry Model from [51], and is discussed in Section 5.3.1. This represents a parametric first-order dynamical system, the one for which we have discussed RPMOR. The second is a Micro-Gyroscope Model from [70], and is discussed in Section 5.3.2. This belongs to the class of parametric second-order dynamical system, which although not our focus here, but can be reduced by RPMOR as well.

The linear systems that arise here have non-symmetric matrices. As earlier, we use iterative methods instead of direct methods. Of the many available iterative methods for solving non-symmetric linear systems, we use Generalized Conjugate Residual Orthogonal (GCRO) [75]. In fact, we use block GCRO [76, 42] because of the reasons discussed before (availability of the multiple right-hand sides together). We also compare usage of GCRO and block GCRO. The stopping tolerance is taken as 10^{-10} for all cases.

Since unpreconditioned Krylov methods often fail, we do not discuss them here[3]. We use a Modified Sparse Approximate Inverse (MSPAI 1.0) proposed in [20] as our SPAI preconditioner. We use standard initial settings of MSPAI as follows: tolerance (ep) of 0.0001 and cache size (cs) of 80.

We perform our numerical experiments on a machine with the following configuration: Intel Xeon (R) CPU E5-1620 V3 @ 3.50 GHz., frequency 1200 MHz., 8 CPU and 64 GB RAM. All the codes are written in MATLAB (2016b) (including RPMOR, GCRO) except SPAI and reusable SPAI. MATLAB is used because of ease of rapid prototyping. Computing SPAI and reusable SPAI in MATLAB is expensive, therefore, we use C++ version of these (SPAI is from MSPAI and reusable SPAI is written by us). MSPAI further uses BLAS, LAPACK and ATLAS libraries.

As discussed in Chapter 4, we do not integrate our MATLAB code base with the C++ based preconditioner. This is because integrating the two is complicated and is not needed here as well.

We compute SPAI and reusable SPAI in-parallel, separately, and save them on the hard-disk in the standard .mtx files [20]. When we run our MATLAB code base, then these files are read from the hard-disk into the main memory and converted into .mat files for further processing.

5.3.1 The Electro-Chemistry Model

This model is a parametric Single Input Multiple Output (SIMO) first-order linear dynamical system, and is given as

$$\begin{aligned}(sE + G + p_1 D_1 + p_2 D_2)x &= Bu(s),\\ y &= Cx,\end{aligned}$$

where $E, G, D_1, D_2 \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$ and $C \in \mathbb{R}^{q\times n}$ with $n = 16,912$, $m = 1$ and $q = 5$.

We use RPMOR settings as below. In the above equation, there is frequency variable s and two parameters p_1, p_2 (or p_j with $j = 1, \ldots, w$, and $w = 2$), all of which

[3]All our linear system matrices are ill-conditioned as well; condition number of the order 10^{12}.

together form a set of expansion points.

The number of moments matched are three (i.e. up to $h = 2$), and we use four sets of expansion points ($i = 1, 2, \ldots, \mathfrak{z}$ with $\mathfrak{z} = 4$) as follows based upon values in [51]:
$[0.1\pi, 5.946 \times 10^{-10}, 1.68 \times 10^{09}]$,
$[0.2\pi, 8.41 \times 10^{-04}, 1.18 \times 10^{03}]$,
$[0.3\pi, 1.18 \times 10^{03}, 8.41 \times 10^{-04}]$, and
$[0.45\pi, 1.68 \times 10^{09}, 5.94 \times 10^{-10}]$. Since we have four sets of expansion points, RPMOR takes four iterative steps. For the settings above, the size of the reduced system r comes out to be 46.

The arising linear systems have structure as below. Since w is two, we obtain that at each RPMOR step (or set of expansion points), following sets of linear systems should be solved: $1, 3$ and 9, corresponding to the number of right-hand sides available together while matching the zeroth order, first-order, and second-order moments, respectively. However, after solving the linear systems while matching the first-order moment, the solution vector corresponding to the third (and last) right-hand side comes to be nearly zero. Thus, this solution vector is not used as a right-hand side in solving the linear systems while matching the second-order moment. Eventually, for the second-order moment, instead of 9 linear systems, 6 are solved. This brings the total number of linear systems solved at each RPMOR step to be $1 + 3 + 6 = 10$. Since, $n = 16,912$, the linear systems here are of size $16,912 \times 16,912$.

Based upon the above data, block GCRO is called ten times at each RPMOR step[4]. It is executed once initially for solving linear system with 1 right-hand side, one more time for solving linear systems with 3 right-hand sides together, and last time for solving linear systems with 6 right-hand sides together. Hence, block GCRO is called three times at each RPMOR step.

In Table 5.1, we compare SPAI and SPAI update. Both preconditioners are computed once at each RPMOR step and applied to all linear systems at that RPMOR step (ten linear systems here each). This is because the linear system matrices do not

[4]We motivate use of block GCRO more by comparing it with GCRO for the next example (5.3.2). Here, we also do SPAI update analysis.

Table 5.1: Computation Time of SPAI and SPAI Update for Electro-Chemistry Model.

RPMOR Steps	Computation Time (seconds)	
	SPAI	SPAI Update
1	90.10	90.10
2	90.19	1.83
3	90.25	1.83
4	90.29	1.84
Total Time	360.83	93.61

change at a RPMOR step (only right-hand sides do). In the first column, we give the step number of the RPMOR algorithm that corresponds to the four expansion points. The computation times of SPAI and SPAI update are given in columns 2 and 3, respectively, corresponding to each RPMOR step. At the first RPMOR step, both preconditioners take the same amount of computation time because, as earlier, SPAI update is not applicable at this step. From the second step onwards, we see substantial savings with SPAI update as compared to SPAI (approximately 74%).

Table 5.2 gives the computation time of block GCRO with SPAI and block GCRO with SPAI update. In the first column, we give the step number of the RPMOR algorithm. The relevant computation times are given in columns 2 and 3, respectively. As above, at the first RPMOR step, the computation time for both the cases is almost the same. From the second step onwards, we see substantial savings in the computation time by using block GCRO with SPAI update over block GCRO with SPAI. From the last row of this table, it is clear that preconditioner update saves approximately 55% computation time (as compared to non-update).

Table 5.2: Computation Time of Block GCRO with SPAI and SPAI Update for Electro-Chemistry Model.

RPMOR Steps	Computation Time (seconds)	
	Block GCRO plus SPAI	Block GCRO plus SPAI Update
1	127.29	126.38
2	123.37	33.86
3	121.87	30.37
4	117.85	27.52
Total Time	490.38	219.04

5.3.2 The Micro-Gyroscope Model

This model is a parametric Single Input Single Output (SISO) second-order linear dynamical system of size $17,931$, and is given as

$$\begin{aligned}(s^2M(d) + sD(\theta,\alpha,\beta,d) + K(d))x &= Bu(s),\\ y &= Cx,\end{aligned} \tag{5.3}$$

where $M(d), D(\theta,\alpha,\beta,d), K(d) \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$ and $C \in \mathbb{R}^{q\times n}$ with $n = 17,931, m = 1$ and $q = 1$. Further, $M(d) = M_1 + dM_2$, $D(\theta,\alpha,\beta,d) = \theta(D_1 + dD_2) + \alpha M(d) + \beta K(d)$, and $K(d) = K_1 + (1/d)K_2 + dK_3$. In the above equation, there are eleven variables and all must be considered as individual parameters.

Next, we discuss the settings for RPMOR. The above parameters form the i^{th} set of expansion points as follows: $\grave{s}_1^i = s^2, \grave{s}_2^i = s^2d, \grave{s}_3^i = s\theta, \grave{s}_4^i = s\theta d, \grave{s}_5^i = s\alpha, \grave{s}_6^i = s\alpha d, \grave{s}_7^i = s\beta, \grave{s}_8^i = s\beta/d, \grave{s}_9^i = s\beta d, \grave{s}_{10}^i = 1/d$ and $\grave{s}_{11}^i = d$. Usually α and β are taken as zero [70], and hence we are left with six parameters; $\grave{s}_1^i, \grave{s}_2^i, \grave{s}_3^i, \grave{s}_4^i, \grave{s}_{10}^i$ and $\grave{s}_{11}^i$. Considering s_1^i as the frequency variable, we have five actual parameters (in the earlier notation p_j with $j = 1,\ldots,w$ and $w = 5$).

The number of moments matched are three (i.e. up to $h = 2$), and we use four sets of expansion points ($i = 1, 2, \ldots, \mathfrak{z}$ with $\mathfrak{z} = 4$) as follows based upon values in [70]:
$\left[-4\pi^2 \times 0.065, -4\pi^2 \times 0.065, 5\pi\sqrt{-1} \times 10^{-7}, 5\pi\sqrt{-1} \times 10^{-7}, 1, 1\right]$,

$\left[-4\pi^2 \times 0.065, -8\pi^2 \times 0.065, 5\pi\sqrt{-1} \times 10^{-7}, 10\pi\sqrt{-1} \times 10^{-7}, 0.5, 2\right]$,
$\left[-4\pi^2 \times 0.0225, -8\pi^2 \times 0.0225, 3\pi\sqrt{-1} \times 10^{-7}, 6\pi\sqrt{-1} \times 10^{-7}, 0.5, 2\right]$, and
$\left[-4\pi^2 \times 0.0225, -4\pi^2 \times 0.0337, 3\pi\sqrt{-1} \times 10^{-7}, 4.5\pi\sqrt{-1} \times 10^{-7}, 0.66, 1.5\right]$.
Since here we have four sets of expansion points, RPMOR takes four iterative steps. As mentioned earlier, RPMOR automatically determines the size of the reduced system (r). For the settings above, r comes out to be 304.

