

RICE UNIVERSITY

**A Parallel-In-Time Gradient-Type Method  
For Optimal Control Problems**

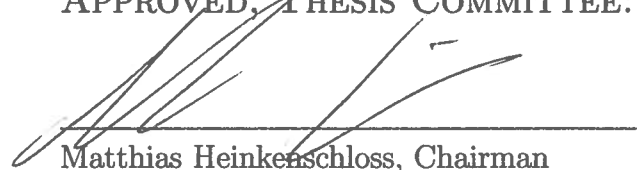
by

**Xiaodi Deng**

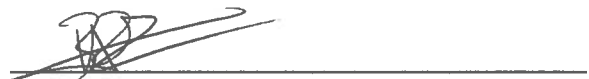
A THESIS SUBMITTED  
IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE

**Doctor of Philosophy**

APPROVED, THESIS COMMITTEE:



Matthias Heinkenschloss, Chairman  
Noah G. Harding Chair and Professor of  
Computational and Applied Mathematics



Beatrice M. Riviere  
Noah G. Harding Chair and Professor of  
Computational and Applied Mathematics



John Mellor-Crummey  
Professor of Computer Science and of  
Electrical and Computer Engineering

HOUSTON, TEXAS

APRIL, 2017

## Abstract

# A Parallel-In-Time Gradient-Type Method For Optimal Control Problems

by

Xiaodi Deng

This thesis proposes and analyzes a new parallel-in-time gradient-type method for time-dependent optimal control problems. When the classical gradient method is applied to such problems, each iteration requires the forward solution of the state equations followed by the backward solution of the adjoint equations before the gradient can be computed and control can be updated. The solution of the state/adjoint equations is computationally expensive and consumes most of the computation time. The proposed parallel-in-time gradient-type method introduces parallelism by splitting the time domain into  $N$  subdomains and executes the forward and backward computation in each time subdomain in parallel using state and adjoint variables at time subdomain boundaries from the last optimization iteration as initial values. The proposed method is generalized to allow different time domain partitions for forward/backward computations and overlapping time subdomains.

Convergence analyses for the parallel-in-time gradient-type method applied to discrete-time optimal control problems are established. For linear-quadratic problems, the method is interpreted as a multiple-part splitting method and convergence is proven by analyzing the corresponding iteration matrix. In addition, the connection of the new method to the multiple shooting reformulation of the problem is revealed

and an alternative convergence proof based on this connection is established. For general non-linear problems, the new method is combined with metric projection to handle bound constraints on the controls and convergence of the method is proven.

Numerically, the parallel-in-time gradient-type method is applied to linear-quadratic optimal control problems and to a well-rate optimization problem governed by a system of highly non-linear partial differential equations. For linear-quadratic problems, the method exhibits strong scaling with up to 50 cores. The parallelization in time is on top of the existing parallelization in space to solve the state/adjoint equations. This is exploited in the well-rate optimization problem. If the existing parallelism in space scales well up to  $K$  processors, the addition of time domain decomposition by the proposed method scales well up to  $K \times N$  processors for small to moderate number  $N$  of time subdomains.

## Acknowledgements

I would like to express my gratitude to my advisor Professor Dr. Matthias Heinkenschloss for his continuous support of my Ph.D. research, guidance, and patience.

I also thank the rest of my committee members: Professor Dr. Beatrice Riviere and Professor Dr. John Mellor-Crummey for spending their precious time on my thesis proposal and defense.

I am grateful to my CAAM friends from the bottom of my heart for their company and friendship during these years.

My special thanks also goes to my former undergraduate school friends, Dr. Qu Lu of University of Illinois at Urbana-Champaign, Dr. Lei Huang of National University of Singapore, and Dr. Zheng Fang of University of Illinois at Chicago for many interesting discussions on related research topics.

I really appreciate the great help from CAAM staff Daria Lawrence, Brenda Aune, Ivy Gonzalez, Latrece McKinney, and Fran Moshiri. Their compassionate first-class support ensures that I can always focus on my research.

I gratefully acknowledge the sponsors of my research. This work was supported in part by a sponsored research agreement with the ExxonMobil Upstream Research Company and by a 2014 OG HPC Graduate Fellowship administered through the Rice University Ken Kennedy Institute. Part of the computation in this thesis was performed on the Rice University DAVinCI cluster, which was supported in part by the Data Analysis and Visualization Cyberinfrastructure funded by NSF under grant OCI-0959097 and Rice University.

Last but not least, I would like to thank C.C. for her understanding, cheering up, and good temper even when I was often absent-minded at dinner times during the last semester of graduate school. I would like to thank my family for their constant love and unconditional support through all these years from my stepping into elementary

school as a kid at the age of seven to eventually making my way out of graduate school the day before my thirtieth birthday on the other side of the ocean. It has been a long time and the world has changed a lot. But I am so fortunate to be a part of my family that has never changed.

# Contents

<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>List of Figures</b>	<b>x</b>
<b>List of Tables</b>	<b>xxi</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Literature Review</b>	<b>8</b>
2.1 Parallel-In-Time Simulation . . . . .	8
2.2 Parallel-In-Time Optimization . . . . .	10
2.3 Reservoir Optimization . . . . .	17
<b>3 Parallel-In-Time Gradient-Type Method in Linear-Quadratic Problems</b>	<b>25</b>
3.1 Gradient Method . . . . .	26
3.2 Derivation of the Parallel-In-Time Gradient-Type Method . . . . .	30
3.3 Interpretation as a $(2N - 1)$ -Part Iteration Scheme . . . . .	31
3.4 A Generalized Framework for Parallelism . . . . .	38
3.4.1 Forward/Backward Computation Subdomains . . . . .	39
3.4.2 Aggregated State/Adjoint Variables . . . . .	41

3.4.3	The Algorithm . . . . .	43
3.4.4	Subdomain Initial/Terminal Processor Rank Function . . . . .	48
3.4.5	$D^F$ and $D^B$ . . . . .	49
3.4.6	Interpretation as a $(D^F + D^B + 1)$ -Part Iteration Scheme . . . . .	52
3.5	Convergence Proof for the Linear-Quadratic Problems . . . . .	58
3.6	Numerical Examples . . . . .	64
3.6.1	1D Boundary Control: Step Size, Non-Monotonic Convergence . . . . .	66
3.6.2	3D Distributed Control: Parallel Implementation, Strong Scaling . . . . .	72
3.6.3	1D Distributed Control: Generalized Algorithm . . . . .	81
3.7	Summary . . . . .	85
<b>4</b>	<b>Multiple-Shooting Formulations</b> . . . . .	<b>87</b>
4.1	Continuous Time Multiple-Shooting Formulation . . . . .	88
4.2	Discrete Time Multiple-Shooting Formulation . . . . .	90
4.3	Convergence Proof by Multiple-Shooting Formulation . . . . .	94
4.4	Summary . . . . .	111
<b>5</b>	<b>Parallel-In-Time Gradient-Type Method in Nonlinear Problems</b> . . . . .	<b>112</b>
5.1	Metric Projection Properties . . . . .	115
5.2	Convergence in Constrained Problem . . . . .	122
5.2.1	Gradient-Type Vector Error Bound . . . . .	125
5.2.2	Convergence of the Control Update . . . . .	135
5.2.3	Bound Iterate Around Optimal Control . . . . .	141
5.2.4	Convergence Theorems . . . . .	151
5.3	As a Perturbed Gradient Method: Iteration-Wise Monotonicity . . . . .	158
5.3.1	Iteration-Wise Gradient-Type Vector Error Bound . . . . .	158
5.3.2	Unconstrained Problem . . . . .	161
5.3.3	Constrained Problem . . . . .	168
5.4	Summary . . . . .	175

<b>6 Reservoir Optimization</b>	<b>178</b>
6.1 Introduction . . . . .	178
6.2 Optimization . . . . .	180
6.2.1 Example Objective Function . . . . .	180
6.2.2 Discretized Form of the Optimization Problem . . . . .	181
6.2.3 Classical Gradient Method . . . . .	182
6.2.4 Adjoint Equation Approach for Gradient Computation . . . . .	183
6.2.5 Projected Gradient Method . . . . .	183
6.3 Numerical Examples . . . . .	186
6.3.1 Classical Gradient Method . . . . .	186
6.3.2 Parallel-In-Time Gradient-Type Method . . . . .	190
6.3.3 A Heuristic Watchdog Type Parallel Algorithm . . . . .	203
6.4 Summary . . . . .	211
<b>7 Conclusions and Outlook</b>	<b>214</b>
<b>Bibliography</b>	<b>218</b>
<b>Appendices</b>	<b>229</b>
<b>A On the Multiple-Shooting Formulation</b>	<b>230</b>
A.1 On the Convergence of the Projected Parallel Gradient-Type Method	230
A.2 Behaviour of Spectral Radius $\rho(\mathbf{I} - \mathbf{W}(\alpha))$ . . . . .	233
A.3 A Parallel-In-Time Krylov Subspace Based Solver . . . . .	236
<b>B Reservoir Simulation</b>	<b>244</b>
B.1 Basic Physics . . . . .	244
B.2 Numerical Scheme . . . . .	249
B.2.1 Solving Pressure Equation . . . . .	251
B.2.2 Solving Saturation . . . . .	253



B.3	Trilinos Implementation . . . . .	256
<b>C</b>	<b>Parallel-In-Time Gradient-Type Method Implementation Details In Reservoir Optimization</b>	<b>258</b>
C.1	Algorithms For Explicit/Implicit Formulation of State Equations . . .	259
C.1.1	Gradient Computation with Implicit State Equations . . . . .	260
C.1.2	Gradient Computation with Explicit State Equations . . . . .	260
C.1.3	Algorithm with Explicit State Equations . . . . .	262
C.1.4	Algorithm with Implicit State Equations . . . . .	262
C.2	Comparison of Algorithms . . . . .	264
C.3	Application to the Reservoir Problem . . . . .	269
<b>D</b>	<b>A Test on Adjusting Time Subdomains for Load Balance</b>	<b>271</b>

# List of Figures

1.1	Workflow of the classical gradient method . . . . .	2
1.2	Workflow of the parallel-in-time gradient-type method . . . . .	3
3.1	Illustration of the positions of ‘1’s in $\mathbf{I}_0, \mathbf{I}_{-1}$ and $\mathbf{I}_{-2}$ in an example where the state dimension is $n_y = 2$ , the $K = 10$ time steps are split into $N = 3$ subdomains with $K_0 = 0, K_1 = 3, K_2 = 6, K_3 = 10$ . . . . .	36
3.2	Illustration of Generalized Parallel-In-Time Gradient-Type Method. Blue(red) arrows represent time subdomains for parallel forward(backward) computation. The regions in different processes shaded by the green dots are aggregated and collectively constitute the resulting state/adjoint on the whole time domain for use in subsequent computations. Its time subdomain representation is in Table 3.1. . . . .	38
3.3	Workflow of One Iteration of the Generalized Parallel-in-Time Gradient-Type Method Algorithm. Serial computation in central processor is negligible compared to the forward/backward computation happening in the parallel. . . . .	44
3.4	Illustration of the positions of ‘1’s in $\mathbf{I}_0^F, \mathbf{I}_{-1}^F, \mathbf{I}_{-2}^F$ , and $\mathbf{I}_{-3}^F$ in an example where the state dimension is $n_y = 2$ and $K = 12$ . The time domain splitting pattern is illustrated in Figure 3.2 and represented in Table 3.1. Compare with Figure 3.1 for the original parallel-in-time gradient-type method. . . . .	55

3.5	Illustration of the positions of ‘1’s in $\mathbf{I}_0^B$ and $\mathbf{I}_{-1}^B$ in an example where the state dimension is $n_y = 2$ and $K = 12$ . The time domain splitting pattern is illustrated in Figure 3.2 and represented in Table 3.1. Compare with Figure 3.1 for the original parallel-in-time gradient-type method. . . . .	56
3.6	Eigenvalues of $\tilde{\mathbf{C}}(\alpha)$ . For sufficiently small $\alpha > 0$ the eigenvalues $\lambda_\alpha$ of the companion matrix $\tilde{\mathbf{C}}(\alpha)$ , defined in (3.5.6) with $\sum_{i=0}^n \mathbf{M}_i$ Hermitian and positive definite, lie in the union of a small open ball around 0 and of the intersection of a small open ball around 1 and the open cone $C_k$ , indicated by the dark shaded regions. In particular, all eigenvalues are inside the unit disk. . . . .	61
3.7	Speed-ups of the parallel-in-time gradient-type method applied to the 1D optimal control problem. If $I(N)$ is the number of iterations needed by the parallel-in-time method with $N$ subdomains and cores, the speed-up curve shows $I(1)/(I(N)/N)$ . . . . .	70
3.8	Optimal step-sizes $\alpha$ for varying number $N$ of time subdomains for the 1D example problem. . . . .	71
3.9	Distance between the optimization control variable trajectory and the optimal control. In this example, 10 time subdomains with optimal step size 0.87 are used. Nonmonotonic convergence is observed and there is a periodic oscillation pattern. Red squares mark the iteration where oscillation reaches its peaks. In this 10 time subdomain case, the peaks happen every 5 iterations on average. See Figure 3.10 for control errors recorded on optimization trajectory on three consecutive peaks. . . . .	72

3.10 The control error, i.e. the difference between the control in the optimization iterations and the precomputed control in the optimal solution, at optimization iteration 44, 49, and 54. They are at the error peaks as depicted in Figure 3.9 as red boxes. Ten time subdomains with optimal step size 0.87 are used. . . . . 73

3.11 Speed-ups of the parallel-in-time gradient-type method for two examples problems with end time  $T = 8$  and  $K = 200$  time steps (red curves), and with end time  $T = 16$  and  $K = 400$  time steps (blue curves). For each case two speed-up curves are shown. If  $I(N)$  denotes the number of iterations needed by parallel-in-time method with  $N$  subdomains and cores, the speed-up curves indicated by ‘ $\dots\circ$ ’ and ‘ $\dots+$ ’ show  $I(1)/(I(N)/N)$ . If  $t(N)$  denotes the run time needed by parallel-in-time method with  $N$  subdomains and cores, the curves indicated by ‘ $-\square$ ’ and ‘ $-*$ ’ show  $t(1)/t(N)$ . Excellent speed-ups are obtained for up to  $N = 20$  time domains when  $T = 8$  and  $K = 200$  and for up to  $N = 50$  time domains when  $T = 16$  and  $K = 400$ . The fixed step size for both of the classical gradient method and the parallel-in-time gradient-type method is chosen by trials. The fixed step size with fastest convergence is used. Refer to Table 3.2 and Table 3.3 for details. . . . . 76

3.12 Trace graph of 1 second of running time, 8 cores, generated by the HPCTOOLKIT [ABF<sup>+</sup>10]. The horizontal axis is the time axis. Eight horizontal rows represent computation timing of eight cores. The top row is for the first time subdomain, etc. One optimization iteration consists of two major blocks, one purple and one brown. The purple blocks are for forward computation, the brown blocks are for backward computation. The smaller green and red blocks correspond to communication and error estimation timing respectively. . . . . 79

3.13	Trace graph of 1 second of running time, 16 cores, generated by the HPCTOOLKIT [ABF <sup>+</sup> 10]. The horizontal axis is the time axis. Sixteen horizontal rows represent computation timing of sixteen cores. The top row is for the first time subdomain, etc. One optimization iteration consists of two major blocks, one shown in purple and the other one shown in brown. The purple blocks are for forward computations and the brown blocks are for backward computation. The smaller green and red blocks correspond to communication and error estimation timing respectively. . . . .	80
3.14	Control Error History, Test Case 1 in Section 3.6.3.1. . . . .	82
3.15	Gradient-Type Vector Comparison, Test Case 1 in Section 3.6.3.1. . . . .	83
3.16	Control Error History, Test Case 2 in Section 3.6.3.2. . . . .	84
3.17	Gradient-Type Vector Comparison, Test Case 2 in Section 3.6.3.2. . . . .	85
4.1	Illustration of direct multiple-shooting optimization variables . . . . .	88
4.2	Plot of $\ (\mathbf{I} - \mathbf{W}(\alpha))^k\ _2$ against $k$ . In the legend, $\rho((\mathbf{I} - \mathbf{W}(\alpha)))$ is also appended. . . . .	109
4.3	Ratio $\ ((\hat{y}^{(j+1)})^T, (g^{(j+1)})^T, (\hat{p}^{(j+1)})^T)\  / \ ((\hat{y}^{(j)})^T, (g^{(j)})^T, (\hat{p}^{(j)})^T)\ $ against iteration index $j$ . Step size $\alpha = 0.01$ . Random initial $\hat{y}^{(0)}, g^{(0)}, \hat{p}^{(0)}$ is used. . . . .	110
5.1	Theorem Dependency Structure in Chapter 5 . . . . .	114
5.2	Illustration for Lemma 5.1.3, Lemma 5.1.4, and Lemma 5.1.5 on the relationship between the blue and red lengths and angles. . . . .	117
5.3	Illustration Lemma 5.1.7. The blue angle is no larger than $\pi/2$ . . . . .	122

5.4 Illustration of the first half of the Theorem 5.2.5 proof for the unconstrained case, i.e.,  $D$  is the whole space and the projection  $\mathcal{P}_D$  is trivial. Note that this is a two dimensional illustration of an arbitrary dimensional problem.  $g^{(i)}$ ,  $\nabla \hat{J}(u^{(i)})$ , and  $u^* - u^{(i)}$  do not necessarily lie in the same two dimensional plane. It is typically not true that  $|\theta_u - \tilde{\theta}_u| = \theta_d$ . 148

5.5 Illustration of the second half of the Theorem 5.2.5 proof for the unconstrained case, i.e.,  $D$  is the whole space and the projection  $\mathcal{P}_D$  is trivial. . . . . 150

6.1 Illustration of a projected gradient step described by Algorithm 15 . . 185

6.2 Location of injection and production wells. Production wells are indicated by crosses; injection are indicated by circles. Markers connected by solid lines are on the 30 feet layer; markers connected by dotted lines are on the 20 feet layer. . . . . 187

6.3 Initial well rates (left) and optimized well-rates (right) for the 25 injection well and 25 production wells. The color bars with negative well rates above the dashed line in the middle represent the 25 production wells; the color bars with positive well rates below the dashed line in the middle represent the 25 injection wells. The markers (crosses and circles) on the left of each plot are consistent with the markers in Figure 6.2. . . . . 188

6.4	Contributions of oil revenue (top blue dashed line), water treatment cost (dot-dash red line blow zero), and water injection cost (bottom green dotted line) to the NPV (solid black line) at the initial well-rate settings (left plot) and at the optimal well-rate settings (right plot), for the example problem in Section 6.3.1. Optimization uses the classical gradient method. The area below the black solid line is the NPV. The NPV for the initial well rates (left) is $6.857 \times 10^6$ ; the NPV for the optimized well rates (right) is $7.986 \times 10^6$ . A 16.5% improvement is achieved. The jumps in the curves are caused by the discontinuity in the piecewise constant wells rates. . . . .	189
6.5	Iteration history of the projected gradient method. After 20 iterations the optimal NPV is essentially reached and the remaining iterations are used to reduce the norm of the projected gradient below the required tolerance. . . . .	190
6.6	Contours at day 100, 300, and 500 (final time) of water saturations resulting from optimized well-rates. Only the bottom half of the reservoir (depth 20-40 feet) is shown to better display water saturations at the 20 feet level, where several of the wells are located. . . . .	191
6.7	2D reservoir example, water saturation at day 25. The blue and red squares mark the injection and production well respectively. . . . .	194
6.8	2D reservoir example, iteration history of gradient method and the parallel gradient-type method. The parallel gradient-type method uses 4 time subdomains. . . . .	194

- 6.9 Trace graph of about 11 optimization iterations, generated by HPC-TOOLKIT [ABF<sup>+</sup>10]. The horizontal axis is the time axis. Four horizontal rows represent computation timing of four cores. The top row is for the first time subdomain, etc. One optimization iteration consists of three major blocks of one purple, one green, and one brown. The purple blocks are for forward computation, the brown blocks are for backward computation, and the green blocks are for waiting. . . . . 195
- 6.10 2D reservoir example, optimized well rates at optimization iteration 200. 196
- 6.11 Location of injection and production wells. Circles and crosses are for injection and production wells respectively. . . . . 197
- 6.12 Contours at day 100 and 500 (final time) of water saturations resulting from optimized well-rates. Only the bottom half of the reservoir (depth 5-10 feet) is shown to better display water saturations at the 5 feet level, where all of the wells are located. . . . . 198
- 6.13 3D reservoir example, iteration history of gradient method and parallel gradient-type method. Gradient method uses 4 cores in the linear solvers and preconditioners in the forward/backward solves; parallel gradient-type method uses 4 time subdomains in each of which 4 cores are used in the forward/backward solves, totally,  $4 \times 4 = 16$  cores are computing simultaneously for the parallel-in-time gradient-method. Parallel gradient-type method is initialized by a full gradient sweep and, as a result, the first step takes as long as a serial gradient step. There are vertical red bars on the parallel gradient-type objective values. The other end of bars indicate the infeasible “objective value” computed, for research purpose, during the parallel method runtime using the state variable with jumps at subdomain boundaries. . . . . 199



6.14	Trace graph of about 10 optimization iterations with evenly split time subdomains, generated by HPC <code>TOOLKIT</code> [ABF <sup>+</sup> 10]. The horizontal axis is the time axis. From top to bottom, the 16 rows corresponding to 16 cores are grouped into 4 groups. The top 4 horizontal rows represent computation timing of the 4 cores performing parallel computing in the first time subdomain, etc. One optimization iteration consists of three major blocks of one purple, one brown, and one green block. Purple blocks are for forward computation, brown blocks are for backward computation, and green blocks are for waiting. . . . .	200
6.15	3D reservoir example, optimized well rates at iteration 50. . . . .	201
6.16	At parallel gradient-type method iteration 50, the 3 dashed lines mark the position of time subdomain boundaries. As a measure of discontinuity, the blue dots are computed by taking the norm of the difference of the states in the current time marching step and the state in the last time marching step. . . . .	202
6.17	At parallel gradient-type method iteration 160, the 3 dashed lines mark the position of time subdomain boundaries. As a measure of discontinuity, the blue dots are computed by taking the norm of the difference of the states in the current time marching step and the state in the last time marching step. . . . .	202
6.18	Speed-Up Estimation for Algorithm 16 . . . . .	207
6.19	Watchdog-Type Algorithm with Suitable Initial Step Size. Initial step size $\alpha = 0.03$ , inflation factor $\rho_{\text{inf}} = 1$ . Serial Gradient Method is checking objective decrease condition (Line 19 in Algorithm 16) in each step. The watchdog-type parallel-in-time gradient-type method with $I_{\text{inner}} = 10$ tests the decreasing condition every $I_{\text{inner}} = 10$ steps, marked by the big red circle. . . . .	208

6.20 Watchdog-Type Algorithm with Large Initial Step Size. Initial step size  $\alpha = 0.45$ , inflation factor  $\rho_{\text{inf}} = 1$ , deflation factor  $\rho_{\text{def}} = 0.5$ . Serial Gradient Method is checking objective decrease condition (Line 19 in Algorithm 16) in each step. Crosses mark the iterations that fails the objective decrease condition test and thus halves the subsequent step sizes. The watchdog-type parallel-in-time gradient-type method with  $I_{\text{inner}} = 5$  tests the decreasing condition every 5 steps, marked by the big red circle (pass) or cross (fail). . . . . 210

6.21 Watchdog-Type Algorithm with Large Initial Step Size. Initial step size  $\alpha = 0.45$ , inflation factor  $\rho_{\text{inf}} = 1.02$ , deflation factor  $\rho_{\text{def}} = 0.5$ . Serial Gradient Method is checking objective decrease condition (Line 19 in Algorithm 16) in each step. Crosses mark the iterations that fails the objective decrease condition test and thus halves the subsequent step sizes. The watchdog-type parallel-in-time gradient-type method with  $I_{\text{inner}} = 5$  tests the decreasing condition every 5 steps, marked by the big red circle (pass) or cross (fail). . . . . 212

A.1 Spectral Radius Against Step Size and Number of Subdomains. Two horizontal axes are for step size  $\alpha$  and number of subdomains. The vertical axis is for the spectral radius. A yellow transparent horizontal layer is added at  $z = 1$ . See Figure A.3 and Figure A.2 for two dimensional slices of this three dimensional plot. . . . . 234

A.2 Spectral Radius Against Step Sizes. Five curves represent five different number of subdomains. . . . . 235

A.3 Spectral Radius Against Number of Subdomains. Four curves represent four different step sizes. . . . . 237

A.4 sparsity pattern of the  $\mathbf{H}_{\mathcal{L}}$ ,  $N = 4$  subdomains, for test case 2 ((3.6.5) in Section 3.6.3 where  $n_u = n_y = 31, K = 30$ ) . . . . . 239

A.5 Performance of the parallel Krylov subspace solver on problem (A.3.2).  
 The red dot represent the serial bench mark. The horizontal axes are  
 number of subdomains. The Top plot shows the number of iterations,  
 $I(N)$ , to reduce control error from initial  $1e-1$  to  $1e-12$ . Theoretical  
 speed-up is  $N \cdot I(1)/I(N)$ . Parallel efficiency is  $I(1)/I(N)$ . . . . . 240

A.6 Performance of the parallel Krylov subspace solver on problem (3.6.1).  
 The red dot represent the serial bench mark. The horizontal axes are  
 number of subdomains. The Top plot shows the number of iterations,  
 $I(N)$ , to reduce control error from initial  $1e1$  to  $1e-9$ . Theoretical  
 speed-up is  $N \cdot I(1)/I(N)$ . Parallel efficiency is  $I(1)/I(N)$ . . . . . 241

A.7 Performance of the parallel Krylov subspace solver on problem (3.6.5).  
 The red dot represent the serial bench mark. The horizontal axes are  
 number of subdomains. The Top plot shows the number of iterations,  
 $I(N)$ , to reduce control error from initial  $1e2$  to  $1e-5$ . Theoretical  
 speed-up is  $N \cdot I(1)/I(N)$ . Parallel efficiency is  $I(1)/I(N)$ . . . . . 242

D.1 Time Subdomains Adjustment . . . . . 272

D.2 Trace graph of 1000 seconds about 8 optimization iterations with evenly  
 split time subdomains, generated by HPC`TOOLKIT` [ABF<sup>+</sup>10]. The  
 time subdomains are  $[0, 250]$ ,  $[250, 500]$ ,  $[500, 750]$ ,  $[750, 1000]$  The hor-  
 izontal axis is the time axis. From top to bottom, the 16 rows corre-  
 sponding to 16 cores are grouped into 4 groups. The top 4 horizontal  
 rows represent computation timing of the 4 cores performing parallel  
 computing in the first time subdomain, etc. One optimization iteration  
 consists of three major blocks of one purple, one brown, and one green  
 block. Purple blocks are for forward computation, brown blocks are  
 for backward computation, and green blocks are for waiting. Model  
 problem description in Section 6.3.2.2. . . . . 273

- D.3 Trace graph of 1000 seconds about 9.5 optimization iterations with adjusted time subdomains, generated by HPCTOOLKIT [ABF<sup>+</sup>10]. The adjusted time subdomains are [0, 196], [196, 442], [442, 720], [720, 1000]. The horizontal axis is the time axis. From top to bottom, the 16 rows corresponding to 16 cores are grouped into 4 groups. The top 4 horizontal rows represent computation timing of the 4 cores performing parallel computing in the first time subdomain, etc. One optimization iteration consists of three major blocks of one purple, one brown, and one green block. Purple blocks are for forward computation, brown blocks are for backward computation, and green blocks are for waiting. Model problem description in Section 6.3.2.2. . . . . . 274
- D.4 Comparison of objective function value history between the parallel-in-time gradient-type method with different time subdomain partitions. The evenly split time subdomains are [0, 250], [250, 500], [500, 750], [750, 1000]; The adjusted time subdomains are [0, 196], [196, 442], [442, 720], [720, 1000]. The first initialization step of a full gradient-sweep is the same for both cases and is not included in the plot. Model problem description in Section 6.3.2.2. . . . . . 275

# List of Tables

3.1	Subdomain Representation of Figure 3.2 . . . . .	41
3.2	$T = 8, K = 200$ example. Number of iterations and time consumed to reduce the Infinity-norm control error from the initial guess $1e+3$ to $1e-3$ . Conjugate Gradient method takes only 530 iterations to reach the same level of control error. . . . .	77
3.3	$T = 16, K = 400$ example. Number of iterations and time consumed to reduce the Infinity-norm control error from the initial guess $1e+3$ to $1e0$ . Conjugate Gradient method takes only 340 iterations to reach the same level of control error. . . . .	78
3.4	Test Case 1 Configuration of Section 3.6.3.1. Total time steps $K = 30$ .	82
3.5	Test Case 1 Configuration of Section 3.6.3.2. Total time steps $K = 30$ .	84
4.1	Notations for the Proof of Lemma 4.3.1. Variables with superscript “*” are corresponding to the optimal solution. . . . .	99
5.1	Notations for the Proof of Lemma 5.1.3, Lemma 5.1.4, and Lemma 5.1.5	118
5.2	Summary of Lemma 5.2.1 Proof. See the proof for the notations. . .	134
B.1	Nomenclature for the Reservoir Simulation . . . . .	245
B.2	Nomenclature for the Numerical Scheme of Reservoir Simulation . .	250

# Chapter 1

## Introduction

Optimal control problems governed by a time-dependent partial differential equation (PDE) system are widely studied in various fields of applications, as unsteady fluid-dynamics problems [Gun03], [BLUU12], oil reservoir water flooding optimization [Jan11], pollutant source inversion [DCZ15], mixing enhancement in microfluidic technologies [HCCT13], etc. Iterative solution of these problems using gradient based methods involve the time consuming repeated solution of a state equation and a so-called adjoint equation. The state equation and adjoint equation of an optimal control problem governed by a time-dependent system are strongly coupled in time and their solution is inherently serial. In other words, the solution of these equations on different parts of the time domain needs to happen in chronological order, which prevents any simple application of parallel computing in the time dimension to accelerate the overall optimization.

To efficiently and rapidly solve these time-dependent optimal control problems by utilizing parallel computing, I propose and analyze a new parallel-in-time gradient-type method (also referred to as “parallel gradient-type method” in this document).

Suppose the PDE is discretized in the space dimension. Let  $T > 0$  be the time domain length and let  $n_y, n_u \in \mathbb{Z}^+$  be the dimension of the state and control variables respectively at any point in time. For  $t \in [0, T]$ , the state is  $y(t) \in \mathbb{R}^{n_y}$ , the control

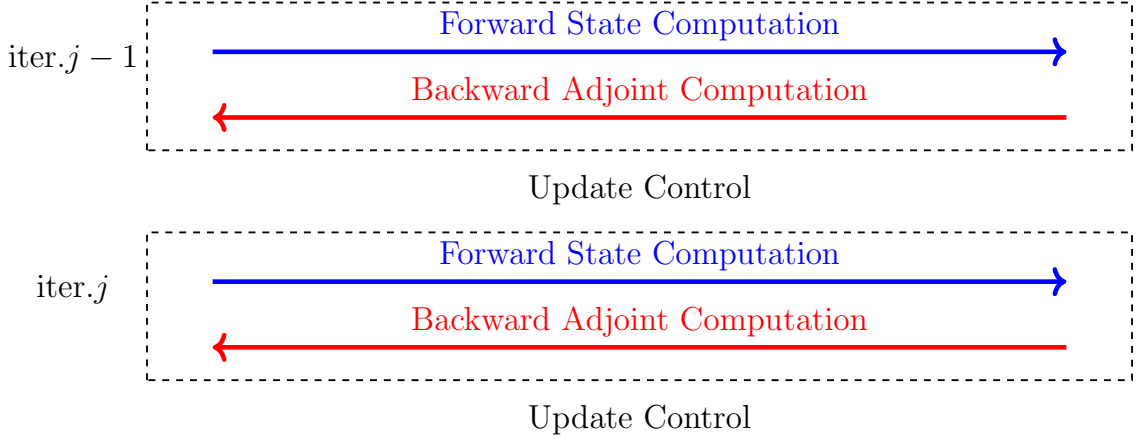


Figure 1.1: Workflow of the classical gradient method

is  $u(t) \in \mathbb{R}^{n_u}$ . Let  $J : \mathbb{R}^{n_y} \times \mathbb{R}^{n_u} \times \mathbb{R} \rightarrow \mathbb{R}$  and  $F : \mathbb{R}^{n_y} \times \mathbb{R}^{n_u} \times \mathbb{R} \rightarrow \mathbb{R}^{n_y}$  be given functions. Let  $y_{\text{given}} \in \mathbb{R}^{n_y}$  be a given initial state. The model problem is in (1.0.1).

$$\min_{y,u} \int_0^T J(y(t), u(t), t) dt, \quad (1.0.1a)$$

$$\text{subject to } \frac{d}{dt} y(t) = F(y(t), u(t), t), \quad t \in (0, T), \quad (1.0.1b)$$

$$y(0) = y_{\text{given}}. \quad (1.0.1c)$$

To solve (1.0.1), each iteration of the classical gradient method in reduced control space requires the forward in time solution of the state equation and the backward in time solution of the adjoint equation before the gradient can be computed and the control can be updated. Both of the forward and backward computation is sequential in nature and consumes most of the computation time in the whole optimization process, see Figure 1.1.

The new parallel-in-time gradient-type method introduces parallelism in the forward and backward computation. In the simplest case, the proposed new parallel gradient-type method evenly splits the time domain into  $N$  time subdomains and executes the forward and backward computation in each of the  $N$  time subdomains in parallel. To enable the parallelism in  $N$  time subdomains, I sacrifice the exactness of the gradient vector and aim only for a gradient-type vector surrogate. To

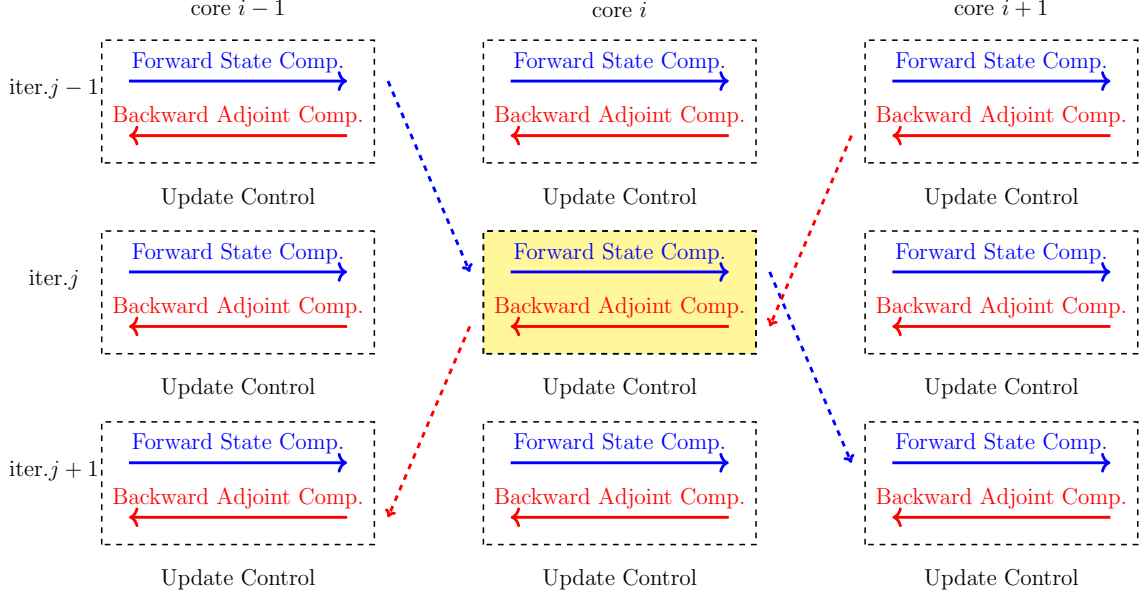


Figure 1.2: Workflow of the parallel-in-time gradient-type method

start computations on all time subdomains simultaneously, quantities computed in the last optimization iteration are used as the initial/terminal conditions for the forward/adjoint solves, which is readily available for each subdomain at the beginning of each optimization iteration. This is the key idea of the proposed new parallel-in-time gradient-type method. However, using quantities from last iteration to start parallel computing on subdomains also results in jumps in the state/adjoint variables at time subdomain boundaries. After the state and adjoint variables are computed, the new method uses the same formula as in the classical gradient method to compute a gradient-type vector to update the control and begin the next iteration. See Figure 1.2.

In the case where  $N$  time subdomains evenly split the time domain, since the length of the forward and backward computation in each time subdomain is  $1/N$  of the original time domain, the computation time for each optimization iteration is expected to be about  $1/N$  of the computing time of the classical gradient method iteration. This is where the parallel-in-time gradient-type method brings potential



speed-ups, if the control update computed in parallel is good enough as a surrogate of the gradient. It is also important to note that the acceleration by parallelism in time multiplies the existing parallelism in the solution of state and adjoint equations.

At any iteration of the parallel gradient-type optimization, the state, adjoint, and control variables typically do not satisfy the state equation and the adjoint equation at the time subdomain boundaries as a result of using quantities from the last iteration. Since the new method does not use an exact gradient in the control update, this new method is not an acceleration of the exact classical gradient method and therefore I refer to it as a ‘gradient-type’ method.

I also present a generalization of this newly proposed method that allows the different time domain partition of the forward and backward computation and overlapping computation domains.

The theoretical analysis consists of results for linear-quadratic problems and the broader class of general non-linear problems.

For convex linear-quadratic discrete-time optimal control (DTC) problems, the generalized method can be interpreted as a multiple part splitting method (for the  $N$  evenly splitted subdomains in forward and backward computation case, it is a  $(2N - 1)$ -part splitting method [de 76, dN81]) for solving the optimality condition

$$\mathbf{H}\mathbf{u} = \mathbf{g},$$

where  $\mathbf{H}$  is the Hessian, and  $\mathbf{g}$  is the gradient of the reduced control space objective.

However, in the literature, there is not a convergence result applicable to the specific splitting pattern that results from this new method and the existing convergence theorems for the classical gradient method also does not apply since the control updates of this method does not use exact gradient. For linear-quadratic problems, I prove the convergence of this new method by using its structure as a multiple part splitting method and spectral radius properties of an implicitly constructed iteration matrix as a block companion matrix [DTW71].

I also study the new parallel-in-time gradient-type method from the point of view of the multiple shooting reformulation of the optimization problem. I show the new method can be seen as using a gradient-type method to solve the saddle point problem of the multiple shooting formulation optimality system. An alternative proof of convergence is given for the linear-quadratic problems using spectral radius type of argument again.

Then, I investigate the behavior of the parallel gradient-type method for general nonlinear optimization problems. Particularly, I consider the projected parallel-in-time gradient-type method whose iteration is the parallel gradient-type iteration appended by a metric projection step that projects the control into a closed convex set. Different from the spectral radius argument used in the previous linear-quadratic problem convergence proofs, I use another approach based on the idea that, when the control update is small, the gradient-type vector computed in parallel is similar to the true gradient. I establish a series of theorems and give convergence proofs with different assumptions on the problem, such as the convexity of the objective function or compact control constraints. Additionally, I present results on the monotonic convergence of the method given small step size.

I present numerical examples to demonstrate the proposed method performance.

In optimal control problems governed by 3D linear advection-diffusion-reaction PDE systems, strong scaling is observed in one example with up to 50 cores, i.e., 50 times speed-ups is achieved by the parallelism of 50 cores compared to the serial classical gradient method. Numerical results to demonstrate other properties of the method are also presented.

This work is motivated by an optimization problem arising in the well rate control in the water flooding process of secondary oil reservoir recovery. This is an optimal control problem governed by a system of highly nonlinear time-dependent PDEs. The forward computation, i.e., reservoir simulation, and the backward computation is both very computationally expensive. In this reservoir optimization prob-

lem, the application of the proposed parallel-in-time gradient-type method results in significant speed-ups. In one experiment, 4-time-subdomain parallel gradient-type method with, in each subdomain, 4 cores in the space dimension for parallel linear solvers/preconditioners (16 cores in total) is compared with classical gradient method with 4 cores in the space dimension. About 4 times speed-ups is obtained.

This document is organized as follows. In Chapter 2, I review related work in the parallel-in-time solution of time-dependent systems, parallel-in-time optimization, and oil reservoir optimization. In Chapter 3, I use the linear-quadratic discrete-time optimal control problem to introduce the idea of the parallel-in-time gradient-type method. I first develop a compact notation for the classical gradient method, then derive and reorganize the parallel-in-time gradient-type algorithm into a similar compact form which reveals its nature of being a  $(2N - 1)$ -part splitting method. Then, I generalize the newly proposed method to accommodate flexible overlapping partition of time domains. I prove convergence of the generalized method and demonstrate its performance by numerical experiments, applying the parallel-in-time gradient-type method to 1D and 3D linear-quadratic optimal control problems. In Chapter 4, I introduce the direct multiple shooting reformulations of linear-quadratic optimal control problems, which leads to another interpretation of the parallel-in-time gradient-type method. An alternative proof of convergence is given using the multiple shooting perspective. In Chapter 5, I give convergence proofs and related properties for general nonlinear problems. In Chapter 6, I describe the wellrates optimization problem and compare numerical results of the classical gradient method and the parallel-in-time gradient-type method.

I include several appendices. Appendix A consists of some research without a clear conclusion but still draws insights into the parallel-in-time gradient-type method, including discussions on one way of proving the projected parallel gradient-type method convergence, the numerical experiments investigating behavior of the spectral radius of an iteration matrix that determines the convergence speed of the method, and a

parallel-in-time Krylov subspace method that is closely related to the multiple shooting formulation of the linear-quadratic problem. Appendix B describes in detail the incompressible immiscible two-phase subsurface flow model PDE and its discretization that is used in the reservoir optimization problem. Appendix C talks about implementation details of the proposed method on the reservoir optimization problem with an emphasis on comparing the parallel-in-time gradient-type method applied to problems with explicit and implicit (as the case of the reservoir problem) expression of state equations. Appendix D presents a test on using a time subdomain partition based on computation load, instead of even number of time steps, for load balance of the parallel-in-time gradient-type method applied to the reservoir optimization problem.

# Chapter 2

## Literature Review

In Section 2.1, I first review some related work on parallel-in-time simulation which give rise to important ideas for parallel-in-time optimization. In Section 2.2, I review related literature on parallel-in-time optimization including some commonly used concepts as additive Schwarz preconditioners, direct/indirect multiple shooting formulations of the time-dependent optimal control problems, Parareal [LMT01] based parallel simulation. At last, in Section 2.3, I review related work in the water flooding optimization problem on oil reservoir management.

### 2.1 Parallel-In-Time Simulation

In [LRSV82], a waveform relaxation family of algorithms was first proposed for analyzing nonlinear dynamical system in the time domain. Waveform relaxation is a Gauss-Jacobi type relaxation iterative method. (It can be also Gauss-Seidel type. Because the Gauss-Jacobi type is more straightforward to parallelize, I discuss the Gauss-Jacobi type below.) First, the unknowns are partitioned into spatial subdomain. Then, in each Gauss-Jacobi relaxation iteration, the algorithm fixes unknowns from all partitions but one and carries out the simulation in time for this one partition of unknowns. A suitable integration formula and steps can be applied only to

this partition of unknowns, which allows different methods to integrate different unknowns. The above algorithm constitutes a Gauss-Jacobi relaxation step that can be parallelized and multiple time-dependent simulations can be readily run in parallel. Strictly speaking, this algorithm does not belong to the specific type of parallel-in-time simulation that I discuss here, since it does not perform computation corresponding to different time subdomains at the same time. But the parallel-in-time gradient-method shares the idea that in one iteration, computation in each partition runs independently with other partitions with essential information from other partitions fixed as quantities computation from the last iteration.

‘Parareal’, an innovative parallel-in-time algorithm for simulation, was proposed in [LMT01]. See also the overview [Gan15]. The method decomposes the time domain into  $N$  subdomains and makes use of a coarse( $\mathcal{C}$ ) and a fine( $\mathcal{F}$ ) grid time integrator. In the  $k$ th iteration, the coarse one integrates over the whole time domain in serial, the fine one integrates each one of the  $N$  time subdomain in parallel, and the initial value of each time subdomain is updated by a simple formula (2.1.1) using integration results from the coarse,  $\mathcal{C}_{T_n \rightarrow T_{n+1}}(Y_{T_n}^{(k+1)})$ , and fine,  $\mathcal{F}_{T_n \rightarrow T_{n+1}}(Y_{T_n}^{(k)})$ , integrations.

$$Y_{T_{n+1}}^{(k+1)} = \mathcal{C}_{T_n \rightarrow T_{n+1}}(Y_{T_n}^{(k+1)}) + \mathcal{F}_{T_n \rightarrow T_{n+1}}(Y_{T_n}^{(k)}) - \mathcal{C}_{T_n \rightarrow T_{n+1}}(Y_{T_n}^{(k)}) \quad (2.1.1)$$

After  $N$  iterations the method produces exactly the same integration result as the fine grid integrator would, but this would not accelerate the simulation. In practice, for many cases, far less than  $N$  iterations are needed to achieve desirable error and thus results in lower computation time when executed in parallel. From my point of view, it can be partly seen as making use of the fact that the temporal dependency between states of two time point is decreasing and diminishing as the time distance between these two time increases. Many works based on ‘Parareal’ have been done since the conception of this idea, some of which are reviewed below, e.g. [Com05, DSSS13].

## 2.2 Parallel-In-Time Optimization

The widely cited paper [TBA86] presents convergence results of a class of distributed asynchronous gradient optimization algorithms. The parallel-in-time gradient-type method does not exactly fall into the category of algorithms discussed in the paper. However, they share the same spirit that, in the case where each processor is computing a gradient-like updating step for one component of the optimization variable, the information from other component suffers a delay. The convergence theorem and its proof provided in the paper [TBA86] bears significant similarity with the parallel-in-time gradient-type method convergence theorem for non-linear problems that I provide in this dissertation. The first similarity is that convergence is established based on a sufficiently small gradient-type step size whose theoretical upper bound is not particularly tight. If we can trace the proof deductions and find such a upper bound, the upper bound will lead to convergence but not to optimal performance. The second similarity is in the idea of the proof that when the step size is small, the information, i.e. state/adjoint/control, in the distributed system is similar to the information in a virtual centralized system at a specific status, and the gradient-type step made in the distributed system is actually in a descent direction for the information in the virtual centralized system.

The paper [CCL89] presents a method that explicitly decomposes a time-dependent optimal control problem (P) with initial state condition into sub-optimal control problems (P-j) with both initial and terminal conditions on small time subdomains as below (see [CCL89, Equation 2.1-2.5]),

$$\begin{aligned}
 \text{(P)} \min_{x,u} g_N(x_N) + \sum_{i=0}^{N-1} g_i(x_i, u_i), & \quad \text{(P-j)} \min_{x,u} \sum_{i=(j-1)T}^{jT-1} g_i(x_i, u_i), \\
 \text{subject to } x_{i+1} = f_i(x_i, u_i), & \quad \text{subject to } x_{i+1} = f_i(x_i, u_i), \\
 i = 0, \dots, N-1, & \quad i = (j-1)T, \dots, jT-1, \\
 x_0 = x_{\text{given}}, & \quad x_{(j-1)T}, x_{jT-1} \text{ given.}
 \end{aligned}$$

This method is an iterative method where in each iteration it takes a two level approach. In the lower level, optimal control problems with both initial and terminal state constraints are solved exactly assuming feasibility. Note that feasibility with both initial and terminal constraints is not trivial for many applications. In the higher level, the initial and terminal state values of each time subdomain are updated, e.g., by a Newton step using first and second order information from the lower level. This method shares a similar spirit with indirect multiple shooting method in the sense that in the higher level of both methods variables at the time subdomain boundaries representing initial/terminal values of states/adjoints are updated and the control  $u$  in the original problem is updated on the lower level. But there is an apparent difference in how they make use of the information from the lower level. This method only uses information at the subdomain boundary, which of course depends on the interior of the subdomain, whereas indirect shooting method explicitly uses control information on the whole time subdomain. In [CCL89], the banded structure of the Hessian matrix in the Newton's iteration is exploited by a parallel general banded matrix cyclic reduction based solver [Hel76]. However, due to hardware constraints in the year 1989, numerical experiments were only carried out with at most 7 processors, though speedups were observed evidently, the scaling potential was not sufficiently demonstrated.

In [Wri90], parallel computing is used in solving banded linear system [Wri91a] arising in the sequential quadratic programming method applied to a time-dependent optimization problem. The banded structure is resulted from the optimality system of the time-dependent problem. However, other than the matrix being generally banded, other structure and characteristics of time-dependent problems are not exploited in this work.

After [Wri90], the same author continued to explore more structure of the problem in [Wri91b]. The method bears similar flavor as in [CCL89] in terms of using a multiple-shooting decomposition of two point boundary value problem style. The dif-



ference lies in that [CCL89] focuses on decomposing the problem into in the nonlinear problem level in which each subproblem has a clear physical meaning and [Wri91b] focuses on decomposing the banded linear system resulted from one iteration of Newton’s method or of the SQP method. In contrast to the commonly seen upper-lower two level structure of multiple-shooting type algorithms, arbitrary number of levels, i.e., recursive parallelism, are introduced. After optimizing number of parallel levels and allocation of processors in each level, different number of levels are compared in terms of performance. The paper does not explicitly decompose the time domain, but decompose the linear system that results from this time-dependent system. The parallelism is in the construction of the small linear system of the “separator variables”, which in some sense resemble shooting variables. One major computation is in the matrix-matrix multiplications in this construction which does not exist in other typical algorithm using multiple shooting idea. The two numerical experiments have state and control dimension in each time step below three with large number of times steps. The parallel algorithm has about 50% more computation than its serial counterpart. With 8 processors, about 5 times speed up is achieved.

The paper [Ral96] uses has similar two level structure for parallelization as [CCL89] and applied Differential Dynamic Programming (DDP) method on DTOC problems. Again, [Ral96] assumes state equation feasibility given both initial and terminal condition. Relatively good parallel efficiency 70% is observed in some test cases compared to 25% – 50% efficiency in related works targeting the same problems. However, in all of the numerical examples, the dimension of state and control variables in each time step is only one or two, which is not the case in PDE constrained optimization.

The paper [BH97] focuses on time-dependent linear-quadratic optimal control problems. In a direct multiple shooting framework, they split time domain, introduce auxiliary (shooting) variables at the time subdomain boundaries and derive an exactly equivalent formulation of the original problem. I will also include the perspective of this formulation to examine my algorithm. By this formulation, the

optimality system can be arranged as a block tridiagonal and they rely on Gauss-Seidel type ‘red-black’ relaxation approach to parallelize the computation. In each optimization iteration, they solve exactly the sub optimization problems whereas the parallel-in-time gradient-type method can be seen as only applying one gradient-type step to very roughly solve the sub optimization problem in some sense. The penalty term in the Augmented Lagrangian to reduce state variables discontinuity at the time subdomain boundaries has significantly improved the convergence.

The paper [MT02] proposes the method that partitions the time domain, adds an auxiliary variable of the initial value of each subdomain into control variables, and includes in the modified objective function a large penalty term of discontinuity of states on subdomain boundaries. Then, in the gradient method in the reduced control space, the forward/backward computation in time subdomains is parallel. The parallel-in-time gradient-type method shares this spirit in the sense that the subproblems are not solved exactly so that optimal solutions are found in each iteration as in [BH97] but only a single gradient-type step is applied to each subproblem as a part of the gradient step of the whole problem. To compensate the delay of information exchange due to the domain partition, a coarse grid integrator is used to precondition the gradient method. In the parallel-in-time gradient-type method, as opposed to the coarse-fine two level of integrator in [MT02], there is only one integrator and, therefore, suffer the delay of information exchange discussed here. Numerical results of [MT02] show that, in a case where time domain is partitioned into 100 parts, with the help of the preconditioner, the parallel algorithm needs even significantly fewer iterations to converge than the plain gradient method with no time domain partitioning and each iteration is 100 times faster than the plain gradient method.

The paper [Hei05] derives a block tridiagonal optimality system akin to that in [BH97]. Instead of solving sub optimization problems, Gauss-Seidel forward/backward sweeps on the whole system are used to cluster the eigenvalues and form a good preconditioner. However, the quality of the preconditioner deteriorates [Com05] when

the Gauss-Seidel forward/backward sweeps are replaced by the parallelizable ‘red-black’ ordering sweeps. I also explored the possibility of using the optimality system in [Hei05] after a permutation to reveal parallelism explicitly to perform a parallel Krylov subspace method in Section A.3 since it is closely related to the proposed parallel-in-time gradient-type method.

The Ph.D. dissertation [Com05] uses the type of KKT system factorization based preconditioner developed in [BG99], the latter of which can be thought of as an extension of [BH98]. The parallelism in time is not explicit in the original [BG99]. However, the preconditioner developed in [BG99] needs an approximate forward solver and [Com05] applies ‘Parareal’ technics [LMT01] in the forward/backward solves to exploit parallelism in the time domain.

With the application background of quantum system control, [MST07] applied alternate direction descent method to obtain a monotonic algorithm. In each iteration, first, the state and adjoint equations are solved by Parareal coarse propagator. Then, instead of using the coarse propagator result directly as shooting variables, an optimization problem with jump penalty added to the original objective function is solved and its solution is used as shooting variables. At last, given shooting variables, controls are sought for in parallel in each of the sub time domain. By this algorithm, the monotonic decreasing in objective function value along convergence is proved for this application. Careful performance timing for scaling study is not provided.

The paper [DSSS13] eliminates the state and adjoint variables from the optimality system (2.2.1) (see [DSSS13, Equation (1.1)])

$$\begin{bmatrix} \mathbf{K} & 0 & \mathbf{E}^T \\ 0 & \mathbf{G} & \mathbf{N}^T \\ \mathbf{E} & \mathbf{N} & 0 \end{bmatrix} \begin{bmatrix} y \\ u \\ p \end{bmatrix} = \begin{bmatrix} f_1 \\ 0 \\ f_2 \end{bmatrix} \quad (2.2.1)$$

in a Schur complement fashion and obtains a reduced Hessian  $\mathbf{H} \stackrel{\text{def}}{=} \mathbf{G} + \mathbf{N}^T \mathbf{E}^{-T} \mathbf{K} \mathbf{E}^{-1} \mathbf{N}$ . To solve for the optimal control from a linear system  $\mathbf{H}u = b$ , variants of the Full Orthogonalization Method (FOM) is used. The matrix-vector multiplication is calcu-

lated approximately. In the matrix-vector multiplication, applying  $\mathbf{E}^{-T}$  and  $\mathbf{E}^{-1}$  to a vectors is approximated by using Parareal method [LMT01] to solve the underlying PDE and the adjoint PDE approximately in parallel inexactly.

In contrast to [BH97, Hei05], [BS14] forms the first-order KKT condition without adding auxiliary (shooting) variables. In this way, the optimality condition system (2.2.2) (see [BS14, Equation (8)]) includes a 3 by 3 block matrix of large dimensionality,

$$\begin{bmatrix} \tau\mathcal{M}_1 & 0 & -\mathcal{K}^T \\ 0 & \beta\tau\mathcal{M}_2 & \tau\mathcal{N}^T \\ -\mathcal{K} & \tau\mathcal{N} & 0 \end{bmatrix} \begin{bmatrix} y \\ u \\ p \end{bmatrix} = \begin{bmatrix} -\tau\mathcal{M}_1\bar{y} \\ 0 \\ d \end{bmatrix}. \quad (2.2.2)$$

To solve this system, they use a preconditioned QMR iterative linear solver. The parallelism lies in the additive Schwarz preconditioner with an overlapping time domain decomposition. By decomposing the time domain, many smaller subproblems are derived by extracting corresponding rows and columns from (2.2.2). These subproblems are solved in parallel.

The paper [ACG14] uses parallelism in a diesel engine optimal control problem where one aims to minimize fuel consumption with fuel injection pattern as control variables, 5-dimensional at each time step, under the constraints of engine model, i.e., required engine speed, emission bound, and other mechanical/thermal constraints. The emission bound is a single inequality constraint involved with a integral over the whole operation time span. After the time domain is decomposed Lagrange multipliers are used to deal with this constraint. In terms of time domain decomposition, it follows [CCL89] where a two-level structure is adopted, the higher level of which solves for the optimal state at the sub time domain boundaries and the lower level of which are optimization problems with both initial and terminal state conditions. The air-path model, i.e., the state equations, is a system of ODEs and has state variable dimension five at each time step. For problems like the oil reservoir optimization problem, typically, it is impossible to force the reservoir pressure/saturation to an

arbitrarily given state by only controlling the well rates. The paper [ACG14] also mentions an interesting notion of “characteristic constant of the underlying system” that, loosely speaking, describes how long into the future the current state will have an effect. It is much smaller than the length of sub time domains, which leads to the fact that the requirement of the terminal states only affects a few time steps prior to the terminal time. The characteristic constant also determines the length of overlapping region of the sub time domains. An overlapping region of every two consecutive time sub domains is introduced to enforce the global state continuity. It is also related to the time for which a zero adjoint variable at a terminal time with no terminal state condition grows to a typical magnitude. The paper does not provide detailed results on the parallel performance and scalability.

An indirect multiple shooting formulation is used in [CGR14] to solve parabolic optimal control problems. This indirect multiple shooting formulation is compared with direct multiple shooting in [CG15]. The work does not focus on the parallelization of computation but on other aspects, such as handling possible instability in the problem. However, it is highly related to the other works in the parallel-in-time optimization. The indirect multiple shooting method consists of a two-step fixed point iteration. In the first step, shooting variables, i.e. boundary values of states and adjoint variables, are fixed and the sub optimality system that determines controls on each time subinterval is solved. One can conveniently parallelize solving these subproblems. In the second step, using the control variable obtained in the first step, shooting variables are sought to fit the continuity condition, i.e., ‘the shooting system’, on time subinterval boundaries. This calculation involves the dynamics on the whole time domain and is not as straightforward as the first step to parallelize.

Similar to [BS14], [DCZ15] use an additive Schwarz preconditioner on the full space KKT system, but the Schwarz preconditioner is in both space and time. They reorder the KKT system in the full space in ‘fully coupled ordering’ where the state,  $y$ , and adjoint,  $\lambda$ , variable are placed near each other if the state variable and the equation

of the adjoint variable correspond to the same space and time; in the application to pollutant source inversion problem, the controls (in this case, the parameters to identify),  $u$ , only exist on  $s$  discrete points and are placed at the end of the unknown vector,

$$\begin{aligned} \mathbf{Unknown} = & (y_0^0, \lambda_0^0, y_1^0, \lambda_1^0, \dots, y_N^0, \lambda_N^0, \\ & y_0^1, \lambda_0^1, y_1^1, \lambda_1^1, \dots, y_N^1, \lambda_N^1, \\ & y_0^{N_T}, \lambda_0^{N_T}, y_1^{N_T}, \lambda_1^{N_T}, \dots, y_N^{N_T}, \lambda_N^{N_T}, \\ & u_0^0, \dots, u_s^0, \dots, u_0^{N_T}, \dots, u_s^{N_T}), \end{aligned} \quad (2.2.3)$$

so that the additive Schwarz preconditioner can be naturally applied by extracting submatrices of continuous row and column indices. In this application, this preconditioner scales reasonably well. In one example, strong scaling is achieved.

In [DCZ16] the parallel preconditioner is improved by replacing it with a two level Schwarz preconditioner of the form (see [DCZ16, Equation (8)]),

$$\begin{cases} y = I_c^h F_c^{-1} I_h^c x, \\ M_{\text{two-level}}^{-1} x = y + M_{\text{one-level}}^{-1} (x - F_h y), \end{cases} \quad (2.2.4)$$

where  $I_c^h$  is the fine mesh to coarse mesh restriction operator,  $I_h^c$  is the coarse to fine mesh interpolation operator, and  $F_c$ ,  $F_h$  are the forward operator of coarse and fine mesh respectively. In some numerical examples, the two-level preconditioner based solver is four times faster than the one-level preconditioner by reducing number of GMRES iterations by a factor of four. Strong scaling is achieved in supercomputer using around 1000 cores on a source inversion problem.

## 2.3 Reservoir Optimization

I introduce oil reservoir simulation/optimization and review related literature on numerical optimization of water flooding in oil reservoir secondary recovery.

**Reservoir simulation.** Reservoir simulation is simulating the fluid (water, oil, etc.) flow in the subsurface rock structure and injection/production wells. Usually, reservoir states such as pressure, phase saturation are computed using a time-dependent nonlinear partial differential equation system [Wie10], [AGL07]. Extensive research has been done on the modeling and on solving the PDEs using the Finite Difference Method, the Finite Volume Method, the Finite Element Method, Discontinuous Galerkin Method, etc. A crash course in reservoir simulation is given in [AGL07] and a more extensive introduction is [CHM06]. My simple finite volume simulator is based on the one described in [AGL07].

**Reservoir optimization.** The development of an oil reservoir requires large investments. To maximize profit, various kinds of optimization can be applied in different stages of exploration, production, etc. In the exploration stage, optimization is done to evaluate potential reservoir profit under uncertainty. In the development stage, location and size of reservoir production facilities such as platforms, wells, and pipes need to be decided with the aid of optimization. In the production stage, optimization can be applied to control reservoir states such as pressure and liquid flow, and manage production/injection constraints on well-rates and wellhead/flow line pressure [Wie10]. Around the year 2000, new ‘smart wells’ appeared that allow monitoring of well hole flow rates/pressure and at the same time can control downhole valves independently. This degree of flexibility added by ‘smart well’ has required more optimization to improve reservoir operation [YDA02]. In this project, the optimization problem focuses on the secondary reservoir production stage where I optimize the well injection/production rates to maximize oil recovery revenue.

**Related Research on Reservoir Optimization.** I discuss selected papers on numerical optimization of oil reservoir secondary/tertiary production and their relationship to my work.

As early as 1987, Fathi and Ramirez [FR87] applied optimal control theory to

revenue optimization of a micellar/polymer flooding enhanced oil recovery system. Previously, optimal control theory was mostly applied only to parameter estimation, but not to the time-dependent dynamic process. They use a one dimensional model with injection control of five chemical components on one boundary. The objective function involves oil revenue and chemical injection costs. Significant improvement was shown compared to manual injection setting. The 1D numerical grid is of size 40 and simulation has 1,500 time steps. Data is stored on magnetic tape. In the gradient based optimization, the paper uses regularization, i.e., smoothing suboptimal control trajectories in the case of convergence stalling, to effectively accelerate optimization iteration. This type of regularization may help my optimization. Multiple local maximums were discovered in this very early study and it remains in my project.

In 1988, Asheim [A<sup>+</sup>88] attempted to use numerical optimization to maximize water sweep efficiency by controlling injection and production rates. Instead of costly finite difference approximation of the gradient, he uses analytically derived derivatives of a very simplified reservoir model. This is one of the earliest papers on numerical optimization of water flooding.

In 1996, Zakirov, Aanonsen, Zakirov, and Palatnik [ZAZP96] use a fully implicit 3D three phase black-oil reservoir model and adjoint based optimal control theory approach to optimize the net present value of a oil reservoir. The size of the numerical grid is not stated and number of wells are below ten. Similarly, I also use the adjoint approach to compute gradient-type quantities for the optimization.

In 2001, Sudaryanto and Yortsos [SY<sup>+</sup>01] worked on water flooding optimization using a model with miscible equal viscosity fluid. They used particles at the water fronts as state variables, which is different from my cell wise saturation/pressure state variable. After some simplification, such as homogeneous permeability, the analysis yields clean and intuitive results. For example, the optimal control is Bang-Bang and there is only one active injector at any time, and injections are conducted in geographical order from farther wells to nearer wells.



In 2002, Yeten, Durlafsky, and Aziz [YDA02] applied optimization on the operation of smart wells. They used forward finite difference approximation to calculate the gradient needed in the nonlinear conjugate gradient method. The detail of the simulation and optimization is not stated since they used the commercial ECLIPSE and FLUVSIM software packages. The experiments were conducted using several highly heterogeneous channelized reservoir which is comparatively hard to simulate numerically.

Also in 2002, Wang, Litvak, and Aziz [WLA<sup>+</sup>02] used sequential quadratic programming (SQP) to simultaneously optimize the well rates and lift-gas rates. It is a derivative based method. However, they do not use a PDE system to model the reservoir. Instead, they used a gathering system as a loopless tree-like network to model the short term production operation. They used SNOPT [GMS06] to implement the SQP.

In 2004, Brouwer studied water flooding optimization using optimal control theory in his PhD thesis [Bro04]. Two phase flow in a horizontal plane and three phase black oil formulation in 3D space are both used to model the reservoir. I am only using a two phase 3D reservoir model. As for the well, Brouwer uses a well model involving geometric factor, rock/fluid property, well flowing pressure, grid block pressure, and well rate. But in my project, I am directly controlling the well rate. In acquiring the gradient, he reviewed differentiate-then-discretize and discretize-then-differentiate approach, and chose the latter which is also my choice. He does numerical experiments on reservoir with both determined and uncertain properties. In the presence of uncertainty, he adds a parameter identification method combined with Kalman Filter in a closed-loop approach. The grid size in the experiments are below 10,000. However, by the rapid development of computing hardware and software, I can handle models with 1,000,000 grid cells.

In 2005, Sarma, Aziz, and Durlafsky [SAD05], use an adjoint based gradient method to solve water flooding like problems. Their implementation of the adjoint

method reuses some intermediate computation results from the forward simulation, such as Jacobians, and avoids some repetitive computation by using some extra storage. This idea appeared in [ZAZP96] too.

In 2006, Lorentzen, Berg, Naevdal, and Vefring [LBN<sup>+</sup>06] creatively used the ensemble Kalman Filter [Eve03] as a new optimization tool. The method treats the reservoir simulator as a black box and does not require the implementation of adjoint computation. Essentially, it exploits the statistical correlation between well controls and objective value, which are both encoded in the Kalman Filter state variable. If one is to use statistical power, one needs an appropriate sample size. Thousands of reservoir simulations are required since in each optimization iteration a simulation must be done for each control setting in the ensemble (of size 100), which resulted in small reservoir grid being used. They apply regularization at a certain interval in optimization iterations and so does [FR87].

In 2009, Masoud Asadollahi and Geir Naevdal [AN09] use the adjoint method to compute the gradient and then applied steepest descent and nonlinear conjugate gradient method to do optimization. The approach is very similar to my choice. In my work, I added parallelism in the time dimension to accelerate the adjoint based gradient-type approach. In [AN09], among bottomhole pressure, oil and liquid production rates, the paper claims well liquid rates are the best control variables used for maximizing net present value using gradient based methods. This is one of the reasons why I use well injection/production liquid rates as control variables.

In 2010, Wiegand [Wie10] performed PDE based water flooding optimization. He used a finite volume simulation and active set Newton method to solve nonlinear programs with mixed linear constraints. I also use similar finite volume based simulation. But instead of active set method, I apply projected gradient method to do optimization. The work is in small 2D grid as a single layer of SPE 10 model (13,200 cells) and number of wells are in order of 10. By the help of MPI parallel computing and appropriate linear solver preconditioners, my work is able to handle bigger problems

as full 3D SPE 10 model (1,122,000 cells) with hundreds of wells.

In 2011, Jansen [Jan11] reviewed the development of adjoint based optimization of multi phase flow. Jansen stated the optimization of a economic objective function governed by a multi phase flow reservoir model which is the problem I am solving in this project. He also viewed the gradient computation from an optimal control point of view as the origin of the adjoint method. In terms of the handling of inequality constraints, he introduced previous work including gradient projection method, generalized reduced gradient method, constraint lumping method, augmented Lagrangian method, and barrier method. He explained the phenomenon of Bang-Bang control in which the optimal control well rate is either at the maximum or minimum well rate bound. I also observe similar behavior in my project, but the Bang-Bang optimal control conditions are not satisfied in my work and thus I can not make use of this feature to improve optimization performance. He also mentioned the possibility to apply reduced order modeling in this optimization for the very limited number of flow patterns that can be controlled. At last, he talked about strategies against reservoir uncertainty as history matching, closed-loop reservoir management, and robust optimization towards realistic operational use.

In 2011, Echeverría, Isebor, and Durlofsky [ECID11] studied derivative-free methods of well control optimization since in some occasions it is difficult to obtain derivative information from the reservoir simulator or to maintaining adjoint code after the simulation is modified. They compared the optimization performance of pattern search method, generalized pattern search method, genetic algorithm (these three are naturally parallelizable), and Hooke-Jeeves direct search. Specifically, they addressed the handling of nonlinear constraints by derivative-free methods by penalty functions and filter method. To keep the derivative-free method computational tractable, they used small grids (1,000 to 10,000 cells) with number of optimization variables below 100 so thousands of simulations, as showed in results, are possible. Though, my project is gradient based, this work provides some future work direction, e.g. hybridize

gradient methods with genetic algorithm in a distributed computing environment.

In 2012, Chen, Li, and Reynolds [CLR<sup>+</sup>12] performed robust optimization (optimize the expectation of objective function over a set of reservoir models) on both reservoir's life cycle long term Net Present Value and short term production. The bound constraints are enforced by the optimizer and nonlinear constraints are incorporated by augmented Lagrangian method. They claim that after the long term optimization, some control variable can still be varied to optimize the short term production at the same time. By a case study, they showed how they incorporated the long term optimization results as an inequality constraint and then applied a short term optimization to did keep the long term objective roughly the same while improving the short term objective. In my experiments, I also found there are wide range of well rates schedule that gives similar long term revenue. The work [CLR<sup>+</sup>12] provides an idea how to utilize the possibility of choosing among these well rates to achieve other goals at the same time of maintaining the long term goal.

In 2014, Kourounis, Durlofsky, Jansen, and Aziz [KDJA14] worked on formulating the adjoint and on handling constraints in oil-gas compositional flow. This is more challenging, since the more complicated simulation also led to more complex adjoint computation, than the oil-water flow problem of my choice. They used automatic differentiation to facilitate calculating the derivatives needed in adjoint computation. I also used automatic differentiation in the gradient computation but only in very limited area. However, the level of their automatic differentiation applied was not explained in detail. During the forward simulation, they saved the converged states on disk and later read back during adjoint computation. I applied a similar idea and in addition, I also stored the Jacobian matrix used in the Newton steps of solving saturation equation to reuse in adjoint computation, as also done in [SAD05]. To prevent residual error from polluting the gradients, they set a tighter termination criterion for the adjoint equation linear solves than in the forward solve. In the adjoint solves residuals are reduced by ten orders of magnitude compared to five orders of

magnitude in the forward solve. They used SNOPT [GMS06] to implement the SQP nonlinear optimization. They also proposed an innovative heuristic approach handling the nonlinear constraints not in the optimizer but in the simulation. However, I am not dealing with nonlinear constraints in this project.

## Chapter 3

# Parallel-In-Time Gradient-Type Method in Linear-Quadratic Problems

In this chapter, I develop the parallel-in-time gradient-type method in the framework of a discretized linear-quadratic version (3.1.1) of the general problem (1.0.1).

This chapter is organized as follows. In Section 3.1, I define a set of compact matrix/vector notation in the classical gradient method. In Section 3.2, I give the parallel-in-time gradient-type method algorithm, organize it in the same compact notation. In Section 3.3, I interpret the proposed method to be a  $(2N - 1)$ -part splitting iteration scheme. In Section 3.4, I generalize the original parallel-in-time gradient-type method to allow different time subdomain partitions for forward/back computation and overlapping subdomains. In Section 3.5, I prove the convergence of the generalized parallel-in-time gradient-type method by linear algebra argument showing the spectral radius of its implicitly constructed iteration matrix is strictly less than 1 with step size less than a threshold. In Section 3.6, I present numerical examples in optimal control problems governed by 1D and 3D advection-diffusion-reaction systems.

### 3.1 Gradient Method

In this section, I introduce the model optimization problem, a set of compact notation for gradient-type optimization, and the classical gradient method as a foundation on which I will develop the parallel-in-time gradient-type method.

As the discretization of a linear-quadratic (affine linear state equations, quadratic objective function) problem in the form of (1.0.1), I consider DTOC problems with state and control variables  $y_0, \dots, y_K \in \mathbb{R}^{n_y}$  and  $u_0, \dots, u_{K-1} \in \mathbb{R}^{n_u}$ . Given symmetric positive semidefinite matrices  $Q_1, \dots, Q_K \in \mathbb{R}^{n_y \times n_y}$ , symmetric positive definite matrices  $R_0, \dots, R_{K-1} \in \mathbb{R}^{n_u \times n_u}$ , matrices  $A_0, \dots, A_{K-1} \in \mathbb{R}^{n_y \times n_y}$ ,  $B_0, \dots, B_{K-1} \in \mathbb{R}^{n_y \times n_u}$ , and vectors  $d_1, \dots, d_K, c_0, \dots, c_{K-1} \in \mathbb{R}^{n_y}$ ,  $e_0, \dots, e_{K-1} \in \mathbb{R}^{n_u}$ , the DTOC problem is given by

$$\text{minimize } \sum_{k=1}^K \left[ \frac{1}{2} y_k^T Q_k y_k + d_k^T y_k + \frac{1}{2} u_{k-1}^T R_{k-1} u_{k-1} + e_{k-1}^T u_{k-1} \right] \quad (3.1.1a)$$

$$\text{subject to } y_0 = y_{\text{given}}, \quad (3.1.1b)$$

$$y_{k+1} = A_k y_k + B_k u_k + c_k, \quad k = 0, \dots, K-1. \quad (3.1.1c)$$

I can use the constrains (3.1.1c) to express  $y_k$  as a function of  $u_0, \dots, u_{k-1}$ . This leads to the following unconstrained formulation of (3.1.1).

$$\text{Minimize}_{u_0, \dots, u_{K-1}} J(u_0, \dots, u_{K-1}), \quad (3.1.2a)$$

where

$$\begin{aligned} J(u_0, \dots, u_{K-1}) = & \sum_{k=1}^K \left[ \frac{1}{2} y_k(u_0, \dots, u_{k-1})^T Q_k y_k(u_0, \dots, u_{k-1}) + d_k^T y_k(u_0, \dots, u_{k-1}) \right. \\ & \left. + \frac{1}{2} u_{k-1}^T R_{k-1} u_{k-1} + e_{k-1}^T u_{k-1} \right]. \end{aligned} \quad (3.1.2b)$$

The problems (3.1.1) and (3.1.2) are equivalent.

It is well-known that the gradient of  $J$  can be computed using the adjoint equation approach [Ber99, Sec. 1.9], [Pol71, Sec. 2.4]. Given controls  $u_0, \dots, u_{K-1}$ , the gradient can be computed as follows.

Compute  $y_0, \dots, y_K$  by solving

$$y_0 = y_{\text{given}}, \quad (3.1.3a)$$

$$y_{k+1} = A_k y_k + B_k u_k + c_k \quad \text{for } k = 0, \dots, K-1. \quad (3.1.3b)$$

Compute  $p_0, \dots, p_{K-1}$  by solving

$$p_{K-1} = d_K + Q_K y_K, \quad (3.1.3c)$$

$$p_{k-1} = d_k + Q_k y_k + A_k^T p_k \quad \text{for } k = K-1, \dots, 1. \quad (3.1.3d)$$

The gradient is given by

$$\nabla_{u_k} J(u_0, \dots, u_{K-1}) = R_k u_k + e_k + B_k^T p_k \quad \text{for } k = 0, \dots, K-1. \quad (3.1.3e)$$

One step of the gradient method is listed in Algorithm 1. I use subscripts  $k$  to denote time steps and superscripts  $(j)$  to denote the iteration in the gradient method.

---

**Algorithm 1**  $j$ th iteration of the gradient method with step size  $\alpha > 0$

---

- 1: Given control  $u_0^{(j)}, \dots, u_{K-1}^{(j)}$ , and initial state  $y_0^{(j)} = y_{\text{given}}$ .
  - 2: **for**  $k = 0, \dots, K-1$  **do** ▷ solve state equation forward in time
  - 3:     Compute  $y_{k+1}^{(j)} = A_k y_k^{(j)} + B_k u_k^{(j)} + c_k$
  - 4: **end for**
  - 5: Compute  $p_{K-1}^{(j)} = d_K + Q_K y_K^{(j)}$  ▷ solve adjoint equation backward in time
  - 6: **for**  $k = K-1, \dots, 1$  **do**
  - 7:     Compute  $p_{k-1}^{(j)} = d_k + Q_k y_k^{(j)} + A_k^T p_k^{(j)}$
  - 8: **end for**
  - 9: **for**  $k = 0, \dots, K-1$  **do** ▷ update control using negative gradient
  - 10:      $u_k^{(j+1)} = u_k^{(j)} - \alpha(R_k u_k^{(j)} + e_k + B_k^T p_k^{(j)})$
  - 11: **end for**
- 

For the following presentation, it will be helpful to develop a set of compact matrix/vector notation for the quantities used in the gradient method. I define vectors



$$\mathbf{u} \stackrel{\text{def}}{=} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{K-1} \end{bmatrix} \in \mathbb{R}^{Kn_u}, \quad \mathbf{y} \stackrel{\text{def}}{=} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix}, \quad \mathbf{p} \stackrel{\text{def}}{=} \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{K-1} \end{bmatrix}, \quad \mathbf{y}_0 \stackrel{\text{def}}{=} \begin{bmatrix} A_0 y_0 \\ A_1 A_0 y_0 \\ \vdots \\ A_{K-1} \dots A_0 y_0 \end{bmatrix} \in \mathbb{R}^{Kn_y}, \quad (3.1.4a)$$

and the matrix

$$\mathbf{L} \stackrel{\text{def}}{=} \left[ \begin{array}{c|c|c|c|c|c|c} I & & & & & & \\ & A_1 & & & & & \\ & A_2 A_1 & I & & & & \\ & A_3 A_2 A_1 & A_2 & I & & & \\ & \vdots & A_3 A_2 & A_3 & I & & \\ & A_{K-1} A_{K-2} \dots A_1 & A_{K-1} A_{K-2} \dots A_2 & \dots & \dots & \ddots & \\ & & & & & A_{K-1} & I \end{array} \right] \in \mathbb{R}^{Kn_y \times Kn_y}, \quad (3.1.4b)$$

Note that  $\mathbf{L}$  is not explicitly constructed in gradient method and in the later proposed parallel-in-time gradient-type method. Also note that

$$\mathbf{L} = \left[ \begin{array}{c|c|c|c|c|c|c} I & & & & & & \\ -A_1 & I & & & & & \\ & -A_2 & I & & & & \\ & & -A_3 & I & & & \\ & & & & I & & \\ & & & & & \ddots & \\ & & & & & -A_{K-1} & I \end{array} \right]^{-1}.$$

Define the constant quantities,

$$\mathbf{Q} \stackrel{\text{def}}{=} \text{diag}(Q_1, \dots, Q_K), \quad \mathbf{R} \stackrel{\text{def}}{=} \text{diag}(R_0, \dots, R_{K-1}), \quad \mathbf{B} \stackrel{\text{def}}{=} \text{diag}(B_0, \dots, B_{K-1}), \quad (3.1.4c)$$

$$\mathbf{c} \stackrel{\text{def}}{=} (c_0^T, \dots, c_{K-1}^T)^T, \quad \mathbf{d} \stackrel{\text{def}}{=} (d_1^T, \dots, d_K^T)^T, \quad \mathbf{e} \stackrel{\text{def}}{=} (e_0^T, \dots, e_{K-1}^T)^T. \quad (3.1.4d)$$

The state computations (3.1.3a,b) can be written as

$$\mathbf{y} = \mathbf{L}(\mathbf{B}\mathbf{u} + \mathbf{c}) + \mathbf{y}_0, \quad (3.1.5)$$

the adjoint computations (3.1.3c,d) can be written as

$$\mathbf{p} = \mathbf{L}^T(\mathbf{Q}\mathbf{y} + \mathbf{d}), \quad (3.1.6)$$

and the gradient (3.1.3e) is given by  $\nabla J(\mathbf{u}) = \mathbf{R}\mathbf{u} + \mathbf{e} + \mathbf{B}^T\mathbf{p}$ . If I insert (3.1.6) and (3.1.5) into the expression for  $\nabla J(\mathbf{u})$ , then

$$\begin{aligned} \nabla J(\mathbf{u}) &= \mathbf{R}\mathbf{u} + \mathbf{e} + \mathbf{B}^T\mathbf{p} \\ &= [\mathbf{R} + (\mathbf{L}\mathbf{B})^T\mathbf{Q}(\mathbf{L}\mathbf{B})]\mathbf{u} + \mathbf{e} + (\mathbf{L}\mathbf{B})^T\mathbf{Q}(\mathbf{L}\mathbf{c} + \mathbf{y}_0) + (\mathbf{L}\mathbf{B})^T\mathbf{d} \\ &= \mathbf{H}\mathbf{u} + \mathbf{g}, \end{aligned} \quad (3.1.7)$$

where

$$\mathbf{H} \stackrel{\text{def}}{=} \mathbf{R} + (\mathbf{L}\mathbf{B})^T\mathbf{Q}(\mathbf{L}\mathbf{B}), \quad \mathbf{g} \stackrel{\text{def}}{=} \mathbf{e} + (\mathbf{L}\mathbf{B})^T\mathbf{Q}(\mathbf{L}\mathbf{c} + \mathbf{y}_0) + (\mathbf{L}\mathbf{B})^T\mathbf{d}. \quad (3.1.8)$$

Since  $R_0, \dots, R_{K-1} \in \mathbb{R}^{n_u \times n_u}$  symmetric positive definite and  $Q_1, \dots, Q_K \in \mathbb{R}^{n_y \times n_y}$  are symmetric positive semidefinite,  $\mathbf{H}$  is symmetric positive definite. Alternatively, I can insert (3.1.5) into (3.1.2b) to obtain

$$J(\mathbf{u}) = \frac{1}{2}\mathbf{u}^T\mathbf{H}\mathbf{u} + \mathbf{g}^T\mathbf{u} + \text{const.} \quad (3.1.9)$$

and derive the expression (3.1.7) by taking the gradient.

I also recall that the gradient method with constant step-size  $\alpha$ , in this context also known as Richardson's iteration,

$$\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} - \alpha(\mathbf{H}\mathbf{u}^{(j)} + \mathbf{g}), \quad (3.1.10)$$

is an iterative method derived from the splitting  $\mathbf{H} = \alpha^{-1}\mathbf{I} - (\alpha^{-1}\mathbf{I} - \mathbf{H})$ . Let  $0 < \lambda_{\min}(\mathbf{H}) \leq \lambda_{\max}(\mathbf{H})$  denote the smallest and largest eigenvalue of  $\mathbf{H}$ . The spectral radius of the iteration matrix  $\mathbf{I} - \alpha\mathbf{H}$  is less than one for all step sizes  $0 < \alpha < 2/\lambda_{\max}(\mathbf{H})$ . The smallest spectral radius of  $\mathbf{I} - \alpha\mathbf{H}$ ,

$$\rho^* = \frac{\lambda_{\max}(\mathbf{H}) - \lambda_{\min}(\mathbf{H})}{\lambda_{\max}(\mathbf{H}) + \lambda_{\min}(\mathbf{H})}$$

is achieved with step size

$$\alpha^* \stackrel{\text{def}}{=} \frac{2}{\lambda_{\max}(\mathbf{H}) + \lambda_{\min}(\mathbf{H})} \quad (3.1.11)$$

## 3.2 Derivation of the Parallel-In-Time Gradient-Type Method

The gradient method in Algorithm 1 requires a full forward-in-time state solve followed by a full backward-in-time adjoint solve before the control can be updated. The parallel-in-time gradient method splits the time indices into  $N$  subsets, where the  $i$ th one is given by  $\{K_i, \dots, K_{i+1}\}$ .  $i = 0, \dots, N - 1$ , and the time indices satisfy  $0 = K_0 < K_1 < \dots < K_N = K$ . Since the indices  $K_i, \dots, K_{i+1}$  correspond to time steps, I refer to  $\{K_i, \dots, K_{i+1}\}$  as the  $i$ th time subdomain, and refer to the indices  $K_i, K_{i+1}$  as the left/right time subdomain boundary.