Further, we discuss the arising sequence of linear systems. Since w is five corresponding to five parameters, we obtain that at each RPMOR step (or a set of expansion points), following number of linear systems are solved: $1, 6$ and 36, corresponding to the number of right-hand sides available together while matching the zeroth order, first-order and second-order moments, respectively. This results in the total linear systems to be solved at each RPMOR step to be $1 + 6 + 36 = 43$. Since $n = 17,931$, the linear systems that arise here are of size $17,931 \times 17,931$.

First, we compare GCRO and block GCRO method (in-terms of number of matrix-vector products and computation time). As mentioned earlier, both GCRO and block GCRO are solved using the SPAI preconditioner, and none of them converged without preconditioning. Usually, preconditioning is discussed before the linear solver. However, here, we first narrow down on our choice of linear solver (in this case block GCRO), and then discuss our more important contribution of the comparison between the SPAI preconditioner and the SPAI update preconditioner.

5.3.2.1 GCRO Vs Block GCRO

Table 5.3 provides a detailed comparison between GCRO and block GCRO. In this table, column 1 lists the RPMOR steps. GCRO is executed 43 times at each RPMOR step because of 43 linear systems. Block GCRO is executed once initially for solving linear system with 1 right-hand side, one more time for solving linear systems with 6 right-hand sides together, and last time for solving linear systems with 36 right-hand sides together. Hence, block GCRO is called three times at each RPMOR step. Both these data are given in columns 2 and 5 of this table, respectively. Number of matrix vector products for both the solvers (GCRO and block GCRO) are given in columns

3 and 6, respectively. Computation times corresponding to these solvers are given in columns 4 and 7, respectively (rounded to the nearest second). For GCRO, we give averaged values because we can extract the required information also from the average. From the last row of Table 5.3 it is clear that block GCRO saves nearly 89% in the number of matrix vector products and about 95% in the computation time as compared to GCRO. Thus, for rest of our experiments we use block GCRO only.

We now analyze Table 5.3 further. The first observation here is, the number of matrix-vector products needed for solving linear systems with 6 and 36 right-hand sides by block GCRO (see column 6) are much smaller than solving 6 and 36 linear systems with 1 right-hand side by GCRO (see column 3; also same as using block GCRO), respectively. This could be because of "deflation" as defined in [77], where-in rapid convergence happens in a block Krylov method. An example of deflation is when the initial residual columns are equal to a linear combination of a set of eigenvectors of the linear system matrix. For us too, there is direct relation between the initial residual (formed from block right-hand sides) and the linear system matrix.

From (5.1), we know that 6 and 36 right-hand sides arise while matching the first-order and second-order moments, respectively. These right-hand sides are formed using dynamical system sub-matrices $(\mathbb{A}_1, \ldots, \mathbb{A}_{w+1})$ and the solution of the previous linear systems $\big(x^{(i)}(1)$ and $x^{(i)}(2), \cdots, x^{(i)}\left((w+1)+1\right)\big)$, where the former are used to build the linear system matrix (recall (3.7), (3.10) and (3.11)).

The second observation from Table 5.3 is that, the computation time per iteration for solving linear systems with 1, 6, and 36 right-hand sides are all very close (data of column 7 divided by respective data of column 6). This is contrary to the fact that in the first case, a sparse matrix-vector product (spMV) is used, time of which is usually much less than the time for a sparse matrix-matrix product (spMM), as used in the last two cases. The reason for this is, our linear system matrices are very sparse (average number of non-zeros of 300 per row in a matrix of size $17,931 \times 17,931$). Thus, the average computation time of spMM is almost the same as the average computation

Table 5.3: GCRO and Block GCRO Iteration Count and Computation Time for Micro-Gyroscope Model. Here, NMVs implies Number of Matrix- Vector Products, and RHSs implies Right-Hand Sides.

RPMOR Steps	GCRO			Block GCRO		
	No. of Linear Solves	Average Iteration Count per Linear Solve	Average Computation Time per Linear Solve (seconds)	No. of Block Linear Solves	Iteration Count for Solving (1, 6, 36) RHSs	Computation Time for Solving (1, 6, 36) RHSs (seconds)
1	43	1472	92	03	(1474, 308, 77)	(92, 34, 25)
2	43	1473	92	03	(1905, 394, 97)	(149, 51, 36)
3	43	1442	88	03	(1991, 400, 97)	(165, 54, 35)
4	43	1472	92	03	(1753, 350, 86)	(129, 42, 30)
Total	$43 \times 4 = 172$	43× Sum of Col. $= 43 \times 5,859$ $= 251,937$ (also NMVs)	43× Sum of Col. $= 43 \times 364$ $= 15,652$ $= 261$ (mins)	$03 \times 4 = 12$	Sum of Sub Col.'s $= (7123, 1452, 357)$ NMVs $= 1 \times 7123 + 6 \times 1452 + 36 \times 357$ $= 28,687$	Sum of Col. $= 842$ $= 14$ (mins)

time of spMV[5].

5.3.2.2 Analyzing SPAI Update

Here, we analyze why SPAI update would be beneficial. Recall, the first linear system matrix, $\mathscr{A}(1)$ defined in (3.7), is given as

$$\mathscr{A}(1) = K_1 + \grave{s}_1^1 M_1 + \grave{s}_2^1 M_2 + \grave{s}_3^1 D_1 + \grave{s}_4^1 D_2 + \grave{s}_{10}^1 K_2 + \grave{s}_{11}^1 K_3,$$

and other matrices, $\mathscr{A}(i)$ for $i = 2, \ldots, 4$ defined in (3.10) and (3.11), are given as

$$\mathscr{A}(i) = K_1 + \grave{s}_1^i M_1 + \grave{s}_2^i M_2 + \grave{s}_3^i D_1 + \grave{s}_4^i D_2 + \grave{s}_{10}^i K_2 + \grave{s}_{11}^i K_3.$$

Based upon the theory of preconditioner updates, SPAI update is useful when $\|I - \mathscr{A}(i)\|_f / \|I\|_f$ is large and $\|\mathscr{A}(1) - \mathscr{A}(i)\|_f / \|\mathscr{A}(1)\|_f$ is small, which is true in this case (see columns 2 and 3 of Table 5.4, respectively). Since the change in matrix for different RPMOR steps is not evident from this table, we plot the five smallest

[5]0.016142 seconds for spMM with 6 right-hand sides and 0.012798 seconds for spMV.

Table 5.4: SPAI and SPAI Update Analysis for Micro-Gyroscope Model.

RPMOR Steps (i)	$\frac{\|I - \mathcal{A}(i)\|_f}{\|I\|_f}$	$\frac{\|\mathcal{A}(1) - \mathcal{A}(i)\|_f}{\|\mathcal{A}(1)\|_f}$
1	1.02	0
2	1.02	2.83×10^{-05}
3	1.02	2.83×10^{-05}
4	1.02	1.77×10^{-05}

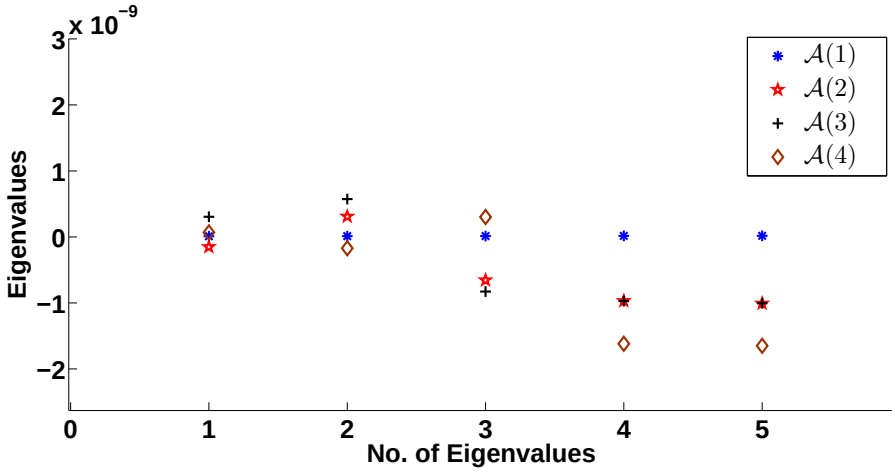

Figure 5.1: Changes in Linear systems for Micro-Gyroscope Model.

eigenvalues of $\mathcal{A}(i),\ i = 1, 2, 3, 4$ in Figure 5.1. It is clear from this figure that matrices $\mathcal{A}(2), \mathcal{A}(3),$ and $\mathcal{A}(4)$ are all very close to $\mathcal{A}(1)$ (eigenvalues of all around zero) as compared to their closeness to the identity matrix (whose eigenvalues are one).

5.3.2.3 Benefit from SPAI Update

As mentioned while discussing cheap preconditioner updates, SPAI and SPAI update are computed only when the matrix changes, which happens once at every RPMOR step. Since the number of iterative steps taken by RPMOR are four, these preconditioners are computed four times. Also, at each RPMOR step, the respective preconditioner is applied to the 43 linear systems (with differing right-hand sides).

In Table 5.5, we compare SPAI and SPAI update[6]. In the first column, we give

[6]The underlying linear solver, GCRO or block GCRO, does not affect this.

the step number of RPMOR algorithm that corresponds to the four expansion points. The computation times of SPAI and SPAI update are given in columns 2 and 3, respectively, corresponding to each RPMOR step. At the first RPMOR step, both preconditioners take the same amount of computation time. This is because SPAI update is not applicable at this step. From second RPMOR step onwards, we see substantial savings with SPAI update as compared to SPAI (approximately 70%).

Table 5.6 gives the computation time of block GCRO with SPAI and block GCRO with SPAI update. In the first column we give the step number of the RPMOR algorithm. The relevant computation times are given in columns 2 and 3, respectively. As above, at the first RPMOR step, the computation time for both cases is almost the same. From the second step onwards, we see substantial savings in computation time by using block GCRO with SPAI update over block GCRO with SPAI. From the last row of this table, it is clear that preconditioner update saves approximately 62% computation time (as compared to non-update).

As cross-checking mechanism, we can see that adding block GCRO time from Table 5.3 (14 mins from column 7) and SPAI time from Table 5.5 (105 mins from column 2), gives us the total block GCRO plus SPAI time as in Table 5.6 (119 mins from column 2). However, adding block GCRO time from Table 5.3 (14 mins from column 7) and SPAI update time from Table 5.5 (33 mins from column 3), does not exactly give us the total block GCRO plus SPAI update time as in Table 5.6 (45.22 mins from column 3). The reason for this (45.22 mins instead of 47 mins) is that block GCRO in Table 5.3 has been executed with SPAI instead of SPAI update, and block GCRO with SPAI update has a slightly faster convergence than block GCRO with SPAI.

To demonstrate the quality/ accuracy of reduced system, we plot the relative error between the transfer function of the original system and the reduced system[7]. We use the reduced system as obtained by using block GCRO and SPAI update (corresponding to the third column of Table 5.6). The error computation as done in [70], forms the basis of our setup here (see Section 5.4 in [70]). We use the following settings in

[7]This is more general than computing the the error between the original system output and the reduced system output since for the latter, input range is also required.

Table 5.5: Computation Time of SPAI and SPAI update for Micro-Gyroscope Model.

RPMOR Steps	Computation Time (minutes)	
	SPAI	SPAI Update
1	27	27
2	26	2
3	26	2
4	26	2
Total Time	105	33

(5.3): $s = 2\pi\sqrt{(-1)}f$, where frequency f vector consists of equally spaced twenty points between 0.025 and 0.25 MHz; d vector consist of equally spaced twenty points between 1 and 2; and $\theta = 10^{-06}$. This result is given in Figure 5.2, with f on the x-axis, parameter d on the y-axis, and error on the z-axis (in log scale). As evident from this figure, the obtained reduced system is good (error is very small).

If we compare the preconditioned iterative method timings for Electro-Chemistry Model and Micro-Gyroscope Model (Table 5.1–5.5 and Table 5.2–5.6), we realize that the former is substantially faster than the latter although both are of almost the same size. The reason behind this is as follows: the linear system matrices in the former are much more sparse than those in the latter (number of non-zeros per row 7 versus 300).

Table 5.6: Computation Time of Block GCRO with SPAI and SPAI Update for Micro-Gyroscope Model.

RPMOR Steps	Computation Time (minutes)	
	Block GCRO plus SPAI	Block GCRO plus SPAI Update
1	29.51	29.73
2	29.92	4.98
3	30.23	5.53
4	29.34	4.98
Total Time	119	45.22

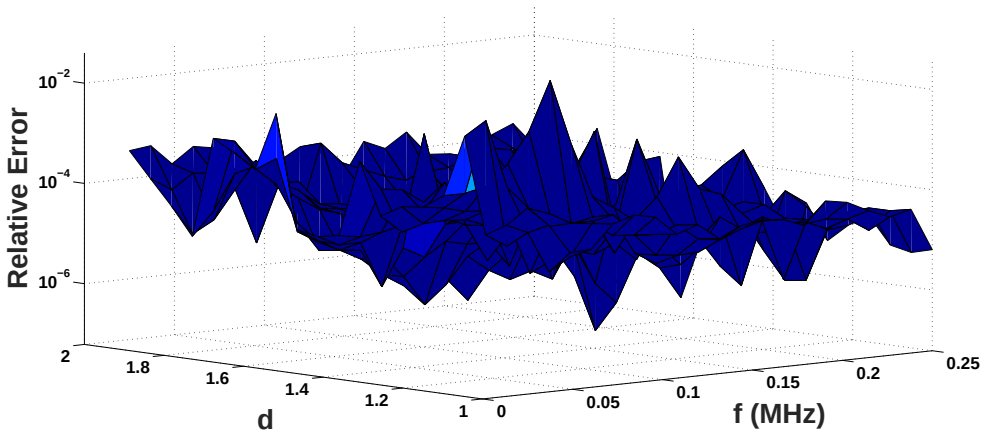

Figure 5.2: Relative Error between the Original and the Reduced System for Micro-Gyroscope Model.

Chapter 6

Conclusions and Future Work

We discuss application of preconditioned iterative methods for solving the large linear systems in MOR algorithms for linear non-parametric/ parametric and first-order/ second-order dynamical systems. These methods find solutions only up to a certain tolerance. Hence, in the first-half of the dissertation we show that under some mild assumptions, these algorithms are backward stable with respect to these inexact linear solves. We also analyze the accuracy of the resulting reduced system, and support all our results with multiple numerical experiments. In the second-half of the dissertation, we present the technique of reusing preconditioners in the linear solves, and again support our theory with multiple numerical experiments.

6.1 Stability Analysis

Chapter 2 focuses on stability analysis of MOR algorithms for non-parametric dynamical systems. Since such an analysis for a popular MOR algorithm for non-parametric first-order dynamical systems (i.e. IRKA) has been done [21], this chapter specifically focuses on a common MOR algorithm for non-parametric second-order dynamical systems (i.e. AIRGA).

Often the matrices of the dynamical systems (i.e. M, D, K etc.) are dependent on multiple parameters (e.g., material property etc.). Algorithms have been proposed for model reduction of such parametric first-order and second-order dynamical systems

as well. Chapter 3 looks at stability analysis of such MOR algorithms. For MOR of parametric first-order dynamical systems, RPMOR algorithm is studied.

Initially, in both Chapters 2 and 3, first and second conditions for stability are analyzed, and then the stability results are supported by numerical experiments. Next, we look at future work in the stability context.

The *first* assumption for stability enforces the use of a Ritz-Galerkin based linear solver, where the residual of a linear system is made orthogonal to the corresponding Krylov subspace. The *second* assumption for stability requires satisfying few other orthogonalities. Since the CG method is the most popular linear solver based upon the Ritz-Galerkin theory and is ideal for SPD linear systems, we focus on SPD systems only. We use Recycling CG (RCG) to achieve the extra orthogonalities. The future work here involves modifying other methods based upon the Ritz-Galerkin theory (to achieve extra orthogonalities), which can be used to solve general non-symmetric indefinite linear systems. For example, the Full Orthogonalization Method (FOM).

The *third* set of assumptions for stability are that $A(s)$, which is a function of the frequency (s), and dynamical system matrices, is invertible and its norm is bounded by one. These assumptions are easily satisfied for all our models, but they may not always hold. One future direction here involves better characterizing these assumptions in-terms of the underlying dynamical system. Another direction is transforming the conditions on $A(s)$ in-terms of $M, D,$ and K since $A(s) = s^2M + sD + K$.

The *fourth and final* set of assumptions for stability involve being able to compute perturbation Z from the given expression and bounding its 2-norm by one. As earlier, although for all our models these assumptions are easily satisfied, they may not always hold. Z is dependent on the linear solver stopping tolerances. Hence, we need to study range of these tolerances when the 2-norm of this perturbation could be bounded by one.

The stability conditions that we have proposed have been sufficiency conditions for the most general input dynamical system matrices. One future direction in this context is to find sufficiency conditions for special matrices (e.g., where the input dynamical system matrices are symmetric, positive definite, diagonally dominant, etc.). Another

direction here is to workout the necessary conditions for stability for both the general case and the special cases since such conditions often help to understand the problem better.

The condition number of the dynamical system, which we use is an approximation to the ideal condition number. That is, condition number of the dynamical system with respect to computing the H_2-norm of the error between the inexactly computed reduced system and the exactly computed reduced system. This is also part of the future work.

6.2 Preconditioner Reuse

Chapter 4, which like Chapter 2, focuses on MOR algorithms for non-parametric dynamical systems, however, the focus here is on the preconditioners as compared to stability earlier.

We show use of the SPAI preconditioner because it is inherently parallel, and especially useful in solving linear systems with exponentially increasing sizes, which arise here. The linear system matrices here change slightly during the model order reduction process. Using this, we propose a technique to cheaply update the SPAI preconditioner.

Since preconditioner reuse in a MOR algorithm for non-parametric first-order dynamical systems (i.e. IRKA) has done in [7, 30], this chapter specifically focused on reusing preconditioners in a MOR algorithm for non-parametric second-order dynamical systems (i.e. AIRGA).