The idea of the parallel-in-time gradient-type method is simple. Suppose I am given the current control on the  $i$ th time subdomain,  $u_{K_i}^{(j)}, u_{K_{i+1}}^{(j)}, \dots, u_{K_{i+1}-1}^{(j)}$ . If I knew the state information  $y_{K_i}^{(j)}$  at the left time subdomain boundary  $K_i$  and the adjoint information  $p_{K_{i+1}}^{(j)}$  at the right time subdomain boundary  $K_{i+1}$ , then I can compute  $y_k^{(j)}$ ,  $k = K_i + 1, \dots, K_{i+1}$ , the adjoints  $p_k^{(j)}$ ,  $k = K_{i+1} - 1, \dots, K_i$ , and then update the controls with indices  $K_i, K_i + 1, \dots, K_{i+1} - 1$ . Since  $y_{K_i}^{(j)}$  and  $p_{K_{i+1}}^{(j)}$  are only available through an entire state and adjoint solve, I use the values from the previous iteration, and exchange these values after time subdomain computations for state and adjoint information are completed. Thus for the  $i$ th time subdomain,  $i \in \{1, \dots, N - 2\}$ , I proceed as follows (for  $i = 0$  and  $i = N - 1$  initial state information or final adjoint information is given): Given  $u_{K_i}^{(j)}, u_{K_{i+1}}^{(j)}, \dots, u_{K_{i+1}-1}^{(j)}$  and  $y_{K_i}^{(j-1)}$  and  $p_{K_{i+1}}^{(j-1)}$  I first compute  $i$ th time subdomain states using

$$y_{K_{i+1}}^{(j)} = A_{K_i} y_{K_i}^{(j-1)} + B_{K_i} u_{K_i}^{(j)} + c_{K_i}, \quad (3.2.1a)$$

$$y_{k+1}^{(j)} = A_k y_k^{(j)} + B_k u_k^{(j)} + c_k, \quad k = K_i + 1, \dots, K_{i+1} - 1. \quad (3.2.1b)$$

Next I compute  $i$ th time subdomain adjoints using

$$p_{K_{i+1}-1}^{(j)} = Q_{K_{i+1}} y_{K_{i+1}}^{(j)} + d_{K_{i+1}} + A_{K_{i+1}}^T p_{K_{i+1}}^{(j-1)}, \quad (3.2.1c)$$

$$p_{k-1}^{(j)} = Q_k y_k^{(j)} + d_k + A_k^T p_k^{(j)}, \quad k = K_{i+1} - 1, \dots, K_i + 1. \quad (3.2.1d)$$

Then I update the  $i$ th time subdomain controls using

$$u_k^{(j+1)} = u_k^{(j)} - \alpha(R_k u_k^{(j)} + e_k + B_k^T p_k^{(j)}), \quad k = K_i, \dots, K_{i+1} - 1. \quad (3.2.1e)$$

Finally, I send  $y_{K_{i+1}}^{(j)}$  to processor  $i + 1$  and  $p_{K_i}^{(j)}$  to processor  $i - 1$ , and I receive  $y_{K_i}^{(j)}$  from processor  $i - 1$  and  $p_{K_{i+1}}^{(j)}$  from processor  $i + 1$ .

The complete statement, taking into account the modifications for time subdomains  $i = 0$  and  $i = N - 1$  is given in Algorithm 2. Since at given controls  $u_0^{(j)}, \dots, u_{K-1}^{(j)}$  the state equation (3.1.3a,b) and the adjoint equation (3.1.3c,d) are not satisfied (there are ‘jumps’ at the time subdomain boundaries  $K_1, \dots, K_{N-1}$ ) when Algorithm 2 is used, I call it ‘gradient-type’.

### 3.3 Interpretation as a $(2N - 1)$ -Part Iteration Scheme

In this section, I express the parallel gradient-type method using the compact matrix/vector notations developed in Section 3.1, which will allow us to interpret this new method as a  $(2N - 1)$ -part iteration scheme corresponding to (3.1.10) and thus form the basis of the convergence proof.

Impatient readers may skip part of the derivation and go directly to the result in (3.3.7) and (3.3.8).

I define the ordered product of matrices

$$\prod_{h=i}^j A_h \stackrel{\text{def}}{=} \begin{cases} A_j A_{j-1} \times \dots \times A_{i+1} A_i, & i \leq j, \\ I, & i > j. \end{cases} \quad (3.3.1)$$

---

**Algorithm 2**  $j$ th iteration of the parallel-in-time gradient-type method with step size  $\alpha > 0$ . Describes the tasks executed by processor of rank  $i \in \{0, \dots, N - 1\}$

---

- 1: Input control  $u_{K_i}^{(j)}, u_{K_i+1}^{(j)}, \dots, u_{K_{i+1}-1}^{(j)}$ . ▷ initialization of the iteration
- 2: **if**  $i > 0$  and  $j = 0$  **then**
- 3:   Input initial  $y_{K_i}^{(-1)}$
- 4: **end if**
- 5: **if**  $i < N - 1$  and  $j = 0$  **then**
- 6:   Input initial  $p_{K_{i+1}}^{(-1)}$
- 7: **end if**
  
- 8: **if**  $i = 0$  **then** ▷ solve the state equation forward in time
- 9:    $y_{K_i+1}^{(j)} = A_{K_i} y_{\text{given}} + B_{K_i} u_{K_i}^{(j)} + c_{K_i}$
- 10: **else**
- 11:    $y_{K_i+1}^{(j)} = A_{K_i} y_{K_i}^{(j-1)} + B_{K_i} u_{K_i}^{(j)} + c_{K_i}$
- 12: **end if**
- 13: **for**  $k = K_i + 1, \dots, K_{i+1} - 1$  **do**
- 14:    $y_{k+1}^{(j)} = A_k y_k^{(j)} + B_k u_k^{(j)} + c_k$
- 15: **end for**
  
- 16: **if**  $i = N - 1$  **then** ▷ solve the adjoint equation backward in time
- 17:    $p_{K_{i+1}-1}^{(j)} = Q_{K_{i+1}} y_{K_{i+1}}^{(j)} + d_{K_{i+1}}$
- 18: **else**
- 19:    $p_{K_{i+1}-1}^{(j)} = Q_{K_{i+1}} y_{K_{i+1}}^{(j)} + d_{K_{i+1}} + A_{K_{i+1}}^T p_{K_{i+1}}^{(j-1)}$
- 20: **end if**
- 21: **for**  $k = K_{i+1} - 1, \dots, K_i + 1$  **do**
- 22:    $p_{k-1}^{(j)} = Q_k y_k^{(j)} + d_k + A_k^T p_k^{(j)}$
- 23: **end for**

(continued on next page)

---

---

24: **for**  $k = K_i, \dots, K_{i+1} - 1$  **do** ▷ update control

25:      $u_k^{(j+1)} = u_k^{(j)} - \alpha(R_k u_k^{(j)} + e_k + B_k^T p_k^{(j)})$

26: **end for**

27: **if**  $i > 0$  **then** ▷ communication between processors

28:     send  $p_{K_i}^{(j)}$  to rank  $i - 1$

29:     receive  $y_{K_i}^{(j)}$  from rank  $i - 1$

30: **end if**

31: **if**  $i < N - 1$  **then**

32:     send  $y_{K_{i+1}}^{(j)}$  to rank  $i + 1$

33:     receive  $p_{K_{i+1}}^{(j)}$  from rank  $i + 1$

34: **end if**

---

Using this notation and recursive substitutions of (3.2.1a,b) I obtain the following. For  $j \geq 0$ ,  $0 \leq i \leq N - 1$ , and  $1 \leq s \leq K_{i+1} - K_i$ , note that  $y_{K_i}^{(-1)}$  is initialized when the algorithm starts,

$$y_{K_i+s}^{(j)} = \left[ \prod_{h=K_i}^{K_i+s-1} A_h \right] y_{K_i}^{(j-1)} + \sum_{t=0}^{s-1} \left[ \prod_{h=K_i+s-t}^{K_i+s-1} A_h \right] (B_{K_i+s-t-1} u_{K_i+s-t-1}^{(j)} + c_{K_i+s-t-1}). \quad (3.3.2)$$

For  $j \geq 1$  and  $i \geq 1$ , applying (3.3.2) with  $j$  replaced by  $j - 1$  and  $K_i$  replaced by  $K_{i-1}$ , and with  $s = K_i - K_{i-1}$  gives

$$y_{K_i}^{(j-1)} = \left[ \prod_{h=K_{i-1}}^{K_i-1} A_h \right] y_{K_{i-1}}^{(j-2)} + \sum_{t=0}^{K_i-K_{i-1}-1} \left[ \prod_{h=K_i-t}^{K_i-1} A_h \right] (B_{K_i-t-1} u_{K_i-t-1}^{(j-1)} + c_{K_i-t-1}). \quad (3.3.3)$$

Inserting (3.3.3) into (3.3.2) gives for  $j \geq 1$  and  $i \geq 1$ ,

$$\begin{aligned}
y_{K_i+s}^{(j)} &= \left[ \prod_{h=K_{i-1}}^{K_i+s-1} A_h \right] y_{K_{i-1}}^{(j-2)} + \sum_{t=0}^{K_i-K_{i-1}-1} \left[ \prod_{h=K_i-t}^{K_i+s-1} A_h \right] (B_{K_i-t-1} u_{K_i-t-1}^{(j-1)} + c_{K_i-t-1}) \\
&\quad + \sum_{t=0}^{s-1} \left[ \prod_{h=K_i+s-t}^{K_i+s-1} A_h \right] (B_{K_i+s-t-1} u_{K_i+s-t-1}^{(j)} + c_{K_i+s-t-1}).
\end{aligned} \tag{3.3.4}$$

Repeating the previous steps leads to the following expression for  $j \geq i$ ,

$$\begin{aligned}
y_{K_i+s}^{(j)} &= \left[ \prod_{h=0}^{K_i+s-1} A_h \right] y_{\text{given}} \\
&\quad + \sum_{d=1}^i \sum_{t=0}^{K_{i-d+1}-K_{i-d}-1} \left[ \prod_{h=K_{i-d+1}-t}^{K_i+s-1} A_h \right] (B_{K_{i-d+1}-t-1} u_{K_{i-d+1}-t-1}^{(j-d)} + c_{K_{i-d+1}-t-1}) \\
&\quad + \sum_{t=0}^{s-1} \left[ \prod_{h=K_i+s-t}^{K_i+s-1} A_h \right] (B_{K_i+s-t-1} u_{K_i+s-t-1}^{(j)} + c_{K_i+s-t-1}).
\end{aligned} \tag{3.3.5}$$

Rearranging (3.3.5) into matrix-vector product form gives, for  $j \geq i$ ,

$$\begin{aligned}
y_{K_i+s}^{(j)} &= \left[ \prod_{h=0}^{K_i+s-1} A_h \right] y_{\text{given}} \\
&+ \sum_{d=1}^i \left[ \prod_{h=K_{i-d}+1}^{K_i+s-1} A_h, \prod_{h=K_{i-d}+2}^{K_i+s-1} A_h, \dots, \prod_{h=K_{i-d+1}}^{K_i+s-1} A_h \right] \begin{bmatrix} B_{K_{i-d}} u_{K_{i-d}}^{(j-d)} + c_{K_{i-d}} \\ B_{K_{i-d}+1} u_{K_{i-d}+1}^{(j-d)} + c_{K_{i-d}+1} \\ \vdots \\ B_{K_{i-d+1}-1} u_{K_{i-d+1}-1}^{(j-d)} + c_{K_{i-d+1}-1} \end{bmatrix} \\
&+ \left[ \prod_{h=K_i+1}^{K_i+s-1} A_h, \prod_{h=K_i+2}^{K_i+s-1} A_h, \dots, \prod_{h=K_i+s-1}^{K_i+s-1} A_h, I \right] \begin{bmatrix} B_{K_i} u_{K_i}^{(j)} + c_{K_i} \\ B_{K_i+1} u_{K_i+1}^{(j)} + c_{K_i+1} \\ \vdots \\ B_{K_i+s-1} u_{K_i+s-1}^{(j)} + c_{K_i+s-1} \end{bmatrix} \quad (3.3.6a) \\
&= \left[ \prod_{h=0}^{K_i+s-1} A_h \right] y_{\text{given}} \\
&+ \sum_{d=1}^i \left[ \overbrace{0, \dots, 0}^{K_{i-d} \text{ zero blocks}}, \prod_{h=K_{i-d}+1}^{K_i+s-1} A_h, \prod_{h=K_{i-d}+2}^{K_i+s-1} A_h, \dots, \prod_{h=K_{i-d+1}}^{K_i+s-1} A_h, \overbrace{0, \dots, 0}^{(K-K_{i-d+1}) \text{ zero blocks}} \right] (\mathbf{B}u^{(j-d)} + \mathbf{c}) \\
&+ \left[ \overbrace{0, \dots, 0}^{K_i \text{ zero blocks}}, \prod_{h=K_i+1}^{K_i+s-1} A_h, \prod_{h=K_i+2}^{K_i+s-1} A_h, \dots, \prod_{h=K_i+s-1}^{K_i+s-1} A_h, I, \overbrace{0, \dots, 0}^{(K-K_i)-s \text{ zero blocks}} \right] (\mathbf{B}u^{(j)} + \mathbf{c}). \quad (3.3.6b)
\end{aligned}$$

In (3.3.6b) each zero block is of size  $n_y \times n_y$ . Notice that the row vectors in (3.3.6b) are row blocks of  $\mathbf{L}$ .

Equation (3.3.6b) represents the  $(K_i + s)$ th block (of length  $n_y$ ) component of the vector  $\mathbf{y}^{(j)}$ . To combine (3.3.6b) into a representation for the entire vector  $\mathbf{y}^{(j)}$  I define matrices  $\mathbf{I}_{-d} \in \{0, 1\}^{K n_y \times K n_y}$ ,  $d = 0, \dots, N - 1$  as follows. I use Matlab notation to indicate submatrices. The matrix  $\mathbf{I}_{-d}$  is a matrix full of zeros except for the submatrices

$$\mathbf{I}_{-d}(K_{i-1}n_y + 1 : K_i n_y, K_{i-1-d}n_y + 1 : K_{i-d}n_y), \quad i = d + 1, \dots, N,$$

which are matrices of all ones ( i.e. the matrix entries of  $\mathbf{I}_{-d}$  in the intersection of rows  $K_{i-1}n_y + 1$  to  $K_i n_y$  and of columns  $K_{i-1-d}n_y + 1$  to  $K_{i-d}n_y$  are equal to one). See Figure 3.1 for an illustration. In a later Section 3.4, I present Algorithm 11



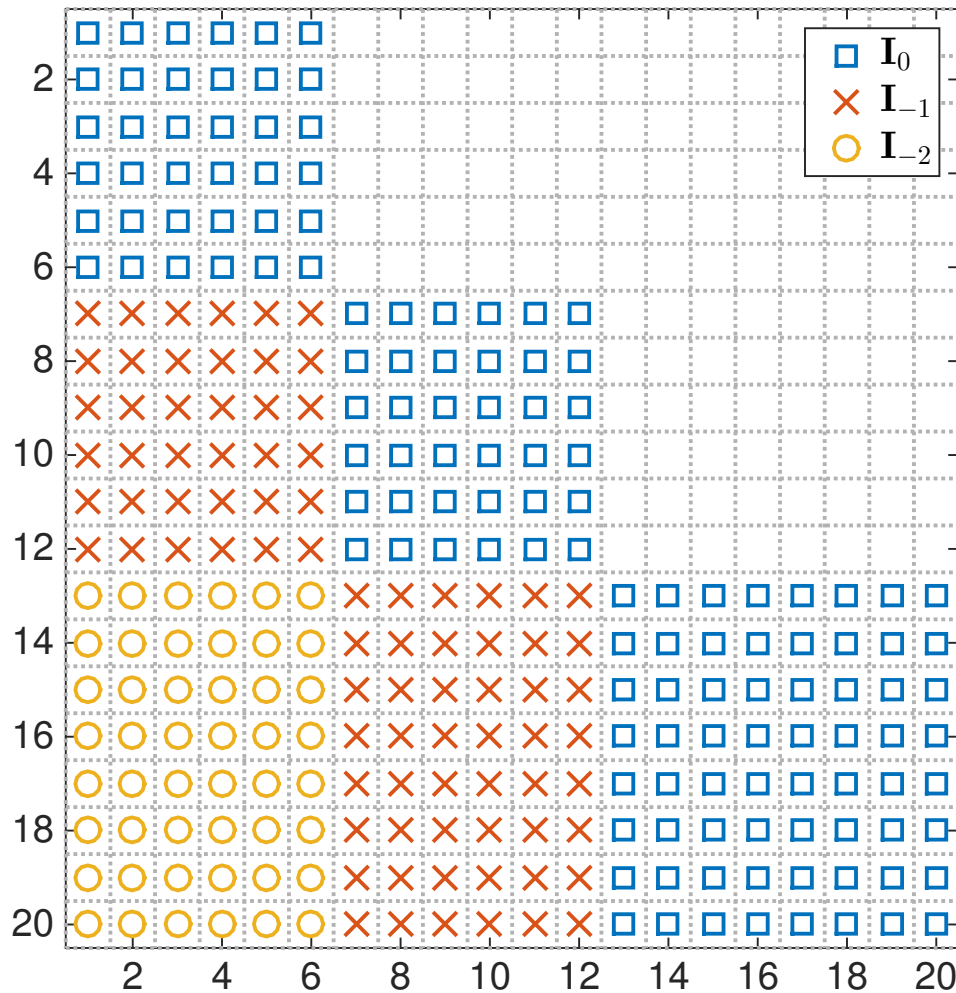


Figure 3.1: Illustration of the positions of ‘1’s in  $\mathbf{I}_0$ ,  $\mathbf{I}_{-1}$  and  $\mathbf{I}_{-2}$  in an example where the state dimension is  $n_y = 2$ , the  $K = 10$  time steps are split into  $N = 3$  subdomains with  $K_0 = 0, K_1 = 3, K_2 = 6, K_3 = 10$ .

and Algorithm 12 for construction of this type of matrices. Note that there are no overlapping nonzero entries for matrices  $\mathbf{I}_{-d}$  for  $d = 0, \dots, N - 1$  and that  $\sum_{d=0}^{N-1} \mathbf{I}_{-d}$  has entire ‘1’ in all positions where  $\mathbf{L}$  is nonzero.

Now, let ‘ $\circ$ ’ refer to the Hadamard product (entry-wise product), then (3.3.6) can be written as

$$\mathbf{y}^{(j)} = \sum_{d=0}^{N-1} (\mathbf{I}_{-d} \circ \mathbf{L})(\mathbf{B}\mathbf{u}^{(j-d)} + \mathbf{c}) + \mathbf{y}_0, \quad \text{for } j \geq N-1. \quad (3.3.7)$$

Similar to the derivation of (3.3.7) I can show that the equations (3.2.1c,d) for the adjoints lead to the compact representation

$$\mathbf{p}^{(j)} = \sum_{d=0}^{N-1} (\mathbf{I}_{-d} \circ \mathbf{L})^T (\mathbf{Q}\mathbf{y}^{(j-d)} + \mathbf{d}), \quad \text{for } j \geq N-1. \quad (3.3.8)$$

Using (3.2.1e), (3.3.7) and (3.3.8) gives the following representation of the parallel-in-time gradient type iteration  $j \geq 2N-2$ ,

$$\begin{aligned} \mathbf{u}^{(j+1)} &= \mathbf{u}^{(j)} - \alpha(\mathbf{R}\mathbf{u}^{(j)} + \mathbf{e} + \mathbf{B}^T \mathbf{p}^{(j)}) \\ &= \mathbf{u}^{(j)} - \alpha \left[ \mathbf{R}\mathbf{u}^{(j)} + \sum_{r=0}^{N-1} \sum_{l=0}^{N-1} (\mathbf{I}_{-r} \circ \mathbf{L}\mathbf{B})^T \mathbf{Q}(\mathbf{I}_{-l} \circ \mathbf{L}\mathbf{B})\mathbf{u}^{(j-r-l)} + \mathbf{g} \right]. \end{aligned} \quad (3.3.9)$$

I define

$$\mathbf{H}_0 \stackrel{\text{def}}{=} \mathbf{R} + (\mathbf{I}_0 \circ \mathbf{L}\mathbf{B})^T \mathbf{Q}(\mathbf{I}_0 \circ \mathbf{L}\mathbf{B}), \quad (3.3.10a)$$

$$\mathbf{H}_d \stackrel{\text{def}}{=} \sum_{\substack{l,r \in \{0, \dots, N-1\} \\ l+r=d}} (\mathbf{I}_{-r} \circ \mathbf{L}\mathbf{B})^T \mathbf{Q}(\mathbf{I}_{-l} \circ \mathbf{L}\mathbf{B}), \quad d = 1, \dots, 2N-2. \quad (3.3.10b)$$

Note that the Hessian (3.1.8) can be split as

$$\mathbf{H} = \sum_{d=0}^{2N-2} \mathbf{H}_d. \quad (3.3.11)$$

Inserting (3.3.10) into (3.3.9) gives

$$\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} - \alpha \left( \sum_{d=0}^{2N-2} \mathbf{H}_d \mathbf{u}^{(j-d)} + \mathbf{g} \right). \quad (3.3.12)$$

The presentation (3.3.12) reveals that the parallel-in-time gradient-type method is a  $(2N-1)$ -part iteration scheme as defined in [de 76], [dN81] derived from the  $(2N-1)$ -part additive splitting

$$\mathbf{H} = \alpha^{-1} \mathbf{I} - (\alpha^{-1} \mathbf{I} - \mathbf{H}_0) - (-\mathbf{H}_1) - \dots - (-\mathbf{H}_{2N-2}),$$

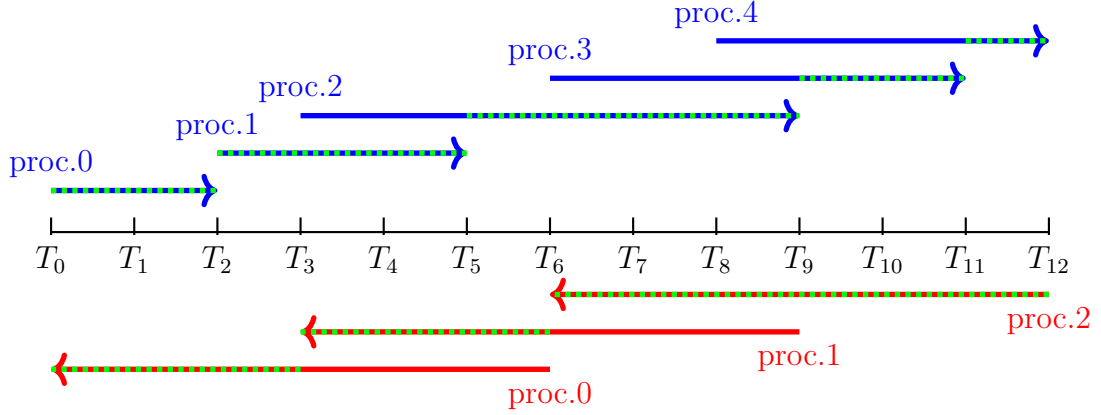


Figure 3.2: Illustration of Generalized Parallel-In-Time Gradient-Type Method. Blue(red) arrows represent time subdomains for parallel forward(backward) computation. The regions in different processes shaded by the green dots are aggregated and collectively constitute the resulting state/adjoint on the whole time domain for use in subsequent computations. Its time subdomain representation is in Table 3.1.

which results from adding/subtracting  $\alpha^{-1}\mathbf{I}$  in (3.3.11).

The convergence results in [de 76] [dN81] are for specific splittings (in notation of this section, for specific matrices  $\mathbf{H}, \mathbf{H}_0, \dots, \mathbf{H}_{2N-2}$ ) only and do not apply to this setting. In Section 3.5, I prove convergence for sufficiently small step-size  $\alpha > 0$  for a broader class of parallel-in-time gradient-type method to be introduced in Section 3.4.

### 3.4 A Generalized Framework for Parallelism

In the previous Section 3.2 and Section 3.3, I introduced the parallel-in-time gradient-type method and how it can be interpreted as a splitting method. With this proper background knowledge, in this section, I can naturally propose a generalized parallel-in-time gradient-type method framework, see Figure 3.2. The generalized parallel-in-time gradient-type method framework is a set of parallel computation rules that, compared to the original parallel-in-time gradient-type method introduced in Sec-

tion 3.2, allows

- different number of parallel computing units in forward/backward computation,
- overlapping computation time subdomains,
- arbitrary length of computation time subdomains,
- different partition of time domain in forward and backward computation.

Computationally, it has potential advantages

- It allows additional computing power to enhance the gradient-type update by improving state/adjoint information propagation;
- in very diffusive systems, overlapping computation subdomains makes it possible to compute nearly exact gradient in parallel

For linear-quadratic optimization problems, the generalized version of the parallel-in-time gradient-type method is provably convergent with a fixed sufficiently small step size. Of course, the original parallel-in-time gradient-type method introduced in Section 3.2 as a special case of this generalized method is also convergent with small step sizes. The proof is given in Section 3.5 after the description of the generalized parallel-in-time gradient-type method in this section.

In Section 3.4.1 and Section 3.4.2, I introduce some new notions in the generalized algorithm. Then, Section 3.4.3 gives the pseudo code algorithm using the notions just introduced. Afterward, Section 3.4.4 and Section 3.4.5 prepare terminologies for the interpretation of the generalized algorithm as a multiple part splitting scheme in Section 3.4.6.

### **3.4.1 Forward/Backward Computation Subdomains**

I first introduce the time subdomain partition.

Start with forward computation. Superscript “F” indicates quantities associated with the forward computation as opposed to “B” for backward computation. The  $i$ th processor, of  $N^F$  processors in total, is responsible for the forward computation from the starting time step  $s_i^F$  to the ending  $e_i^F$ ,  $i = 0, \dots, N^F - 1$ . The computation on  $N^F$  processors collectively covers the whole time domain, i.e.,

$$\cup_{i=0}^{N^F-1} [s_i^F, e_i^F] = [0, K] \quad (3.4.1)$$

Also assume,

$$[s_i^F, e_i^F] \not\subset [s_j^F, e_j^F] \quad \forall i \neq j \quad (3.4.2)$$

otherwise the computation done in the contained set will be in vain since they will be wholly discarded in the later described algorithm. As a consequence of the above assumption,

$$s_i^F \neq s_j^F, e_i^F \neq e_j^F \text{ for } i \neq j \quad (3.4.3)$$

For convenience, assume the time subdomains are ordered,

$$s_i^F < s_j^F, e_i^F < e_j^F \text{ for } 0 \leq i < j \leq N - 1 \quad (3.4.4)$$

Similarly, for backward computation, The  $i$ th processor, of  $N^B$  processors in total, is responsible for the backward computation from the terminal time step  $e_i^B$  to the time step  $s_i^B$  at the beginning of this time subdomain,  $i = 0, \dots, N - 1$ . Assume

$$\cup_{i=0}^{N^B-1} [s_i^B, e_i^B] = [0, K] \quad (3.4.5)$$

and

$$[s_i^B, e_i^B] \not\subset [s_j^B, e_j^B] \quad \forall i \neq j \quad (3.4.6)$$

as a consequence,

$$s_i^B \neq s_j^B, e_i^B \neq e_j^B \text{ for } i \neq j \quad (3.4.7)$$

also assume the time subdomains are ordered,

$$s_i^B < s_j^B, e_i^B < e_j^B \text{ for } 0 \leq i < j \leq M - 1 \quad (3.4.8)$$

$i$	$s_i^F$	$e_i^F$	$s_i^B$	$e_i^B$
0	0	2	0	6
1	2	5	3	9
2	3	9	6	12
3	6	11	N/A	N/A
4	8	12	N/A	N/A

Table 3.1: Subdomain Representation of Figure 3.2

Note that for both forward and backward computation subdomains, the notation  $s$  and  $e$  indicate the smallest and the largest time step indices in the subdomains respectively. However, in forward computation,  $s_i^F$  is where the computation in subdomain  $i$  begins and, in contrast, backward computation begins at  $e_i^B$ .

For the time domain splitting illustrated in Figure 3.2, if  $T_k$  stands for time step  $k$  for  $k = 0, \dots, 12$ , then the subdomain representation is in Table 3.1.

### 3.4.2 Aggregated State/Adjoint Variables

In Section 3.4.1, I defined the computation subdomains. Since there are possibly overlapping subdomains, different processors may compute the state/adjoint variables corresponding to a same time step. It is needed to design a rule to decide which computation result to keep and aggregate these results for subsequent use.

In this section, I introduce the aggregated state/adjoint variables that keeps proper computation results from different processors. This notion of the aggregated state/adjoint in the original algorithm is less obvious is for that

1. there are no overlap in computation domains in which case all forward/backward computation results are kept for subsequent use;
2. forward/backward computation domains coincides and almost all state variables needed to do backward computation are computed by the same processor that

is performing backward computation.

To express this idea in formula, first, I introduce for clarity new notation adding another subscript to the state/adjoint variables.

$$y_{i,k}^{(j)}, i = 0, \dots, N-1, k = 0, \dots, K, j = 0, 1, \dots \quad (3.4.9)$$

which is the state variable of global time step  $k$  on processor  $i$  in iteration  $j$ . Similarly,

$$p_{i,k}^{(j)}, i = 0, \dots, N-1, k = 0, \dots, K-1, j = 0, 1, \dots \quad (3.4.10)$$

for adjoint variables.

After forward computation of all subdomains in iteration  $j$ , define the aggregated state to collect the computation results from all subdomains,

$$y_k^{(j)}, k = 0, \dots, K.$$

The aggregated state is used to store the computation result for later use, as showed in Figure 3.2 by the green dots in forward computation. For initial condition of the whole time domain,

$$y_0^{(j)} = y_{\text{given}} \quad (3.4.11a)$$

For time subdomain index  $i = 0, \dots, N-1$ ,

$$y_k^{(j)} = y_{i,k}^{(j)}, \quad k = e_{i-1}^F + 1, \dots, e_i^F \quad (3.4.11b)$$

with  $e_{-1}^F \stackrel{\text{def}}{=} 0$  for notational simplicity. Because of (3.4.1), all  $y_k^{(j)}$  are defined for  $k = 0, \dots, K$ . These aggregated states in (3.4.11) will be used in the subsequent computations, e.g., to compute adjoint variable, gradient-type vector and to serve as initial value in next forward computation, as described by algorithms in Section 3.4.3.

After backward computation of all subdomains in iteration  $j$ , For the terminal condition of the whole time domain,

$$p_K^{(j)} = 0 \quad (3.4.12a)$$

For time subdomain index  $i = 0, \dots, N - 1$ ,

$$p_k^{(j)} = p_{i,k}^{(j)}, \quad k = s_i^B, \dots, s_{i+1}^B - 1 \quad (3.4.12b)$$

with  $s_N^B \stackrel{\text{def}}{=} K$  for notational simplicity. The aggregated adjoint variables in (3.4.12) will be used in gradient-type vector computation and serve as terminal values for the next backward computation in iteration  $(j + 1)$ .

The heuristic reason why I made the choice of what computation results go into the aggregated variables in (3.4.11b) and (3.4.12b) is that these choice incorporates more most recent information. For example, in Figure 3.2, between computed state variable at time  $T_7$  from proc.2 and proc.3, state from proc.2 is kept in the aggregated state variable since it incorporate the influence of the most recent control variable in  $T_3, T_4, T_5, T_6$  but that from proc.3 only used the most recent control from  $T_6$ .

### 3.4.3 The Algorithm

In this section, I give the algorithm for the generalized parallel-in-time gradient-type method. See Figure 3.3 for an illustration of the workflow. For clarity, the algorithm is given with redundant processors and data transmission to convey the idea of the iterations. The pseudo code to follow is not optimized in the following ways:

- The algorithm is written in the way that a group of processors is responsible for forward computing, another group of processors is responsible for backward computing, and a separate individual processor, which I call “central processor”, is performing the data aggregation and control update. The forward computing and backward computing groups have separate ranks. Forward computing processors have ranks  $\{0, \dots, N^F - 1\}$ ; backward computing processors have ranks  $\{0, \dots, N^B - 1\}$ . Since the two rank index sets above share some common indices, when addressing the data communication, I will emphasize if it is between the central processor and the forward computing group or between the central processor and the backward computing group. In a practical implementation,



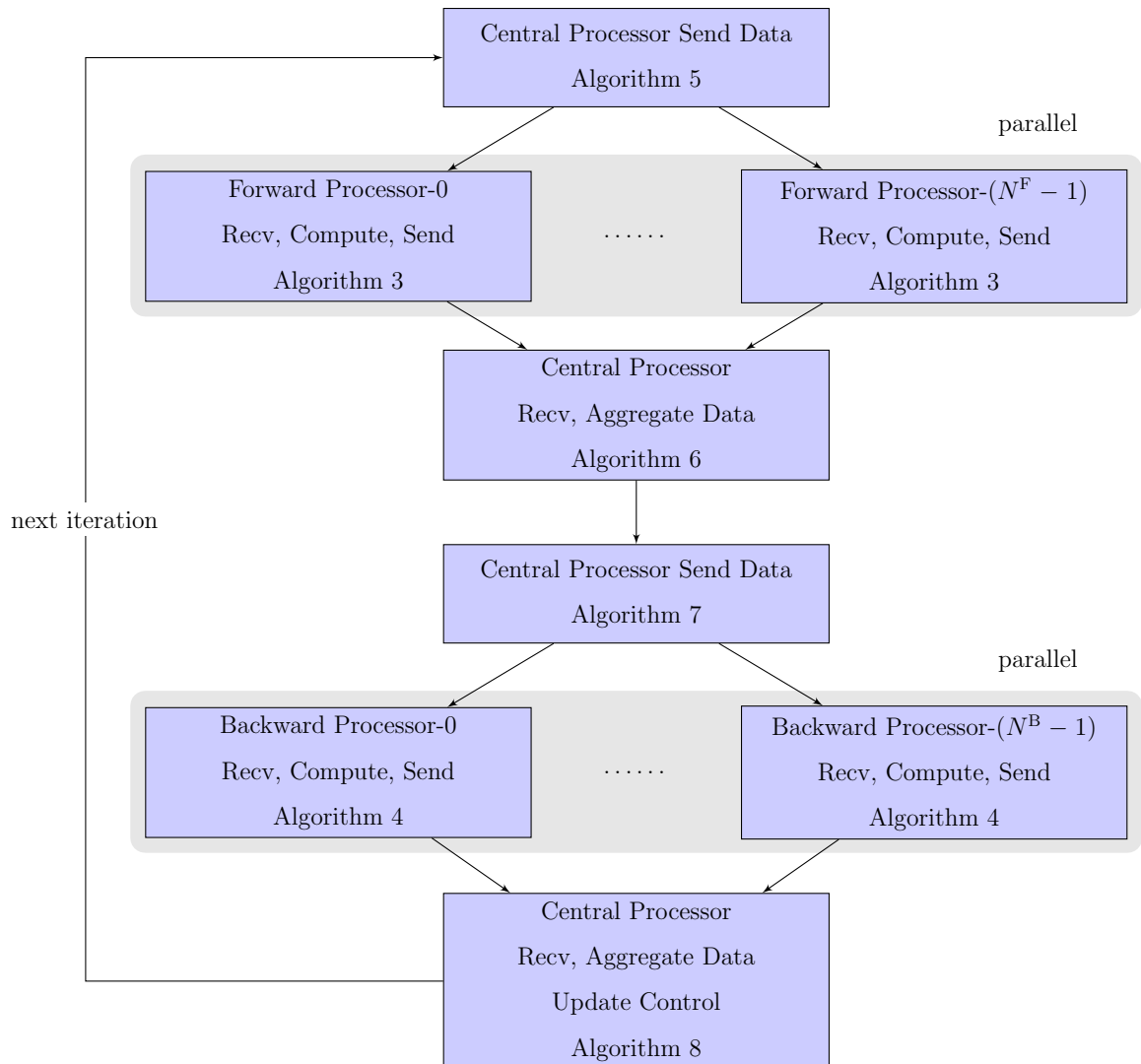


Figure 3.3: Workflow of One Iteration of the Generalized Parallel-in-Time Gradient-Type Method Algorithm. Serial computation in central processor is negligible compared to the forward/backward computation happening in the parallel.

of course, forward/backward computing group and the central processor can use some common processors., as in the original parallel-in-time gradient-type method Algorithm 2, forward/backward computation is using the same processors and the tasks of the central processor is distributed to related computation processors. However, in this section, to keep a low write-up complexity level, I write the algorithm as if forward/backward groups do not share any common processors and there is a dedicated central processor for data aggregation and control update.

- The algorithms are not optimized in terms of data transmission. It is not necessary that, as in Time B of Algorithm 3 and Time D of Algorithm 4, after each step of computation, all results are gathered in the central processor. Actually, there does not need to be a central processor. Each processor can collect necessary information from the related processor(s) as is done in the original parallel-in-time gradient-type method where there is no central processor and only adjacent processors exchange information.

The following Algorithm 3, Algorithm 4, Algorithm 5, Algorithm 6, Algorithm 7, and Algorithm 8 are to be read with the illustration in Figure 3.3.

---

**Algorithm 3**  $j$ th iteration of the generalized parallel-in-time gradient-type method.

Executed by **FORWARD** processor of rank  $i \in \{0, \dots, N^F - 1\}$ .

---

- 1: receive  $u_{s_i^F}^{(j)}, \dots, u_{e_i^F - 1}^{(j)}$ , and  $y_{s_i^F}^{(j-1)}$  from the central processor ▷ Time A
  - 2:  $y_{i, s_i^F}^{(j)} = y_{s_i^F}^{(j-1)}$
  - 3: **for**  $k = s_i^F, \dots, e_i^F - 1$  **do**
  - 4:  $y_{i, k+1}^{(j)} = A_k y_{i, k}^{(j)} + B_k u_k^{(j)} + c_k$
  - 5: **end for**
  - 6: send  $y_{i, e_i^F - 1 + 1}^{(j)}, \dots, y_{i, e_i^F}^{(j)}$  to the central processor. ▷ Time B
-

---

**Algorithm 4**  $j$ th iteration of the generalized parallel-in-time gradient-type method.

Executed by **BACKWARD** processor of rank  $i \in \{0, \dots, N^B - 1\}$

Dummy matrix  $A_K$  of proper dimension is defined to be arbitrary value.

---

- 1: receive  $y_{s_i^B+1}^{(j)}, \dots, y_{e_i^B}^{(j)}$ , and  $p_{e_i^B}^{(j-1)}$  from the central processor ▷ Time C
  - 2:  $p_{i, e_i^B}^{(j)} = p_{e_i^B}^{(j-1)}$
  - 3: **for**  $k = e_i^B, \dots, s_i^B + 1$  **do**
  - 4:      $p_{i, k-1}^{(j)} = Q_k y_k^{(j)} + d_k + A_k^T p_{i, k}^{(j)}$
  - 5: **end for**
  - 6: send  $p_{i, s_i^B}^{(j)}, \dots, p_{i, s_{i+1}^B-1}^{(j)}$  to the central processor. ▷ Time D
- 

---

**Algorithm 5**  $j$ th iteration of the generalized parallel-in-time gradient-type method.

Describes the tasks executed by central processor at **Time A** in Algorithm 3

---

- 1: **for**  $i = 1, \dots, N^F - 1$  **do** ▷ Communication
  - 2:     send to **FORWARD** processor  $i$ ,  $u_{s_i^F}^{(j)}, \dots, u_{e_i^F-1}^{(j)}$ , and  $y_{s_i^F}^{(j-1)}$
  - 3: **end for**
- 

---

**Algorithm 6**  $j$ th iteration of the generalized parallel-in-time gradient-type method.

Describes the tasks executed by central processor at **Time B** in Algorithm 3

---

- 1: **for**  $i = 1, \dots, N^F - 1$  **do** ▷ Communication
  - 2:     receive from **FORWARD** processor  $i$ ,  $y_{i, e_{i-1}^F+1}^{(j)}, \dots, y_{i, e_i^F}^{(j)}$
  - 3: **end for**
  - 4:  $y_0^{(j)} = y_{\text{given}}$  ▷ Aggregate State
  - 5: **for**  $i = 1, \dots, N^F - 1$  **do**
  - 6:     **for**  $k = e_{i-1}^F + 1, \dots, e_i^F$  **do**
  - 7:          $y_k^{(j)} = y_{i, k}^{(j)}$
  - 8:     **end for**
  - 9: **end for**
-

---

**Algorithm 7**  $j$ th iteration of the generalized parallel-in-time gradient-type method.

Describes the tasks executed by central processor at **Time C** in Algorithm 4

---

- 1: **for**  $i = 1, \dots, N^B - 1$  **do** ▷ Communication
  - 2:     send to **BACKWARD** processor  $i$ ,  $y_{s_i^B+1}^{(j)}, \dots, y_{e_i^B}^{(j)}$ , and  $p_{e_i^B}^{(j-1)}$
  - 3: **end for**
- 

---

**Algorithm 8**  $j$ th iteration of the generalized parallel-in-time gradient-type method.

Describes the tasks executed by central processor at **Time D** in Algorithm 4

---

- 1: **for**  $i = 1, \dots, N^B - 1$  **do** ▷ Communication
  - 2:     receive from **BACKWARD** processor  $i$ ,  $p_{i,s_i^B}^{(j)}, \dots, p_{i,s_{i+1}^B-1}^{(j)}$
  - 3: **end for**
  
  - 4: **for**  $i = 1, \dots, N^B - 1$  **do** ▷ Aggregate Adjoint
  - 5:     **for**  $k = s_i^B, \dots, s_{i+1}^B - 1$  **do**
  - 6:          $p_k^{(j)} = p_{i,k}^{(j)}$
  - 7:     **end for**
  - 8: **end for**
  
  - 9: **for**  $k = 0, \dots, K - 1$  **do** ▷ Update Control
  - 10:      $u_k^{(j+1)} = u_k^{(j)} - \alpha(R_k u_k^{(j)} + e_k + B_k^T p_k^{(j)})$
  - 11: **end for**
-

### 3.4.4 Subdomain Initial/Terminal Processor Rank Function

In this section, I introduce an auxiliary processor rank index valued function  $I^F$  and  $I^B$  in (3.4.13) for analysis of algorithm later.

In the original parallel-in-time gradient-type method introduced in Section 3.2, the forward/backward computation in a subdomain always starts by the previous iteration computation results from an adjacent subdomain with index different by 1. Getting information from the adjacent subdomain is not necessarily the most reasonable way to carry out computation and propagate information in this generalized framework, for which reason I designed the data aggregation rule in Section 3.4.2.

Consider the example in Figure 3.2, for the forward computation, processor 1 uses the previous iteration terminal computation result, at  $T_2$ , of processor 2 as initial state, which is the exact pattern introduced in the original parallel-in-time algorithm; processor 2 uses a previous iteration intermediate, at  $T_3$  (rather than terminal, at  $T_5$ ), computation result of processor 1 as initial state, which is slightly different from the rule of the original parallel-in-time algorithm. However, in these two cases above, information is still only transmitted between subdomains with adjacent indices, similar to the original algorithm.

The situation is different for processor 4. The computation domains of both processor 2 and processor 3 includes  $T_8$  which is the initial time step for processor 4. There is a choice to make on which state, from processor 2 or processor 3, to use in processor 4 as initial state. Heuristically, the state at  $T_8$  in processor 2 encodes the control of the previous iteration from  $T_3$  to  $T_7$ . However, The state at  $T_8$  in processor 3 only encodes the control of the previous iteration at  $T_6$  and  $T_7$ . Since the state at  $T_8$  in processor 2 incorporates more time steps of recent controls, I restrict the generalized algorithm to use the state at  $T_8$  in processor 2 as the initial state for processor 4. In this case information is transmitted from subdomain 2 in processor 2 to subdomain 4 in processor 4, the indices of which are not adjacent.

Define the index valued functions encoding the forward/backward computation

rule as follows,

$$I^F(k) \stackrel{\text{def}}{=} \min\{j \in \{0, \dots, N-1\} | s_j^F \leq k \leq e_j^F\} \text{ for } k = 0, \dots, K \quad (3.4.13a)$$

$$I^B(k) \stackrel{\text{def}}{=} \max\{j \in \{0, \dots, M-1\} | s_j^B \leq k \leq e_j^B\} \text{ for } k = 0, \dots, K \quad (3.4.13b)$$

which are well defined since (3.4.1) and (3.4.5) holds. Then, for forward computation, at iteration  $j$ , for time subdomain  $i \geq 1$ , the initial state variable

$$y_{i,s_i^F}^{(j)} = y_{I^F(s_i^F),s_i^F}^{(j-1)}$$

and, for backward computation, at iteration  $j$ , for time subdomain  $i \leq N-2$ , the terminal adjoint variable

$$p_{i,e_i^B}^{(j)} = p_{I^B(e_i^B),e_i^B}^{(j-1)}$$

Note that in the original algorithm for  $i \geq 1$ ,

$$I^F(s_i^F) = i - 1$$

and for  $i \leq N-2$ ,

$$I^B(e_i^B) = i + 1$$

Note that the algorithms given in Section 3.4.3 avoided these functions by using aggregated variables. However, for theoretical purpose, definition of  $I^F$  and  $I^B$  is necessary.

### 3.4.5 $D^F$ and $D^B$

Similar to the parallel-in-time gradient-type algorithm analyzed before, in the analysis, I reduce the iteration on state/adjoint/control to the iteration on control only. As mentioned before, state variable in the parallel-in-time gradient-type method is a joint result of control variable from several previous iterations. From another perspective, control information at the beginning of the whole time domain can take several iterations to reach the end of the whole time domain. For adjoint variable it

is also similar. It takes several iterations for a perturbation at the end of the whole time domain to propagate to the beginning. In this section, I count exactly how many previous iterations are needed by  $D^F$  and  $D^B$ . Note that, the current iteration is not counted. For example, in the case with only one time subdomain which is the whole time domain,  $D^F = D^B = 0$ .

In the example demonstrated by Figure 3.2, for  $j \geq 3$ , I trace back the computation of  $y_{4,12}^{(j)}$ , state-type vector at overall time step 12 in subdomain-4, in the generalized parallel-in-time gradient-type method as introduced in Section 3.4.3. The ordered product of matrices  $\prod_{h=i}^j A_h$  is defined in (3.3.1).

$$\begin{aligned}
y_{4,12}^{(j)} &= A_{11}y_{4,11}^{(j)} + B_{11}u_{11}^{(j)} + c_{11} \\
&= A_{11}[A_{10}y_{4,10}^{(j)} + B_{10}u_{10}^{(j)} + c_{10}] + B_{11}u_{11}^{(j)} + c_{11} \\
&= \left[\prod_{h=8}^{11} A_h\right]y_{4,8}^{(j)} + \sum_{k=8}^{11} \left[\prod_{h=k+1}^{11} A_h\right][B_k u_k^{(j)} + c_k]
\end{aligned} \tag{3.4.14}$$

Corresponding to the state variables aggregation in Algorithm 6, data communication in Algorithm 5 and Algorithm 3,

$$y_{4,8}^{(j)} = y_8^{(j-1)} = y_{2,8}^{(j-1)} \tag{3.4.15}$$

and similarly,

$$y_{2,3}^{(j-1)} = y_3^{(j-2)} = y_{1,3}^{(j-2)} \tag{3.4.16}$$

$$y_{1,2}^{(j-2)} = y_2^{(j-3)} = y_{0,2}^{(j-3)} \tag{3.4.17}$$

Then,

$$\begin{aligned}
y_{4,12}^{(j)} &= \left[ \prod_{h=8}^{11} A_h \right] y_{2,8}^{(j-1)} + \sum_{k=8}^{11} \left[ \prod_{h=k+1}^{11} A_h \right] [B_k u_k^{(j)} + c_k] \\
&= \left[ \prod_{h=3}^{11} A_h \right] y_{2,3}^{(j-1)} + \sum_{k=3}^7 \left[ \prod_{h=k+1}^{11} A_h \right] [B_k u_k^{(j-1)} + c_k] + \sum_{k=8}^{11} \left[ \prod_{h=k+1}^{11} A_h \right] [B_k u_k^{(j)} + c_k] \\
&= \left[ \prod_{h=0}^{11} A_h \right] y_0 + \sum_{k=0}^{11} \left[ \prod_{h=k+1}^{11} A_h \right] c_k + \sum_{k=0}^1 \left[ \prod_{h=k+1}^{11} A_h \right] B_k u_k^{(j-3)} + \sum_{k=2}^2 \left[ \prod_{h=k+1}^{11} A_h \right] B_k u_k^{(j-2)} \\
&\quad + \sum_{k=3}^7 \left[ \prod_{h=k+1}^{11} A_h \right] B_k u_k^{(j-1)} + \sum_{k=8}^{11} \left[ \prod_{h=k+1}^{11} A_h \right] B_k u_k^{(j)}
\end{aligned} \tag{3.4.18}$$

It can be seen that  $y_{4,12}^{(j)}$  is determined by part of controls from current iteration  $j$  and 3 previous iterations  $j-1, j-2, j-3$ . Also notice that in the computation of  $y_{4,12}^{(j)}$ , forward computation results from subdomain-3,  $T_6$  to  $T_1$ , are not used, see (3.4.15). Although, there are processor-0 to processor-4, 5 forward computation processors, only controls from 3 previous iterations are involved in the computation of the terminal state of whole time domain  $y_{4,12}^{(j)}$ , in contrast to the case of the original parallel-in-time gradient-type method where in a  $N$  processor algorithm, there are always control variables from  $N-1$  previous iteration collectively determining the terminal state, see (3.3.7).

Given a subdomain setup, I use the notation  $D^F$  to denote the number of previous control iterates that are needed to determine the current terminal state.

The generalized parallel-in-time gradient-type algorithm in Section 3.4.3 does not need to be given  $D^F$  explicitly. However,  $D^F$  is useful in analysis of the algorithm convergence, which can be counted by Algorithm 9.

Similar to Algorithm 9, the following Algorithm 10 does the counterpart for the backward computation.



---

**Algorithm 9** Count  $D^F$  (for forward computation)

---

- 1:  $D^F = -1, s = e_{N^F-1}^F$
  - 2: **repeat**
  - 3:      $s = s_{I^F(s)}^F$
  - 4:      $D^F = D^F + 1$
  - 5: **until**  $s = 0$
  - 6: **return**  $D^F$
- 

**Algorithm 10** Count  $D^B$  (for backward computation)

---

- 1:  $D^B = -1, e = s_0^B$
  - 2: **repeat**
  - 3:      $e = e_{I^B(e)}^B$
  - 4:      $D^B = D^B + 1$
  - 5: **until**  $e = K$
  - 6: **return**  $D^B$
- 

### 3.4.6 Interpretation as a $(D^F + D^B + 1)$ -Part Iteration Scheme

In Section 3.3, I developed compact formula (3.3.7) and (3.3.8) that summarizes the original parallel-in-time gradient-type method iteration, where the most important ingredients are the “mask” matrices  $\mathbf{I}_{-d}, d = 0, 1, \dots, N - 1$ .

Now, for the generalized version of algorithm, by very similar analysis along the line of (3.4.18), I can construct “mask” matrices for both forward,  $\mathbf{I}_{-d}^F, d = 0, 1, \dots, D^F$ , and backward,  $\mathbf{I}_{-d}^B, d = 0, 1, \dots, D^B$ , computation. Note that in the original algorithm with  $N$  subdomains in both forward and backward computation,  $D^F = D^B = N - 1$ .

The derivation of  $\mathbf{I}_{-d}^F$  and  $\mathbf{I}_{-d}^B$  is the similar to that of  $\mathbf{I}_{-d}$  in Section 3.3. I omit that and give directly the Algorithm 11 and Algorithm 12 for constructing them.

The explicitly constructed  $\mathbf{I}_{-d}^F, d = 0, 1, \dots, D^F$  and  $\mathbf{I}_{-d}^B, d = 0, 1, \dots, D^B$  are illustrated in Figure 3.4 and Figure 3.5 for the time domain splitting in Figure 3.2.

Now that all tools are developed. I state the compact notation for the generalized

---

**Algorithm 11** Construction of  $\mathbf{I}_{-d}^F$ 

Matrix index is zero based using NUMPY style notation where  $\mathbf{M}[1 : 3, :]$  represents the submatrix consisting of the whole 2nd and 3rd row of matrix  $\mathbf{M}$ ;

$e_{-1}^F \stackrel{\text{def}}{=} 0$  for notational simplicity

---

- 1: Initialize  $\mathbf{I}_{-d}^F, d = 0, 1, \dots, D^F$  to be  $Kn_y \times Kn_y$  zero matrices.
  - 2: **for**  $i = 0, \dots, N^F - 1$  **do** ▷ Loop over matrix block rows
  - 3:      $row_{\text{start}} = e_{i-1}^F, row_{\text{end}} = e_i^F, col_{\text{end}} = e_i^F$
  - 4:      $d = 0$
  - 5:     **repeat** ▷ Loop over blocks in one block row
  - 6:          $col_{\text{start}} = s_{I^F}^F(col_{\text{end}})$
  - 7:          $\mathbf{I}_{-d}^F[row_{\text{start}} : row_{\text{end}}, col_{\text{start}} : col_{\text{end}}] = 1$  ▷ Mark block of all “1”s
  - 8:          $col_{\text{end}} = col_{\text{start}}$
  - 9:          $d = d + 1$
  - 10:     **until**  $col_{\text{end}} = 0$
  - 11: **end for**
-

---

**Algorithm 12** Construction of  $\mathbf{I}_{-d}^B$ 

Matrix index is zero based using NUMPY style notation where  $\mathbf{M}[1 : 3, :]$  represents the submatrix consisting of the whole 2nd and 3rd row of matrix  $\mathbf{M}$ ;

$s_{NB}^B \stackrel{\text{def}}{=} K$  for notational simplicity

---

- 1: Initialize  $\mathbf{I}_{-d}^B, d = 0, 1, \dots, D^B$  to be  $Kn_y \times Kn_y$  zero matrices.
  - 2: **for**  $i = 0, \dots, N^B - 1$  **do** ▷ Loop over matrix block rows
  - 3:      $row_{start} = s_i^B, row_{end} = s_{i+1}^B, col_{start} = s_i^B$
  - 4:      $d = 0$
  - 5:     **repeat** ▷ Loop over blocks in one block row
  - 6:          $col_{end} = e_{I^B}^B(col_{start})$
  - 7:          $\mathbf{I}_{-d}^B[row_{start} : row_{end}, col_{start} : col_{end}] = 1$  ▷ Mark block of all “1”s
  - 8:          $col_{start} = col_{end}$
  - 9:          $d = d + 1$
  - 10:     **until**  $col_{start} = K$
  - 11: **end for**
  - 12: Transpose  $\mathbf{I}_{-d}^B, d = 0, 1, \dots, D^B$
-

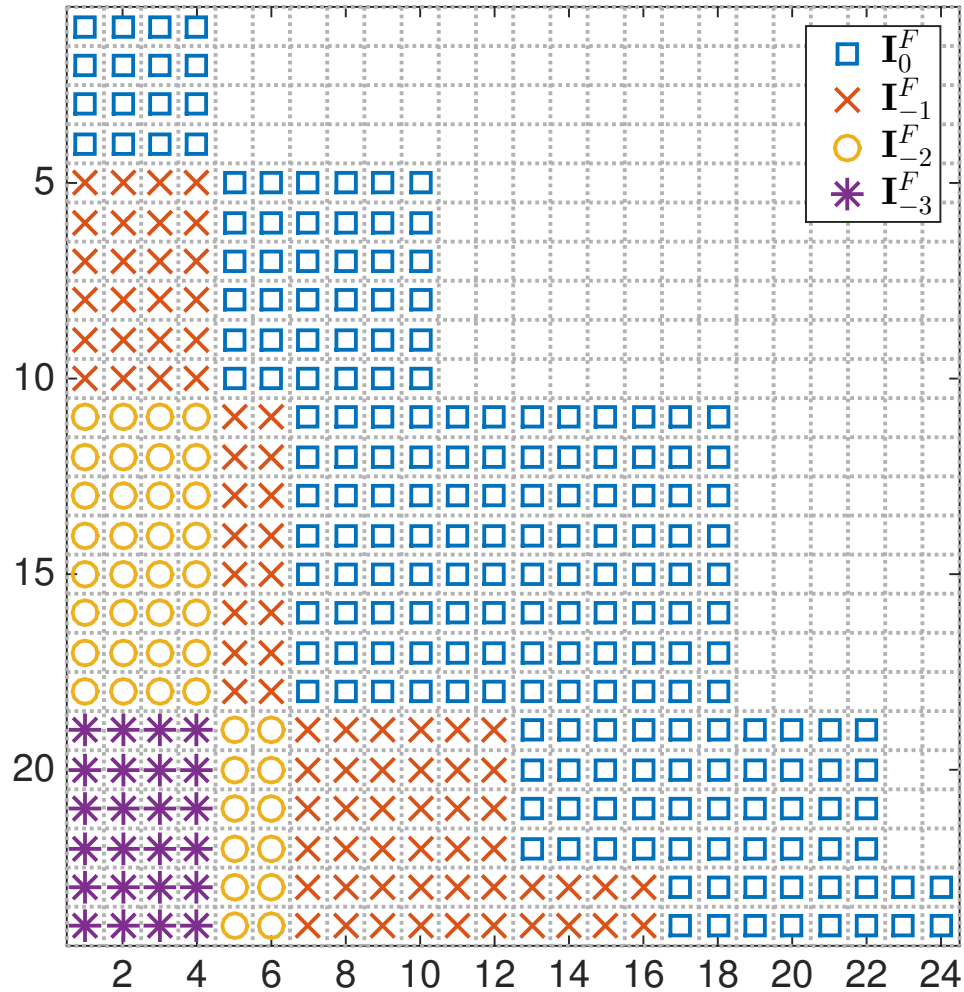


Figure 3.4: Illustration of the positions of ‘1’s in  $\mathbf{I}_0^F$ ,  $\mathbf{I}_{-1}^F$ ,  $\mathbf{I}_{-2}^F$ , and  $\mathbf{I}_{-3}^F$  in an example where the state dimension is  $n_y = 2$  and  $K = 12$ . The time domain splitting pattern is illustrated in Figure 3.2 and represented in Table 3.1. Compare with Figure 3.1 for the original parallel-in-time gradient-type method.

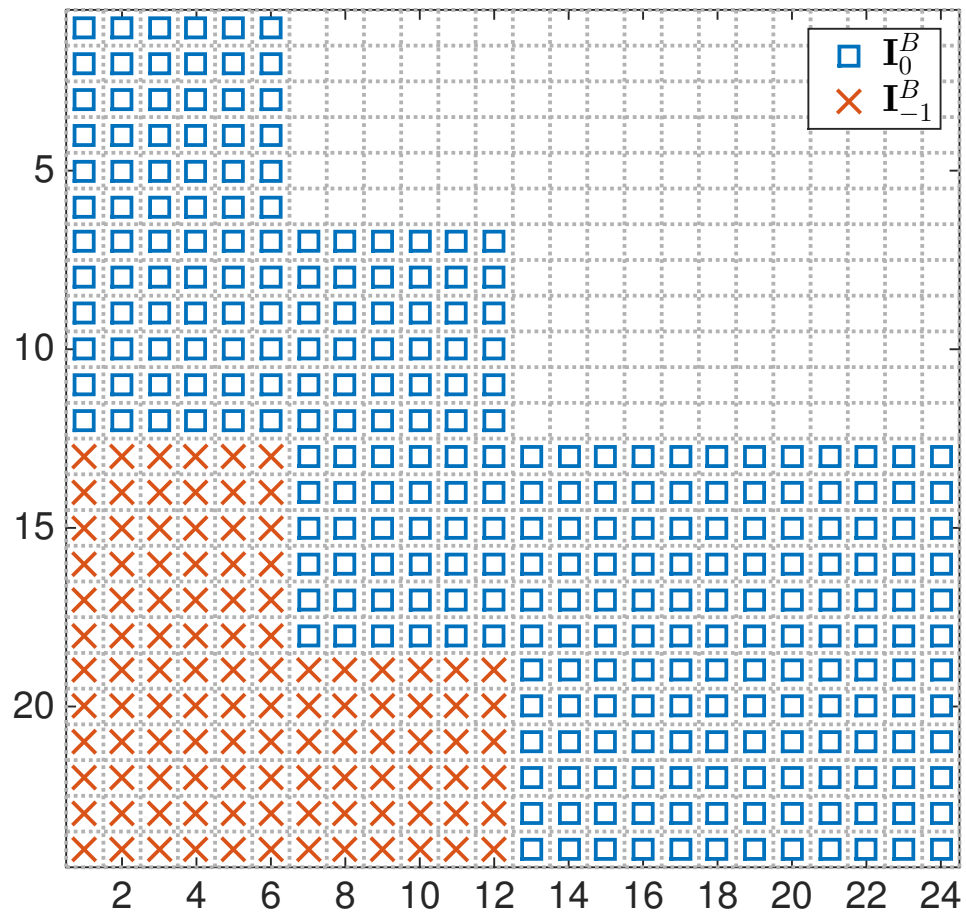


Figure 3.5: Illustration of the positions of ‘1’s in  $\mathbf{I}_0^B$  and  $\mathbf{I}_{-1}^B$  in an example where the state dimension is  $n_y = 2$  and  $K = 12$ . The time domain splitting pattern is illustrated in Figure 3.2 and represented in Table 3.1. Compare with Figure 3.1 for the original parallel-in-time gradient-type method.

parallel-in-time gradient-type method analogous to that of the original method in Section 3.3. Similar to (3.3.7),

$$\mathbf{y}^{(j)} = \sum_{d=0}^{D^F} (\mathbf{I}_{-d}^F \circ \mathbf{L})(\mathbf{B}\mathbf{u}^{(j-d)} + \mathbf{c}) + \mathbf{y}_0, \quad \text{for } j \geq D^F. \quad (3.4.19)$$

Analogous to (3.3.8),

$$\mathbf{p}^{(j)} = \sum_{d=0}^{D^B} (\mathbf{I}_{-d}^B \circ \mathbf{L})^T (\mathbf{Q}\mathbf{y}^{(j-d)} + \mathbf{d}), \quad \text{for } j \geq D^B. \quad (3.4.20)$$

Using (3.2.1e), (3.4.19) and (3.4.20) gives the following representation of our parallel-in-time gradient type iteration  $j \geq 2N - 2$ ,

$$\begin{aligned} \mathbf{u}^{(j+1)} &= \mathbf{u}^{(j)} - \alpha(\mathbf{R}\mathbf{u}^{(j)} + \mathbf{e} + \mathbf{B}^T \mathbf{p}^{(j)}) \\ &= \mathbf{u}^{(j)} - \alpha \left[ \mathbf{R}\mathbf{u}^{(j)} + \sum_{r=0}^{D^B} \sum_{l=0}^{D^F} (\mathbf{I}_{-r}^B \circ \mathbf{L}\mathbf{B})^T \mathbf{Q}(\mathbf{I}_{-l}^F \circ \mathbf{L}\mathbf{B})\mathbf{u}^{(j-r-l)} + \mathbf{g} \right]. \end{aligned} \quad (3.4.21)$$

I define

$$\mathbf{H}_0^G \stackrel{\text{def}}{=} \mathbf{R} + (\mathbf{I}_0^B \circ \mathbf{L}\mathbf{B})^T \mathbf{Q}(\mathbf{I}_0^F \circ \mathbf{L}\mathbf{B}), \quad (3.4.22a)$$

$$\mathbf{H}_d^G \stackrel{\text{def}}{=} \sum_{\substack{l \in \{0, \dots, D^F\} \\ r \in \{0, \dots, D^B\} \\ l+r=d}} (\mathbf{I}_{-r}^B \circ \mathbf{L}\mathbf{B})^T \mathbf{Q}(\mathbf{I}_{-l}^F \circ \mathbf{L}\mathbf{B}), \quad d = 1, \dots, D^F + D^B. \quad (3.4.22b)$$

Where I use super script  $G$  to emphasize it is related to the generalized algorithm.

Note that the Hessian (3.1.8) can be split as

$$\mathbf{H} = \sum_{d=0}^{D^F+D^B} \mathbf{H}_d^G. \quad (3.4.23)$$

Inserting (3.4.22) into (3.4.21) gives

$$\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} - \alpha \left( \sum_{d=0}^{D^F+D^B} \mathbf{H}_d^G \mathbf{u}^{(j-d)} + \mathbf{g} \right). \quad (3.4.24)$$

which is a generalized version of (3.3.12).

Convergence of the generalized parallel-in-time gradient-type method is given use the developed compact notation above in Section 3.5.

### 3.5 Convergence Proof for the Linear-Quadratic Problems

Using the representation (3.1.9) it follows immediately that the optimal control  $\mathbf{u}^{(*)}$  satisfies  $\mathbf{H}\mathbf{u}^{(*)} + \mathbf{g} = \mathbf{0}$  and, using (3.4.23),

$$\mathbf{u}^{(*)} = \mathbf{u}^{(*)} - \alpha \left( \sum_{d=0}^{D^F+D^B} \mathbf{H}_d^G \mathbf{u}^{(*)} + \mathbf{g} \right). \quad (3.5.1)$$

Subtracting (3.4.24) from (3.5.1) shows that the errors

$$\boldsymbol{\varepsilon}^{(j)} = \mathbf{u}^{(*)} - \mathbf{u}^{(j)} \quad (3.5.2)$$

obey the recursion

$$\boldsymbol{\varepsilon}^{(j+1)} = \boldsymbol{\varepsilon}^{(j)} - \alpha \left( \sum_{d=0}^{D^F+D^B} \mathbf{H}_d^G \boldsymbol{\varepsilon}^{j-d} \right). \quad (3.5.3)$$

If I define the block companion matrix

$$\mathbf{C}(\alpha) = \begin{bmatrix} \mathbf{I} - \alpha \mathbf{H}_0^G & -\alpha \mathbf{H}_1^G & -\alpha \mathbf{H}_2^G & \dots & -\alpha \mathbf{H}_{D^F+D^B-1}^G & -\alpha \mathbf{H}_{D^F+D^B}^G \\ & \mathbf{I} & & & & \\ & & \mathbf{I} & & & \\ & & & \mathbf{I} & & \\ & & & & \ddots & \\ & & & & & \mathbf{I} \\ & & & & & & \mathbf{0} \end{bmatrix}, \quad (3.5.4)$$

then the recursion (3.5.3) for the errors is equivalent to

$$\begin{bmatrix} \boldsymbol{\varepsilon}^{(j+1)} \\ \boldsymbol{\varepsilon}^{(j)} \\ \vdots \\ \boldsymbol{\varepsilon}^{(j-[D^F+D^B]+1)} \end{bmatrix} = \mathbf{C}(\alpha) \begin{bmatrix} \boldsymbol{\varepsilon}^{(j)} \\ \boldsymbol{\varepsilon}^{(j-1)} \\ \vdots \\ \boldsymbol{\varepsilon}^{(j-[D^F+D^B])} \end{bmatrix}. \quad (3.5.5)$$

Thus, convergence of the parallel-in-time gradient method is guaranteed if the spectral radius of the block companion matrix (3.5.4) is less than one. In the remainder of

this section I will show that this is true for sufficiently small fixed step size, i.e., Theorem 3.5.1.

**Theorem 3.5.1** *For sufficiently small step size  $\alpha > 0$ , the matrix  $\mathbf{C}(\alpha)$  defined in (3.5.4) has spectral radius less than one.*

The rest of this section is devoted to the proof of Theorem 3.5.1. Specifically, Theorem 3.5.1 is a special case of Theorem 3.5.3 below. I will aim to prove Theorem 3.5.3.

In the convergence proof the positive definiteness of the Hessian is important, but not the particular structure of the matrices  $\mathbf{H}_d$  in the splitting (3.3.11). Therefore, given complex  $m \times m$  matrices  $\mathbf{M}_0, \mathbf{M}_1, \dots, \mathbf{M}_n$  with  $\sum_{i=0}^n \mathbf{M}_i$  Hermitian and positive definite, I consider the block companion matrix

$$\tilde{\mathbf{C}}(\alpha) \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{I} - \alpha\mathbf{M}_0 & -\alpha\mathbf{M}_1 & -\alpha\mathbf{M}_2 & \dots & -\alpha\mathbf{M}_{n-1} & -\alpha\mathbf{M}_n \\ \mathbf{I} & & & & & \\ & \mathbf{I} & & & & \\ & & \mathbf{I} & & & \\ & & & \ddots & & \\ & & & & \mathbf{I} & \mathbf{0} \end{bmatrix}. \quad (3.5.6)$$

I will prove that the spectral radius of  $\tilde{\mathbf{C}}(\alpha)$  is strictly less than one for sufficient small step sizes  $\alpha > 0$ .

The proof is based on an analysis of the location of the roots of the characteristic polynomial of  $\tilde{\mathbf{C}}(\alpha)$ ,

$$Q(\alpha, \lambda) \stackrel{\text{def}}{=} \det(\tilde{\mathbf{C}}(\alpha) - \lambda\mathbf{I}), \quad (3.5.7)$$

for small  $\alpha > 0$ . I also define

$$P(\alpha, \lambda) \stackrel{\text{def}}{=} \lambda^{n+1}\mathbf{I} + \lambda^n(\alpha\mathbf{M}_0 - \mathbf{I}) + \alpha \sum_{i=1}^n \lambda^{n-i}\mathbf{M}_i. \quad (3.5.8)$$

Note that  $Q(\alpha, \lambda) = (-1)^{mn} \det(P(\alpha, \lambda))$  (see, e.g., [DTW71, p.17]). The following three statements are equivalent



- i.  $\lambda_\alpha$  is an eigenvalue of  $\tilde{\mathbf{C}}(\alpha)$ .
- ii.  $\lambda_\alpha$  is an latent root of  $P(\alpha, \cdot)$ , i.e.  $P(\alpha, \lambda_\alpha)$  is singular.
- iii.  $\lambda_\alpha$  is a root of  $Q(\alpha, \cdot)$ .

Since  $P(0, \lambda) = \lambda^n(\lambda - 1)\mathbf{I}$  and  $\det(P(0, \lambda)) = \lambda^{mn}(\lambda - 1)^m$ , the companion matrix  $\tilde{\mathbf{C}}(0)$  only has eigenvalues 0 and 1. By continuity of polynomial roots with respect to polynomial coefficients [US77], the roots  $\lambda_\alpha$  of  $Q(\alpha, \cdot)$  are contained in small balls around 0 and around 1 for sufficiently small  $\alpha > 0$ . See Figure 3.6. For the roots  $\lambda_\alpha$  in the small ball around 1, I can actually show that they must be contained in the open cone  $C_k$  defined in (3.5.9) below, and that they have magnitude less than one. See Figure 3.6 and Lemma 3.5.2 below. This implies that the spectral radius of  $\tilde{\mathbf{C}}(\alpha)$  is less than one for sufficiently small  $\alpha > 0$  (see Theorem 3.5.3 below).

In the following  $B(c, r)$  denotes the open ball in the complex plane of radius  $r$  centered at  $c$ . The real and imaginary parts of a complex number  $z$  are denoted by  $\operatorname{Re}(z)$  and  $\operatorname{Im}(z)$ , respectively.

**Lemma 3.5.2** *If  $\mathbf{M}_0, \mathbf{M}_1, \dots, \mathbf{M}_n$  are  $m \times m$  complex matrices with  $\sum_{i=0}^n \mathbf{M}_i$  Hermitian and positive definite, then the following two statements are valid.*

- i. *For any  $\delta_2$  there exists a  $\delta_1 > 0$  such that for all  $\alpha \in (0, \delta_1) \subset \mathbb{R}$  all latent roots of  $P(\alpha, \lambda)$  are contained in  $B(0, \delta_2) \cup B(1, \delta_2) \subset \mathbb{C}$ .*
- ii. *Let  $\delta_2 \in (0, 1/2)$  and  $\delta_1$  be given as in part i. For all  $k > 0$  there exists  $\delta_3 \in (0, \delta_1)$  such that for all  $\alpha \in (0, \delta_3) \subset \mathbb{R}$  the latent roots  $\lambda \in B(1, \delta_2)$  of  $P(\alpha, \lambda)$  satisfy*

$$\lambda \in C_k \stackrel{\text{def}}{=} \{z \in \mathbb{C} : \operatorname{Re}(z) < 1 \text{ and } |\operatorname{Im}(z)|/(1 - \operatorname{Re}(z)) < k\}. \quad (3.5.9)$$

**Proof:** i. The first statement is a direct consequence of the fact that the roots of  $\det(P(0, \lambda)) = \det(\lambda^n(\lambda - 1)\mathbf{I}) = \lambda^{mn}(\lambda - 1)^m$  are  $\lambda = 0$  and  $\lambda = 1$  and of the

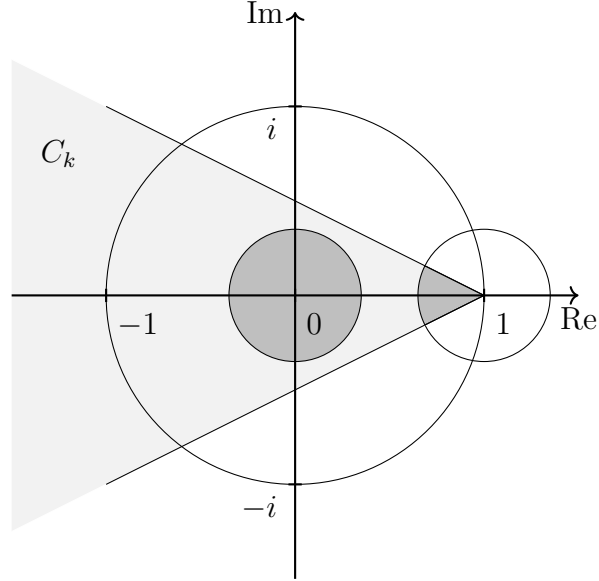


Figure 3.6: Eigenvalues of  $\tilde{\mathbf{C}}(\alpha)$ . For sufficiently small  $\alpha > 0$  the eigenvalues  $\lambda_\alpha$  of the companion matrix  $\tilde{\mathbf{C}}(\alpha)$ , defined in (3.5.6) with  $\sum_{i=0}^n \mathbf{M}_i$  Hermitian and positive definite, lie in the union of a small open ball around 0 and of the intersection of a small open ball around 1 and the open cone  $C_k$ , indicated by the dark shaded regions. In particular, all eigenvalues are inside the unit disk.

continuity of polynomial roots with respect to polynomial coefficients. (See, e.g., [US77].)

ii. I first prove the second part of the lemma for the case of  $n = 0$ . In this case  $P(\alpha, \lambda) = \lambda \mathbf{I} - (\mathbf{I} - \alpha \mathbf{M}_0)$  and  $\mathbf{M}_0$  is Hermitian and positive definite. All latent roots of  $P(\alpha, \lambda) = \lambda \mathbf{I} - (\mathbf{I} - \alpha \mathbf{M}_0)$  are given by  $\lambda_\alpha = 1 - \alpha \sigma$ , where  $\sigma > 0$  is an eigenvalue of  $\mathbf{M}_0$ . Let  $\sigma_{\max}$  denote the largest eigenvalue of  $\mathbf{M}_0$ . For  $\alpha \in (0, \delta_3)$ ,  $\delta_3 < \min\{2/\sigma_{\max}(\mathbf{M}_0), \delta_1\}$ , the latent roots of  $P(\alpha, \lambda)$  are contained in  $(-1, 1) \subset C_k$  for any  $k > 0$ .

Now let  $n \geq 1$ . I can write

$$\begin{aligned} P(\alpha, \lambda) &= \lambda^{n+1}\mathbf{I} + \lambda^n(\alpha\mathbf{M}_0 - \mathbf{I}) + \alpha \sum_{i=1}^n \lambda^{n-i}\mathbf{M}_i \\ &= \alpha \sum_{i=0}^n \mathbf{M}_i + (\lambda - 1)\mathbf{I} + \left[ (\lambda^n - 1)(\lambda - 1)\mathbf{I} + \alpha \sum_{i=0}^{n-1} (\lambda^{n-i} - 1)\mathbf{M}_i \right]. \end{aligned} \quad (3.5.10)$$

The last term in (3.5.10) can be estimated using

$$\begin{aligned} & \left\| (\lambda^n - 1)(\lambda - 1)\mathbf{I} + \alpha \sum_{i=0}^{n-1} (\lambda^{n-i} - 1)\mathbf{M}_i \right\|_2 \\ &= \left\| (\lambda - 1)^2 \sum_{i=0}^{n-1} \lambda^i \mathbf{I} + \alpha(\lambda - 1) \sum_{i=0}^{n-1} \left( \sum_{j=0}^{n-i-1} \lambda^j \right) \mathbf{M}_i \right\|_2 \\ &\leq |\lambda - 1| \left[ |\lambda - 1| \sum_{i=0}^{n-1} |\lambda|^i + \alpha \sum_{i=0}^{n-1} \left( \sum_{j=0}^{n-i-1} |\lambda|^j \right) \|\mathbf{M}_i\|_2 \right]. \end{aligned} \quad (3.5.11)$$

There exist  $\epsilon_1, \epsilon_2 > 0$  such that for all  $\alpha \in (0, \epsilon_1)$ ,  $\lambda \in B(1, \epsilon_2)$ ,

$$|\lambda - 1| \sum_{i=0}^{n-1} |\lambda|^i + \alpha \sum_{i=0}^{n-1} \left( \sum_{j=0}^{n-i-1} |\lambda|^j \right) \|\mathbf{M}_i\|_2 < \frac{1}{\sqrt{2}}$$

and, hence

$$\left\| (\lambda^n - 1)(\lambda - 1)\mathbf{I} + \alpha \sum_{i=0}^n (\lambda^{n-i} - 1)\mathbf{M}_i \right\|_2 \leq \frac{1}{\sqrt{2}}|\lambda - 1|. \quad (3.5.12)$$

Let  $\sigma_{\min}(\cdot)$  denote the minimum singular value of a matrix and recall that  $\sum_{i=0}^n \mathbf{M}_i$  is Hermitian and positive definite. For  $\alpha > 0$  and  $\lambda \in \mathbb{C}$  the first two terms in (3.5.10) can be estimated using

$$\begin{aligned} \sigma_{\min} \left( \alpha \sum_{i=0}^n \mathbf{M}_i + (\lambda - 1)\mathbf{I} \right) &= \left| \alpha \sigma_{\min} \left( \sum_{i=0}^n \mathbf{M}_i \right) + (\lambda - 1) \right| \\ &= \left[ \left( \alpha \sigma_{\min} \left( \sum_{i=0}^n \mathbf{M}_i \right) + \operatorname{Re}(\lambda - 1) \right)^2 + \operatorname{Im}(\lambda - 1)^2 \right]^{1/2}. \end{aligned} \quad (3.5.13a)$$

Furthermore, if  $\operatorname{Re}(\lambda) \geq 1$ , then

$$\begin{aligned}
\sigma_{\min}\left(\alpha \sum_{i=0}^n \mathbf{M}_i + (\lambda - 1)\mathbf{I}\right) &= \left[\left(\alpha \sigma_{\min}\left(\sum_{i=0}^n \mathbf{M}_i\right) + \operatorname{Re}(\lambda - 1)\right)^2 + \operatorname{Im}(\lambda - 1)^2\right]^{1/2} \\
&\geq \frac{1}{\sqrt{2}} \left| \alpha \sigma_{\min}\left(\sum_{i=0}^n \mathbf{M}_i\right) + \operatorname{Re}(\lambda - 1) \right| + \frac{1}{\sqrt{2}} \left| \operatorname{Im}(\lambda - 1) \right| \\
&= \frac{1}{\sqrt{2}} \alpha \sigma_{\min}\left(\sum_{i=0}^n \mathbf{M}_i\right) + \frac{1}{\sqrt{2}} \operatorname{Re}(\lambda - 1) + \frac{1}{\sqrt{2}} \left| \operatorname{Im}(\lambda - 1) \right| \\
&\geq \frac{1}{\sqrt{2}} \alpha \sigma_{\min}\left(\sum_{i=0}^n \mathbf{M}_i\right) + \frac{1}{\sqrt{2}} |\lambda - 1|. \tag{3.5.13b}
\end{aligned}$$

Combining (3.5.12) and (3.5.13b) shows that for all  $\alpha \in (0, \epsilon_1)$  and all  $\lambda \in B(1, \epsilon_2)$  with  $\operatorname{Re}(\lambda) \geq 1$ ,

$$\sigma_{\min}\left(\alpha \sum_{i=0}^n \mathbf{M}_i + (\lambda - 1)\mathbf{I}\right) > \|(\lambda^n - 1)(\lambda - 1)\mathbf{I} + \alpha \sum_{i=0}^n (\lambda^{n-i} - 1)\mathbf{M}_i\|_2 \tag{3.5.14}$$

and, consequently, that  $P(\alpha, \lambda)$  is non-singular. Therefore, for all  $\alpha \in (0, \epsilon_1)$  all latent roots of  $P(\alpha, \lambda)$  that are contained in  $B(1, \epsilon_2)$  satisfy  $\operatorname{Re}(\lambda_\alpha) < 1$ .

Given an arbitrary  $k > 0$ . For  $\lambda$  with  $\operatorname{Re}(\lambda) < 1$  and  $\lambda \notin C_k$ ,  $|\operatorname{Im}(\lambda - 1)| = |\operatorname{Im}(\lambda)| \geq k(1 - \operatorname{Re}(\lambda)) = k \operatorname{Re}(1 - \lambda) = k|\operatorname{Re}(\lambda - 1)|$  and, thus,

$$\frac{k+1}{k} |\operatorname{Im}(\lambda - 1)| \geq |\operatorname{Im}(\lambda - 1)| + |\operatorname{Re}(\lambda - 1)| \geq |\lambda - 1|.$$

Combining this with (3.5.13a) I obtain the estimate

$$\sigma_{\min}\left(\alpha \sum_{i=0}^n \mathbf{M}_i + (\lambda - 1)\mathbf{I}\right) \geq |\operatorname{Im}(\lambda - 1)| \geq \frac{k}{k+1} |\lambda - 1|. \tag{3.5.15}$$

There exist  $\epsilon_{1,k}, \epsilon_{2,k} > 0$  such that for all  $\alpha \in (0, \epsilon_{1,k})$ ,  $\lambda \in B(1, \epsilon_{2,k})$ ,

$$|\lambda - 1| \sum_{i=0}^{n-1} |\lambda|^i + \alpha \sum_{i=0}^{n-1} \left( \sum_{j=0}^{n-i-1} |\lambda|^{(j)} \right) \|\mathbf{M}_i\|_2 < \frac{k}{k+1}.$$

Combining this inequality with (3.5.11) and (3.5.15) shows that for all  $\alpha \in (0, \epsilon_{1,k})$  and all  $\lambda \in B(1, \epsilon_{2,k})$  with  $\operatorname{Re}(\lambda) < 0$  and  $\lambda \notin C_k$  the inequality (3.5.14) holds and, thus, that  $P(\alpha, \lambda)$  is non-singular. Therefore, for all  $\alpha \in (0, \min\{\epsilon_1, \epsilon_{1,k}\})$  all

latent roots of  $P(\alpha, \lambda)$  that are contained in  $B(1, \min\{\epsilon_2, \epsilon_{2,k}\})$  satisfy  $\text{Re}(\lambda_\alpha) < 1$  and  $\lambda_\alpha \in C_k$ .