This chapter begins by proposing a general theory for reusing preconditioners in MOR algorithms (irrespective of parametrization and order of the dynamical systems). It then makes the following contributions:

Multiple ways of reusing preconditioners within the algorithm, efficient implementation to ensure that the savings because of reusing preconditioners are not negated by bad coding, and experimentation on a massively large industrial problem. Numerical experiments show the effectiveness of our approach, where for a problem of size 1.2

million, we save upto 64% in the computation time. In absolute terms, this gives a saving of 5 days.

Chapter 5, which like Chapter 3, focuses on MOR algorithms for parametric dynamical systems (both first-order and second-order) while looking at the preconditioner aspect. Again, in the first-order context, RPMOR algorithm is looked at. The novelty here is that our update exploits the slowly changing behaviour of the model parameters, which was not done earlier. For a model problem, while using GCRO as the underlying iterative solver, we show that using a block GCRO saves about 95% of computation time over its non-block version. In absolute terms, this gives a saving of 4 hours. Further, for two model problems, we show that by using block GCRO with SPAI update (instead of block GCRO with SPAI) around 60% of the time is saved. In absolute terms, for first problem, this gives a saving of 1 hour 14 minutes. And, for the second problem, this gives a saving of 4 minutes.

In future, we will investigate more sophisticated preconditioning strategies that will further exploit the properties of the underlying MOR algorithms as well as the resulting linear systems. Specifically, we will explore five directions as below.

(a) Besides the currently used basic SPAI preconditioner, we will investigate the use of high-order convergent approximate inverse preconditioners [57, 78] as well as hybrid versions, which use a combination of factorization and approximate inverse techniques [59, 56].

(b) For MOR algorithms where general shifted linear systems arise, we will investigate the use of our reusable SPAI preconditioner along with shift-invariant preconditioners that have been developed specifically for such shifted linear systems [65, 68].

(c) We will investigate exploiting the block structure of the linear system coefficient matrices in some MOR algorithms such that the SPAI and its reuse can be done more efficiently [79].

(d) Since randomized preconditioners have shown promising results in recent years, we will explore their use in the context of linear systems in MOR algorithms.

(e) Finally, we would also investigate combining machine learning techniques (e.g., spiking neural networks) to optimize the parameters inside the preconditioners

6.3 General Future Directions

In-general, we also plan to perform inexact linear solves in MOR algorithms[1] based upon other techniques (besides projection, which we focus on). Balanced truncation is an alternative to the projection based methods for performing model reduction. Here, large scale Lyapunov equations have to be solved. Alternating Directions Implicit (ADI) based algorithms are widely used for solving such equations. ADI algorithms also require solving sequences of large sparse linear systems.

Another example is a MOR algorithm that is data-driven, which is very common these days, i.e. Data-Driven Parametrized Model Reduction algorithm in the Loewner Framework (PMOR-L) [54]. We have yet to perform the stability analysis for this algorithm but we have worked out the reuse of preconditioners for it in Appendix B.

Finally, in future, we intend to look at performing iterative solves in MOR algorithms for non-linear dynamical systems, with all permutations and combinations related to parametrization (non-parametric/ parametric) and order of the dynamical system (first-order/ second-order/ higher-orders).

Bilinear dynamical systems are a bridge between linear and highly nonlinear dynamical systems. Since stability analyses of such types of systems (with respect to inexact linear solves) have been extensively worked out (see [80, 22, 23]), we apply preconditioner reuse theory to two algorithms of this category in Appendices C and D.

[1] linear; non-parametric/ parametric; first-order/ second-order.

Bibliography

[1] L. Feng, P. Benner, and J. G. Korvink. Subspace recycling accelerates the parametric macro-modeling of MEMS. *International Journal for Numerical Methods in Engineering*, 94(1):84–110, 2013.

[2] E. J. Grimme. *Krylov projection methods for model reduction.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 1997.

[3] A. C. Antoulas. *Approximation of Large-Scale Dynamical Systems.* SIAM Advances in Design and Control, Philadelphia, PA, USA, 2005.

[4] S. Gugercin, A. C. Antoulas, and C. Beattie. $\mathscr{H}_2$ model reduction for large-scale linear dynamical systems. *SIAM Journal on Matrix Analysis and Applications*, 30(2):609–638, 2008.

[5] T. Breiten. *Interpolation methods for model reduction of large-scale dynamical systems.* PhD thesis, Otto Von Guericke University of Magdeburg, Magdeburg, Germany, 2013.

[6] A. Bunse-Gerstner, D. Kubalińska, G. Vossen, and D. Wilczek. H_2-norm optimal model reduction for large scale discrete dynamical mimo systems. *Journal of Computational and Applied Mathematics*, 233(5):1202–1216, 2010.

[7] S. A. Wyatt. *Issues in Interpolatory Model Reduction: Inexact Solves, Second-order Systems and DAEs.* PhD thesis, Virginia Tech, USA, 2012.

[8] Z. Y. Qiu, Y. L. Jiang, and J. W. Yuan. Interpolatory model order reduction method for second order systems. *Asian Journal of Control*, 20(1):312–322, 2018.

[9] Z. Bai and Y. Su. Dimension reduction of large-scale second-order dynamical systems via a second-order Arnoldi method. *SIAM Journal on Scientific Computing*, 26(5):1692–1709, 2005.

[10] T. Bonin, H. Faßbender, A. Soppa, and M. Zaeh. A fully adaptive rational global Arnoldi method for the model-order reduction of second-order MIMO systems with proportional damping. *Mathematics and Computers in Simulation*, 122:1–19, 2016.

[11] U. Baur, C. Beattie, P. Benner, and S. Gugercin. Interpolatory projection methods for parameterized model reduction. *SIAM Journal on Scientific Computing*, 33(5):2489–2518, 2011.

[12] P. Benner and L. Feng. A robust algorithm for parametric model order reduction based on implicit moment matching. In A. Quarteroni and G. Rozza, editors, *Reduced Order Methods for Modeling and Computational Reduction*, pages 159–185. Springer International Publishing, Cham, 2014.

[13] N. Son and T. Stykel. Solving parameter-dependent Lyapunov equations using the reduced basis method with application to parametric model order reduction. *SIAM Journal on Matrix Analysis and Applications*, 38(2):478–504, 2017.

[14] M. Saadvandi, K. Meerbergen, and W. Desmet. Parametric dominant pole algorithm for parametric model order reduction. *Journal of Computational and Applied Mathematics*, 259:259–280, 2014.

[15] M. Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418 – 477, 2002.

[16] T. Davis. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2006.

[17] N. Alon and R. Yuster. Matrix sparsification and nested dissection over arbitrary fields. *Journal of the ACM*, 60(4):1–18, 2013.

[18] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[19] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, 19(3):995–1023, 1998.

[20] A. Kallischko. *Modified Sparse Approximate Inverses (MSPAI) for Parallel Preconditioning*. PhD thesis, Technische Universität München, Germany, 2007.

[21] C. Beattie, S. Gugercin, and S. A. Wyatt. Inexact solves in interpolatory model reduction. *Linear Algebra and its Applications*, 436(8):2916–2943, 2012.

[22] R. Choudhary and K. Ahuja. Stability analysis of bilinear iterative rational Krylov algorithm. *Linear Algebra and its Applications*, 538:56–88, 2018.

[23] R. Choudhary and K. Ahuja. Inexact linear solves in model reduction of bilinear dynamical systems. *IEEE Access*, 7:72297–72307, 2019.

[24] D. Lu, Y. Su, and Z. Bai. Stability analysis of the two-level orthogonal Arnoldi procedure. *SIAM Journal on Matrix Analysis and Applications*, 37(1):195–214, 2016.

[25] K. Ahuja. *Recycling Krylov Subspaces and Preconditioners*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, 2011.

[26] K. Ahuja, B. K. Clark, E. de Sturler, D. M. Ceperley, and J. Kim. Improved scaling for quantum Monte Carlo on insulators. *SIAM Journal on Scientific Computing*, 33(4):1837–1859, 2011.

[27] S. Bellavia, V. De Simone, D. di Serafino, and B. Morini. Efficient preconditioner updates for shifted linear systems. *SIAM Journal on Scientific Computing*, 33(4):1785–1809, 2011.

[28] L. Wei-Hua, H. Ting-Zhu, L. Liang, Z. Yong, and G. Xian-Ming. Efficient preconditioner updates for unsymmetric shifted linear systems. *Computers & Mathematics with Applications*, 67(9):1643–1655, 2014.

[29] A. K. Grim-McNally, E. de Sturler, and S. Gugercin. Preconditioning parametrized linear systems. *Preprint:arXiv:1601.05883*, 2020.

[30] A. K. Grim-McNally. Reusing and updating preconditioners for sequences of matrices. Master's thesis, Virginia Tech, USA, 2015.

[31] J. M. Wang, C. C. Chu, Q. Yu, and E. S. Kuh. On projection-based algorithms for model-order reduction of interconnects. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 49(11):1563–1585, 2002.

[32] C. Beattie and S. Gugercin. Krylov-based model reduction of second-order systems with proportional damping. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 2278–2283, 2005.

[33] K. Jbilou, A. Messaoudi, and H. Sadok. Global FOM and GMRES algorithms for matrix equations. *Applied Numerical Mathematics*, 31(1):49–63, 1999.

[34] M. Sadkane. Block-Arnoldi and Davidson methods for unsymmetric large eigenvalue problems. *Numerische Mathematik*, 64(1):195–211, 1993.

[35] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[36] H. A. Van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, New York, USA, 2003.

[37] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, and S. Maiti. Recycling Krylov subspaces for sequences of linear systems. *SIAM Journal on Scientific Computing*, 28(5):1651–1674, 2006.

[38] S. Wang, E. de Sturler, and G. H. Paulino. Large-scale topology optimization using preconditioned Krylov subspace methods with recycling. *International Journal for Numerical Methods in Engineering*, 69(12):2441–2468, 2007.

[39] K. Ahuja, E. de Sturler, S. Gugercin, and E. R. Chang. Recycling BiCG with an application to model reduction. *SIAM Journal on Scientific Computing*, 34(4):A1925–A1949, 2012.

[40] K. Ahuja, P. Benner, E. de Sturler, and L. Feng. Recycling BiCGSTAB with an application to parametric model order reduction. *SIAM Journal on Scientific Computing*, 37(5):S429–S446, 2015.

[41] D. M. Young and K. C. Jea. Generalized conjugate-gradient acceleration of non-symmetrizable iterative methods. *Linear Algebra and its Applications*, 34:159–194, 1980.

[42] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.

[43] M. L. Parks, P. K.V.V. Nukala, and S. Šimunović. Efficient simulation of large-scale 3D fracture networks via Krylov subspace recycling. *Paper Draft*, 2010.

[44] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h. A deflated version of the conjugate gradient algorithm. *SIAM Journal on Scientific Computing*, 21(5):1909–1926, 2000.

[45] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[46] C. D. Meyer. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[47] D. Billger. The Butterfly Gyro (35889). `https://portal.uni-freiburg.de/imteksimulation/downloads/benchmark/The%20Butterfly%20Gyro%20%2835889%29/`.

[48] F. Tisseur and K. Meerbergen. The quadratic eigenvalue problem. *SIAM Review*, 43(2):235–286, 2001.

[49] T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation*, 34(150):473–497, 1980.

[50] P. Benner, S. Gugercin, and K. Willcox. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM Review*, 57(4):483–531, 2015.

[51] P. Benner and L. Feng. A robust algorithm for parametric model order reduction based on implicit moment matching. In A. Quarteroni and G. Rozza, editors, *Reduced Order Methods for Modeling and Computational Reduction*, pages 159–185. Springer International Publishing, Cham, 2014.

[52] S. Gugercin. *Projection methods for model reduction of large-scale dynamical systems*. PhD thesis, ECE Dept., Rice University, Houston, TX, USA, 2002.

[53] K. Ogata. *Modern Control Engineering*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[54] A. Ionita and A. Antoulas. Data-driven parametrized model reduction in the Loewner framework. *SIAM Journal on Scientific Computing*, 36(3):A984–A1007, 2014.

[55] M. Benzi and M. Tuma. A sparse approximate inverse preconditioner for non-symmetric linear systems. *SIAM Journal on Scientific Computing*, 19(3):968–994, 1998.

[56] S. C. Buranay and O. C. Iyikal. Approximate schur-block ilu preconditioners for regularized solution of discrete ill-posed problems. *Mathematical Problems in Engineering*, 1912535, 2019.

[57] S. C. Buranay, D. Subasi, and O. C. Iyikal. On the two classes of high-order convergent methods of approximate inverse preconditioners for solving linear systems. *Numerical Linear Algebra with Applications*, 24(6):e2111, 2017.

[58] F. Soleymani. A fast convergent iterative solver for approximate inverse of matrices. *Numerical Linear Algebra with Applications*, 21(3):439–452, 2014.

[59] L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditionings i. theory. *SIAM Journal on Matrix Analysis and Applications*, 14(1):45–58, 1993.

[60] N. I. M. Gould and J. A. Scott. Sparse approximate-inverse preconditioners using norm-minimization techniques. *SIAM Journal on Scientific Computing*, 19(2):605–625, 1998.

[61] Y. Saad and M. H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[62] H. A. van der Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.

[63] P. Sonneveld and M. B. van Gijzen. Idr(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. *SIAM Journal on Scientific Computing*, 31(2):1035–1062, 2009.

[64] V. Simoncini. Restarted full orthogonalization method for shifted linear systems. *BIT Numerical Mathematics*, 43:459–466, 2003.

[65] S. Ladenheim A. K. Saibaba T. Bakhos, P. K. Kitanidis and D. B. Szyld. Multi-preconditioned gmres for shifted systems. *SIAM Journal on Scientific Computing*, 39(5):S222–S247, 2017.

[66] B. Carpentieri A. Imakura K. Zhang X.-M. Gu, T.-Z. Huang and L. Du. Efficient variants of the cmrh method for solving a sequence of multi-shifted non-hermitian linear systems simultaneously. *Journal of Computational and Applied Mathematics*, 375:112788, 2020.

[67] O. Axelsson and A. Kucherov. Real valued iterative methods for solving complex symmetric linear systems. *Numerical Linear Algebra with Applications*, 7(4):197–218, 2000.

[68] G. Yin B. Carpentieri X.-M. Gu, T.-Z. Huang and C. Wen. Restarted hessenberg method for solving shifted nonsymmetric linear systems. *Journal of Computational and Applied Mathematics*, 331:166–177, 2018.

[69] S. Quraishi C. Schröder N. Gräbner, V. Mehrmann and V. U. Wagner. Numerical methods for parametric model reduction in the simulation of disk brake squeal. *Journal of Applied Mathematics and Mechanics*, 96(12):1388–1405, 2016.

[70] L. Feng, P. Benner, and J. G. Korvink. Subspace recycling accelerates the parametric macro-modeling of MEMS. *International Journal for Numerical Methods in Engineering*, 94(1):84–110, 2013.

[71] D. P. O'Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra and its Applications*, 29:293–322, 1980.

[72] V. Simoncini and E. Gallopoulos. A hybrid block GMRES method for nonsymmetric systems with multiple right-hand sides. *Journal of Computational and Applied Mathematics*, 66(1):457–469, 1996.

[73] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, and S. Maiti. Recycling Krylov subspaces for sequences of linear systems. *SIAM Journal on Scientific Computing*, 28(5):1651–1674, 2006.

[74] A. Frommer, K. Lund, and D. B. Szyld. Block Krylov subspace methods for computing functions of matrices applied to multiple vectors. *Electronic Transactions on Numerical Analysis*, 47(4):100–126, 2017.

[75] E. de Sturler. Nested Krylov methods based on GCR. *Journal of Computational and Applied Mathematics*, 67(1):15–41, 1996.

[76] M. L. Parks and K. M. Soodhalter. Block GCRO-DR. *in Belos package of the Trilinos C++ Library, 2011.*

[77] M. H. Gutknecht. Block Krylov space methods for linear systems with multiple right-hand sides. Presentation in Joint Workshop on Computational Chemistry and Numerical Analysis. Also available online as a report with URL: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.189.5036`, 2005–2006.

[78] F. Soleymani. A fast convergent iterative solver for approximate inverse of matrices. *Numerical Linear Algebra with Applications*, 21(3):439–452, 2014.

[79] C. C. K. Mikkelsen. *Numerical methods for Lyapunov equations.* PhD thesis, Purdue University, Indiana, USA, 2009.

[80] R. Choudhary and K. Ahuja. *Stability analysis of inexact linear solves in model order reduction.* PhD thesis, Indian Institute of Technology Indore, Indore, MP, India, 2019.

[81] P. Benner and T. Breiten. Interpolation-based H_2-model reduction of bilinear control systems. *SIAM Journal on Matrix Analysis and Applications*, 33(3):859–885, 2012.

[82] M. M. A. Asif, M. I Ahmad, P. Benner, L. Feng, and T. Stykel. Implicit higher-order moment matching technique for model reduction of quadratic-bilinear systems. *Journal of the Franklin Institute*, 2020.

Appendix A

Satisfying Second Condition of Backward Stability

For satisfying the second condition of backward stability (given in (2.9)) in the listed MOR algorithms (in Table 1.1, mainly for AIRGA[10] and RPMOR[51]), we use Theorem 4.3 from [21]. This translates to the following result.

Theorem A.1 *If* $\|Z\|_2 < \frac{1}{\|\mathscr{A}(arg)^{-1}\|_{H_\infty}}$ *then*

$$\|H(arg) - \tilde{H}(arg)\|_{H_2} \leq \frac{\|\mathscr{A}(arg)^{-1}B\|_{H_\infty}\|\mathscr{C}^T\mathscr{A}(arg)^{-1}\|_{H_2}}{1 - \|\mathscr{A}(arg)^{-1}\|_{H_\infty}\|Z\|_2}\|Z\|_2, \tag{A.1}$$

where $\mathscr{A}(arg) = A(s) = (s^2M + sD + K)$ *(in case of non-parametric; AIRGA);* $\mathscr{A}(arg) = \mathscr{A}(i) = \mathbb{A}_0 + \grave{s}_1^i\mathbb{A}_1 + \ldots + \grave{s}_{w+1}^i\mathbb{A}_{w+1}$ *(in case of parametric; RPMOR);* $H(arg) = \mathscr{C}^T\mathscr{A}(arg)^{-1}\mathscr{B}$ *and* $\tilde{H}(arg) = \mathscr{C}^T\big(\mathscr{A}(arg) + Z\big)^{-1}\mathscr{B}$.