Given  $k > 0$ , choose  $\delta_3 \leq \min\{\epsilon_1, \epsilon_{1,k}\}$  such that for all  $\alpha \in (0, \delta_3)$  the latent roots of  $P(\alpha, \cdot)$  are in  $B(0, \delta_2) \cup B(1, \min\{\delta_2, \epsilon_2, \epsilon_{2,k}\}) \subset \mathbb{C}$ . This is possible by continuity of polynomial roots with respect to polynomial coefficients. From the previous steps it follows that all latent roots  $\lambda_\alpha \in B(1, \delta_2)$  of  $P(\alpha, \cdot)$  are in fact contained in  $B(1, \min\{\delta_2, \epsilon_2, \epsilon_{2,k}\})$  and satisfy  $\lambda_\alpha \in C_k$ .  $\square$

**Theorem 3.5.3** *If  $\mathbf{M}_0, \mathbf{M}_1, \dots, \mathbf{M}_n$  are  $m \times m$  complex matrices with  $\sum_{i=0}^n \mathbf{M}_i$  Hermitian and positive definite, then the block companion matrix (3.5.6) has spectral radius strictly less than 1 for sufficiently small real  $\alpha > 0$ .*

**Proof:** Let  $\delta_2 \in (0, 1/2)$  and  $k > 0$  arbitrary. Lemma 3.5.2 guarantees the existence of  $\delta_1 > 0$  and  $\delta_3 \in (0, \delta_1)$  such that for all  $\alpha \in (0, \delta_3)$  the latent roots of  $P(\alpha, \cdot)$  are contained in  $B(0, \delta_2) \cup (B(1, \delta_2) \cap C_k) \subset B(0, 1)$ . Therefore, for all  $\alpha \in (0, \delta_3)$ , the spectral radius of  $\tilde{\mathbf{C}}(\alpha)$  is strictly less than 1.  $\square$

Because  $\sum_{i=0}^n \mathbf{H}_i = \mathbf{H}$  by (3.3.11) and  $\mathbf{H}$  is symmetric positive definite by (3.1.8), Theorem 3.5.1 is a special case of Theorem 3.5.3.

## 3.6 Numerical Examples

In this Section, I present numerical examples of parallel-in-time gradient-type method applied to linear-quadratic discrete time optimal problems of type (3.1.1). Since the proposed parallel method is based on the classic serial gradient method, I mainly compare its performance against the fixed step size gradient method.

While positive results will be presented in this section that there can be great parallel speed-ups compared to the gradient method, one should bear in mind that,

to solve linear-quadratic problem whose optimality system is a linear system, the proposed method is still not competitive against some other methods essentially solving the linear system of the optimality system as Conjugate Gradient method (CG), since the classic gradient method as a starting point of the conception of the parallel-in-time gradient-method has no advantage over CG in terms of solving a linear system in most cases.

However, a parallel-in-time Krylov subspace method inspired by and closed related to the parallel-in-time gradient-type method is competitive in some linear-quadratic problems against CG. The related material is in Section A.3.

A common way in which I choose the step size in the following experiments is conducting a exhaustive grid search of the step size  $\alpha$  for the one that results in the best convergence speed. Currently, except for very small problems where the reduced control space Hessian is tractable to explicitly construct in which case for experiment purpose I can determine the optimal fixed step size by (3.1.11), I do not have other systematic approach to find a suitable step size. For fixed step size gradient-type method, the convergence speed is closed related to the spectral radius of a certain iteration matrix, i.e.  $\mathbf{C}(\alpha)$  in (3.5.5) or  $[I - \mathbf{W}(\alpha)]$  in (4.3.3). The spectral radius depends on the step size. Some numerics for investigating the dependence is in Section A.2.

It is worth mentioning that, for classical gradient method, there are some good methods to systematically choose step sizes, as line search methods and the simple and often very efficient Borzilai-Borwein method [BB88, Ray93, Ray97, Fle05]. However, they do not easily transition to the parallel-in-time gradient-type method because, after all, in the parallel gradient-type method does not compute exact gradient and the state/adjoint variables in memory is not feasible before convergence.

I present three set of examples in the following three sections.

- In Section 3.6.1, I introduce some common aspects of parallel gradient-type method implementation, demonstrate the dependence of convergence on step

size choices and non-monotonic convergence. Parallel efficiency is around 35% in this example.

- In Section 3.6.2, I present the numerical results from a parallel implementation indicating strong scaling with up to 50 cores. It is a great improvement from the classical gradient method. However, it still does not compete with Conjugate Gradient method. Trace plot produced by HPCTOOLKIT [ABF<sup>+</sup>10] is also included to illustrate forward/backward computation and data communication.
- In Section 3.6.3, I demonstrate the generalized parallel-in-time gradient-type method by a simple example. By adding overlapping computation subdomains to a non-overlapping one, I compare the iteration outputs.

### 3.6.1 1D Boundary Control: Step Size, Non-Monotonic Convergence

The first example is a Dirichlet boundary control problem governed by a linear advection-diffusion-reaction equation. Given  $T > 0$ ,  $\kappa > 0$ ,  $\beta > 0$ ,  $\gamma, v \geq 0$ , and functions  $\hat{y} \in L^2(0, T)$ ,  $f \in L^2((0, 1) \times (0, T))$ , the optimal control problem is

$$\text{minimize } \frac{1}{2} \int_0^T (y(1, t) - \hat{y}(t))^2 dt + \frac{\beta}{2} \int_0^T u^2(t) dt \quad (3.6.1a)$$

subject to

$$\begin{aligned} \frac{\partial y(x, t)}{\partial t} - \kappa \frac{\partial^2 y(x, t)}{\partial x^2} \\ + v \frac{\partial y(x, t)}{\partial x} + \gamma y(x, t) = f(x, t) \quad x \in (0, 1), t \in (0, T), \end{aligned} \quad (3.6.1b)$$

$$\frac{\partial y(1, t)}{\partial x} = 0 \quad t \in (0, T), \quad (3.6.1c)$$

$$y(0, t) = u(t) \quad t \in (0, T), \quad (3.6.1d)$$

$$y(x, 0) = y_0(x) \quad x \in (0, 1). \quad (3.6.1e)$$

This problem has a unique solution  $u \in L^2(0, T)$  and corresponding state  $y \in W(0, T)$ . See, e.g., [Trö10, Ch.3] or [Lio71] for details.

I will discretize the problem using piecewise linear finite elements in space and backward Euler in time. This leads to a fully discretized problem of the type (3.1.1). I will present the details below. Theorem 3.5.1 guarantees convergence of the parallel-in-time gradient-type method for sufficiently small step size  $\alpha$ , but unfortunately does not provide bounds on the feasible step sizes. I use this simple example to numerically explore the dependence of step sizes.

The optimal control problem is discretized in space using piecewise linear finite elements. This leads to

$$\text{minimize } \frac{1}{2} \int_0^T (y(t) - \hat{y}(t))^T Q (y(t) - \hat{y}(t)) dt + \frac{1}{2} \int_0^T u(t)^T R u(t) dt \quad (3.6.2a)$$

subject to

$$M \frac{d}{dt} y(t) + A y(t) = B u(t) + f(t), \quad t \in (0, T), \quad (3.6.2b)$$

$$y(0) = y_{\text{given}}. \quad (3.6.2c)$$

Here  $y(t) \in \mathbb{R}^{n_y}$ ,  $n_y$  is the number of spatial subdomains in  $(0, 1)$ , and  $u(t) \in \mathbb{R}^{n_u}$ ,  $n_u = 1$ .

To discretize (3.6.2) in time, I use the backward Euler method with time step size  $\Delta t = T/K$  and time steps  $t_k = \Delta t k$ ,  $k = 0, \dots, K$ . This leads to the problem

$$\text{minimize } \frac{\Delta t}{2} \sum_{k=0}^{K-1} (y_{k+1} - \hat{y}(t_{k+1}))^T Q (y_{k+1} - \hat{y}(t_{k+1})) + \frac{\Delta t}{2} \sum_{k=0}^{K-1} u_{k+1}^T R u_{k+1} \quad (3.6.3a)$$

subject to

$$(M + \Delta t A) y_{k+1} = \Delta t B u_{k+1} + \Delta t M y_k + \Delta t f(t_{k+1}), \quad k = 0, \dots, K-1, \quad (3.6.3b)$$

$$y_0 = y_{\text{given}}. \quad (3.6.3c)$$

The fully discretized problem (3.6.3) is of the type (3.1.1) if I set

$$\begin{aligned} A_k &= \Delta t (M + \Delta t A)^{-1} M, & B_k &= \Delta t (M + \Delta t A)^{-1} B, & c_k &= \Delta t (M + \Delta t A)^{-1} f(t_{k+1}), \\ Q_k &= \Delta t Q, & R_k &= \Delta t R, & d_k &= -\Delta t Q \hat{y}(t_k), & e_k &= 0, \end{aligned}$$



and perform an index shift  $u_{k+1} \rightarrow u_k$ . The matrices  $(M + \Delta t A)^{-1}M$  and  $(M + \Delta t A)^{-1}B$  are never formed explicitly, but linear systems with matrix  $(M + \Delta t A)$  are solved whenever matrix-vector products of the type  $(M + \Delta t A)^{-1}Mv$  or  $(M + \Delta t A)^{-1}Bw$  have to be computed.

On the computations I use  $T = 10$ ,  $\beta = 0.1$ ,  $\kappa = 0.001$ ,  $v = 5$ ,  $\gamma = 1$ ,  $f(x, t) = 0$ ,  $y_0(x) = 0$ , and  $\hat{y}(t) = \sin(t^2)$ . Furthermore, the discretization uses  $n_y = 128$  spatial subdomains in  $(0, 1)$ , and  $K = 1000$  time steps.

To start the parallel-in-time gradient-type method iterations, it is needed to initialize time subinterval boundary values  $y_{K_i}^{(-1)}, p_{K_i}^{(-1)}$  for  $i = 1, \dots, N - 1$ . In the numerical examples we provide, large number of iterations are needed for convergence, which is also the case with fixed step size classical gradient method. Each iteration does not change control variable significantly and, loosely speaking,  $2N - 1$  iterations will clear the effect of difference choice of  $y_{K_i}^{(-1)}, p_{K_i}^{(-1)}$  for  $i = 1, \dots, N - 1$ . Therefore, we do not investigate the effect of  $y_{K_i}^{(-1)}, p_{K_i}^{(-1)}$  for  $i = 1, \dots, N - 1$  on the convergence and focus on the difference choice of step size  $\alpha$ .

The speed of convergence of the parallel gradient-type method depends the step size  $\alpha$ . To estimate the optimal step size for a given number  $N$  of time subdomains, I conduct the following experiment. Given a number of time subdomains  $N$ , I subdivide  $0, \dots, K$  into approximately equally sized groups. In the case where  $N$  does not divide the number of time steps  $K$ , I split the time steps so that  $0 \leq (K_i - K_{i-1}) - (K_j - K_{j-1}) \leq 1$  for all  $1 \leq i < j \leq N$ . Then for a number of equally distributed time steps  $\alpha \in (0.3, 5)$  I run the parallel gradient-type algorithm started with zero initial guess until the error between computed control  $\mathbf{u}^{(j)}$  and exact control  $\mathbf{u}^{(*)}$ , computed solving (3.6.3) with high accuracy, is less  $10^{-6}$ , or a maximum number of iterations is exceeded.

For  $N \in \{1, \dots, 100\}$  I report the speed-ups in Figure 3.7. The speed-up is measured by number of iterations of the parallel-in-time gradient-type method for various  $N$  divided by the number of iterations of the parallel-in-time gradient-type method

for various  $N$  over the number of cores  $N$ . Thus, if  $I(N)$  is the number of iterations needed by parallel-in-time method with  $N$  subdomains and cores, one speed-up curve shows  $I(1)/(I(N)/N)$ . Although this ignores communication cost it is a good indicator of actual speed-up. For this 1D problem, I do not have the timing for a parallel implementation I will show the actual timing of a parallel implementation of the parallel-in-time gradient-type method for the 3D problem in the next numerical example in Subsection 3.6.2. In Figure 3.7, there is speed-up, but the strong scaling is not demonstrated in this case. In the region of 2 to 40 time subdomains, a linear speed-up of slope around 0.35 is observed, which indicates a 35% parallel efficiency. Note that the step size in each specific subdomain setting is obtained by search on a step size grid for the fastest convergence. In the linear speed-up region of 2 to 40 time subdomains, the step sizes used are roughly the same according to Figure 3.8.

For  $N \in \{1, \dots, 100\}$  I also report the step size  $\alpha$  for the parallel gradient-type algorithm requires the fewest iterations. These step size  $\alpha$  are shown in Figure 3.8. In the region of 2 to 40 time subdomains, the best step size remain roughly constant compared to decreasing step size for number of time subdomains greater than 40. As the number of time subdomain increases, usually, the best step size decreases. The best step size for a specific number of time subdomains may be bad for a case with more time subdomains. For example, according to Figure 3.8, the step size 0.8 is good for 2 to 40 time subdomains, but it leads to divergence, by numerical experiments, when applied to cases with 50 or more subdomains. In this 1D example, there is a significant drop in best step size from the 1 time subdomain (classical gradient method) to 2 time subdomain, which is not the case for the 3D example in Subsection 3.6.2. If I look at Figure 3.7 and Figure 3.8 together, it is easy to find that the linear speed-up region and the constant best step size region coincide. I also note that, of course, the optimal step sizes depend on other problem data as well and, thus, step sizes that are good for the 1D example problem (3.6.1) may not be good for other problems, such as (3.6.4) below.

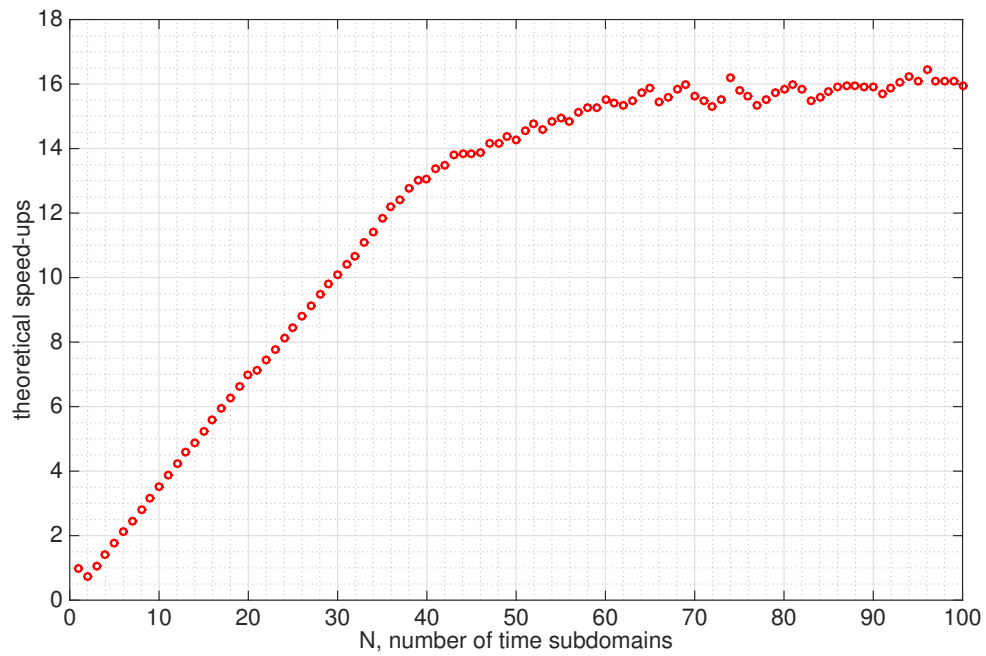


Figure 3.7: Speed-ups of the parallel-in-time gradient-type method applied to the 1D optimal control problem. If  $I(N)$  is the number of iterations needed by the parallel-in-time method with  $N$  subdomains and cores, the speed-up curve shows  $I(1)/(I(N)/N)$ .

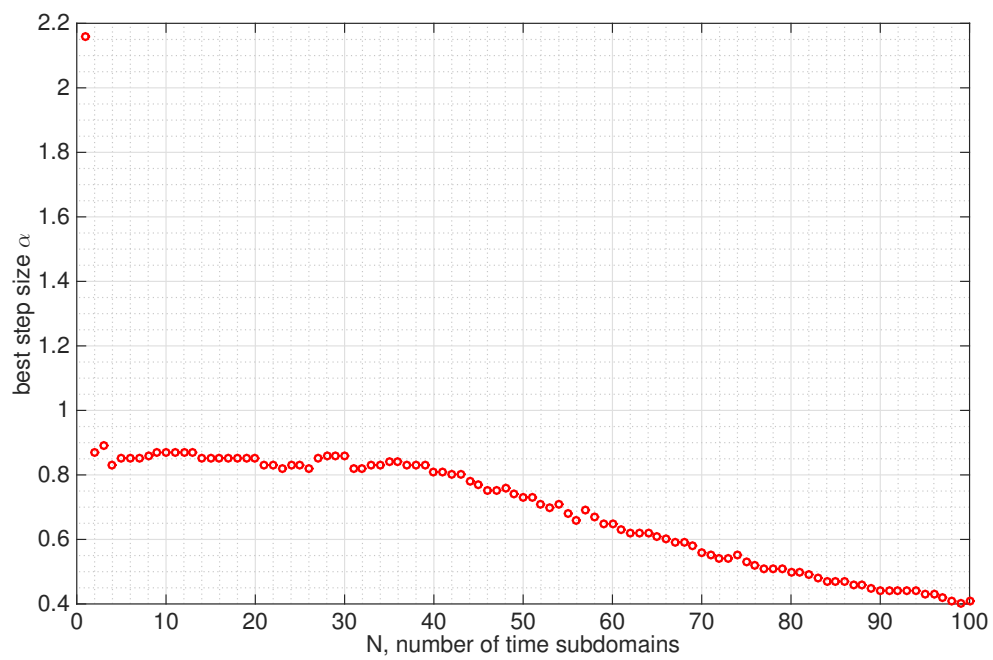


Figure 3.8: Optimal step-sizes  $\alpha$  for varying number  $N$  of time subdomains for the 1D example problem.

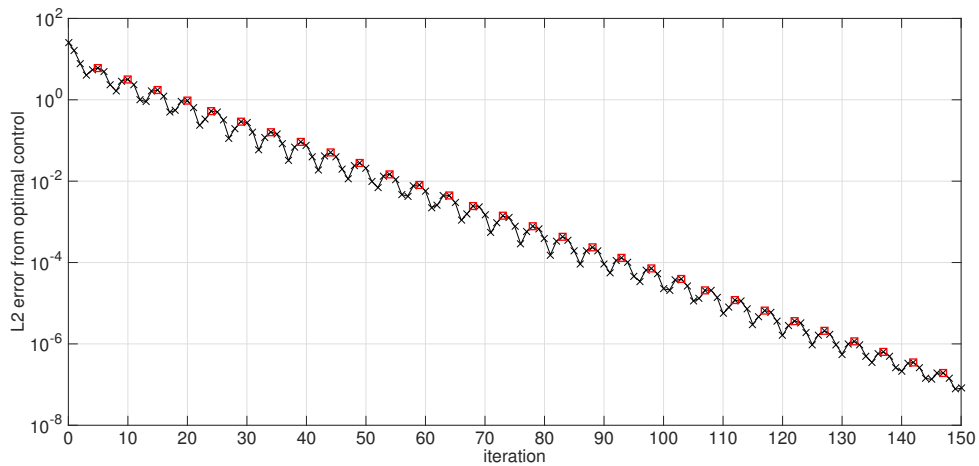


Figure 3.9: Distance between the optimization control variable trajectory and the optimal control. In this example, 10 time subdomains with optimal step size 0.87 are used. Nonmonotonic convergence is observed and there is a periodic oscillation pattern. Red squares mark the iteration where oscillation reaches its peaks. In this 10 time subdomain case, the peaks happen every 5 iterations on average. See Figure 3.10 for control errors recorded on optimization trajectory on three consecutive peaks.

Figure 3.9 and Figure 3.10 illustrate that using the optimal step size, the convergence is not monotonic in the distance to optimal control. In Figure 3.10, it can be seen that in the optimization iterations, the control error near the time subdomain boundaries is significantly larger than error away from the time subdomain boundaries and the error oscillates periodically.

### 3.6.2 3D Distributed Control: Parallel Implementation, Strong Scaling

I consider an optimal control problem governed by an advection diffusion reaction PDE posed on the spatial domain  $\Omega \subset \mathbb{R}^3$  and time domain  $(0, T)$ . Let  $\Omega_o \subset \Omega$  be the observation region and let  $\Omega_c \subset \Omega$  be the domain on which the control is

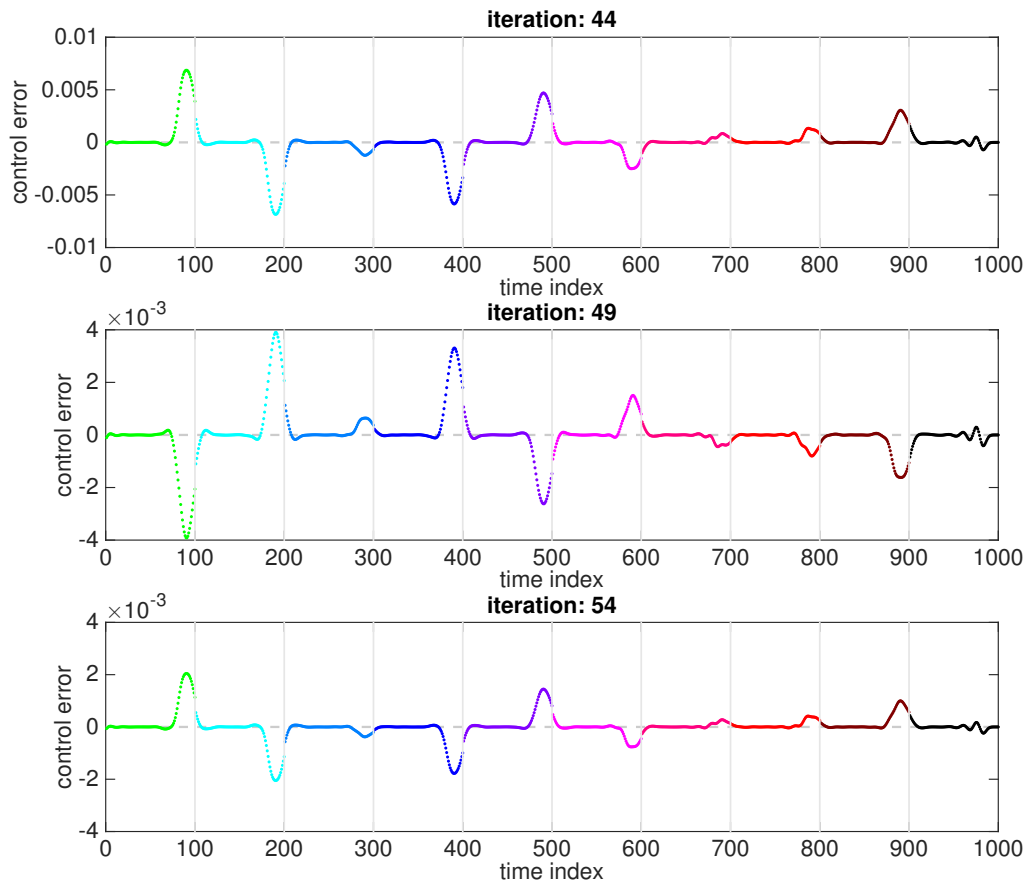


Figure 3.10: The control error, i.e. the difference between the control in the optimization iterations and the precomputed control in the optimal solution, at optimization iteration 44, 49, and 54. They are at the error peaks as depicted in Figure 3.9 as red boxes. Ten time subdomains with optimal step size 0.87 are used.

applied. Let  $\chi_{\Omega_c}$  denote the indicator function of the control domain  $\Omega_c$ . Given scalars  $\kappa > 0$ ,  $\beta > 0$ ,  $\gamma \geq 0$ , the advection  $v \in \mathbb{R}^3$ , and functions  $\hat{y} \in L^2(\Omega_o \times (0, T))$ ,  $f \in L^2(\Omega \times (0, T))$ , the optimal control problem is

$$\text{minimize } \frac{1}{2} \int_0^T \int_{\Omega_o} (y(x, t) - \hat{y}(x, t))^2 dx dt + \frac{\beta}{2} \int_0^T \int_{\Omega_c} u^2(x, t) dx dt \quad (3.6.4a)$$

subject to

$$\frac{\partial y(x, t)}{\partial t} - \kappa \Delta y(x, t)$$

$$+ v \cdot \nabla y(x, t) + \gamma y(x, t) = f(x, t) + \chi_{\Omega_c}(x) u(x, t), \quad x \in \Omega, t \in (0, T), \quad (3.6.4b)$$

$$\nabla y(x, t) \cdot n = 0, \quad x \in \partial\Omega, t \in (0, T), \quad (3.6.4c)$$

$$y(x, 0) = y_{\text{given}}(x), \quad x \in \Omega. \quad (3.6.4d)$$

This problem has a unique solution  $u \in L^2(\Omega_c \times (0, T))$  and corresponding state  $y \in W(0, T)$ . See, e.g., [Trö10, Ch.3] or [Lio71] for details.

I consider  $\Omega = (0, 1)^3$ . The optimal control problem is discretized in space using a standard cell-centered finite volume method with hexahedral cells of size  $(1/n_1) \times (1/n_2) \times (1/n_3)$  [EGH00, Sec. 3.3]. I assume that the observation region  $\Omega_o$  is a hexahedron given as the union of  $\ell$  cells and that the control region  $\Omega_c$  is a union of hexahedra given as the union of  $n_u$  cells. This leads to the semidiscretized problem (3.6.2) where  $y(t) \in \mathbb{R}^n$ ,  $n_y = n_1 n_2 n_3$ , and  $u(t) \in \mathbb{R}^{n_u}$ .

To discretize (3.6.2) in time, I use the backward Euler method with time step size  $\Delta t = T/K$  and time steps  $t_k = \Delta t k$ ,  $k = 0, \dots, K$ . This leads to the fully discretized problem (3.6.3), which is of the type (3.1.1).

In the experiments, GMRES with a multigrid preconditioner is used to solve all of the linear systems. The implementation uses Epetra linear algebra libraries, AztecOO linear solvers, and ML multigrid preconditioning packages from the Trilinos Project [HBH<sup>+</sup>05] to solve the linear systems with matrices  $M + \Delta t A$  and  $(M + \Delta t A)^T$ . Computations are performed on the Rice University DAVinCI cluster.

For the numerical example I choose the control region  $\Omega_c = (0.1, 0.3) \times (0.2, 0.8) \times (0.2, 0.8) \cup (0.7, 0.9) \times (0.2, 0.8) \times (0.2, 0.8)$ , the observation region  $\Omega_o = (0.4, 0.6) \times$

$(0, 1) \times (0, 1)$ , and the final time  $T = 8$  or  $T = 16$ . In the objective (3.6.4a) I have  $\hat{y}(x, t) = 10[(x_2 - 0.5) \cos(2\pi t) + (x_3 - 0.5) \sin(2\pi t)]^3$  and  $\beta = 1e - 6$ . In the PDE (3.6.4b),  $\kappa = 0.1$ ,  $v = [1, 1, 1]^T$ ,  $\gamma = 0.01$ , and  $y_{\text{given}}(x) = 0$ . For discretization, I use  $K = 200$  (if  $T = 8$ ) or  $K = 400$  (if  $T = 16$ ) time steps and a spatial discretization  $n_1 = n_2 = n_3 = 10$ .

So far, there is not a good a-priori way of determining a good step size for the gradient type method. In this experiment, I select the step size by examining the iteration for a few different step sizes and then selecting the  $\alpha$  from these trials. For the case  $T = 8$ ,  $K = 200$ , this resulted in the step size  $\alpha = 2400$  for  $N = 1, 4, 6, 8, 12, 16, 20$  time subdomains, and  $\alpha = 2060, 1860, 1500, 1000$  for  $N = 25, 30, 40, 60$ . For the case  $T = 16$ ,  $K = 400$ , this resulted in the step size  $\alpha = 950$  for  $N = 1, 30, 40, 50$  time subdomains, and  $\alpha = 900, 800, 650$  for  $N = 60, 70, 80$ . In comparison, in the numerical examples, the classical gradient method uses step size  $\alpha = 2400$  in the  $T = 8$ ,  $K = 200$  case; and uses step size  $\alpha = 950$  in the  $T = 16$ ,  $K = 400$  case. They are roughly the optimal fixed step size by trials. Note that these step sizes coincide with the optimal fixed step size in the parallel-in-time gradient-type method with not too many time subintervals.

Figure 3.11 shows the speeds-ups. For each of the two cases ( $T = 8$ ,  $K = 200$  and  $T = 16$ ,  $K = 400$ ) I include two speeds-up curves. One shows speed-up measured by number of iterations of the parallel-in-time gradient-type method for various  $N$  divided by the number of iterations of the parallel-in-time gradient-type method for various  $N$  over the number of cores  $N$ . Thus, if  $I(N)$  is the number of iterations needed by parallel-in-time method with  $N$  subdomains and cores, one speed-up curve shows  $I(1)/(I(N)/N)$ . Although this ignores communication cost it is a good indicator of actual speed-up, which is shown in the other curve. The actual speed-up is measured by run time of the gradient method divided by the run time of a parallel implementation of parallel-in-time gradient-type method for various  $N$ . Thus, if  $t(N)$  is the run time required by parallel-in-time method with  $N$  subdomains and



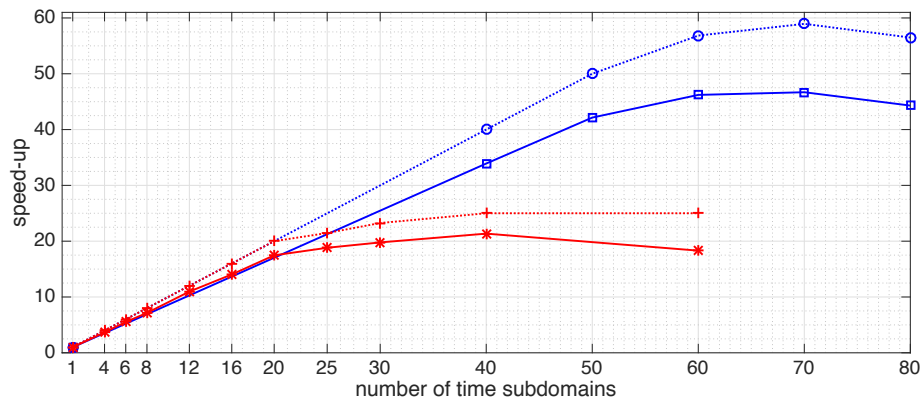


Figure 3.11: Speed-ups of the parallel-in-time gradient-type method for two examples problems with end time  $T = 8$  and  $K = 200$  time steps (red curves), and with end time  $T = 16$  and  $K = 400$  time steps (blue curves). For each case two speed-up curves are shown. If  $I(N)$  denotes the number of iterations needed by parallel-in-time method with  $N$  subdomains and cores, the speed-up curves indicated by ‘ $\dots \circ$ ’ and ‘ $\dots +$ ’ show  $I(1)/(I(N)/N)$ . If  $t(N)$  denotes the run time needed by parallel-in-time method with  $N$  subdomains and cores, the curves indicated by ‘ $-\square$ ’ and ‘ $-*$ ’ show  $t(1)/t(N)$ . Excellent speed-ups are obtained for up to  $N = 20$  time domains when  $T = 8$  and  $K = 200$  and for up to  $N = 50$  time domains when  $T = 16$  and  $K = 400$ . The fixed step size for both of the classical gradient method and the parallel-in-time gradient-type method is chosen by trials. The fixed step size with fastest convergence is used. Refer to Table 3.2 and Table 3.3 for details.

cores, the actual speed-up is  $t(1)/t(N)$ . The number  $N$  of time-subdomains changes the parallel-in-time gradient-type method and its convergence rate. If the number of time-subdomains is too large, convergence deteriorates and negatively impact speed-up obtained by parallelizing the work performed in each iteration of the parallel-in-time gradient-type method. For the problem settings I observe excellent speed-ups for up to  $N \approx 20$  and  $N \approx 50$  time subdomains.

However, in this linear-quadratic problem, the parallel-in-time gradient-type

Subinterval Number	optimal fixed step size	Number of Iterations	Time (sec.)
1 (serial gradient method)	2400	134100	9.01e+04
4	2400	134066	2.45e+04
6	2400	134066	1.62e+04
8	2400	134065	1.26e+04
12	2400	134066	8.21e+03
16	2400	134067	6.41e+03
20	2400	134065	5.15e+03
25	2060	156193	4.78e+03
30	1860	172991	4.55e+03
40	1500	214515	4.21e+03
60	1000	321776	4.92e+03

Table 3.2:  $T = 8, K = 200$  example. Number of iterations and time consumed to reduce the Infinity-norm control error from the initial guess  $1e+3$  to  $1e-3$ . Conjugate Gradient method takes only 530 iterations to reach the same level of control error.

method does not compete with Conjugate Gradient method solving the reduced control space optimality system directly. For the  $T = 8, K = 200$  problem, the parallel-in-time gradient-type method with 40 cores uses 214515 iterations to reduce control error by 6 orders of magnitude whereas Conjugate Gradient (CG) method takes only 530 iterations to reach the same level of control error. With the assumption that the parallelism in the parallel method ideally reduces the time of a single iteration to  $1/40$  of a CG iteration, the parallel method is still about 10 ( $\approx 10.12 = 214515/40/530$ ) times slower than CG. Similarly, for  $T = 16, K = 400$  example, the parallel-in-time gradient-type method with 70 cores uses 192274 iterations to reduce control error by 3 orders of magnitude whereas CG uses only 340 iterations. The parallel method is still about 8 ( $\approx 8.08 = 192274/70/340$ ) times slower than CG. So, this is not a practical method in this example problem.

Subinterval Number	optimal fixed step size	Number of Iterations	Time (sec.)
1 (serial gradient method)	950	161900	2.20e+05
30	950	161900	8.51e+03
40	950	161900	6.50e+03
50	950	161900	5.23e+03
60	900	170900	4.77e+03
70	800	192274	4.72e+03
80	650	229289	4.97e+03

Table 3.3:  $T = 16, K = 400$  example. Number of iterations and time consumed to reduce the Infinity-norm control error from the initial guess  $1e+3$  to  $1e0$ . Conjugate Gradient method takes only 340 iterations to reach the same level of control error.

The Figure 3.12 and Figure 3.13 show the 1 second trace plots of a 8 time subdomains and a 16 time subdomains run of the parallel-in-time gradient-type method. The forward and backward computation in each time subdomain consume roughly the same amount of time. When the number of time subdomains is doubled, the forward and backward computation per iteration halves. However, the time consumed by the data communication and some other serial tasks stays the same, which makes up more proportion of the overall time consumption. This is one of the reasons why the theoretical and actual speed-ups are different in Figure 3.11.

It is also important to note that the speed-up due to time decomposition multiplies existing speed-up in the solution of the systems with matrices  $M + \Delta tA$  or  $(M + \Delta tA)^T$  that arise in the solutions of time subdomain state and adjoint equations. The spatial discretization is small so that no such parallelization in the time-stepping was useful. An example where spatial parallelization is applied upon the parallel-in-time gradient-type method is presented in a reservoir water flooding optimization example in Example 2 of Section 6.3.2.



Figure 3.12: Trace graph of 1 second of running time, 8 cores, generated by the HPCTOOLKIT [ABF<sup>+</sup>10]. The horizontal axis is the time axis. Eight horizontal rows represent computation timing of eight cores. The top row is for the first time subdomain, etc. One optimization iteration consists of two major blocks, one purple and one brown. The purple blocks are for forward computation, the brown blocks are for backward computation. The smaller green and red blocks correspond to communication and error estimation timing respectively.

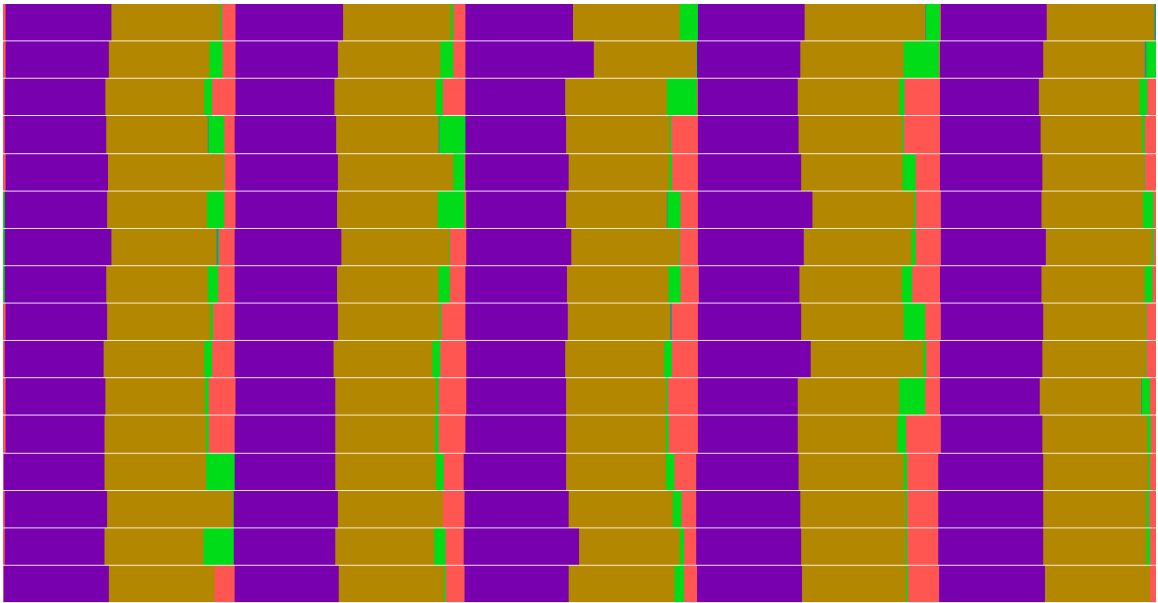


Figure 3.13: Trace graph of 1 second of running time, 16 cores, generated by the HPCTOOLKIT [ABF<sup>+</sup>10]. The horizontal axis is the time axis. Sixteen horizontal rows represent computation timing of sixteen cores. The top row is for the first time subdomain, etc. One optimization iteration consists of two major blocks, one shown in purple and the other one shown in brown. The purple blocks are for forward computations and the brown blocks are for backward computation. The smaller green and red blocks correspond to communication and error estimation timing respectively.

### 3.6.3 1D Distributed Control: Generalized Algorithm

In this section, I demonstrate properties of the generalized parallel-in-time gradient-type method by comparing its performance against the original parallel-in-time gradient-type method on a distributed control problem in a one dimension advection-diffusion-reaction system,

$$\text{minimize } \frac{1}{2} \int_0^T \int_0^1 (y(x, t) - \widehat{y}(x, t))^2 dx dt + \frac{\beta}{2} \int_0^T \int_0^1 u^2(x, t) dx dt \quad (3.6.5a)$$

subject to

$$\begin{aligned} \frac{\partial y(x, t)}{\partial t} - \kappa \frac{\partial^2 y(x, t)}{\partial x^2} \\ + v \frac{\partial y(x, t)}{\partial x} + \gamma y(x, t) = u(x, t) \quad x \in (0, 1), t \in (0, T), \end{aligned} \quad (3.6.5b)$$

$$y(0, t) = y(1, t) = 0 \quad t \in (0, T), \quad (3.6.5c)$$

$$y(x, 0) = y_0(x) \quad x \in (0, 1). \quad (3.6.5d)$$

where  $\kappa = 0.1, v = 5, \gamma = 0, \beta = 0.0005$ . Initial state

$$y_0(x) = x$$

and target function

$$\widehat{y}(x, t) = \sin(2\pi t) \sin(4\pi x)$$

First order Finite Element discretization with 32 cells are used with Backward Euler Method of 30 time steps.

This is a relatively small size discretization that, for experiment, allows one to explicitly construct the reduced control space Hessian  $\mathbf{H}$  defined in (3.1.8) and thus, by (3.1.11), find the fixed step size  $\alpha^*$  optimal for gradient method in terms of smallest spectral radius of the iteration matrix  $\mathbf{I} - \alpha\mathbf{H}$ . In this case,  $\alpha^* \approx 4239 \approx 4200$  which leads to  $\rho(\mathbf{I} - \alpha^*\mathbf{H}) \approx 0.978$ . I use the serial gradient method with fixed step size  $\alpha = 4200$  as a bench mark.

Configuration Name	para.2-core	gen.para.3-core
Forward Computation Subdomains	[0, 15], [15, 30]	[0, 15], [10, 20], [15, 30]
Backward Computation Subdomains	[0, 15], [15, 30]	[0, 15], [10, 20], [15, 30]

Table 3.4: Test Case 1 Configuration of Section 3.6.3.1. Total time steps  $K = 30$ .

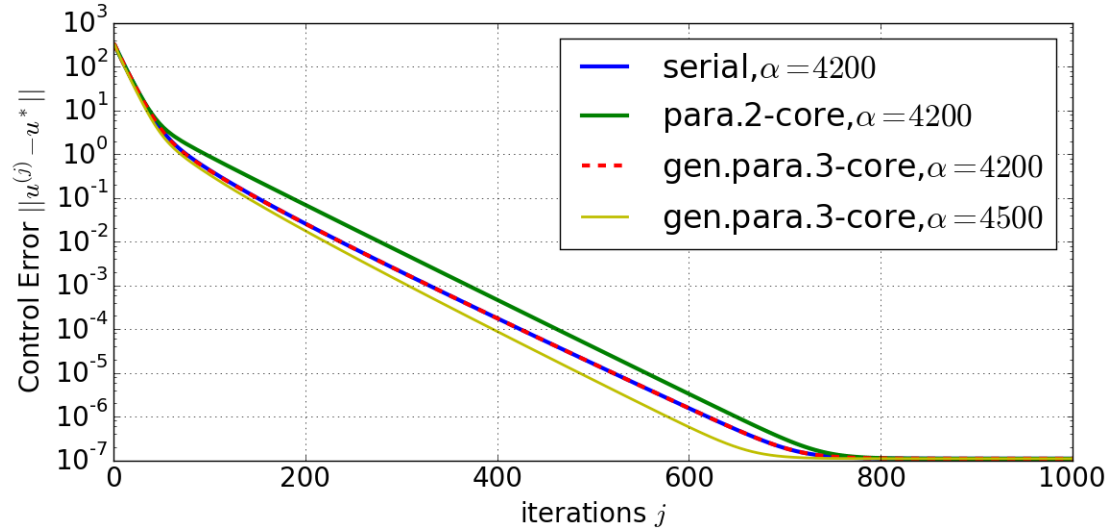


Figure 3.14: Control Error History, Test Case 1 in Section 3.6.3.1.

### 3.6.3.1 Test Case 1

In this test case, I compare the numerics of two time subdomain configurations in Table 3.4. The original parallel-in-time gradient-type “para.2-core” configuration splits the time domain into two even parts. The generalized method “gen.para.3-core” uses an extra core and adds an extra subdomain  $[10, 20]$  overlapping with the existing two subdomains. Ideally, without consideration of data communication and possibly imbalance computation load, both of the configurations consume similar time to perform one iteration which is half of the time used by the serial gradient method. One expects the one more core used in “gen.para.3-core” to bring better performance.

In Figure 3.14, I compare the convergence in terms of distance between the iterates

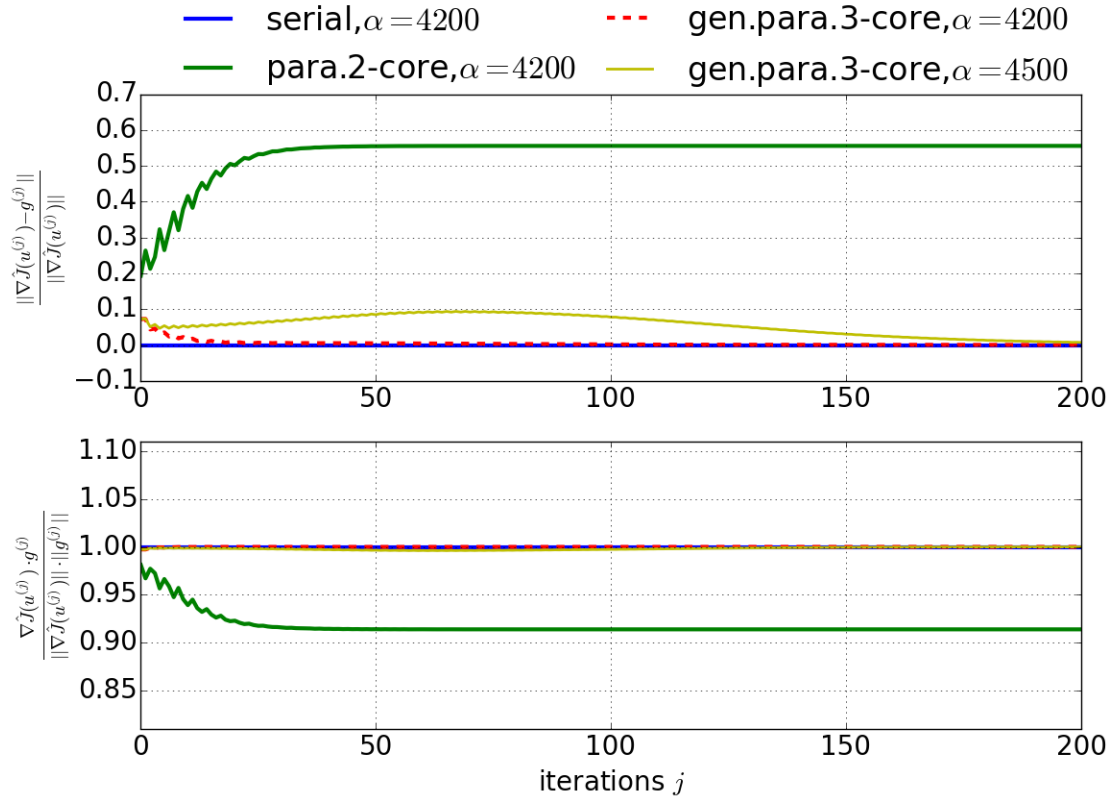


Figure 3.15: Gradient-Type Vector Comparison, Test Case 1 in Section 3.6.3.1.

and a precomputed optimal control. As mentioned, for the serial classic gradient method, the best fixed step size determined by (3.1.11) is approximately 4200. By a grid search, the best step size for “para.2-core” is also approximately 4200. But the convergence of “para.2-core” compared to the serial gradient method is slower iteration-wise. “gen para.3-core” configuration with step size  $\alpha = 4200$  converges with very similar speed as the serial gradient method. In addition, it allows bigger step size and its best step size  $\alpha = 4500$  leads to a faster convergence. Figure 3.15 compare the gradient-type vectors used in the parallel methods for control update with the exact gradient in terms of relative difference and cosine of the angle between them. Although, after the result is presented in Figure 3.14 that the “gen para.3-core” yields faster convergence than the serial gradient method, it is not clear if, for



Configuration Name	para.3-core	gen.para.5-core
Forward Computation Subdomains	[0, 10], [10, 20], [20, 30]	[0, 10], [5, 15], [10, 20] [15, 25], [20, 30]
Backward Computation Subdomains	[0, 10], [10, 20], [20, 30]	[0, 10], [5, 15], [10, 20] [15, 25], [20, 30]

Table 3.5: Test Case 1 Configuration of Section 3.6.3.2. Total time steps  $K = 30$ .

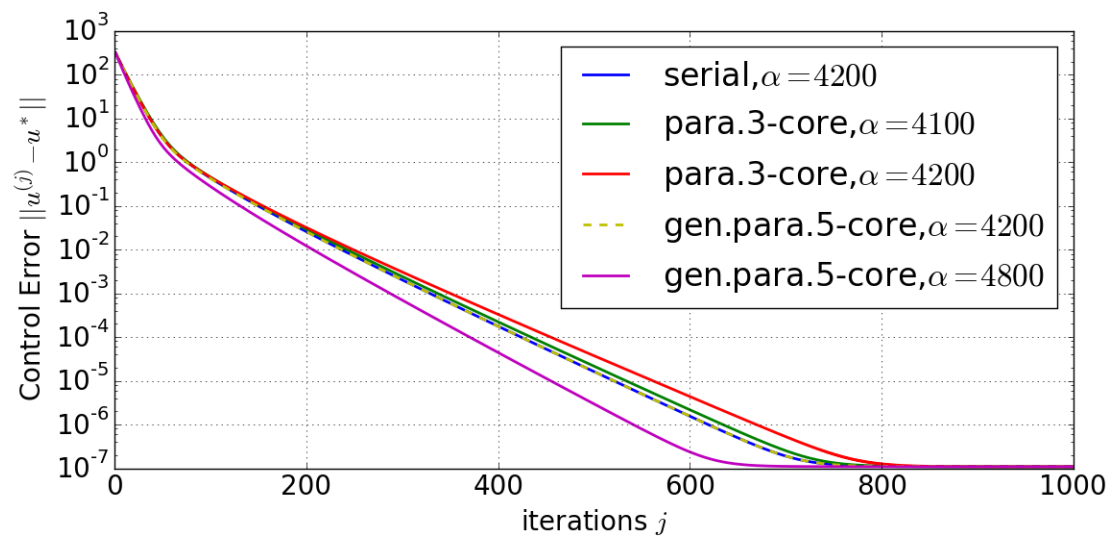


Figure 3.16: Control Error History, Test Case 2 in Section 3.6.3.2.

gradient-type method, the negative exact gradient is the best update direction, it can be seen that with step size  $\alpha = 4200$ , compared to “para.2-core”, the generalized method “gen.para.3-core” produces gradient-type vectors much more similar to the true gradient.

### 3.6.3.2 Test Case 2

In this test case, I compare a 3-subdomain original parallel-in-time algorithm with a 5-subdomain generalized algorithm, as in Table 3.5.

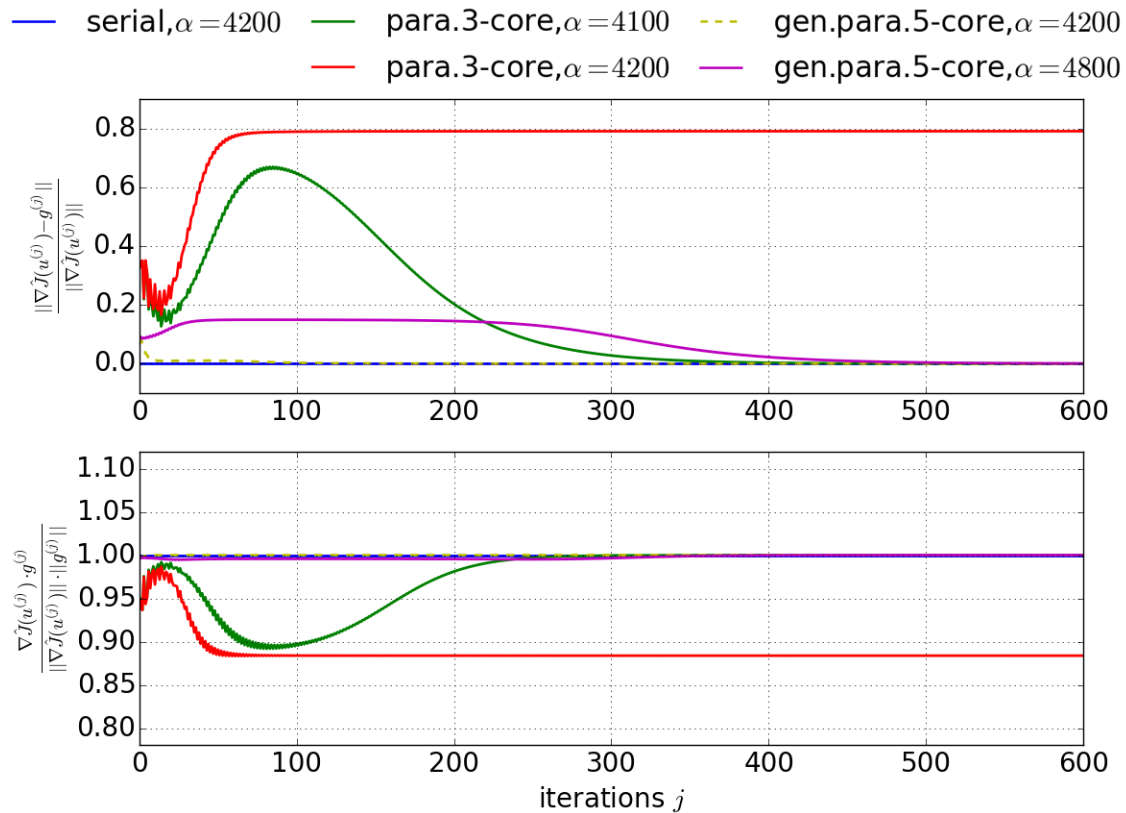


Figure 3.17: Gradient-Type Vector Comparison, Test Case 2 in Section 3.6.3.2.

In Figure 3.16, it can be seen, In this case, the best step size of “para.3-core” on the grid of granularity 100 is 4100 compared to the best step size for the serial method 4200. “gen para.5-core” has best step size 4800. Similar to the test case in Section 3.6.3.1, Figure 3.17 demonstrate, by comparing the dotted yellow curve and the red curve, the extra 2 cores in “gen para.5-core” helps to maintain the gradient-type vector similar to the true gradient.

## 3.7 Summary

In this chapter, I introduced the idea of the parallel-in-time gradient-type method using a linear quadratic model problem. Then, I generalized the newly introduced

method for more flexible partition of forward/backward computation domains. I interpreted the generalized method as a multiple part splitting method [de 76, dN81] in the reduce control space. To prove the convergence of the generalized method with fixed sufficiently small step size, I used a spectra radius argument of an implicitly constructed iteration matrix as a block companion matrix [DTW71].

I presented numerical examples applying the parallel gradient-type method to optimization problems governed by linear advection-diffusion-reaction systems.

- One example demonstrates the strong scaling of the parallel gradient-type method with up to 50 cores. In other words, in this example, compared to to classical gradient method with fixed step sizes, the parallel gradient-type method has speed-ups of 50 times with 50 time subdomains. However, for solving linear-quadratic optimization problems, it still does not compete with Conjugate Gradient method directly solving the reduced control space optimality system in terms of computation time.
- I showed the influence of the number of subdomains on the optimal fixed step size. When the number of subdomains increases beyond a threshold, smaller step size must be used for convergence, which may have a negative impact on convergence speed.
- I illustrated the non-monotonic convergence of the parallel gradient-type method with optimal step size. The distance of control variables to the optimal solution is not monotonically decreasing.
- I demonstrated that by adding overlapping computation subdomains, the generalized parallel-in-time gradient-type method produces gradient-type vector more similar to the true gradient compared to the basic gradient-type method without adding overlapping subdomains.

# Chapter 4

## Multiple-Shooting Formulations

In this chapter, I use a multiple shooting point of view to analyze the proposed parallel-in-time gradient-type method.

First, in Section 4.1, I introduce the multiple-shooting formulation of the optimization problem (1.0.1) in the continuous time setting. Then, in Section 4.2, I reformulate the optimization problem (3.1.1) in the fully discrete linear quadratic setting as in Chapter 3 and provide an alternative proof of convergence for linear-quadratic problems using the multiple-shooting formulation in Section 4.3.

In Appendix A, I also include several discussions closely related to the multiple-shooting formulation. These discussions bring insights of the parallel-in-time gradient-type method. However, I did not draw a clear conclusion from them. In Section A.1, two unsuccessful attempts to prove the convergence of parallel-in-time gradient-type method combined with metric projection using spectral radius argument are presented to show what is not possible (A valid convergence proof for the projected algorithm is in Chapter 5). In Section A.2, I present numerical experiment demonstration of the behavior of the spectral radius of an iteration matrix that determines the convergence speed of the parallel method. In Section A.3, I show parallel performance results, without theoretical explanation, of a parallel Krylov subspace method which is closely related to the parallel-in-time gradient-type method in the multiple shooting

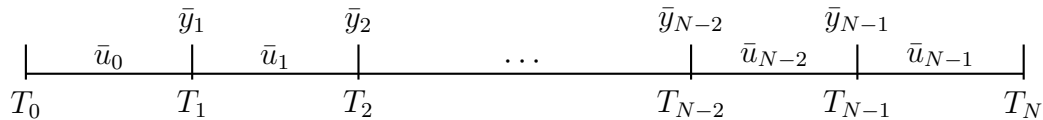


Figure 4.1: Illustration of direct multiple-shooting optimization variables

formulation. In some cases, the parallel Krylov subspace method has perfect parallel speed-up compared to the serial Conjugate Gradient method.

Some of the material of this chapter is developed in [BDH16].

## 4.1 Continuous Time Multiple-Shooting Formulation

Pick  $T_0, \dots, T_N$  such that

$$0 = T_0 < T_1 < \dots < T_{N-1} < T_N = T$$

and split the time domain into  $N$  subdomains.

I reformulate the problem (1.0.1) in a direct multiple-shooting manner so that the optimization variables are

1. control variable  $u$  restricted to time subdomain  $(T_i, T_{i+1})$ ,  $\bar{u}_i$ , for  $i = 0, \dots, N-1$ ;
2. initial condition of state variable  $y$  at the beginning of the time subdomain  $(T_i, T_{i+1})$ ,  $\bar{y}_i$ , for  $i = 1, \dots, N-1$ .

See Figure 4.1. Extra state continuity constraints are also enforced at each time subdomain boundary. For notation simplicity, define  $\bar{y}_0 = y_{\text{given}}$ . The problem (1.0.1) is rearranged into the following multiple-shooting form

$$\min_{\bar{y}, \bar{u}} \sum_{i=0}^{N-1} \int_{T_i}^{T_{i+1}} J(y_i(t; \bar{y}_i, \bar{u}_i), \bar{u}_i(t), t) dt \quad (4.1.1a)$$

$$\text{subject to} \quad y_i(T_{i+1}; \bar{y}_i, \bar{u}_i) = \bar{y}_{i+1} \quad i = 0, \dots, N-2 \quad (4.1.1b)$$

where, for  $i = 0, \dots, N - 1$ ,  $y_i(t; \bar{y}_i, \bar{u}_i)$  denotes the solution of the state equation on subdomain  $(T_i, T_{i+1})$ ,

$$\begin{aligned} \frac{d}{dt}y(t) &= F(y(t), \bar{u}_i(t), t) \quad t \in (T_i, T_{i+1}) \\ y(T_i) &= \bar{y}_i \end{aligned} \quad (4.1.2)$$

Introduce Lagrange multiplier,  $\bar{p}_0, \dots, \bar{p}_{N-1}$ , and the Lagrangian of (4.1.1)

$$\mathcal{L}(\bar{y}, \bar{u}, \bar{p}) = \sum_{i=0}^{N-1} \int_{T_i}^{T_{i+1}} J(y_i(t; \bar{y}_i, \bar{u}_i), \bar{u}_i(t), t) dt + \sum_{i=0}^{N-2} \bar{p}_{i+1}^T (y_i(T_{i+1}; \bar{y}_i, \bar{u}_i) - \bar{y}_{i+1}) \quad (4.1.3)$$

To find gradient of  $\mathcal{L}$  with respect to  $\bar{u}, \bar{y}$ , decouple  $y_i$  from  $\bar{y}_i, \bar{u}_i$  and define

$$\begin{aligned} \hat{\mathcal{L}}(\bar{y}, \bar{u}, \bar{p}, y, p) &= \sum_{i=0}^{N-1} \int_{T_i}^{T_{i+1}} J(y_i(t), \bar{u}_i(t), t) dt + \sum_{i=0}^{N-2} \bar{p}_{i+1}^T (y_i(T_{i+1}) - \bar{y}_{i+1}) \\ &+ \sum_{i=0}^{N-1} \int_{T_i}^{T_{i+1}} p_i(t)^T (F(y_i(t), \bar{u}_i(t), t) - \frac{d}{dt}y_i(t)) dt \\ &+ \sum_{i=0}^{N-1} \mu_i^T (y_i(T_i) - \bar{y}_i) \end{aligned} \quad (4.1.4)$$

Setting derivatives with respect to  $y$  to zeros, for  $i = 0, \dots, N - 1$ , leads to the adjoint PDE of adjoint variables  $p_i$  and  $\mu_i$ ,

$$\frac{d}{dt}p_i(t) = -F_y(y_i(t), \bar{u}_i(t), t)^T p(t) + J_y(y_i(t), \bar{u}_i(t), t) \quad t \in (T_i, T_{i+1}) \quad (4.1.5a)$$

$$p_i(T_{i+1}) = \bar{p}_{i+1} \quad (4.1.5b)$$

and  $\mu_i = -p_i(T_i)$ . In (4.1.5), to simplify notation,  $\bar{p}_N \stackrel{\text{def}}{=} 0$ . I use  $p_i(t; \bar{y}_i, \bar{u}_i, \bar{p}_{i+1})$  to denote the solution of (4.1.5). Then, the gradients of (4.1.3) can be written as

$$\begin{aligned} \nabla_{\bar{p}_{i+1}} \mathcal{L}(\bar{y}, \bar{u}, \bar{p}) &= y_i(T_{i+1}; \bar{y}_i, \bar{u}_i) - \bar{y}_{i+1} & i = 0, \dots, N - 2 \\ \nabla_{\bar{y}_i} \mathcal{L}(\bar{y}, \bar{u}, \bar{p}) &= p_i(T_i; \bar{y}_i, \bar{u}_i, \bar{p}_{i+1}) - \bar{p}_i & i = 1, \dots, N - 1 \\ \nabla_{\bar{u}_i} \mathcal{L}(\bar{y}, \bar{u}, \bar{p})(t) &= F_u(y_i(t), \bar{u}_i(t), t)^T p_i(t; \bar{y}_i, \bar{u}_i, \bar{p}_{i+1}) + J_u(\bar{y}_i, \bar{u}_i, t) \\ & i = 0, \dots, N - 1; \quad t \in (T_i, T_{i+1}) \end{aligned} \quad (4.1.6)$$

by which the connection between the state/adjoint jumps and the Lagrangian gradient is exposed. In the next section, analogous reformulation is done for the discrete time linear-quadratic problem.

## 4.2 Discrete Time Multiple-Shooting Formulation

To reformulate (3.1.1), I introduce new set of optimization variables

1. For  $i = 0, \dots, N - 1$ ,  $\bar{u}_i$  consisting of  $\bar{u}_{i,K_i}, \dots, \bar{u}_{i,K_{i+1}-1}$ , which can be considered as the original control variable restricted to steps  $K_i, \dots, K_{i+1} - 1$ ;
2. For  $i = 1, \dots, N - 1$ ,  $\bar{y}_i$  which is the initial condition of state variable at the beginning of the time subdomain  $K_i, \dots, K_{i+1}$ .

Define  $\bar{y}_0 \stackrel{\text{def}}{=} y_{\text{given}}$  for notation simplicity and note that  $\bar{y}_0$  is not an optimization variable but a constant. Then, for  $i = 0, \dots, N - 1$ , let  $y_i(\bar{y}_i, \bar{u}_i)$  that consists of

$$y_{i,K_i}(\bar{y}_i, \bar{u}_i), \dots, y_{i,K_{i+1}}(\bar{y}_i, \bar{u}_i)$$

For  $i = 0, \dots, N - 1$ , denotes the solution of the following state equations in the subdomain  $K_i, \dots, K_{i+1}$ ,

$$\begin{aligned} y_{i,k+1} &= A_k y_{i,k} + B_k \bar{u}_{i,k} + c_k \quad k = K_i, \dots, K_{i+1} - 1 \\ y_{i,K_i} &= \bar{y}_i \end{aligned} \tag{4.2.1}$$

With  $Q_0 \stackrel{\text{def}}{=} 0, d_0 \stackrel{\text{def}}{=} 0$  for notation simplicity, problem (3.1.1) is equivalent to

$$\begin{aligned} \min_{\bar{y}, \bar{u}} \sum_{i=0}^{N-1} \sum_{k=K_i+1}^{K_{i+1}} & \left[ \frac{1}{2} y_{i,k}^T(\bar{y}_i, \bar{u}_i) Q_k y_{i,k}(\bar{y}_i, \bar{u}_i) + d_k^T y_{i,k}(\bar{y}_i, \bar{u}_i) \right. \\ & \left. + \frac{1}{2} \bar{u}_{i,k-1}^T R_{k-1} \bar{u}_{i,k-1} + e_{k-1}^T \bar{u}_{i,k-1} \right] \\ \text{subject to} \quad & y_{i,K_{i+1}}(\bar{y}_i, \bar{u}_i) = \bar{y}_{i+1} \quad i = 0, \dots, N - 2 \end{aligned} \tag{4.2.2}$$

It has Lagrangian

$$\begin{aligned} \mathcal{L}(\bar{y}, \bar{u}, \bar{p}) &= \sum_{i=0}^{N-1} \sum_{k=K_{i+1}}^{K_{i+1}} \left[ \frac{1}{2} y_{i,k}^T(\bar{y}_i, \bar{u}_i) Q_k y_{i,k}(\bar{y}_i, \bar{u}_i) + d_k^T y_{i,k}(\bar{y}_i, \bar{u}_i) \right. \\ &\quad \left. + \frac{1}{2} \bar{u}_{i,k-1}^T R_{k-1} \bar{u}_{i,k-1} + e_{k-1}^T \bar{u}_{i,k-1} \right] \\ &\quad + \sum_{i=0}^{N-2} \bar{p}_{i+1}^T (y_{i,K_{i+1}}(\bar{y}_i, \bar{u}_i) - \bar{y}_{i+1}) \end{aligned} \quad (4.2.3)$$

To compute its gradients with respect to  $\bar{y}, \bar{u}, \bar{p}$ , define

$$\begin{aligned} \hat{\mathcal{L}}(\bar{y}, \bar{u}, \bar{p}; y, p) &= \sum_{i=0}^{N-1} \sum_{k=K_i}^{K_{i+1}-1} \left[ \frac{1}{2} y_{i,k}^T Q_k y_{i,k} + d_k^T y_{i,k} + \frac{1}{2} \bar{u}_{i,k-1}^T R_{k-1} \bar{u}_{i,k-1} + e_{k-1}^T \bar{u}_{i,k-1} \right] \\ &\quad + \sum_{i=0}^{N-2} \bar{p}_{i+1}^T (y_{i,K_{i+1}} - \bar{y}_{i+1}) \\ &\quad + \sum_{i=0}^{N-1} \sum_{k=K_i}^{K_{i+1}-1} p_{i,k}^T (A_k y_{i,k} + B_k \bar{u}_{i,k} + c_k - y_{i,k+1}) \\ &\quad + \sum_{i=0}^{N-1} p_{i,K_{i+1}-1}^T (\bar{y}_i - y_{i,K_i}) \end{aligned} \quad (4.2.4)$$

Setting derivatives to zero with respect to  $y_{i,k}$  with all proper subscripts yields the following adjoint equations of adjoint variables  $p$  and  $\mu$ , for  $i = 0, \dots, N-2$ ,

$$\begin{aligned} \bar{p}_{i+1} - p_{i,K_{i+1}-1} &= 0 \\ Q_k y_{i,k} + d_k + A_k^T p_{i,k} - p_{i,k-1} &= 0 \quad i = K_{i+1} - 1, \dots, K_i \end{aligned} \quad (4.2.5)$$

For  $i = 0, \dots, N-2$ , let  $p_i(\bar{y}_i, \bar{u}_i, \bar{p}_{i+1})$ , consisting of

$$p_{i,K_i-1}(\bar{y}_i, \bar{u}_i, \bar{p}_{i+1}), \dots, p_{i,K_{i+1}-1}(\bar{y}_i, \bar{u}_i, \bar{p}_{i+1})$$

be the solution of adjoint equations

$$\begin{aligned} \bar{p}_{i+1} - p_{i,K_{i+1}-1} &= 0 \\ Q_k y_{i,k}(\bar{y}_i, \bar{u}_i) + d_k + A_k^T p_{i,k} - p_{i,k-1} &= 0 \quad k = K_{i+1} - 1, \dots, K_i \end{aligned} \quad (4.2.6)$$

Note that in (4.2.6), the adjoint variable for the state continuity constraint at time step  $K_{i+1}$  is the terminal condition for the adjoint equation at step  $K_{i+1} - 1$ .



Similarly for the last time subdomain  $N - 1$ ,

$$\begin{aligned} Q_{K_N} y_{N-1, K_N} + d_{K_N} - p_{N-1, K_N-1} &= 0 \\ Q_k y_{N-1, k} + d_k + A_k^T p_{N-1, k} - p_{N-1, k-1} &= 0 \quad k = K_N - 1, \dots, K_{N-1} \end{aligned} \quad (4.2.7)$$

let  $p_{N-1}(\bar{y}_{N-1}, \bar{u}_{N-1})$ , consisting of

$$p_{N-1, K_{N-1}-1}(\bar{y}_{N-1}, \bar{u}_{N-1}), \dots, p_{N-1, K_N-1}(\bar{y}_{N-1}, \bar{u}_{N-1})$$

be the solution of adjoint equations

$$\begin{aligned} Q_{K_N} y_{N-1, K_N}(\bar{y}_{N-1}, \bar{u}_{N-1}) + d_{K_N} - p_{N-1, K_N-1} &= 0 \\ Q_k y_{N-1, k}(\bar{y}_{N-1}, \bar{u}_{N-1}) + d_k + A_k^T p_{N-1, k} - p_{N-1, k-1} &= 0 \quad k = K_N - 1, \dots, K_{N-1} \end{aligned} \quad (4.2.8)$$

Then, analogous to the result of the continuous time case (4.1.6),

$$\frac{\partial \mathcal{L}(\bar{y}, \bar{u}, \bar{p})}{\partial \bar{y}_i} = p_{i, K_{i-1}} - \bar{p}_i \quad i = 1, \dots, N - 1 \quad (4.2.9a)$$

$$\frac{\partial \mathcal{L}(\bar{y}, \bar{u}, \bar{p})}{\partial \bar{u}_{i, k}} = R_k \bar{u}_{i, k} + e_k + B_k^T p_{i, k} \quad i = 0, \dots, N - 1; k = K_i, \dots, K_{i+1} - 1 \quad (4.2.9b)$$

$$\frac{\partial \mathcal{L}(\bar{y}, \bar{u}, \bar{p})}{\partial \bar{p}_i} = y_{i, K_{i+1}} - \bar{y}_i \quad i = 0, \dots, N - 2 \quad (4.2.9c)$$

According to (4.2.9), define the jumps in state and adjoint,

$$\hat{y} \stackrel{\text{def}}{=} \begin{bmatrix} y_{0, K_1} - \bar{y}_1 \\ y_{1, K_2} - \bar{y}_2 \\ \vdots \\ y_{N-2, K_{N-1}} - \bar{y}_{N-1} \end{bmatrix}, \quad \hat{p} \stackrel{\text{def}}{=} \begin{bmatrix} p_{1, K_1-1} - \bar{p}_1 \\ p_{2, K_2-1} - \bar{p}_2 \\ \vdots \\ p_{N-1, K_{N-1}-1} - \bar{p}_{N-1} \end{bmatrix}, \quad (4.2.10)$$

and the gradient-type vector,

$$g \stackrel{\text{def}}{=} \begin{bmatrix} R_0 \bar{u}_{0,0} + e_0 + B_0^T p_{0,0} \\ R_1 \bar{u}_{0,1} + e_1 + B_1^T p_{0,1} \\ \vdots \\ R_{K_1-1} \bar{u}_{0,K_1-1} + e_{K_1-1} + B_{K_1-1}^T p_{0,K_1-1} \\ \hline R_{K_1} \bar{u}_{1,K_1} + e_{K_1} + B_{K_1}^T p_{1,K_1} \\ R_{K_1+1} \bar{u}_{1,K_1+1} + e_{K_1+1} + B_{K_1+1}^T p_{1,K_1+1} \\ \vdots \\ R_{K_2-1} \bar{u}_{1,K_2-1} + e_{K_2-1} + B_{K_2-1}^T p_{1,K_2-1} \\ \hline \vdots \\ \hline R_{K_{N-1}} \bar{u}_{N-1,K_{N-1}} + e_{K_{N-1}} + B_{K_{N-1}}^T p_{N-1,K_{N-1}} \\ R_{K_{N-1}+1} \bar{u}_{N-1,K_{N-1}+1} + e_{K_{N-1}+1} + B_{K_{N-1}+1}^T p_{N-1,K_{N-1}+1} \\ \vdots \\ R_{K_N-1} \bar{u}_{N-1,K_N-1} + e_{K_N-1} + B_{K_N-1}^T p_{N-1,K_N-1} \end{bmatrix} \quad (4.2.11)$$

I omitted the arguments  $\bar{y}$ ,  $\bar{u}$ ,  $\bar{p}$  of  $\hat{p}$ ,  $g$  and  $\hat{y}$  when it does not lead to confusion. Then, (4.2.9) is summarized into

$$\nabla \mathcal{L} \begin{pmatrix} \bar{y} \\ \bar{u} \\ \bar{p} \end{pmatrix} = \begin{bmatrix} \hat{p}(\bar{y}, \bar{u}, \bar{p}) \\ g(\bar{y}, \bar{u}, \bar{p}) \\ \hat{y}(\bar{y}, \bar{u}, \bar{p}) \end{bmatrix} \quad (4.2.12)$$

By (4.2.12), it is clear that to find the tuple  $(\bar{y}, \bar{u}, \bar{p})$  such that there are no jumps in state/adjoint, i.e.,  $\hat{y} = 0$  and  $\hat{p} = 0$ , and the gradient-type vector  $g$  vanishes is equivalent to solve the saddle point problem

$$\nabla \mathcal{L} \begin{pmatrix} \bar{y} \\ \bar{u} \\ \bar{p} \end{pmatrix} = 0. \quad (4.2.13)$$

It is easy to verify the generalized parallel-in-time gradient-type algorithm (refer to Section 3.4 for the generalized parallel-in-time gradient-type method framework

context) with

$$\begin{aligned}
s_i^F &= K_i, & e_i^F &= K_{i+1} & i &= 0, \dots, N-1 \\
s_i^B &= K_i - 1, & e_i^B &= K_{i+1} - 1 & i &= 1, \dots, N-1 \\
s_0^B &= K_0, & e_0^B &= K_1 - 1
\end{aligned} \tag{4.2.14}$$

is implicitly performing the updating

$$\begin{bmatrix} \bar{y}^{(j+1)} \\ \bar{u}^{(j+1)} \\ \bar{p}^{(j+1)} \end{bmatrix} = \begin{bmatrix} \bar{y}^{(j)} \\ \bar{u}^{(j)} \\ \bar{p}^{(j)} \end{bmatrix} - \mathbf{R}(\alpha) \nabla \mathcal{L} \left( \begin{bmatrix} \bar{y}^{(j)} \\ \bar{u}^{(j)} \\ \bar{p}^{(j)} \end{bmatrix} \right) \tag{4.2.15}$$

where

$$\mathbf{R}(\alpha) \stackrel{\text{def}}{=} \begin{bmatrix} 0 & 0 & -I_{(N-1) \times n_y} \\ 0 & \alpha I_{K \times n_u} & 0 \\ -I_{(N-1) \times n_y} & 0 & 0 \end{bmatrix} \tag{4.2.16}$$

where the gradient is arranged in the order of  $\bar{y}$ ,  $\bar{u}$ , and  $\bar{p}$ . Note that the length of  $\bar{y}$  and  $\bar{p}$  is  $(N-1)n_y$  and the length of  $\bar{u}$  is  $Kn_u$ .  $\alpha$  is the step size parameter in parallel-in-time gradient-type method.

According to (4.2.15), the parallel-in-time gradient-type method is updating control and shooting variables by a rotated and scaled gradient to solve a saddle point problem (4.2.13). In the next section, I prove its convergence.

### 4.3 Convergence Proof by Multiple-Shooting Formulation

In Section 3.5, I have showed that with sufficiently small positive step size  $\alpha$ , the generalized parallel-in-time gradient-type method converges and therefore the iterations defined by (4.2.15) converges. In this section, I prove the convergence of (4.2.15) from the multiple shooting point of view which draws some new insights into the parallel-in-time gradient method. Note that, compared to the proof in this section, the proof given in the previous Section 3.5 for the generalized parallel-in-time gradient-type

method is for a broader class of algorithms where partitions arrangement allows a higher degree of flexibility.

After the convergence proof, at the end of section, I present discussion on how to determine if the step size in use leads to convergence according to the observable quantities during the iterations. The discussion reveals some properties of the iterations, but does not leads to a good way of such determination.

For the multiple shooting reformulated linear-quadratic problem (4.2.2), the constant valued Hessian matrix  $\mathbf{H}_{\mathcal{L}}$  of the Lagrangian  $\mathcal{L}(\bar{y}, \bar{u}, \bar{p})$  in (4.2.3) is symmetric indefinite, see Figure A.4 for an example of explicitly constructed  $\mathbf{H}_{\mathcal{L}}$ . To used a gradient based method to solve a general saddle point problem is not trivial.

For problem (4.2.2), with generic complex vectors  $\bar{y}, \bar{u}, \bar{p}$ , the mapping

$$\begin{bmatrix} \bar{y} \\ \bar{u} \\ \bar{p} \end{bmatrix} \rightarrow \nabla \mathcal{L} \left( \begin{bmatrix} \bar{y} \\ \bar{u} \\ \bar{p} \end{bmatrix} \right)$$

is affine. Let  $\bar{y}^*, \bar{u}^*, \bar{p}^*$  be the optimal solution of (4.2.2) and then

$$\nabla \mathcal{L} \left( \begin{bmatrix} \bar{y}^* \\ \bar{u}^* \\ \bar{p}^* \end{bmatrix} \right) = 0 \quad (4.3.1)$$

Therefore, by (4.3.1),

$$\nabla \mathcal{L} \left( \begin{bmatrix} \bar{y} \\ \bar{u} \\ \bar{p} \end{bmatrix} \right) = \nabla \mathcal{L} \left( \begin{bmatrix} \bar{y} \\ \bar{u} \\ \bar{p} \end{bmatrix} \right) - \nabla \mathcal{L} \left( \begin{bmatrix} \bar{y}^* \\ \bar{u}^* \\ \bar{p}^* \end{bmatrix} \right) = \mathbf{H}_{\mathcal{L}} \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} \quad (4.3.2)$$

Insert (4.3.2) into (4.2.15) and subtract the optimal solution from both sides,

$$\begin{bmatrix} \bar{y}^{(j+1)} - \bar{y}^* \\ \bar{u}^{(j+1)} - \bar{u}^* \\ \bar{p}^{(j+1)} - \bar{p}^* \end{bmatrix} = [I - \mathbf{W}(\alpha)] \begin{bmatrix} \bar{y}^{(j)} - \bar{y}^* \\ \bar{u}^{(j)} - \bar{u}^* \\ \bar{p}^{(j)} - \bar{p}^* \end{bmatrix} \quad (4.3.3)$$

with,  $\mathbf{R}(\alpha)$  defined in (4.2.16),

$$\mathbf{W}(\alpha) \stackrel{\text{def}}{=} \mathbf{R}(\alpha)\mathbf{H}_{\mathcal{L}} = \begin{bmatrix} 0 & 0 & -I_{(N-1) \times n_y} \\ 0 & \alpha I_{K \times n_u} & 0 \\ -I_{(N-1) \times n_y} & 0 & 0 \end{bmatrix} \mathbf{H}_{\mathcal{L}} \quad (4.3.4)$$

In the following of this section, I will show the spectra radius of  $[I - \mathbf{W}(\alpha)]$  is strictly less than one with sufficiently small positive  $\alpha$ . Note that since  $[I - \mathbf{W}(\alpha)]$  is asymmetric, some eigenvalues are possibly complex. To examine the spectral radius of  $[I - \mathbf{W}(\alpha)]$ , complex vectors  $\bar{y}, \bar{u}, \bar{p}$  will be involved, although in the parallel-in-time gradient-type iterations, only real vectors are generated.

First I show  $[I - \mathbf{W}(0)]$  only has eigenvalue 0 and 1. In the case of  $N$  time subdomains,  $2N - 1$  steps of parallel-in-time gradient-type method (4.2.15) with step size  $\alpha = 0$ , i.e. no control update, lead to the feasible state and adjoint variables corresponding to the control and by executing further steps no modification of shooting variables  $\bar{y}$  and  $\bar{p}$  will be made. In other words, for all complex vectors  $\bar{y}, \bar{u}, \bar{p}$  with proper dimensions,

$$[\mathbf{I} - \mathbf{W}(\alpha)]^{2N-1} \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} = [\mathbf{I} - \mathbf{W}(\alpha)]^{2N-1+j} \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} \quad j = 1, 2, 3, \dots \quad (4.3.5)$$

Because any eigenvector  $[\bar{y} - \bar{y}^*, \bar{u} - \bar{u}^*, \bar{p} - \bar{p}^*]$  corresponding to a non 0/1 eigenvalue does not satisfy (4.3.5),  $[I - \mathbf{W}(0)]$  only has 0/1 eigenvalues and thus  $\mathbf{W}(0)$  also only has 0/1 eigenvalues. To investigate the eigenvalues of  $[I - \mathbf{W}(\alpha)]$ , I focus on  $\mathbf{W}(\alpha)$  and in particular those eigenvalues of  $\mathbf{W}(\alpha)$  around zero with small  $\alpha$ . I will show with sufficiently small positive  $\alpha$ , those eigenvalues are to the right of zero, i.e., Lemma 4.3.2. Before the proofs, I state several useful observations.

- For  $\alpha \neq 0$ ,  $\mathbf{W}(\alpha)$  is nonsingular and equivalently does not have a zero eigenvalue. Since, otherwise, exist  $\bar{y}, \bar{u}, \bar{p}$  corresponding to the eigenvector of zero

eigenvalue, i.e.

$$\begin{bmatrix} -\hat{y} \\ \alpha g \\ -\hat{p} \end{bmatrix} = \mathbf{W}(\alpha) \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} = 0 \cdot \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} = 0$$

which leads to  $\hat{y} = 0$ ,  $\hat{p} = 0$ , i.e. state/adjoint feasible, and thus the true gradient  $\nabla \hat{J}(\bar{u}) = g = 0$  for  $\alpha \neq 0$ . Then,  $\bar{y} = \bar{y}^*$ ,  $\bar{u} = \bar{u}^*$ ,  $\bar{p} = \bar{p}^*$  and

$$\begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} = 0$$

which contradicts with the assumption that it is an eigenvector.

- The eigenvalues of  $\mathbf{W}(\alpha)$  around zero approach zero as  $\alpha$  goes to zero, i.e.,

$$\lim_{\alpha \rightarrow 0} \max\{|\lambda| \mid \exists x \text{ s.t. } \mathbf{W}(\alpha)x = \lambda x \text{ and } |\lambda| < 0.5\} = 0 \quad (4.3.6)$$

This is a result of continuity of polynomial roots with respect to the polynomial coefficients since  $\mathbf{W}(0)$  has 0 and 1 as its only eigenvalues.

- For  $\alpha \geq 0$ , if  $\bar{y}, \bar{u}, \bar{p}$  is corresponding to a  $\mathbf{W}(\alpha)$  eigenvector of eigenvalue  $\lambda \neq 1$ , then  $\bar{u} \neq \bar{u}^*$ . Consider

$$\begin{bmatrix} -\hat{y} \\ \alpha g \\ -\hat{p} \end{bmatrix} = \mathbf{W}(\alpha) \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} = \lambda \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} \quad (4.3.7)$$

By examining the forward/backward computation, it can be seen that, if the control  $\bar{u} = \bar{u}^*$ , then the shooting variable error is equal to the negative jump, i.e.,

$$\bar{y}_{K_1} - \bar{y}_{K_1}^* = -(y_{K_1} - \bar{y}_{K_1}) = -\hat{y}$$

since  $\bar{y}_{K_1}^* = y_{K_1}$  when control is optimal. Therefore, if  $\bar{y}_{K_1} - \bar{y}_{K_1}^* \neq 0$ , by (4.3.7),  $\lambda = 1$ . Otherwise, if  $\bar{y}_{K_1} - \bar{y}_{K_1}^* = 0$ , look at subdomain boundary at  $K_2$  and

repeat similar arguments. This argument summarizes to (4.3.8).

$$\begin{cases} \bar{u} - \bar{u}^* = 0 \\ \bar{y} - \bar{y}^* \neq 0 \end{cases} \Rightarrow \lambda = 1 \quad \text{and} \quad \begin{cases} \bar{u} - \bar{u}^* = 0 \\ \bar{y} - \bar{y}^* = 0 \\ \bar{p} - \bar{p}^* \neq 0 \end{cases} \Rightarrow \lambda = 1 \quad (4.3.8)$$

Then, if  $\lambda \neq 1$ ,

$$\bar{u} - \bar{u}^* = 0 \Rightarrow \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} = 0$$

which leads to this bullet point fact.