Now,

$$\begin{aligned}
H\,(arg) - \widetilde{H}\,(arg) &= \mathscr{C}^T\mathscr{A}(arg)^{-1}\mathscr{B} - \mathscr{C}^T\big(\mathscr{A}(arg) + Z\big)^{-1}\mathscr{B}, \\
&= \mathscr{C}^T\Big(\mathscr{A}(arg)^{-1}\mathscr{B} - \big(\mathscr{A}(arg) + Z\big)^{-1}\mathscr{B}\Big), \\
&= \mathscr{C}^T\Big(\mathscr{A}(arg)^{-1}\mathscr{B} - \big(I + \mathscr{A}(arg)^{-1}Z\big)^{-1}\mathscr{A}(arg)^{-1}\mathscr{B}\Big), \\
&= \mathscr{C}^T\Big(I - \big(I + \mathscr{A}(arg)^{-1}Z\big)^{-1}\Big)\mathscr{A}(arg)^{-1}\mathscr{B}, \\
&= \mathscr{C}^T\mathscr{A}(arg)^{-1}Z\Big(I + \mathscr{A}(arg)^{-1}Z\Big)^{-1}\mathscr{A}(arg)^{-1}\mathscr{B},
\end{aligned} \tag{A.2}$$

where $I \in \mathbb{R}^{n\times n}$ is an identity matrix.

Thus,

$$H\left(arg\right) - \widetilde{H}\left(arg\right) = \mathscr{C}^T \mathscr{A}(arg)^{-1} \mathscr{M}(arg) \mathscr{A}(arg)^{-1} \mathscr{B}, \tag{A.3}$$

where $\mathscr{M}(arg) = Z\Big(I + \mathscr{A}(arg)^{-1} Z\Big)^{-1}$. Taking H_2-norm on both sides of (A.3) and squaring we get

$$\left\| H\left(arg\right) - \widetilde{H}\left(arg\right) \right\|_{H_2}^2 = \frac{1}{2\pi} \int_{-\infty}^{\infty} \left\| \mathscr{C}^T \mathscr{A}(i\omega)^{-1} \mathscr{M}(i\omega) \mathscr{A}(i\omega)^{-1} \mathscr{B} \right\|_f^2 d\omega.$$

From the compatible norm property (i.e., $\|AB\|_f \leq \|A\|_f \|B\|_2$) we get

$$\left\| H\left(arg\right) - \widetilde{H}\left(arg\right) \right\|_{H_2}^2 \leq \frac{1}{2\pi} \int_{-\infty}^{\infty} \left\| \mathscr{C}^T \mathscr{A}(i\omega)^{-1} \right\|_f^2 \left\| \mathscr{M}(i\omega) \right\|_2^2 \left\| \mathscr{A}(i\omega)^{-1} \mathscr{B} \right\|_2^2 d\omega. \tag{A.4}$$

According to the mean value theorem for integrals, if $f(x)$ and $g(x)$ are continuous functions and $g(x)$ is not changing sign for any x, then

$$\textstyle\int_{-\infty}^{\infty} f(x)g(x)dx \leq \max\limits_{c\in\mathbb{R}} g(c) \int_{-\infty}^{\infty} f(x)dx.$$

Using this property in (A.4) we get

$$\begin{aligned} \left\| H\left(arg\right) - \widetilde{H}\left(arg\right) \right\|_{H_2}^2 \leq & \max_{\omega\in\mathbb{R}} \left\| \mathscr{M}(i\omega) \right\|_2^2 \cdot \max_{\omega\in\mathbb{R}} \left\| \mathscr{A}(i\omega)^{-1} \mathscr{B} \right\|_2^2 \\ & \frac{1}{2\pi} \int_{-\infty}^{\infty} \left\| \mathscr{C}^T \mathscr{A}(i\omega)^{-1} \right\|_f^2 d\omega \end{aligned} \tag{A.5}$$

or

$$\left\| H\left(arg\right) - \widetilde{H}\left(arg\right) \right\|_{H_2}^2 \leq \left\| \mathscr{M}(i\omega) \right\|_{H_\infty}^2 \left\| \mathscr{A}(i\omega)^{-1} \mathscr{B} \right\|_{H_\infty}^2 \left\| \mathscr{C}^T \mathscr{A}(i\omega)^{-1} \right\|_{H_2}^2.$$

Here, second and last terms in the right hand side are independent of perturbation Z. Let's take a look at the term $\left\| \mathscr{M}(arg) \right\|_{H_\infty}^2$. $\mathscr{M}(arg)$ can be rewritten as

$$\mathscr{M}(arg) = \big(I - \mathscr{M}(arg) \mathscr{A}(arg)^{-1}\big) Z.$$

Hence,

$$\begin{aligned} \left\| \mathscr{M}\left(arg\right) \right\|_{H_\infty} = \max_{\omega\in\mathbb{R}} \left\| \mathscr{M}\left(i\omega\right) \right\| &\leq \max_{\omega\in\mathbb{R}} \left\| I - \mathscr{M}\left(i\omega\right) \mathscr{A}(i\omega)^{-1} \right\| \left\| Z \right\|_2 \\ &\leq \left(1 + \left\| \mathscr{M}\left(arg\right) \right\|_{H_\infty} \left\| \mathscr{A}(arg)^{-1} \right\|_{H_\infty}\right) \left\| Z \right\|_2. \end{aligned}$$

Thus,

$$\|\mathscr{M}\,(arg)\|_{H_\infty} \leq \frac{\|Z\|_2}{\left(1 - \|\mathscr{A}(arg)^{-1}\|_{H_\infty}\,\|Z\|_2\right)}. \tag{A.6}$$

Substituting (A.6) in (A.5), we get

$$\|H(arg) - \tilde{H}(arg)\|_{H_2} \leq \frac{\|\mathscr{A}(arg)^{-1}B\|_{H_\infty}\|\mathscr{C}^T\mathscr{A}(arg)^{-1}\|_{H_2}}{1 - \|\mathscr{A}(arg)^{-1}\|_{H_\infty}\|Z\|_2}\|Z\|_2, \tag{A.7}$$

If $\|Z\|_2 < 1$ and $\|\mathscr{A}(arg)^{-1}\|_{H_\infty} < 1$, then

$$\left\|H\,(arg) - \widetilde{H}\,(arg)\right\|_{H_2} = \bigcirc\big(\|Z\|_2\big).$$

Hence, AIRGA and RPMOR satisfies the second condition of backward stability with respect to the inexact linear solves, i.e. (2.9).

Appendix B

Reusing Preconditioners for PMOR-L

PMOR-L [54] is a Loewner framework based model order reduction algorithm for all orders of linear parametric dynamical systems. One important step here is computing a matrix called the Loewner matrix, which further requires computation of the transfer function of the dynamical system. This function for (1.2), with $B = B(p_j)$ and $C = C(p_j)$ is given by

$$H(s_k, p_j) = C(p_j)^T A(s_k, p_j)^{-1} B(p_j) \text{ for } k = 1, \ldots, v, \text{ and } j = 1, \ldots, w, \tag{B.1}$$

where

$$A(s_k, p_j) = \left(s_k D(p_j) + K(p_j)\right), \tag{B.2}$$

with $A(\cdot), D(\cdot), K(\cdot) \in \mathscr{R}^{n \times n}$ and $C(\cdot),\ B(\cdot) \in \mathscr{R}^{n \times 1}$. The variables s_k and p_j are the frequency variables and the parameters, respectively. To compute the transfer function $H(s_k, p_j)$, one needs to solve sequences of linear systems as

$$\begin{cases} A(s_1,p_1)x(11) = B(p_1), \\ A(s_2,p_1)x(21) = B(p_1), \\ \quad \vdots \qquad\qquad \vdots \\ A(s_k,p_1)x(k1) = B(p_1), \end{cases} \quad \cdots \quad \begin{cases} A(s_1,p_j)x(1j) = B(p_j), \\ A(s_2,p_j)x(2j) = B(p_j), \\ \quad \vdots \qquad\qquad \vdots \\ A(s_k,p_j)x(kj) = B(p_j). \end{cases} \tag{B.3}$$

Next, we look at cheap updates for the above sequence of linear systems. Here, we can express the relation between the matrices of the sequence of linear systems by the following two ways: *First*, by capturing the changes in frequency variables (s_k), and *second*, by capturing the changes in parameters (p_j). In general, savings in computation time are less in the *first* case since frequency variables change more rapidly than parameters. The reason for this is, the frequency range is often not known to the modeler, while the range of parameters is more easily quantifiable from the application.

We now discuss the *first* way. Since frequency variables change rapidly, the consecutive matrices here are more closer to each other than the non-consecutive ones. In other words, this is equivalent to the preconditioner updates for non-parametric model order reduction. Hence, the second approach from Table 4.1 is preferred. Let $A(s_{k-1},p_1)$ and $A(s_k,p_1)$ be two coefficient matrices for parameter p_1 and different values of frequency variables s_{k-1} and s_k, respectively. Expressing $A(s_k,p_1)$ in-terms of $A(s_{k-1},p_1)$ we get

$$A(s_k,p_1) = A(s_{k-1},p_1)\left[I + (s_k - s_{k-1})A(s_{k-1},p_1)^{-1}D(p_1)\right]. \tag{B.4}$$

Let $P_{(k-1)1}$ be a good initial SPAI preconditioner for $A(s_{k-1},p_1)$. We want to compute a good SPAI preconditioner P_{k1} corresponding to $A(s_k,p_1)$. We can do this by enforcing $A(s_{k-1},p_1)P_{(k-1)1} = A(s_k,p_1)P_{k1}$. Rewriting this equation by using (B.4)

we get

$$\begin{aligned} A(s_{k-1},p_1)P_{(k-1)1} &= A(s_{k-1},p_1)\left[I+(s_k-s_{k-1})A(s_{k-1},p_1)^{-1}D(p_1)\right] \\ &\quad \left[I+(s_k-s_{k-1})A(s_{k-1},p_1)^{-1}D(p_1)\right]^{-1}P_{(k-1)1}, \\ &= A(s_k,p_1)P_{k1}, \end{aligned}$$

where $P_{k1} = \left[I+(s_k-s_{k-1})A(s_{k-1},p_1)^{-1}D(p_1)\right]^{-1}P_{(k-1)1}$. Let $Q_{k1} = \left[I+(s_k-s_{k-1})A(s_{k-1},p_1)^{-1}D(p_1)\right]^{-1}$, then the above implies $A(s_{k-1},p_1)P_{(k-1)1} = A(s_k,p_1)Q_{k1}P_{(k-1)1}$. Thus, instead of computing a preconditioner by minimizing $\|I-A(s_k,p_1)P_{k1}\|_f^2$ with respect to P_{k1}, we solve a simpler problem given below

$$\min_{Q_{k1}} \|A(s_{k-1},p_1)-A(s_k,p_1)Q_{k1}\|_f^2$$

with $P_{k1} = Q_{k1}P_{(k-1)1}$. Similarly, for any parameter p_j (for $j=2,\ldots,w$), we solve for Q_{kj} from

$$\min_{Q_{kj}} \|A(s_{k-1},p_j)-A(s_k,p_j)Q_{kj}\|_f^2,$$

leading to $P_{kj} = Q_{kj}P_{(k-1)j}$.

Next, we discuss the *second* case. The preconditioner update technique, as explained in the dissertation for RPMOR is directly applicable here (because of slowly changing parameters). Hence, the first approach from Table 4.1 is preferred. Here, unless we know how the matrices D and K from (3.1) depend on p_j, we are unable to express $A(s_k,p_j)$ in-terms of $A(s_k,p_1)$. However, this can be easily worked out once the input model is known. Therefore, we give this derivation for a commonly used example from [54] (the paper that proposed PMOR-L). The linear system matrices that arise here have the following form:

$$D(p_j) = I,\ K(p_j) = diag\left(K_1(p_j), K_2, K_3, K_4\right) \text{ for } j=1,\ldots w,$$

where $K_1(p_j) = \begin{bmatrix} -1 & p_j \\ -p_j & -1 \end{bmatrix}$, $K_2 = \begin{bmatrix} -1 & 200 \\ -200 & -1 \end{bmatrix}$, $K_3 = \begin{bmatrix} -1 & 400 \\ -400 & -1 \end{bmatrix}$, and $K_4 =$

$diag\,(1, 2, \ldots 1000)$. Thus, from (B.2) we have

$$A(s_k, p_j) = \; s_k I + \begin{bmatrix} K_1(p_j) & & & \\ & K_2 & & \\ & & K_3 & \\ & & & K_4 \end{bmatrix}.$$

Expressing $A(s_k, p_j)$ in-terms of $A(s_k, p_1)$ we get

$$\begin{aligned} A(s_k, p_j) &= A(s_k, p_1) + \begin{bmatrix} K_1(p_j) - K_1(p_1) & & \\ & 0 & \\ & & \ddots \\ & & & 0 \end{bmatrix} \\ &= A(s_k, p_1) + diag\left((K_1(p_j) - K_1(p_1)), 0, \ldots, 0\right). \\ &= A(s_k, p_1)\left[I + A(s_k, p_1)^{-1} diag\left((K_1(p_j) - K_1(p_1)), 0, \ldots, 0\right)\right]. \end{aligned} \tag{B.5}$$

As earlier, we want to enforce $A(s_k, p_1)P_{k1} = A(s_k, p_j)P_{kj}$, where P_{k1}, P_{kj} are good SPAI preconditioners corresponding to the two respective matrices and P_{k1} is already computed. However, by using (B.5), we define $Q_{kj} = [I + A(s_k, p_1)^{-1} diag\left((K_1(p_j) - K_1(p_1)), 0, \ldots, 0\right)]^{-1}$, and enforce $A(s_k, p_1)P_{k1} = A(s_k, p_j)Q_{kj}P_{k1}$. Thus, instead of minimizing $\|I - A(s_k, p_j)P_{kj}\|_f^2$ with respect to P_{kj}, we solve a simpler problem

$$\min_{Q_{kj}} \|A(s_k, p_1) - A(s_k, p_j)Q_{kj}\|_f^2$$

to get $P_{kj} = Q_{kj}P_{k1}$.

Appendix C

Reusing Preconditioners for BIRKA

Bilinear Iterative Rational Krylov Algorithm (BIRKA) [81] is a Petrov-Galerkin projection based algorithm for MOR of the bilinear non-parametric first-order dynamical systems, which for the Multiple Input Multiple Output (MIMO) case are represented as

$$\begin{aligned} \dot{x}(t) &= Kx(t) + \sum_{j=1}^{m} N_j x(t) u_j(t) + Bu(t), \\ y(t) &= C^T x(t), \end{aligned} \tag{C.1}$$

where $K,\ N_j \in \mathbb{R}^{n\times n},\ B \in \mathbb{R}^{n\times m},\ C \in \mathbb{R}^{n\times q}$, and $u = [u_1,\ u_2,\ \ldots, u_m] \in \mathbb{R}^m$. Let columns of $V, W \in \mathbb{R}^{n\times r}$ span two r-dimension subspaces (where, as earlier, $r \ll n$). In principle, the Petrov-Galerkin projection method involves the steps below.

- Approximating the reduced state vector $\hat{x}(t)$ using V as $x(t) \approx V\hat{x}(t)$ leads to

$$V\dot{\hat{x}}(t) - KV\hat{x}(t) - \sum_{j=1}^{m} N_j V\hat{x}(t) u_j(t) - Bu(t) = r(t),$$

$$\hat{y}(t) = C^T V\hat{x}(t),$$

  where $r(t)$ is the residual after projection.

- Enforcing the residual $r(t)$ to be orthogonal to W or $W^T r(t) = 0$ leads to the

reduced system given by

$$\dot{\hat{x}}(t) - \hat{K}\hat{x}(t) - \sum_{j=1}^{m} \hat{N}_j \hat{x}(t) u_j(t) - \hat{B}u(t) = 0,$$

$$\hat{y}(t) = \hat{C}^T \hat{x}(t),$$

where $\hat{K} = (W^T V)^{-1} W^T K V$, $\hat{N}_j = (W^T V)^{-1} W^T N_j V$, $\hat{B} = (W^T V)^{-1} W^T B$, $\hat{C}^T = C^T V$, and $(W^T V)^{-1}$ is assumed to be invertible. Here, V and W are computed by using interpolation, where the original system transfer function and its derivative are respectively matched with the reduced system transfer function and its derivative at a set of points. We briefly summarize BIRKA in Algorithm 4, where again, only parts related to solving linear systems are listed.

Algorithm 4 : BIRKA [81]

Input $K,\ N_1,\ \ldots,\ N_m,\ B,\ C$; error tolerance (ϵ_3); and initial guess of the reduced system $\check{K},\ \check{N}_1,\ \ldots,\ \check{N}_m,\ \check{F},\ \check{C}$

Output $\hat{K},\ \hat{N}_1,\ \ldots,\ \hat{N}_m,\ \hat{B},$ and $\hat{C}$

1: $z = 1$

2: **while** relative change in eigenvalues of $\check{K}$ is greater than equal to the ϵ_3 **do**

3: $R\Lambda R^{-1} = \check{K},\ \check{\check{B}} = \check{B}^T R^{-T},\ \check{\check{C}} = \check{C}R,\ \check{\check{N}}_j = R^T \check{N}_j R^{-T}$
for $j = 1,\ \ldots,\ m$

4: $vec\,(V) = \left(-\Lambda \otimes I_n - I_r \otimes K - \sum_{j=1}^{m} \check{\check{N}}_j^T \otimes N_j \right)^{-1} \left(\check{\check{B}}^T \otimes B \right)\ vec(I_m)$

5: $vec\,(W) = \left(-\Lambda \otimes I_n - I_r \otimes K^T - \sum_{j=1}^{m} \check{\check{N}}_j \otimes N_j^T \right)^{-1} \left(\check{\check{C}}^T \otimes C^T \right)\ vec(I_q)$

6: $V = orth\,(V)\,,\ W = orth\,(W)$

7: $\check{K} = (W^T V)^{-1} W^T K V,\ \check{N}_j = \left(W^T V\right)^{-1} W^T N_j V,\ \check{B} = \left(W^T V\right)^{-1} W^T B,$
$\check{C} = CV$