**Lemma 4.3.1** *If control  $\bar{u} \in \mathbb{C}^{K n_u}$  and shooting variables  $\bar{y}, \bar{p} \in \mathbb{C}^{(N-1)n_y}$  are corresponding to an eigenvector of  $\mathbf{W}(\alpha)$  with eigenvalue  $\lambda \in \mathbb{C}$ , i.e.,*

$$\begin{bmatrix} -\hat{y} \\ \alpha g \\ -\hat{p} \end{bmatrix} = \mathbf{W}(\alpha) \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} = \lambda \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} \quad (4.3.9)$$

then, exists  $r > 0, C > 0$ , for all  $0 \leq |\lambda| \leq r$ ,

$$\|\nabla \hat{J}(\bar{u}) - g\| \leq C|\lambda| \|\nabla \hat{J}(\bar{u})\| \quad (4.3.10)$$

where  $\nabla \hat{J}(\bar{u})$  is the true gradient of the reduced control space objective function.

**Proof:** In this proof, I relate both of  $\|\nabla \hat{J}(\bar{u}) - g\|$  and  $\|\nabla \hat{J}(\bar{u})\|$  to  $\|\bar{u} - \bar{u}^*\|$  respectively and thus compare their magnitude via their relationship with  $\|\bar{u} - \bar{u}^*\|$ .

First, by  $\nabla \hat{J}(\bar{u}) = \mathbf{H}(\bar{u} - \bar{u}^*)$  with  $\mathbf{H}$  being the positive definite Hessian of the reduced control space objective function defined in (3.1.8),

$$\|\nabla \hat{J}(\bar{u})\| = \|\mathbf{H}(\bar{u} - \bar{u}^*)\| \geq \sigma_{\min}(\mathbf{H}) \|\bar{u} - \bar{u}^*\| \quad (4.3.11)$$

Second, examine  $\|\nabla \hat{J}(\bar{u}) - g\|$ . The following steps maneuver the forward/back computation formula and state/adjoint jumps so that an upper bound of  $\|\nabla \hat{J}(\bar{u}) - g\|$

notation	definition	notation	definition
$\bar{u}$	control variable		
$\bar{y}$	state shooting variable	$\hat{y}$	state jump, as in (4.2.10)
$\mathbf{y}$	computed state-type variable	$\tilde{\mathbf{y}}$	$\mathbf{y} = \mathbf{L}(\mathbf{B}\bar{u} + \mathbf{c}) + \mathbf{y}_0$ , as in (3.1.5)
$\bar{p}$	adjoint shooting variable	$\hat{p}$	adjoint jump, as in (4.2.10)
$\mathbf{p}$	computed adjoint-type variable	$\tilde{\mathbf{p}}$	$\mathbf{p} = \mathbf{L}^T(\mathbf{Q}\mathbf{y} + \mathbf{d})$ , as in (3.1.6)

Table 4.1: Notations for the Proof of Lemma 4.3.1. Variables with superscript “\*” are corresponding to the optimal solution.

by  $\|\bar{u} - \bar{u}^*\|$ , (4.3.27), can be derived. Symbols  $\mathbf{B}, \mathbf{L}, \mathbf{Q}$  are defined in Section 3.1. I will use additional notations in Table 4.1.  $\mathbf{y}, \mathbf{p}$  are the state/adjoint variables corresponding to all time steps computed in the parallel-in-time gradient-type method, as opposed to shooting variables  $\bar{y}, \bar{p}$  which only represents state/adjoint at the subdomain boundaries.  $\tilde{\mathbf{y}}, \tilde{\mathbf{p}}$  are the feasible state and adjoint corresponding to the current control  $\bar{u}$ , as opposed to  $\mathbf{y}, \mathbf{p}$  which are state-type and adjoint-type vector computed in the parallel algorithm. By the formula to compute gradient and gradient-type vector,

$$\nabla \hat{J}(\bar{u}) - g = \mathbf{B}^T(\tilde{\mathbf{p}} - \mathbf{p}) \quad (4.3.12)$$

The difference between the true adjoint variables  $\tilde{\mathbf{p}}$  corresponding to  $\bar{u}$ ,  $\tilde{\mathbf{y}}$  and the adjoint type vector computed in parallel-in-time gradient-type algorithm is caused by the difference between states and the adjoint jumps on time subdomain boundaries,

$$\begin{aligned} \tilde{\mathbf{p}} - \mathbf{p} &= \mathbf{L}^T \mathbf{Q}(\tilde{\mathbf{y}} - \mathbf{y}) + \mathbf{M}_p \hat{p} \\ &= \mathbf{L}^T \mathbf{Q}(\tilde{\mathbf{y}} - \mathbf{y}) - \lambda \mathbf{M}_p(\bar{p} - \bar{p}^*) \end{aligned} \quad (4.3.13)$$



where  $\mathbf{M}_p$  characterizes the influence of the adjoint jump to the adjoint-type vectors,

$$\mathbf{M}_p \stackrel{\text{def}}{=} \left[ \begin{array}{ccc|ccc} A_1 A_2 \dots A_{K_1-1} & A_1 A_2 \dots A_{K_2-1} & & & A_1 A_2 \dots A_{K_{N-1}-1} & \\ A_2 A_3 \dots A_{K_1-1} & A_2 A_3 \dots A_{K_2-1} & & & A_2 A_3 \dots A_{K_{N-1}-1} & \\ \vdots & \vdots & \vdots & & \vdots & \\ A_{K_1-1} & A_{K_1-1} \dots A_{K_2-1} & & & A_{K_1-1} \dots A_{K_{N-1}-1} & \\ I & A_{K_1} \dots A_{K_2-1} & & & A_{K_1} \dots A_{K_{N-1}-1} & \\ \hline & 0 & A_{K_1+1} \dots A_{K_2-1} & & A_{K_1+1} \dots A_{K_{N-1}-1} & \\ & 0 & A_{K_1+2} \dots A_{K_2-1} & & A_{K_1+2} \dots A_{K_{N-1}-1} & \\ & \vdots & \vdots & \vdots & \vdots & \\ & 0 & A_{K_2-1} & & A_{K_2-1} \dots A_{K_{N-1}-1} & \\ & 0 & I & & A_{K_2} \dots A_{K_{N-1}-1} & \\ \hline & 0 & & 0 & \vdots & \vdots \\ \hline & 0 & & 0 & 0 & A_{K_{N-2}+1} A_{K_{N-2}+2} \dots A_{K_{N-1}-1} \\ & 0 & & 0 & 0 & A_{K_{N-2}+2} A_{K_{N-2}+3} \dots A_{K_{N-1}-1} \\ & \vdots & & \vdots & \vdots & \vdots \\ & 0 & & 0 & 0 & A_{K_{N-1}-1} \\ & 0 & & 0 & 0 & I \\ \hline & 0 & & 0 & 0 & 0 \\ & \vdots & & \vdots & \vdots & \vdots \\ & 0 & & 0 & 0 & 0 \end{array} \right]_{K n_y \times (N-1) n_y}$$

The difference between the true state variables  $\tilde{\mathbf{y}}$  corresponding to the control  $\bar{\mathbf{u}}$  and the state type vector computed in parallel-in-time gradient-type algorithm is the result of the state jumps,

$$\tilde{\mathbf{y}} - \mathbf{y} = \mathbf{M}_y \hat{\mathbf{y}} = -\mathbf{M}_y \lambda (\bar{\mathbf{y}} - \bar{\mathbf{y}}^*) \quad (4.3.14)$$

where

$$\mathbf{M}_y \stackrel{\text{def}}{=} \left[ \begin{array}{c|c|c|c} 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 \\ \hline I & 0 & 0 & 0 \\ A_{K_1} & 0 & 0 & 0 \\ A_{K_1+1}A_{K_1} & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ A_{K_2-2}A_{K_2-3}\dots A_{K_1} & 0 & 0 & 0 \\ \hline A_{K_2-1}A_{K_2-2}\dots A_{K_1} & I & & \\ A_{K_2}A_{K_2-1}\dots A_{K_1} & A_{K_2} & & \\ \vdots & \vdots & \vdots & \vdots \\ A_{K_3-2}A_{K_3-3}\dots A_{K_1} & A_{K_3-2}\dots A_{K_2} & & \\ \hline \vdots & \vdots & \vdots & 0 \\ \hline A_{K_{N-1}-1}\dots K_1 & A_{K_{N-1}-1}\dots K_2 & I & \\ A_{K_{N-1}}A_{K_{N-1}-1}\dots K_1 & A_{K_{N-1}}A_{K_{N-1}-1}\dots K_2 & A_{K_{N-1}} & \\ A_{K_{N-1}+1}A_{K_{N-1}}\dots K_1 & A_{K_{N-1}+1}A_{K_{N-1}}\dots K_2 & A_{K_{N-1}+1}A_{K_{N-1}} & \\ \vdots & \vdots & \vdots & \vdots \\ A_{K_{N-1}}\dots A_{K_1} & A_{K_{N-1}}\dots A_{K_2} & A_{K_{N-1}}\dots A_{K_{N-1}} & \end{array} \right]$$

Let  $[\mathbf{Z}_y]_{(N-1)n_y \times Kn_y}$  be the matrix form of the linear operator that only keep state shooting variables given input vectors of states from all time steps, then,

$$\begin{aligned} \bar{y} - \bar{y}^* &= \mathbf{Z}_y[\mathbf{y} - \mathbf{y}^*] = \mathbf{Z}_y[\tilde{\mathbf{y}} - \mathbf{M}_y\hat{y} - \mathbf{y}^*] \\ &= \mathbf{Z}_y[\mathbf{LB}(\bar{u} - \bar{u}^*) - \mathbf{M}_y\hat{y}] \\ &= \mathbf{Z}_y[\mathbf{LB}(\bar{u} - \bar{u}^*) + \lambda\mathbf{M}_y(\bar{y} - \bar{y}^*)] \end{aligned} \tag{4.3.15}$$

which is rearranged into

$$[\mathbf{I} - \lambda\mathbf{Z}_y\mathbf{M}_y](\bar{y} - \bar{y}^*) = \mathbf{Z}_y\mathbf{LB}(\mathbf{u} - \mathbf{u}^*)$$

Exists  $r_1 > 0$  such that when

$$0 \leq |\lambda| \leq r_1 \quad (4.3.16)$$

the LHS matrix  $[\mathbf{I} - \lambda \mathbf{Z}_y \mathbf{M}_y]$  is invertible and thus

$$\bar{y} - \bar{y}^* = [\mathbf{I} - \lambda \mathbf{Z}_y \mathbf{M}_y]^{-1} \mathbf{Z}_y \mathbf{L} \mathbf{B} (\mathbf{u} - \mathbf{u}^*) \quad (4.3.17)$$

When (4.3.16) holds, exist  $C_1 > 0$  such that

$$\|\bar{y} - \bar{y}^*\| \leq \|\mathbf{I} - \lambda \mathbf{Z}_y \mathbf{M}_y\|^{-1} \|\mathbf{Z}_y \mathbf{L} \mathbf{B}\| \|\bar{u} - \bar{u}^*\| \leq C_1 \|\bar{u} - \bar{u}^*\| \quad (4.3.18)$$

Similarly, let  $[\mathbf{Z}_p]_{(N-1)n_y \times Kn_y}$  be the matrix form of the linear operator that only keep adjoint shooting variables given input vectors of adjoints from all time steps,

$$\begin{aligned} \bar{p} - \bar{p}^* &= \mathbf{Z}_p [\mathbf{p} - \mathbf{p}^*] \\ &= \mathbf{Z}_p [\mathbf{L}^T \mathbf{Q} (\mathbf{y} - \mathbf{y}^*) - \mathbf{M}_p \hat{p}] \\ &= \mathbf{Z}_p [\mathbf{L}^T \mathbf{Q} (\mathbf{y} - \tilde{\mathbf{y}}) + \mathbf{L}^T \mathbf{Q} (\tilde{\mathbf{y}} - \mathbf{y}^*) + \lambda \mathbf{M}_p (\bar{p} - \bar{p}^*)] \end{aligned} \quad (4.3.19)$$

and

$$\tilde{\mathbf{y}} - \mathbf{y}^* = \mathbf{L} \mathbf{B} (\bar{u} - \bar{u}^*) \quad (4.3.20)$$

Substitute terms in (4.3.19) using (4.3.20) and (4.3.14),

$$[\mathbf{I} - \lambda \mathbf{Z}_p \mathbf{M}_p] (\bar{p} - \bar{p}^*) = \lambda \mathbf{Z}_p \mathbf{L}^T \mathbf{Q} \mathbf{M}_y (\bar{y} - \bar{y}^*) + \mathbf{Z}_p \mathbf{L}^T \mathbf{Q} \mathbf{L} \mathbf{B} (\bar{u} - \bar{u}^*) \quad (4.3.21)$$

Exists  $r_2 > 0$  such that when

$$0 \leq |\lambda| \leq r_2 \quad (4.3.22)$$

the LHS matrix  $[\mathbf{I} - \lambda \mathbf{Z}_p \mathbf{M}_p]$  is invertible and thus

$$\bar{p} - \bar{p}^* = [\mathbf{I} - \lambda \mathbf{Z}_p \mathbf{M}_p]^{-1} [\lambda \mathbf{Z}_p \mathbf{L}^T \mathbf{Q} \mathbf{M}_y (\bar{y} - \bar{y}^*) + \mathbf{Z}_p \mathbf{L}^T \mathbf{Q} \mathbf{L} \mathbf{B} (\bar{u} - \bar{u}^*)] \quad (4.3.23)$$

When (4.3.16) and (4.3.22) holds, using (4.3.18), exist  $C_2, C_3 > 0$  such that

$$\begin{aligned} \|\bar{p} - \bar{p}^*\| &\leq |\lambda| \|\mathbf{I} - \lambda \mathbf{Z}_p \mathbf{M}_p\|^{-1} \|\mathbf{Z}_p \mathbf{L}^T \mathbf{Q} \mathbf{M}_y\| \|\bar{y} - \bar{y}^*\| \\ &\quad + \|\mathbf{I} - \lambda \mathbf{Z}_p \mathbf{M}_p\|^{-1} \|\mathbf{Z}_p \mathbf{L}^T \mathbf{Q} \mathbf{L} \mathbf{B}\| \|\bar{u} - \bar{u}^*\| \\ &\leq C_2 \|\bar{y} - \bar{y}^*\| + C_3 \|\bar{u} - \bar{u}^*\| \\ &\leq (C_1 C_2 + C_3) \|\bar{u} - \bar{u}^*\| \end{aligned} \quad (4.3.24)$$

By (4.3.14) and (4.3.18),

$$\begin{aligned}
\|\tilde{\mathbf{y}} - \mathbf{y}\| &= \| -\mathbf{M}_y \lambda (\bar{y} - \bar{y}^*) \| \\
&\leq |\lambda| \|\mathbf{M}_y\| \|\bar{y} - \bar{y}^*\| \\
&\leq C_1 |\lambda| \|\mathbf{M}_y\| \|\bar{u} - \bar{u}^*\|
\end{aligned} \tag{4.3.25}$$

Define  $C_4 \stackrel{\text{def}}{=} C_1 \|\mathbf{M}_y\|$  and thus

$$\|\tilde{\mathbf{y}} - \mathbf{y}\| \leq C_4 |\lambda| \|\bar{u} - \bar{u}^*\| \tag{4.3.26}$$

Using (4.3.12), (4.3.13), and the two bounds obtained above, (4.3.24) and (4.3.26),

$$\begin{aligned}
\|\nabla \hat{J}(\bar{u}) - g\| &\leq \|\mathbf{B}\| \|\tilde{\mathbf{p}} - \mathbf{p}\| \\
&\leq \|\mathbf{B}\| [\|\mathbf{L}^T \mathbf{Q}\| \|\tilde{\mathbf{y}} - \mathbf{y}\| + \lambda \|\mathbf{M}_p\| \|\bar{p} - \bar{p}^*\|] \\
&\leq \|\mathbf{B}\| [C_1 |\lambda| \|\mathbf{L}^T \mathbf{Q}\| \|\mathbf{M}_y\| \|\bar{u} - \bar{u}^*\| + (C_1 C_2 + C_3) |\lambda| \|\mathbf{M}_p\| \|\bar{u} - \bar{u}^*\|] \\
&= [\|\mathbf{B}\| (C_1 \|\mathbf{L}^T \mathbf{Q}\| \|\mathbf{M}_y\| + (C_1 C_2 + C_3) \|\mathbf{M}_p\|)] |\lambda| \|\bar{u} - \bar{u}^*\|
\end{aligned} \tag{4.3.27}$$

Let

$$C \stackrel{\text{def}}{=} \frac{\|\mathbf{B}\| (C_1 \|\mathbf{L}^T \mathbf{Q}\| \|\mathbf{M}_y\| + (C_1 C_2 + C_3) \|\mathbf{M}_p\|)}{\sigma_{\min}(\mathbf{H})}$$

and

$$r = \min(r_1, r_2)$$

Then, when

$$|\lambda| \leq r$$

by (4.3.11),

$$\begin{aligned}
\|\nabla \hat{J}(\bar{u}) - g\| &\leq C |\lambda| \sigma_{\min}(\mathbf{H}) \|\bar{u} - \bar{u}^*\| \\
&\leq C |\lambda| \|\nabla \hat{J}(\bar{u})\|
\end{aligned}$$

which concludes the proof  $\square$

The result (4.3.10) of Lemma 4.3.1 is intuitive in the sense that, when  $\lambda$  is small, the state/adjoint jumps  $\hat{y}, \hat{p}$  are relatively small compared to the shooting variables. Making use of Lemma 4.3.1, I can characterize the location of eigenvalues of  $\mathbf{W}(\alpha)$  in the next lemma.

**Lemma 4.3.2** *For any  $k > 0$ , exists  $\bar{\alpha} > 0$  such that for all  $0 < \alpha < \bar{\alpha}$ , any eigenvalue  $\lambda$  of  $\mathbf{W}(\alpha)$  with  $|\lambda| < 0.5$  is in the cone*

$$\hat{C}_k \stackrel{\text{def}}{=} \{z \in \mathbb{C} : \text{Re}(z) > 0 \text{ and } |\text{Im}(z)|/\text{Re}(z) < k\} \quad (4.3.28)$$

**Proof:** Let  $\lambda$  be an eigenvalue of  $\mathbf{W}(\alpha)$ , with

$$|\lambda| < 0.5 \quad (4.3.29)$$

There are corresponding controls  $\bar{u}$ , shooting variables  $\bar{y}, \bar{p}$ , state/adjoint jumps  $\hat{y}, \hat{p}$ , and gradient-type vector  $g$  such that

$$\begin{bmatrix} -\hat{y} \\ \alpha g \\ -\hat{p} \end{bmatrix} = \mathbf{W}(\alpha) \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} = \lambda \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} \quad (4.3.30)$$

Then,

$$\begin{aligned} \lambda(\bar{u} - \bar{u}^*) &= \alpha g \\ \mathbf{H}(\bar{u} - \bar{u}^*) &= \nabla \hat{J}(\bar{u}) \end{aligned}$$

where  $\mathbf{H}$ , defined in (3.1.8), is the Hessian of the reduced control space objective function  $\hat{J}(\bar{u})$  (control as the only argument, no multiple shooting). For  $\alpha > 0$ ,

$$\left(\mathbf{H} - \frac{\lambda}{\alpha}\right)(\bar{u} - \bar{u}^*) = \nabla \hat{J}(\bar{u}) - g$$

Let  $r$  and  $C$  be given by Lemma 4.3.1, when

$$|\lambda| < r \quad (4.3.31)$$

The following holds, where  $\sigma_{\min}(\cdot)$  stands for the smallest singular value of a matrix,

$$\begin{aligned} \sigma_{\min}\left(\mathbf{H} - \frac{\lambda}{\alpha}\right)\|\bar{u} - \bar{u}^*\| &\leq \left\|\left(\mathbf{H} - \frac{\lambda}{\alpha}\right)(\bar{u} - \bar{u}^*)\right\| = \|\nabla \hat{J}(\bar{u}) - g\| \\ &\leq |\lambda|C\|\nabla \hat{J}(\bar{u})\| = |\lambda|C\|\mathbf{H}(u - u^*)\| \leq |\lambda|C\|\mathbf{H}\|\|u - u^*\| \end{aligned} \quad (4.3.32)$$

By the fact discussed earlier,

- For any  $\alpha \neq 0$ , any eigenvalue of  $\mathbf{W}(\alpha)$  is non-zero,

$$\lambda \neq 0 \quad (4.3.33)$$

- For  $0 < |\lambda| < 1$ , the control part in the eigenvector is not the optimal control,

$$\|\bar{u} - \bar{u}^*\| \neq 0 \quad (4.3.34)$$

Then, using (4.3.34), the inequality (4.3.32) can be reduced to

$$\sigma_{\min}(\mathbf{H} - \frac{\lambda}{\alpha}) \leq |\lambda|C\|\mathbf{H}\| \quad (4.3.35)$$

Note that

$$\sqrt{(\sigma_{\min}(\mathbf{H}) - \frac{\operatorname{Re} \lambda}{\alpha})^2 + (\frac{\operatorname{Im} \lambda}{\alpha})^2} = |\sigma_{\min}(\mathbf{H}) - \frac{\lambda}{\alpha}| \leq \sigma_{\min}(\mathbf{H} - \frac{\lambda}{\alpha}) \quad (4.3.36)$$

and therefore, by (4.3.35) and (4.3.36),

$$\begin{cases} |\sigma_{\min}(\mathbf{H}) - \frac{\operatorname{Re} \lambda}{\alpha}| & \leq |\lambda|C\|\mathbf{H}\| \\ |\frac{\operatorname{Im} \lambda}{\alpha}| & \leq |\lambda|C\|\mathbf{H}\| \end{cases}$$

Using (4.3.6),

$$\begin{cases} \lim_{\alpha \rightarrow 0} \max \{ |\sigma_{\min}(\mathbf{H}) - \frac{\operatorname{Re} \lambda}{\alpha}| : \exists x \text{ s.t. } \mathbf{W}(\alpha)x = \lambda x \text{ and } |\lambda| < 0.5 \} & = 0 \\ \lim_{\alpha \rightarrow 0} \max \{ |\frac{\operatorname{Im} \lambda}{\alpha}| : \exists x \text{ s.t. } \mathbf{W}(\alpha)x = \lambda x \text{ and } |\lambda| < 0.5 \} & = 0 \end{cases}$$

and therefore

$$\begin{aligned} & \lim_{\alpha \rightarrow 0} \max \left\{ \left| \frac{\operatorname{Im} \lambda}{\operatorname{Re} \lambda} \right| : \exists x \text{ s.t. } \mathbf{W}(\alpha)x = \lambda x \text{ and } |\lambda| < 0.5 \right\} \\ & \leq \lim_{\alpha \rightarrow 0} \frac{\max \left\{ \left| \frac{\operatorname{Im} \lambda}{\alpha} \right| : \dots \right\}}{\min \left\{ \left| \frac{\operatorname{Re} \lambda}{\alpha} \right| : \dots \right\}} \leq \lim_{\alpha \rightarrow 0} \frac{\max \left\{ \left| \frac{\operatorname{Im} \lambda}{\alpha} \right| : \dots \right\}}{\min \left\{ \sigma_{\min}(\mathbf{H}) - \left| \sigma_{\min}(\mathbf{H}) - \frac{\operatorname{Re} \lambda}{\alpha} \right| : \dots \right\}} \\ & = \lim_{\alpha \rightarrow 0} \frac{\max \left\{ \left| \frac{\operatorname{Im} \lambda}{\alpha} \right| : \dots \right\}}{\sigma_{\min}(\mathbf{H}) - \max \left\{ \left| \sigma_{\min}(\mathbf{H}) - \frac{\operatorname{Re} \lambda}{\alpha} \right| : \dots \right\}} \\ & = 0 \end{aligned}$$

which concludes the proof.  $\square$

Using Lemma 4.3.2, the convergence of iteration (4.2.15) can be readily proven using the argument that with sufficiently small  $\alpha > 0$ , the spectral radius of  $[I - \mathbf{W}(\alpha)]$  in (4.3.3) is less than 1.

**Theorem 4.3.3** *With sufficiently small  $\alpha > 0$ ,*

$$\rho(I - \mathbf{W}(\alpha)) < 1$$

**Proof:** For small  $\alpha$ , there are eigenvalues around 0 and 1. For those around one, in this proof, I will make use of the cone defined in (4.3.28) of Lemma 4.3.2 to show they are inside the unit circle. So, all eigenvalues are in the unit circle. Details follow.

Since  $\mathbf{I} - \mathbf{W}(\alpha)$  only has eigenvalue 0 and 1, by the continuity of eigenvalues with respect to matrix coefficients, exist  $\bar{\alpha}_1 > 0$  such that with

$$\alpha \leq \bar{\alpha}_1 \tag{4.3.37}$$

All eigenvalues are around 0 and 1, i.e.,

$$\{\lambda \in \mathbb{C} : \exists x \text{ s.t. } (\mathbf{I} - \mathbf{W}(\alpha))x = \lambda x\} \subset B(0, 0.5) \cup B(1, 0.5) \tag{4.3.38}$$

To use Lemma 4.3.2, let  $k = 1$  and by Lemma 4.3.2, exists  $\bar{\alpha}_2 > 0$  such that with

$$0 < \alpha \leq \bar{\alpha}_2 \tag{4.3.39}$$

the eigenvalues of  $\mathbf{W}(\alpha)$  around 0 is in a cone, i.e.,

$$\{\lambda \in \mathbb{C} : \exists x \text{ s.t. } \mathbf{W}(\alpha)x = \lambda x \text{ and } |\lambda| < 0.5\} \subset \hat{C}_1 \tag{4.3.40}$$

with  $\hat{C}_1$  defined in (4.3.28). Looking at  $\mathbf{I} - \mathbf{W}(\alpha)$  instead of  $\mathbf{W}(\alpha)$ , (4.3.40) is equivalent to

$$\begin{aligned} & \{\lambda \in \mathbb{C} : \exists x \text{ s.t. } (\mathbf{I} - \mathbf{W}(\alpha))x = \lambda x \text{ and } |\lambda| > 0.5\} \\ & \subset \{z \in \mathbb{C} : \operatorname{Re}(z) < 1 \text{ and } |\operatorname{Im}(z)|/(1 - \operatorname{Re}(z)) < 1\} \end{aligned} \tag{4.3.41}$$

By (4.3.38),

$$\begin{aligned} & \{\lambda \in \mathbb{C} : \exists x \text{ s.t. } (\mathbf{I} - \mathbf{W}(\alpha))x = \lambda x\} \\ & = \{\lambda \in \mathbb{C} : \exists x \text{ s.t. } (\mathbf{I} - \mathbf{W}(\alpha))x = \lambda x \text{ and } \lambda \in B(0, 0.5)\} \\ & \cup \{\lambda \in \mathbb{C} : \exists x \text{ s.t. } (\mathbf{I} - \mathbf{W}(\alpha))x = \lambda x \text{ and } \lambda \in B(1, 0.5)\} \end{aligned} \tag{4.3.42}$$

By (4.3.41), the second set in the union in (4.3.42),

$$\begin{aligned}
& \{\lambda \in \mathbb{C} : \exists x \text{ s.t. } (\mathbf{I} - \mathbf{W}(\alpha))x = \lambda x \text{ and } \lambda \in B(1, 0.5)\} \\
&= \{\lambda \in \mathbb{C} : \exists x \text{ s.t. } (\mathbf{I} - \mathbf{W}(\alpha))x = \lambda x \text{ and } \lambda \in B(1, 0.5)\} \\
&\cap \{z \in \mathbb{C} : \operatorname{Re}(z) < 1 \text{ and } |\operatorname{Im}(z)|/(1 - \operatorname{Re}(z)) < 1\} \\
&\subset B(0, 1)
\end{aligned} \tag{4.3.43}$$

Then both sets in the RHS of (4.3.42) are subsets of  $B(0, 1)$  which concludes the proof that

$$\{\lambda \in \mathbb{C} : \exists x \text{ s.t. } (\mathbf{I} - \mathbf{W}(\alpha))x = \lambda x\} \subset B(0, 1)$$

given  $\alpha$  satisfies (4.3.37) and (4.3.39). □

So far, the alternative convergence proof of the parallel-in-time gradient-type method applied to linear-quadratic problem (3.1.1) is finished.

**How to choose step size.** In this convergence proof, similar to Section 3.5, I only showed the guaranteed convergence given sufficiently small step size. However, the proof does not shed light on how one should run an parallel-in-time gradient-type method based algorithm to guarantee convergence, i.e., how to choose step size, when to discard recent steps and return to a previous step, etc. I present the following discussion related to the desired ability of determining, according to quantities observed in the parallel-in-time gradient-type iteration, a step size that leads to convergence. However, the discussion does not lead to a way of choosing step size.

The iterates in the iteration (4.3.3), namely,

$$\begin{bmatrix} \bar{y}^{(j+1)} - \bar{y}^* \\ \bar{u}^{(j+1)} - \bar{u}^* \\ \bar{p}^{(j+1)} - \bar{p}^* \end{bmatrix} = [\mathbf{I} - \mathbf{W}(\alpha)] \begin{bmatrix} \bar{y}^{(j)} - \bar{y}^* \\ \bar{u}^{(j)} - \bar{u}^* \\ \bar{p}^{(j)} - \bar{p}^* \end{bmatrix} = [\mathbf{I} - \mathbf{R}(\alpha)\mathbf{H}_{\mathcal{L}}] \begin{bmatrix} \bar{y}^{(j)} - \bar{y}^* \\ \bar{u}^{(j)} - \bar{u}^* \\ \bar{p}^{(j)} - \bar{p}^* \end{bmatrix}.$$

is not directly observable since the optimal solution  $\bar{y}^*, \bar{u}^*, \bar{p}^*$  is unknown. I rewrite the iteration into another form where the iterates are explicitly known. Multiply  $\mathbf{H}_{\mathcal{L}}$



on both sides of (4.3.3) yields

$$\mathbf{H}_{\mathcal{L}} \begin{bmatrix} \bar{y}^{(j+1)} - \bar{y}^* \\ \bar{u}^{(j+1)} - \bar{u}^* \\ \bar{p}^{(j+1)} - \bar{p}^* \end{bmatrix} = [\mathbf{I} - \mathbf{H}_{\mathcal{L}}\mathbf{R}(\alpha)]\mathbf{H}_{\mathcal{L}} \begin{bmatrix} \bar{y}^{(j)} - \bar{y}^* \\ \bar{u}^{(j)} - \bar{u}^* \\ \bar{p}^{(j)} - \bar{p}^* \end{bmatrix}. \quad (4.3.44)$$

Since both  $\mathbf{H}_{\mathcal{L}}$  and  $\mathbf{R}(\alpha)$  are symmetric,

$$[\mathbf{I} - \mathbf{H}_{\mathcal{L}}\mathbf{R}(\alpha)] = [\mathbf{I} - \mathbf{R}(\alpha)\mathbf{H}_{\mathcal{L}}]^T = [\mathbf{I} - \mathbf{W}(\alpha)]^T.$$

By (4.3.2) and (4.2.12),

$$\mathbf{H}_{\mathcal{L}} \begin{bmatrix} \bar{y} - \bar{y}^* \\ \bar{u} - \bar{u}^* \\ \bar{p} - \bar{p}^* \end{bmatrix} = \begin{bmatrix} \hat{p}(\bar{y}, \bar{u}, \bar{p}) \\ g(\bar{y}, \bar{u}, \bar{p}) \\ \hat{y}(\bar{y}, \bar{u}, \bar{p}) \end{bmatrix} \quad (4.3.45)$$

where  $\hat{y}, g, \hat{p}$  are defined in (4.2.10) and (4.2.11). Then, the implicit iteration is derived on the state/adjoint jumps and the gradient-type vector, which are readily observable after each iteration,

$$\begin{bmatrix} \hat{y}^{(j+1)} \\ g^{(j+1)} \\ \hat{p}^{(j+1)} \end{bmatrix} = [\mathbf{I} - \mathbf{W}(\alpha)]^T \begin{bmatrix} \hat{y}^{(j)} \\ g^{(j)} \\ \hat{p}^{(j)} \end{bmatrix}. \quad (4.3.46)$$

The newly derived (4.3.46) is closely related to (4.3.3). Note that

$$\rho([\mathbf{I} - \mathbf{W}(\alpha)]^T) = \rho([\mathbf{I} - \mathbf{W}(\alpha)])$$

and by (4.3.45) and  $\mathbf{H}_{\mathcal{L}}$  being non-singular,

$$\begin{bmatrix} \hat{y}^{(j)} \\ g^{(j)} \\ \hat{p}^{(j)} \end{bmatrix} = 0 \Leftrightarrow \begin{bmatrix} \bar{y}^{(j)} - \bar{y}^* \\ \bar{u}^{(j)} - \bar{u}^* \\ \bar{p}^{(j)} - \bar{p}^* \end{bmatrix} = 0$$

The iterates in iteration (4.3.46) is observable, however, one can not adjust step size directly according to the change in norm of the iterates. It is worth noting that it is proved above that with small  $\alpha$ , the spectral radius  $\rho(\mathbf{I} - \mathbf{W}(\alpha)) < 1$ , however,

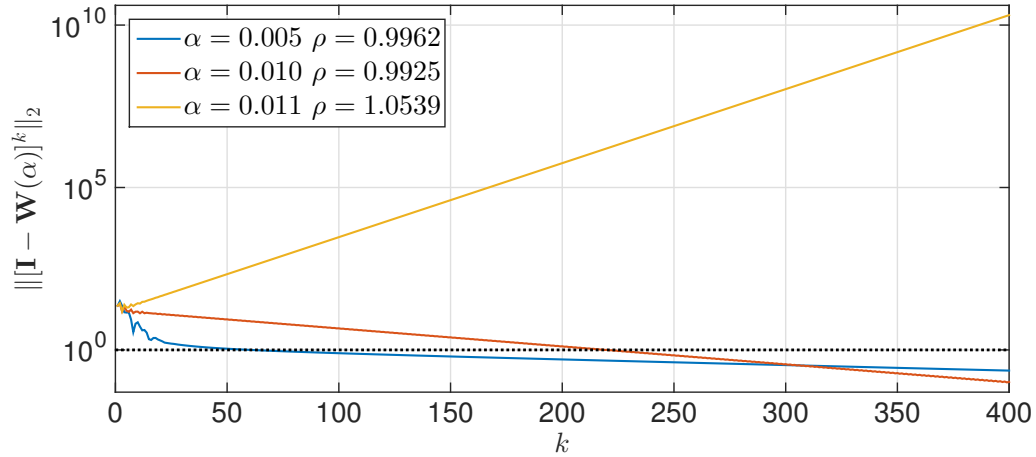


Figure 4.2: Plot of  $\|(\mathbf{I} - \mathbf{W}(\alpha))^k\|_2$  against  $k$ . In the legend,  $\rho(\mathbf{I} - \mathbf{W}(\alpha))$  is also appended.

the 2-norm of the asymmetric iteration matrix  $\|\mathbf{I} - \mathbf{W}(\alpha)\|_2$  is often greater than 1 in the case number of subdomains  $N > 1$ . In the some cases, it might requires a large power  $k$  so that

$$\|(\mathbf{I} - \mathbf{W}(\alpha))^k\|_2 < 1$$

Experiment shows that it is not necessary that

$$\min \{k \in \mathbb{Z}^+ : \|(\mathbf{I} - \mathbf{W}(\alpha))^k\|_2 < 1\}$$

can be bounded by a function of number of time subdomains.

I demonstrate this by an example problem (A.3.2) with  $K = 200$ ,  $\rho = 0.9$ ,  $r = 0.1$  and  $N = 10$  time subdomains. Note that below, the notation  $\rho$  refers to  $\rho(\mathbf{I} - \mathbf{W}(\alpha))$  rather than the problem parameter above. Step size  $\alpha = 0.01$  approximately minimizes  $\rho(\mathbf{I} - \mathbf{W}(\alpha))$  which leads to  $\rho(\mathbf{I} - \mathbf{W}(\alpha)) = 0.9925$ . In Figure 4.2, for three different step sizes, I plot  $\|(\mathbf{I} - \mathbf{W}(\alpha))^k\|_2$  against  $k$ . In the case when  $\rho(\mathbf{I} - \mathbf{W}(\alpha)) < 1$ , it requires a big power  $k$  so that  $\|(\mathbf{I} - \mathbf{W}(\alpha))^k\|_2 < 1$ .

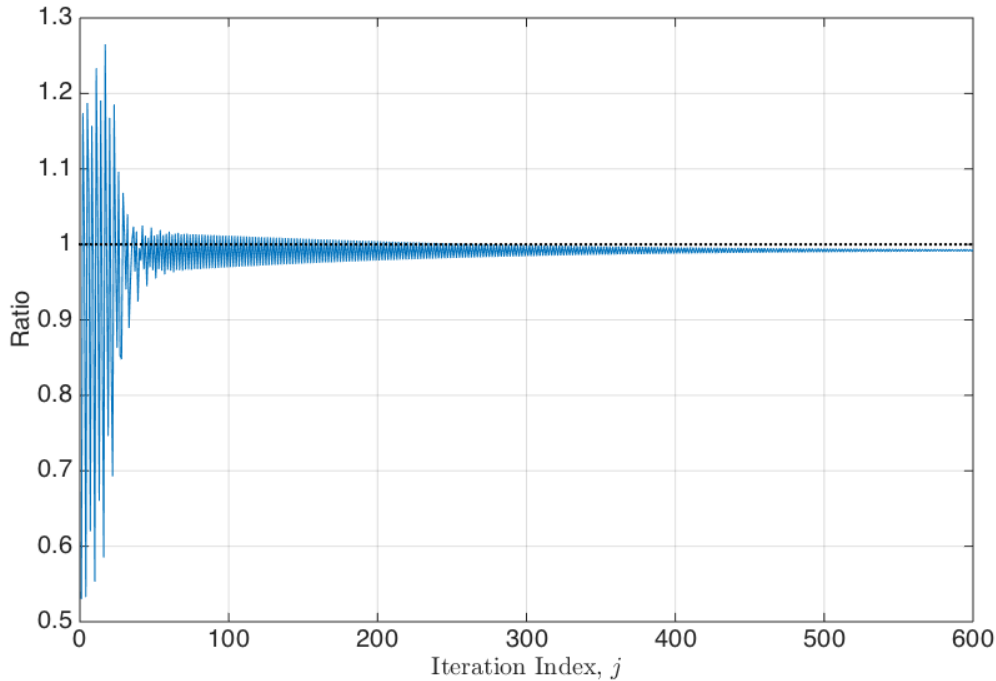


Figure 4.3: Ratio  $\|((\hat{y}^{(j+1)})^T, (g^{(j+1)})^T, (\hat{p}^{(j+1)})^T)\| / \|((\hat{y}^{(j)})^T, (g^{(j)})^T, (\hat{p}^{(j)})^T)\|$  against iteration index  $j$ . Step size  $\alpha = 0.01$ . Random initial  $\hat{y}^{(0)}, g^{(0)}, \hat{p}^{(0)}$  is used.

It is also natural to check the ratio between norms of consecutive iterates, i.e.,

$$\left\| \begin{bmatrix} \hat{y}^{(j+1)} \\ g^{(j+1)} \\ \hat{p}^{(j+1)} \end{bmatrix} \right\| / \left\| \begin{bmatrix} \hat{y}^{(j)} \\ g^{(j)} \\ \hat{p}^{(j)} \end{bmatrix} \right\| \quad (4.3.47)$$

in (4.3.46) in Figure 4.3. In this case with random initial  $\hat{y}^{(0)}, g^{(0)}, \hat{p}^{(0)}$ , it takes many iterations before this ratio is stably below 1.

In conclusion, currently, for linear-quadratic problem, although it is proved that for sufficiently small  $\alpha > 0$ , it is true that  $\rho(\mathbf{I} - \mathbf{W}(\alpha)) < 1$ , I do not have an efficient method to determine if a step size  $\alpha$  leads to  $\rho(\mathbf{I} - \mathbf{W}(\alpha)) < 1$ . One possible direction is to investigate the conjecture that, if the iteration is initialized by a full gradient sweep, i.e.,  $2N - 1$  steps of  $\alpha = 0$  iteration, then a sufficiently small fixed step size

$\alpha > 0$  leads to the monotonic decreasing of the norm of the iterates in (4.3.46) and also the convergence to the optimal solution.

## 4.4 Summary

In this chapter, I revealed the connection between the parallel-in-time gradient-type method and the direct multiple shooting reformulation of the optimization problem. I showed that the gradient of the multiple shooting formulation Lagrangian contains three components that are key quantities in the parallel-in-time gradient-type method:

- the state variable jumps at the subdomain boundaries,
- the adjoint variable jumps at the subdomain boundaries,
- and the gradient-type vector in the parallel-in-time gradient-type method.

By this fact, the parallel-in-time gradient-type method is interpreted as applying rotated and scaled gradient updates to solve the multiple shooting formulation optimality system. For linear-quadratic problems, This optimality system is symmetric positive indefinite and therefore solving this optimality system is to solve a saddle point problem. I proved that the gradient-type update results from the parallel-in-time gradient-type method, with sufficiently small fixed step size, is guaranteed to converge to the optimal solution, i.e., to solve the saddle point problem. The proof is using spectral radius argument of another iteration matrix different from the implicitly constructed iteration matrix used in the convergence proof in Section 3.5.

However, although the parallel-in-time gradient-type method is guaranteed to converge with sufficiently small step sizes, there is not a good way to determine how small the step size should be.

# Chapter 5

## Parallel-In-Time Gradient-Type Method in Nonlinear Problems

In this chapter, I prove the convergence of parallel-in-time gradient-type method applied to nonlinear optimization problems (as opposed to the linear-quadratic problem (3.1.1) discussed in previous chapters) with sufficient small step size. In Section 3.5 and Section 4.3, convergence proofs are given for linear-quadratic problems which are based on arguments using spectral radius of iteration matrices that arise from difference perspectives. The proofs I will provide in this chapter for general non-linear problems rely less on linear algebra and the step sizes are not required to be fixed.

The convergence proof was written with the idea in mind that with sufficiently small step size, the parallel-in-time gradient-type method should behave similarly to the gradient method since with small step sizes, the gradient-type vector produced by the parallel algorithm is similar to the true gradient and thus the control update is similar to that of the gradient method. I point out that the sufficiently small step size, which guarantees convergence by these following Lemmas and Theorems in this chapter, is not particularly tight. Proofs are derived in the pursuit of “convergence with sufficiently small step size” and not much effort is devoted to obtaining a tight upper bound of the “sufficiently small” step size. If I trace the proofs and find the step

size upper bound, it might be too small for optimal convergence speed in practice. A convergence theorem and proof that shares the same flavor for a related topic, distributed asynchronous gradient optimization algorithms, was given in [TBA86].

Most proofs in this chapter are based on Lemma 5.2.1. I prove Lemma 5.2.1 for the original parallel-in-time gradient-type method (same partition for forward/back computation, no subdomain overlapping) written in the pseudo code form in Algorithm 13. However, although not proved in this thesis a similar result is most likely also true for the generalized method proposed in Section 3.4.

Since in some applications, as in the oil reservoir optimization problem discussed in Chapter 6, the parallel-in-time gradient-type algorithm will be used in combination with metric projection in constrained problems where the control variables are restricted in a closed convex set, in this chapter, majority of the proofs are for the parallel-in-time gradient-type method combined with metric projection. These are more general results than convergence theorems for the unconstrained problem, because the unconstrained problem can be seen as special constrained problem where control variables are constrained in the whole space.

In the proofs, properties of projection onto closed convex set are extensively used. In Section 5.1, I state with proofs a few standard results regarding metric projection and some other useful projection properties, which are less seen in text books but mostly interesting and intuitive, that will be used in this chapter.

Figure 5.1 illustrates the structure of this chapter by showing the dependence between theorems. The proofs are organized as follows.

First, Lemma 5.2.1 proves an intuitive error bound between the gradient-type vector and true gradient with the assumption that all generated control iterates stay in a predetermined region. Then, with the similar assumption, Theorem 5.2.3 and Proposition 5.2.7 proved convergence of the parallel method. This assumption is restrictive and hard to verify. So, Theorem 5.2.5 and Corollary 5.2.6 used strong convexity to replace the assumption mentioned above, so that a convergence theorem

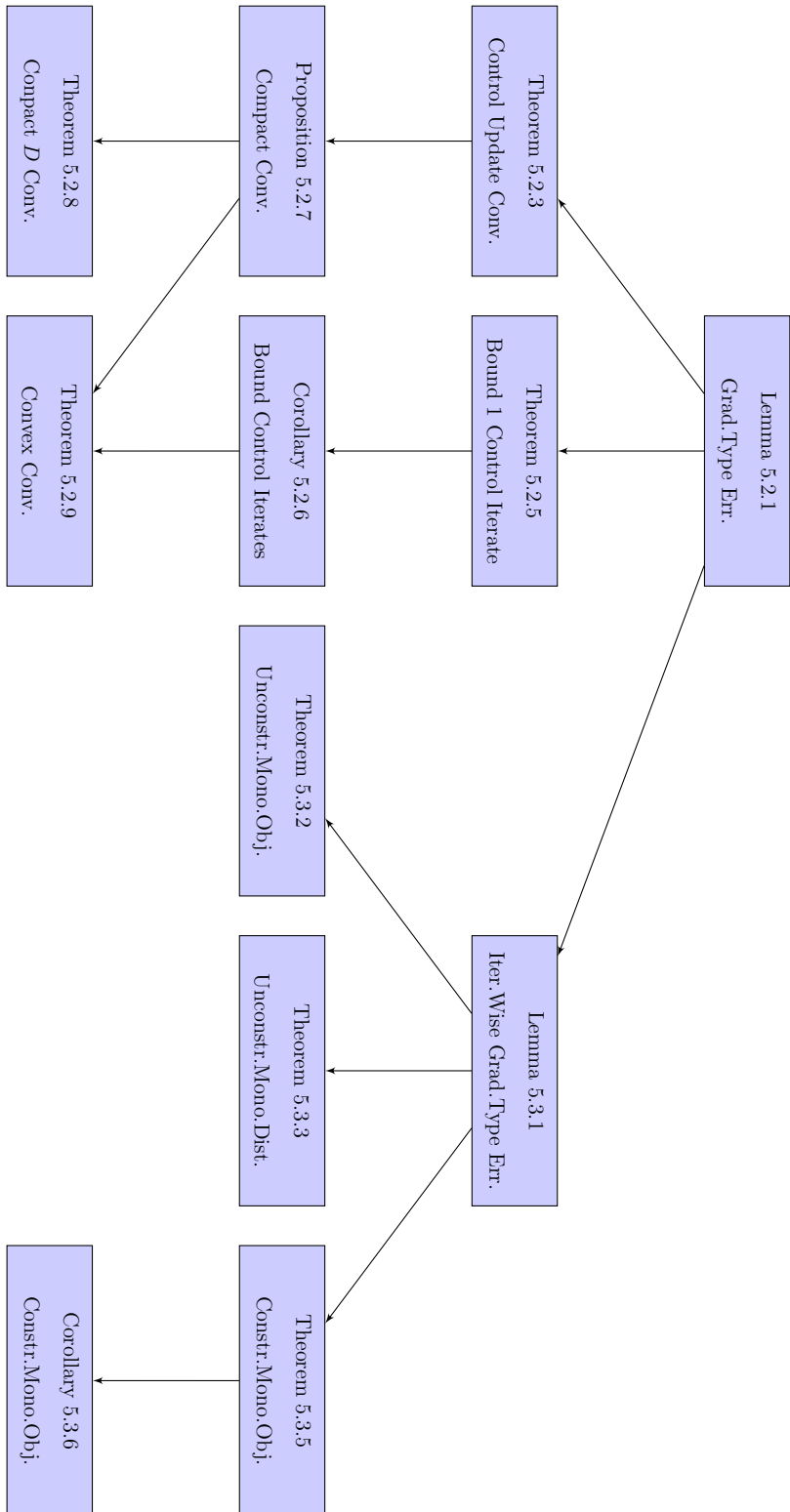


Figure 5.1: Theorem Dependency Structure in Chapter 5

for convex problem Theorem 5.2.9 is proved. Alternatively, Theorem 5.2.8, rather than adding the assumption of convexity, assumption that the projection set  $D$  is compact is used.

Another branch starting with Lemma 5.3.1 in Figure 5.1 aims for monotonic convergence results. Lemma 5.3.1 uses a mathematical induction type of proof to show, with small step sizes, an iteration-wise error bound between the gradient-type vector and true gradient can be given in contrast to the bound in Lemma 5.2.1 which involves gradient-type vectors of previous iterations. Based on this lemma, Theorem 5.3.2 and Theorem 5.3.3 develops proof of monotonic convergence in terms of objective function value and control error for unconstrained problem. For constrained problem, Theorem 5.3.5 and Corollary 5.3.6 gave similar but slightly weaker results.

## 5.1 Metric Projection Properties

In this section, I give some metric projection properties since the parallel-in-time gradient-type method convergence related proofs are given in the context of its being combined with metric projection. All lemmas are intuitive, some of which are better known in the standard texts and the rest of which I did not find proofs elsewhere and thus I provide proofs here. All vector norms used are 2-norm (or, equivalently, Frobenius norm) unless otherwise explained .

Given a closed convex set  $D \in \mathbb{R}^n$ , the metric projection onto  $D$  is defined as follows, for any  $x \in \mathbb{R}^n$ ,

$$\mathcal{P}_D(x) \stackrel{\text{def}}{=} \arg \min_{y \in D} \|x - y\|_2$$

This is well defined for  $D$  being closed and convex.

The following standard result Lemma 5.1.1 characterizes metric projection onto a closed convex set, on which most of the other lemmas rest.

**Lemma 5.1.1** *If  $D \in \mathbb{R}^n$  is a closed convex set and  $\mathcal{P}_D$  is the metric projection to*



$D$ ,  $\delta x \in \mathbb{R}^n$ , then

$$\mathcal{P}_D(x + \delta x) = x \Rightarrow (x - y)^T \delta x \geq 0, \forall y \in D$$

**Proof:** Since  $\mathcal{P}_D(x + \delta x) = x$ ,

$$x = \arg \min_{x' \in D} \|x' - (x + \delta x)\|. \quad (5.1.1)$$

For all  $s \in [0, 1]$ , due to the convexity of  $D$ ,  $x + s(y - x) \in D$ , by (5.1.1),

$$\|x - (x + \delta x)\| \leq \|x + s(y - x) - (x + \delta x)\|. \quad (5.1.2)$$

Squaring both sides of (5.1.2) and rearranging terms leads to

$$(x - y)^T \delta x \geq -\frac{s}{2} \|y - x\|^2$$

Since this inequality holds for all  $s \in [0, 1]$ ,  $(x - y)^T \delta x \geq 0$  is proved.  $\square$

The Lemma 5.1.2 states that 1 is a Lipschitz constant for the projection  $\mathcal{P}_D$ . In the proofs of this chapter, Lemma 5.1.2 is sometimes used without reference to it.

**Lemma 5.1.2** *If  $D \in \mathbb{R}^n$  is a closed convex set and  $\mathcal{P}_D$  is the metric projection to  $D$ , then*

$$\|\mathcal{P}_D(x) - \mathcal{P}_D(y)\| \leq \|x - y\| \quad \forall x, y \in \mathbb{R}^n$$

**Proof:** The lemma is true when  $\mathcal{P}_D(x) = \mathcal{P}_D(y)$ . Below, assume  $\mathcal{P}_D(x) \neq \mathcal{P}_D(y)$ .

Define unit vector  $h \stackrel{\text{def}}{=} [\mathcal{P}_D(y) - \mathcal{P}_D(x)] / \|\mathcal{P}_D(y) - \mathcal{P}_D(x)\|$ , then

$$y - x = [y - \mathcal{P}_D(y)] + [\mathcal{P}_D(y) - \mathcal{P}_D(x)] + [\mathcal{P}_D(x) - x]$$

The magnitude of  $y - x$  is greater than that of its projection onto the direction of  $h$ ,

$$\|y - x\| \geq \|hh^T [y - \mathcal{P}_D(y)] + [\mathcal{P}_D(y) - \mathcal{P}_D(x)] + [\mathcal{P}_D(x) - x]\|$$

By Lemma 5.1.1,  $h^T [y - \mathcal{P}_D(y)] \geq 0$  and  $h^T [\mathcal{P}_D(x) - x] \geq 0$ , therefore

$$\begin{aligned} \|y - x\| &\geq \|hh^T [\mathcal{P}_D(y) - \mathcal{P}_D(x)]\| \\ &= \|\mathcal{P}_D(y) - \mathcal{P}_D(x)\| \end{aligned}$$

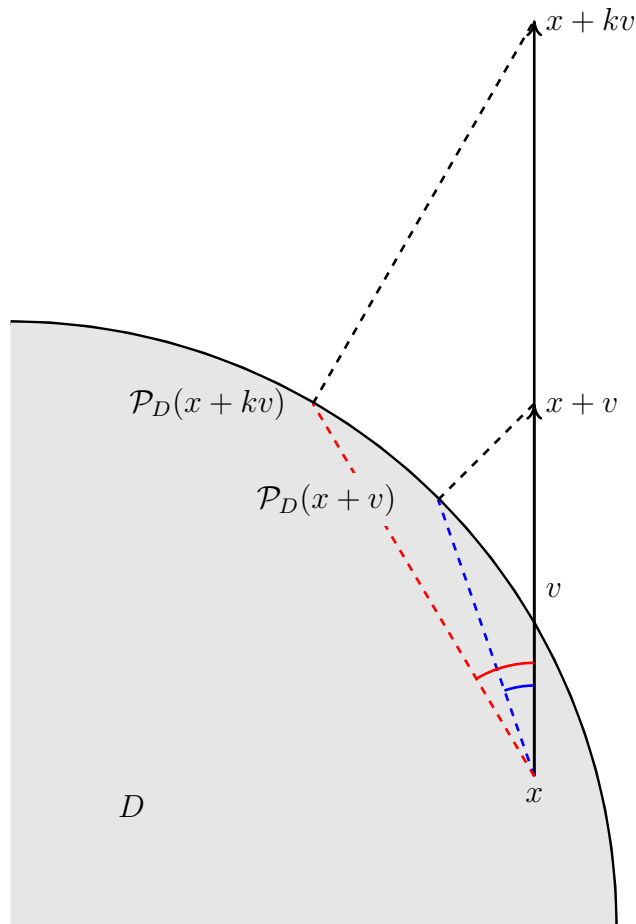


Figure 5.2: Illustration for Lemma 5.1.3, Lemma 5.1.4, and Lemma 5.1.5 on the relationship between the blue and red lengths and angles.

□

The following Lemma 5.1.3, Lemma 5.1.4, and Lemma 5.1.5 compare quantities arising from a pair of related projection, see Figure 5.2 for illustration. Lemma 5.1.3 states that a greater offset leads to a greater offset after projection.

**Lemma 5.1.3** *If  $D \subset \mathbb{R}^n$  is a closed convex set and  $\mathcal{P}_D$  is the metric projection to  $D$ ,  $x \in D$ ,  $v \in \mathbb{R}^n$ , and  $k \geq 1$  then*

$$\|\mathcal{P}_D(x + kv) - x\| \geq \|\mathcal{P}_D(x + v) - x\|$$

Notation	Definition
$a$	$\mathcal{P}_D(x + v) - x$
$b$	$x + v - \mathcal{P}_D(x + v)$
$c$	$x + kv - \mathcal{P}_D(x + kv)$
$d$	$\mathcal{P}_D(x + kv) - \mathcal{P}_D(x + v)$

Table 5.1: Notations for the Proof of Lemma 5.1.3, Lemma 5.1.4, and Lemma 5.1.5

**Proof:** If  $k = 1$ , the lemma is true. Below, assume  $k > 1$ . In this and several following proofs, slightly complex algebraic computation is involved, to simplify the equations I define notations in Table 5.1,

Decompose  $v$  and  $kv$ ,

$$v = a + b \tag{5.1.3a}$$

$$kv = a + c + d \tag{5.1.3b}$$

Multiply  $k$  on (5.1.3a) and combine the result with (5.1.3b),

$$(k - 1)a = -kb + c + d \tag{5.1.4}$$

By Lemma 5.1.1,

$$-kb^T d \geq 0$$

$$c^T d \geq 0$$

Then, (5.1.4) and the assumption that  $k > 1$  leads to

$$a^T d = \frac{-kb^T d + c^T d + \|d\|^2}{k - 1} \geq 0 \tag{5.1.5}$$

Hence,

$$\begin{aligned} \|\mathcal{P}_D(x + kv) - x\|^2 &= \|a + d\|^2 \\ &= \|a\|^2 + 2a^T d + \|d\|^2 \\ &\geq \|a\|^2 = \|\mathcal{P}_D(x + v) - x\|^2 \end{aligned}$$

which concludes the proof.  $\square$

The following Lemma 5.1.4 together with the above Lemma 5.1.3 forms a pair of upper and lower bound of  $\|\mathcal{P}_D(x + kv) - x\|$  in terms of  $\|\mathcal{P}_D(x + v) - x\|$ .

**Lemma 5.1.4** *If  $D \subset \mathbb{R}^n$  is a closed convex set and  $\mathcal{P}_D$  is the metric projection to  $D$ ,  $x \in D$ ,  $v \in \mathbb{R}^n$ , and scalar  $k \geq 1$  then*

$$k\|\mathcal{P}_D(x + v) - x\| \geq \|\mathcal{P}_D(x + kv) - x\|$$

**Proof:** If  $k = 1$ , the lemma is true. Below, assume  $k > 1$ . Use the set of notations defined in Table 5.1. To prove this lemma is to show

$$\|ka\| \geq \|a + d\| \tag{5.1.6}$$

Since

$$k(a + b) = a + c + d$$

by the assumption that  $k > 1$ ,

$$a = \frac{c + d - kb}{k - 1}$$

The inequality (5.1.6) is equivalent to

$$\left\|k \frac{c + d - kb}{k - 1}\right\|^2 \geq \left\|\frac{c + d - kb}{k - 1} + d\right\|^2$$

which, by rearranging terms, is equivalent to

$$k\|c\|^2 + k\|kb\|^2 + \|c - kb\|^2 + 2kc^T d - 2k^2 b^T d \geq 0 \tag{5.1.7}$$

By Lemma 5.1.1,  $c^d \geq 0$  and  $b^T d \leq 0$ , and thus (5.1.7) holds, and subsequently (5.1.6) holds, which concludes the proof.  $\square$

Lemma 5.1.5 is related to the intuition that the more offset in the update direction of  $v$ , the more the projected update deviates from the direction  $v$ .

**Lemma 5.1.5** *If  $D \subset \mathbb{R}^n$  is a closed convex set and  $\mathcal{P}_D$  is the metric projection to  $D$ ,  $x \in D$ ,  $v \in \mathbb{R}^n$ , and  $k \geq 1$  assume*

$$\mathcal{P}_D(x + v) \neq x \quad (\text{by Lemma 5.1.6, equivalently, } \mathcal{P}_D(x + kv) \neq x)$$

then

$$v^T \frac{(\mathcal{P}_D(x + kv) - x)}{\|\mathcal{P}_D(x + kv) - x\|} \leq v^T \frac{(\mathcal{P}_D(x + v) - x)}{\|\mathcal{P}_D(x + v) - x\|}$$

**Proof:** Use the set of notations defined in Table 5.1. The assumption  $\mathcal{P}_D(x + v) \neq x$  implies that  $\|a + b\| \neq 0$  and  $\|a\| \neq 0$ . To prove this lemma is to show

$$\frac{(a + b)^T(a + d)}{\|a + d\|} \leq \frac{(a + b)^T a}{\|a\|} \quad (5.1.8)$$

which, by rearranging terms, is equivalent to

$$\|a\|^2(a^T d + \|d\|^2 - b^T d) + a^T b(2a^T d + \|d\|^2) \geq 0 \quad (5.1.9)$$

By (5.1.5) in Lemma 5.1.3,  $a^T d \geq 0$ . By Lemma 5.1.1,  $a^T b \geq 0$  and  $b^T d \leq 0$ . So, (5.1.9) and (5.1.8) are valid, which concludes the proof.  $\square$

Lemma 5.1.6 states that the property of projected update of a point being itself is a property of the update direction independent of the scaling of the update.

**Lemma 5.1.6** *If  $D \subset \mathbb{R}^n$  is a closed convex set and  $\mathcal{P}_D$  is the metric projection to  $D$ ,  $x \in D$ ,  $v \in \mathbb{R}^n$ , and  $\alpha > 0$  then*

$$\mathcal{P}_D(x + \alpha v) = x \Leftrightarrow \mathcal{P}_D(x + \beta v) = x, \forall \beta > 0$$

**Proof:** If  $v = 0$ , the lemma is true. Below, assume  $v \neq 0$ . Only need to prove the implication

$$\mathcal{P}_D(x + \alpha v) = x \Rightarrow \mathcal{P}_D(x + \beta v) = x, \forall \beta > 0$$

For an arbitrary  $\beta > 0$ , there are two cases,

- If  $\beta \geq \alpha$ , by Lemma 5.1.4,

$$\|\mathcal{P}_D(x + \beta v) - x\| \leq \frac{\beta}{\alpha} \|\mathcal{P}_D(x + \alpha v) - x\| = 0 \quad (5.1.10)$$

- If  $\beta < \alpha$ , by Lemma 5.1.3,

$$\begin{aligned}
\|\mathcal{P}_D(x + \beta v) - x\| &\leq \|\mathcal{P}_D(x + \frac{\alpha}{\beta}\beta v) - x\| \\
&= \|\mathcal{P}_D(x + \alpha v) - x\| \\
&= 0
\end{aligned} \tag{5.1.11}$$

In conclusion,

$$\|\mathcal{P}_D(x + \beta v) - x\| = 0$$

□

**Lemma 5.1.7** *If  $D \in \mathbb{R}^n$  is a closed convex set and  $\mathcal{P}_D$  is the metric projection to  $D$ , then*

$$[x + \delta x - \mathcal{P}_D(x + \delta x)]^T [\mathcal{P}_D(x + \delta x) - x] \geq 0 \quad \forall x, \delta x \in \mathbb{R}^n$$

$$\delta x^T [\mathcal{P}_D(x + \delta x) - x] \geq 0 \quad \forall x, \delta x \in \mathbb{R}^n$$

and  $\delta x^T [\mathcal{P}_D(x + \delta x) - x] = 0$  only when

$$\delta x = 0 \quad \text{or} \quad \mathcal{P}_D(x + \delta x) - x = 0$$

see Figure 5.3.

**Proof:** By the definition of metric projection,

$$\mathcal{P}_D(\mathcal{P}_D(x + \delta x) + [x + \delta x - \mathcal{P}_D(x + \delta x)]) = \mathcal{P}_D(x + \delta x)$$

By Lemma 5.1.1,

$$(\mathcal{P}_D(x + \delta x) - x)^T [x + \delta x - \mathcal{P}_D(x + \delta x)] \geq 0$$

adding  $(\mathcal{P}_D(x + \delta x) - x)^T (\mathcal{P}_D(x + \delta x) - x) \geq 0$  onto the inequality above yields the second result,

$$(\mathcal{P}_D(x + \delta x) - x)^T \delta x \geq 0$$

□

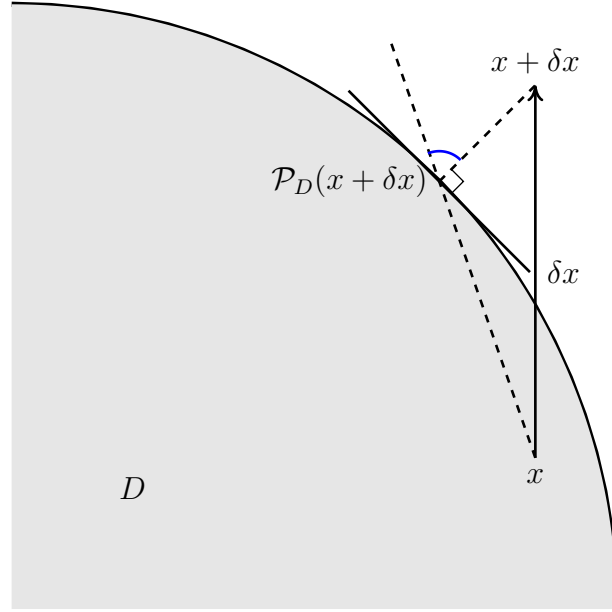


Figure 5.3: Illustration Lemma 5.1.7. The blue angle is no larger than  $\pi/2$ .

## 5.2 Convergence in Constrained Problem

In this chapter, I consider the constrained Discrete-Time-Optimal-Control problem,

$$\text{Minimize } \sum_{k=0}^K J_k(y_k, u_k) \quad (5.2.1a)$$

$$\text{subject to } y_{k+1} = F_k(y_k, u_k), \quad k = 0, \dots, K-1, \quad (5.2.1b)$$

$$y_0 = y_{\text{given}} \quad (5.2.1c)$$

$$u \in D \quad (5.2.1d)$$

where  $u \in \mathbb{R}^{n_u \times K}$  and  $u_k \in \mathbb{R}^{n_u}$ ,  $k = 0, \dots, K-1$  denotes the  $k$ th component vector of  $u$  representing the control variable for time step  $k$ .  $D \subset \mathbb{R}^{n_u \times K}$  is a closed convex sets. Frobenius norm is used for the space  $\mathbb{R}^{n_u \times K}$  a direct result of which is the implication relationship

$$u \in \bar{B}(u^*, R) \subset \mathbb{R}^{n_u \times K} \Rightarrow u_k \in \bar{B}(u_k^*, R) \subset \mathbb{R}^{n_u} \text{ for } k = 0, \dots, K-1.$$

with generic  $u^* \in \mathbb{R}^{n_u \times K}$  and  $R > 0$ . The norm used in  $\mathbb{R}^{n_u}$  is also Frobenius norm (for vectors, it is the same as 2-norm).

In the context of this chapter, for  $u, v \in \mathbb{R}^{n_u \times K}$ , when they are seen as a whole variable as opposed to the combination of  $K$  length  $n_u$  vectors, they are treated as long vectors of length  $Kn_u$  for common linear algebra operations. Particularly, when inner product “ $u^T v$ ” is used, it refers to the vector inner product treating  $u$  and  $v$  as “flattened” vectors of length  $Kn_u$ .

In some occasions, with  $S \in \mathbb{R}^n$  being a non-open set, I use notation  $C^k(S)$  with  $k = 1, 2$  to refer to function class that has  $k$ -th order continuous derivative in the set  $S$ . This is not standard because usually differentiability is defined in an open set. One understands  $C^k(S)$  in the sense that functions in  $C^k(S)$  are defined on an open set containing  $S$  and their  $k$ -th order derivatives are continuous in  $S$ . The notation  $C^k(S)$  is used to describe functions  $F_k, y_k, k = 0, \dots, K - 1$  and is used in Lemma 5.3.4, so that the following implication can be used. In the case where  $S$  is compact, the  $k$ -th derivative of a function in  $C^k(S)$  is bounded in  $S$  by the boundedness of continuous function (i.e., the norm of the derivative) in a compact set.

Since the states  $y_1, \dots, y_K$  are determined by controls  $u_0, \dots, u_{K-1}$  by the state equations (5.2.1b), the notation

$$\hat{J}(u) \stackrel{\text{def}}{=} \sum_{k=0}^K J_k(y_k(u_0, \dots, u_{k-1}), u_k)$$

is used to represent the reduced control space objective function. In some cases,  $\nabla \hat{J}$  or  $\nabla^2 \hat{J}$  is used without differentiability of  $F_k, J_k$  being explicitly assumed in advance. In these cases, the differentiability is not the central topic. Please assume that the proper differentiability assumption of  $F_k, J_k$  is valid.

The parallel gradient-type method with  $N$  time subdomains is combined with metric projection. See Algorithm 13. The step sizes  $\{\alpha_j\}_{j=0}^\infty$  is arbitrary in the algorithm but the convergence theorems below enforce restrictions on them.



---

**Algorithm 13**  $j$ th iteration of the parallel-in-time gradient-type method with step size  $\alpha_j > 0$ . Describes the tasks executed by processor of rank  $n \in \{0, \dots, N - 1\}$

---

- 1: Input control  $u_{K_n}^{(j)}, u_{K_{n+1}}^{(j)}, \dots, u_{K_{n+1}-1}^{(j)}$ . ▷ initialization of the iteration
- 2: **if**  $n > 0$  and  $j = 0$  **then**
- 3:   Input initial  $y_{K_n}^{(-1)}$
- 4: **end if**
- 5: **if**  $n < N - 1$  and  $j = 0$  **then**
- 6:   Input initial  $p_{K_{n+1}}^{(-1)}$
- 7: **end if**
  
- 8: **if**  $n = 0$  **then** ▷ solve the state equation forward in time
- 9:    $y_{K_{n+1}}^{(j)} = F_{K_n}(y_{\text{given}}, u_{K_n}^{(j)})$
- 10: **else**
- 11:    $y_{K_{n+1}}^{(j)} = F_{K_n}(y_{K_n}^{(j-1)}, u_{K_n}^{(j)})$
- 12: **end if**
- 13: **for**  $k = K_n + 1, \dots, K_{n+1} - 1$  **do**
- 14:    $y_{k+1}^{(j)} = F_k(y_k^{(j)}, u_k^{(j)})$
- 15: **end for**
  
- 16: **if**  $n = N - 1$  **then** ▷ solve the adjoint equation backward in time
- 17:    $p_{K_{n+1}-1}^{(j)} = \frac{\partial J_{K_{n+1}}(y_{K_{n+1}}^{(j)}, u_{K_{n+1}}^{(j)})}{\partial y_{K_{n+1}}^{(j)}}$
- 18: **else**
- 19:    $p_{K_{n+1}-1}^{(j)} = \frac{\partial J_{K_{n+1}}(y_{K_{n+1}}^{(j)}, u_{K_{n+1}}^{(j)})}{\partial y_{K_{n+1}}^{(j)}} + \left( \frac{\partial F_{K_{n+1}}(y_{K_{n+1}}^{(j)}, u_{K_{n+1}}^{(j)})}{\partial y_{K_{n+1}}^{(j)}} \right) T p_{K_{n+1}}^{(j-1)}$
- 20: **end if**
- 21: **for**  $k = K_{n+1} - 1, \dots, K_n + 1$  **do**
- 22:    $p_{k-1}^{(j)} = \frac{\partial J_k(y_k^{(j)}, u_k^{(j)})}{\partial y_k^{(j)}} + \left( \frac{\partial F_k(y_k^{(j)}, u_k^{(j)})}{\partial y_k^{(j)}} \right) T p_k^{(j)}$
- 23: **end for**

(continued on next page)

---

---

```

24: for  $k = K_n, \dots, K_{n+1} - 1$  do ▷ update control
25:    $u_k^{(j+1)} = \mathcal{P}_D(u_k^{(j)} - \alpha_j \left[ \frac{\partial J_k(y_k^{(j)}, u_k^{(j)})}{\partial u_k^{(j)}} + \left( \frac{\partial F_k(y_k^{(j)}, u_k^{(j)})}{\partial u_k^{(j)}} \right)^T p_k^{(j)} \right])$ 
26: end for

27: if  $n > 0$  then ▷ communication between processors
28:   send  $p_{K_n}^{(j)}$  to rank  $n - 1$ 
29:   receive  $y_{K_n}^{(j)}$  from rank  $n - 1$ 
30: end if

31: if  $n < N - 1$  then
32:   send  $y_{K_{n+1}}^{(j)}$  to rank  $n + 1$ 
33:   receive  $p_{K_{n+1}}^{(j)}$  from rank  $n + 1$ 
34: end if

```

---

### 5.2.1 Gradient-Type Vector Error Bound

Since the parallel-in-time gradient-type method breaks up the whole time domain into several subdomains, the convergence analysis of the method involves state equation corresponding to individual time steps instead of treating all state equations of all time steps as a whole. To build the convergence theorems upon the state equations properties, I introduce the notion of control set and state set to use frequently in this chapter,  $U \subset \mathbb{R}^{n_u \times K}$  and  $Y(U) \subset \mathbb{R}^{n_y \times K}$ . Use notation  $U_k \subset \mathbb{R}^{n_u}$ ,  $k = 0, \dots, K - 1$  to denote the sets of control variables corresponding to time step  $k$ .

$$U \stackrel{\text{def}}{=} U_0 \times U_1 \times \dots \times U_{K-1}$$

Usually, it is assumed  $U$  is convex and equivalently each of  $U_k$ ,  $k = 0, \dots, K - 1$ , is convex. Given  $U$ , define for  $k = 1, \dots, K$ , convex hull  $Y_k(U) \subset \mathbb{R}^{n_y}$ ,

$$Y_k(U) \stackrel{\text{def}}{=} \text{conv}(\{y_k | y_0 = y_{\text{given}}, y_{l+1} = F_l(y_l, u_l), u_l \in U_l, l = 0, \dots, k - 1\}) \quad (5.2.2a)$$

and

$$Y(U) = Y_1(U) \times Y_2(U) \times \dots \times Y_K(U), \quad (5.2.2b)$$

i.e.,  $Y_k(U)$  contains all states at time step  $k$  possibly generated by controls in  $U$  and convex combination of them. Expression “ $u \in U$ ” is used to represent that “ $u_k \in U_k$ , for  $k = 0, \dots, K-1$ ”. The convexity assumed above guarantees the Lipschitz continuity assumptions made on these sets are well defined.

Use  $\{\alpha_i\}$  to denote the step size used in the parallel-in-time gradient-type method, where iteration  $i$  uses step size  $\alpha_i$ . Notation  $\nabla J_k$  is the gradient of  $J_k$  and  $\nabla F_k$  is the Jacobian of  $F_k$  with respect to the combined argument  $[y_k^T, u_k^T]^T$ .

The following Lemma 5.2.1 is simple in spirit and tedious (but still straightforward) in proof.

**Lemma 5.2.1** *If there exists  $U$  such that*

- $\nabla J_k$  and  $\nabla F_k$  is Lipschitz and bounded for  $u_k \in U_k$  and  $y_k \in Y_k(U)$  for all  $k$

*Then, when the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), exist  $\bar{\alpha}_{rel.gs} > 0$  and constants  $G_U, M_U, \bar{M}_U > 0$  such that, for any  $0 \leq \bar{\alpha} \leq \bar{\alpha}_{rel.gs}$  if for a certain  $i \geq 4(N-1)$*

- $0 \leq \alpha_j \leq \bar{\alpha}$  for all  $i - 2(N-1) \leq j < i$
- $u_k^{(j)} \in U_k$ , for all  $0 \leq k \leq K-1$ , for all  $i - 4(N-1) \leq j \leq i$

*then, the gradient-type vector is bounded, i.e.,*

$$\|g^{(j)}\| \leq G_U \text{ for } i - 2(N-1) \leq j \leq i$$

*and*

$$\begin{aligned} \|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| &\leq M_U \sum_{j=1}^{2(N-1)} \|u^{(i-j+1)} - u^{(i-j)}\| \\ &\leq \bar{\alpha} M_U \sum_{j=1}^{2(N-1)} \|g^{(i-j)}\| \leq \bar{\alpha} \bar{M}_U \end{aligned} \tag{5.2.3}$$

*where the parallel-in-time gradient-type vector in optimization iteration  $i$ ,  $g^{(i)}$ , at time step  $k$  is*

$$g_k^{(i)} \stackrel{\text{def}}{=} \frac{\partial J_k(y_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} + \left( \frac{\partial F_k(y_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} \right)^T p_k^{(i)}.$$

See Algorithm 13 for other notations.

Note that,

- Step size upper bound  $\bar{\alpha}_{\text{rel.gs}}$  is named so because this upper bound is used to establish an inequality describing the relative relationship between the difference from  $g^{(i)}$  to  $\nabla \hat{J}(u^{(i)})$  and consecutive  $g^{(i-1)}, g^{(i-2)}, \dots, g^{(i-2(N-1))}$ .
- Showed in the proof, given a problem, the value of  $\bar{\alpha}_{\text{rel.gs}}$  only depends on  $U$ .

**Proof:** For the  $N = 1$  case, the Lemma 5.2.1 is trivially true since for all  $i$ ,

$$\nabla \hat{J}(u^{(i)}) \equiv g^{(i)}.$$

In the following proof, assume  $N > 1$ . Table 5.2 at the end of the proof summarizes the proof steps below, which may serve as an atlas to the detailed proof.

Since

$$u_k^{(j)} \in U_k, k = 0, \dots, K - 1, j = i - 4(N - 1), \dots, i,$$

we have

$$y_k^{(j)} \in Y_k(U), k = 1, \dots, K, j = i - 3(N - 1), \dots, i.$$

The boundedness and Lipschitz continuity below is in the context where

$$u_k^{(j)} \in U_k, y_y^{(j)} \in Y_k(U) \text{ for all proper } k, j = i - 3(N - 1), \dots, i. \quad (5.2.4)$$

By the assumed boundedness, I first show the computed  $p_k^{(j)}, j \geq i - 2(N - 1)$ , for all  $0 \leq k \leq K - 1$ , is bounded. Note that the “ $p_k^{(j)}$ ”s computed by Algorithm 13 are not the actual adjoint variables but adjoint-type vectors resulted from the formula in Algorithm 13 using a similar formula as the computation of adjoint variables.

By the boundedness of  $\nabla J_k$ , let  $B_{\nabla J}$  be a constant such that  $\|\nabla J_k\|_2 < B_{\nabla J}$  for all time step  $k$ . For simplicity, I do not assign separate constants to the upper bounds of  $\nabla J_k$  with respect to  $y$  and  $u$  respectively.

Using  $\nabla F_k$  to denote the Jacobian of  $F_k$ , let  $B_{\nabla F}$  be a constant such that  $\|\nabla F_k\|_2 < B_{\nabla F}$  for all time step  $k$ . Note that  $B_{\nabla F}$  is also an upper bound of  $\nabla F_k$  with respect to  $y$  and  $u$  respectively.

For  $j \geq i - 2(N - 1)$ , time subdomain  $n \in \{0, \dots, N - 2\}$ , and time step index  $k \in \{K_n, \dots, K_{n+1} - 1\}$ , by the following telescopic back substitution,

$$\begin{aligned}
\|p_k^{(j)}\| &= \left\| \frac{\partial J_{k+1}(y_{k+1}^{(j)}, u_{k+1}^{(j)})}{\partial y_{k+1}^{(j)}} + \left( \frac{\partial F_{k+1}(y_{k+1}^{(j)}, u_{k+1}^{(j)})}{\partial y_{k+1}^{(j)}} \right)^T p_{k+1}^{(j)} \right\| \\
&\leq B_{\nabla J} + B_{\nabla F} \|p_{k+1}^{(j)}\| \leq B_{\nabla J} + B_{\nabla F} (B_{\nabla J} + B_{\nabla F} \|p_{k+2}^{(j)}\|) \\
&\leq \left( \sum_{s=0}^{K_{n+1}-k-1} B_{\nabla F}^s \right) B_{\nabla J} + B_{\nabla F}^{K_{n+1}-k} \|p_{K_{n+1}}^{(j-1)}\| \\
&\leq \left( \sum_{s=0}^{K-2-k} B_{\nabla F}^s \right) B_{\nabla J} + B_{\nabla F}^{K-1-k} \|p_{K-1}^{(j-(N-1-n))}\| \\
&\leq \left( \sum_{s=0}^{K-2-k} B_{\nabla F}^s \right) B_{\nabla J} + B_{\nabla F}^{K-1-k} B_{\nabla J} = \left( \sum_{s=0}^{K-1-k} B_{\nabla F}^s \right) B_{\nabla J} \\
&\leq \left( \sum_{s=0}^{K-1} B_{\nabla F}^s \right) B_{\nabla J}.
\end{aligned} \tag{5.2.5}$$

When  $n = N - 1$  and  $k \in \{K_n, \dots, K_{n+1} - 1\}$ , this bound also holds. So,  $p_k^{(j)}$  is bounded for  $i \geq i - 2(N - 1)$ . It is required that  $j \geq i - 2(N - 1)$  because  $N$  parallel-in-time gradient-type method iterations finish a backward sweep starting from time step  $K$ , the end of the whole time window. For  $j \geq i - 2(N - 1)$ , the  $p_k^{(j)}$  only depends on  $u^{(l)}, y^{(l)}, l = j - 3(N - 1), \dots, j$  by (5.2.5). Since  $u^{(l)} \in U, y^{(l)} \in Y(U), l = j - (N - 1), \dots, j$ , the boundedness and Lipschitz continuity hold in (5.2.5) by (5.2.4).

Then, by (5.2.5), for  $j \geq i - 2(N - 1)$  and all  $k$ ,

$$\begin{aligned}
\|g_k^{(j)}\| &= \left\| \frac{\partial J_k(y_k^{(j)}, u_k^{(j)})}{\partial u_k^{(j)}} + \left( \frac{\partial F_k(y_k^{(j)}, u_k^{(j)})}{\partial u_k^{(j)}} \right)^T p_k^{(j)} \right\| \\
&\leq B_{\nabla J} + B_{\nabla F} \left[ \left( \sum_{s=0}^{K-1} B_{\nabla F}^s \right) B_{\nabla J} \right] = \left( \sum_{s=0}^K B_{\nabla F}^s \right) B_{\nabla J}.
\end{aligned} \tag{5.2.6}$$

Define  $B_g \stackrel{\text{def}}{=} \left( \sum_{s=0}^K B_{\nabla F}^s \right) B_{\nabla J}$  and thus

$$\|g_k^{(j)}\| \leq B_g, j \geq i - 2(N - 1), k = 0, \dots, K - 1$$

which leads to the boundedness ( $g^{(j)} \in \mathbb{R}^{n_u \times K}$ ), with  $G_U \stackrel{\text{def}}{=} \sqrt{K} B_g$ ,

$$\|g^{(j)}\|_{\text{Frobenius}} = \sqrt{\sum_{k=0}^{K-1} \|g_k^{(j)}\|^2} \leq \sqrt{K} B_g = G_U, j \geq i - 2(N - 1)$$

Assume  $\{\alpha_i\}$  has an upper bound  $\bar{\alpha}$ . This leads to a bound on the difference of controls generated by several consecutive iterations, i.e., for  $1 \leq j \leq 2(N - 1)$ ,

$$\begin{aligned} \|u^{(i)} - u^{(i-j)}\| &= \|u^{(i)} - (u^{(i)} + \sum_{s=1}^j \alpha_{i-s} g^{(i-s)})\| \\ &\leq \sum_{s=1}^j \alpha_{i-s} \|g^{(i-s)}\| \leq \bar{\alpha} \sum_{s=1}^j \|g^{(i-s)}\| \\ &\leq \bar{\alpha} j B_g \leq \bar{\alpha} B_g (2N - 2). \end{aligned} \quad (5.2.7)$$

Note that this holds trivially for  $j = 0$ .

Let  $L_{\nabla F}$  be a Lipschitz coefficients for all Jacobians of  $F_k$ ,  $k = 0, \dots, K - 1$ . Then, by the assumed Lipschitz continuity of the Jacobians, for any  $k$ , for generic  $y, u$ , and  $\delta_u$  of proper dimensions,

$$\|F_k(y, u) - F_k(y, u + \delta_u)\| \quad (5.2.8a)$$

$$= \left\| \int_0^1 \frac{\partial F_k(y, u + s\delta_u)}{\partial u} \delta_u ds \right\| \quad (5.2.8b)$$

$$= \left\| \int_0^1 \left[ \frac{\partial F_k(y, u)}{\partial u} + \left( \frac{\partial F_k(y, u + s\delta_u)}{\partial u} - \frac{\partial F_k(y, u)}{\partial u} \right) \right] \delta_u ds \right\| \quad (5.2.8c)$$

$$\leq \int_0^1 \left[ \left\| \frac{\partial F_k(y, u)}{\partial u} \right\| + \left\| \frac{\partial F_k(y, u + s\delta_u)}{\partial u} - \frac{\partial F_k(y, u)}{\partial u} \right\| \right] \|\delta_u\| ds \quad (5.2.8d)$$

$$\leq \int_0^1 \left[ \left\| \frac{\partial F_k(y, u)}{\partial u} \right\| + L_{\nabla F} \|s\delta_u\| \right] \|\delta_u\| ds \quad (5.2.8e)$$

$$\leq \left\| \frac{\partial F_k(y, u)}{\partial u} \right\| \|\delta_u\| + \frac{L_{\nabla F}}{2} \|\delta_u\|^2. \quad (5.2.8f)$$

Lipschitz continuity with respect to the second argument of  $F_k$  on a convex set containing the line segment between  $u$  and  $u + \delta_u$  is used from (5.2.8d) to (5.2.8e).

Use  $\tilde{y}_k^{(i)}$ ,  $k = 0, \dots, K$ , to denote the true solution of the state equations (5.2.1b)

(5.2.1c), given control  $u_0^{(i)}, u_1^{(i)}, \dots, u_{K-1}^{(i)}$ , i.e.

$$\begin{cases} \tilde{y}_{k+1}^{(i)} &= F_k(\tilde{y}_k^{(i)}, u_k^{(i)}), \quad k = 0, \dots, K-1, \\ \tilde{y}_0^{(i)} &= y_{\text{given}}. \end{cases}$$

For  $0 \leq j \leq 2(N-1)$ , at the first time step of state equations,

$$\begin{aligned} \|\tilde{y}_1^{(i)} - y_1^{(i-j)}\| &= \|F_0(y_{\text{given}}, u_0^{(i)}) - F_0(y_{\text{given}}, u_0^{(i-j)})\| \\ &\leq \left\| \frac{\partial F(y_{\text{given}}, u_0^{(i)})}{\partial u_0^{(i)}} \right\| \|u_0^{(i)} - u_0^{(i-j)}\| + \frac{L_{\nabla F}}{2} \|u_0^{(i)} - u_0^{(i-j)}\|^2 \\ &= \left( \left\| \frac{\partial F(y_{\text{given}}, u_0^{(i)})}{\partial u_0^{(i)}} \right\| + \frac{L_{\nabla F}}{2} \|u_0^{(i)} - u_0^{(i-j)}\| \right) \|u_0^{(i)} - u_0^{(i-j)}\| \\ &\leq (B_{\nabla F} + \frac{\bar{\alpha} L_{\nabla F} B_g (2N-2)}{2}) \|u_0^{(i)} - u_0^{(i-j)}\|. \end{aligned} \tag{5.2.9a}$$

For  $k \in \{1, 2, \dots, K-1\} \setminus \{K_1, K_2, \dots, K_{N-1}\}$ ,

$$\begin{aligned} \|\tilde{y}_{k+1}^{(i)} - y_{k+1}^{(i-j)}\| &= \|F_k(\tilde{y}_k^{(i)}, u_k^{(i)}) - F_k(y_k^{(i-j)}, u_k^{(i-j)})\| \\ &\leq \|F_k(\tilde{y}_k^{(i)}, u_k^{(i)}) - F_k(\tilde{y}_k^{(i)}, u_k^{(i-j)})\| \\ &\quad + \|F_k(\tilde{y}_k^{(i)}, u_k^{(i-j)}) - F_k(y_k^{(i-j)}, u_k^{(i-j)})\| \\ &\leq (B_{\nabla F} + \frac{\bar{\alpha} L_{\nabla F} B_g (2N-2)}{2}) \|u_k^{(i)} - u_k^{(i-j)}\| \\ &\quad + (B_{\nabla F} + \frac{L_{\nabla F}}{2} \|\tilde{y}_k^{(i)} - y_k^{(i-j)}\|) \|\tilde{y}_k^{(i)} - y_k^{(i-j)}\|. \end{aligned} \tag{5.2.9b}$$

Similarly, but at the time subdomain boundaries, for  $k \in \{K_1, K_2, \dots, K_{N-1}\}$ ,

$$\begin{aligned} \|\tilde{y}_{k+1}^{(i)} - y_{k+1}^{(i-j)}\| &= \|F_k(\tilde{y}_k^{(i-1)}, u_k^{(i)}) - F_k(y_k^{(i-j-1)}, u_k^{(i-j)})\| \\ &\leq \|F_k(\tilde{y}_k^{(i-1)}, u_k^{(i)}) - F_k(\tilde{y}_k^{(i-1)}, u_k^{(i-j)})\| \\ &\quad + \|F_k(\tilde{y}_k^{(i-1)}, u_k^{(i-j)}) - F_k(y_k^{(i-j-1)}, u_k^{(i-j)})\| \\ &\leq (B_{\nabla F} + \frac{\bar{\alpha} L_{\nabla F} B_g (2N-2)}{2}) \|u_k^{(i)} - u_k^{(i-j)}\| \\ &\quad + (B_{\nabla F} + \frac{L_{\nabla F}}{2} \|\tilde{y}_k^{(i-1)} - y_k^{(i-j-1)}\|) \|\tilde{y}_k^{(i-1)} - y_k^{(i-j-1)}\|. \end{aligned} \tag{5.2.9c}$$

Note that in (5.2.9a)(5.2.9b)(5.2.9c),  $\frac{\bar{\alpha} L_{\nabla F} B_g (2N-2)}{2}$  as a part of coefficient of  $\|u_k^{(i)} - u_k^{(i-j)}\|$  in the equalities RHS, can be made arbitrarily small by choosing  $\bar{\alpha}$ , e.g., for

sufficiently small  $\bar{\alpha}$

$$B_{\nabla F} + \frac{\bar{\alpha} L_{\nabla F} B_g (2N - 2)}{2} \leq B_{\nabla F} + 1.$$

By (5.2.9a)(5.2.9b)(5.2.9c) and at most  $K - 1$  back substitutions, one can see that  $\|\tilde{y}_k^{(i)} - y_k^{(i-j)}\|$  is bounded by  $\|u^{(i)} - u^{(i-j)}\|$  scaled by a constant scalar. Additionally, using the fact that, for any  $0 \leq j \leq 2(N - 1)$ ,

$$\|u^{(i)} - u^{(i-j)}\| \leq \sum_{k=1}^j \|u^{(i-k+1)} - u^{(i-k)}\| \leq \sum_{k=1}^{2(N-1)} \|u^{(i-k+1)} - u^{(i-k)}\| \quad (5.2.10)$$

exists  $M_y > 0$ , such that, with sufficiently small  $\bar{\alpha} > 0$ , for  $0 \leq j \leq N - 1$ ,

$$\|\tilde{y}^{(i)} - y^{(i-j)}\| \leq M_y \|u^{(i)} - u^{(i-j)}\| \leq M_y \sum_{j=1}^{2(N-1)} \|u^{(i-j+1)} - u^{(i-j)}\|. \quad (5.2.11)$$

Note that the reason why  $0 \leq j \leq N - 1$  instead of  $0 \leq j \leq 2(N - 1)$  is for the back substitutions of (5.2.9a)(5.2.9b)(5.2.9c) involve  $N - 1$  previous iterations, i.e.  $i - j - 1, i - j - 2, \dots, i - j - (N - 1)$ . Let  $\tilde{p}^{(i)}$  be the adjoint variable corresponding to  $u^{(i)}$  and  $\tilde{y}^{(i)}$ , i.e.

$$\begin{cases} \tilde{p}_{K-1}^{(i)} &= \frac{\partial J_K(\tilde{y}_K^{(i)}, u_K^{(i)})}{\partial \tilde{y}_K^{(i)}} \\ \tilde{p}_{k-1}^{(i)} &= \frac{\partial J_k(\tilde{y}_k^{(i)}, u_k^{(i)})}{\partial \tilde{y}_k^{(i)}} + \left( \frac{\partial F_k(\tilde{y}_k^{(i)}, u_k^{(i)})}{\partial \tilde{y}_k^{(i)}} \right)^T \tilde{p}_k^{(i)}, \quad k = K - 1, \dots, 1 \end{cases}$$

By the same logic as (5.2.9), with the condition (5.2.11), exists  $M_p > 0$ , such that, with sufficiently small  $\bar{\alpha} > 0$ ,

$$\|\tilde{p}^{(i)} - p^{(i)}\| \leq \bar{\alpha} M_p \sum_{j=1}^{2(N-1)} \|u^{(i-j+1)} - u^{(i-j)}\|. \quad (5.2.12)$$

i.e. a bound on the adjoint variables for the single iteration  $i$ . Given  $J_0, \dots, J_K$  and  $F_0, \dots, F_{K-1}$  in (5.2.1), the upper bound of the sufficiently small step size required in (5.2.11) and (5.2.12) is determined by  $U, Y(U)$  and ultimately only by  $U$ . Denote this upper bound by  $\bar{\alpha}_{\text{rel.gs}}$ . Below, I require

$$\bar{\alpha} \leq \bar{\alpha}_{\text{rel.gs}} \quad (5.2.13)$$



Finally, for  $i \geq 2(N-1)$ , build the bound of  $\|\nabla \hat{J}(u^{(i)}) - g^{(i)}\|$ . Since

$$\nabla_{u_k^{(i)}} \hat{J}(u^{(i)}) = \frac{\partial J_k(\tilde{y}_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} + \left( \frac{\partial F_k(\tilde{y}_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} \right)^T \tilde{p}_k^{(i)}$$

and

$$g_k^{(i)} = \frac{\partial J_k(y_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} + \left( \frac{\partial F_k(y_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} \right)^T p_k^{(i)},$$

the difference in time step  $k$  is

$$\nabla_{u_k^{(i)}} \hat{J}(u^{(i)}) - g_k^{(i)} = \left[ \frac{\partial J_k(\tilde{y}_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} - \frac{\partial J_k(y_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} \right] \quad (5.2.14a)$$

$$+ \left( \frac{\partial F_k(\tilde{y}_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} \right)^T (\tilde{p}_k^{(i)} - p_k^{(i)}) \quad (5.2.14b)$$

$$+ \left[ \left( \frac{\partial F_k(\tilde{y}_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} \right)^T - \left( \frac{\partial F_k(y_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} \right)^T \right] p_k^{(i)}. \quad (5.2.14c)$$

Now, bound each of the three terms in the RHS (5.2.14). By the Lipschitz continuity, let  $L_{\nabla J}$  be a Lipschitz coefficient of  $\nabla J_k$  for all  $k$ .

The RHS of (5.2.14a), by (5.2.11),

$$\begin{aligned} \left\| \frac{\partial J_k(\tilde{y}_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} - \frac{\partial J_k(y_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} \right\| &\leq L_{\nabla J} \|\tilde{y}_k^{(i)} - y_k^{(i)}\| \\ &\leq L_{\nabla J} M_y \sum_{j=1}^{2(N-1)} \|u^{(i-j+1)} - u^{(i-j)}\|. \end{aligned} \quad (5.2.15)$$

The term in (5.2.14b), by (5.2.12),

$$\begin{aligned} \left\| \left( \frac{\partial F_k(\tilde{y}_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} \right)^T (\tilde{p}_k^{(i)} - p_k^{(i)}) \right\| &\leq B_{\nabla F} \|\tilde{p}_k^{(i)} - p_k^{(i)}\| \\ &\leq B_{\nabla F} M_p \sum_{j=1}^{2(N-1)} \|u^{(i-j+1)} - u^{(i-j)}\|. \end{aligned} \quad (5.2.16)$$

Lastly, the term in (5.2.14c), by (5.2.5) and (5.2.11),

$$\begin{aligned} &\left\| \left[ \left( \frac{\partial F_k(\tilde{y}_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} \right)^T - \left( \frac{\partial F_k(y_k^{(i)}, u_k^{(i)})}{\partial u_k^{(i)}} \right)^T \right] p_k^{(i)} \right\| \\ &\leq L_{\nabla F} \|\tilde{y}_k^{(i)} - y_k^{(i)}\| \|p_k^{(i)}\| \\ &\leq L_{\nabla F} M_y \left( \sum_{s=0}^{K-1} B_{\nabla F}^s \right) B_{\nabla J} \sum_{j=1}^{2(N-1)} \|u^{(i-j+1)} - u^{(i-j)}\|. \end{aligned} \quad (5.2.17)$$

By inserting (5.2.15)(5.2.16), and (5.2.17) into (5.2.14), there exist  $M'_U > 0$  such that,

$$\|\nabla_{u_k^{(i)}} \hat{J}(u^{(i)}) - g_k^{(i)}\| \leq M'_U \sum_{j=1}^{2(N-1)} \|u^{(i-j+1)} - u^{(i-j)}\| \quad \text{for } 0 \leq k \leq K-1. \quad (5.2.18)$$

Then, exist a scalar  $M_U$  for the inequality corresponding to the whole time domain,

$$\|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \leq M_U \sum_{j=1}^{2(N-1)} \|u^{(i-j+1)} - u^{(i-j)}\|. \quad (5.2.19)$$

This is the first inequality in (5.2.3).

By projected parallel-in-time gradient-type method,

$$u^{(j+1)} = \mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) \quad \forall j \geq 0.$$

Then the control update is bounded, for all  $j \geq 0$ ,

$$\begin{aligned} \|u^{(j+1)} - u^{(j)}\| &= \|\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - u^{(j)}\| \\ &= \|\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - \mathcal{P}_D(u^{(j)})\| \\ &\leq \|(u^{(j)} - \alpha_j g^{(j)}) - u^{(j)}\| \\ &= \alpha_j \|g^{(j)}\| \end{aligned} \quad (5.2.20)$$

This yields the second inequality in (5.2.3),

$$M_U \sum_{j=1}^{2(N-1)} \|u^{(i-j+1)} - u^{(i-j)}\| \leq \bar{\alpha} M_U \sum_{j=1}^{2(N-1)} \|g^{(i-j)}\| \quad (5.2.21)$$

By (5.2.6),  $\|g^{(i-j)}\|$  is bounded for  $j = 1, \dots, 2(N-1)$ , then exists  $\bar{M}_U$  such that

$$\bar{\alpha} M_U \sum_{j=1}^{2(N-1)} \|g^{(i-j)}\| \leq \bar{\alpha} \bar{M}_U \quad (5.2.22)$$

which is the third inequality in (5.2.3). Note that these conclusions are made under the assumption  $\bar{\alpha} \leq \bar{\alpha}_{\text{rel.gs}}$  in (5.2.13).

□

$j \geq i - 4(N - 1)$	$u_k^{(j)} \in U_k$
$j \geq i - 3(N - 1)$	$y_k^{(j)} \in Y_k(U)$
$j \geq i - 2(N - 1)$	$p_k^{(j)} \leq (\sum_{s=0}^{K-1} B_{\nabla F}^s) B_{\nabla J}$ $\ u^{(i)} - u^{(i-j)}\  \leq \bar{\alpha} B_g(2N - 2)$
$j \geq i - (N - 1)$	$\ \tilde{y}^{(i)} - y^{(i-j)}\  \leq \bar{\alpha} M_y \sum_{j=1}^{2N-2} \ g^{(i-j)}\ $
$j = i$	$\ \tilde{p}^{(i)} - p^{(i)}\  \leq \bar{\alpha} M_p \sum_{j=1}^{2N-2} \ g^{(i-j)}\ $ $\ \nabla \hat{J}(u^{(i)}) - g^{(i)}\  \leq \bar{\alpha} M_U \sum_{j=1}^{2N-2} \ g^{(i-j)}\ .$

Table 5.2: Summary of Lemma 5.2.1 Proof. See the proof for the notations.

Consider a special scenario where step size  $\alpha = 0$ . In this case,  $2N - 1$  parallel-in-time gradient-type iterations with the same control (as a result of  $2(N - 1)$  steps of step size  $\alpha = 0$  parallel-in-time gradient-type iteration) are virtually a pair of a full forward and a full backward computation. Starting from any iteration  $i$ , if  $2(N - 1)$  step size 0 iterations are executed, any further step size 0 iteration will not change any state/adjoint variables in the algorithm, i.e.,

$$p^{(i+(2N-1))} = p^{(i+(2N-1)+k)} \quad \text{and} \quad y^{(i+(2N-1))} = y^{(i+(2N-1)+k)} \quad , \forall k > 0$$

So, practically, by performing a full forward/backward sweep, using  $2(N - 1)$  steps of step size 0 parallel-in-time gradient-type iterations, one achieves the same state/adjoint variables values as arbitrarily many (more than  $2(N - 1)$ ) step size 0 iterations will have. By this observation, I introduce the Corollary 5.2.2 on the effect of full gradient step on the parallel algorithm.

**Corollary 5.2.2** *If there exists  $U$  such that*

- $\nabla J_k$  and  $\nabla F_k$  is Lipschitz and bounded for  $u_k \in U_k$  and  $y_k \in Y_k(U)$  for all  $k$
- exists  $i^*$  such that  $u^{(i^*)} \in U$  and step sizes  $\alpha_{i^*}, \alpha_{i^*+1}, \dots, \alpha_{i^*+2(N-1)} = 0$   
 (“full gradient step”)

Then, when the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), exist  $\bar{\alpha}_{rel.gs} > 0$  and constants  $M_U, \bar{M}_U > 0$  such that, for any  $0 \leq \bar{\alpha} \leq \bar{\alpha}_{rel.gs}$  if

- for a certain  $i \geq i^* + 2(N - 1)$ ,  $0 \leq \alpha_j \leq \bar{\alpha}$  for all  $\max(i - 2(N - 1), i^*) \leq j < i$
- $u^{(j)} \in U$ , for all  $\max(i - 4(N - 1), i^*) \leq j \leq i$

then

$$\|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \leq M_U \sum_{j=1}^{2(N-1)} \|u^{(i-j+1)} - u^{(i-j)}\| \leq \bar{\alpha} M_U \sum_{j=1}^{2(N-1)} \|g^{(i-j)}\| \leq \bar{\alpha} \bar{M}_U$$

Corollary 5.2.2 says if a full gradient step is executed, to check the assumptions in Lemma 5.2.1 of control being in a predetermined set, one does not need to examine steps before a full gradient step.

## 5.2.2 Convergence of the Control Update

Lemma 5.2.1 gives a bound of the difference between the gradient-type vector and the true gradient based on which the similarity of behavior of the parallel algorithm and the serial classic gradient method can be utilized to prove the following convergence theorem.

**Theorem 5.2.3** *In problem (5.2.1), if there exists  $U$  such that*

- $\nabla J_k$  and  $\nabla F_k$  is Lipschitz and bounded for  $u_k \in U_k$  and  $y_k \in Y_k(U)$  for all  $k$
- $\hat{J}(u)$  is bounded from below

When the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), exists  $\bar{\alpha}_{conv.p.} > 0$  such that, if for a certain iteration index  $i$ ,

- the step sizes satisfies

$$\begin{aligned} 0 \leq \alpha_j &\leq \bar{\alpha}_{conv.p.} && \text{for } i - 2(N - 1) \leq j < i \\ 0 < \alpha_j &\leq \bar{\alpha}_{conv.p.} && \text{for } j \geq i \end{aligned}$$

- $u_k^{(j)} \in U_k$ , for all  $0 \leq k \leq K - 1$ , for  $j \geq i - 4(N - 1)$

then

$$\lim_{j \rightarrow +\infty} \|\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}\| = 0 \quad (5.2.23)$$

If in addition, there exists  $0 < \underline{\alpha}_{conv.p.} \leq \bar{\alpha}_{conv.p.}$  such that

- the step size is bounded away from zero

$$\underline{\alpha}_{conv.p.} \leq \alpha_j \leq \bar{\alpha}_{conv.p.} \quad \text{for } j \geq i \quad (5.2.24)$$

then

$$\lim_{j \rightarrow +\infty} \|\mathcal{P}_D(u^{(j)} - \nabla \hat{J}(u^{(j)})) - u^{(j)}\| = 0 \quad (5.2.25)$$

**Proof:**

In the first part of the proof, I show

$$\lim_{j \rightarrow \infty} \|u^{(j+1)} - u^{(j)}\| = 0 \quad (5.2.26)$$

For  $j \geq i$ , the parallel-in-time gradient-type method iteration is defined as

$$u^{(j+1)} = \mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) \quad (5.2.27)$$

By (5.2.27), the convexity of  $D$ , and Lemma 5.1.7,

$$(u^{(j)} - \alpha_j g^{(j)} - u^{(j+1)})^T (u^{(j+1)} - u^{(j)}) \geq 0 \quad (5.2.28)$$

which is rearranged, by the fact that  $\alpha_j \geq \underline{\alpha}_{\text{conv.p.}} > 0$ , to

$$-g^{(j)T}(u^{(j+1)} - u^{(j)}) \geq \frac{1}{\alpha_j} \|u^{(j+1)} - u^{(j)}\|^2 \quad (5.2.29)$$

Using Taylor expansion and Lipschitz constant  $L$  of  $\nabla \hat{J}$  due to the assumption that  $\nabla J_k, \nabla F_k$  are Lipschitz and bounded for all proper  $k$ ,

$$\begin{aligned} & \hat{J}(u^{(j+1)}) - \hat{J}(u^{(j)}) \\ & \leq \nabla \hat{J}(u^{(j)})^T (u^{(j+1)} - u^{(j)}) + \frac{L}{2} \|u^{(j+1)} - u^{(j)}\|^2 \\ & = g^{(j)T} (u^{(j+1)} - u^{(j)}) + [\nabla \hat{J}(u^{(j)}) - g^{(j)}]^T (u^{(j+1)} - u^{(j)}) + \frac{L}{2} \|u^{(j+1)} - u^{(j)}\|^2 \end{aligned}$$

Pick  $\mu > 0$  whose value will be determined later in this proof and use (5.2.29),

$$\begin{aligned} & \hat{J}(u^{(j)}) - \hat{J}(u^{(j+1)}) \\ & \geq -g^{(j)T} (u^{(j+1)} - u^{(j)}) - [\nabla \hat{J}(u^{(j)}) - g^{(j)}]^T (u^{(j+1)} - u^{(j)}) - \frac{L}{2} \|u^{(j+1)} - u^{(j)}\|^2 \\ & \geq \left(\frac{1}{\alpha_j} - \frac{L}{2}\right) \|u^{(j+1)} - u^{(j)}\|^2 - \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\| \|u^{(j+1)} - u^{(j)}\| \\ & \geq \left(\frac{1}{\bar{\alpha}_{\text{conv.p.}}} - \frac{L}{2} - \mu\right) \|u^{(j+1)} - u^{(j)}\|^2 + \mu \|u^{(j+1)} - u^{(j)}\|^2 - \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\| \|u^{(j+1)} - u^{(j)}\| \end{aligned} \quad (5.2.30)$$

Sum up the above inequality (5.2.30) from  $j = i$  to  $j = m$ ,

$$\begin{aligned} \hat{J}(u^{(i)}) - \hat{J}(u^{(m+1)}) & \geq \left(\frac{1}{\bar{\alpha}_{\text{conv.p.}}} - \frac{L}{2} - \mu\right) \sum_{j=i}^m \|u^{(j+1)} - u^{(j)}\|^2 \\ & \quad + \sum_{j=i}^m [\mu \|u^{(j+1)} - u^{(j)}\|^2 - \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\| \|u^{(j+1)} - u^{(j)}\|] \end{aligned} \quad (5.2.31)$$

By the assumptions that  $\hat{J}(u)$  is bounded below for  $u \in U$  and  $u^{(j)} \in U$  for  $j \geq i$ , with a fixed  $i$ , exists  $B_j$ , such that

$$\hat{J}(u^{(i)}) - \hat{J}(u^{(m+1)}) \leq B_j \quad \forall m \geq i \quad (5.2.32)$$

To show (5.2.26), I bound the second summation in (5.2.31) from below as follows.

$$\begin{aligned}
& \sum_{j=i}^m [\mu \|u^{(j+1)} - u^{(j)}\|^2 - \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\| \|u^{(j+1)} - u^{(j)}\|] \\
&= \mu \sum_{j=i}^m \|u^{(j+1)} - u^{(j)}\|^2 - \sum_{j=i}^m \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\| \|u^{(j+1)} - u^{(j)}\| \\
&\geq \mu \sum_{j=i}^m \|u^{(j+1)} - u^{(j)}\|^2 - \frac{1}{2} \sum_{j=i}^m [\|\nabla \hat{J}(u^{(j)}) - g^{(j)}\|^2 + \|u^{(j+1)} - u^{(j)}\|^2] \\
&= (\mu - \frac{1}{2}) \sum_{j=i}^m \|u^{(j+1)} - u^{(j)}\|^2 - \frac{1}{2} \sum_{j=i}^m \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\|^2
\end{aligned} \tag{5.2.33}$$

In (5.2.33), the control update term is rearranged below into an average form so that the bound in Lemma 5.2.1 that involves terms in multiple iterations can be used and ultimately provides a lower bound,

$$\sum_{j=i}^m \|u^{(j+1)} - u^{(j)}\|^2 = C + \frac{1}{2(N-1)} \sum_{j=i+2(N-1)}^m \sum_{k=1}^{2(N-1)} \|u^{(j+1-k)} - u^{(j-k)}\|^2 \tag{5.2.34}$$

where the term  $C$  only involves fractions of the first several terms in the series in the LHS of (5.2.34), namely,

$$\begin{aligned}
C &\stackrel{\text{def}}{=} \sum_{j=i}^m \|u^{(j+1)} - u^{(j)}\|^2 - \frac{1}{2(N-1)} \sum_{j=i+2(N-1)}^m \sum_{k=1}^{2(N-1)} \|u^{(j+1-k)} - u^{(j-k)}\|^2 \\
&= \frac{1}{2(N-1)} \sum_{j=i}^{i+2(N-1)-2} [i + 2(N-1) - 1 - j] \|u^{(j+1)} - u^{(j)}\|^2 \geq 0
\end{aligned} \tag{5.2.35}$$

The term  $C$  is positive, but its being positive is not an essential part of the proof though it is used in (5.2.36) for simplicity. What is important is that  $C$  is independent from the iteration index  $m$ . By Cauchy-Schwarz inequality, (5.2.35), and (5.2.34),

$$\begin{aligned}
\sum_{j=i}^m \|u^{(j+1)} - u^{(j)}\|^2 &\geq \frac{1}{2(N-1)} \sum_{j=i+2(N-1)}^m \sum_{k=1}^{2(N-1)} \|u^{(j+1-k)} - u^{(j-k)}\|^2 \\
&\geq \frac{1}{4(N-1)^2} \sum_{j=i+2(N-1)}^m \left( \sum_{k=1}^{2(N-1)} \|u^{(j+1-k)} - u^{(j-k)}\| \right)^2
\end{aligned} \tag{5.2.36}$$

Let  $\bar{\alpha}_{\text{rel.gs.}}$  and  $M_U$  be given by Lemma 5.2.1. In the case when

$$\bar{\alpha}_{\text{conv.p.}} \leq \bar{\alpha}_{\text{rel.gs.}}, \quad (5.2.37)$$

By (5.2.36) and Lemma 5.2.1,

$$\sum_{j=i}^m \|u^{(j+1)} - u^{(j)}\|^2 \geq \frac{1}{4(N-1)^2} \sum_{j=i+2(N-1)}^m \frac{1}{M_U^2} \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\|^2 \quad (5.2.38)$$

Insert (5.2.38) into (5.2.33) and use  $\bar{M}_U$  in Lemma 5.2.1,

$$\begin{aligned} & \sum_{j=i}^m [\mu \|u^{(j+1)} - u^{(j)}\|^2 - \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\| \|u^{(j+1)} - u^{(j)}\|] \\ & \geq (\mu - \frac{1}{2}) \left[ \frac{1}{4(N-1)^2} \sum_{j=i+2(N-1)}^m \frac{1}{M_U^2} \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\|^2 \right] - \frac{1}{2} \sum_{j=i}^m \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\|^2 \\ & = \left( \frac{\mu - \frac{1}{2}}{4(N-1)^2 M_U^2} - \frac{1}{2} \right) \sum_{j=i+2(N-1)}^m \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\|^2 - \frac{1}{2} \sum_{j=i}^{i+2(N-1)-1} \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\|^2 \\ & \geq \left( \frac{\mu - \frac{1}{2}}{4(N-1)^2 M_U^2} - \frac{1}{2} \right) \sum_{j=i+2(N-1)}^m \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\|^2 - \frac{2(N-1)-1}{2} \bar{\alpha}_{\text{conv.p.}} \bar{M}_U \end{aligned} \quad (5.2.39)$$

For any

$$\mu \geq \frac{4(N-1)^2 M_U^2 + 1}{2}, \quad (5.2.40)$$

by (5.2.39),

$$\sum_{j=i}^m [\mu \|u^{(j+1)} - u^{(j)}\|^2 - \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\| \|u^{(j+1)} - u^{(j)}\|] \geq -\frac{2(N-1)-1}{2} \bar{\alpha}_{\text{conv.p.}} \bar{M}_U \quad (5.2.41)$$

Using (5.2.31), (5.2.32), and (5.2.41), when  $\mu$  satisfies (5.2.40),

$$\left( \frac{1}{\bar{\alpha}_{\text{conv.p.}}} - \frac{L}{2} - \mu \right) \sum_{j=i}^m \|u^{(j+1)} - u^{(j)}\|^2 \leq B_j + \frac{2(N-1)-1}{2} \bar{\alpha}_{\text{conv.p.}} \bar{M}_U \quad (5.2.42)$$

If the step size upper bound  $\bar{\alpha}_{\text{conv.p.}}$  meets the requirement below,

$$\bar{\alpha}_{\text{conv.p.}} < \frac{1}{\frac{L}{2} + \mu}, \quad (5.2.43)$$



(5.2.42) leads to

$$\sum_{j=i}^m \|u^{(j+1)} - u^{(j)}\|^2 \leq \frac{B_j + \frac{2(N-1)-1}{2} \bar{\alpha}_{\text{conv.p.}} \bar{M}_U}{\frac{1}{\bar{\alpha}_{\text{conv.p.}}} - \frac{L}{2} - \mu}$$

The process of  $m \rightarrow +\infty$  implies (5.2.26). This concludes the first part of the proof.

The rest of the proof is simple. (5.2.26) is rewritten into

$$\lim_{j \rightarrow \infty} \|\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - u^{(j)}\| = 0 \quad (5.2.44)$$

By (5.2.26) again and Lemma 5.2.1,

$$\lim_{j \rightarrow +\infty} \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\| = 0 \quad (5.2.45)$$

By triangle inequality and Lemma 5.1.2,

$$\begin{aligned} 0 &\leq \|\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}^{(j)}) - u^{(j)}\| \\ &\leq \|\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - u^{(j)}\| \\ &\quad + \|\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - \mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}^{(j)})\| \\ &\leq \|\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - u^{(j)}\| + \|\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - \mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}^{(j)})\| \\ &\leq \|\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - u^{(j)}\| + \|[u^{(j)} - \alpha_j g^{(j)}] - [u^{(j)} - \alpha_j \nabla \hat{J}^{(j)}]\| \\ &\leq \|\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - u^{(j)}\| + \bar{\alpha}_{\text{conv.p.}} \|\nabla \hat{J}(u^{(j)}) - g^{(j)}\| \end{aligned} \quad (5.2.46)$$

Then, use (5.2.44) and (5.2.45),

$$\lim_{j \rightarrow +\infty} \|\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}\| = 0 \quad (5.2.47)$$

This is (5.2.23).

If, in addition, (5.2.24) holds, for all  $j \geq i$ ,

- In the case  $\underline{\alpha}_{\text{conv.p.}} < 1$ , by Lemma 5.1.3 and Lemma 5.1.4,

$$\begin{aligned} &\|\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}\| \\ &\geq \|\mathcal{P}_D(u^{(j)} - \underline{\alpha}_{\text{conv.p.}} \nabla \hat{J}(u^{(j)})) - u^{(j)}\| \\ &= \underline{\alpha}_{\text{conv.p.}} \frac{1}{\underline{\alpha}_{\text{conv.p.}}} \|\mathcal{P}_D(u^{(j)} - \underline{\alpha}_{\text{conv.p.}} \nabla \hat{J}(u^{(j)})) - u^{(j)}\| \\ &\geq \underline{\alpha}_{\text{conv.p.}} \|\mathcal{P}_D(u^{(j)} - \nabla \hat{J}(u^{(j)})) - u^{(j)}\| \end{aligned} \quad (5.2.48)$$

- In the case  $\underline{\alpha}_{\text{conv.p.}} \geq 1$ , by Lemma 5.1.3,

$$\begin{aligned}
& \|\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}\| \\
& \geq \|\mathcal{P}_D(u^{(j)} - \underline{\alpha}_{\text{conv.p.}} \nabla \hat{J}(u^{(j)})) - u^{(j)}\| \\
& \geq \|\mathcal{P}_D(u^{(j)} - \nabla \hat{J}(u^{(j)})) - u^{(j)}\|
\end{aligned} \tag{5.2.49}$$

Combine (5.2.48) and (5.2.49),

$$\|\mathcal{P}_D(u^{(j)} - \nabla \hat{J}(u^{(j)})) - u^{(j)}\| \leq \max\left(1, \frac{1}{\underline{\alpha}_{\text{conv.p.}}}\right) \|\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}\| \tag{5.2.50}$$

By (5.2.47) and (5.2.50),

$$\lim_{j \rightarrow +\infty} \|\mathcal{P}_D(u^{(j)} - \nabla \hat{J}(u^{(j)})) - u^{(j)}\| = 0 \tag{5.2.51}$$

This is (5.2.25).

□

### 5.2.3 Bound Iterate Around Optimal Control

In Theorems and lemmas in Section 5.3.1 and Section 5.2.2, it is assumed that control stays in a control set  $U$  so that various boundedness and Lipschitz continuity holds. Next, I show that when the reduced control space objective function  $\hat{J}(u)$  is convex in a region around the optimal solution, there is an upper bound of step size that guarantees the newly generated controls stays in this region given the previous control iterates are in this region so that the assumption of controls in  $U$  can be dropped. To do this, I will use the following Lemma 5.2.4 which holds by triangle inequality on a sphere.

**Lemma 5.2.4** *Given three vectors  $v_1, v_2, v_3$  in an inner product space, let  $\theta_{ij}, 1 \leq i < j \leq 3$ , be the angle between  $v_i$  and  $v_j$  induced by the inner product. If  $\theta_{12}, \theta_{23} \geq 0$  and  $\theta_{12} + \theta_{23} \leq \pi/2$ , then*

$$|\theta_{12} - \theta_{23}| \leq \theta_{13} \leq \theta_{12} + \theta_{23}$$

Given control  $u \in \mathbb{R}^{n_u \times K}$  and  $R > 0$ , since the  $\mathbb{R}^{n_u \times K}$  space is equipped with Frobenius norm and  $\mathbb{R}^{n_u}$  is equipped with 2-norm,

$$\bar{B}(u, R) \subset \overbrace{\bar{B}(u_0, R) \times \bar{B}(u_1, R) \times \cdots \times \bar{B}(u_{K-1}, R)}^{\text{Cartesian product of } K \text{ terms}}.$$

Define set

$$Y(u, R) \stackrel{\text{def}}{=} Y(\bar{B}(u_0, R) \times \bar{B}(u_1, R) \times \cdots \times \bar{B}(u_{K-1}, R)) \quad (5.2.52)$$

with the RHS defined in (5.2.2). If  $F_k \in C(Y_k(u, R) \times \bar{B}(u_k, R))$  for all  $k$ , then  $Y_k(u, R)$  is compact for all  $k$  and thus  $Y(u, R)$  as the Cartesian product (5.2.2b) is compact.

**Theorem 5.2.5** *Let  $u^*$  be a local minimum of the constrained problem, if*

- *exists  $\sigma_{min}, \sigma_{max}, R > 0$  such that, for  $u \in \bar{B}(u^*, R)$ ,*

$$\sigma_{min}I \prec \nabla^2 \hat{J}(u) \prec \sigma_{max}I.$$

- *$J_k, F_k \in C^2(\overline{Y_k(u^*, R) \times B(u_k^*, R)})$  for all time  $k$   
(  $Y_k(u^*, R)$  defined in (5.2.52). Differentiability in a non-open set, refer to explanation on Page 123 )*

*When the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), exists  $\bar{\alpha}_{u.b.} > 0$  such that, for a certain  $i \geq 4(N-1)$ , if*

- *$0 \leq \alpha_j \leq \bar{\alpha}_{u.b.}$  for all  $i - 2(N-1) \leq j \leq i$*
- *$u^{(j)} \in \bar{B}(u^*, R) \cap D$  for all  $i - 4(N-1) \leq j \leq i$*

*then,*

$$u^{(i+1)} \in \bar{B}(u^*, R) \cap D$$

Before the proof, note that,

- Two sets  $B(u^*, R) \subset \mathbb{R}^{n_u \times K}$  and  $B(u_k^*, R) \subset \mathbb{R}^{n_u}$  are in different spaces and

$$u \in \bar{B}(u^*, R) \Rightarrow u_k \in \bar{B}(u_k^*, R) \text{ for } k = 0, \dots, K - 1.$$

This implication relationship ensures the validity of the recursive argument that yields Corollary 5.2.6.

- $\bar{\alpha}_{\text{u.b.}}$  is named so because it is a threshold related to bounding iterates  $u^{(i)}$  in a bounded set.
- similar to Lemma 5.2.1, the conditions on the history of past iterations can be conveniently satisfied by using several  $\alpha = 0$  steps.

**Proof:** At the beginning of the proof, I state an implication of the assumptions,

- To use Lemma 5.2.1, let

$$U_k \stackrel{\text{def}}{=} \bar{B}(u_k^*, R)$$

and  $U = U_0 \times \dots \times U_{K-1}$ . Then, for all  $k$ ,  $U_k \times Y_k(U) \subset \overline{Y_k(u^*, R) \times B(u_k^*, R)}$ . By the assumption of  $J_k, F_k \in C^2(\overline{Y_k(u^*, R) \times B(u_k^*, R)})$ , the second order derivatives of  $J_k, F_k$  is bounded in  $\overline{Y_k(u^*, R) \times B(u_k^*, R)}$  and therefore also bounded in  $U_k \times Y_k(U)$ , which implies Jacobian  $\nabla F_k$  and gradient  $\nabla J_k$  are Lipschitz and bounded for  $u_k \in U_k$  and  $y_k \in Y_k(U)$ . Then, assumptions in Lemma 5.2.1 hold. Let constants  $\alpha_{\text{rel.gs}}, G_U, \bar{M}_U$  be given by Lemma 5.2.1, if

$$\bar{\alpha}_{\text{u.b.}} \leq \bar{\alpha}_{\text{rel.gs}}. \tag{5.2.53}$$

then,

$$\|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \leq \bar{\alpha}_{\text{u.b.}} \bar{M}_U \tag{5.2.54}$$

This proof has two parts. Pick any  $0 < r < R$ .

- In the first part, I prove if

$$u^{(i)} \in (\bar{B}(u^*, R) \setminus B(u^*, r)) \cap D$$

then  $\|u^{(i+1)} - u^*\| \leq \|u^{(i)} - u^*\|$  and thus

$$u^{(i+1)} \in \bar{B}(u^*, R) \cap D$$

See Figure 5.4.

- In the second part, I prove if

$$u^{(i)} \in B(u^*, r) \cap D$$

then

$$u^{(i+1)} \in \bar{B}(u^*, R) \cap D$$

See Figure 5.5.

Combination of these two parts yields the claim of this theorem.

**First part of the proof.** I will show if  $u^{(i)} \in (\bar{B}(u^*, R) \setminus B(u^*, r)) \cap D$ , then  $\|u^{(i+1)} - u^*\| \leq \|u^{(i)} - u^*\|$  with small step size. I first show that the negative gradient direction,  $-\nabla \hat{J}(u^{(i)})$ , form an acute angle with the direction,  $u^* - u^{(i)}$ , to the optimal control  $u^*$ ,

$$\begin{aligned} & (u^* - u^{(i)})^T [-\nabla \hat{J}(u^{(i)})] \\ &= (u^* - u^{(i)})^T [-\nabla \hat{J}(u^*) - \int_0^1 \nabla^2 \hat{J}(u^* + s(u^{(i)} - u^*)) (u^{(i)} - u^*) ds] \\ &= (u^* - u^{(i)})^T [-\nabla \hat{J}(u^*)] + (u^* - u^{(i)})^T [-\int_0^1 \nabla^2 \hat{J}(u^* + s(u^{(i)} - u^*)) (u^{(i)} - u^*) ds] \end{aligned} \tag{5.2.55}$$

Since  $u^*$  is a stationary point of the standard gradient projection iteration, i.e.,

$$\mathcal{P}_D(u^* - \nabla \hat{J}(u^*)) = u^*,$$

by Lemma 5.1.1, the first term in RHS of (5.2.55),  $(u^* - u^{(i)})^T [-\nabla \hat{J}(u^*)] \geq 0$ . Then,

$$\begin{aligned} (u^* - u^{(i)})^T [-\nabla \hat{J}(u^{(i)})] &\geq (u^* - u^{(i)})^T [-\int_0^1 \nabla^2 \hat{J}(u^* + s(u^{(i)} - u^*)) (u^{(i)} - u^*) ds] \\ &\geq \sigma_{\min} \|u^* - u^{(i)}\|^2 \end{aligned} \tag{5.2.56}$$

Now, construct an upper bound of  $\|\nabla\hat{J}(u^{(i)})\|$  using  $\|u^* - u^{(i)}\|$ . Upper bound of the Hessian singular values lead to Lipschitz continuity,

$$\begin{aligned} \|u^* - u^{(i)}\| &\geq \int_0^1 \frac{\|\nabla^2\hat{J}(u^* + s(u^{(i)} - u^*))\|}{\sigma_{\max}} \|(u^{(i)} - u^*)\| ds \\ &\geq \frac{1}{\sigma_{\max}} \left\| \int_0^1 \nabla^2\hat{J}(u^* + s(u^{(i)} - u^*)) (u^{(i)} - u^*) ds \right\| \\ &= \frac{1}{\sigma_{\max}} \|\nabla\hat{J}(u^*) - \nabla\hat{J}(u^{(i)})\| \end{aligned} \quad (5.2.57)$$

By the assumption that  $u^{(i)} \in \bar{B}(u^*, R) \setminus B(u^*, r)$ ,

$$\|u^* - u^{(i)}\| > r = \frac{r}{\|\nabla\hat{J}(u^*)\|} \|\nabla\hat{J}(u^*)\| \quad (5.2.58)$$

For any  $t \in [0, 1]$ , by (5.2.57) and (5.2.58),

$$\begin{aligned} \|u^* - u^{(i)}\| &= t\|u^* - u^{(i)}\| + (1-t)\|u^* - u^{(i)}\| \\ &\geq \frac{t}{\sigma_{\max}} \|\nabla\hat{J}(u^*) - \nabla\hat{J}(u^{(i)})\| + \frac{r(1-t)}{\|\nabla\hat{J}(u^*)\|} \|\nabla\hat{J}(u^*)\| \end{aligned} \quad (5.2.59)$$

Equate two coefficients  $\frac{t}{\sigma_{\max}}$  and  $\frac{r(1-t)}{\|\nabla\hat{J}(u^*)\|}$  by choosing  $t = \sigma_{\max}/(\frac{\|\nabla\hat{J}(u^*)\|}{r} + \sigma_{\max})$ ,

$$\begin{aligned} \|u^* - u^{(i)}\| &\geq \frac{1}{\frac{\|\nabla\hat{J}(u^*)\|}{r} + \sigma_{\max}} (\|\nabla\hat{J}(u^*) - \nabla\hat{J}(u^{(i)})\| + \|\nabla\hat{J}(u^*)\|) \\ &\geq \frac{1}{\frac{\|\nabla\hat{J}(u^*)\|}{r} + \sigma_{\max}} \|\nabla\hat{J}(u^{(i)})\| \end{aligned} \quad (5.2.60)$$

Combine (5.2.60) and (5.2.56),

$$\begin{aligned} (u^* - u^{(i)})^T [-\nabla\hat{J}(u^{(i)})] &\geq \sigma_{\min} \|u^* - u^{(i)}\|^2 \\ &\geq \frac{\sigma_{\min}}{\frac{\|\nabla\hat{J}(u^*)\|}{r} + \sigma_{\max}} \|u^* - u^{(i)}\| \|\nabla\hat{J}(u^{(i)})\| \end{aligned} \quad (5.2.61)$$

Note that in the case where  $\|\nabla\hat{J}(u^*)\| = 0$ , as in the unconstrained case,  $\sigma_{\min}/(\frac{\|\nabla\hat{J}(u^*)\|}{r} + \sigma_{\max})$  in (5.2.61) reduces to the reciprocal of an upper bound of Hessian condition number in  $(\bar{B}(u^*, R) \setminus B(u^*, r)) \cap D$ . Define

$$\tilde{\kappa} \stackrel{\text{def}}{=} \frac{\frac{\|\nabla\hat{J}(u^*)\|}{r} + \sigma_{\max}}{\sigma_{\min}}.$$

Let  $\tilde{\theta}_u$  be the angle between  $u^* - u^{(i)}$  and  $-\nabla \hat{J}(u^{(i)})$ ,

$$\cos(\tilde{\theta}_u) \geq \frac{1}{\tilde{\kappa}} \quad (5.2.62)$$

For all  $u \in (\bar{B}(u^*, R) \setminus B(u^*, r)) \cap D$ ,  $\|\nabla \hat{J}(u)\| > 0$ , otherwise  $u$  is the unique solution of the optimization problem with a convex objective function with a convex constraint. Since  $\|\nabla \hat{J}(u)\|$  is continuous and  $\bar{B}(u^*, R) \setminus B(u^*, r)$  is compact, there exist  $l_{\|\nabla \hat{J}\|}$  such that

$$\|\nabla \hat{J}(u)\| \geq l_{\|\nabla \hat{J}\|}, \quad \forall u \in (\bar{B}(u^*, R) \setminus B(u^*, r)) \cap D \quad (5.2.63)$$

By (5.2.54), when

$$\bar{\alpha}_{\text{u.b.}} \leq \frac{l_{\|\nabla \hat{J}\|}}{2\bar{M}_U}, \quad (5.2.64)$$

then

$$\|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \leq \bar{\alpha}_{\text{u.b.}} \bar{M}_U \leq \frac{l_{\|\nabla \hat{J}\|}}{2\bar{M}_U} \bar{M}_U = \frac{l_{\|\nabla \hat{J}\|}}{2}$$

The following holds,

$$\|g^{(i)}\| \geq \|\nabla \hat{J}(u^{(i)})\| - \|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \geq l_{\|\nabla \hat{J}\|} - \frac{l_{\|\nabla \hat{J}\|}}{2} = \frac{l_{\|\nabla \hat{J}\|}}{2} \quad (5.2.65)$$

Again, by (5.2.54),

$$\|g^{(i)} - \nabla \hat{J}(u^{(i)})\| \leq \bar{\alpha}_{\text{u.b.}} \bar{M}_U \quad (5.2.66)$$

Taking the square of (5.2.66) leads to

$$2(\nabla \hat{J}(u^{(i)}), g^{(i)}) \geq \|\nabla \hat{J}(u^{(i)})\|^2 + \|g^{(i)}\|^2 - \bar{\alpha}_{\text{u.b.}}^2 \bar{M}^2 \quad (5.2.67)$$

Let  $\theta_d$  be the angle between  $\nabla \hat{J}(u^{(i)})$  and  $g^{(i)}$ . By (5.2.67)(5.2.63)(5.2.65),

$$\begin{aligned} \cos(\theta_d) &= \frac{(\nabla \hat{J}(u^{(i)}), g^{(i)})}{\|\nabla \hat{J}(u^{(i)})\| \|g^{(i)}\|} \\ &\geq \frac{\|\nabla \hat{J}(u^{(i)})\|^2 + \|g^{(i)}\|^2 - \bar{\alpha}_{\text{u.b.}}^2 \bar{M}^2}{2\|\nabla \hat{J}(u^{(i)})\| \|g^{(i)}\|} \\ &= \frac{\|\nabla \hat{J}(u^{(i)})\|^2 + \|g^{(i)}\|^2}{2\|\nabla \hat{J}(u^{(i)})\| \|g^{(i)}\|} - \frac{\bar{\alpha}_{\text{u.b.}}^2 \bar{M}^2}{2\|\nabla \hat{J}(u^{(i)})\| \|g^{(i)}\|} \\ &\geq 1 - \frac{\bar{\alpha}_{\text{u.b.}}^2 \bar{M}^2}{l_{\|\nabla \hat{J}\|}^2} \end{aligned} \quad (5.2.68)$$

Note that  $\theta_d$  is also the angle between  $-\nabla \hat{J}(u^{(i)})$  and  $-g^{(i)}$ . Let  $\theta_u$  be the angle between  $u^* - u^{(i)}$  and  $-g^{(i)}$ . Since

$$\lim_{\bar{\alpha}_{\text{u.b.}} \rightarrow 0} \cos(\theta_d(\bar{\alpha}_{\text{u.b.}})) = 1 \text{ and } \lim_{\bar{\alpha}_{\text{u.b.}} \rightarrow 0} \sin(\theta_d(\bar{\alpha}_{\text{u.b.}})) = 0,$$

exists  $\bar{\alpha}_{\text{angle}} > 0$ , such that for

$$0 < \bar{\alpha}_{\text{u.b.}} < \bar{\alpha}_{\text{angle}}, \quad (5.2.69)$$

by Lemma 5.2.4,

$$\begin{aligned} \cos(\theta_u) &\geq \cos(\tilde{\theta}_u + \theta_d) = \cos(\tilde{\theta}_u) \cos(\theta_d) - \sin(\tilde{\theta}_u) \sin(\theta_d) \\ &\geq \frac{1}{\tilde{\kappa}} \cos(\theta_d) - \sqrt{1 - \frac{1}{\tilde{\kappa}^2}} \sin(\theta_d) \geq \frac{1}{2\tilde{\kappa}} \end{aligned} \quad (5.2.70)$$

Using this inequality, a relation on the distance to optimal control change of a parallel-in-time gradient-type method iteration before projection is derived,

$$\begin{aligned} &\|u^{(i)} - u^*\|^2 - \|(u^{(i)} - \alpha g^{(i)}) - u^*\|^2 \\ &= -\alpha_i^2 \|g^{(i)}\|^2 + 2\alpha_i (u^{(i)} - u^*)^T g^{(i)} \\ &= \alpha_i \|g^{(i)}\|^2 \left( \frac{2(u^{(i)} - u^*)^T g^{(i)}}{\|g^{(i)}\|^2} - \alpha_i \right) \\ &= \alpha_i \|g^{(i)}\|^2 \left( \frac{2\|u^{(i)} - u^*\|}{\|g^{(i)}\|} \frac{(u^{(i)} - u^*)^T g^{(i)}}{\|u^{(i)} - u^*\| \|g^{(i)}\|} - \alpha_i \right) \\ &= \alpha_i \|g^{(i)}\|^2 \left( \frac{2\|u^{(i)} - u^*\|}{\|g^{(i)}\|} \cos(\theta_u) - \alpha_i \right) \\ &\geq \alpha_i \|g^{(i)}\|^2 \left( \frac{r}{G_U \tilde{\kappa}} - \alpha_i \right) \end{aligned} \quad (5.2.71)$$

Recall that the constant  $G_U$  above is an upper bound of  $\|g^{(i)}\|$  given by Lemma 5.2.1.

Inequality (5.2.71) implies that when

$$\bar{\alpha}_{\text{u.b.}} \leq \frac{r}{G_U \tilde{\kappa}} \quad (5.2.72)$$

the relation  $\|(u^{(i)} - \alpha g^{(i)}) - u^*\| \leq \|u^{(i)} - u^*\|$  holds. Therefore,

$$\begin{aligned} \|u^{(i+1)} - u^*\| &= \|\mathcal{P}_D(u^{(i)} - \alpha g^{(i)}) - u^*\| = \|\mathcal{P}_D(u^{(i)} - \alpha g^{(i)}) - \mathcal{P}_D(u^*)\| \\ &\leq \|(u^{(i)} - \alpha g^{(i)}) - u^*\| \leq \|u^{(i)} - u^*\| \end{aligned}$$



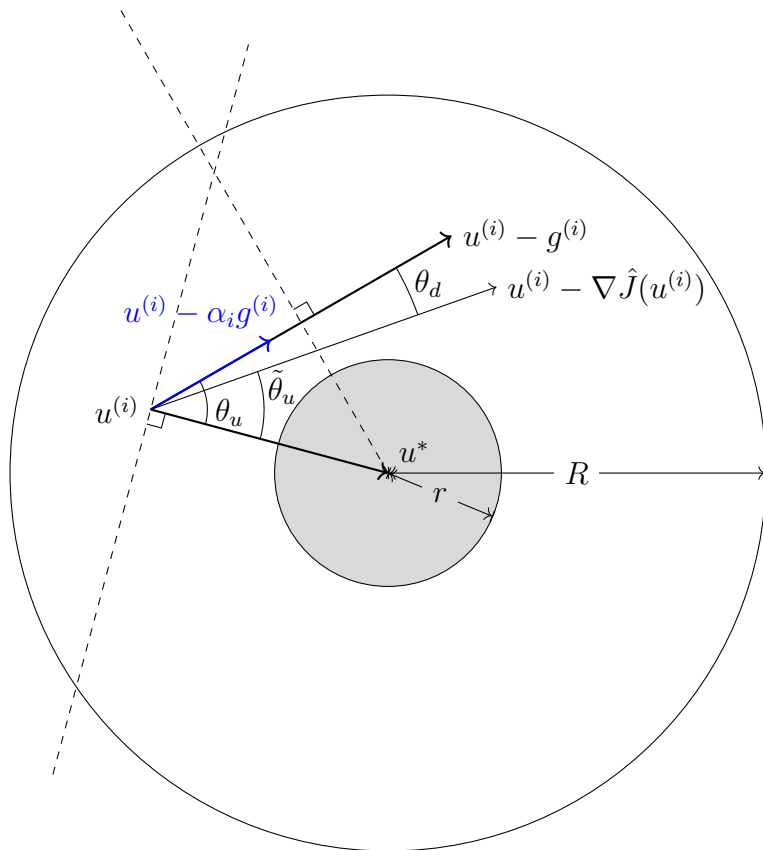


Figure 5.4: Illustration of the first half of the Theorem 5.2.5 proof for the unconstrained case, i.e.,  $D$  is the whole space and the projection  $\mathcal{P}_D$  is trivial. Note that this is a two dimensional illustration of an arbitrary dimensional problem.  $g^{(i)}$ ,  $\nabla \hat{J}(u^{(i)})$ , and  $u^* - u^{(i)}$  do not necessarily lie in the same two dimensional plane. It is typically not true that  $|\theta_u - \tilde{\theta}_u| = \theta_d$ .

which means the updated control is closer to the optimal solution. By this, I conclude that in the case where  $\bar{\alpha}_{u,b}$  satisfies the bounds in (5.2.53), (5.2.64), (5.2.69), and (5.2.72),

$$u^{(i)} \in (\bar{B}(u^*, R) \setminus B(u^*, r)) \cap D \Rightarrow u^{(i+1)} \in \bar{B}(u^*, R) \cap D = U \quad (5.2.73)$$

See Figure 5.4 for an illustration of the first part of the proof.

**Second half of the proof** for the case  $u^{(i)} \in B(u^*, r) \cap D$ . Since there is an

upper bound  $G_U$  for  $\|g^{(i)}\|$ , it is easy to see, with

$$\bar{\alpha}_{\text{u.b.}} \leq \frac{R-r}{G_U} \quad (5.2.74)$$

The new control is guaranteed to stay in  $\bar{B}(u^*, R) \cap D$ , since

$$\begin{aligned} \|u^{(i+1)} - u^*\| &= \|\mathcal{P}_D(u^{(i)} - \alpha_i g^{(i)}) - u^*\| = \|\mathcal{P}_D(u^{(i)} - \alpha_i g^{(i)}) - \mathcal{P}_D(u^*)\| \\ &\leq \|u^{(i)} - \alpha_i g^{(i)} - u^*\| \leq \|u^{(i)} - u^*\| + \alpha_i \|g^{(i)}\| \\ &\leq r + (R-r) \frac{\|g^{(i)}\|}{G_U} \leq R \end{aligned}$$

I conclude that, in the case where  $\bar{\alpha}_{\text{u.b.}}$  satisfies the bounds in (5.2.53), (5.2.74),

$$u^{(i)} \in B(u^*, r) \cap D \Rightarrow u^{(i+1)} \in \bar{B}(u^*, R) \cap D = U \quad (5.2.75)$$

See Figure 5.5 for an illustration of the second part of the proof.

Finally, Combining the results above, by (5.2.73) and (5.2.75), if  $\bar{\alpha}_{\text{u.b.}}$  sufficiently small, i.e., it satisfies the bounds in (5.2.53), (5.2.64), (5.2.69), (5.2.72), and (5.2.74),

$$u^{(i+1)} \in \bar{B}(u^*, R) \cap D = U$$

□

Theorem 5.2.5 guarantees that under certain condition  $u^{(i+1)}$  stays in a fixed region around optimal solution given  $u^{(i)}$  is in the region. Recursive application of Theorem 5.2.5 results in Corollary 5.2.6.

**Corollary 5.2.6** *Let  $u^*$  be a local minimum of the constrained problem, if*

- *exists  $\sigma_{\min}, \sigma_{\max}, R > 0$  such that, for  $u \in \bar{B}(u^*, R)$ ,*

$$\sigma_{\min} I \prec \nabla^2 \hat{J}(u) \prec \sigma_{\max} I.$$

- *$J_k, F_k \in C^2(\overline{Y_k(u^*, R)} \times B(u_k^*, R))$  for all time  $k$   
(  $Y_k(u^*, R)$  defined in (5.2.52). Differentiability in a non-open set, refer to explanation on Page 123 )*

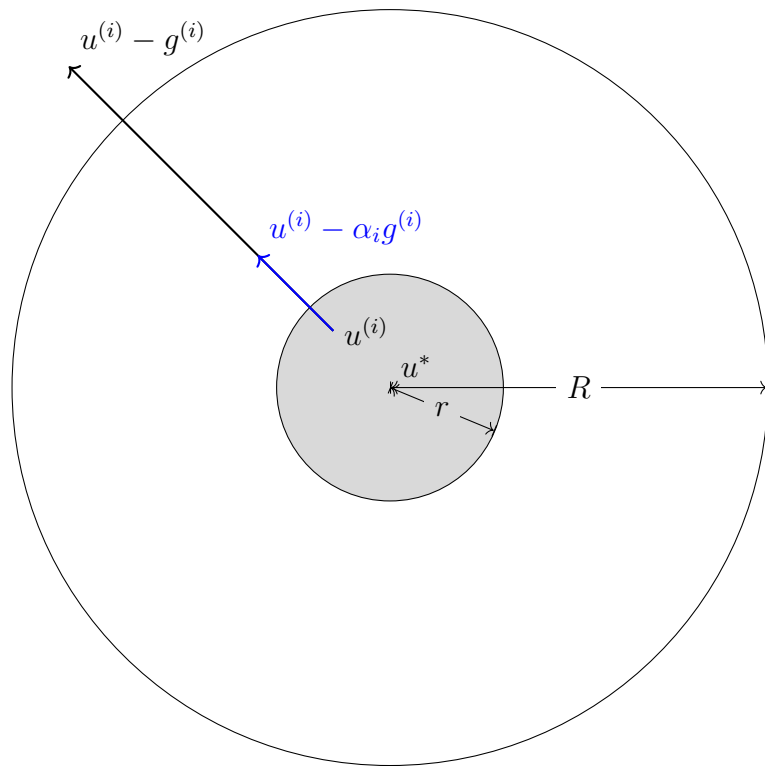


Figure 5.5: Illustration of the second half of the Theorem 5.2.5 proof for the unconstrained case, i.e.,  $D$  is the whole space and the projection  $\mathcal{P}_D$  is trivial.

When the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), there is a step size  $\bar{\alpha}_{u.b.}$  such that, if for a certain index  $i \geq 4(N-1)$  and a certain index  $i^\#$

- $0 \leq \alpha_j \leq \bar{\alpha}_{u.b.}$  for all  $i - 2(N-1) \leq j < i^\#$
- $u^{(j)} \in \bar{B}(u^*, R) \cap D$  for all  $i - 4(N-1) \leq j \leq i$

then,

$$u^{(j)} \in \bar{B}(u^*, R) \cap D \quad \text{for all } i < j \leq i^\#$$

## 5.2.4 Convergence Theorems

In this section, results from Section 5.2.2 and Section 5.2.3 are combined to build convergence theorems.

With the assumptions of Theorem 5.2.3, extra conditions of uniqueness of stationary point of classic gradient projection methods and step size being bounded away from zero yields the following convergence theorem.

**Proposition 5.2.7** *In problem (5.2.1), if there exists compact sets*

$$U_0, U_1, \dots, U_{K-1} \subset \mathbb{R}^{n_u}$$

and  $U \stackrel{\text{def}}{=} U_0 \times U_1 \times \dots \times U_{K-1}$ , such that,

- $J_k, F_k \in C^2(\overline{Y_k(U)} \times U_k)$  for all time step  $k$   
( Differentiability in a non-open set, refer to explanation on Page 123 )
- there is a unique  $u^* \in U$  such that

$$\mathcal{P}_D(u^* - \nabla \hat{J}(u^*)) = u^*$$

When the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), exists  $\bar{\alpha}_{conv.c.} > 0$  such that, for any  $0 < \underline{\alpha}_{conv.c.} \leq \bar{\alpha}_{conv.c.}$  if for a certain iteration index  $i$ ,

- the step sizes satisfies

$$\begin{aligned} 0 \leq \alpha_j \leq \bar{\alpha}_{\text{conv.c.}} & \quad \text{for } i - 2(N - 1) \leq j < i \\ \underline{\alpha}_{\text{conv.c.}} \leq \alpha_j \leq \bar{\alpha}_{\text{conv.c.}} & \quad \text{for } j \geq i \end{aligned}$$

- $u^{(j)} \in U$ , for  $j \geq i - 4(N - 1)$

then

$$\lim_{j \rightarrow +\infty} u^{(j)} = u^*$$

**Proof:** By the continuity of  $F_k, J_k$  of all  $k$ ,  $\hat{J}$  is continuous with  $u \in U$ . Using the compactness of  $U$ ,

- $\hat{J}(u)$  is bounded from below for  $u \in U$ .

By the compactness of  $\overline{Y_k(U) \times U_k}$  and second order continuous differentiability of  $J_k, F_k$  for all proper time step index  $k$ ,

- when  $u_k \in U_k$  and  $y_k \in Y_k(U)$ ,  $\nabla J_k$  and  $\nabla F_k$  is Lipschitz and bounded for all  $k$

The assumptions of Theorem 5.2.3 are satisfied. Let  $\bar{\alpha}_{\text{conv.p.}}$  be given by Theorem 5.2.3, require in the proof below

$$\bar{\alpha}_{\text{conv.c.}} \leq \bar{\alpha}_{\text{conv.p.}} \tag{5.2.76}$$

Define function

$$S(u; \alpha) = \|\mathcal{P}_D(u - \alpha \nabla \hat{J}(u)) - u\| \quad u \in U, \alpha \in \mathbb{R} \tag{5.2.77}$$

By the differentiability assumptions, with any scalar  $\alpha$ , the non-negative  $S(u; \alpha)$  is continuous with respect to  $u$ . For any  $\epsilon > 0$ , by the assumption of the unique stationary point

$$S(u; 1) > 0 \quad u \in U \setminus B(u^*, \epsilon) \tag{5.2.78}$$

by Lemma 5.1.6,

$$S(u; \underline{\alpha}_{\text{conv.c.}}) > 0 \quad u \in U \setminus B(u^*, \epsilon) \tag{5.2.79}$$

Due to the continuity of  $S(u; \underline{\alpha}_{\text{conv.c.}})$  with respect to  $u$  and the compactness of  $U \setminus B(u^*, \epsilon)$ , there exists  $\delta_\epsilon > 0$  such that

$$S(u; \underline{\alpha}_{\text{conv.c.}}) > \delta_\epsilon \quad u \in U \setminus B(u^*, \epsilon) \quad (5.2.80)$$

For any  $j \geq i$ , the step size assumption requires  $\alpha_j \geq \underline{\alpha}_{\text{conv.c.}}$ , by Lemma 5.1.3,

$$S(u; \alpha_j) \geq S(u; \underline{\alpha}_{\text{conv.c.}}) > \delta_\epsilon \quad u \in U \setminus B(u^*, \epsilon) \quad (5.2.81)$$

By Theorem 5.2.3, when the step size condition (5.2.76) holds,

$$\lim_{j \rightarrow +\infty} S(u^{(j)}; \alpha_j) = 0 \quad (5.2.82)$$

Then, when the step size threshold  $\bar{\alpha}_{\text{conv.c.}}$  satisfies (5.2.76), by the fact that  $\epsilon > 0$  in (5.2.81) being arbitrary and (5.2.82),

$$\lim_{j \rightarrow +\infty} u^{(j)} = u^*$$

Otherwise, there exists  $\epsilon > 0$  such that for any  $j' > 0$ , there is  $j > j'$  such that

$$\|u^{(j)} - u^*\| > \epsilon$$

which, by (5.2.81), leads to

$$S(u^{(j)}; \alpha_j) > \delta_\epsilon$$

contradicting with (5.2.82). This concludes the proof.  $\square$

The condition in “ $u^{(j)} \in U$ , for  $j \geq i - 4(N - 1)$ ” in Proposition 5.2.7 is not straightforward to check. The following Theorem 5.2.8 and Theorem 5.2.9 replaces this condition by compactness of  $D$  and convexity of the reduced control space objective function respectively.

In the case that the control variables are projected to a compact set, Proposition 5.2.7 leads to the dropping of condition “ $u^{(j)} \in U$ , for  $j \geq i - 4(N - 1)$ ” without additional assumption of convexity.

**Theorem 5.2.8** *In problem (5.2.1), if  $D = D_0 \times D_1 \times \cdots \times D_{K-1}$  with  $D_k$  compact for all  $k$  and*

- $J_k, F_k \in C^2(\overline{Y_k(D)} \times D_k)$  for all time step  $k$   
( Differentiability in a non-open set, refer to explanation on Page 123 )
- there is a unique  $u^* \in D$  such that

$$\mathcal{P}_D(u^* - \nabla \hat{J}(u^*)) = u^*$$

When the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), exists  $\bar{\alpha}_{conv.c.} > 0$  such that, for any  $0 < \underline{\alpha}_{conv.c.} \leq \bar{\alpha}_{conv.c.}$  if for a certain iteration index  $i$ ,

- the step sizes satisfies

$$\begin{aligned} 0 \leq \alpha_j \leq \bar{\alpha}_{conv.c.} & \quad \text{for } i - 2(N - 1) \leq j < i \\ \underline{\alpha}_{conv.c.} \leq \alpha_j \leq \bar{\alpha}_{conv.c.} & \quad \text{for } j \geq i \end{aligned}$$

- $u^{(0)} \in D$

then

$$\lim_{j \rightarrow +\infty} u^{(j)} = u^*$$

**Proof:** Define  $U_k \stackrel{\text{def}}{=} D_k$  for all  $k$  and  $U \stackrel{\text{def}}{=} U_1 \times \dots \times U_{K-1}$ . The projected step in every iteration guarantees that all control iterates are in  $U$ , i.e.,

$$u^{(j+1)} = \mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) \in U = D, \quad j = 0, 1, \dots$$

Application of Proposition 5.2.7 completes the proof □

Adding Additional convexity assumptions to Proposition 5.2.7 enables Corollary 5.2.6 and yields the following convergence theorem.

**Theorem 5.2.9** *Let  $u^*$  be a local minimum of the constrained problem, if*

- *exists  $\sigma_{min}, \sigma_{max}, R > 0$  such that, for  $u \in \bar{B}(u^*, R)$ ,*

$$\sigma_{min}I \prec \nabla^2 \hat{J}(u) \prec \sigma_{max}I.$$

- *$J_k, F_k \in C^2(\overline{Y_k(u^*, R) \times B(u_k^*, R)})$  for all time  $k$   
(  $Y_k(u^*, R)$  defined in (5.2.52). Differentiability in a non-open set, refer to explanation on Page 123 )*

*When the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), exists  $\bar{\alpha}_{conv.} > 0$  such that, for any  $0 < \underline{\alpha}_{conv.} \leq \bar{\alpha}_{conv.}$ , if for a certain iteration index  $i$ ,*

- *the step sizes satisfies*

$$\begin{aligned} 0 \leq \alpha_j \leq \bar{\alpha}_{conv.} & \quad \text{for } i - 2(N - 1) \leq j < i \\ \underline{\alpha}_{conv.} \leq \alpha_j \leq \bar{\alpha}_{conv.} & \quad \text{for } j \geq i \end{aligned}$$

- *$u^{(j)} \in \bar{B}(u^*, R) \cap D$ , for  $i - 4(N - 1) \leq j \leq i$*

*then*

$$\lim_{j \rightarrow +\infty} u^{(j)} = u^*$$

**Proof:** Let

$$U_k \stackrel{\text{def}}{=} \bar{B}(u_k^*, R), k = 0, \dots, K - 1 \text{ and } U \stackrel{\text{def}}{=} U_0 \times \dots \times U_{K-1}$$

Let  $\bar{\alpha}_{u.b.}$  be given by Corollary 5.2.6. If the step size condition

$$\bar{\alpha}_{conv.} \leq \bar{\alpha}_{u.b.} \tag{5.2.83}$$

holds, by Corollary 5.2.6,

$$u^{(j)} \in U \quad \text{for } j > i$$



By first order necessary condition of optimality

$$\mathcal{P}_D(u^* - \nabla \hat{J}(u^*)) = u^*$$

For any  $u \in U \setminus \{u^*\}$ ,

$$\nabla \hat{J}(u) + \int_0^1 \nabla^2 \hat{J}(u + s(u^* - u))(u^* - u) ds = \nabla \hat{J}(u^*) \quad (5.2.84)$$

left multiply  $(u^* - u)^T$  on both sides, by the optimality of  $u^*$ ,

$$(u^* - u)^T \left[ \nabla \hat{J}(u) + \int_0^1 \nabla^2 \hat{J}(u + s(u^* - u))(u^* - u) ds \right] = (u^* - u)^T \nabla \hat{J}(u^*) \leq 0$$

which yields, with the strong convexity of  $\hat{J}$ ,

$$(u^* - u)^T \nabla \hat{J}(u) \leq - \int_0^1 (u^* - u)^T \nabla^2 \hat{J}(u + s(u^* - u))(u^* - u) ds < 0 \quad (5.2.85)$$

which leads to

$$\mathcal{P}_D(u - \nabla \hat{J}(u)) \neq u \quad \text{for } u \in U \setminus \{u^*\}$$

Otherwise, it contradicts with (5.2.85) according to Lemma 5.1.1. This implies the uniqueness of the point that satisfies the first order necessary condition of optimality.

Let  $\bar{\alpha}_{\text{conv.c}}$  be given by Proposition 5.2.7, require

$$\bar{\alpha}_{\text{conv}} \leq \bar{\alpha}_{\text{conv.c}} \quad (5.2.86)$$

With step size threshold  $\bar{\alpha}_{\text{conv}}$  satisfies (5.2.83) and (5.2.86), By Proposition 5.2.7

$$\lim_{j \rightarrow +\infty} u^{(j)} = u^*$$

which concludes the prove. □

Again, the condition of the theorem

$$" u^{(j)} \in \bar{B}(u^*, R) \cap D, \text{ for } i - 4(N - 1) \leq j \leq i "$$

can be easily satisfied by initializing  $u^{(i-4(N-1))} \in \bar{B}(u^*, R) \cap D$  and setting  $\alpha_j = 0$  for  $i - 4(N - 1) \leq j < i$ . As mentioned before,  $2N - 1$  steps of parallel-in-time

gradient-type iteration with no control update, which is corresponding to  $2(N - 1)$  steps of step size 0 parallel-in-time gradient-type step, compute the exact gradient and further step size 0 steps will not alter the control and shooting variables any more. This observation yields the following corollary as a simple result of Theorem 5.2.9.

**Corollary 5.2.10** *Let  $u^*$  be a local minimum of the constrained problem, if*

- *exists  $\sigma_{min}, \sigma_{max}, R > 0$  such that, for  $u \in \bar{B}(u^*, R)$ ,*

$$\sigma_{min}I \prec \nabla^2 \hat{J}(u) \prec \sigma_{max}I.$$

- *$J_k, F_k \in C^2(\overline{Y_k(u^*, R) \times B(u_k^*, R)})$  for all time  $k$   
(  $Y_k(u^*, R)$  defined in (5.2.52). Differentiability in a non-open set, refer to explanation on Page 123 )*

*When the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), exists  $\bar{\alpha}_{conv.} > 0$  such that, for any  $0 < \underline{\alpha}_{conv.} \leq \bar{\alpha}_{conv.}$ , if*

- *$u^{(0)} \in \bar{B}(u^*, R) \cap D$*
- *the step sizes satisfies*

$$\begin{aligned} \alpha_j &= 0 && \text{for } 0 \leq j < 2(N - 1) \\ \underline{\alpha}_{conv.} &\leq \alpha_j \leq \bar{\alpha}_{conv.} && \text{for } j \geq 2(N - 1) \end{aligned}$$

*then*

$$\lim_{j \rightarrow +\infty} u^{(j)} = u^*$$

## 5.3 As a Perturbed Gradient Method: Iteration-Wise Monotonicity

This section discuss about the possibility of enforcing an iteration-wise difference bound, of type (5.3.2), i.e.,

$$\|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \leq \gamma \|\nabla \hat{J}(u^{(i)})\|$$

between the exact gradient and the parallel-in-time gradient-type vector, as opposed to the bound (5.2.3) involved with many iterations in Lemma 5.2.1. The consequent results of guaranteed monotonic convergence in several senses are presented afterward.

The convergence proofs in the previous sections of this chapter does not provide any information on if the convergence is monotonic, in the sense of objective function value and control error. In the linear-quadratic problem examples, it is usually observed that with an optimal fixed step size, the convergence is monotonic in neither objective function value and control error, see Figure 3.9.

A natural question is: can I guarantee monotonic convergence by enforcing another step size upper bound? The answer is yes, partly.

Now, I give additional theorems stating that apart from the step size upper bound for convergence, there is other step size upper bounds that guarantees monotonic convergence of the parallel-in-time gradient-type method. In some sense, these theorems shows that some monotonicity properties, namely, in objective function value and, for unconstrained problem, in distance to optimal control, of classic gradient method with sufficiently small step size are maintained in the parallel-in-time gradient-type method, although the required step size may be different.

### 5.3.1 Iteration-Wise Gradient-Type Vector Error Bound

In this section, I prove Lemma 5.3.1 stating that with suitable initial condition and small step sizes, an iteration-wise error bound between the gradient-type vector com-

puted in the parallel-in-time gradient-type vector and the true gradient corresponding to control iterate. This lemma will be used to establish the monotonic convergence results.

**Lemma 5.3.1** *If there exists  $U$  such that*

- $\nabla J_k$  and  $\nabla F_k$  is Lipschitz and bounded for  $u_k \in U_k$  and  $y_k \in Y_k(U)$  for all  $k$
- exists  $\sigma_{min}, \sigma_{max} > 0$  such that, for  $u \in U$ ,

$$\sigma_{min}I \prec \nabla^2 \hat{J}(u) \prec \sigma_{max}I.$$

*Then, when the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), for any  $\gamma > 0$ , exists  $\bar{\alpha}_\gamma > 0$ , such that for parallel-in-time gradient-type method with step size less than  $\bar{\alpha}_\gamma$ , if there is a certain index  $i^*$  and a certain index  $i^\#$  such that*

- previous several iterations has iteration-wise relative error bounded by  $\gamma$

$$\|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \leq \gamma \|\nabla \hat{J}(u^{(i)})\|, \quad i = i^* - 2(N - 1), \dots, i^* - 1 \quad (5.3.1)$$

- $0 \leq \alpha_i \leq \bar{\alpha}_\gamma$  for all  $i^* - 2(N - 1) \leq i < i^\#$
- $u^{(i)} \in U$ , for all  $i^* - 4(N - 1) \leq i \leq i^\#$

*then*

$$\|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \leq \gamma \|\nabla \hat{J}(u^{(i)})\|, \quad \forall i^* \leq i \leq i^\# \quad (5.3.2)$$

**Proof:** Prove by mathematical induction.

**(induction base step)** Let  $i = i^* - 1$ . (5.3.1) certifies that

$$\|\nabla \hat{J}(u^{(i-j)}) - g^{(i-j)}\| \leq \gamma \|\nabla \hat{J}(u^{(i-j)})\|, \quad j = 0, \dots, 2(N - 1) - 1 \quad (5.3.3)$$

**(induction step)** Let  $i = k \geq i^*$ , assume that (5.3.3) is true for  $i = k - 1$ , i.e.,

$$\|\nabla \hat{J}(u^{(k-1-j)}) - g^{(k-1-j)}\| \leq \gamma \|\nabla \hat{J}(u^{(k-1-j)})\|, \quad j = 0, \dots, 2(N - 1) - 1 \quad (5.3.4)$$

By triangle inequality, induction assumption (5.3.4) implies

$$\|g^{(k-1-j)}\| \leq (1 + \gamma)\|\nabla\hat{J}(u^{(k-1-j)})\|, \quad j = 0, \dots, 2(N-1) - 1 \quad (5.3.5)$$

The projected parallel-in-time gradient-type method executes the control update

$$u^{(k-j+1)} = \mathcal{P}_D(u^{(k-j)} - \alpha_{k-j}g^{(k-j)}), \quad j = 1, \dots, 2(N-1)$$

and therefore, for  $j = 1, \dots, 2(N-1)$ ,

$$\nabla\hat{J}(u^{(k-j+1)}) = \nabla\hat{J}(u^{(k-j)}) + \int_0^1 \nabla^2\hat{J}(u^{(k-j)} - s(u^{(k-j+1)} - u^{(k-j)}))(u^{(k-j+1)} - u^{(k-j)})ds \quad (5.3.6)$$

Since  $\sigma_{\min} \prec \nabla^2\hat{J} \prec \sigma_{\max}$ ,

$$\begin{aligned} \|\nabla\hat{J}(u^{(k-j+1)})\| &\geq \|\nabla\hat{J}(u^{(k-j)})\| - \sigma_{\max}\|u^{(k-j+1)} - u^{(k-j)}\| \\ &= \|\nabla\hat{J}(u^{(k-j)})\| - \sigma_{\max}\|P_D(u^{(k-j)} - \alpha_{k-j}g^{(k-j)}) - u^{(k-j)}\| \\ &\geq \|\nabla\hat{J}(u^{(k-j)})\| - \sigma_{\max}\alpha_{k-j}\|g^{(k-j)}\| \\ &\geq \|\nabla\hat{J}(u^{(k-j)})\| - \sigma_{\max}\alpha_{k-j}(\gamma + 1)\|\nabla\hat{J}(u^{(k-j)})\| \\ &= [1 - \alpha_{k-j}\sigma_{\max}(\gamma + 1)]\|\nabla\hat{J}(u^{(k-j)})\| \\ &\geq [1 - \bar{\alpha}_\gamma\sigma_{\max}(\gamma + 1)]\|\nabla\hat{J}(u^{(k-j)})\| \end{aligned} \quad (5.3.7)$$

By recursive substitution, for  $j = 1, \dots, 2(N-1)$ ,

$$\|\nabla\hat{J}(u^{(k)})\| \geq [1 - \bar{\alpha}_\gamma\sigma_{\max}(\gamma + 1)]^j\|\nabla\hat{J}(u^{(k-j)})\| \quad (5.3.8)$$

Let  $\bar{\alpha}_{\text{rel,gs}}, M_U$  be given by applying Lemma 5.2.1. Require,

$$\bar{\alpha}_\gamma \leq \bar{\alpha}_{\text{rel,gs}} \quad (5.3.9)$$

by Lemma 5.2.1, (5.3.5), (5.3.8), and the assumption that

$$\bar{\alpha}_\gamma \leq \frac{1}{2\sigma_{\max}(\gamma + 1)} \quad (5.3.10)$$

The following bound holds,

$$\begin{aligned}
\|\nabla \hat{J}(u^{(k)}) - g^{(k)}\| &\leq \bar{\alpha}_\gamma M_U \sum_{j=1}^{2(N-1)} \|g^{(k-j)}\| \\
&\leq \bar{\alpha}_\gamma (1 + \gamma) M_U \sum_{j=1}^{2(N-1)} \|\nabla \hat{J}(u^{(k-j)})\| \\
&\leq \bar{\alpha}_\gamma (1 + \gamma) M_U \sum_{j=1}^{2(N-1)} [1 - \bar{\alpha}_\gamma \sigma_{\max}(\gamma + 1)]^{-j} \|\nabla \hat{J}(u^{(k)})\|
\end{aligned} \tag{5.3.11}$$

When  $\bar{\alpha}_\gamma$  is smaller than a threshold, i.e.,

$$\bar{\alpha}_\gamma \leq \frac{\gamma}{2^{2(N-1)+1}(1 + \gamma)M_U} \tag{5.3.12}$$

assume (5.3.10) holds,

$$\begin{aligned}
\bar{\alpha}_\gamma (1 + \gamma) M_U \sum_{j=1}^{2(N-1)} [1 - \bar{\alpha}_\gamma \sigma_{\max}(\gamma + 1)]^{-j} &\leq \bar{\alpha}_\gamma (1 + \gamma) M_U \sum_{j=1}^{2(N-1)} \left[1 - \frac{1}{2}\right]^{-j} \\
&= \bar{\alpha}_\gamma (1 + \gamma) M_U 2 \frac{1 - 2^{2(N-1)}}{1 - 2} \\
&\leq \bar{\alpha}_\gamma 2^{2(N-1)+1} (1 + \gamma) M_U \\
&\leq \gamma
\end{aligned} \tag{5.3.13}$$

By (5.3.11) and (5.3.13), when the step size threshold  $\bar{\alpha}_\gamma$  satisfies (5.3.9), (5.3.10), and (5.3.12),

$$\|\nabla \hat{J}(u^{(k)}) - g^{(k)}\| \leq \gamma \|\nabla \hat{J}(u^{(k)})\| \tag{5.3.14}$$

This concludes the induction step.  $\square$

### 5.3.2 Unconstrained Problem

In this section, making use of Lemma 5.3.1, I give two monotonic convergence results for unconstrained problem, i.e.  $D$  is the whole control space. Theorem 5.3.2 is for monotonic convergence in objective function value and Theorem 5.3.3 is for monotonic convergence in distance to optimal control.