8: $z = z + 1$

9: **end while**

10: $\hat{K} = \check{K},\ \hat{N}_j = \check{N}_j,\ \hat{B} = \check{B},$ and $\hat{C} = \check{C}$

In the Algorithm 4, we solve linear systems of equations at lines 4 and 5. We first apply our proposed theory of reusing preconditioners to line 4, which is given as

$$vec\,(V) = \left(-\Lambda \otimes I_n - I_r \otimes K - \sum_{j=1}^{m} \check{\check{N}}_j^T \otimes N_j\right)^{-1} \left(\check{\check{B}}^T \otimes B\right)\ vec(I_m).$$

Here, Λ is a diagonal matrix comprising of interpolation points, which is updated at the start of the `while` loop at line 2. For ease of explanation, we take $j = 1$ here. Similar steps can be executed for $j = 2,\ \ldots,\ m$. Let $A_{z-1} = -\Lambda_{z-1} \otimes I_n - I_r \otimes K - \left(\check{\check{N}}_1^T\right)_{z-1} \otimes N_1$ and $A_z = -\Lambda_z \otimes I_n - I_r \otimes K - \left(\check{\check{N}}_1^T\right)_z \otimes N_1$ be the coefficient matrices corresponding to Λ_{z-1} and Λ_z, respectively (for $z = 1,\ \ldots,\ \mathfrak{z}$ (until covergence)). Expressing A_z in terms of A_{z-1}, we get

$$\begin{aligned} A_z = A_{z-1}\Bigg(& I_{nr} + A_{z-1}^{-1}(-\Lambda_z \otimes I_n) + A_{z-1}^{-1}(\Lambda_{z-1} \otimes I_n) + \\ & A_{z-1}^{-1}\left(-\left(\check{\check{N}}_1^T\right)_z \otimes N_1\right) + A_{z-1}^{-1}\left(\left(\check{\check{N}}_1^T\right)_{z-1} \otimes N_1\right)\Bigg), \end{aligned}$$

where $I_{nr} \in \mathbb{R}^{n\cdot r \times n\cdot r}$ is the Identity matrix. If we define
$Q_z = \Bigg(I_{nr} + A_{z-1}^{-1}(-\Lambda_z \otimes I_n) + A_{z-1}^{-1}(\Lambda_{z-1} \otimes I_n) +$
$A_{z-1}^{-1}\left(-\left(\check{\check{N}}_1^T\right)_z \otimes N_1\right) + A_{z-1}^{-1}\left(\left(\check{\check{N}}_1^T\right)_{z-1} \otimes N_1\right)\Bigg)^{-1}$, then above is equivalent to

$$A_z = A_{z-1}Q_z^{-1}. \tag{C.2}$$

Now, we enforce

$$A_{z-1}P_{z-1} = A_zP_z. \tag{C.3}$$

Using (C.2), instead we enforce

$$A_{z-1}P_{z-1} = A_{z-1}Q_z^{-1}P_z.$$

If we take $P_z = Q_zP_{z-1}$, then we eventually enforce

$$A_{z-1}P_{z-1} = A_{z-1}Q_z^{-1}Q_zP_{z-1},$$

which is true.

Thus, instead of solving for P_z by enforcing (C.3), which is harder to solve, we obtain the preconditioner at the z^{th} iteration ($P_z = Q_z P_{z-1}$) by enforcing

$$A_{z-1} P_{z-1} = A_z Q_z P_{z-1},$$

which is more easily solvable. The remaining derivation here is same as earlier. We reuse preconditioners at line 5 similarly.

Appendix D

Reusing Preconditioners for QB-IHOMM

Quadratic Bilinear-Implicit Higher Order Moment Matching (QB-IHOMM) [82] is a Petrov-Galerkin projection based algorithm for MOR of the quadratic-bilinear non-parametric first-order dynamical systems, which for the SISO case are represented as[1]

$$\begin{aligned} D\dot{x}(t) &= Kx(t) + Nx(t)u(t) + H\left(x(t) \otimes x(t)\right) + Bu(t), \\ y(t) &= C^T x(t), \end{aligned} \tag{D.1}$$

where $D,\ K,\ N \in \mathbb{R}^{n\times n},\ H \in \mathbb{R}^{n\times n^2},\ B \in \mathbb{R}^{n\times 1},\ C \in \mathbb{R}^{n\times 1}$. Let columns of $V,\ W \in \mathbb{R}^{n\times r}$ span two r-dimension subspaces (where as earlier, $r \ll n$). In principle, the Petrov-Galerkin projection method involves the steps below.

- As before, approximating the reduced state vector $\hat{x}(t)$ using V as $x(t) \approx V\hat{x}(t)$ leads to

$$\begin{aligned} DV\dot{\hat{x}}(t) - KV\hat{x}(t) - NV\hat{x}(t)u(t) - H\left(V\hat{x}(t) \otimes V\hat{x}(t)\right) - Bu(t) &= r(t), \\ y(t) &= C^T V\hat{x}(t), \end{aligned}$$

  where $r(t)$ is the residual after projection.

[1] A variant of BIRKA for MOR of the quadratic-bilinear first-order dynamical systems also exists. Preconditioned iterative solves and reusing preconditioners can be applied here as done for BIRKA. Hence, we focus on QB-IHOMM that has been developed for the SISO case only.

- Enforcing the residual $r(t)$ to be orthogonal to W or $W^T r(t) = 0$ leads to the reduced system given by

$$\hat{D}\dot{\hat{x}}(t) - \hat{K}\hat{x}(t) - \hat{N}\hat{x}(t)u(t) - \hat{H}\left(\hat{x}(t) \otimes \hat{x}(t)\right) - \hat{B}u(t) = 0,$$
$$y(t) = \hat{C}^T\hat{x}(t),$$

where $\hat{D} = W^T DV$, $\hat{K} = W^T KV$, $\hat{N} = W^T NV$, $\hat{H} = W^T H(V \otimes V)$, $\hat{B} = W^T B$, $\hat{C}^T = C^T V$. Here, V and W are computed by matching the moments of the original system transfer function and the reduced system transfer function. We briefly summarize QB-IHOMM in Algorithm 5, where as earlier, only parts related to solving linear systems are listed. Here, as in [82], the computation is done with the first two regular transfer function terms.

In the Algorithm 5, we solve linear systems of equations at line 4 and 10. Again, we first apply our proposed theory of reusing preconditioners to line 4, which is given as

$$X_j(\sigma_i) = [(\sigma_i D - K)^{-1} D]^j (\sigma_i D - K)^{-1} B,$$
$$\text{for } j = 1,\ \ldots,\ P + Q \text{ and } i = 1,\ \ldots,\ \ell.$$

Let $A_{i-1} = \sigma_{i-1} D - K$ and $A_i = \sigma_i D - K$ be the two coefficient matrices for different interpolation points σ_{i-1} and σ_i, respectively (for $i = 1,\ \ldots,\ \ell$). Expressing A_i in terms of A_{i-1}, we get

$$A_i = A_{i-1}(I + (\sigma_i - \sigma_{i-1})A_{i-1}^{-1} D).$$

If we define $Q_i = (I + (\sigma_i - \sigma_{i-1})A_{i-1}^{-1} D)^{-1}$, then above is equivalent to

$$A_i = A_{i-1} Q_i^{-1}.$$

As earlier, instead of obtaining P_i by enforcing

$$A_{i-1} P_{i-1} = A_i P_i,$$

which is harder to solve, we obtain the preconditioner at the i^{th} iteration ($P_i = Q_i P_{i-1}$) by enforcing

$$A_{i-1} P_{i-1} = A_i Q_i P_{i-1},$$

Algorithm 5 : QB-IHOMM [82]

Input: $D,\ K,\ N,\ H,\ B,\ C$; interpolation points $\sigma_i \in \mathbb{C}$ for $i = 1,\ \ldots,\ \ell$; higher orders moments numbers $P, Q \in \mathbb{N}$

Output: $\hat{D},\ \hat{K},\ \hat{N},\ \hat{H},\ \hat{B},\ \hat{C}$

1: $V = [\,]\,,\ \ W = [\,]$
2: **for** $j = 0,\ \ldots,\ P + Q$ **do**
3: **for** $i = 1,\ \ldots,\ \ell$ **do**
4: $X_j(\sigma_i) = [(\sigma_i D - K)^{-1} D]^j (\sigma_i D - K)^{-1} B$
5: $V = [V\ \ X_j(\sigma_i)]$
6: **end for**
7: **end for**
8: **for** $j = 0,\ \ldots,\ Q$ **do**
9: **for** $i = 1,\ \ldots,\ \ell$ **do**
10: $X_j(2\sigma_i)^T = [(2\sigma_i D - K)^{-T} D^T]^j (2\sigma_i D - K)^{-T} C^T$
11: $W = \left[W\ \ X_j(2\sigma_i)^T\right]$
12: **end for**
13: **end for**
14: $U = orth([V\ W])$
15: Construct the reduced system as

$\hat{D} = U^T D U,\ \hat{K} = U^T K U,\ \hat{N} = U^T N U,\ \hat{H} = U^T H (U \otimes U),\ \hat{B} = U^T B,\ \hat{C}^T = C^T U.$

which is more easily solvable. Again, here also, the remaining derivation is same as earlier. We reuse preconditioners at line 10 similarly.