**Theorem 5.3.2** *If there exists  $U$  such that*

- $u^* \in U$  is a local minimum of the constrained problem
- $\nabla J_k$  and  $\nabla F_k$  is Lipschitz and bounded for  $u_k \in U_k$  and  $y_k \in Y_k(U)$  for all  $k$
- exists  $\sigma_{min}, \sigma_{max} > 0$  such that, for  $u \in U$ ,

$$\sigma_{min}I \prec \nabla^2 \hat{J}(u) \prec \sigma_{max}I.$$

*Then, when the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), exists  $\bar{\alpha}_{mono}$ . and  $\gamma > 0$ , such that if there is a certain index  $i^*$  and a certain index  $i^\#$  such that*

- *previous several iterations has iteration-wise relative error bounded by  $\gamma$*

$$\|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \leq \gamma \|\nabla \hat{J}(u^{(i)})\|, \quad i = i^* - 2(N - 1), \dots, i^* - 1 \quad (5.3.15)$$

- $0 \leq \alpha_i \leq \bar{\alpha}_{mono}$ . for all  $i^* - 2(N - 1) \leq i < i^\#$
- $u^{(i)} \in U$ , for all  $i^* - 4(N - 1) \leq i \leq i^\#$

*then*

$$\hat{J}(u^{(i+1)}) \leq \hat{J}(u^{(i)}) \quad \text{for all } i^* \leq i < i^\#$$

*where the equality holds only when  $\alpha_i = 0$  or  $u^{(i)} = u^*$ .*

**Proof:** In the unconstrained problem, for  $i \geq 0$ ,

$$u^{(i+1)} = u^{(i)} - \alpha_i g^{(i)}$$

First, look at the first order approximation of the difference between the objective function values corresponding to  $u^{(i+1)}$  and  $u^{(i)}$ , for  $i \geq i^*$ ,

$$\begin{aligned}
\nabla \hat{J}(u^{(i)})^T[-\alpha_i g^{(i)}] &= -\alpha_i \nabla \hat{J}(u^{(i)})^T [\nabla \hat{J}(u^{(i)}) + (g^{(i)} - \nabla \hat{J}(u^{(i)}))] \\
&= -\alpha_i \|\nabla \hat{J}(u^{(i)})\|^2 \left[ 1 + \frac{\nabla \hat{J}(u^{(i)})^T (g^{(i)} - \nabla \hat{J}(u^{(i)}))}{\|\nabla \hat{J}(u^{(i)})\|^2} \right] \\
&\leq -\alpha_i \|\nabla \hat{J}(u^{(i)})\|^2 \left[ 1 - \frac{\|\nabla \hat{J}(u^{(i)})\| \|g^{(i)} - \nabla \hat{J}(u^{(i)})\|}{\|\nabla \hat{J}(u^{(i)})\|^2} \right] \\
&= -\alpha_i \|\nabla \hat{J}(u^{(i)})\|^2 \left[ 1 - \frac{\|g^{(i)} - \nabla \hat{J}(u^{(i)})\|}{\|\nabla \hat{J}(u^{(i)})\|} \right]
\end{aligned} \tag{5.3.16}$$

Pick any

$$0 < \gamma < 1 \tag{5.3.17}$$

and let  $\bar{\alpha}_\gamma$  be given by Lemma 5.3.1, when

$$\bar{\alpha}_{\text{mono.}} \leq \bar{\alpha}_\gamma \tag{5.3.18}$$

for  $i \geq i^*$ ,

$$\|\nabla \hat{J}(u^{(i)})\| \leq \gamma \|g^{(i)} - \nabla \hat{J}(u^{(i)})\|$$

The following holds by (5.3.16),

$$\nabla \hat{J}(u^{(i)})^T[-\alpha_i g^{(i)}] \leq -\alpha_i (1 - \gamma) \|\nabla \hat{J}(u^{(i)})\|^2. \tag{5.3.19}$$

Again, by Lemma 5.3.1,

$$\|g^{(i)}\| \leq (1 + \gamma) \|\nabla \hat{J}(u^{(i)})\| \tag{5.3.20}$$

Then, examine the exact relationship between consecutive objective function values, using (5.3.19) and (5.3.20),

$$\begin{aligned}
\hat{J}(u^{(i+1)}) &= \hat{J}(u^{(i)} - \alpha_i g^{(i)}) \\
&\leq \hat{J}(u^{(i)}) + \nabla \hat{J}(u^{(i)})^T[-\alpha_i g^{(i)}] + \frac{1}{2} \sigma_{\max} \|\alpha_i g^{(i)}\|^2 \\
&\leq \hat{J}(u^{(i)}) - \alpha_i (1 - \gamma) \|\nabla \hat{J}(u^{(i)})\|^2 + \frac{\sigma_{\max} \alpha_i^2 (1 + \gamma)^2}{2} \|\nabla \hat{J}(u^{(i)})\|^2 \\
&= \hat{J}(u^{(i)}) - \alpha_i \left[ (1 - \gamma) - \frac{\sigma_{\max} \alpha_i (1 + \gamma)^2}{2} \right] \|\nabla \hat{J}(u^{(i)})\|^2
\end{aligned} \tag{5.3.21}$$



When

$$\alpha_{\text{mono.}} < \frac{2(1 - \gamma)}{\sigma_{\max}(1 + \gamma)^2} \quad (5.3.22)$$

The equality holds

$$\hat{J}(u^{(i+1)}) \leq \hat{J}(u^{(i)}) \quad (5.3.23)$$

In conclusion, for any  $\gamma$  satisfying (5.3.17), when the step size threshold  $\bar{\alpha}_{\text{mono.}}$  satisfies (5.3.18) and (5.3.22), the monotonic decreasing in objective function, i.e. (5.3.23), holds. And the equality holds only when  $\alpha_i = 0$  or  $u^{(i)} = u^*$ .  $\square$

By the same set of conditions as Theorem 5.3.2 requires, a monotonic theorem on the control iterate error compared to the optimal control can be proved.

**Theorem 5.3.3** *If there exists  $U$  such that*

- $u^* \in U$  is a local minimum of the constrained problem
- $\nabla J_k$  and  $\nabla F_k$  is Lipschitz and bounded for  $u_k \in U_k$  and  $y_k \in Y_k(U)$  for all  $k$
- exists  $\sigma_{\min}, \sigma_{\max} > 0$  such that, for  $u \in U$ ,

$$\sigma_{\min}I \prec \nabla^2 \hat{J}(u) \prec \sigma_{\max}I.$$

Then, when the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), exists  $\bar{\alpha}_{\text{mono.}}$  and  $\gamma > 0$ , such that if there is a certain index  $i^*$  and a certain index  $i^\#$  such that

- previous several iterations has iteration-wise relative error bounded by  $\gamma$

$$\|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \leq \gamma \|\nabla \hat{J}(u^{(i)})\|, \quad i = i^* - 2(N - 1), \dots, i^* - 1 \quad (5.3.24)$$

- $0 \leq \alpha_i \leq \bar{\alpha}_{\text{mono.}}$  for all  $i^* - 2(N - 1) \leq i < i^\#$
- $u^{(i)} \in U$ , for all  $i^* - 4(N - 1) \leq i \leq i^\#$

then

$$\|u^{(i+1)} - u^*\| \leq \|u^{(i)} - u^*\|$$

where the equality holds only when  $\alpha_i = 0$  or  $u^{(i)} = u^*$ .

**Proof:** By a similar argument as in the proof of Theorem 5.2.5, let  $\tilde{\theta}_u$  be the angle between  $u^* - u^{(i)}$  and  $-\nabla \hat{J}(u^{(i)})$ , using (5.2.62) with  $\|\nabla \hat{J}(u^*)\| = 0$  in the unconstrained problem and  $\kappa \stackrel{\text{def}}{=} \sigma_{\max}/\sigma_{\min}$ ,

$$\cos(\tilde{\theta}_u) \geq \frac{1}{\kappa} \quad (5.3.25)$$

Pick

$$0 < \gamma < 1, \quad (5.3.26)$$

whose value will be determined later, and let  $\bar{\alpha}_\gamma$  be given by Lemma 5.3.1, then, when

$$\bar{\alpha}_{\text{mono.}} \leq \bar{\alpha}_\gamma \quad (5.3.27)$$

the following holds by Lemma 5.3.1,

$$\|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \leq \gamma \|\nabla \hat{J}(u^{(i)})\| \quad (5.3.28)$$

Square both sides and rearrange terms,

$$\nabla \hat{J}(u^{(i)})^T g^{(i)} \geq \frac{(1 - \gamma^2) \|\nabla \hat{J}(u^{(i)})\|^2 + \|g^{(i)}\|^2}{2}$$

and therefore, with  $\theta_d$  denoting the angle between  $-\nabla \hat{J}(u^{(i)})$  and  $-g^{(i)}$ ,

$$\begin{aligned} \cos(\theta_d) &= \frac{\nabla \hat{J}(u^{(i)})^T g^{(i)}}{\|\nabla \hat{J}(u^{(i)})\| \|g^{(i)}\|} \\ &\geq \frac{(1 - \gamma^2) \|\nabla \hat{J}(u^{(i)})\|^2 + \|g^{(i)}\|^2}{2 \|\nabla \hat{J}(u^{(i)})\| \|g^{(i)}\|} \\ &\geq \frac{(1 - \gamma^2) (\|\nabla \hat{J}(u^{(i)})\|^2 + \|g^{(i)}\|^2)}{2 \|\nabla \hat{J}(u^{(i)})\| \|g^{(i)}\|} \geq 1 - \gamma^2 \end{aligned} \quad (5.3.29)$$

Let  $\theta_u$  be the angle between  $u^* - u^{(i)}$  and  $-g^{(i)}$ . By Lemma 5.2.4,  $\theta_u \leq \tilde{\theta}_u + \theta_d$ , and using (5.3.25) and (5.3.29),

$$\begin{aligned} \cos(\theta_u) &\geq \cos(\theta_u + \theta_d) = \cos(\theta_u) \cos(\theta_d) - \sin(\theta_u) \sin(\theta_d) \\ &\geq \frac{1 - \gamma^2}{\kappa} - \sqrt{1 - \frac{1}{\kappa^2}} \sqrt{1 - (1 - \gamma^2)^2} \end{aligned} \quad (5.3.30)$$

Since the RHS of (5.3.30) converges to  $1/\kappa$  as  $\gamma$  approaches 0, Exists  $\bar{\gamma} > 0$  and  $\underline{c} > 0$  such that if

$$\gamma < \bar{\gamma} \quad (5.3.31)$$

The following holds by (5.3.30),

$$\cos(\theta_u) > \underline{c}$$

Now that it is proved that the  $-g^{(i)}$  direction forms an acute angle  $\theta_u$  with  $u^* - u^{(i)}$ , to show a control update  $-\alpha_i g^{(i)}$  yields a decrease in control error, one only needs to show when  $\alpha$  is less than a threshold, the actual step is not too large, see Figure 5.4. By Lemma 5.3.1,

$$\|g^{(i)}\| \leq (1 + \gamma) \|\nabla \hat{J}(u^{(i)})\| \leq (1 + \gamma) \sigma_{\max} \|u - u^*\| \quad (5.3.32)$$

When

$$\bar{\alpha}_{\text{mono.}} < \frac{2\underline{c}}{(1 + \gamma)\sigma_{\max}} \quad (5.3.33)$$

the following holds, using (5.3.30) and (5.3.32),

$$\begin{aligned} \|u^{(i+1)} - u^*\|^2 - \|u^{(i)} - u^*\|^2 &= \|u^{(i)} - \alpha_i g^{(i)} - u^*\|^2 - \|u^{(i)} - u^*\|^2 \\ &= \alpha_i^2 \|g^{(i)}\|^2 - 2\alpha_i (u^* - u^{(i)})^T [-g^{(i)}] \\ &= \alpha_i^2 \|g^{(i)}\|^2 - 2\alpha_i \cos(\theta_u) \|u^* - u^{(i)}\| \|g^{(i)}\| \\ &\leq \alpha_i^2 \|g^{(i)}\|^2 - 2\alpha_i \underline{c} \|u^* - u^{(i)}\| \|g^{(i)}\| \\ &= \alpha_i \|g^{(i)}\|^2 \left( \alpha_i - 2\underline{c} \frac{\|u^* - u^{(i)}\|}{\|g^{(i)}\|} \right) \\ &\leq \alpha_i \|g^{(i)}\|^2 \left( \alpha_i - \frac{2\underline{c}}{(1 + \gamma)\sigma_{\max}} \right) \\ &\leq 0 \end{aligned} \quad (5.3.34)$$

In conclusion, for  $\gamma$  satisfying (5.3.26) and (5.3.31), when step size upper bound  $\bar{\alpha}_{\text{mono.}}$  satisfies (5.3.27) and (5.3.33),

$$\|u^{(i+1)} - u^*\| \leq \|u^{(i)} - u^*\| \quad (5.3.35)$$

holds. By (5.3.28),

$$(1 - \gamma)\|\nabla\hat{J}(u^{(i)})\| \leq \|g^{(i)}\| \leq (1 + \gamma)\|\nabla\hat{J}(u^{(i)})\| \quad (5.3.36)$$

with  $\lambda < 1$  by assumption (5.3.27) and therefore in  $U$ ,

$$\|g^{(i)}\| = 0 \Leftrightarrow \|\nabla\hat{J}(u^{(i)})\| = 0 \Leftrightarrow u^{(i)} = u^*$$

Since the equality in (5.3.34) holds if and only if  $\alpha_i = 0$  or  $\|g^{(i)}\| = 0$ , the equality in (5.3.35) holds if and only if  $\alpha_i = 0$  or  $u^{(i)} = u^*$ .  $\square$

These two theorems talk about the potential monotonicity of objective function value and distance to optimal control, which, however, are both unobservable. Because, if the parallel-in-time gradient-type method with step size  $\alpha > 0$  is used, i.e., no full forward sweep, which can be achieved by  $N$  steps of degenerate step size  $\alpha = 0$  parallel-in-time gradient-type method, then the objective function value,  $\hat{J}(u^{(i)})$ , is not observable. And the distance to optimal control,  $\|u^{(i)} - u^*\|$ , is also not observable since the optimal control is unknown.

In the contrast, so far, I did not obtain monotonicity results on the observable state/adjoint jumps and gradient-type vector magnitude.

Another remark is that for the classic gradient method on an unconstrained minimization problem of a strongly convex function, the monotonic decreasing gradient norm can be guaranteed. But I have difficulty proving the same result for the parallel-in-time gradient-type method, if it is true at all. In the assumption of Lemma 5.3.1, I attempted to prove the monotonicity in the following way,

$$\begin{aligned} \nabla\hat{J}(u^{(i+1)}) &= \nabla\hat{J}(u^{(i)}) + \int_0^1 \nabla^2\hat{J}(u^{(i)} + s[-\alpha_i g^{(i)}])[-\alpha_i g^{(i)}] ds \\ &= \int_0^1 (I - \alpha_i \nabla^2\hat{J}(u^{(i)} + s[-\alpha_i g^{(i)}])) \nabla\hat{J}(u^{(i)}) ds \\ &\quad + \alpha_i \int_0^1 \nabla^2\hat{J}(u^{(i)} + s[-\alpha_i g^{(i)}])[\nabla\hat{J}(u^{(i)}) - g^{(i)}] ds \end{aligned}$$

Then

$$\begin{aligned}
\|\nabla \hat{J}(u^{(i+1)})\| &\leq (1 - \alpha_i \sigma_{\min}) \|\nabla \hat{J}(u^{(i)})\| + \alpha_i \sigma_{\max} \|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \\
&\leq (1 - \alpha_i \sigma_{\min}) \|\nabla \hat{J}(u^{(i)})\| + \alpha_i \sigma_{\max} (1 + \gamma) \|\nabla \hat{J}(u^{(i)})\| \\
&= [1 + \alpha_i \sigma_{\max} \gamma + \alpha_i (\sigma_{\max} - \sigma_{\min})] \|\nabla \hat{J}(u^{(i)})\|
\end{aligned}$$

However, the coefficient  $[1 + \alpha_i \sigma_{\max} \gamma + \alpha_i (\sigma_{\max} - \sigma_{\min})]$  can not be bounded below one by only manipulating  $\alpha_i$ .

### 5.3.3 Constrained Problem

In the following Lemma 5.3.4, I first introduce two simple and intuitive bounds related to classic projected gradient step when control variable is away from the optimal control. The first bound (5.3.37) is a lower bound of the angle between the negative gradient direction and the actual control update after the metric projection. The second bound (5.3.38) is a lower bound of the ratio of the actually control update magnitude over the tentative gradient step magnitude.

**Lemma 5.3.4** *Consider generic constrained minimization problem,*

$$\begin{aligned}
&\min_{u \in \mathbb{R}^n} J(u) \\
&\text{subject to } u \in D \subset \mathbb{R}^n
\end{aligned}$$

If  $D$  is closed, convex and exists compact  $U$  such that,

- $J \in C^1(U)$  (For non-open  $U$ , refer to explanation on Page 123)
- $J$  is strongly convex in  $U$ .
- $u^*$  is a local minimum of  $J$  in  $U$ .

Then, for arbitrary  $r > 0$  and arbitrary  $\bar{\alpha} > 0$ , exist  $\tau_{r, \bar{\alpha}}^{angle} > 0, \tau_{r, \bar{\alpha}}^{ratio} > 0$  such that

$$\Theta(u, \alpha) \stackrel{\text{def}}{=} \frac{-\nabla J(u)^T [\mathcal{P}_D(u - \alpha \nabla J(u)) - u]}{\|\nabla J(u)\| \|\mathcal{P}_D(u - \alpha \nabla J(u)) - u\|} \geq \tau_{r, \bar{\alpha}}^{angle} \quad 0 \leq \alpha \leq \bar{\alpha} \quad (5.3.37)$$

$$\Psi(u, \alpha) \stackrel{\text{def}}{=} \frac{\|\mathcal{P}_D(u - \alpha \nabla J(u)) - u\|}{\|\alpha \nabla J(u)\|} \geq \tau_{r, \bar{\alpha}}^{ratio} \quad 0 < \alpha \leq \bar{\alpha} \quad (5.3.38)$$

for all  $u \in (U \setminus B(u^*, r)) \cap D$ .

**Proof:** Due to strong convexity,  $u \in (U \setminus B(u^*, r)) \cap D$  implies  $\nabla J(u) \neq 0$ . By  $u$  being not optimal,  $\mathcal{P}_D(u - \bar{\alpha}\nabla J(u)) - u \neq 0$ . Lemma 5.1.7 yields,

$$-\nabla J(u)^T[\mathcal{P}_D(u - \bar{\alpha}\nabla J(u)) - u] > 0 \quad u \in (U \setminus B(u^*, r)) \cap D$$

and trivially

$$\Theta(u, \bar{\alpha}) > 0 \quad u \in (U \setminus B(u^*, r)) \cap D$$

By the continuous differentiability of  $J$  and continuity of metric projection  $\mathcal{P}_D$ ,  $\Theta(u, \bar{\alpha})$  is continuous with respect to  $u$  in the compact set  $(U \setminus B(u^*, r)) \cap D$ , and thus exists  $\tau_{r, \bar{\alpha}}^{\text{angle}} > 0$  such that

$$\Theta(u, \bar{\alpha}) \geq \tau_{r, \bar{\alpha}}^{\text{angle}} \quad u \in (U \setminus B(u^*, r)) \cap D \quad (5.3.39)$$

Since  $\alpha \leq \bar{\alpha}$ , using Lemma 5.1.5,

$$\Theta(u, \alpha) \geq \Theta(u, \bar{\alpha}) \geq \tau_{r, \bar{\alpha}}^{\text{angle}} > 0 \quad (5.3.40)$$

Similarly, exists  $\tau_{r, \bar{\alpha}}^{\text{ratio}} > 0$  such that

$$\Psi(u, \bar{\alpha}) \geq \tau_{r, \bar{\alpha}}^{\text{ratio}} \quad u \in (U \setminus B(u^*, r)) \cap D \quad (5.3.41)$$

When  $0 < \alpha \leq \bar{\alpha}$ , using Lemma 5.1.4,

$$\begin{aligned} \Psi(u, \alpha) &= \frac{\|\mathcal{P}_D(u - \alpha\nabla J(u)) - u\|}{\|\alpha\nabla J(u)\|} = \frac{\|\frac{\bar{\alpha}}{\alpha}\mathcal{P}_D(u - \alpha\nabla J(u)) - u\|}{\|\bar{\alpha}\nabla J(u)\|} \\ &\geq \frac{\|\mathcal{P}_D(u - \bar{\alpha}\nabla J(u)) - u\|}{\|\bar{\alpha}\nabla J(u)\|} = \Psi(u, \bar{\alpha}) \geq \tau_{r, \bar{\alpha}}^{\text{ratio}} \end{aligned}$$

□

Making use of Lemma 5.3.4, the following Theorem claims that monotonic decreasing in true objective function value corresponding to the control iterates can be guaranteed before the distance to the optimal control falls below a threshold.

**Theorem 5.3.5** *If there exists compact  $U$  such that*

- $u^* \in U$  is a local minimum of the constrained problem
- $\nabla J_k$  and  $\nabla F_k$  is Lipschitz and bounded for  $u_k \in U_k$  and  $y_k \in Y_k(U)$  for all  $k$
- $\hat{J} \in C^1(U)$  (For non-open  $U$ , refer to explanation on Page 123)
- exists  $\sigma_{min}, \sigma_{max} > 0$  such that, for  $u \in U$ ,

$$\sigma_{min}I \prec \nabla^2 \hat{J}(u) \prec \sigma_{max}I.$$

When the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), for any

$$r > 0, \tag{5.3.42}$$

there is a step size  $\bar{\alpha}_{mono.p.}$  and  $\gamma > 0$  such that, if for a certain  $i \geq 4(N - 1)$  and a certain index  $i^\#$

- previous several iterations has iteration-wise relative error bounded by  $\gamma$

$$\|\nabla \hat{J}(u^{(j)}) - g^{(j)}\| \leq \gamma \|\nabla \hat{J}(u^{(j)})\|, \quad j = i - 2(N - 1), \dots, i - 1$$

- step size satisfies

$$0 \leq \alpha_j \leq \bar{\alpha}_{mono.p.} \quad \text{for } i - 2(N - 1) \leq j < i^\#$$

- $u^{(j)} \in U$  for all  $i - 4(N - 1) \leq j \leq i^\#$
- control is away from the optimal control in the sense of

$$u^{(j)} \notin B(u^*, r) \text{ for all } i \leq j < i^\# \tag{5.3.43}$$

then,

$$\hat{J}(u^{(j+1)}) \leq \hat{J}(u^{(j)}) \quad \text{for all } i \leq j < i^\#$$

where the equality holds only when  $\alpha_j = 0$ .

**Proof:** Use previous results,

- For arbitrary  $\gamma > 0$  to be determined later in this proof, by Lemma 5.3.1, if

$$\bar{\alpha}_{\text{mono.p.}} \leq \bar{\alpha}_\gamma, \quad (5.3.44)$$

where  $\bar{\alpha}_\gamma$  is given by Lemma 5.3.1, the iteration-wise gradient-type vector error is bounded, i.e.,

$$\|\nabla \hat{J}(u^{(i)}) - g^{(i)}\| \leq \gamma \|\nabla \hat{J}(u^{(i)})\|, \quad \forall i^* \leq j \leq i^\# \quad (5.3.45)$$

To compare  $\hat{J}(u^{(j)})$  and  $\hat{J}(u^{(j+1)})$ , first look at the first order term in Taylor expansion,

$$\begin{aligned} & \nabla \hat{J}(u^{(j)})^T (u^{(j+1)} - u^{(j)}) = \nabla \hat{J}(u^{(j)})^T (\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - u^{(j)}) \\ & = \nabla \hat{J}(u^{(j)})^T (\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}) \\ & + \nabla \hat{J}(u^{(j)})^T [\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - \mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)}))] \end{aligned} \quad (5.3.46)$$

Examine the magnitude of the second term in (5.3.46), by (5.3.45),

$$\begin{aligned} & \nabla \hat{J}(u^{(j)})^T [\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - \mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)}))] \\ & \leq \|\nabla \hat{J}(u^{(j)})\| \|\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - \mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)}))\| \\ & \leq \|\nabla \hat{J}(u^{(j)})\| \|(u^{(j)} - \alpha_j g^{(j)}) - (u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)}))\| \\ & = \alpha_j \|\nabla \hat{J}(u^{(j)})\| \|g^{(j)} - \nabla \hat{J}(u^{(j)})\| \\ & \leq \alpha_j \gamma \|\nabla \hat{J}(u^{(j)})\|^2 \end{aligned} \quad (5.3.47)$$

Apply orthogonal decomposition to the projected gradient step update in the first term of (5.3.46),

$$\begin{aligned} & \mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)} \\ & = \frac{\nabla \hat{J}(u^{(j)})}{\|\nabla \hat{J}(u^{(j)})\|} \frac{\nabla \hat{J}(u^{(j)})^T}{\|\nabla \hat{J}(u^{(j)})\|} (\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}) \\ & + [I - \frac{\nabla \hat{J}(u^{(j)})}{\|\nabla \hat{J}(u^{(j)})\|} \frac{\nabla \hat{J}(u^{(j)})^T}{\|\nabla \hat{J}(u^{(j)})\|}] (\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}) \end{aligned} \quad (5.3.48)$$



which is well defined since  $\nabla \hat{J}(u^{(j)}) \neq 0$  due to  $u^{(j)} \notin B(u^*, r) \cap D$ . Because the second term in (5.3.48) is perpendicular to  $\nabla \hat{J}(u^{(j)})$ ,

$$\begin{aligned}
& \nabla \hat{J}(u^{(j)})^T (\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}) \\
&= \nabla \hat{J}(u^{(j)})^T \frac{\nabla \hat{J}(u^{(j)})}{\|\nabla \hat{J}(u^{(j)})\|} \frac{\nabla \hat{J}(u^{(j)})^T}{\|\nabla \hat{J}(u^{(j)})\|} (\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}) \\
&= -\alpha_j \|\nabla \hat{J}(u^{(j)})\|^2 \frac{-\nabla \hat{J}(u^{(j)})^T \mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}}{\|-\nabla \hat{J}(u^{(j)})\| \|\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}\|} \\
&\quad \times \frac{\|\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}\|}{\|-\alpha_j \nabla \hat{J}(u^{(j)})\|}
\end{aligned} \tag{5.3.49}$$

To avoid a potential seemingly recursive dependence of the step size upper bound, pick arbitrary  $\bar{\alpha}_{\text{upper}} > 0$ . By Lemma 5.3.4, if

$$\bar{\alpha}_{\text{mono.p.}} \leq \bar{\alpha}_{\text{upper}} \tag{5.3.50}$$

Then, in (5.3.49),

$$\begin{aligned}
\frac{-\nabla \hat{J}(u^{(j)})^T \mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}}{\|-\nabla \hat{J}(u^{(j)})\| \|\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}\|} &\geq \tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{angle}} \\
\frac{\|\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}\|}{\|-\alpha_j \nabla \hat{J}(u^{(j)})\|} &\geq \tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{ratio}}
\end{aligned} \tag{5.3.51}$$

and therefore,

$$\nabla \hat{J}(u^{(j)})^T (\mathcal{P}_D(u^{(j)} - \alpha_j \nabla \hat{J}(u^{(j)})) - u^{(j)}) \leq -\alpha_j \tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{angle}} \tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{ratio}} \|\nabla \hat{J}(u^{(j)})\|^2 \tag{5.3.52}$$

Combine (5.3.47) and (5.3.52) using (5.3.46),

$$\begin{aligned}
\nabla \hat{J}(u^{(j)})^T (u^{(j+1)} - u^{(j)}) &\leq -\alpha_j \tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{angle}} \tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{ratio}} \|\nabla \hat{J}(u^{(j)})\|^2 + \alpha_j \gamma \|\nabla \hat{J}(u^{(j)})\|^2 \\
&= -\alpha_j (\tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{angle}} \tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{ratio}} - \gamma) \|\nabla \hat{J}(u^{(j)})\|^2
\end{aligned}$$

If the iteration-wise relative error satisfies,

$$\gamma \leq \frac{\tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{angle}} \tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{ratio}}}{2} \tag{5.3.53}$$

A bound for the first order term in Taylor expansion is obtained,

$$\nabla \hat{J}(u^{(j)})^T (u^{(j+1)} - u^{(j)}) \leq -\alpha_j \frac{\tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{angle}} \tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{ratio}}}{2} \|\nabla \hat{J}(u^{(j)})\|^2 \tag{5.3.54}$$

Control update is also bounded by the true gradient scaled by step size,

$$\begin{aligned} \|u^{(j+1)} - u^{(j)}\| &= \|\mathcal{P}_D(u^{(j)} - \alpha_j g^{(j)}) - u^{(j)}\| \\ &\leq \alpha_j \|g^{(j)}\| \\ &\leq \alpha_j (\gamma + 1) \|\nabla \hat{J}(u^{(j)})\| \end{aligned} \tag{5.3.55}$$

Now, look at the objective function value change, with  $L$  being a Lipschitz constant for  $\nabla J$  which exists for the boundedness of the largest singular value of  $\nabla^2 \hat{J}$  in  $\bar{B}(u^*, R) \cap D$ , using (5.3.54) and (5.3.55),

$$\begin{aligned} \hat{J}(u^{(j+1)}) - \hat{J}(u^{(j)}) &\leq J(u^{(j)})^T(u^{(j+1)} - u^{(j)}) + \frac{L}{2} \|u^{(j+1)} - u^{(j)}\|^2 \\ &\leq -\alpha_j \frac{\tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{angle}} \tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{ratio}}}{2} \|\nabla \hat{J}(u^{(j)})\|^2 + \frac{L}{2} [\alpha_j (\gamma + 1) \|\nabla \hat{J}(u^{(j)})\|]^2 \\ &= -\alpha_j \frac{L(\gamma + 1)^2}{2} \left( \frac{\tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{angle}} \tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{ratio}}}{L(\gamma + 1)^2} - \alpha_j \right) \|\nabla \hat{J}(u^{(j)})\|^2 \end{aligned} \tag{5.3.56}$$

This leads to an upper bound of the step size,

$$\bar{\alpha}_{\text{mono.p.}} < \frac{\tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{angle}} \tau_{r, \bar{\alpha}_{\text{upper}}}^{\text{ratio}}}{L(\gamma + 1)^2} \tag{5.3.57}$$

In conclusion, for arbitrary  $\bar{\alpha}_{\text{upper}}$ , any  $\gamma$  that meets (5.3.53) and any  $\bar{\alpha}_{\text{mono.p.}}$  that validates (5.3.44), (5.3.50), and (5.3.57), will guarantee, by (5.3.56),

$$\hat{J}(u^{(j+1)}) \leq \hat{J}(u^{(j)})$$

where the equality holds only when  $\alpha_j = 0$ . □

In Theorem 5.3.5, the monotonic decreasing objective function values are guaranteed away from a certain proximity of the optimal control. By strong convexity of objective function, the condition on control that the distance to optimal solution greater than  $r$  can be replaced by a condition on objective function value that it is greater than the optimal objective function value plus an arbitrarily small positive number,  $\epsilon$ . In other words,

**Corollary 5.3.6** *If there exists compact  $U$  such that*

- $u^* \in U$  is a local minimum of the constrained problem
- $\nabla J_k$  and  $\nabla F_k$  is Lipschitz and bounded for  $u_k \in U_k$  and  $y_k \in Y_k(U)$  for all  $k$
- $\hat{J} \in C^1(U)$  (For non-open  $U$ , refer to explanation on Page 123)
- exists  $\sigma_{min}, \sigma_{max} > 0$  such that, for  $u \in U$ ,

$$\sigma_{min}I \prec \nabla^2 \hat{J}(u) \prec \sigma_{max}I.$$

When the projected parallel-in-time gradient-type method with step size  $\{\alpha_j\}$ , Algorithm 13, is applied to problem (5.2.1), for any

$$\epsilon > 0, \tag{5.3.58}$$

there is a step size  $\bar{\alpha}_{mono.p.}$  and  $\gamma > 0$  such that, if for a certain  $i \geq 4(N - 1)$  and a certain index  $i^\#$

- previous several iterations has iteration-wise relative error bounded by  $\gamma$

$$\|\nabla \hat{J}(u^{(j)}) - g^{(j)}\| \leq \gamma \|\nabla \hat{J}(u^{(j)})\|, \quad j = i - 2(N - 1), \dots, i - 1$$

- step size satisfies

$$0 \leq \alpha_j \leq \bar{\alpha}_{mono.p.} \quad \text{for } i - 2(N - 1) \leq j < i^\#$$

- $u^{(j)} \in U$  for all  $i - 4(N - 1) \leq j \leq i^\#$

- control is away from the optimal control in the sense of

$$\hat{J}(u^{(j)}) > \hat{J}(u^*) + \epsilon \text{ for all } i \leq j < i^\# \tag{5.3.59}$$

then,

$$\hat{J}(u^{(j+1)}) \leq \hat{J}(u^{(j)}) \quad \text{for all } i \leq j < i^\#$$

where the equality holds only when  $\alpha_j = 0$ .

**Proof:** Let  $L > 0$  be a Lipschitz constant for  $\nabla \hat{J}$ , since

$$\begin{aligned} \hat{J}(u) - \hat{J}(u^*) &\leq \nabla \hat{J}(u^*)(u - u^*) + \frac{L}{2} \|u - u^*\|^2 \\ &\leq \|\nabla \hat{J}(u^*)\| \|u - u^*\| + \frac{L}{2} \|u - u^*\|^2 \end{aligned} \quad (5.3.60)$$

By root formula of second order polynomial, for any  $\epsilon > 0$ , when

$$\|u - u^*\| \leq r \stackrel{\text{def}}{=} \frac{\sqrt{\|\nabla \hat{J}(u^*)\|^2 + 2L\epsilon} - \|\nabla \hat{J}(u^*)\|}{L}$$

The relation  $\hat{J}(u) \leq \hat{J}(u^*) + \epsilon$  holds and therefore the condition

$$\hat{J}(u^{(j)}) > \hat{J}(u^*) + \epsilon \text{ for all } i \leq j < i^\#$$

implies

$$u^{(j)} \notin B(u^*, r) \cap D \text{ for all } i \leq j < i^\#$$

Then, applying Theorem 5.3.5 completes the proof.  $\square$

However, the guaranteed monotonic decreasing objective function values are not observable during the parallel-in-time gradient-type iterations with non-zero step sizes, since the state variables in memory are not feasible.

Similarly, given any open neighborhood of the optimal control, when control variables are outside such neighborhood, the monotonic convergence of constrained problem in terms of distance to the optimal control is also guaranteed. The proof is in the first part of Theorem 5.2.5 and I do not repeat it here.

## 5.4 Summary

In this chapter, I investigated the behavior of the parallel-in-time gradient-type method in general nonlinear problems, as opposed to the linear-quadratic problem discussed in previous chapters. The projected version of the parallel-in-time gradient-type method is considered, which appends a projection of control variable to a closed convex set to each of the parallel-in-time gradient-type control update.

I established a series of theorems to ultimately build the convergence proof of the method. For the logic dependency structure of the theorems, please refer to Figure 5.1. The results obtained in this chapter are based on the similarity between the true gradient and the gradient-type vector. There are existing convergence proofs for the classical gradient method, e.g. [NW06]. I gave the proofs for the parallel-in-time gradient-type method by combining the techniques used in classical gradient method proofs and carefully bounding the error introduced by the gradient-type vector considered as a type of inexact gradient.

The convergence is guaranteed only when sufficiently small step size is used. If someone traces back the proofs and finds the upper bound of step size that guarantees convergence, it maybe too small to yield good performance in practice. This situation is similar to the one in [TBA86]. The theorems in this chapter are of a different type from the practical convergence theorems, for example, of line search methods based on the Armijo-Goldstein condition, which provide a path to compute step sizes.

Two types of assumption of the problem yields convergence theorems that is particularly convenient to apply,

- the reduced control space objective function is strongly convex (Theorem 5.2.9);
- the projection set of control variable is compact (Theorem 5.2.8).

After the convergence proofs, I studied the possibility of monotonic convergence in terms of objective function value and distance to the optimal control. The previous numerical experiment in Section 3.6.1 showed that with an optimal fixed step size, the convergence can be non-monotonic in terms of distance to the optimal control. However, in this chapter, I proved that the monotonicity can be achieved. For unconstrained problem, with sufficient small step size, the parallel-in-time gradient-type method is guaranteed to converge monotonically. For constrained problem, outside any neighborhood of the optimal solution, the parallel-in-time gradient-type method is guaranteed to converge monotonically with a sufficiently small step size that de-

pende on the neighborhood.

# Chapter 6

## Reservoir Optimization

### 6.1 Introduction

The energy demand of the world is increasing. Currently, renewable sources as solar, wind, biomass, and geothermal energy is not able to substitute the fossil based energy as oil and coal. More large oil fields are moving into their recovery operation mature stage. Oil companies are considering an efficient way to develop and maintain the production quality in the existing fields. [Bro04]

In the secondary oil reservoir recovery, water is injected into an oil reservoir to displace oil. In a reservoir, water is injected into a large number of wells at fixed locations across hundreds of miles and over many years. By this injection, oil is displaced through the subsurface porous media and produced from a number of production wells.

Seeking the optimal scheme of injection/production wellrates allocation in secondary oil reservoir recovery can be quite challenging. Various factors, as large complex geographical structure and fluid components interaction, influences the reservoir at the same time.

In the early 1980s, mainly vertical conventional wells were used, which penetrate the reservoir vertically and has the disadvantage of limited contact area with the

reservoir. In the late 1980s, horizontal wells were technically available. In the beginning of the 2000s, multilateral wells that branches in the reservoir greatly increasing the contact area and smart wells with downhole sensors and valves appeared. These valves allow continuous (both in time and flow rate) adjustment of injection and production wellrates (compared to wells with only on/off modes), which offer flexible operation scheme to control the fluid flow pattern. With this possibility of flexible operation schemes, petroleum engineers can potentially improve oil reservoir recovery by using an optimal operation scheme. Therefore, more work is needed to find the optimal operation of wells taking advantage of these flexibilities.

In this project, optimal control theory is applied to solve the problem of finding the optimal wellrates governed by a highly nonlinear PDE system. Using optimal control theory (using adjoint approach to compute gradient) to solve this optimization has the advantage that the gradient computation load is independent of the number of controls which can be large taking into account large number of wells and time dependent wellrate adjustment in the simulation window. Therefore, the adjoint method is especially suitable to apply here.

However, even with adjoint method to avoid the costly finite difference approximation of the gradient, the forward and backward computation required to obtain the gradient is still expensive since both of them involve solving time dependent PDEs. Therefore, I apply the proposed parallel-in-time gradient-type method to accelerate this time dependent PDE constrained optimization problem and compare its numerical performance with the classical gradient method.

The numerical method is formulated to maximize the net present value of an (secondary) oil recovery process. An oil/water two-phase immiscible incompressible model is used and solved by a finite volume based sequential splitting method. Bound constraints on the wellrates are implemented explicitly. The practical limit of adjusting wellrates in discrete time points is enforced implicitly, by using piecewise constant basis function for the control variables, in the scenarios where wellrates are only peri-



odically adjustable in several predetermined time. Wells are models as point sources and sinks. Objective function gradient is computed by the adjoint method.

The implementation uses the Sandia National Lab’s Trilinos Project [HBH<sup>+</sup>05] libraries framework to facilitate parallel linear algebra, linear system solver and preconditioner, and automatic differentiation.

The related work on reservoir water flooding optimization is reviewed in Section 2.3.

This chapter is organized as follows. In Section 6.2, I introduce the objective function for the water flooding optimization problem, its discretized form, discrete adjoint variable computation, and projected gradient method. In Section 6.3, I present numerical examples using both classical gradient method and parallel-in-time gradient-type method.

The reservoir simulator used in this project is standard brick-cell finite volume method with backward Euler scheme in time. In Appendix B, I include a detailed description of the reservoir physics of the incompressible, immiscible water/oil two-phase model and discretization of the PDE. Implementation details of the parallel-in-time gradient-type applied to this particular problem is discussed in Appendix C.

## 6.2 Optimization

In this section, I introduce the oil reservoir well rates optimization problem, its objective function, fully discretized form, and structure of the classical gradient method applied to this problem. Reservoir model details are included in Appendix B.

### 6.2.1 Example Objective Function

The Net Present Value (NPV) of the oil recovery process is used as the optimization objective function.

Let  $I_{\text{inj}}$  and  $I_{\text{pro}}$  be the index sets of injection and production wells respectively.

For  $i \in I_{\text{inj}}$  or  $i \in I_{\text{pro}}$ ,  $x_i$  denotes the location of that well. Since the well rates  $q$ ,  $q_w$  in the infinite dimensional formulation (B.1.15) are in unit  $[\text{time}^{-1}]$ , I assign a constant  $\gamma$   $[\text{length}^3]$  for all wells. This scaling can be different for different wells and in the fully discrete case it will be related to the finite volume discretization, As an example, I will optimize the following net present value (NPV) of the sum of water injection cost, water treatment cost, and oil revenue,

$$\int_0^T \frac{\overbrace{-r_{\text{inj}} \sum_{i \in I_{\text{inj}}} \gamma q(x_i, t)}^{\text{injection cost}} - \overbrace{r_{\text{oper}} \sum_{i \in I_{\text{pro}}} \gamma |q(x_i, t)| f_w(s_w(x_i, t))}^{\text{water treatment cost}} + \overbrace{r_{\text{oil}} \sum_{i \in I_{\text{pro}}} \gamma |q(x_i, t)| f_o(s_w(x_i, t))}^{\text{oil revenue}}}{(1 + r_{\text{dis}})^t} dt \quad (6.2.1)$$

Note that  $q(x_i, t) < 0$  for production wells and  $q(x_i, t) > 0$  for injection wells. The coefficients  $r_{\text{inj}}, r_{\text{oper}}, r_{\text{oil}}, r_{\text{dis}} \geq 0$  represent water injection cost, cost of water production and treatment, oil price, and time discount rate, respectively.

As is common, the abstract formulation of the optimization problem is written as a minimization problem. Therefore, I minimize the negative of (6.2.1), i.e.,  $J$  in (6.2.2a) below represents the negative of the discretization of (6.2.1).

## 6.2.2 Discretized Form of the Optimization Problem

I discretize the PDE system (B.1.15) by a finite volume method, as described in Section B.2. Furthermore, to be specific I use a particular sequential approach to discretize the resulting semidiscretized PDE system.

Let  $K$  be number of discrete time steps, and  $N$  be number of Finite Volume Method cells. The vectors representing water saturation, global pressure, and well rates at different time steps are denoted by  $s_0, s_1, \dots, s_K$ , by  $p_0, \dots, p_{K-1}$ , and by  $q_0, q_1, \dots, q_K$ , respectively. Furthermore,  $Q_0, \dots, Q_K$  are the closed convex sets of feasible well rates, which incorporates the discretization of the zero-sum condition of incompressible flow model  $\int_{\Omega} q(x, t) dx = 0$  for all  $t$ , as well as pointwise constraints on the well-rates. In the examples all sets are equal  $Q_0 = Q_1 = \dots = Q_K$ .

The abstract form of the discretized optimization problem is given by

$$\text{Minimize} \quad J(s_0, s_1, \dots, s_K, q_0, \dots, q_K) := \sum_{k=0}^K J_k(s_k, q_k) \quad (6.2.2a)$$

subject to

$$H_k(s_k, p_k, q_k) = 0 \quad \text{for } k = 0, \dots, K - 1, \quad (6.2.2b)$$

$$G_k(s_k, s_{k+1}, p_k, q_k) = 0 \quad \text{for } k = 0, \dots, K - 1, \quad (6.2.2c)$$

$$s_0 = s_{\text{given}}, \quad (6.2.2d)$$

$$q_k \in Q_k \quad \text{for } k = 0, \dots, K - 1, \quad (6.2.2e)$$

where  $H_k$  and  $G_k$  is representing the pressure and saturation equation developed in Subsection B.2 at time step  $t_k$  respectively. (I include an auxiliary control  $q_K$  in the objective to avoid distinction between time steps  $k = 0, \dots, K - 1$  and time step  $k = K$ , although in this time stepping the constraints (6.2.2b-e) do not involve  $q_K$ .)

The abstract formulation also appears in [Wie10], where other variations of the sequential time stepping method and the IMPSAT time stepping method have also been considered.

### 6.2.3 Classical Gradient Method

I assume that for given well-rates  $q_k \in Q_k$ ,  $k = 0, \dots, K - 1$ , the discretized saturation and pressure equations (6.2.2b-d) have a unique solution  $s_0, s_1, \dots, s_K, p_0, \dots, p_{K-1}$ . Then I can equivalently formulate (6.2.2) as an optimization problem in the well-rates  $q_0, \dots, q_K$  and define discretized saturations and pressures  $s_0, s_1, \dots, s_K, p_0, \dots, p_{K-1}$  as implicit functions of well-rates. This implicit constrained formulation of the optimization problem is given by

$$\text{Minimize} \quad \widehat{J}(q_0, \dots, q_K) := \sum_{k=0}^K J_k(s_k(q_0, \dots, q_K), q_k), \quad (6.2.3a)$$

$$\text{subject to } q_k \in Q_k, \quad \text{for } k = 0, \dots, K - 1, \quad (6.2.3b)$$

where for given  $q = (q_0, \dots, q_K)$  the discretized saturations and pressures  $s_0, s_1, \dots, s_K, p_0, \dots, p_{K-1}$  are given as solutions of

$$H_k(s_k, p_k, q_k) = 0, \quad \text{for } k = 0, \dots, K-1, \quad (6.2.4a)$$

$$G_k(s_k, s_{k+1}, p_k, q_k) = 0, \quad \text{for } k = 0, \dots, K-1, \quad (6.2.4b)$$

$$s_0 = s_{\text{given}}. \quad (6.2.4c)$$

## 6.2.4 Adjoint Equation Approach for Gradient Computation

The gradient of  $\hat{J}$  is computed via the adjoint equation approach as in [Hei08, Wie10].

The Lagrangian associated with the minimization problem (6.2.2) is

$$\begin{aligned} \mathcal{L}(s, p, q, \mu, \eta, \xi) = & \sum_{k=0}^K J_k(s_k, q_k) + \xi^T (s_0 - s_{\text{given}}) + \sum_{k=0}^K \mu_k^T H_k(s_k, p_k, q_k) \\ & + \sum_{k=0}^{K-1} \eta_k^T G_k(s_k, s_{k+1}, p_k, q_k). \end{aligned}$$

The adjoint equation is obtained by setting the partial derivatives of  $\mathcal{L}$  with respect to  $s_k, p_k, k = 0, \dots, K$ , to zero. The gradient of  $\hat{J}$  is then given by the partial gradient of  $\mathcal{L}$  with respect to  $q_k, k = 0, \dots, K$ . The computation of the gradient of  $\hat{J}$  is summarized in the following Algorithm 14.

## 6.2.5 Projected Gradient Method

Given the gradient, a number of optimization algorithms can be used. For completeness I state the projected gradient method for (6.2.3) as Algorithm 15 below and illustrate in Figure 6.1. See, e.g., [Kel99]. In Algorithm 15,  $\mathcal{P}_Q$  denotes the projection onto the closed convex set  $Q \stackrel{\text{def}}{=} Q_0 \times \dots \times Q_K$ .

In the implementation, the projection step as a quadratic programming problem is solved by Gurobi [GO16] or CGAL (The Computational Geometry Algorithms Library) [The16].

---

**Algorithm 14** Gradient Computation via Adjoint Equations
 

---

- 1: Input  $q = (q_0, \dots, q_K)$ .
- 2: Solve the state equations

$$\begin{aligned} H_k(s_k, p_k, q_k) &= 0 & k = 0, \dots, K-1, \\ G_k(s_k, s_{k+1}, p_k, q_k) &= 0 & k = 0, \dots, K-1, \\ s_0 &= s_{\text{given}}, \end{aligned}$$

forward in time for  $s_1, \dots, s_K, p_0, \dots, p_{K-1}$ .

- 3: Solve the adjoint equations (I omit the arguments of  $H_k, G_k, J_k$ )

$$\begin{aligned} \mu_K &= 0, \\ \eta_{K-1} &= -\left(\frac{\partial G_{K-1}}{\partial s_K}\right)^{-T} \nabla_{s_K} J_K, \\ \mu_k &= -\left(\frac{\partial H_k}{\partial p_k}\right)^{-T} \left(\frac{\partial G_k}{\partial p_k}\right)^T \eta_k, & k = K-1, \dots, 0, \\ \eta_{k-1} &= -\left(\frac{\partial G_{k-1}}{\partial s_k}\right)^{-T} \left[ \nabla_{s_k} J_k + \left(\frac{\partial G_k}{\partial s_k}\right)^T \eta_k + \left(\frac{\partial H_k}{\partial s_k}\right)^T \mu_k \right], & k = K-1, \dots, 1, \end{aligned}$$

backward in time for  $\mu_1, \dots, \mu_K, \eta_0, \dots, \eta_{K-1}$ .

- 4: Compute the gradient (I omit the arguments of  $H_k, G_k, J_k$ )

$$\nabla_{q_k} \widehat{J}(q_0, \dots, q_K) = \nabla_{q_k} J_k + \left(\frac{\partial H_k}{\partial q_k}\right)^T \mu_k + \left(\frac{\partial G_k}{\partial q_k}\right)^T \eta_k, \quad k = 0, \dots, K-1$$


---

---

**Algorithm 15** Projected Gradient Method for Solving (6.2.3), illustrated in Figure 6.1

---

- 1: Input  $q^{(0)} = (q_0^{(0)}, \dots, q_K^{(0)})$ .
  - 2: **for**  $j = 0, \dots, \text{MAX\_ITER} - 1$  **do**
  - 3:   Compute  $\hat{J}(q^{(j)})$  and  $\nabla \hat{J}(q^{(j)})$ .
  - 4:   Check termination criteria; stop if termination criteria is satisfied.
  - 5:   Find step length  $\alpha^{(j)}$  such that  $\mathcal{P}_Q(q^{(j)} - \alpha^{(j)} \nabla \hat{J}(q^{(j)}))$  satisfies the sufficient decrease condition.
  - 6:   Set  $q^{(j+1)} = \mathcal{P}_Q(q^{(j)} - \alpha^{(j)} \nabla \hat{J}(q^{(j)}))$ .
  - 7: **end for**
- 

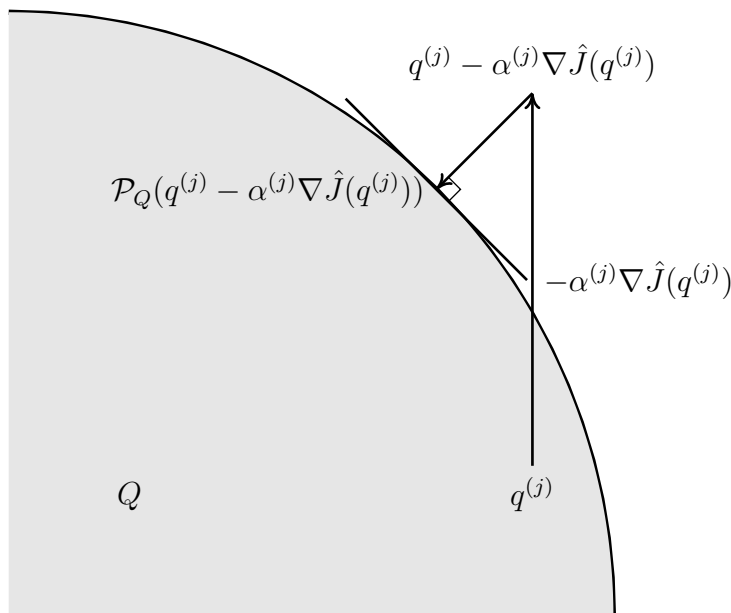


Figure 6.1: Illustration of a projected gradient step described by Algorithm 15

## 6.3 Numerical Examples

I present numerical results,

- In Section 6.3.1, I give optimization results using classical gradient method as a validity check of the model problem.
- In Section 6.3.2, performance of parallel-in-time gradient-type method with a predetermined step size is demonstrated.
- In Section 6.3.3, I tested a heuristic watchdog type of algorithm that modify the parallel-in-time gradient-type method by periodically executing full serial gradient in order to obtain true objective function value by which the algorithm potentially apply backtracking and adjust step size.

All experiments use permeability and porosity data extracted from the SPE 10 data set (<http://www.spe.org/web/csp/datasets/set02.htm>). The coefficients in the NPV objective function are set as follows. The oil price is  $r_{oil} = 80$  dollars/bbl, the water injection cost is  $r_{inj} = 5$  dollars/bbl, the water treatment operation cost is  $r_{oper} = 5$  dollars/bbl, and the time discount rate is  $r_{dis} = 2 \times 10^{-4}$ /day.

### 6.3.1 Classical Gradient Method

This numerical experiment uses a portion of permeability and porosity data from the upper quarter of the SPE 10 data set. The model dimension is  $1200 \times 2200 \times 40$  feet, which is discretized into  $60 \times 220 \times 20 = 264,000$  finite volume cells. Simulation time span is 500 days, split into  $K = 1000$  time marching steps. I place 25 injection wells and 25 production wells in the reservoir in two layers at 20 feet and 30 feet depth, Figure 6.2. Well injection/production rate bounds are set to be 566.37 bbl/day (floating points by the unit conversion). Well rates are piecewise constant functions with constant pieces of length 25 days.

The computations reported on in this subsection are run using 24 cores on the Rice University DAVinCI cluster <sup>1</sup>. The 24 processor cores locate in 2 Westmere nodes (12 cores in each node) at 2.83 GHz with 48 GB of RAM per node. Nodes are connected between each other and to the GPFS fast scratch storage by QDR InfiniBand (40 Gb/s). A typical iteration of the Projected Gradient Algorithm 15, including forward/backward computation, steps size computation, and control updating, takes about 40 minutes.

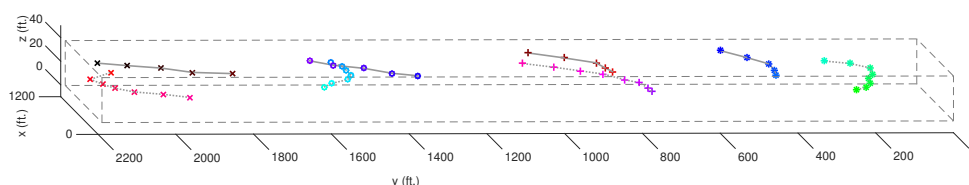


Figure 6.2: Location of injection and production wells. Production wells are indicated by crosses; injection are indicated by circles. Markers connected by solid lines are on the 30 feet layer; markers connected by dotted lines are on the 20 feet layer.

To generate initial well-rates, I first assume that the well rates are constant across all time steps. This leads to a version of (6.2.3) with 25+25 well-rates (one for each production and injection well). This smaller problem is optimized and the optimized constant well-rates are used as initial conditions in the formulation of (6.2.3) in which each well-rate can be adjusted in 25-day intervals. The left plot in Figure 6.3 shows the constant initial well-rates and the right plot in Figure 6.3 shows the optimal well-rates. Note that it is not necessary to obtain the initial well-rates by this procedure for the optimization algorithm to converge. In this example, I take this procedure since it gives the optimization iterations a reasonable well-rates to start, by which the objective function value decrease is more meaningful.

The time dependent behavior of the individual term in the NPV objective function

---

<sup>1</sup>This cluster was supported in part by the Data Analysis and Visualization Cyberinfrastructure funded by NSF under grant OCI-0959097 and Rice University.



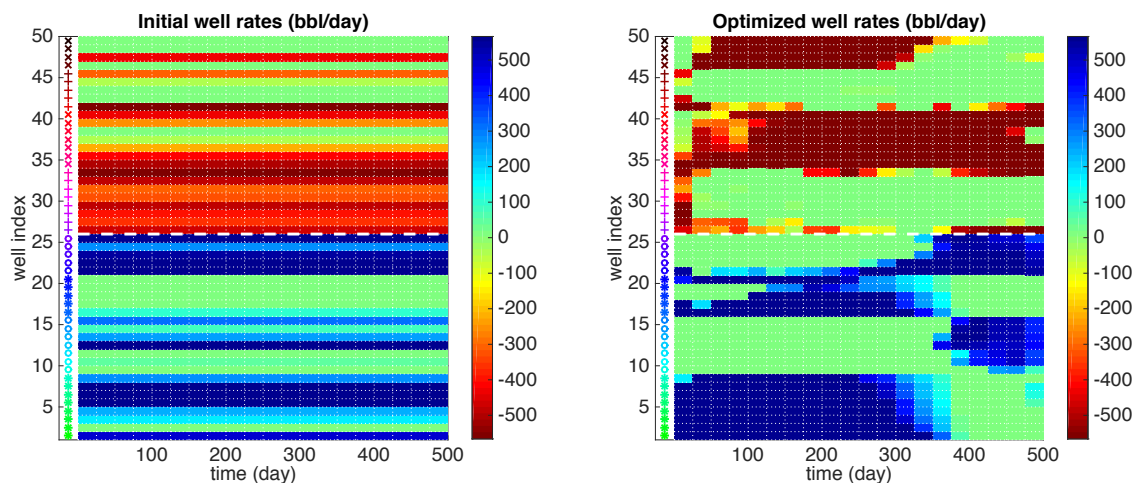


Figure 6.3: Initial well rates (left) and optimized well-rates (right) for the 25 injection well and 25 production wells. The color bars with negative well rates above the dashed line in the middle represent the 25 production wells; the color bars with positive well rates below the dashed line in the middle represent the 25 injection wells. The markers (crosses and circles) on the left of each plot are consistent with the markers in Figure 6.2.

is shown in Figure 6.4. The left plot shows the contributions of the various terms in the objective function at the initial well-rate settings; the right plot shows the contributions of the various terms in the objective function at the optimal well-rate settings. The objective function values correspond to the area under the black curve.

The objective function Iteration history of the projected gradient method is shown in Figure 6.5. The optimization of the well-rates reduced the negative NPV from  $-6.857 \times 10^6$  to  $-7.986 \times 10^6$ , a 16.5% improvement. The optimal NPV is essentially reached after 20 iterations. The remaining iterations are used to reduce the norm of the projected gradient below the required tolerance.

Contours at day 100, 300, and 500 (final time) of water saturations resulting from optimized well-rates are shown in Figure 6.6.

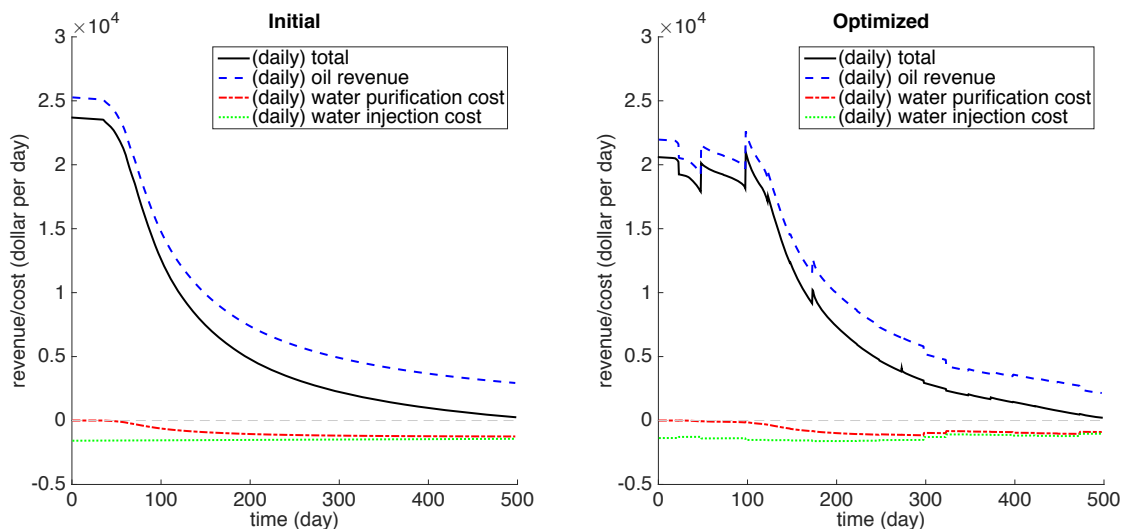


Figure 6.4: Contributions of oil revenue (top blue dashed line), water treatment cost (dot-dash red line blow zero), and water injection cost (bottom green dotted line) to the NPV (solid black line) at the initial well-rate settings (left plot) and at the optimal well-rate settings (right plot), for the example problem in Section 6.3.1. Optimization uses the classical gradient method. The area below the black solid line is the NPV. The NPV for the initial well rates (left) is  $6.857 \times 10^6$ ; the NPV for the optimized well rates (right) is  $7.986 \times 10^6$ . A 16.5% improvement is achieved. The jumps in the curves are caused by the discontinuity in the piecewise constant wells rates.

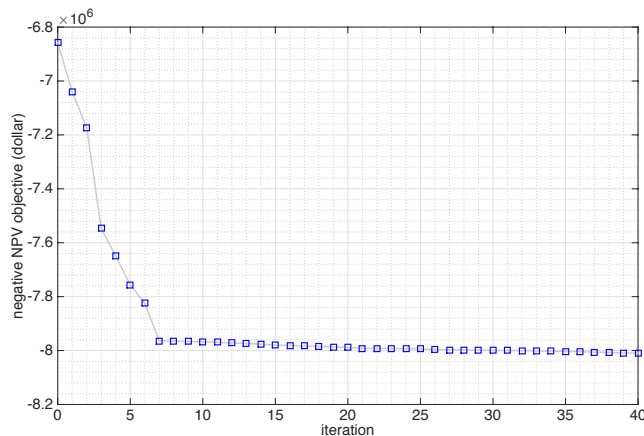


Figure 6.5: Iteration history of the projected gradient method. After 20 iterations the optimal NPV is essentially reached and the remaining iterations are used to reduce the norm of the projected gradient below the required tolerance.

### 6.3.2 Parallel-In-Time Gradient-Type Method

In this section, I present numerical results of applying the fixed step size parallel-in-time gradient-type method to this reservoir optimization problem. The implementation is described in Algorithm 18 of Appendix C with an emphasis on the effect of implicit state equations on the data communication. Aiming for a heuristic way of adjusting step size to help convergence, I present a varying step size modification of the parallel-in-time gradient-type method in Section 6.3.3.

I test the fixed step size parallel-in-time gradient-type method.

1. The fixed step size parallel-in-time gradient-type method is used and timed without any kind of globalization technique, such as backtracking, to guarantee convergence. Because, the states and adjoints are not feasible, i.e. The discrete state and adjoint equations at time subdomain boundaries are not satisfied, during the optimization process in parallel gradient-type method and therefore, the objective function value corresponding to the control at a specific iteration can not be accurately evaluated, I do not have an effective merit function (which

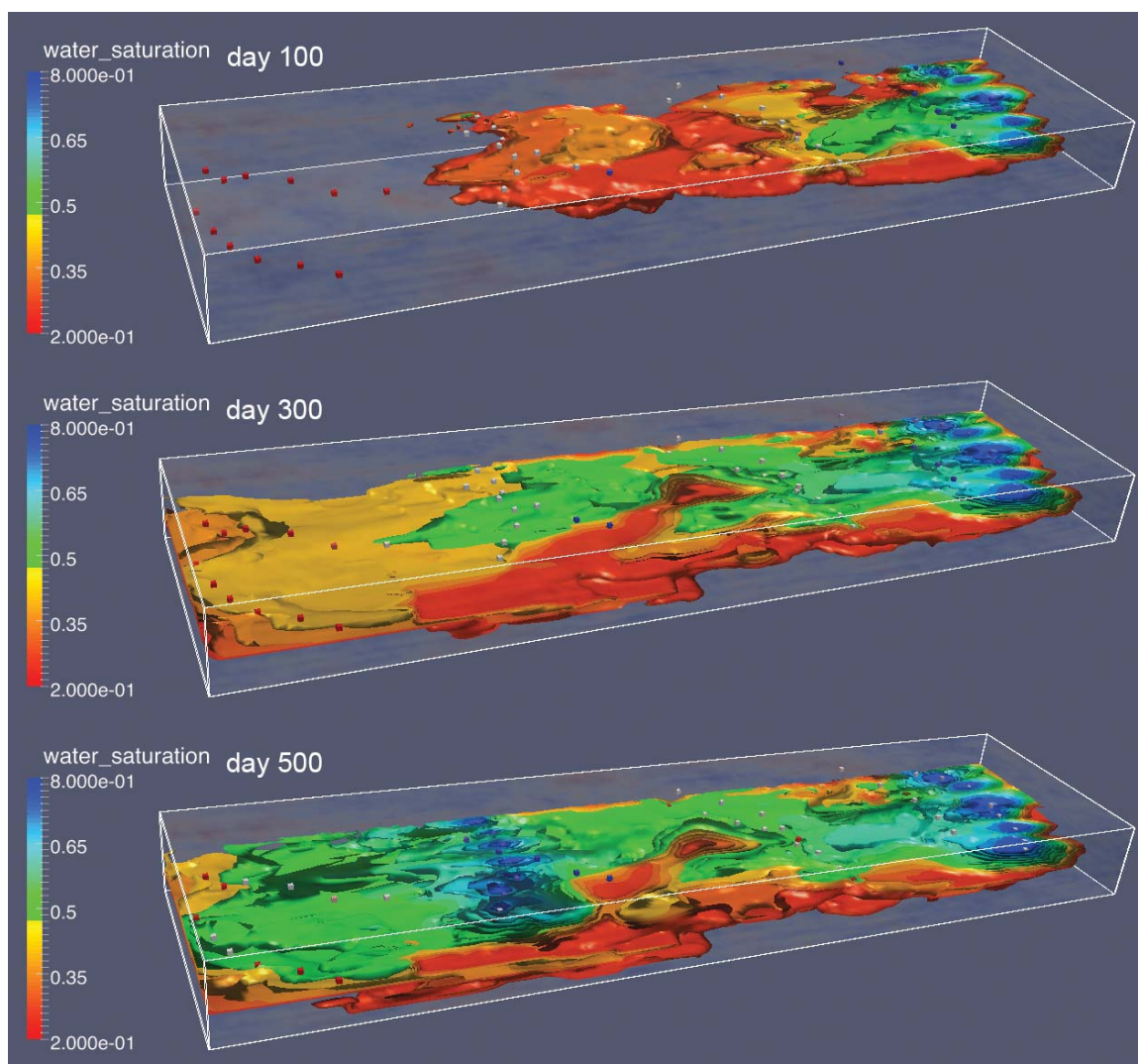


Figure 6.6: Contours at day 100, 300, and 500 (final time) of water saturations resulting from optimized well-rates. Only the bottom half of the reservoir (depth 20-40 feet) is shown to better display water saturations at the 20 feet level, where several of the wells are located.

is the objective function for classical gradient method) to evaluate the new tentative control produced by the parallel gradient-type method. The true objective function value corresponding to a control iterates can too different (for backtracking purpose) from the “objective value” computed during the parallel-in-time gradient-type method runtime using state variables with jumps at subdomain boundaries, see an example in Figure 6.13. So, I accept all the control updates. The tested parallel-in-time gradient-type method optimization process stop after a predetermined number of iterations.

2. Objective function values used in the performance comparison are evaluated after the optimization process by running full forward simulations with every wellrate schedule on the optimization trajectory. This takes time and is not included in the timing of the parallel-in-time gradient-type method.
3. The parallel-in-time gradient-type method subdomain boundary state/adjoint variables are initialized by a full gradient forward-backward sweep. As a result, the first step takes longer time than the rest of the steps.

The step size of both gradient method and parallel gradient-type method are set to be the same. In these tests below, the step size is found by running classical gradient method with backtracking to convergence and choosing a suitable step size from its optimization history.

However, according to experience, a fixed step size, even chosen from a previous optimization history, is not always suitable for the highly nonlinear reservoir optimization problem. Because, usually, during the optimization process, in different region of control variables, the most suitable step size is different. For example, in some cases, in the region near optimal solution, relatively small step size need to be used to converge. But if this small step size is used from the beginning of the optimization iterations, slow convergence will hinder me from comparing the performance of the classical gradient method and the proposed parallel gradient-type method, since us-

ing step sizes that are smaller than necessary in the early iterations far from optimal control gives artificial advantage to parallel gradient-type method over the classical gradient method. Therefore, I chose a moderate step size that will not lead to the final convergence of a very high accuracy, i.e. iterates from both methods with this step size eventually jitter around the optimal solution, but is sufficient to demonstrate the performance of the two methods before the need for decreasing step size.

### 6.3.2.1 Example 1

In this example of small reservoir size, I apply the parallel gradient-type method in a problem of 2 wells in a 3D reservoir. 4 time subdomains are used in the parallel gradient-type method. The parallel computation is run on a desktop computer with 4 cores in 1 CPU.

The model dimension is  $640 \times 320 \times 2$  feet<sup>3</sup> discretized into  $32 \times 32 \times 1 = 1,024$  finite volume cells. Simulation time is 100 days with 100 time marching steps of size 1 days. 1 injection wells and 1 production wells are placed in the only one layer in the discretization. See Figure 6.7 for the location of wells. Wellrates are piecewise constant functions with constant pieces of length 10 days.

In the iteration history in Figure 6.8 of first 200 iterations, I observe that in each optimization iteration, both gradient method and parallel gradient-type method decreases objective function value similarly in each optimization iteration. However, time-wisely, the parallel gradient-type method has around 3.5 times speed-ups compared to the gradient method.

I did not optimize the splitting of time domain and used the evenly split subdomains, which usually results in suboptimal performance, since the time expense in forward computation is different between time marching steps. In Figure 6.9, it can be seen that the forward computation corresponding to the earlier time subdomain is slower than that to the later subdomain, though every time subdomain consists of the same number of time marching steps. It is due to the fact that in the earlier

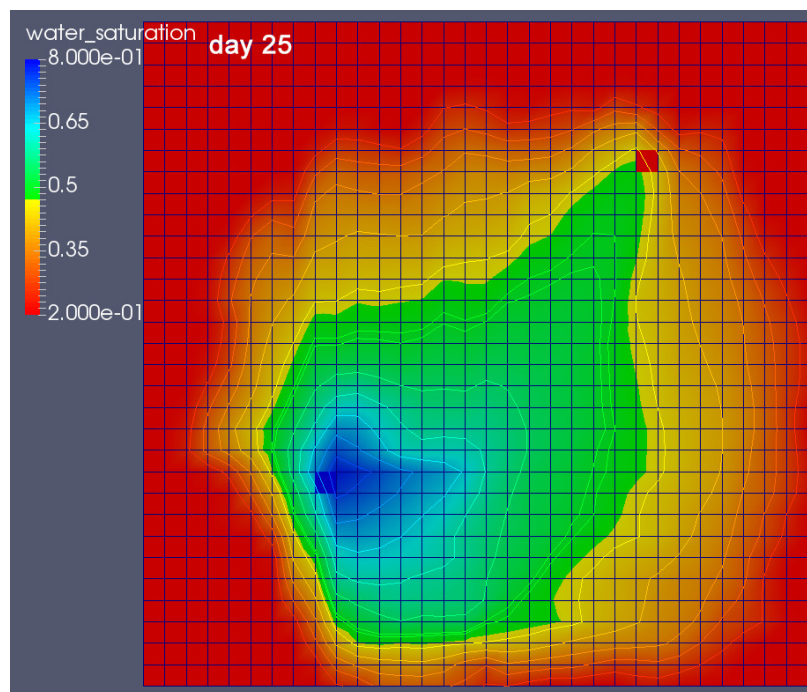


Figure 6.7: 2D reservoir example, water saturation at day 25. The blue and red squares mark the injection and production well respectively.

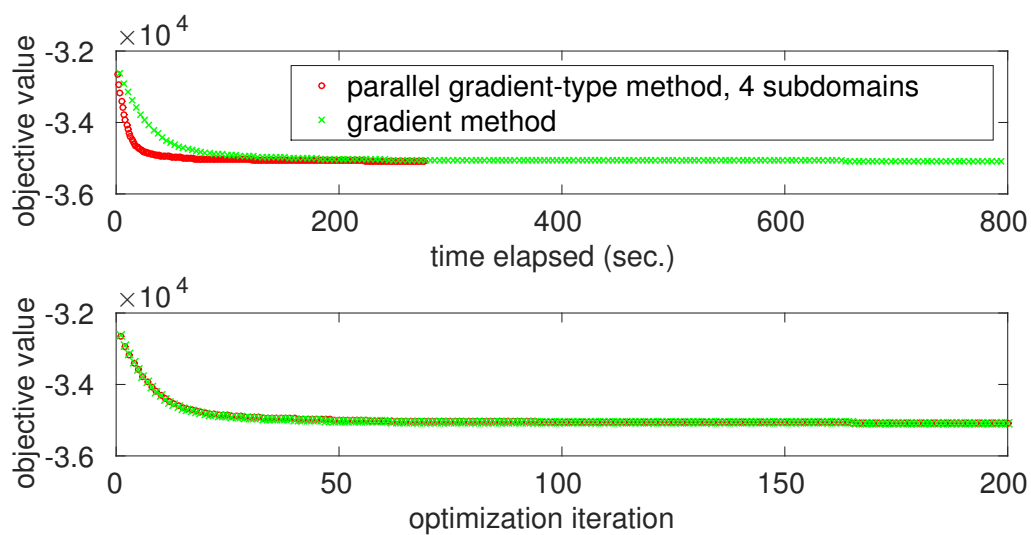


Figure 6.8: 2D reservoir example, iteration history of gradient method and the parallel gradient-type method. The parallel gradient-type method uses 4 time subdomains.

stage of the simulation, the saturation change between time marching steps is often larger than in the later stage of the simulation, which leads to more computation, such as more Newton's iterations in solving the nonlinear time marching saturation equations. This imbalance of computation load causes the cores, other than the first one, waits for the first core to complete forward simulation. This is the reason why the timing plot in Figure 6.8 does not show 4 times speedups instead of around 3.5. In Figure 6.9, one can also see that the backward computation of each time subdomain roughly takes a similar amount of time. A test on using a time subdomain partition based on computation load, instead of even number of time steps, for load balance is included in Appendix D.

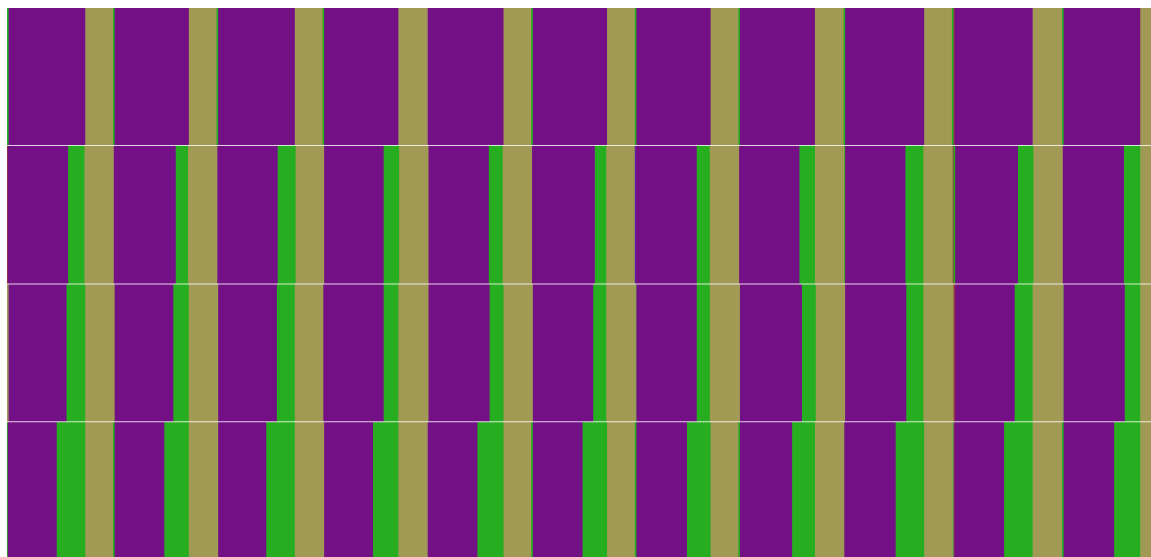


Figure 6.9: Trace graph of about 11 optimization iterations, generated by HPC-TOOLKIT [ABF<sup>+</sup>10]. The horizontal axis is the time axis. Four horizontal rows represent computation timing of four cores. The top row is for the first time subdomain, etc. One optimization iteration consists of three major blocks of one purple, one green, and one brown. The purple blocks are for forward computation, the brown blocks are for backward computation, and the green blocks are for waiting.

According to Figure 6.8, the objective function value is apparently convergent.



But the control variable, i.e. well rates, has yet to converge at iteration 200. At iteration 200, the optimized control variable from two methods are similar, but different due to different optimization trajectory, and plotted in Figure 6.10. This fixed step size will converge neither the gradient nor parallel-in-time gradient-type method even given more iterations by experiments. I show the iteration history only of the first 200 iterations in Figure 6.8 for the reason of fixed step sizes, as I stated in the beginning of this section. The point is clear that the parallel-in-time gradient-type method has significant potential to have a speed-ups compared to the commonly used gradient method in reservoir optimization. However, it needs to be equipped with an efficient step size choice algorithm and a globalization technique.

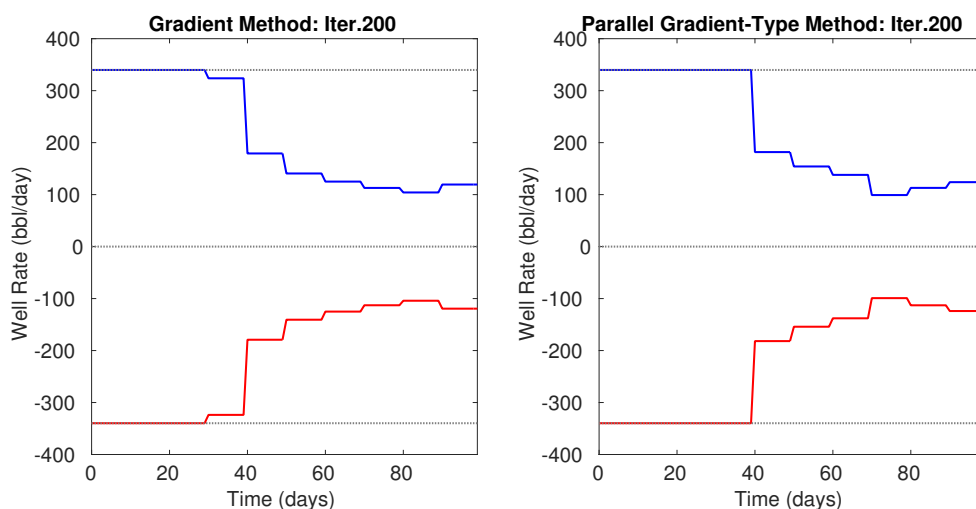


Figure 6.10: 2D reservoir example, optimized well rates at optimization iteration 200.

### 6.3.2.2 Example 2

In this example, I apply the parallel gradient-type method in a problem of 20 wells in a 3D reservoir. 4 time subdomains are used in the parallel gradient-type method and 4 cores in each time subdomains perform the computation in the forward/backward computation. In total, 2 level parallelism of 16 cores is utilized in parallel computing.

The parallel computation is run on a cluster using 16 cores in 2 computing nodes.

The model dimension is  $1200 \times 600 \times 10$  feet<sup>3</sup> discretized into  $60 \times 60 \times 5 = 18,000$  finite volume cells. Simulation time is 500 days with 1000 time marching steps of size 0.5 days. 10 injection wells and 10 production wells are placed in the middle layer of 5 feet depth. See Figure 6.11 and Figure 6.12 for the location of wells. Wellrates are piecewise constant functions with constant pieces of length 25 days.

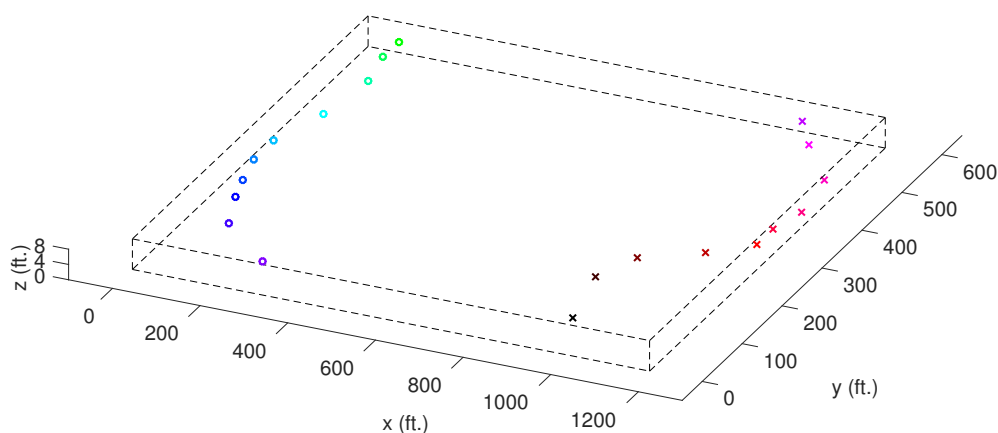


Figure 6.11: Location of injection and production wells. Circles and crosses are for injection and production wells respectively.

For the parallel-in-time gradient-type method, two dimensions of parallelism is exploited in this example. Computations on 4 time subdomains are executed at the same time in parallel; 4 cores are used in forward/backward computations in each of the 4 time subdomains. Totally,  $4 \times 4 = 16$  cores are computing in parallel.

Both gradient method and parallel gradient-type method run 50 iterations with no backtracking. Objective function values are evaluated after the optimization iterations finish by conducting full forward simulations for each wellrate schedule in the optimization trajectory.

In the iteration history in Figure 6.13, per optimization iteration, two method

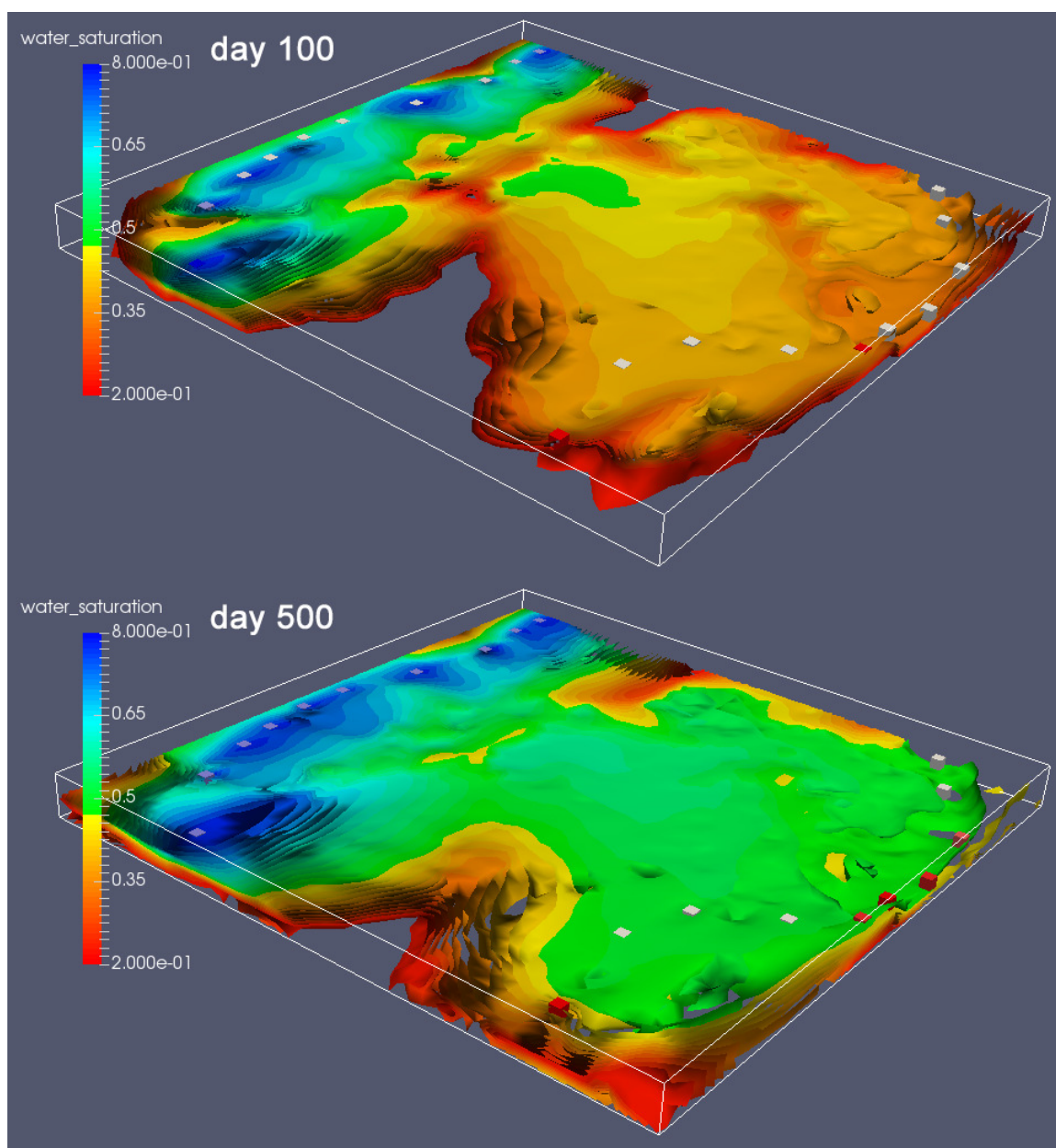


Figure 6.12: Contours at day 100 and 500 (final time) of water saturations resulting from optimized well-rates. Only the bottom half of the reservoir (depth 5-10 feet) is shown to better display water saturations at the 5 feet level, where all of the wells are located.

decreases objective function value similarly. The time consumed for each parallel gradient-type method iteration is roughly 1/4 of that for gradient method and therefore significant speed-ups are observed when objective function values are plotted against computation time.

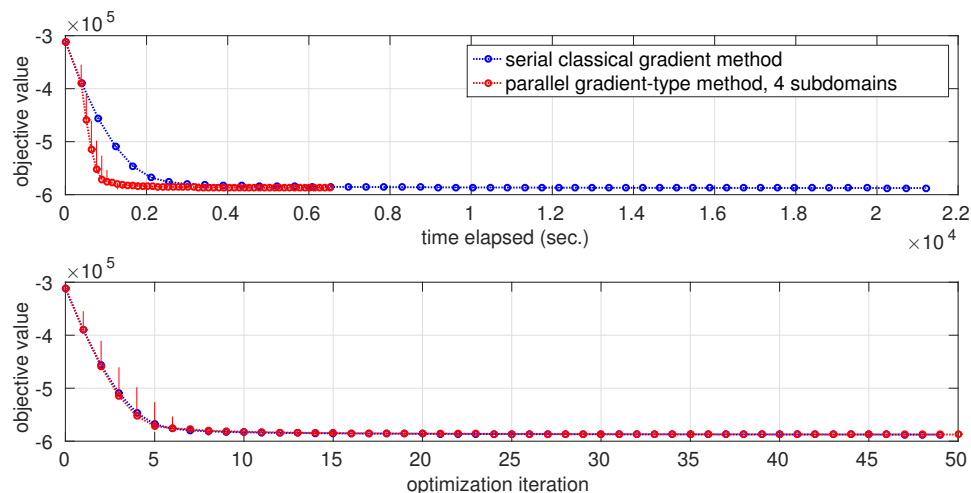


Figure 6.13: 3D reservoir example, iteration history of gradient method and parallel gradient-type method. Gradient method uses 4 cores in the linear solvers and preconditioners in the forward/backward solves; parallel gradient-type method uses 4 time subdomains in each of which 4 cores are used in the forward/backward solves, totally,  $4 \times 4 = 16$  cores are computing simultaneously for the parallel-in-time gradient-method. Parallel gradient-type method is initialized by a full gradient sweep and, as a result, the first step takes as long as a serial gradient step. There are vertical red bars on the parallel gradient-type objective values. The other end of bars indicate the infeasible “objective value” computed, for research purpose, during the parallel method runtime using the state variable with jumps at subdomain boundaries.

Figure 6.13 shows that 50 iterations of 4 time subdomain parallel gradient-type method take slightly longer than 1/4 of the running time of 50 iterations of the classical gradient method. This is due to the same reason of unbalanced computing

load as stated in the example in Section 6.3.2.1. The trace plot of the 16 cores is presented in Figure 6.14. The cores performing the forward computation of the first time subdomain take the longest time in every optimization iteration and fairly large blocks of time exist for other cores waiting for them. Adjusting the current evenly splitting of the time marching steps among the 4 time subdomains will likely improve the load balance. A test on using a time subdomain partition based on computation load, instead of even number of time steps, for load balance is included in Appendix D.

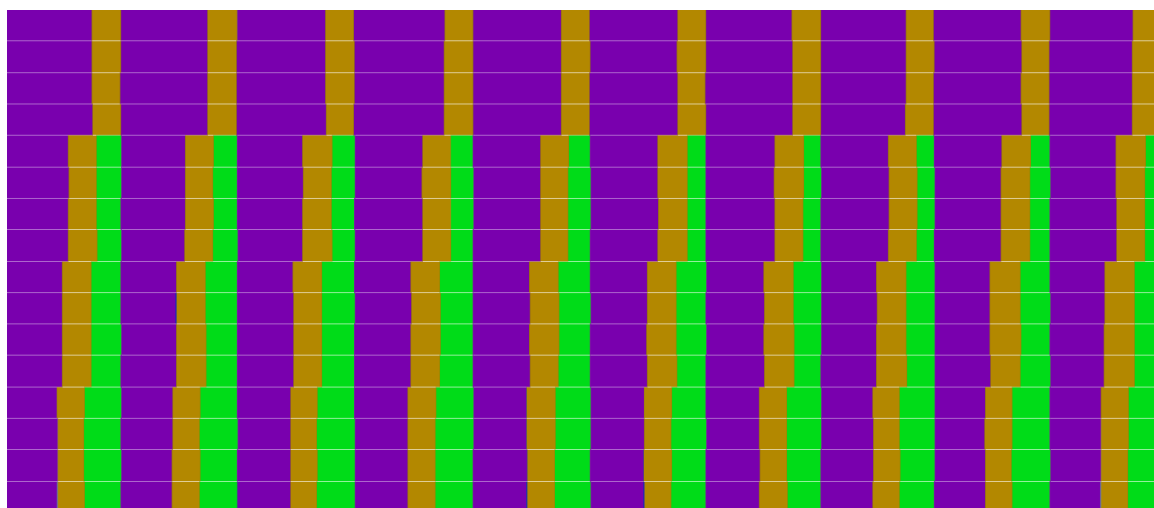


Figure 6.14: Trace graph of about 10 optimization iterations with evenly split time subdomains, generated by HPC`TOOLKIT` [ABF<sup>+</sup>10]. The horizontal axis is the time axis. From top to bottom, the 16 rows corresponding to 16 cores are grouped into 4 groups. The top 4 horizontal rows represent computation timing of the 4 cores performing parallel computing in the first time subdomain, etc. One optimization iteration consists of three major blocks of one purple, one brown, and one green block. Purple blocks are for forward computation, brown blocks are for backward computation, and green blocks are for waiting.

For this optimization problem, there are several local minimums and the controls within a region around a local minimum often gives similar objective function values even the well rates can be noticeably different. For the sake of step size being fixed, the

number of iterations is predetermined to be 50. The last iteration of both methods decreases the objective function less than 0.01% relative to the objective function value. Two methods produce similar control at iteration 50, see Figure 6.15.

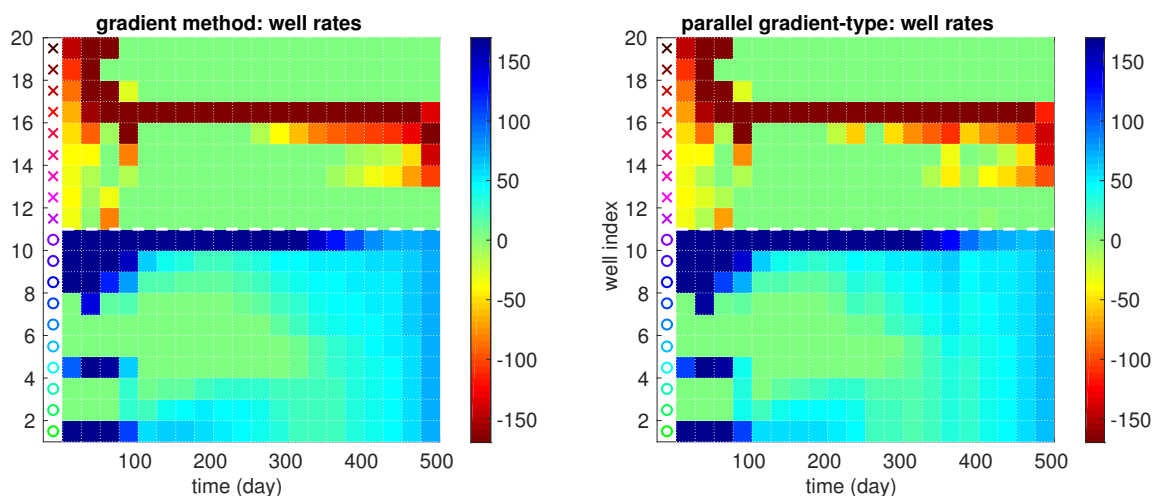


Figure 6.15: 3D reservoir example, optimized well rates at iteration 50.

The numerical state variables are nearly continuous for the parallel gradient-type method at iteration 50 in the sense that no significant jumps of the state of the discrete time steps at the time subdomain boundary are observed. In the numerical solution of a discretized in time system, the state difference between consecutive time marching steps can be used as a measure of continuity. In Figure 6.16, however, the change in state variable between two discrete time steps is still larger at the time subdomain boundaries than in the interior of subdomains at iteration 50. But as iteration goes further, the discontinuity at the time subdomain boundaries becomes not noticeable, i.e. in the same magnitude as between two time marching steps in the interior of a time subdomain, see Figure 6.17. The discontinuity of the curve at locations other than the subdomain boundaries is caused by the change in wellrates every 50 discrete steps and different wellrate setting leads to different rate of the state change from a time step to the next time step.

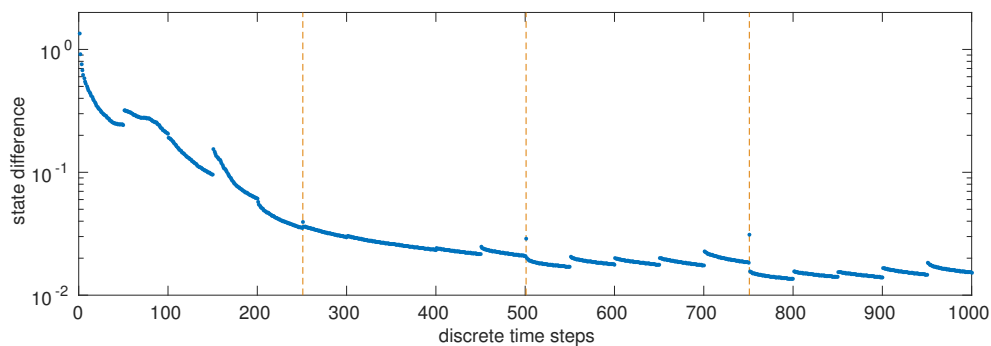


Figure 6.16: At parallel gradient-type method iteration 50, the 3 dashed lines mark the position of time subdomain boundaries. As a measure of discontinuity, the blue dots are computed by taking the norm of the difference of the states in the current time marching step and the state in the last time marching step.

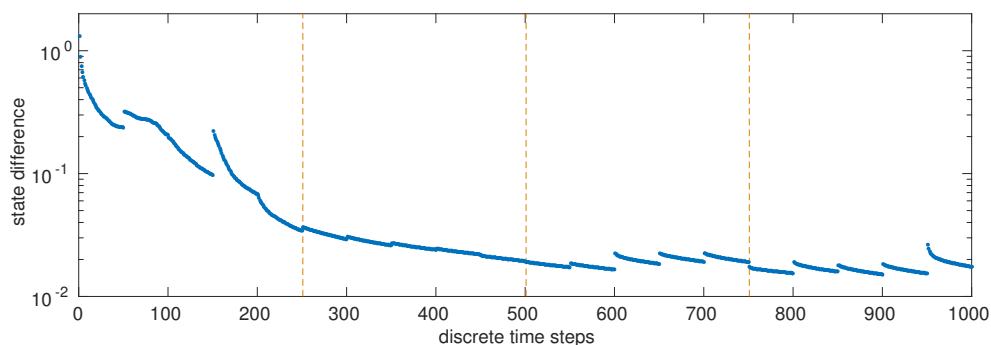


Figure 6.17: At parallel gradient-type method iteration 160, the 3 dashed lines mark the position of time subdomain boundaries. As a measure of discontinuity, the blue dots are computed by taking the norm of the difference of the states in the current time marching step and the state in the last time marching step.

### 6.3.3 A Heuristic Watchdog Type Parallel Algorithm

In this section, I introduce a heuristic watchdog type of algorithm as a modification of the proposed parallel-in-time gradient-type method for the two following reasons,

1. The reservoir water-flooding optimization problem does not necessarily meet all the assumptions for convergence in Chapter 5, e.g., of the reduced control space objective function, uniqueness of the stationary point to use Theorem 5.2.8 or convexity to use Theorem 5.2.9. The convergence is not guaranteed.
2. Need a method to adjust step size according to the algorithm behavior instead of manually empirically setting an assumed suitable fixed step size before the algorithm starts.

This watchdog type algorithm aims to provide a practical way to heuristically converge the parallel iterations by periodically sacrificing the parallelism and applying serial gradient step. In the serial gradient step, exact objective function corresponding to the current control iterates are obtained and used as a backtracking criteria. In addition, the serial gradient step also helps the state/adjoint information propagation across the whole time domain.

Although, no convergence proof is given for this watchdog type algorithm, numerical experiments show that it works well in the sense that the algorithm converges in test cases and the parallel speed-ups persists to some extent (see Figure 6.18) after adding the serial gradient step into the originally fully parallel algorithm.

**Algorithm.** The algorithm in a big picture is roughly represented as follows.

- serial gradient step
  - parallel-in-time gradient-type step
  - parallel-in-time gradient-type step ...
- serial gradient step, decide if to backtrack to previous step



- parallel-in-time gradient-type step
- parallel-in-time gradient-type step ...
- serial gradient step, decide if to backtrack to previous step ...

Pseudo code of the watchdog type algorithm is in Algorithm 16.

Algorithm 16 has an outer-inner two loops structure. In the inner loop (Line 11 to Line 14), parallel-in-time gradient-type control updates are performed, which is the major part of the algorithm. The outer loop iterates on the control variable  $u^{(j)}, j = 1, \dots, I_{\text{outer}}$ . The inner loop iterates on the tentative control variables,  $u^{(j,l)}, l = 1, \dots, I_{\text{inner}}$ , updated by the parallel-in-time gradient-type steps. The algorithm outside the inner loop ensures necessary additional forward and backward computation is executed so that exact objective function value and true gradient are obtained corresponding to the control variable as a result of the inner loop, based on which decision of accepting the tentative new control or returning a previous control is made in Line 19. Line 3 computes the exact objective value corresponding to the initial control  $u^{(0)}$ . Line 4 distributed the serial computation result to each of the cores whose computation involves these state variables. In the for-loop (Line 6) a series of backward computation is performed without state and control update. The  $N - 1$  backward computation steps in this for-loop and the first backward computation (Line 11) in the inner loop constitute a full backward sweep. Similarly, in the for-loop (Line 16) a series of forward computation is done without control update. These  $N - 1$  forward computation steps in this for-loop and the first backward computation (Line 14) in the inner loop constitute a full forward sweep which yields the exact objective value corresponding to  $u^{(j,l)}$  used in the objective value decrease condition of Line 19. The objective decrease condition test in Line 19 is heuristic without guarantee of convergence. If the test is passed, the outer loop iterate  $u^{(j)}$  is set to the result of the inner loop parallel-in-time gradient-type iterations result. Otherwise, If the test fails,  $u^{(j)}$  is not changed from  $u^{(j-1)}$ , step size is cut by  $\rho_{\text{def.}}$ ,

---

**Algorithm 16** A Heuristic Watchdog Type Parallel Algorithm,  $N$  time subdomains

---

```

1: Input  $u^{(0)}$ 
2: Set initial step size  $\alpha$ , inflation factor  $\rho_{\text{inf.}} \geq 1$ , deflation factor  $0 < \rho_{\text{def.}} < 1$ .
3: Execute serial full forward computation with  $u^{(0)}$  and obtain  $\hat{J}(u^{(0)})$ .
4: Distribute feasible state variables corresponding to  $u^{(0)}$ 
5: for  $j = 1, \dots, I_{\text{outer}}$  do
6:   for  $l = 1, \dots, N - 1$  do
7:     (*)Do parallel-in-time backward comp. and communicate adjoint
8:   end for
9:    $u^{(j,0)} = u^{(j)}$ 
10:  for  $l = 1, \dots, I_{\text{inner}}$  do
11:    (*)Do parallel-in-time backward comp. and communicate adjoint
12:    Compute gradient-type vector  $g$  and  $u^{(j,l)} = u^{(j,l-1)} - \alpha g$ 
13:     $\alpha = \rho_{\text{inf.}} \alpha$ 
14:    Do parallel-in-time forward comp. and communicate state
15:  end for
16:  for  $l = 1, \dots, N - 1$  do
17:    Do parallel-in-time forward computation and communicate state
18:  end for
19:  if  $\hat{J}(u^{(j,l)}) < \hat{J}(u^{(j-1)})$  then
20:     $u^{(j)} = u^{(j,l)}$ 
21:  else
22:     $u^{(j)} = u^{(j-1)}$ 
23:    Distribute the computed state/adjoint variables corresponding to  $u^{(j-1)}$ 
24:     $\alpha = \rho_{\text{def.}} \alpha$ 
25:  end if
26: end for

```

---

and the previously computed feasible state/adjoint variables corresponding to  $u^{(j-1)}$  is restored (Line 23) to memory which does not require any forward/backward computation. In this case, in the next outer iteration, the for-loop (Line 6) and the first ( $l = 1$ ) backward computation in Line 11 is not necessary (\* in the pseudo code to indicate this) since exactly the same computation has been done before and the result is restored in Line 23. To maintain the pseudo code readability, I did not write extra code to indicate that this unnecessary computation should be skipped.

This watchdog type Algorithm 16 is not fully parallel in the sense that the two for-loops in Line 6 and Line 16 are essentially performing serial computation on the whole time domain and consequently one can not expect strong scaling as in the previous experiments in Section 6.3.2.1 and Section 6.3.2.2.

I estimate the potential speed-ups of Algorithm 16 compared to the classical gradient method below. Consider a single outer iteration in the for-loop in Line 5.  $I_{\text{inner}}$  gradient-type control updates are executed in Line 12. If I assume each control update is comparable to a control update in classical gradient method in terms of decreasing objective value, what I compare the watch dog algorithm with is the computation time of  $I_{\text{inner}}$  iterations of classical gradient method which I denote as

$$I_{\text{inner}}T_{\text{serial}} \tag{6.3.1}$$

For the watchdog algorithm, in one for-loop in Line 5, one full forward/back computation consuming  $T_{\text{serial}}$  and  $I_{\text{inner}} - 1$  parallel computation in subdomains consuming  $(I_{\text{inner}} - 1)\frac{T_{\text{serial}}}{N}$  are executed, which sum up to

$$T_{\text{serial}} + (I_{\text{inner}} - 1)\frac{T_{\text{serial}}}{N} \tag{6.3.2}$$

By computing the ratio of the timing (6.3.1) over (6.3.2), the estimation is obtained

$$\text{speed-ups} \approx \frac{I_{\text{inner}}T_{\text{serial}}}{T_{\text{serial}} + (I_{\text{inner}} - 1)\frac{T_{\text{serial}}}{N}} = \frac{1}{\frac{1}{I_{\text{inner}}} + \frac{1}{N} - \frac{1}{I_{\text{inner}}N}} < \min(I_{\text{inner}}, N) \tag{6.3.3}$$

In Figure 6.18, I plot the estimated speed-up as a function of  $N$  and  $I_{\text{inner}}$ .

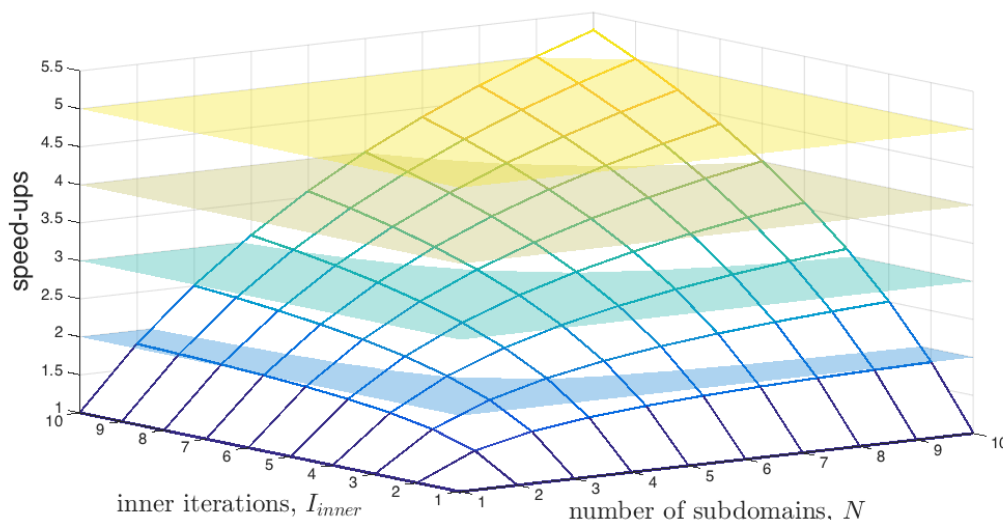


Figure 6.18: Speed-Up Estimation for Algorithm 16

To avoid unnecessary repeated backtracking steps, Algorithm 16 keeps the accepted step size from the previous iteration to use as the first tentative step size in the following computation, as opposed to returning to the predetermined (but maybe inappropriate, e.g., too large) step size once the test in Line 19 is passed.

**Numerical Experiments.** I demonstrate the performance of Algorithm 16 using the same wellrates optimization problem as in Section 6.3.2.2 by two tests.

In the first test, inflation factor  $\rho_{\text{inf}} = 1$ , the initial step size  $\alpha$  is set to be a suitable value, 0.03 in this case, and no backtracking is needed, illustrated in Figure 6.19. The watchdog-type parallel-in-time gradient-type method start the algorithm by the first iteration of a full forward/backward computation so that the state/adjoint subdomain boundary value can be set properly. From the plot, it can be seen the first iteration of the watchdog parallel-in-time gradient-type method takes about 4 times as long as the subsequent iterations, which uses about the same time as one iteration of, and also produces exactly the same control update as, the classical gradient method.

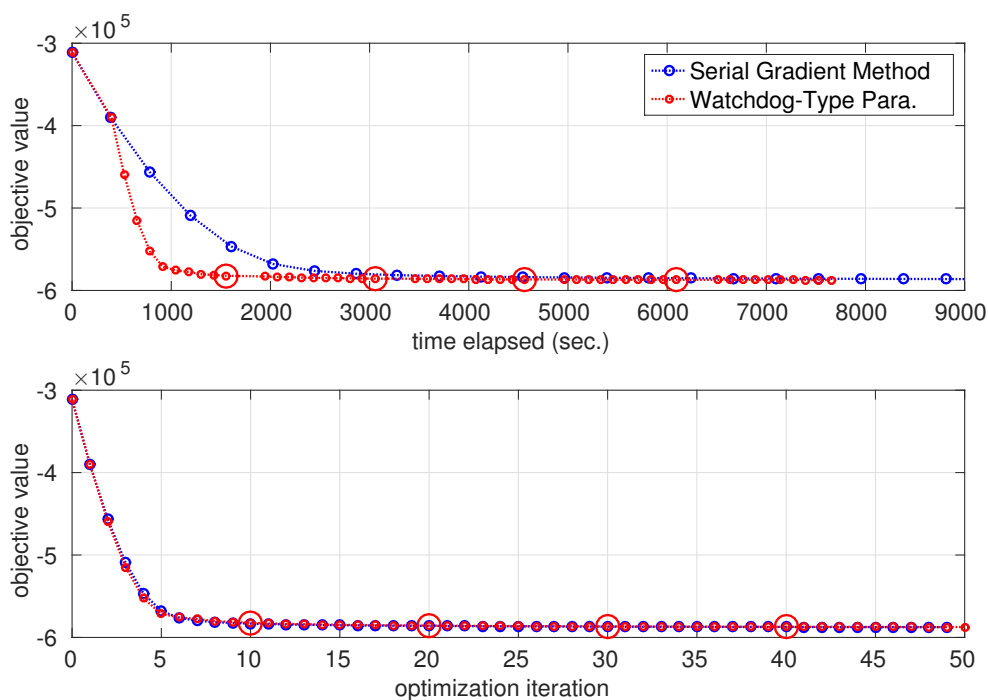


Figure 6.19: Watchdog-Type Algorithm with Suitable Initial Step Size. Initial step size  $\alpha = 0.03$ , inflation factor  $\rho_{\text{inf}} = 1$ . Serial Gradient Method is checking objective decrease condition (Line 19 in Algorithm 16) in each step. The watchdog-type parallel-in-time gradient-type method with  $I_{\text{inner}} = 10$  tests the decreasing condition every  $I_{\text{inner}} = 10$  steps, marked by the big red circle.

With  $I_{\text{inner}} = 10$ , the algorithm with tests the decreasing condition once every 10 iterations, marked by the big red circle. For example, at iteration 10, a full forward computation (for-loop in Line 16 in Algorithm 16) is executed so that the accurate objective function value corresponding to the current control iterates can be obtained and compared with that at iteration 0. At iteration 20, the same is done to compared with iteration 10, etc. Iteration-wise, two methods behaves similarly, demonstrated by the lower subplot. The serial gradient method finishes 50 iterations by 22730 seconds and the watchdog parallel-in-time gradient-type method uses 7655 seconds which brings 2.97 times speed-up. This agrees with the speed-up estimation yielded by the formula (6.3.3), i.e.,

$$\text{speed-ups} \approx \frac{1}{\frac{1}{I_{\text{inner}}} + \frac{1}{N} - \frac{1}{I_{\text{inner}}N}} = \frac{1}{\frac{1}{10} + \frac{1}{4} - \frac{1}{10 \times 4}} \approx 3.08$$

In the case, 4 time subdomains yields 3 times speed-ups, one can see some parallelism is sacrificed by the regularly applied serial gradient steps. I also tested the case where  $I_{\text{inner}} = 5$  instead of 10, then 2.45 times speed-up is measured in experiment against the 2.5 times speed-ups estimated by (6.3.3).

In the second test, with inflation factor  $\rho_{\text{inf}} = 1$ , deflation factor  $\rho_{\text{def}} = 0.5$ , I intentionally set the initial step size too large,  $\alpha = 0.45$  as opposed to the appropriate 0.03. The gradient method checks decrease condition (Line 19 in Algorithm 16) every iteration. In the first tentative gradient step, the decrease condition fails and consequently the tentative step is discarded and step size is halved. In a later iteration, step size is cut to 0.028 (slightly smaller than the step size 0.03 used in the first test) by successive backtracking and the iterations continues from this point with this suitable step size found. With  $I_{\text{inner}} = 5$ , the watchdog-type parallel algorithm checks the decrease condition every 5 iteration. Although, the first iteration yields an increase in the objective, it is not observed. The resulting control of the 5th iteration has a lower objective than the initial objective value and hence accepted. In the later iterations, it can be seen that, as a drawback of the watchdog-type algorithm, it also takes much more iterations than the classic gradient method to find a suitable step

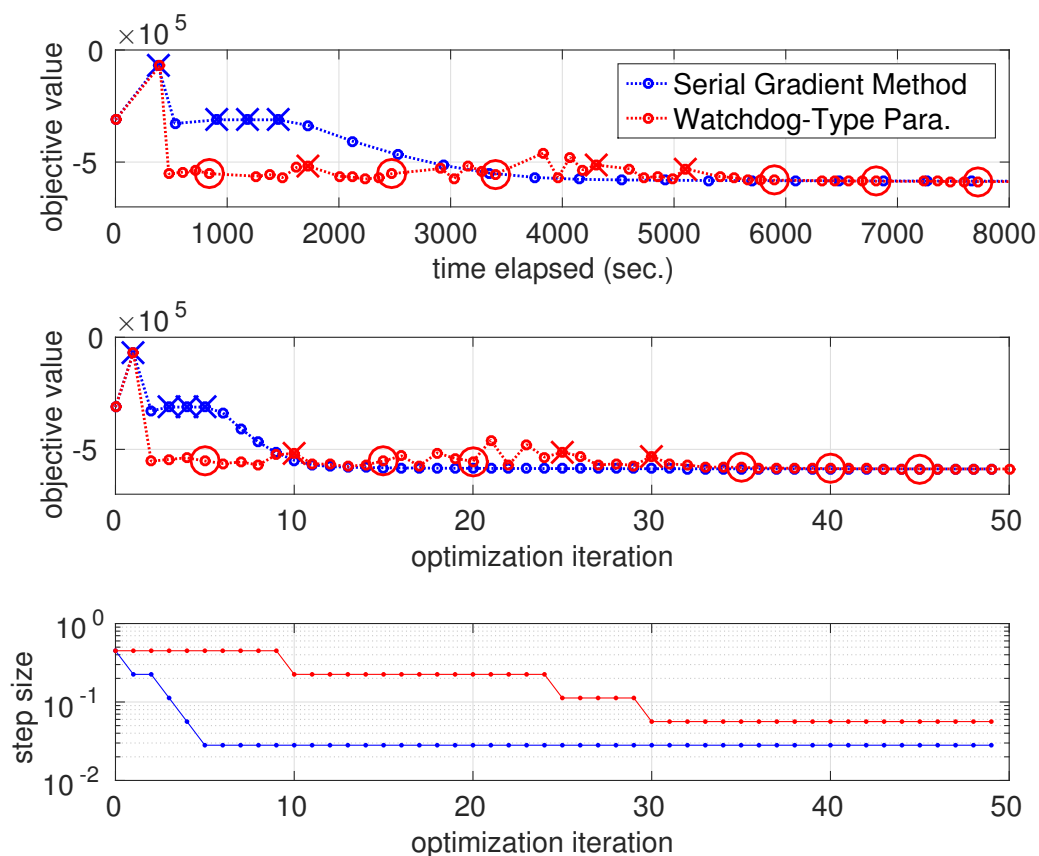


Figure 6.20: Watchdog-Type Algorithm with Large Initial Step Size. Initial step size  $\alpha = 0.45$ , inflation factor  $\rho_{\text{inf}} = 1$ , deflation factor  $\rho_{\text{def}} = 0.5$ . Serial Gradient Method is checking objective decrease condition (Line 19 in Algorithm 16) in each step. Crosses mark the iterations that fails the objective decrease condition test and thus halves the subsequent step sizes. The watchdog-type parallel-in-time gradient-type method with  $I_{\text{inner}} = 5$  tests the decreasing condition every 5 steps, marked by the big red circle (pass) or cross (fail).

size, since in the watchdog-type algorithm, there is not a mechanism to monitor the behavior of the iteration in each optimization step.

For completeness, I also include in Figure 6.21 an example with initial step size  $\alpha = 0.45$ , inflation factor  $\rho_{\text{inf}} = 1.02$ , deflation factor  $\rho_{\text{def}} = 0.5$ . Comparing this with Figure 6.20, one sees that the backtracking iteration has changed for the watchdog-type algorithm.

## 6.4 Summary

In this chapter, I presented numerical experiment results of the classical gradient method and the parallel-in-time gradient-type method applied to the reservoir water flooding optimization problem.

The reservoir water flooding optimization problem maximizes the oil reservoir recovery Net Present Value (NPV) by adjusting the wellrates of injection and production wells to manipulate the subsurface flow pattern. A two-phase immiscible incompressible reservoir subsurface flow model is used. Reservoir permeability and porosity data is from the SPE-10 dataset. This governing PDE system is highly nonlinear.

The parallel-in-time gradient-type method uses two levels of parallelism, the newly introduced parallelism in the time dimension and the existing parallelism in the space dimension, i.e., in linear solver/preconditioner. The speed-ups from two level of parallelism multiply.

In one example, the parallel-in-time gradient-type method with 4 time subdomains and 4 parallel process in space dimension in each subdomain (16 cores in total) is tested. Compared to the gradient method with 4 parallel process in space dimension, around 4 times speed-ups is achieved. In his case, the step size for both method is predetermined to be the same proper value according to knowledge from previous experiments.



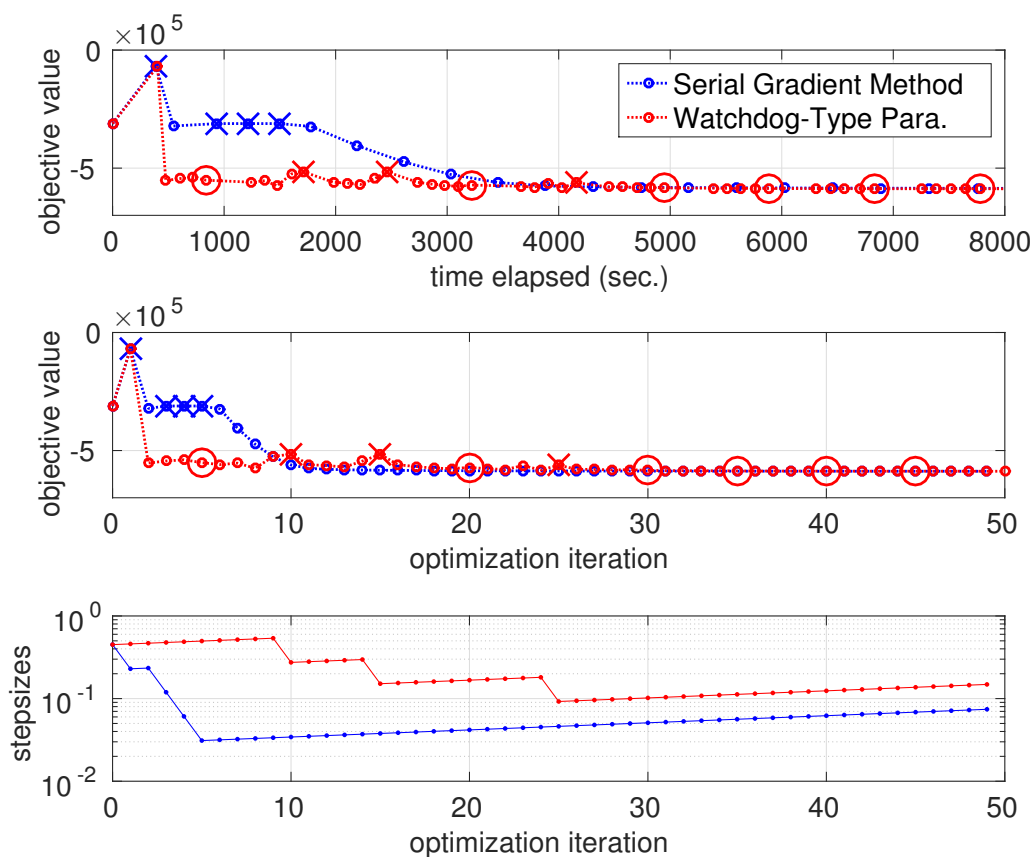


Figure 6.21: Watchdog-Type Algorithm with Large Initial Step Size. Initial step size  $\alpha = 0.45$ , inflation factor  $\rho_{\text{inf}} = 1.02$ , deflation factor  $\rho_{\text{def}} = 0.5$ . Serial Gradient Method is checking objective decrease condition (Line 19 in Algorithm 16) in each step. Crosses mark the iterations that fails the objective decrease condition test and thus halves the subsequent step sizes. The watchdog-type parallel-in-time gradient-type method with  $I_{\text{inner}} = 5$  tests the decreasing condition every 5 steps, marked by the big red circle (pass) or cross (fail).

In the parallel-in-time gradient-type iterations, because the state variables in memory is not feasible (there are jumps at time subdomain boundaries), there is not a good merit function to evaluate the control updates to adjust the iterations by backtracking, changing step size, etc. Without a good merit function, one can not dynamically adjust the iterations conveniently. Then, if there is no prior knowledge of what a good step size is, it is hard to set a proper step size at the beginning of the iterations. Too small step size leads to slow convergence. Too large step size leads to divergence.

As an attempt to dynamically adjust the parallel-in-time gradient-type method in the oil reservoir optimization application, I introduced a heuristic watchdog type of algorithm. This algorithm adds a full serial gradient sweep periodically in the parallel-in-time gradient-type method to obtain the feasible state corresponding to the current control iterates and thereby acquire the objective function value. Then, by the objective function value, the algorithm performs backtracking and step size adjustment.

Experiments show that this heuristic watchdog type of algorithm maintains a part of the parallel speed-ups in the parallel-in-time gradient-type method and in some cases is able to adjust an improper initial large step size to a suitable step size. However, compare to backtracking in the classical gradient method, this watchdog type of algorithm is less efficient in the sense that it takes more iterations to find a suitable step size because only in the periodically executed serial full gradient sweep the merit function is evaluated.

# Chapter 7

## Conclusions and Outlook

**Conclusions.** Using a gradient-type method to solve time-dependent optimization problem can be computationally expensive, especially the problem is governed by a time-dependent Partial Differential Equation (PDE) system [Gun03, BLUU12, Jan11, DCZ15, HCCT13]. The repeated solution of the inherently serial-in-time forward PDEs and the backward PDEs is required by the gradient computation and consumes a lot of computation time.

I proposed a new parallel-in-time gradient-type method for time-dependent optimal control problems to introduce parallelism in the time dimension. This method decomposes the time domain into multiple time subdomains. The forward and backward computation in these multiple time subdomains is performed at the same time in parallel using boundary information of state and adjoint variables from last optimization iteration. In the time consumed by one iteration of the classical gradient method, multiple iterations of the parallel-in-time gradient-type method can be executed. In the evenly splitting time domain into  $N$  subdomains case, ideally,  $N$  iterations of the parallel algorithm consumes similar time as one classical gradient method, which brings significant speed-up in example problems.

Then, by introducing a set of data aggregation/communication rules, I generalized the newly proposed parallel-in-time gradient-type method to accommodate more

flexible partition of the time domain, allowing overlapping subdomains and different number of forward/backward subdomains.

For linear-quadratic problems, I interpreted the generalized parallel-in-time gradient-type method as a multiple part splitting method [de 76, dN81]. I proved the convergence of the generalized parallel gradient-type method with sufficiently small fixed step sizes by the spectral radius arguments of an implicitly constructed iteration matrix.

I also showed that there is a close connection between the parallel-in-time gradient-type method and the multiple shooting reformulation of the optimization problem. The state/adjoint jumps, gradient-type vector in the parallel gradient-type method constitute the gradient of the multiple shooting reformulation Lagrangian. The proposed parallel-in-time gradient-type method is interpreted as a gradient type method to solve the indefinite optimality system of the multiple shooting reformulation of the optimization problem. An alternative convergence proof for the linear-quadratic problem is given from the multiple shooting perspective.

Then, for general nonlinear problems, I study the properties of the parallel gradient-type method combined with the metric projection of control variable onto a closed convex set. I derived a series of theorems to establish the convergence proof of the method with sufficiently small step size. Results on monotonic convergence are also derived.

I presented numerical experiment results. In a distributed control problem governed by a 3D linear advection-diffusion-reaction system, the parallel-in-time gradient-type method achieved strong scaling with 50-60 processors. Other numerical examples demonstrated different properties of the proposed method.

I introduced the wellrates optimization problem in oil reservoir optimization. I used a two-phase immiscible incompressible subsurface fluid model with SPE-10 permeability and porosity data sets, which forms a highly nonlinear problem. The parallel-in-time gradient-type method uses two levels of parallelism, the newly in-

roduced parallelism in the time dimension and the existing parallelism in the space dimension. The speed-ups from the two levels of parallelism multiply. Compared to the classical gradient method, the parallel-in-time gradient-type method leads to significant optimization acceleration.

**Outlook.** I proposed and investigated properties of the parallel-in-time gradient-type method in this thesis. I answered some questions, which also opens up more questions.

Below, as an outlook on possible future research, I summarize the aspects related to the parallel-in-time gradient-type method that I was not able to take care of during my research time. The aspects covered by different bullet points are not mutually exclusive.

- **Practical “sufficient decrease” criteria that guarantee convergence.**

The convergence theorems in Section 3.5, Section 4.3, and Chapter 5 all only state that for a sufficiently small step size the proposed algorithm converges. But they are not informative in terms of how small the step should be. In practice, these theorems are not convenient to use if not impossible. Consider the classical gradient method. The backtracking line search method based on Armijo-Goldstein “sufficient decrease” condition often efficiently leads to convergence by discarding unsatisfactory tentative control updates and adjusting step size. It is desired that an analogous mechanism can be designed for the parallel-in-time gradient-type method that monitors the iterations and make adaptive adjustments according to its behavior.

- **Determination of suitable problems.** Demonstrated by the examples in Section 3.6.1 and Section 3.6.2, the speed-up of the proposed parallel method compared to the classic gradient method heavily depends on the specific optimization problem it is applied to. The proposed method can have heterogeneous performance where it is applied to different problems. Definitely, the proposed

method does not work well on all problems. It will be helpful if there is a guideline of how to identify suitable optimization problems to use the parallel-in-time gradient-type method on.

Intuitively, optimal control problems in a very diffusive system with a Dirichlet boundary condition is likely to be a suitable type of problem to apply the proposed method.

- **How to partition the domain?** Apart from the choice of how many subdomains to use in the evenly splitting time domain case, the generalized parallel-in-time gradient-type method (Section 3.4) has offered a variety of supported computation subdomain arrangements, which I did not thoroughly study (an exploration is in Section 3.6.3). There should be a guideline of domain partitioning according to the nature of each specific optimization problem.

For example, in the problem with strong temporal dependence, overlapping subdomains should probably be encouraged.

- **Explanation of why the proposed method yields speed-ups.** I demonstrated by numerical examples in Section 3.6 and Section 6.3.2 that, in some optimization problems, the proposed method has a significant advantage over the classical gradient method, by utilizing parallel computing, in terms of computation time. However, I did not clearly explain why this advantage exists. I only have the heuristic argument, which also serves as the intuitive motivation of this whole research, that in the time for one classical gradient method iteration, the parallel-in-time gradient-type method can execute multiple iterations. If the gradient-type update is in some sense good enough, the fast iterations yields speed-ups. But I did not explain carefully how good the gradient-type vector the parallel method produces is in comparison with the exact gradient, especially in terms of its effect in achieving the optimization goal, i.e., decreasing the objective function value or control error. And I did not give reason why

the gradient-type vector is good in the case it is good. A better understanding of the reasons behind the observed speed-ups also helps answer the questions in other bullet points.

The material in Section 3.6.3 and Section A.2 can be a starting point of this discussion.

Further research that takes good care of these bullet points will potentially make the parallel-in-time gradient-type method a very practical method.

# Bibliography

- [A<sup>+</sup>88] H. Asheim et al. Maximization of water sweep efficiency by controlling production and injection rates. In *European Petroleum Conference*. Society of Petroleum Engineers, 1988.
- [ABF<sup>+</sup>10] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. Hpctoolkit: tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010.
- [ACG14] J. Asprion, O. Chinellato, and L. Guzzella. Efficient solution of the diesel-engine optimal control problem by time-domain decomposition. *Control Engineering Practice*, 30:34–44, 2014.
- [AGL07] J. E. Aarnes, T. Gimse, and K.-A. Lie. An introduction to the numerics of flow in porous media using Matlab. In G. Hasle, K.-A. Lie, and E. Quak, editors, *Geometrical Modeling, Numerical Simulation and Optimisation: Industrial Mathematics at SINTEF*, pages 261–302, Heidelberg, 2007. Springer-Verlag.
- [AN09] M. Asadollahi and G. Naevdal. Waterflooding optimization using gradient based methods. In *Paper number 125331-MS. SPE/EAGE Reservoir Characterization and Simulation Conference, 19-21 October 2009, Abu Dhabi, UAE*, 2009.



- [BB88] J. Barzilai and J. M. Borwein. Two-point step size gradient methods. *IMA J. Numer. Anal.*, 8(1):141–148, 1988.
- [BC64] R. H. Brooks and A.T. Corey. Hydraulic properties of porous media. Technical report, Colorado State University, Fort Collins, CO, 1964.
- [BDH16] D. Beigel, X. Deng, and M. Heinkenschloss. Parallel-in-time methods for optimal control. Technical report, Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005–1892, 2016. in preparation.
- [Ber99] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, Massachusetts, second edition, 1999.
- [BG99] G. Biros and O. Ghattas. Parallel preconditioners for KKT systems arising in optimal control of viscous incompressible flows. In *Proceedings of Parallel CFD '99, Williamsburg, VA, May 23–26, 1999*, Amsterdam, London, New-York, 1999. North Holland. <http://www.cs.cmu.edu/~oghattas>.
- [BH97] M. Berggren and M. Heinkenschloss. Parallel solution of optimal-control problems by time-domain decomposition. In M.-O. Bristeau, G. Etgen, W. Fitzgibbon, J. L. Lions, J. Periaux, and M. F. Wheeler, editors, *Computational Science for the 21st Century*, pages 102–112, Chichester, 1997. J. Wiley.
- [BH98] A. Battermann and M. Heinkenschloss. Preconditioners for Karush–Kuhn–Tucker systems arising in the optimal control of distributed systems. In W. Desch, F. Kappel, and K. Kunisch, editors, *Optimal Control of Partial Differential Equations*, Int. Series of Numer. Math. Vol. 126, pages 15–32, Basel, Boston, Berlin, 1998. Birkhäuser Verlag.
- [BLUU12] C. Brandenburg, F. Lindemann, M. Ulbrich, and S. Ulbrich. Advanced numerical methods for PDE constrained optimization with application to

- optimal design in Navier Stokes flow. In *Constrained optimization and optimal control for partial differential equations*, volume 160 of *Internat. Ser. Numer. Math.*, pages 257–275. Birkhäuser/Springer Basel AG, Basel, 2012.
- [Bro04] D. R. Brouwer. *Dynamic water flood optimization with smart wells using optimal control theory*. PhD thesis, Civil Engineering and Geosciences, Technical University Delft, Netherlands, 2004.
- [BS14] A. T. Barker and M. Stoll. Domain decomposition in time for PDE-constrained optimization. Technical Report MPIMD/11-08, Max Planck Institute Magdeburg, October 2014. Published as [BS15].
- [BS15] A. T. Barker and M. Stoll. Domain decomposition in time for PDE-constrained optimization. *Comput. Phys. Commun.*, 197:136–143, 2015.
- [CCL89] S.-C. Chang, T.-S. Chang, and P. B. Luh. A hierarchical decomposition for large-scale optimal control problems with parallel processing structure. *Automatica*, 25:77–86, 1989.
- [CG15] T. Carraro and M. Geiger. Direct and indirect multiple shooting for parabolic optimal control problems. In T. Carraro, M. Geiger, S. Körkel, and R. Rannacher, editors, *Multiple Shooting and Time Domain Decomposition Methods. MuS-TDD, Heidelberg, May 6-8, 2013*, volume 9 of *Contributions in Mathematical and Computational Sciences*, pages 35–67, Heidelberg, 2015. Springer-Verlag.
- [CGR14] T. Carraro, M. Geiger, and R. Rannacher. Indirect multiple shooting for nonlinear parabolic optimal control problems with control constraints. *SIAM J. Sci. Comput.*, 36(2):A452–A481, 2014.

- [CHM06] Z. Chen, G. Huan, and Y. Ma. *Computational methods for multiphase flows in porous media*. Computational Science & Engineering. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2006.
- [CJ86] G. Chavent and J. Jaffré. *Mathematical Models and Finite Elements for Reservoir Simulation: Single Phase, Multiphase and Multicomponent Flows through Porous Media*. Studies in Mathematics and its Applications. Elsevier, Amsterdam, New York, 1986.
- [CLR<sup>+</sup>12] C. Chen, G. Li, A. Reynolds, et al. Robust constrained optimization of short-and long-term net present value for closed-loop reservoir management. *SPE Journal*, 17(03):849–864, 2012.
- [Com05] A. Comas. *Time-Domain Decomposition Preconditioners for the Solution of Discretized Parabolic Optimal Control Problem*. PhD thesis, Department of Computational and Applied Mathematics, Rice University, Houston, TX, 2005. Available as CAAM TR06–01.
- [DCZ15] X. Deng, X. Cai, and J. Zou. A parallel space-time domain decomposition method for unsteady source inversion problems. *Inverse Probl. Imaging*, 9(4):1069–1091, 2015.
- [DCZ16] X. Deng, X.-C. Cai, and J. Zou. Two-Level Space–Time Domain Decomposition Methods for Three-Dimensional Unsteady Inverse Source Problems. *J. Sci. Comput.*, 67(3):860–882, 2016.
- [de 76] J. de Pillis.  $k$ -part splittings and operator parameter overrelaxation. *J. Math. Anal. Appl.*, 53(2):313–342, 1976.
- [DH16] X. Deng and M. Heinkenschloss. A parallel-in-time gradient-type method for discrete time optimal control problems. Technical report, Department of Computational and Applied Mathematics, Rice University, 2016.

- [dN81] J. de Pillis and M. Neumann. Iterative methods with  $k$ -part splittings. *IMA J. Numer. Anal.*, 1(1):65–79, 1981.
- [DSSS13] X. Du, M. Sarkis, C. E. Schaerer, and D. B. Szyld. Inexact and truncated parareal-in-time Krylov subspace methods for parabolic optimal control problems. *Electron. Trans. Numer. Anal.*, 40:36–57, 2013.
- [DTW71] J. E. Dennis Jr., J. F. Traub, and R. P. Weber. On the matrix polynomial, lamda-matrix and block eigenvalue problems. Technical Report CMU-CS-71-110, Computer Science Department, Carnegie Mellon University, 1971.
- [ECID11] D. Echeverria Ciaurri, O. J. Isebor, and L. J. Durlofsky. Application of derivative-free methodologies to generally constrained oil production optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 2(2):134–161, 2011.
- [EGH00] R. Eymard, T. Gallouët, and R. Herbin. Finite volume methods. In P. G. Ciarlet and J. L. Lions, editors, *Handbook of numerical analysis, Vol. VII*, *Handb. Numer. Anal.*, VII, pages 713–1020. North-Holland, Amsterdam, 2000.
- [EHM03] R. Eymard, R. Herbin, and A. Michel. Mathematical study of a petroleum-engineering scheme. *M2AN Math. Model. Numer. Anal.*, 37(6):937–972, 2003.
- [Eve03] G. Evensen. The Ensemble Kalman Filter: theoretical formulation and practical implementation. *Ocean Dynamics*, 53:343–367, 2003.
- [Fle05] R. Fletcher. On the Barzilai-Borwein method. In *Optimization and control with applications*, volume 96 of *Appl. Optim.*, pages 235–256. Springer, New York, 2005.

- [FR87] Z. Fathi and W. F. Ramirez. Optimization of an enhanced oil recovery process with boundary controls? a large-scale non-linear maximization. *Automatica*, 23(3):301–310, 1987.
- [FS12] D. C.-L. Fong and M. A. Saunders. Cg versus minres: An empirical comparison. *SQU Journal for Science*, 17(1):44–62, 2012.
- [Gan15] M. J. Gander. 50 years of time parallel time integration. In T. Carraro, M. Geiger, S. Körkel, and R. Rannacher, editors, *Multiple Shooting and Time Domain Decomposition Methods. MuS-TDD, Heidelberg, May 6-8, 2013*, volume 9 of *Contributions in Mathematical and Computational Sciences*, pages 69–113. Springer-Verlag, Heidelberg, 2015.
- [GMS06] P. E. Gill, W. Murray, and M. A. Saunders. User’s guide for snopt version 7: Software for large-scale nonlinear programming. ., 2006.
- [GO16] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2016.
- [Gun03] M. D. Gunzburger. *Perspectives in Flow Control and Optimization*. SIAM, Philadelphia, 2003.
- [HBH<sup>+</sup>05] 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.
- [HCCT13] F. Hwang, X. Cai, Y. Cheng, and C. Tsao. A parallel fully coupled implicit domain decomposition method for numerical simulation of microfluidic mixing in 3D. *Int. J. Comput. Math.*, 90(3):615–629, 2013.

- [Hei05] M. Heinkenschloss. A time-domain decomposition iterative method for the solution of distributed linear quadratic optimal control problems. *J. Comput. Appl. Math.*, 173(1):169–198, 2005.
- [Hei08] M. Heinkenschloss. Numerical solution of implicitly constrained optimization problems. Technical Report TR08–05, Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005–1892, 2008.
- [Hel76] D. Heller. Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems. *SIAM J. Numer. Anal.*, 13(4):484–496, 1976.
- [Jan11] J.-D. Jansen. Adjoint-based optimization of multi-phase flow through porous media: A review. *Computers & Fluids*, 46(1):40 – 51, 2011.
- [KDJA14] D. Kourounis, L. J. Durlofsky, J. D. Jansen, and K. Aziz. Adjoint formulation and constraint handling for gradient-based optimization of compositional reservoir flow. *Comput. Geosci.*, 18(2):117–137, 2014.
- [Kel99] C. T. Kelley. *Iterative Methods for Optimization*. SIAM, Philadelphia, 1999.
- [LBN<sup>+</sup>06] R. J. Lorentzen, A. Berg, G. Nævdal, E. H. Vefring, et al. A new approach for dynamic optimization of water flooding problems. In *Intelligent Energy Conference and Exhibition*. Society of Petroleum Engineers, 2006.
- [Lio71] J.-L. Lions. *Optimal Control of Systems Governed by Partial Differential Equations*. Springer Verlag, Berlin, Heidelberg, New York, 1971.
- [LMT01] J.-L. Lions, Y. Maday, and G. Turinici. Résolution d’edp par un schéma en temps “pararéel”. *Comptes Rendus de l’Académie des Sciences. Série I. Mathématique*, 332:1–6, 2001.

- [LRSV82] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli. The waveform relaxation method for time-domain analysis of large scale integrated circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 1(3):131–145, 1982.
- [Mic03] A. Michel. A finite volume scheme for two-phase immiscible flow in porous media. *SIAM J. Numer. Anal.*, 41(4):1301–1317 (electronic), 2003.
- [MST07] Y. Maday, J. Salomon, and G. Turinici. Monotonic parareal control for quantum systems. *SIAM J. Numer. Anal.*, 45(6):2468–2482 (electronic), 2007.
- [MT02] Y. Maday and G. Turinici. A parareal in time procedure for the control of partial differential equations. *Comptes Rendus de l’Académie des Sciences. Série I. Mathématique*, 335:387–392, 2002.
- [NW06] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Verlag, Berlin, Heidelberg, New York, second edition, 2006.
- [Pea77] D. W. Peaceman. *Fundamentals of Numerical Reservoir Simulation*. Elsevier Science Inc., New York, NY, USA, 1977.
- [Pol71] E. Polak. *Computational Methods in Optimization. A Unified Approach*. Academic Press, New York, London, Paris, San Diego, San Francisco, 1971.
- [PS75] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.
- [Ral96] D. Ralph. A parallel method for unconstrained discrete-time optimal control problems. *SIAM J. Optimization*, 6:488–512, 1996.
- [Ray93] M. Raydan. On the Barzilai and Borwein choice of steplength for the gradient method. *IMA J. Numer. Anal.*, 13(3):321–326, 1993.

- [Ray97] M. Raydan. The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem. *SIAM J. Optim.*, 7(1):26–33, 1997.
- [SAD05] P. Sarma, K. Aziz, and L. Durlafsky. Implementation of adjoint solution for optimal control of smart wells. In *Paper SPE 92864. Proc. SPE Reservoir Simulation Symposium, Houston, TX, USA (2005)*, 2005.
- [SY<sup>+</sup>01] B. Sudaryanto, Y. C. Yortsos, et al. Optimization of displacements in porous media using rate control. In *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers, 2001.
- [TBA86] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Trans. Automat. Control*, 31(9):803–812, 1986.
- [The16] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.9 edition, 2016.
- [Trö10] F. Tröltzsch. *Optimal Control of Partial Differential Equations: Theory, Methods and Applications*, volume 112 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2010.
- [US77] D. J. Uherka and A. M. Sergott. On the continuous dependence of the roots of a polynomial on its coefficients. *Amer. Math. Monthly*, 84(5):368–370, 1977.
- [Wie10] K. D. Wiegand. A numerical study of an adjoint based method for reservoir optimization. Master’s thesis, Department of Computational and Applied Mathematics, Rice University, Houston, TX, May 2010.
- [WLA<sup>+</sup>02] P. Wang, M. Litvak, K. Aziz, et al. Optimization of production operations in petroleum fields. In *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers, 2002.



- [Wri90] S. J. Wright. Solution of discrete-time optimal control problems on parallel computers. *Parallel Computing*, 16:221–237, 1990.
- [Wri91a] S. J. Wright. Parallel algorithms for banded linear systems. *SIAM J. Sci. Statist. Comput.*, 12:824–842, 1991.
- [Wri91b] S. J. Wright. Partitioned dynamic programming for optimal control. *SIAM J. Optimization*, 1:620–642, 1991.
- [YDA02] B. Yeten, L. J. Durlofsky, and K. Aziz. Optimization of smart well control. In *SPE-79031-MS SPE Conference Paper - 2002*, 2002.
- [ZAZP96] I. Zakirov, S. I. Aanonsen, E. S. Zakirov, and B. M. Palatnik. Optimizing reservoir performance by automatic allocation of well rates. In *5th European Conference on the Mathematics of Oil Recovery*, 1996.

# Appendices

# Appendix A

## On the Multiple-Shooting Formulation

### A.1 On the Convergence of the Projected Parallel Gradient-Type Method

In this section, I discuss the convergence of the parallel-in-time gradient-type algorithm combine with metric projection, a topic that is also covered in Section 5.2 for general nonlinear problem. Now, I investigate the convergence from the multiple shooting point of view for linear-quadratic problem.

In Section 4.3, the convergence proof heavily relies on the spectral radius of the matrix  $[\mathbf{I} - \mathbf{W}(\alpha)]$  in (4.3.3) being smaller than 1. It is proved that with sufficiently small  $\alpha$ ,  $\rho(\mathbf{I} - \mathbf{W}(\alpha))$  is smaller than 1. It is natural that one conjecture that with a  $\alpha$  such that  $\rho(\mathbf{I} - \mathbf{W}(\alpha)) < 1$ , the projected parallel-in-time gradient-type method iteration converges. Below, I first define the projected iteration in (A.1.2) and then use two counter examples to show the conjecture does not hold.

Given a convex set  $D \in n_u \times K$ , define projection

$$\mathcal{P}_{u,D} \begin{pmatrix} \bar{y} \\ \bar{u} \\ \bar{p} \end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix} \bar{y} \\ \mathcal{P}_D(\bar{u}) \\ \bar{p} \end{pmatrix}$$

note that  $\mathcal{P}_{u,D}$  is indeed a metric projection that projects the whole  $\bar{y}, \bar{u}, \bar{p}$  vector onto the convex set  $\mathbb{R}^{n_y \times K} \times D \times \mathbb{R}^{n_y \times K}$ , i.e.,

$$\mathcal{P}_{u,D} \begin{pmatrix} \bar{y} \\ \bar{u} \\ \bar{p} \end{pmatrix} = \underset{(\bar{y}', \bar{u}', \bar{p}') \in \mathbb{R}^{n_y \times K} \times D \times \mathbb{R}^{n_y \times K}}{\text{arg min}} \left\| \begin{pmatrix} \bar{y}' \\ \bar{u}' \\ \bar{p}' \end{pmatrix} - \begin{pmatrix} \bar{y} \\ \bar{u} \\ \bar{p} \end{pmatrix} \right\|$$

The parallel-in-time gradient-type method combined with metric projection executes the following iteration modified from (4.2.15),

$$\begin{pmatrix} \bar{y}^{(j+1)} \\ \bar{u}^{(j+1)} \\ \bar{p}^{(j+1)} \end{pmatrix} = \mathcal{P}_{u,D} \left( \begin{pmatrix} \bar{y}^{(j)} \\ \bar{u}^{(j)} \\ \bar{p}^{(j)} \end{pmatrix} - \begin{bmatrix} 0 & 0 & -I_{N-1} \\ 0 & \alpha I_K & 0 \\ -I_{N-1} & 0 & 0 \end{bmatrix} \nabla \mathcal{L} \left( \begin{pmatrix} \bar{y}^{(j)} \\ \bar{u}^{(j)} \\ \bar{p}^{(j)} \end{pmatrix} \right) \right) \quad (\text{A.1.1})$$

Then, using expression (4.3.2) and definition in (4.3.4) leads to

$$\begin{aligned} \begin{pmatrix} \bar{y}^{(j+1)} \\ \bar{u}^{(j+1)} \\ \bar{p}^{(j+1)} \end{pmatrix} &= \mathcal{P}_{u,D} \left( \begin{pmatrix} \bar{y}^{(j)} \\ \bar{u}^{(j)} \\ \bar{p}^{(j)} \end{pmatrix} - \begin{bmatrix} 0 & 0 & -I_{N-1} \\ 0 & \alpha I_K & 0 \\ -I_{N-1} & 0 & 0 \end{bmatrix} \mathbf{H}_{\mathcal{L}} \begin{pmatrix} \bar{y}^{(j)} - \bar{y}^* \\ \bar{u}^{(j)} - \bar{u}^* \\ \bar{p}^{(j)} - \bar{p}^* \end{pmatrix} \right) \\ &= \mathcal{P}_{u,D} \left( [\mathbf{I} - \mathbf{W}(\alpha)] \begin{pmatrix} \bar{y}^{(j)} \\ \bar{u}^{(j)} \\ \bar{p}^{(j)} \end{pmatrix} + \mathbf{W}(\alpha) \begin{pmatrix} \bar{y}^* \\ \bar{u}^* \\ \bar{p}^* \end{pmatrix} \right) \end{aligned} \quad (\text{A.1.2})$$

where  $\bar{y}^*, \bar{u}^*, \bar{p}^*$  is the optimal solution of the unconstrained problem.

To prove the convergence iteration (A.1.2), following are two fruitless attempts.

- It is tempting to prove a seeming general result that the iteration defined by

$$x_{k+1} = \mathcal{P}_D(Ax_k + b)$$

always converges with  $\rho(A) < 1$ , which however does not hold. And the example

$$D = \{[x_1, x_2]^T \in \mathbb{R}^2 | x_1 = x_2\}, A = \begin{bmatrix} 0.1 & 10 \\ 0 & 0.1 \end{bmatrix}, b = [0, 0]^T, x_0 = [1, 1]^T,$$

results in divergence.

- One may want to prove that with a small  $\alpha > 0$  such that  $\rho(\mathbf{I} - \mathbf{W}(\alpha)) < 1$ , the projected parallel-in-time gradient-type method defined by (A.1.1) or (A.1.2) is always convergent. This is wrong. A counter example follows.

Consider optimization problem

$$\begin{aligned} \min \quad & \frac{1}{2}(y_1^2 + y_2^2) + \frac{1}{2}(u_0^2 + u_1^2) \\ \text{subject to} \quad & y_0 = 0 \\ & y_1 = y_0 + u_0 \\ & y_2 = y_1 + u_1 \end{aligned} \tag{A.1.3}$$

Its adjoint equations read

$$\begin{aligned} p_1 &= y_2 \\ p_0 &= y_1 + p_1 \end{aligned}$$

Let the parallel-in-time gradient-type algorithm run with two time subdomains,

$$\begin{aligned} s_0^F &= 0, s_1^F = 1; \\ s_0^B &= 0, s_1^B = 0. \end{aligned} \tag{A.1.4}$$

In this simple case, it does not need to do any backward computation in the first time subdomain. Straightforward computation yields symmetric

$$\mathbf{H}_{\mathcal{L}} = \begin{bmatrix} 2 & 0 & 1 & -1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 2 & 0 \\ -1 & 1 & 0 & 0 \end{bmatrix}$$

On the other hand, obviously, the optimal solution of the problem (A.1.3) is

$$\bar{y}_1^* = 0, \bar{u}^* = [0, 0]^T, \bar{p}_0^* = 0.$$

By (4.3.3) and (4.3.4),

$$\begin{bmatrix} \bar{y}^{(j+1)} \\ \bar{u}^{(j+1)} \\ \bar{p}^{(j+1)} \end{bmatrix} = [I - \mathbf{W}(\alpha)] \begin{bmatrix} \bar{y}^{(j)} \\ \bar{u}^{(j)} \\ \bar{p}^{(j)} \end{bmatrix} \quad (\text{A.1.5})$$

with

$$\mathbf{W}(\alpha) \stackrel{\text{def}}{=} \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & \alpha & 0 & 0 \\ 0 & 0 & \alpha & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix} \mathbf{H}_{\mathcal{L}} \quad (\text{A.1.6})$$

When step size  $\alpha = 0.4$ , the spectral radius  $\rho(\mathbf{I} - \mathbf{W}(\alpha)) \approx 0.97723 < 1$  and, of course, the unconstrained iteration (4.2.15) converges. However, experiments with

$$D = \{0\} \times [-1, 1] \text{ or } D = \{1\} \times [-1, 1] \text{ or } D = [-1, 1] \times [-1, 1]$$

show that the projected iteration, (A.1.1) or (A.1.2), diverges.

By knowledge in Chapter 5, if the iteration is started by a pair of full forward/backward sweep so that an initial true state/adjoint is computed, then the projected iteration converges with sufficiently step size, but that is beyond the scope of this section.

## A.2 Behaviour of Spectral Radius $\rho(\mathbf{I} - \mathbf{W}(\alpha))$

In Section 4.3, from (4.3.3), it can be seen that the convergence speed of the parallel-in-time gradient-type method iteration (4.2.15) relies on the spectral radius of  $[\mathbf{I} - \mathbf{W}(\alpha)]$ . In this Section, I use numerical experiments to demonstrate some typical

relationship between number of subdomains  $N$ , step size  $\alpha$ , and the spectral radius  $\rho(\mathbf{I} - \mathbf{W}(\alpha))$ .

Use the one dimensional distributed control problem in Section 3.6.3 as an example. I compute  $\rho(\mathbf{I} - \mathbf{W}(\alpha))$  corresponding to different number of subdomains and step size  $\alpha$ .

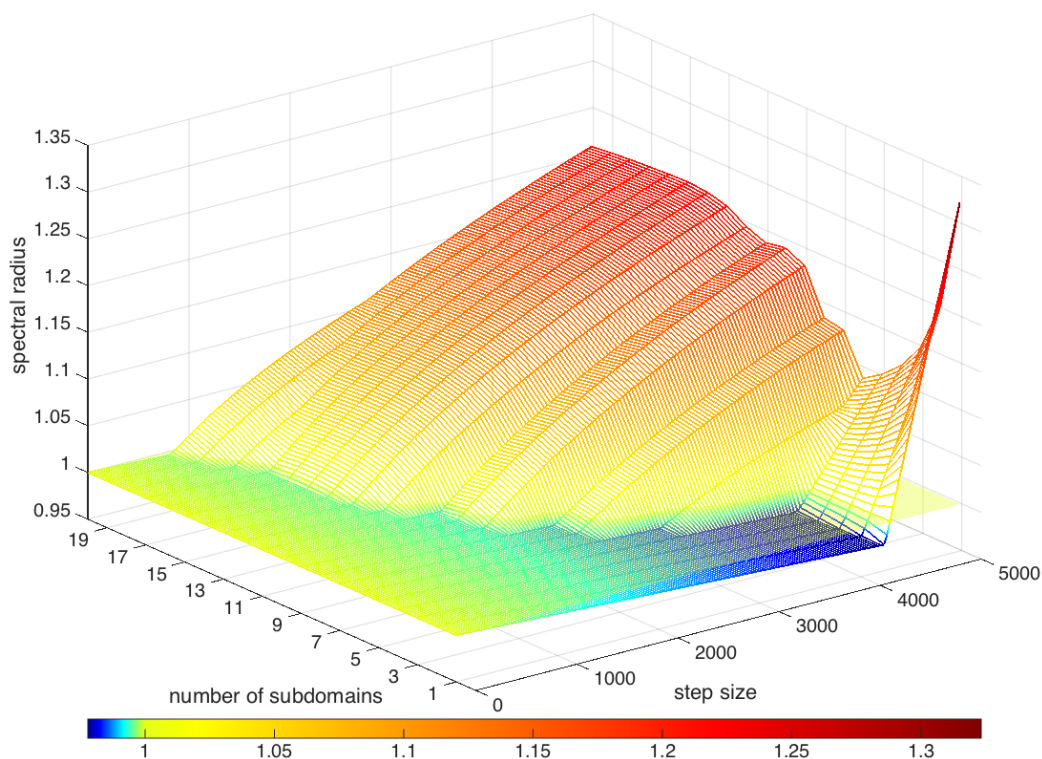


Figure A.1: Spectral Radius Against Step Size and Number of Subdomains. Two horizontal axes are for step size  $\alpha$  and number of subdomains. The vertical axis is for the spectral radius. A yellow transparent horizontal layer is added at  $z = 1$ . See Figure A.3 and Figure A.2 for two dimensional slices of this three dimensional plot.

Figure A.1 presents a overall relationship between the three quantities. Figure A.2 illustrates that for a fixed time subdomain splitting pattern, with sufficiently small

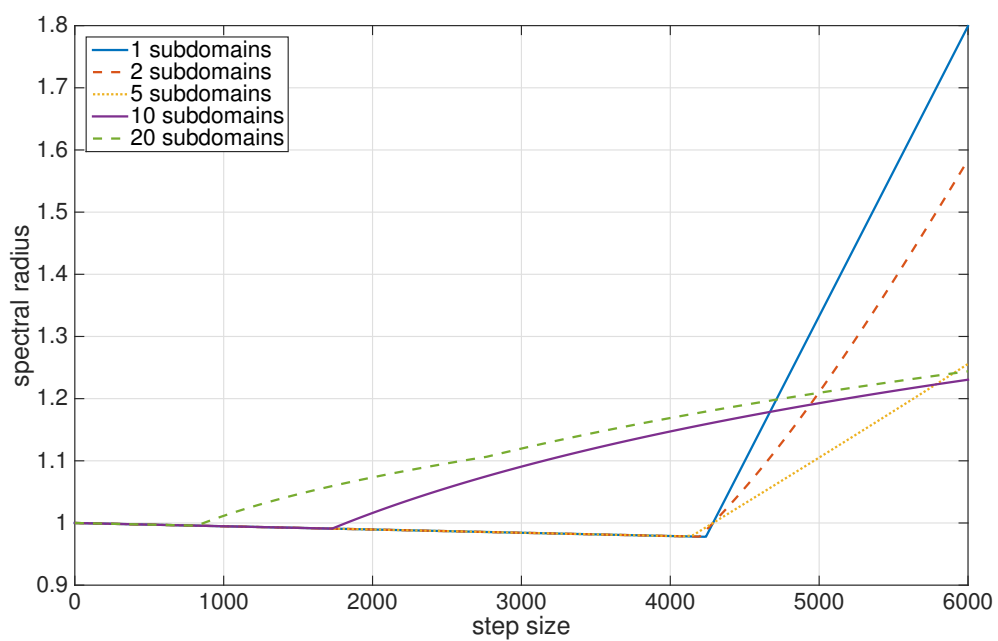


Figure A.2: Spectral Radius Against Step Sizes. Five curves represent five different number of subdomains.



step size  $\alpha$ , the spectral radius  $\rho(\mathbf{I}-\mathbf{W}(\alpha)) < 1$  as proved in Section 4.3. Additionally, increasing  $\alpha$  from 0 decreases the spectral radius almost linearly below a threshold. One should notice that in the classical serial gradient method iterations, the spectral radius of the corresponding iteration matrix  $\rho(\mathbf{I} - \alpha\mathbf{H})$  as seen in (3.1.10) decrease linearly as  $\alpha$  increases when

$$0 < \alpha < \alpha^* \stackrel{\text{def}}{=} \frac{2}{\lambda_{\max}(\mathbf{H}) + \lambda_{\min}(\mathbf{H})}$$

which is the behavior of the blue curve in Figure A.2. Actually, it is easy to see that, for the serial classic gradient method (blue curve),

$$\rho(\mathbf{I} - \alpha\mathbf{H}) = \begin{cases} 1 - \alpha\lambda_{\min}(\mathbf{H}) & 0 \leq \alpha \leq \alpha^* \\ \alpha\lambda_{\max}(\mathbf{H}) - 1 & \alpha > \alpha^* \end{cases}$$

which is a piecewise linear function with respect to  $\alpha$ . Another observation is that, in this example, as the number of subdomain grows, the step size corresponding to the minimum spectral radius as well as the spectral radius itself decreases.

Figure A.3 shows that with relatively small step sizes, when number of subdomains increases before a certain threshold is reached, the spectral radius is not significantly affected.

### A.3 A Parallel-In-Time Krylov Subspace Based Solver

In this section I show numerical results using a parallel Krylov subspace solver to solve the optimality system in multiple shooting formulation. The multiple formulation is also a permutation of the block tridiagonal optimality system developed in [Hei05]. This permutation reveals explicitly the structure that allows for parallel computation in left multiplying the matrix to a vector. I present both positive and negative results as for the performance of this parallel Krylov subspace solver. However, I do not provide theoretical explanation of the different speed-up results.

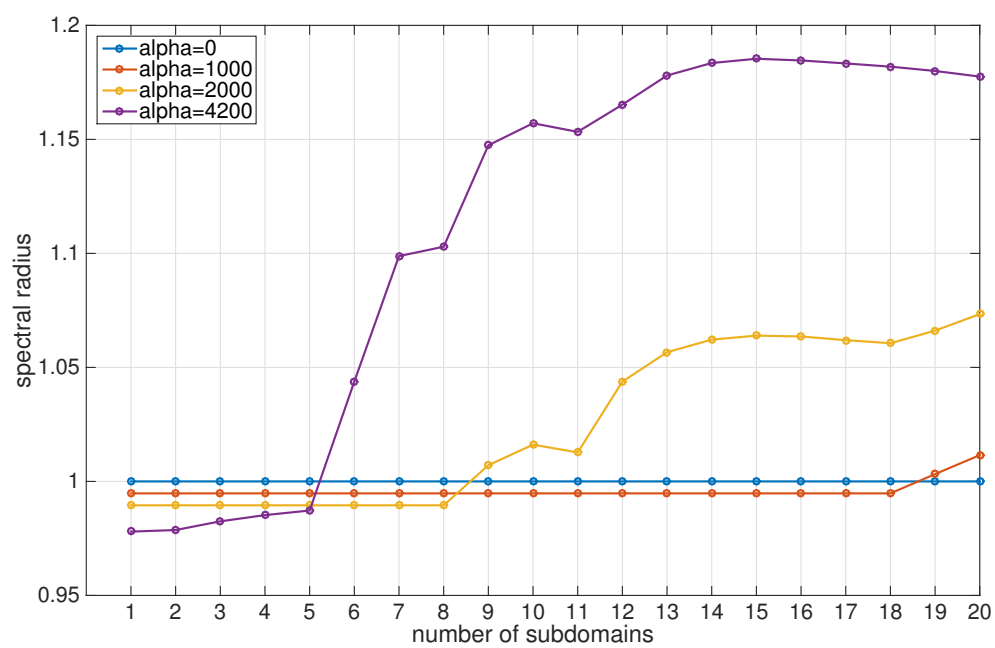


Figure A.3: Spectral Radius Against Number of Subdomains. Four curves represent four different step sizes.

In the context of linear-quadratic optimization problem, solving the saddle point problem (4.2.13) is equivalent to solve the linear system

$$\mathbf{H}_{\mathcal{L}} \begin{bmatrix} \bar{y} \\ \bar{u} \\ \bar{p} \end{bmatrix} = \nabla \mathcal{L} \left( \begin{bmatrix} \bar{y} \\ \bar{u} \\ \bar{p} \end{bmatrix} \right) - \nabla \mathcal{L} \left( \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) = -\nabla \mathcal{L} \left( \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) \quad (\text{A.3.1})$$

The linear operator  $\mathbf{H}_{\mathcal{L}}$  has the following important properties

- The application of the linear operator  $\mathbf{H}_{\mathcal{L}}$  to a vector can be executed in parallel with  $N$  parallel process in a  $N$ -subdomain multiple shooting setting. One should note that applying an explicitly constructed matrix to a vector can be trivially parallelized, but it is not always the case for linear operator whose matrix representation is not explicitly constructed as for  $\mathbf{H}_{\mathcal{L}}$ .
- The matrix representation of  $\mathbf{H}_{\mathcal{L}}$  is symmetric indefinite. The symmetry can be easily seen since  $\mathbf{H}_{\mathcal{L}}$  is the Hessian of  $\mathcal{L}$ .

To solve a linear system, gradient method is often not competitive against Krylov subspace methods. Since the application of the linear operator  $\mathbf{H}_{\mathcal{L}}$  to a vector is parallel, the Krylov subspace methods can be executed in parallel. In this section, I explore the performance of Krylov subspace based method in solving the optimality system written in the multiple shooting form (A.3.1).

I test solving the symmetric indefinite system (A.3.1) using Krylov subspace solver MINRES[PS75][FS12], which I refer to as the “parallel Krylov subspace solver” below. For solving symmetric positive definite system as in the serial optimality system,

$$\mathbf{H}u = -\mathbf{g}$$

with  $\mathbf{H}$  and  $\mathbf{g}$  defined in (3.1.8), MINRES and Conjugate Gradient method uses similar number of iterations. Three examples are used to demonstrate the performance of the parallel Krylov subspace solver.

1. Simple example with a scalar state at each time step.

$$\begin{aligned} \min \quad & \frac{1}{2} \sum_{k=1}^K y_k^2 + \frac{r}{2} \sum_{k=0}^{K-1} u_k^2 \\ \text{subject to} \quad & y_{k+1} = \rho y_k + u_k \\ & y_0 = 1 \end{aligned} \tag{A.3.2}$$

where  $K = 1000, r = 100, \rho = 0.99$ .

2. The 1D boundary control problem (3.6.1) in Section 3.6.1.
3. The 1D distributed control problem (3.6.5) in Section 3.6.3. Figure A.4 explicitly plots the sparsity pattern of the implicitly constructed matrix  $\mathbf{H}_{\mathcal{L}}$  for this test case.

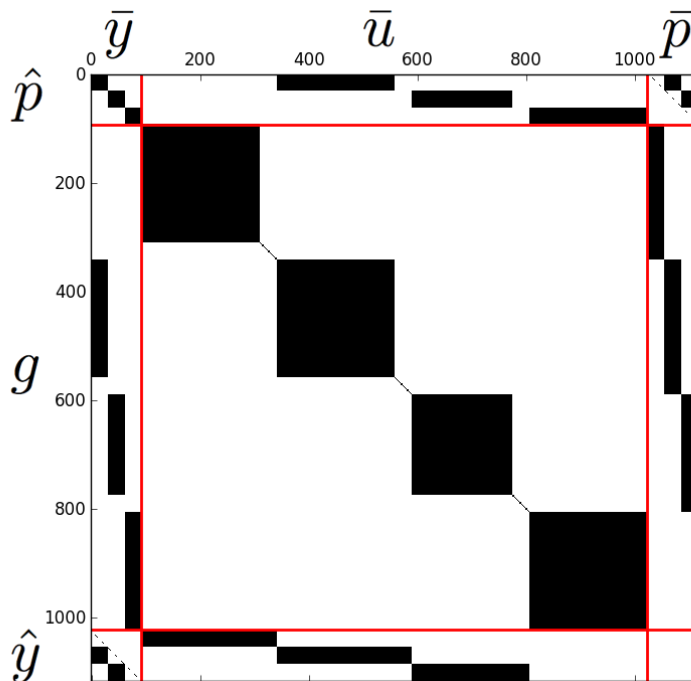


Figure A.4: sparsity pattern of the  $\mathbf{H}_{\mathcal{L}}$ ,  $N = 4$  subdomains, for test case 2 ((3.6.5) in Section 3.6.3 where  $n_u = n_y = 31, K = 30$ )

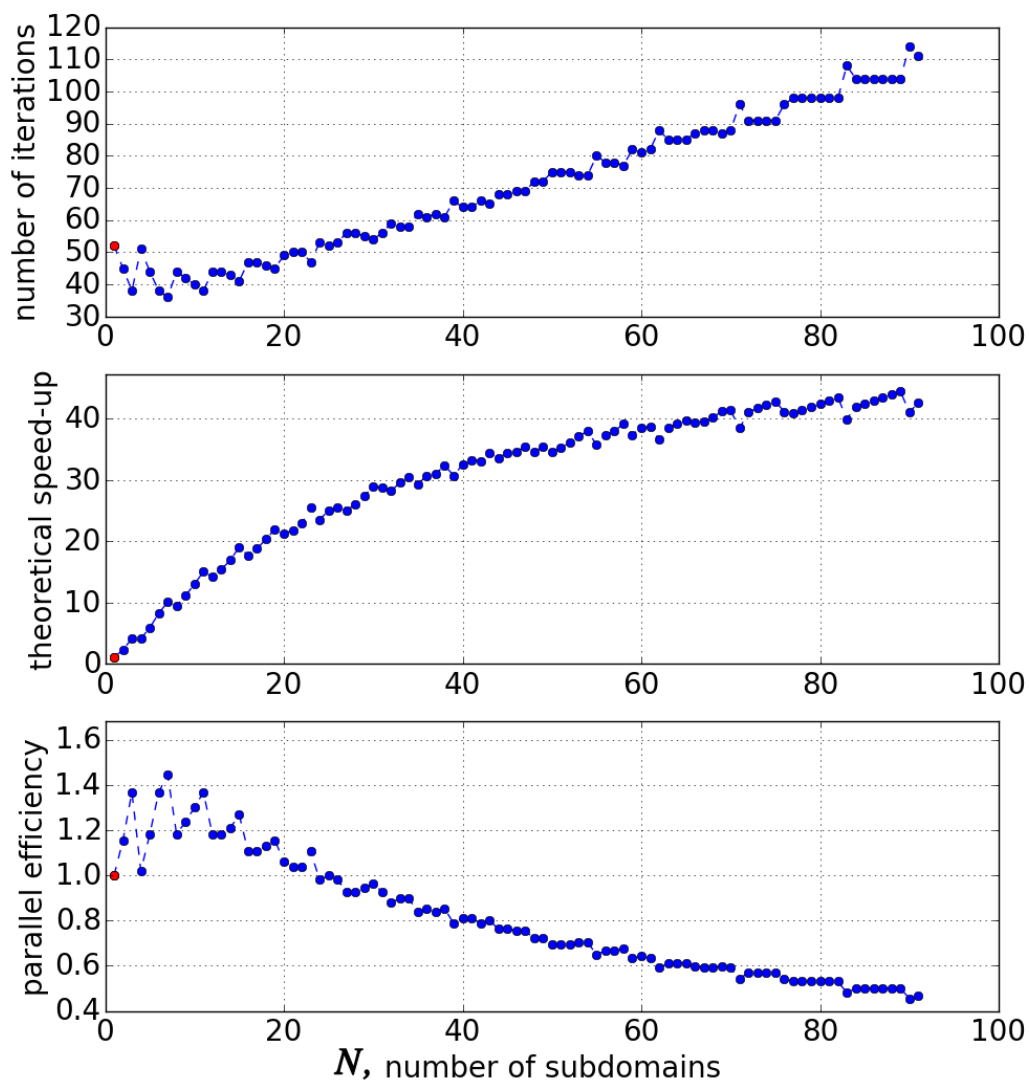


Figure A.5: Performance of the parallel Krylov subspace solver on problem (A.3.2). The red dot represent the serial bench mark. The horizontal axes are number of subdomains. The Top plot shows the number of iterations,  $I(N)$ , to reduce control error from initial  $1e - 1$  to  $1e - 12$ . Theoretical speed-up is  $N \cdot I(1)/I(N)$ . Parallel efficiency is  $I(1)/I(N)$ .

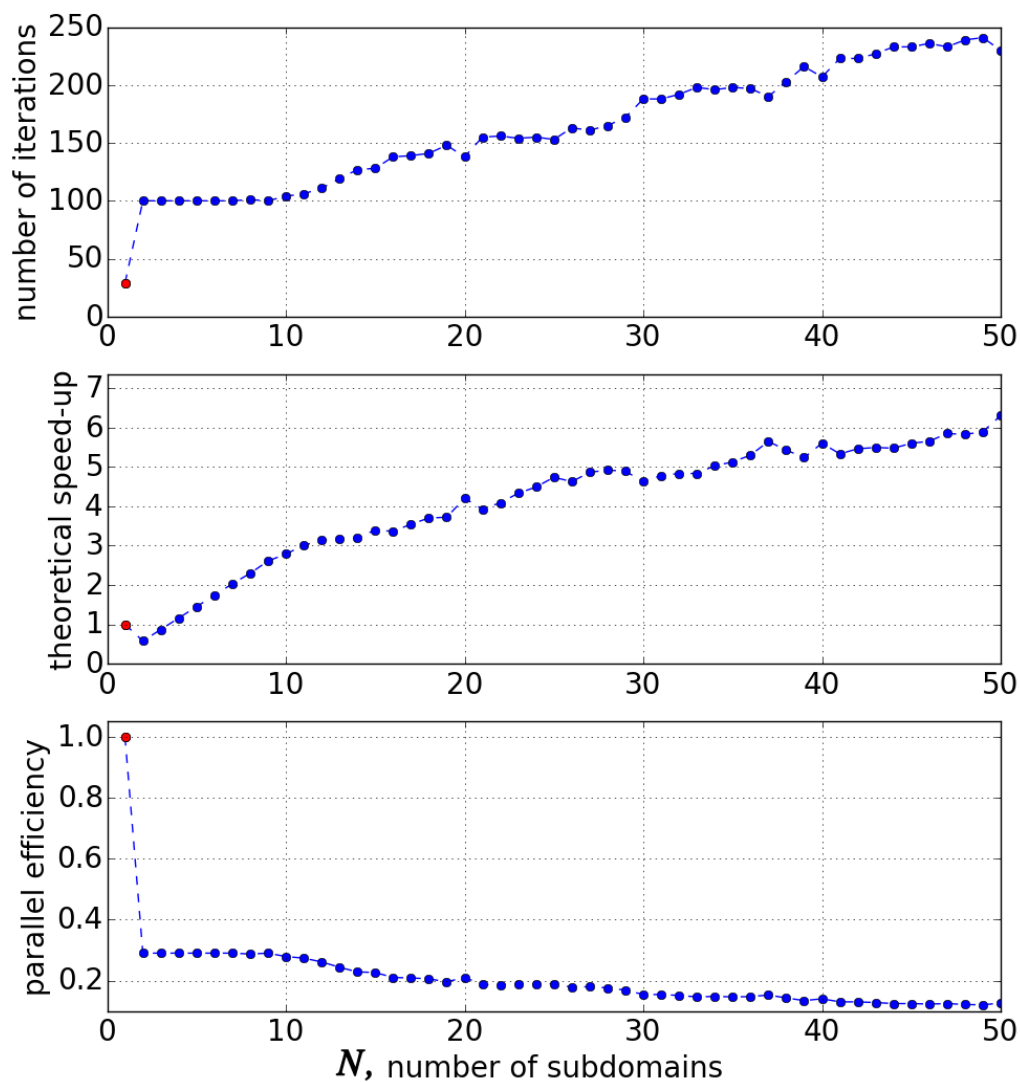


Figure A.6: Performance of the parallel Krylov subspace solver on problem (3.6.1). The red dot represent the serial bench mark. The horizontal axes are number of subdomains. The Top plot shows the number of iterations,  $I(N)$ , to reduce control error from initial  $1e1$  to  $1e - 9$ . Theoretical speed-up is  $N \cdot I(1)/I(N)$ . Parallel efficiency is  $I(1)/I(N)$ .

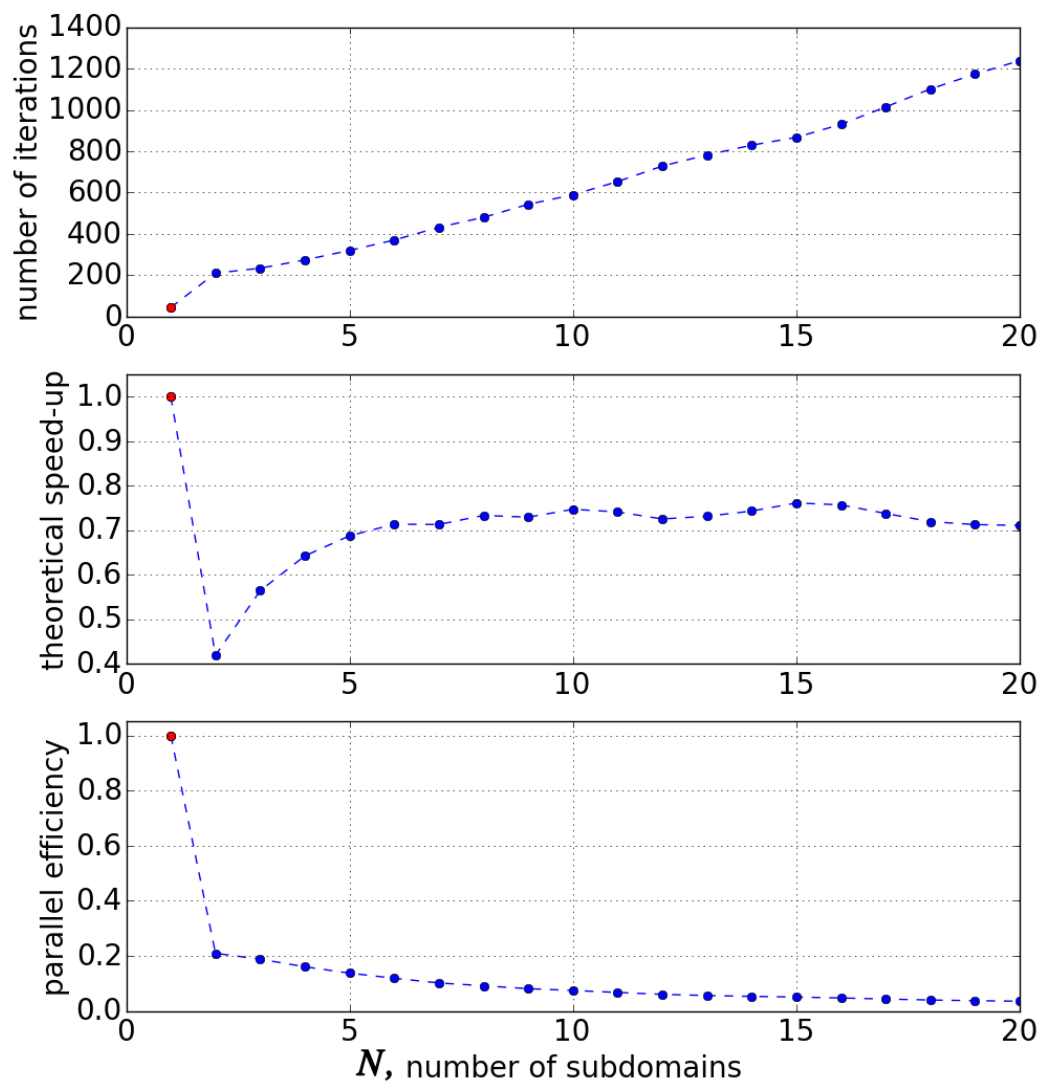


Figure A.7: Performance of the parallel Krylov subspace solver on problem (3.6.5). The red dot represent the serial bench mark. The horizontal axes are number of subdomains. The Top plot shows the number of iterations,  $I(N)$ , to reduce control error from initial  $1e2$  to  $1e - 5$ . Theoretical speed-up is  $N \cdot I(1)/I(N)$ . Parallel efficiency is  $I(1)/I(N)$ .

Figure A.5, Figure A.6, and Figure A.7 present contrasting numerical results for three test cases. In Figure A.5 for problem (A.3.2), it is observed that for less than 20 subdomains, the parallel algorithm takes less iteration than the serial algorithm, in which case the scaling is better than strong scaling with parallel efficiency larger than 100%. In Figure A.6 for problem (3.6.1), by introduction of the parallelism, number of iterations immediately tripled from around 30 to 100 and stayed roughly constant with less than 10 subdomains with parallel efficiency around 30%. In Figure A.7 for problem (3.6.5), parallelism does not yield any speed-up in all the tests with various number of subdomains.

However, I do not have explanation for the different parallel performance in the three test cases currently.



# Appendix B

## Reservoir Simulation

To do a reservoir simulation is to simulate the behavior of reservoir variables, such as pressure, saturation, etc. , by a math model based computer program. I summarize the basic reservoir physics in the PDE model and explain the numerical scheme to do the simulation, which follows the framework of [CHM06], [CJ86], and [Pea77].

### B.1 Basic Physics

In this project, I use a two-phase, immiscible, incompressible flow model [AGL07] [CHM06]. There is a water phase and a oil phase. Being immiscible means that no mass transfer between phases. The water phase consists of pure water; the oil phase consists of pure oil. Being incompressible means rock, water and hydrocarbon are all incompressible, i.e. rock porosity and fluid density does not change due to the pressure change.

---

$\Omega \subset \mathbb{R}^3$	open set representing the dimension of the reservoir
$x \in \Omega$	space coordinate of 3-dimensional reservoir
$T \in \mathbb{R}^+$	total simulation time
$t \in [0, T]$	time after the beginning of simulation
$\alpha \in \{\text{water(w),oil(o)}\}$	phase in the composite fluid

$\mathbf{K}(x) \in \mathbb{R}^{3 \times 3}$	permeability matrix at location $x$
$\phi(x) \in \mathbb{R}$	reservoir porosity
$\rho_\alpha \in \mathbb{R}$	density of phase $\alpha$ , assumed to be constant all space all time
$s_\alpha(x, t) \in \mathbb{R}$	saturation of phase $\alpha$ . $\forall x, \forall t, s_w(x, t) + s_o(x, t) = 1$
$p_\alpha(x, t) \in \mathbb{R}$	pressure of phase $\alpha$
$p_{cow}(s_w) \in \mathbb{R}$	capillary pressure, $p_o - p_w = p_{cow}$
$p(x, t) \in \mathbb{R}$	global pressure, defined in (B.1.5)
$q_\alpha(x, t) \in \mathbb{R}$	well (mass) rate of phase $\alpha$ at location $x$ at time $t$
$q(x, t) \in \mathbb{R}$	total well (volume) rate, i.e. $\frac{q_w(x, t)}{\rho_w} + \frac{q_o(x, t)}{\rho_o}$
$v_\alpha(x, t) \in \mathbb{R}^3$	flux of phase $\alpha$ , volume flow rate per unit area
$v(x, t) \in \mathbb{R}^3$	total flux, i.e. $v_w(x, t) + v_o(x, t)$
$k_{r\alpha}(s_w) \in \mathbb{R}$	relative permeability coefficient for phase $\alpha$ , fully dependent on the local phase saturation
$\mu_\alpha \in \mathbb{R}$	viscosity of phase $\alpha$ , assumed to be constant all space all time
$\lambda_\alpha(s_w) \in \mathbb{R}$	phase mobility, defined as $\frac{k_{r\alpha}(s_w)}{\mu_\alpha}$
$\lambda(s_w) \in \mathbb{R}$	total monility, i.e. $\lambda_w(s_w) + \lambda_o(s_w)$
$f_w(s_w) \in \mathbb{R}$	fractional flow function, $f_w(s_w) = \frac{\lambda_w(s_w)}{\lambda(s_w)}$
$G \in \mathbb{R}^3$	vector of gravity acceleration, pointing downward
$s_{or} \in \mathbb{R}^+$	irreducible oil saturation i.e. the assumed lowest oil saturation that can be achieved by a certain kind of recovery
$s_{wc} \in \mathbb{R}^+$	connate water saturation i.e. the saturation of water trapped permanently in the rock
$P_d \in \mathbb{R}^+$	entry pressure, used in Brooks-Corey formula

Table B.1: Nomenclature for the Reservoir Simulation

**Derivation of Pressure Equation** Water saturation  $s_w(x, t)$  and oil saturation  $s_o(x, t)$  describe the volume fraction of water and oil phase in the mixed fluid. In the

two phase model  $s_w(x, t) + s_o(x, t) = 1$ . Phase density  $\rho_w, \rho_o$  define the density of each fluid phase and, due to the incompressibility assumption, they are constant in all space and time of simulation. Porosity  $\phi(x)$  is the the ratio of porous medium pore space and the bulk volume of the rock. For the incompressibility assumption,  $\phi(x)$  is a function of position and invariant with respect to time. I use  $v_w(x, t), v_o(x, t)$  to denote phase flux/velocity, in the dimension of volume per area per time  $[\frac{L^3}{L^2T} = \frac{L}{T}]$ .  $q_w(x, t)$  and  $q_o(x, t)$  is the rate of fluid phase mass injected/produced in a unit volume  $[\frac{M}{TL^3}]$ . By mass conservation of two phases, for  $\alpha \in \{o, w\}$ ,

$$\frac{\partial \rho_\alpha \phi(x) s_\alpha(x, t)}{\partial t} + \nabla_x \cdot (\rho_\alpha v_\alpha(x, t)) = q_\alpha(x, t) \quad (\text{B.1.1})$$

Rock absolute permeability  $\mathbf{K}(x) \in \mathbb{R}^{3 \times 3}$  defines the flow direction in the presence of pressure gradient, each component of which has dimension  $[L^2]$ . Relative permeability  $k_{r\alpha}(s_\alpha(x, t))$  describes the influence of phase saturation on the phase velocity magnitude, dimensionless. Both absolute and relative permeability affects the phase flow through Darcy's Law

$$v_\alpha(x, t) = -\mathbf{K}(x) \frac{k_{r\alpha}(s_\alpha(x, t))}{\mu_\alpha} (\nabla_x p_\alpha(x, t) - \rho_\alpha G)$$

where  $\mu_\alpha$  represents phase viscosity in the dimension of  $[\frac{F}{T}]$ .  $G$  is the vector of gravitational acceleration  $[\frac{L}{T^2}]$ . Define phase mobility

$$\lambda_\alpha(s_\alpha(x, t)) = \frac{k_{r\alpha}(s_\alpha(x, t))}{\mu_\alpha}$$

and total velocity

$$v(x, t) := v_o(x, t) + v_w(x, t) \quad (\text{B.1.2})$$

and the source term of total volume rate per space in the dimension of  $[\frac{1}{T}]$

$$q(x, t) := q_w(x, t)/\rho_w + q_o(x, t)/\rho_o \quad (\text{B.1.3})$$

Adding (B.1.1) of two phases and using the definitions above yields

$$\begin{cases} v(x, t) & = -[\mathbf{K}(x)\lambda_w(s_w(x, t))(\nabla_x p_w(x, t) - \rho_w G) + \mathbf{K}(x)\lambda_o(s_o(x, t))(\nabla_x p_o(x, t) - \rho_o G)] \\ \nabla_x \cdot v(x, t) & = q(x, t) \end{cases} \quad (\text{B.1.4})$$

Following [AGL07, P.27] and [CHM06, P.24], define a global pressure  $p$

$$p(x, t) := p_o(x, t) - \int_1^{s_w(x, t)} f_w(\xi) \frac{dp_{cow}}{ds_w}(\xi) d\xi \quad (\text{B.1.5})$$

in which capillary pressure  $p_{cow}(s_w(x, t))$  represents the pressure difference between oil phase and water phase,  $p_{cow} = p_o - p_w$ . It is determined by the phase saturation and I wrote it as a function of water saturation.

$$\begin{aligned} \nabla_x p(x, t) &= \nabla_x p_o(x, t) - f_w(s_w(x, t)) \frac{dp_{cow}}{ds_w}(s_w(x, t)) \nabla_x s_w(x, t) \\ &= \nabla_x p_o(x, t) - f_w(s_w(x, t)) \nabla_x p_{cow}(s_w(x, t)) \\ &= \nabla_x p_o(x, t) - \frac{\lambda_w(x, t)}{\lambda_w(x, t) + \lambda_o(x, t)} \nabla_x (p_o(x, t) - p_w(x, t)) \\ &= \frac{\lambda_o(x, t)}{\lambda_w(x, t) + \lambda_o(x, t)} \nabla_x p_o(x, t) + \frac{\lambda_w(x, t)}{\lambda_w(x, t) + \lambda_o(x, t)} \nabla_x p_w(x, t) \end{aligned} \quad (\text{B.1.6})$$

Define total mobility

$$\lambda(s_w(x, t)) := \lambda_w(s_w(x, t)) + \lambda_o(s_w(x, t)) \quad (\text{B.1.7})$$

Insert (B.1.6) into (B.1.4). Instead of two phase pressures, global pressure is the new unknown variable

$$\begin{cases} v(x, t) &= -[\mathbf{K}(x) \lambda(s_w(x, t)) \nabla_x p(x, t) - \mathbf{K}(x) (\lambda_w(s_w(x, t)) \rho_w + \lambda_o(s_w(x, t)) \rho_o) G] \\ \nabla_x \cdot v(x, t) &= q(x, t) \end{cases} \quad (\text{B.1.8})$$

This is the pressure equation

$$-\nabla_x \cdot [\mathbf{K}(x) \lambda(s_w(x, t)) \nabla_x p(x, t) - \mathbf{K}(x) (\lambda_w(s_w(x, t)) \rho_w + \lambda_o(s_w(x, t)) \rho_o) G] = q(x, t) \quad (\text{B.1.9})$$

**Derivation of Saturation Equation** Define fractional flow function

$$f_w = \frac{\lambda_w}{\lambda_w + \lambda_o} \quad (\text{B.1.10})$$

Start from water phase of (B.1.1), apply Darcy's Law, and plug in  $v_o = v - v_w$

$$\begin{cases} v_w(x, t) = f_w(s_w(x, t)) \left[ v(x, t) + \mathbf{K}(x) \lambda_o(s_w(x, t)) \nabla_x p_{cow}(x, t) \right. \\ \left. + \mathbf{K}(x) \lambda_o(s_w(x, t)) (\rho_w - \rho_o) G \right] \\ \phi(x) \frac{\partial s_w(x, t)}{\partial t} + \nabla_x \cdot v_w(x, t) = \frac{q_w(x, t)}{\rho_w} \end{cases} \quad (\text{B.1.11})$$

This is the saturation equation.

**About Capillary Pressure** The capillary pressure  $p_{cow}$  is a scalar argument function of saturation  $s_w(x, t)$ , so

$$\nabla_x p_{cow}(s_w(x, t)) = p'_{cow}(s_w(x, t)) \nabla_x s_w(x, t) \quad (\text{B.1.12})$$

where  $p'_{cow}$  is the derivative of scalar argument function  $p_{cow}$ . The capillary pressure related part of  $\nabla_x \cdot v_w$  is

$$\begin{aligned} & \nabla_x \cdot [f_w(s_w(x, t)) \mathbf{K}(x) \lambda_o(s_w(x, t)) \nabla_x p_{cow}(s_w(x, t))] \\ &= \nabla_x \cdot [f_w(s_w(x, t)) \mathbf{K}(x) \lambda_o(s_w(x, t)) p'_{cow}(s_w(x, t)) \nabla_x s_w(x, t)] \\ &= \nabla_x \cdot \left[ \mathbf{K}(x) \frac{\lambda_w(s_w(x, t)) \lambda_o(s_w(x, t))}{\lambda_w(s_w(x, t)) + \lambda_o(s_w(x, t))} p'_{cow}(s_w(x, t)) \nabla_x s_w(x, t) \right] \end{aligned} \quad (\text{B.1.13})$$

Following Brooks-Corey formula [BC64] for approximation of capillary pressure, with normalized saturation  $s_w^*(x, t) := \frac{s_w(x, t) - s_{wc}}{1 - s_{or} - s_{wc}}$ , I use  $P_d s_w^*(x, t)^{-\frac{1}{2}}$  to approximate capillary pressure. This approximation has a singularity at  $s_w^*(x, t) = 0$ . Numerically, I apply a linearization near the singularity

$$p_{cow}(s_w) = \begin{cases} P_d \left( \frac{s_w - s_{wc}}{1 - s_{or} - s_{wc}} \right)^{-\frac{1}{2}} & \text{if } \epsilon \leq \frac{s_w - s_{wc}}{1 - s_{or} - s_{wc}} \\ -\frac{1}{2} P_d \epsilon^{-\frac{3}{2}} \left( \frac{s_w - s_{wc}}{1 - s_{or} - s_{wc}} - \epsilon \right) + P_d \epsilon^{-\frac{1}{2}} & \text{if } 0 \leq \frac{s_w - s_{wc}}{1 - s_{or} - s_{wc}} < \epsilon \end{cases} \quad (\text{B.1.14})$$

Since,  $p'_{cow} < 0$  everywhere, (B.1.13) is a nonlinear diffusion term in the saturation equation.

**Coupled System** The PDE system (B.1.9) and (B.1.11) describes the subsurface flow in oil reservoir. Pressure equation (B.1.9) and saturation equation (B.1.11) are coupled. In pressure equation, mobilities depend on saturation; in saturation equation, total velocity is computed in pressure equation.

I neglect gravity. Let  $\Omega$  be an open set in  $\mathbb{R}^3$ . In domain  $\Omega \times [0, T]$ , the PDE

system is

for  $x \in \Omega, t \in [0, T]$

$$-\nabla_x \cdot [\mathbf{K}(x)\lambda(s_w(x, t))\nabla_x p(x, t)] = q(x, t) \quad (\text{B.1.15a})$$

$$-\mathbf{K}(x)\lambda(s_w(x, t))\nabla_x p(x, t) = v(x, t) \quad (\text{B.1.15b})$$

$$f_w(s_w(x, t))(v(x, t) + \mathbf{K}(x)\lambda_o(s_w(x, t))\nabla_x p_{cow}(s_w(x, t))) = v_w(x, t) \quad (\text{B.1.15c})$$

$$\phi(x)\frac{\partial s_w(x, t)}{\partial t} + \nabla_x \cdot v_w(x, t) = \frac{q_w(x, t)}{\rho_w} \quad (\text{B.1.15d})$$

for  $x \in \partial\Omega, t \in [0, T]$

$$v(x, t) \cdot n = 0 \quad (\text{B.1.15e})$$

$$v_w(x, t) \cdot n = 0 \quad (\text{B.1.15f})$$

for  $x \in \Omega$

$$s_w(x, 0) = s_0(x) \quad (\text{B.1.15g})$$

$n$  is the outward normal vector on the boundary. The water phase source term is determined by the total source term through

$$q_w(x, t) = \begin{cases} q(x, t) & q(x, t) \geq 0 \\ f_w(s_w(x, t))q(x, t) & q(x, t) < 0 \end{cases}$$

By (B.1.2), two Neumann boundary conditions (B.1.15e)(B.1.15f) enforces no flow condition for both water and oil phase.

## B.2 Numerical Scheme

I apply Finite Volume Method to discretize (B.1.15) in space [EHM03] [Mic03] [Wie10], solve pressure equation and saturation equation alternatively in a sequential time stepping manner [AGL07]. In solving saturation equation, I use backward Euler method.

---

---

$N \in \mathbb{Z}^+$	grid size (number of cells), $N = N_x \times N_y \times N_z$
$K \in \mathbb{Z}^+$	last time step index (time step index is 0 based)
$\Delta t \in \mathbb{R}$	time step size
$\mathbf{e}_i \in \mathbb{R}^N$	$\mathbf{e}_i = (0, \dots, 0, 1, 0, \dots, 0)^T$ with ‘1’ at the $i$ th position
$\mathbf{p}^k \in \mathbb{R}^N$	pressure vector of length $N$ at time step $k$ . The $i$ th component, $p_i^k$ , is the pressure for cell $\Omega_i$
$\hat{\mathbf{q}}^k \in \mathbb{R}^{N_q}$	well injection/production rates (mass per space per time) at time step $k$ . The $j$ th component, $\hat{q}_j^k$ , is the well rates for cell $\Omega_{l_j}$ . $\hat{q}_j^k = \int_{\Omega_{l_j}} q(x, t_k) dx$ with $q(x, t)$ defined in table B.1.
$\mathbf{q}^k \in \mathbb{R}^N$	$\mathbf{q}^k = \sum_{j=1}^{N_q} \hat{q}_j^k \mathbf{e}_{l_j}$ , source vector of length $N$ at time step $k$ . The $i$ th component, $q_i^k$ , is positive for injection, negative for production There are only $N_q$ non-zero components representing $N_q$ controlled wells. In the context of water flooding, define water source term $q_{w,i}^k = \begin{cases} q_i^k & q_i^k \geq 0 \\ f_w(s_i^k) q_i^k & q_i^k < 0 \end{cases}$ corresponding to $q_w(x, t)$ in (B.1.15d). $f_w$ is defined in (B.1.10).
$\mathbf{s}^k \in \mathbb{R}^N$	water saturation vector of length $N$ at time step $k$ . The $i$ th component, $s_i^k$ , is the saturation for cell $\Omega_i$
$\boldsymbol{\kappa}^k \in \mathbb{R}^{3 \times 3 \times N}$	$3 \times 3$ relative permeability matrix array of length $N$ at time step $k$ . The $i$ th component, $\boldsymbol{\kappa}_i^k$ , is the $3 \times 3$ relative permeability matrix for cell $\Omega_i$
$\mathbf{v}^k \in \mathbb{R}^M$	$M = (N_x - 1) \times N_y \times N_z + N_x \times (N_y - 1) \times N_z + N_x \times N_y \times (N_z - 1)$ flux between adjacent cells at time step $k$ . The component indexed by ‘ $ij$ ’, $v_{ij}^k$ , is the flux from cell $\Omega_i$ to $\Omega_j$ .

---

Table B.2: Nomenclature for the Numerical Scheme of Reservoir Simulation

The domain of the reservoir is discretized into a rectangular grid of size

$$N = N_x \times N_y \times N_z$$

I use a series of evenly spaced time steps

$$t_0 < t_1 < t_2 < \dots < t_k < \dots < t_K = T \quad (\text{B.2.1})$$

where  $t_k = t_0 + k\Delta t$ ,  $\forall k \in \{0, 1, \dots, K\}$ . Number of wells in the reservoir is  $N_q$ . All of the well locations are fixed during the simulation. Well  $i \in I_{well} := \{l_1, \dots, l_{N_q}\}$  is located in cell  $\Omega_i$ .  $I_{inj}$  and  $I_{pro}$  represent the index set of injection and production wells respectively

$$I_{inj} \cup I_{pro} = I_{well}, \quad I_{inj} \cap I_{pro} = \emptyset, \quad I_{inj}, I_{pro} \neq \emptyset$$

$\forall x \in \Omega_i, \forall t \in [0, T]$

$$q(x, t) \begin{cases} \geq 0 & i \in I_{inj} \\ \leq 0 & i \in I_{pro} \\ = 0 & i \notin (I_{inj} \cup I_{pro}) \end{cases} \quad (\text{B.2.2})$$

### B.2.1 Solving Pressure Equation

I use Two Point Flux Approximation (TPFA) [AGL07] to solve (B.1.15a). At time step  $k$ , define  $3 \times 3$  relative permeability matrix  $\kappa(x, t_k)$

$$\kappa(x, t_k) = \mathbf{K}(x)\lambda(s_w(x, t_k)) \quad (\text{B.2.3})$$

and the discretized version of permeability matrix for time step  $k$

$$\boldsymbol{\kappa}^k \in \mathbb{R}^{3 \times 3 \times N}$$

is a  $3 \times 3$  matrix array of length  $N$ , i.e. an array of length  $N$  whose every component consists of a  $3 \times 3$  matrix, approximating  $\kappa(x, t_k)$  by cell wise constant values.

Insert (B.2.3) into (B.1.15a),

$$-\nabla_x \cdot \kappa(x, t_k) \nabla_x p(x, t_k) = q(x, t_k) \quad (\text{B.2.4})$$

Apply divergence theorem on grid cell  $\Omega_i$ ,

$$-\int_{\partial\Omega_i} \kappa(x, t_k) \nabla_x p \cdot n = \int_{\Omega_i} q(x, t_k) \quad (\text{B.2.5})$$



where  $n$  is the outward normal vector on the boundary  $\partial\Omega_i$  of  $\Omega_i$ . In this project, I use rectangular hexahedron grid cells.  $\gamma_{ij}$  is the interface between grid cell  $\Omega_i$  and  $\Omega_j$  with normal vector  $n_{ij}$  pointing from  $\Omega_i$  to  $\Omega_j$ .

$$-\int_{\partial\Omega_i} \kappa(x, t_k) \nabla_x p(x, t_k) \cdot n = - \sum_{\Omega_j \text{ is adjacent to } \Omega_i} \int_{\gamma_{ij}} \kappa(x, t_k) \nabla_x p(x, t_k) \cdot n_{ij} \quad (\text{B.2.6})$$

I replace  $\nabla p \cdot n_{ij}$  on  $\gamma_{ij}$  with numerical approximation

$$\delta p_{ij}^k = \frac{(p_j^k - p_i^k)}{\frac{\Delta h_i + \Delta h_j}{2}} \quad (\text{B.2.7})$$

where  $\Delta h_i$  and  $\Delta h_j$  are the dimensions of grid cell  $\Omega_i$  and  $\Omega_j$  in the direction from  $\Omega_i$  to  $\Omega_j$ . Then I obtain the approximation

$$\int_{\gamma_{ij}} \kappa(x, t_k) \nabla_x p(x, t_k) \cdot n_{ij} \approx \delta p_{ij}^k \int_{\gamma_{ij}} \kappa(x, t_k) \quad (\text{B.2.8})$$

Since  $\kappa^k$  is discontinuous on  $\gamma_{ij}$ , the value of  $\kappa(x, t_k)$  on  $\gamma_{ij}$  is approximated by a distance-weighted harmonic average of  $\kappa_{i,ij}^k = n_{ij} \cdot \kappa_i^k n_{ij}$  and  $\kappa_{j,ij}^k = n_{ij} \cdot \kappa_j^k n_{ij}$ ,

$$\kappa_{ij}^k = (\Delta h_i + \Delta h_j) \left( \frac{\Delta h_i}{\kappa_{i,ij}^k} + \frac{\Delta h_j}{\kappa_{j,ij}^k} \right)^{-1} \quad (\text{B.2.9})$$

Combine (B.2.5), (B.2.6), (B.2.7), (B.2.8), and (B.2.9)

$$\int_{\Omega_i} q(x, t_k) \approx \sum_{\Omega_j \text{ is adjacent to } \Omega_i} 2|\gamma_{ij}| \left( \frac{\Delta h_i}{\kappa_{i,ij}^k} + \frac{\Delta h_j}{\kappa_{j,ij}^k} \right)^{-1} (p_i^k - p_j^k) \quad (\text{B.2.10})$$

Conventionally, define TPFA transmissibility

$$\tau_{ij}^k = 2|\gamma_{ij}| \left( \frac{\Delta h_i}{\kappa_{i,ij}^k} + \frac{\Delta h_j}{\kappa_{j,ij}^k} \right)^{-1} \quad (\text{B.2.11})$$

Then, I have TPFA discretized version of (B.1.15a) as follows

$$\int_{\Omega_i} q(x, t_k) \approx \sum_{\Omega_j \text{ is adjacent to } \Omega_i} \tau_{ij}^k (p_j^k - p_i^k) \quad \forall \Omega_i \subset \Omega \quad (\text{B.2.12})$$

In matrix form

$$\mathbf{A}(\mathbf{s}^k) \mathbf{p}^k = \mathbf{q}^k \quad (\text{B.2.13})$$

In which  $\mathbf{A}(\mathbf{s}^k) = [a_{im}]$ , a  $N$  by  $N$  matrix, with  $\mathbf{s}^k$  in parenthesis to emphasize  $\gamma_{ij}^k$ 's dependence on saturation

$$a_{im} = \begin{cases} \sum_{\Omega_j \text{ is adjacent to } \Omega_i} \tau_{ij}^k & m = i \\ -\tau_{im}^k & m \neq i \end{cases} \quad (\text{B.2.14})$$

The matrix  $\mathbf{A}(\mathbf{s}^k)$  is singular. When  $\sum_{i=1}^N q_i^t = 0$ , this system has solutions which are determined up to a constant. By adding a positive constant to  $a_{11}$ , I make this matrix non-singular and force the resulting pressure in  $\Omega_1$  to be 0.

The flux from  $\Omega_i$  to adjacent cell  $\Omega_j$  is computed by

$$\int_{\gamma_{ij}} v(x, t) \cdot n_{ij} \approx v_{ij}^k = \tau_{ij}^k (p_i^k - p_j^k) \quad (\text{B.2.15})$$

## B.2.2 Solving Saturation

For the time dependent saturation equation, (B.1.15d), integrate over grid cell  $\Omega_i$

$$\int_{\Omega_i} \phi(x) \frac{\partial s_w(x, t)}{\partial t} + \int_{\Omega_i} \nabla_x \cdot v_w(x, t) = \frac{\int_{\Omega_i} q_w(x, t) dx}{\rho_w} \quad (\text{B.2.16})$$

substitute (B.1.15b) into the equation above

$$\begin{aligned} \int_{\Omega_i} \nabla_x \cdot v_w(x, t) &= \int_{\Omega_i} \nabla_x \cdot f_w(s_w(x, t)) \mathbf{K}(x) \lambda_o(s_w(x, t)) \nabla_x p_{cow}(s_w(x, t)) \\ &\quad + \int_{\Omega_i} \nabla_x \cdot f_w(s_w(x, t)) v(x, t) \end{aligned} \quad (\text{B.2.17})$$

**The first integral of (B.2.17)**

$$\int_{\Omega_i} \nabla_x \cdot f_w(s_w(x, t)) \mathbf{K}(x) \lambda_o(s_w(x, t)) \nabla_x p_{cow}(s_w(x, t))$$

corresponds to capillary pressure, by (B.1.13) and divergence theorem

$$\begin{aligned} &\int_{\Omega_i} \nabla_x \cdot \mathbf{K}(x) \lambda_o(s_w(x, t)) \nabla_x p_{cow}(s_w(x, t)) \\ &= \int_{\Omega_i} \nabla_x \cdot \left[ \mathbf{K}(x) \frac{\lambda_w(s_w(x, t)) \lambda_o(s_w(x, t))}{\lambda_w(s_w(x, t)) + \lambda_o(s_w(x, t))} p'_{cow}(s_w(x, t)) \nabla_x s_w(x, t) \right] \\ &= \int_{\partial \Omega_i} n \cdot \left[ \mathbf{K}(x) \frac{\lambda_w(s_w(x, t)) \lambda_o(s_w(x, t))}{\lambda_w(s_w(x, t)) + \lambda_o(s_w(x, t))} p'_{cow}(s_w(x, t)) \nabla_x s_w(x, t) \right] \end{aligned} \quad (\text{B.2.18})$$

Below, I derive its discrete version. In the computation from time step  $k$  to time step  $k + 1$ , in terms of evaluating (B.2.18), I either choose  $k$  or  $k + 1$  as the time index for saturation. For example, in a backward Euler scheme, all of the saturation used in (B.2.18) is in step  $k + 1$ . Since the superscripts are all the same, I will omit them on the saturation notation below for simplicity. Since,

$$\partial\Omega_i = \cup_{\Omega_j \text{ is adjacent to } \Omega_i} \gamma_{ij}$$

Look at the integration of (B.2.18) on the boundary surface of two cells,

$$\int_{\gamma_{ij}} n_{ij} \cdot [\mathbf{K}(x) \frac{\lambda_w(s_w(x, t)) \lambda_o(s_w(x, t))}{\lambda_w(s_w(x, t)) + \lambda_o(s_w(x, t))} p'_{cow}(s_w(x, t)) \nabla_x s_w(x, t)] \quad (\text{B.2.19})$$

The permeability term,  $\mathbf{K}(x)$ , on the face is approximated by a constant, the harmonic average of permeability on the two cells,  $\Omega_i$  and  $\Omega_j$ , sharing this face,  $\gamma_{ij}$ . In this project,  $\mathbf{K}(x)$  is diagonal. Assume the permeability diagonal component of the direction normal to  $\gamma_{ij}$  is indexed by  $s$  ( $s = 1, 2, 3$ ),

$$\mathbf{K}_s(x) \approx \frac{2\mathbf{K}_{i,s}\mathbf{K}_{j,s}}{\mathbf{K}_{i,s} + \mathbf{K}_{j,s}} \quad (\text{B.2.20})$$

Saturation used in the rest of the integral is the arithmetic average of saturation in  $\Omega_i$  and  $\Omega_j$ ,

$$\bar{s}_w = \frac{s_i + s_j}{2} \quad (\text{B.2.21})$$

And in (B.2.19) the term

$$\frac{\lambda_w(s_w(x, t)) \lambda_o(s_w(x, t))}{\lambda_w(s_w(x, t)) + \lambda_o(s_w(x, t))} p'_{cow}(s_w(x, t)) \approx \frac{\lambda_w(\bar{s}_w) \lambda_o(\bar{s}_w)}{\lambda_w(\bar{s}_w) + \lambda_o(\bar{s}_w)} p'_{cow}(\bar{s}_w) \quad (\text{B.2.22})$$

The component in the direction between  $\Omega_i$  and  $\Omega_j$  of  $\nabla_x s_w(x, t)$  is approximated by a two point finite difference. Combine (B.2.20) and (B.2.21), on  $\gamma_{ij}$ ,

$$n_{ij} \cdot \mathbf{K}(x) \nabla_x s_w(x, t) \approx \frac{2\mathbf{K}_{i,s}\mathbf{K}_{j,s}}{\mathbf{K}_{i,s} + \mathbf{K}_{j,s}} \frac{s_j - s_i}{h} \quad (\text{B.2.23})$$

Lastly, by (B.2.22) and (B.2.23), (B.2.18) is approximated numerically

$$\begin{aligned} & \int_{\partial\Omega_i} n \cdot \left[ \mathbf{K}(x) \frac{\lambda_w(s_w(x,t))\lambda_o(s_w(x,t))}{\lambda_w(s_w(x,t)) + \lambda_o(s_w(x,t))} p'_{cow}(s_w(x,t)) \nabla_x s_w(x,t) \right] \\ & \approx \sum_{\Omega_j \text{ is adjacent to } \Omega_i} |\gamma_{ij}| \frac{\lambda_w(\bar{s}_w)\lambda_o(\bar{s}_w)}{\lambda_w(\bar{s}_w) + \lambda_o(\bar{s}_w)} p'_{cow}(\bar{s}_w) \frac{2\mathbf{K}_{i,s}\mathbf{K}_{j,s}}{\mathbf{K}_{i,s} + \mathbf{K}_{j,s}} \frac{s_j - s_i}{h} \end{aligned} \quad (\text{B.2.24})$$

and I define

$$E_{ij}(\mathbf{s}^m) = |\gamma_{ij}| \frac{\lambda_w(\bar{s}_w^m)\lambda_o(\bar{s}_w^m)}{\lambda_w(\bar{s}_w^m) + \lambda_o(\bar{s}_w^m)} p'_{cow}(\bar{s}_w^m) \frac{2\mathbf{K}_{i,s}\mathbf{K}_{j,s}}{\mathbf{K}_{i,s} + \mathbf{K}_{j,s}} \frac{s_j^m - s_i^m}{h} \quad (\text{B.2.25})$$

**The second part of (B.2.17)**

$$\int_{\Omega_i} \nabla_x \cdot f_w(s_w(x,t))v(x,t)$$

represents the fluid flow due to pressure gradient, apply divergence theorem

$$\begin{aligned} \int_{\Omega_i} \nabla_x \cdot f_w(s_w(x,t))v(x,t) &= \int_{\partial\Omega_i} f_w(s_w(x,t))v(x,t) \cdot n \\ &= \sum_{\Omega_j \text{ is adjacent to } \Omega_i} \int_{\gamma_{ij}} f_w(s_w(x,t))v(x,t) \cdot n_{ij} \end{aligned} \quad (\text{B.2.26})$$

Here, to do approximation, I want

$$F_{ij}(\mathbf{s}^m, v_{ij}^k, \mathbf{p}^k) \approx \int_{\gamma_{ij}} f_w(s_w(x, t_m))v(x, t_k) \cdot n_{ij} \quad (\text{B.2.27})$$

The numerical approximation of saturation is cell wise constant and discontinuous on the boundary of cells. I use upstream weighting for the fractional flow

$$\int_{\gamma_{ij}} f_w(s_w(x, t_m))v(x, t_k) \cdot n_{ij} \approx \begin{cases} \int_{\gamma_{ij}} f_w(s_i^m)v(x, t_k) \cdot n_{ij} & p_i^k \geq p_j^k \\ \int_{\gamma_{ij}} f_w(s_j^m)v(x, t_k) \cdot n_{ij} & p_i^k < p_j^k \end{cases} \quad (\text{B.2.28})$$

Since,  $\int_{\gamma_{ij}} f_w(s_i^m)v(x, t_m) \cdot n_{ij} = f_w(s_i^m)v_{ij}^m$  and  $\int_{\gamma_{ij}} f_w(s_j^m)v(x, t_m) \cdot n_{ij} = f_w(s_j^m)v_{ij}^m$  by (B.2.15) Then, (B.2.27) and (B.2.28) are reduced to

$$F_{ij}(\mathbf{s}^m, v_{ij}^k, \mathbf{p}^k) := \begin{cases} f_w(s_i^m)v_{ij}^k & p_i^k \geq p_j^k \\ f_w(s_j^m)v_{ij}^k & p_i^k < p_j^k \end{cases} \quad (\text{B.2.29})$$

Since, flux fully depends on pressure. By (B.2.15), I can hide the dependence of  $F_{ij}$  on  $v_{ij}^k$  into the dependence on  $\mathbf{p}^k$  as follows

$$F_{ij}(\mathbf{s}^m, v_{ij}^k, \mathbf{p}^k) = F_{ij}(\mathbf{s}^m, \mathbf{p}^k) = \begin{cases} f_w(s_i^m) \tau_{ij}^k (p_i^k - p_j^k) & p_i^k \geq p_j^k \\ f_w(s_j^m) \tau_{ij}^k (p_i^k - p_j^k) & p_i^k < p_j^k \end{cases} \quad (\text{B.2.30})$$

**Assemble two parts of (B.2.17)** With  $E_{ij}(\mathbf{s}^m)$  and  $F_{ij}(\mathbf{s}^m, \mathbf{p}^k)$  calculated in the above two parts, (B.2.17) yields the discrete relationship for saturation

$$\frac{\phi_i}{\Delta t} (s_i^{k+1} - s_i^k) + \frac{1}{|\Omega_i|} \sum_{\Omega_j \text{ is adjacent to } \Omega_i} [\theta G_{ij}(\mathbf{s}^{k+1}, \mathbf{p}^k) + (1 - \theta) G_{ij}(\mathbf{s}^k, \mathbf{p}^k)] = \frac{q_{w,i}^k}{\rho_w} \quad (\text{B.2.31})$$

where

$$G_{ij}(\mathbf{s}^m, \mathbf{p}^k) = E_{ij}(\mathbf{s}^m) + F_{ij}(\mathbf{s}^m, \mathbf{p}^k) \quad (\text{B.2.32})$$

$\theta$  switches the scheme between implicit and explicit, 1 for implicit and 0 for explicit.

**Implicit Saturation Equation** When  $\theta$  is set to 1, (B.2.31) yields a backward Euler implicit scheme for the saturation equation

$$\frac{\phi_i}{\Delta t} (s_i^{k+1} - s_i^k) + \frac{1}{|\Omega_i|} \sum_{\Omega_j \text{ is adjacent to } \Omega_i} G_{ij}(\mathbf{s}^{k+1}, \mathbf{p}^k) = \frac{q_{w,i}^k}{\rho_w} \quad (\text{B.2.33})$$

I use a quasi-Newton method to solve this equation.

## B.3 Trilinos Implementation

Trilinos Project Packages [HBH<sup>+</sup>05] are extensively used in this reservoir simulation.

- **Epetra** provides the data structures and routines in parallel linear algebra. I use it to store saturation, pressure, permeability, porosity in distributed memory and sparse system matrices in distributed memory.

- **AztecOO** provides an object-oriented interface to the Aztec linear solver library. It solves the symmetric pressure systems by Conjugate Gradient and solves the asymmetric linear equation systems arising from nonlinear saturation equation by GMRES.
- **ML** provides a black-box parallel multigrid preconditioner which significantly accelerates the linear solvers in forward simulation and backward adjoint computation.
- **SACADO** provides classes for automatic differentiation.
- **Teuchos** provides a set of convenient common tools, e.g. smart pointers, parameter lists, XML parsers.

# Appendix C

## Parallel-In-Time Gradient-Type Method Implementation Details In Reservoir Optimization

The reservoir optimization problem (6.2.2) has three characteristics that influences the parallel-in-time gradient-type method implementation:

1. its state equations are implicit, i.e., the state in a new time step is given by solving an equation system, which has an impact on adjoint variable values;
2. there are two state equations in each time step, i.e., the pressure equation (6.2.2b) and the saturation equation (6.2.2c);
3. there are two state variables, i.e., pressure and saturation.

Of course, it can be reduced to the standard form of (5.2.1) and apply for the parallel-in-time gradient-type method, Algorithm 13. However, in the implementation, some new aspects needs to be considered, e.g. data communication with the implicit state equation. In addition, I made slight adjustment of the backward computation time subdomain partition in Algorithm 13 so that the parallel gradient-type algorithm is more efficient.

Therefore, for clarity, I give the implementation details of the parallel-in-time gradient-type method applied to the reservoir problem.

I first give the parallel-in-time gradient-type algorithm with implicit state equations and show its equivalence to the parallel-in-time gradient-type algorithm with explicit state equations in the sense that they produce exactly the same control update. Then, I discuss the parallel-in-time gradient-type algorithm used in the reservoir optimization.

## C.1 Algorithms For Explicit/Implicit Formulation of State Equations

In Chapter 5, the state equations of the model problem (5.2.1) has explicit form. For reading convenience, I repeat (5.2.1) here:

$$\text{Minimize } \sum_{k=0}^K J_k(y_k, u_k) \quad (\text{C.1.1a})$$

$$\text{subject to } y_{k+1} = F_k(y_k, u_k), \quad k = 0, \dots, K - 1, \quad (\text{C.1.1b})$$

$$y_0 = y_{\text{given}} \quad (\text{C.1.1c})$$

$$u \in D \quad (\text{C.1.1d})$$

In practice, as the case in the reservoir optimization problem, state equations are often implicitly defined as in (C.1.2b).

$$\text{Minimize } \sum_{k=0}^K J_k(y_k, u_k) \quad (\text{C.1.2a})$$

$$\text{subject to } \tilde{F}_k(y_k, y_{k+1}, u_k) = 0, \quad k = 0, \dots, K - 1, \quad (\text{C.1.2b})$$

$$y_0 = y_{\text{given}} \quad (\text{C.1.2c})$$

$$u \in D \quad (\text{C.1.2d})$$

Assume that, for  $k = 0, \dots, K - 1$ , given  $y_k, u_k$ , the state equation  $F_k(y_k, y_{k+1}, u_k) = 0$  uniquely determine  $y_{k+1}$ .



### C.1.1 Gradient Computation with Implicit State Equations

Lagrangian of problem (C.1.2),

$$\tilde{\mathcal{L}} = \sum_{k=0}^K J_k(y_k, u_k) + \sum_{k=0}^{K-1} \lambda_k^T \tilde{F}_k(y_k, y_{k+1}, u_k) + \psi^T(y_0 - y_{\text{given}}) \quad (\text{C.1.3})$$

Take derivatives of  $\tilde{\mathcal{L}}$  with respect to state variables, set them to be zero and obtain adjoint equations,

$$0 = \frac{\partial \mathcal{L}}{\partial y_K} = \frac{\partial J_K(y_K, u_K)}{\partial y_K} + \left[ \frac{\partial \tilde{F}_{K-1}(y_{K-1}, y_K, u_{K-1})}{\partial y_K} \right]^T \lambda_{K-1} \quad (\text{C.1.4a})$$

$$0 = \frac{\partial \mathcal{L}}{\partial y_k} = \frac{\partial J_k(y_k, u_k)}{\partial y_k} + \left[ \frac{\partial \tilde{F}_{k-1}(y_{k-1}, y_k, u_{k-1})}{\partial y_k} \right]^T \lambda_{k-1} + \left[ \frac{\partial \tilde{F}_k(y_k, y_{k+1}, u_k)}{\partial y_k} \right]^T \lambda_k$$

$$k = 1, \dots, K - 1 \quad (\text{C.1.4b})$$

Denote the reduced control space objective as  $\hat{J}$ . The gradient is

$$\frac{\partial \hat{J}}{\partial u_K} = \frac{\partial J_K(y_K, u_K)}{\partial u_K} \quad (\text{C.1.5a})$$

$$\frac{\partial \hat{J}}{\partial u_k} = \frac{\partial J_k(y_k, u_k)}{\partial u_k} + \left[ \frac{\partial \tilde{F}_k(y_k, y_{k+1}, u_k)}{\partial u_k} \right]^T \lambda_k \quad k = 1, \dots, K - 1 \quad (\text{C.1.5b})$$

### C.1.2 Gradient Computation with Explicit State Equations

Note that problem with explicit state equations (C.1.1) is closely related to the special case of (C.1.2) with  $\tilde{F}_k(y_k, y_{k+1}, u_k) = F_k(y_k, u_k) - y_{k+1}$ .

Lagrangian of problem (C.1.1),

$$\mathcal{L} = \sum_{k=0}^K J_k(y_k, u_k) + \sum_{k=0}^{K-1} \gamma_k^T (F_k(y_k, u_k) - y_{k+1}) + \phi^T(y_0 - y_{\text{given}}) \quad (\text{C.1.6})$$

Adjoint equation

$$0 = \frac{\partial \mathcal{L}}{\partial y_K} = \frac{\partial J_K(y_K, u_K)}{\partial y_K} - \gamma_{K-1} \quad (\text{C.1.7a})$$

$$0 = \frac{\partial \mathcal{L}}{\partial y_k} = \frac{\partial J_k(y_k, u_k)}{\partial y_k} - \gamma_{k-1} + \left[ \frac{\partial F_k(y_k, u_k)}{\partial y_k} \right]^T \gamma_k \quad (\text{C.1.7b})$$

$$= \frac{\partial J_k(y_k, u_k)}{\partial y_k} - \gamma_{k-1} + \left[ - \left\{ \frac{\partial \tilde{F}_k(y_k, y_{k+1} u_k)}{\partial y_{k+1}} \right\}^{-1} \frac{\partial \tilde{F}_k(y_k, y_{k+1} u_k)}{\partial y_k} \right]^T \gamma_k \quad (\text{C.1.7c})$$

$$= \frac{\partial J_k(y_k, u_k)}{\partial y_k} - \gamma_{k-1} - \left[ \frac{\partial \tilde{F}_k(y_k, y_{k+1} u_k)}{\partial y_k} \right]^T \left[ \frac{\partial \tilde{F}_k(y_k, y_{k+1} u_k)}{\partial y_{k+1}} \right]^{-T} \gamma_k \quad (\text{C.1.7d})$$

$$k = 0, \dots, K - 1$$

Compare adjoint equations (C.1.4) and (C.1.7) and the adjoint relationship is clear

$$- \left[ \frac{\partial \tilde{F}_k(y_k, y_{k+1} u_k)}{\partial y_{k+1}} \right]^T \lambda_k = \gamma_k, \quad k = 0, \dots, K - 1 \quad (\text{C.1.8})$$

Then, the derivative

$$\frac{\partial \hat{J}}{\partial u_K} = \frac{\partial J_K(y_K, u_K)}{\partial u_K} \quad (\text{C.1.9a})$$

$$\frac{\partial \hat{J}}{\partial u_k} = \frac{\partial J_k(y_k, u_k)}{\partial u_k} + \left[ \frac{\partial F_k(y_k, u_k)}{\partial u_k} \right]^T \gamma_k \quad (\text{C.1.9b})$$

$$= \frac{\partial J_k(y_k, u_k)}{\partial u_k} + \left[ - \left\{ \frac{\partial \tilde{F}_k(y_k, y_{k+1}, u_k)}{\partial y_{k+1}} \right\} \frac{\partial \tilde{F}_k(y_k, y_{k+1}, u_k)}{\partial u_k} \right]^T \gamma_k \quad (\text{C.1.9c})$$

$$= \frac{\partial J_k(y_k, u_k)}{\partial u_k} - \left[ \frac{\partial \tilde{F}_k(y_k, y_{k+1}, u_k)}{\partial u_k} \right]^T \left[ \frac{\partial \tilde{F}_k(y_k, y_{k+1}, u_k)}{\partial y_{k+1}} \right]^{-T} \gamma_k \quad (\text{C.1.9d})$$

$$= \frac{\partial J_k(y_k, u_k)}{\partial u_k} + \left[ \frac{\partial \tilde{F}_k(y_k, y_{k+1}, u_k)}{\partial u_k} \right]^T \lambda_k \quad (\text{C.1.9e})$$

$$k = 1, \dots, K - 1$$

This verifies in form that the gradient computed by (C.1.5) and (C.1.9) is the same.

I summarize,

- The form of the state equation is not unique. Consequently, the resulting adjoint variable corresponding to state equations of different form can be different.

- If the state variables are all feasible, i.e.  $\tilde{F}_k(y_k, y_{k+1}, u_k) = 0, k = 0, \dots, K - 1$  or equivalently  $F_k(y_k, u_k) = y_{k+1}, k = 0, \dots, K - 1$ , the adjoint variable relationship between an implicit and explicit state equation is in (C.1.8).

### C.1.3 Algorithm with Explicit State Equations

In the generalized parallel-in-time gradient-type method framework, Algorithm 13 for problem (C.1.1) has the forward/backward arrangement of

$$s_i^F = s_i^B = K_i, \quad e_i^B = e_i^F = K_{i+1} \quad i = 0, \dots, N - 1 \quad (\text{C.1.10})$$

Refer to Section 3.4 for the generalized parallel-in-time gradient-type method framework context. In the implementation of the parallel-in-time gradient-type method applied to the reservoir problem, I adjust the arrangement to

$$\begin{aligned} s_i^F &= K_i, & e_i^F &= K_{i+1} & i &= 0, \dots, N - 1 \\ s_i^B &= K_i - 1, & e_i^B &= K_{i+1} - 1 & i &= 1, \dots, N - 1 \\ s_0^B &= K_0, & e_0^B &= K_1 - 1 \end{aligned} \quad (\text{C.1.11})$$

Refer to (4.2.14) for its connection to the multiple shooting reformulation. I give the pseudo code in Algorithm 17 with explicit state equation.

### C.1.4 Algorithm with Implicit State Equations

Suggested by (C.1.8), in the parallel-in-time gradient-type algorithm with implicit state equation, adjoint variable to communicate between processors are not  $\lambda_k$  but  $-\left[\frac{\partial \tilde{F}_k(y_k, y_{k+1}, u_k)}{\partial y_{k+1}}\right]^T \lambda_k$ , which results in the Algorithm 18.

One can familiarize himself with the Algorithm 17 and Algorithm 18 by comparing them line by line with the special assumption  $\tilde{F}(y_k, y_{k+1}, u_k) = F(y_k, u_k) - y_{k+1}$  and quickly realizing that both algorithms are performing the same computation under this assumption.

---

**Algorithm 17**  $j$ th iteration of the parallel-in-time gradient-type method with step size  $\alpha_j > 0$  for problem with explicit state equation (C.1.1). Describes the tasks executed by processor of rank  $n \in \{0, \dots, N - 1\}$ . The first subscript of variables indicates the subdomain it is associated to.

---

- 1: Input control  $u_{n,K_n}^{(j)}, u_{n,K_n+1}^{(j)}, \dots, u_{n,K_{n+1}-1}^{(j)}$  ▷ initialization of the iteration
- 2: **if**  $n > 0$  and  $j = 0$  **then**
- 3:   Input initial  $y_{n,K_n}^{(-1)}$
- 4: **end if**
- 5: **if**  $n < N - 1$  and  $j = 0$  **then**
- 6:   Input initial  $\gamma_{n,K_{n+1}}^{(-1)}$
- 7: **end if**
  
- 8: **if**  $n = 0$  **then** ▷ solve the state equation forward in time
- 9:    $y_{n,K_{n+1}}^{(j)} = F_{K_n}(y_{\text{given}}, u_{n,K_n}^{(j)})$
- 10: **else**
- 11:    $y_{n,K_{n+1}}^{(j)} = F_{K_n}(y_{n,K_n}^{(j-1)}, u_{n,K_n}^{(j)})$
- 12: **end if**
- 13: **for**  $k = K_n + 1, \dots, K_{n+1} - 1$  **do**
- 14:    $y_{n,k+1}^{(j)} = F_k(y_{n,k}^{(j)}, u_{n,k}^{(j)})$
- 15: **end for**
  
- 16: **if**  $n = N - 1$  **then** ▷ solve the adjoint equation backward in time
- 17:    $\gamma_{n,K_{n+1}-1}^{(j)} = \frac{\partial J_{K_{n+1}}(y_{n,K_{n+1}}^{(j)}, u_{n,K_{n+1}}^{(j)})}{\partial y_{n,K_{n+1}}^{(j)}}$
- 18: **else**
- 19:    $\gamma_{n,K_{n+1}-1}^{(j)} = \gamma_{n+1,K_{n+1}-1}^{(j-1)}$
- 20: **end if**
- 21: **for**  $k = K_{n+1} - 1, \dots, K_n$  **do**
- 22:    $\gamma_{n,k-1}^{(j)} = \frac{\partial J_k(y_{n,k}^{(j)}, u_{n,k}^{(j)})}{\partial y_{n,k}^{(j)}} + \left( \frac{\partial F_k(y_{n,k}^{(j)}, u_{n,k}^{(j)})}{\partial y_{n,k}^{(j)}} \right)^T \gamma_{n,k}^{(j)}$
- 23: **end for**

(continued on next page)

---

---

```

24: for  $k = K_n, \dots, K_{n+1} - 1$  do ▷ update control
25:    $u_k^{(j+1)} = \mathcal{P}_{D_k}(u_{n,k}^{(j)} - \alpha_j \left[ \frac{\partial J_k(y_{n,k}^{(j)}, u_{n,k}^{(j)})}{\partial u_{n,k}^{(j)}} + \left( \frac{\partial F_k(y_{n,k}^{(j)}, u_{n,k}^{(j)})}{\partial u_{n,k}^{(j)}} \right)^T \gamma_{n,k}^{(j)} \right])$ 
26: end for

27: if  $n > 0$  then ▷ communication between processors
28:   send  $\gamma_{n, K_{n-1}}^{(j)}$  to rank  $n - 1$ 
29:   receive  $y_{n-1, K_n}^{(j)}$  from rank  $n - 1$ 
30: end if

31: if  $n < N - 1$  then
32:   send  $y_{n, K_{n+1}}^{(j)}$  to rank  $n + 1$ 
33:   receive  $\gamma_{n+1, K_{n+1}-1}^{(j)}$  from rank  $n + 1$ 
34: end if

```

---

## C.2 Comparison of Algorithms

In this section, I briefly show Algorithm 17 and Algorithm 18 are equivalent in the sense that produces exactly the same control iterates. I use a mathematical induction type of argument. Assume both algorithm has the same state/control variables up to iteration  $j$  and

$$- \left[ \frac{\partial \tilde{F}_k(y_{n,k}^{(i)}, y_{n,k+1}^{(i)}, u_{n,k}^{(i)})}{\partial y_{n,k+1}^{(i)}} \right]^T \lambda_{n,k}^{(i)} = \gamma_{n,k}^{(j)} \quad (\text{C.2.1})$$

is true for  $i = 0, \dots, j$ ,  $n = 0, \dots, N - 1$ , and  $k = K_n, \dots, K_{n+1} - 1$ , with  $\gamma_{n,k}^{(i)}$  and  $\lambda_{n,k}^{(i)}$  defined in Algorithm 17 and Algorithm 18 respectively.

By the assumption, the control updates in iteration  $j$  will be the same for both algorithms. Therefore, in iteration  $(j + 1)$ , the control variables are the same, consequently the state variables are the same, and (C.2.1) is true for  $i = j + 1, n = N - 1, k = K_n, \dots, K_{n+1} - 1$ .

To show that for  $i = j + 1, n < N - 1$ , (C.2.1) is valid for  $k = K_n, \dots, K_{n+1} - 1$ , the only non-trivial work is to show (C.2.1) holds for  $k = K_{n+1} - 1$ , i.e., at the time

---

**Algorithm 18**  $j$ th iteration of the parallel-in-time gradient-type method with step size  $\alpha_j > 0$  for problem with implicit state equation (C.1.2). Describes the tasks executed by processor of rank  $n \in \{0, \dots, N - 1\}$ . The first subscript of variables indicates the subdomain it is associated to.

---

- 1: Input control  $u_{n,K_n}^{(j)}, u_{n,K_n+1}^{(j)}, \dots, u_{n,K_{n+1}-1}^{(j)}$  ▷ initialization of the iteration
- 2: **if**  $n > 0$  and  $j = 0$  **then**
- 3:   Input initial  $y_{n,K_n}^{(-1)}$
- 4: **end if**
- 5: **if**  $n < N - 1$  and  $j = 0$  **then**
- 6:   Input initial auxiliary variable  $\xi_{n+1}^{(-1)}$
- 7: **end if**
  
- 8: **if**  $n = 0$  **then** ▷ solve the state equation forward in time
- 9:   solve  $\tilde{F}_{K_n}(y_{\text{given}}, y_{n,K_n+1}^{(j)}, u_{n,K_n}^{(j)})$  for  $y_{n,K_n+1}^{(j)}$
- 10: **else**
- 11:   solve  $\tilde{F}_{K_n}(y_{n,K_n}^{(j-1)}, y_{n,K_n+1}^{(j)}, u_{n,K_n}^{(j)})$  for  $y_{n,K_n+1}^{(j)}$
- 12: **end if**
- 13: **for**  $k = K_n + 1, \dots, K_{n+1} - 1$  **do**
- 14:   solve  $\tilde{F}_k(y_{n,k}^{(j)}, y_{n,k+1}^{(j)}, u_{n,k}^{(j)})$  for  $y_{n,k+1}^{(j)}$
- 15: **end for**

(continued on next page)

---

---

16: **if**  $n = N - 1$  **then** ▷ solve the adjoint equation backward in time

17: solve

$$\frac{\partial J_{K_{n+1}}(y_{n,K_{n+1}}^{(j)}, u_{n,K_{n+1}}^{(j)})}{\partial y_{n,K_{n+1}}^{(j)}} + \left[ \frac{\partial \tilde{F}_{K_{n+1}-1}(y_{n,K_{n+1}-1}^{(j)}, y_{n,K_{n+1}}^{(j)}, u_{n,K_{n+1}-1}^{(j)})}{\partial y_{n,K_{n+1}}^{(j)}} \right]^T \lambda_{n,K_{n+1}-1}^{(j)} = 0$$

for  $\lambda_{n,K_{n+1}-1}^{(j)}$

18: **else**

19: solve

$$\left[ \frac{\partial \tilde{F}_{K_{n+1}-1}(y_{n,K_{n+1}-1}^{(j)}, y_{n,K_{n+1}}^{(j)}, u_{n,K_{n+1}-1}^{(j)})}{\partial y_{n,K_{n+1}}^{(j)}} \right]^T \lambda_{n,K_{n+1}-1}^{(j)} = \xi_{n+1}^{(j-1)}$$

for  $\lambda_{n,K_{n+1}-1}^{(j)}$

20: **end if**

21: **for**  $k = K_{n+1} - 1, \dots, K_n + 1$  **do**

22: solve

$$\frac{\partial J_k(y_{n,k}^{(j)}, u_{n,k}^{(j)})}{\partial y_{n,k}^{(j)}} + \left[ \frac{\partial \tilde{F}_{k-1}(y_{n,k-1}^{(j)}, y_{n,k}^{(j)}, u_{n,k-1}^{(j)})}{\partial y_{n,k}^{(j)}} \right]^T \lambda_{n,k-1}^{(j)} + \left[ \frac{\partial \tilde{F}_k(y_{n,k}^{(j)}, y_{n,k+1}^{(j)}, u_{n,k}^{(j)})}{\partial y_{n,k}^{(j)}} \right]^T \lambda_{n,k}^{(j)} = 0$$

for  $\lambda_{n,k-1}^{(j)}$

23: **end for**

24: **if**  $n > 0$  **then**

$$25: \quad \xi_n^{(j)} = - \left[ \frac{\partial J_{K_n}(y_{n,K_n}^{(j)}, u_{n,K_n}^{(j)})}{\partial y_{n,K_n}^{(j)}} + \left[ \frac{\partial \tilde{F}_{K_n}(y_{n,K_n}^{(j)}, y_{n,K_{n+1}}^{(j)}, u_{n,K_n}^{(j)})}{\partial y_{n,K_n}^{(j)}} \right]^T \lambda_{n,K_n}^{(j)} \right]$$

26: **end if**

(continued on next page)

---

---

```

27: for  $k = K_n, \dots, K_{n+1} - 1$  do ▷ update control
28:    $u_k^{(j+1)} = \mathcal{P}_{D_k}(u_{n,k}^{(j)} - \alpha_j \left[ \frac{\partial J_k(y_{n,k}^{(j)}, u_{n,k}^{(j)})}{\partial u_{n,k}^{(j)}} + \left( \frac{\partial \tilde{F}_k(y_{n,k}^{(j)}, y_{n,k+1}^{(j)}, u_{n,k}^{(j)})}{\partial u_{n,k}^{(j)}} \right)^T \lambda_{n,k}^{(j)} \right])$ 
29: end for

30: if  $n > 0$  then ▷ communication between processors
31:   send  $\xi_n^{(j)}$  to rank  $n - 1$ 
32:   receive  $y_{n-1, K_n}^{(j)}$  from rank  $n - 1$ 
33: end if

34: if  $n < N - 1$  then
35:   send  $y_{n, K_{n+1}}^{(j)}$  to rank  $n + 1$ 
36:   receive  $\xi_{n+1}^{(j)}$  from rank  $n + 1$ 
37: end if

```

---

subdomain boundaries where the parallel-in-time gradient-type algorithms do data communication and introduce information lag, since the related control and state



variables are the same for both algorithms.

$$- \left[ \frac{\partial \tilde{F}_k(y_{n,K_{n+1}-1}^{(j+1)}, y_{n,K_{n+1}}^{(j+1)}, u_{n,K_{n+1}-1}^{(j+1)})}{\partial y_{n,K_{n+1}}^{(j+1)}} \right]^T \lambda_{n,K_{n+1}-1}^{(j+1)} \quad (\text{C.2.2a})$$

$$= - \xi_{n+1}^{(j)} \quad (\text{C.2.2b})$$

$$= \frac{\partial J_{K_{n+1}}(y_{n+1,K_{n+1}}^{(j)}, u_{n+1,K_{n+1}}^{(j)})}{\partial y_{n+1,K_{n+1}}^{(j)}} + \left[ \frac{\partial \tilde{F}_{K_{n+1}}(y_{n+1,K_{n+1}}^{(j)}, y_{n+1,K_{n+1}+1}^{(j)}, u_{n+1,K_{n+1}}^{(j)})}{\partial y_{n+1,K_{n+1}}^{(j)}} \right]^T \lambda_{n+1,K_{n+1}}^{(j)} \quad (\text{C.2.2c})$$

$$= \frac{\partial J_{K_{n+1}}(y_{n+1,K_{n+1}}^{(j)}, u_{n+1,K_{n+1}}^{(j)})}{\partial y_{n+1,K_{n+1}}^{(j)}} + \left[ \frac{\partial \tilde{F}_{K_{n+1}}(y_{n+1,K_{n+1}}^{(j)}, y_{n+1,K_{n+1}+1}^{(j)}, u_{n+1,K_{n+1}}^{(j)})}{\partial y_{n+1,K_{n+1}}^{(j)}} \right]^T \cdot \left[ - \frac{\partial \tilde{F}_{K_{n+1}}(y_{n+1,K_{n+1}}^{(j)}, y_{n+1,K_{n+1}+1}^{(j)}, u_{n+1,K_{n+1}}^{(j)})}{\partial y_{n+1,K_{n+1}+1}^{(j)}} \right]^{-T} \gamma_{n+1,K_{n+1}}^{(j)} \quad (\text{C.2.2d})$$

$$= \frac{\partial J_{K_{n+1}}(y_{n+1,K_{n+1}}^{(j)}, u_{n+1,K_{n+1}}^{(j)})}{\partial y_{n+1,K_{n+1}}^{(j)}} + \left[ \frac{\partial F_{K_{n+1}}(y_{n+1,K_{n+1}}^{(j)}, u_{n+1,K_{n+1}}^{(j)})}{\partial y_{n+1,K_{n+1}}^{(j)}} \right]^T \gamma_{n+1,K_{n+1}}^{(j)} \quad (\text{C.2.2e})$$

$$= \gamma_{n+1,K_{n+1}-1}^{(j)} \quad (\text{C.2.2f})$$

$$= \gamma_{n,K_{n+1}-1}^{(j+1)} \quad (\text{C.2.2g})$$

The equality (C.2.2a)-(C.2.2b) is for Algorithm 18 Line 19, (C.2.2b)-(C.2.2c) for Algorithm 18 Line 25, (C.2.2c)-(C.2.2d) for (C.2.1) with  $i = j$ , (C.2.2d)-(C.2.2e) for implicit function theorem since the explicit state equation and the implicit state equation define the same relation ship between states and control, (C.2.2e)-(C.2.2f) for Algorithm 17 Line 22, (C.2.2f)-(C.2.2g) for Algorithm 17 Line 19.

Then, (C.2.1) is true for  $i = j + 1$ ,  $n = 0, \dots, N - 1$ , and  $k = K_n, \dots, K_{n+1} - 1$ . The mathematical induction completes with a proper base case assumption.

In conclusion, Algorithm 17 and Algorithm 18 are equivalent in the sense that produces exactly the same control iterates.

### C.3 Application to the Reservoir Problem

In the reservoir problem, saturation and pressure are state variables,

$$y_k = \begin{bmatrix} s_k \\ p_{k-1} \end{bmatrix}, \quad k = 1, \dots, K \quad (\text{C.3.1})$$

Use  $q$  as notation for control for compatibility with other parts of this thesis. Let  $H_k$  and  $G_k$  represent pressure and saturation equations respectively,

$$\tilde{F}_k(y_k, y_{k+1}, q_k) = \begin{bmatrix} H_k(s_k, p_k, q_k) \\ G_k(s_k, s_{k+1}, p_k, q_k) \end{bmatrix}, \quad k = 1, \dots, K - 1 \quad (\text{C.3.2})$$

Then, Algorithm 18 is used. Naturally, the resulting adjoint variables can be partitioned into two parts

$$\lambda_k = \begin{bmatrix} \mu_k \text{ (for pressure equation } H_k) \\ \eta_k \text{ (for saturation equation } G_k) \end{bmatrix}, \quad k = 1, \dots, K - 1 \quad (\text{C.3.3})$$

One special feature of this reservoir application is in the data communication at time subdomain boundaries. According to Algorithm 18 Line 25, with  $n < N - 1$ ,

$$\xi_n^{(j)} = - \left[ \frac{\partial J_{K_n}(y_{n,K_n}^{(j)}, q_{n,K_n}^{(j)})}{\partial y_{n,K_n}^{(j)}} + \left[ \frac{\partial \tilde{F}_{K_n}(y_{n,K_n}^{(j)}, y_{n,K_n+1}^{(j)}, q_{n,K_n}^{(j)})}{\partial y_{n,K_n}^{(j)}} \right]^T \lambda_{n,K_n}^{(j)} \right] \quad (\text{C.3.4})$$

In the reservoir problem, objective function value does not depend on pressure and therefore the gradient of the objective with respect to pressure is zero,

$$\frac{\partial J_{K_n}(y_{n,K_n}^{(j)}, q_{n,K_n}^{(j)})}{\partial y_{n,K_n}^{(j)}} = \begin{bmatrix} \frac{\partial J_{K_n}(y_{n,K_n}^{(j)}, q_{n,K_n}^{(j)})}{\partial s_{n,K_n}^{(j)}} \\ 0 \end{bmatrix} \quad (\text{C.3.5})$$

The Jacobian matrix has special structure

$$\begin{aligned}
& \frac{\partial \tilde{F}_{K_n}(y_{n,K_n}^{(j)}, y_{n,K_n+1}^{(j)}, q_{n,K_n}^{(j)})}{\partial y_{n,K_n}^{(j)}} \\
&= \begin{bmatrix} \frac{\partial H_{K_n}(s_{n,K_n}^{(j)}, p_{n,K_n}^{(j)}, q_{n,K_n}^{(j)})}{\partial s_{n,K_n}^{(j)}} & \frac{\partial H_{K_n}(s_{n,K_n}^{(j)}, p_{n,K_n}^{(j)}, q_{n,K_n}^{(j)})}{\partial p_{n,K_n-1}^{(j)}} \\ \frac{\partial G_{K_n}(s_{n,K_n}^{(j)}, s_{n,K_n+1}^{(j)}, p_{n,K_n}^{(j)}, q_{n,K_n}^{(j)})}{\partial s_{n,K_n}^{(j)}} & \frac{\partial G_{K_n}(s_{n,K_n}^{(j)}, s_{n,K_n+1}^{(j)}, p_{n,K_n}^{(j)}, q_{n,K_n}^{(j)})}{\partial p_{n,K_n-1}^{(j)}} \end{bmatrix} \quad (\text{C.3.6}) \\
&= \begin{bmatrix} \frac{\partial H_{K_n}(s_{n,K_n}^{(j)}, p_{n,K_n}^{(j)}, q_{n,K_n}^{(j)})}{\partial s_{n,K_n}^{(j)}} & 0 \\ \frac{\partial G_{K_n}(s_{n,K_n}^{(j)}, s_{n,K_n+1}^{(j)}, p_{n,K_n}^{(j)}, q_{n,K_n}^{(j)})}{\partial s_{n,K_n}^{(j)}} & 0 \end{bmatrix}
\end{aligned}$$

By (C.3.3), (C.3.4), (C.3.5), and (C.3.6), in the adjoint data communication, one does not need to exchange the full  $\xi_n^{(j)}$ , since the lower half of it is zero, but only need to exchange

$$\begin{aligned}
& - \left[ \frac{\partial J_{K_n}(y_{n,K_n}^{(j)}, q_{n,K_n}^{(j)})}{\partial s_{n,K_n}^{(j)}} + \left[ \frac{\partial H_{K_n}(s_{n,K_n}^{(j)}, p_{n,K_n}^{(j)}, q_{n,K_n}^{(j)})}{\partial s_{n,K_n}^{(j)}} \right]^T \mu_{n,K_n}^{(j)} \right. \\
& \quad \left. + \left[ \frac{\partial G_{K_n}(s_{n,K_n}^{(j)}, s_{n,K_n+1}^{(j)}, p_{n,K_n}^{(j)}, q_{n,K_n}^{(j)})}{\partial s_{n,K_n}^{(j)}} \right]^T \eta_{n,K_n}^{(j)} \right]
\end{aligned}$$

as a summation involving adjoint variables corresponding to both pressure and saturation equation.

# Appendix D

## A Test on Adjusting Time Subdomains for Load Balance

I present a simple test on adjusting the time subdomains for the parallel-in-time gradient-type method to achieve better computation load balance across subdomains.

The parallel-in-time gradient-type method performs parallel forward/backward computing on different time subdomains in parallel. With evenly split time subdomains, in each time subdomain, the time steps of computation are the same. However, the computation load is not necessarily the same as demonstrated in the reservoir optimization example in Section 6.3.2.1 and Section 6.3.2.2, where some part of the time domain takes longer computation time than others. This prevents the parallel method from obtaining good efficiency, since processors that finishes their computation tasks on their subdomain waits idly for processors working subdomains with heavier computation load, see Figure D.2.

It is natural to attempt to remedy this load imbalance by repartition the time domain not by number of time steps, but by computation load.

Now, using the model problem in Section 6.3.2.2, I test this idea.

The problem has 1000 time steps, the evenly split subdomains are  $[0, 250]$ ,  $[250, 500]$ ,  $[500, 750]$ ,  $[750, 1000]$ . By a timed test run using initial well rates

control, I find the adjusted subdomain partition that roughly distribute the workload, including forward and backward computation, across subdomains. The adjusted subdomains are  $[0, 196]$ ,  $[196, 442]$ ,  $[442, 720]$ ,  $[720, 1000]$ . See Figure D.1.

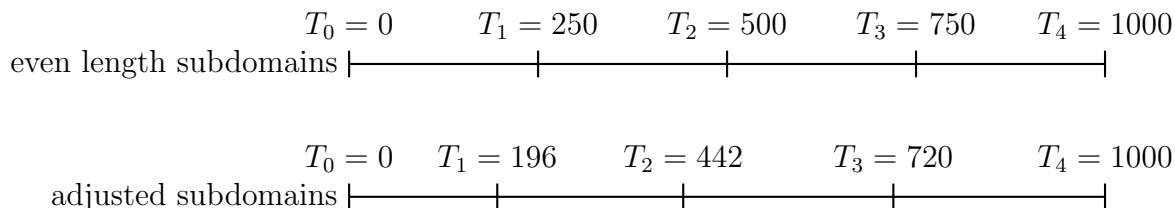


Figure D.1: Time Subdomains Adjustment

Figure D.2 and Figure D.3 show the 1000 seconds trace plot of the evenly split subdomain case and the adjusted subdomain case respectively. It can be seen that by comparison of these two plots,

- the significant waiting time (green) in the even subdomain case is mostly avoided in the adjusted subdomain case.
- In the 1000 seconds, the adjusted subdomain case runs more iterations. As a matter of fact, completing 49 iterations, The even subdomain case takes 6077 seconds and the adjusted case takes 5307 seconds.

$$\frac{5307}{6077} = 87.3\% \quad (\text{D.0.1})$$

By adjusting the partition, around 10% of time is saved in terms of competing a fixed number of iterations.

- In the adjust subdomain case, the backward computation in the 3rd and 4th subdomains takes longer to finish than that in the 1st and 2nd subdomain. It is because the 3rd and 4th subdomain has more time steps and the time consumed by a step of backward computation is roughly the same no matter it is at an earlier or later part of the time domain.

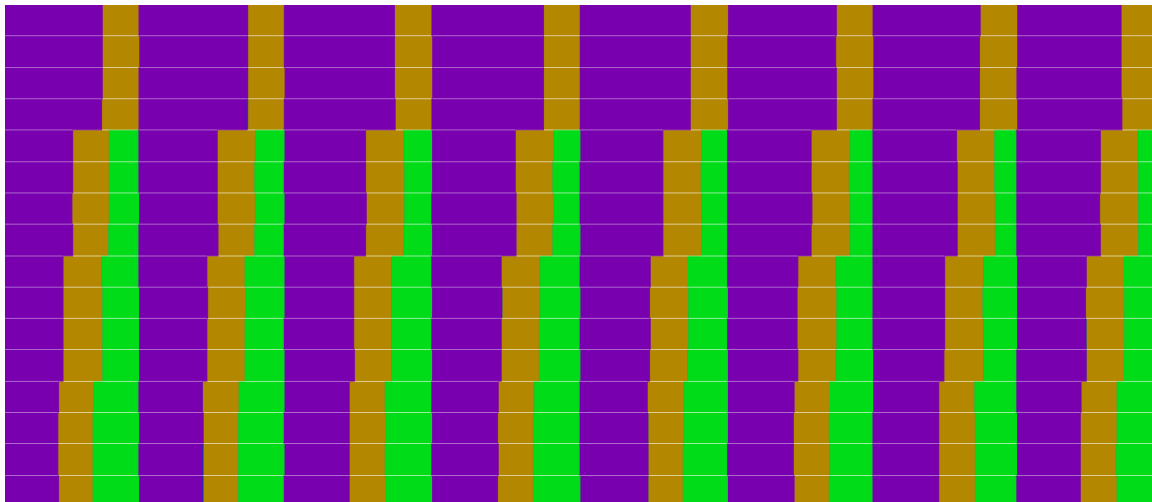


Figure D.2: Trace graph of 1000 seconds about 8 optimization iterations with evenly split time subdomains, generated by `HPCTOOLKIT` [ABF<sup>+</sup>10].

The time subdomains are  $[0, 250]$ ,  $[250, 500]$ ,  $[500, 750]$ ,  $[750, 1000]$ . The horizontal axis is the time axis. From top to bottom, the 16 rows corresponding to 16 cores are grouped into 4 groups. The top 4 horizontal rows represent computation timing of the 4 cores performing parallel computing in the first time subdomain, etc. One optimization iteration consists of three major blocks of one purple, one brown, and one green block. Purple blocks are for forward computation, brown blocks are for backward computation, and green blocks are for waiting. Model problem description in Section 6.3.2.2.

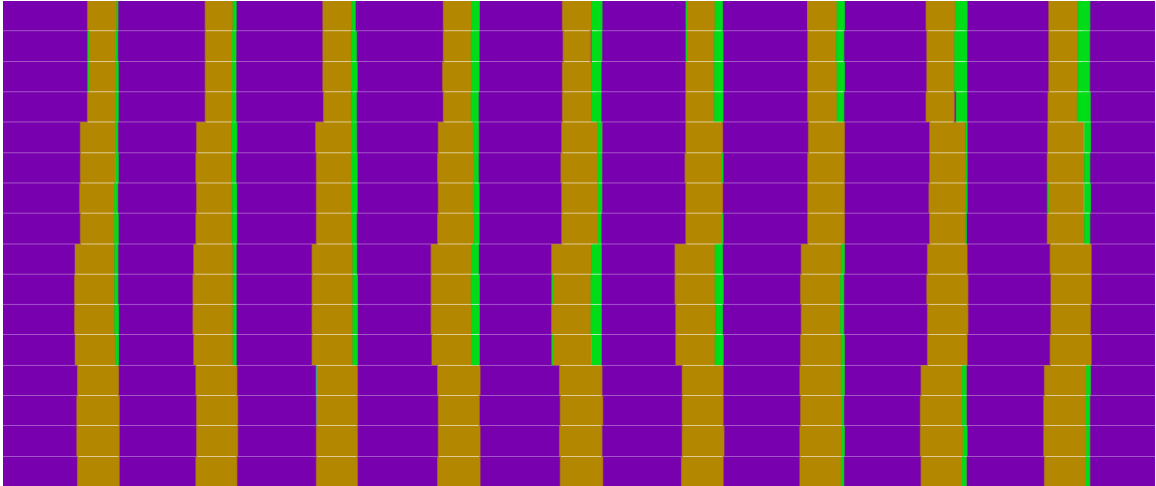


Figure D.3: Trace graph of 1000 seconds about 9.5 optimization iterations with adjusted time subdomains, generated by HPCTOOLKIT [ABF<sup>+</sup>10].

The adjusted time subdomains are  $[0, 196]$ ,  $[196, 442]$ ,  $[442, 720]$ ,  $[720, 1000]$ . The horizontal axis is the time axis. From top to bottom, the 16 rows corresponding to 16 cores are grouped into 4 groups. The top 4 horizontal rows represent computation timing of the 4 cores performing parallel computing in the first time subdomain, etc. One optimization iteration consists of three major blocks of one purple, one brown, and one green block. Purple blocks are for forward computation, brown blocks are for backward computation, and green blocks are for waiting. Model problem description in Section 6.3.2.2.

The adjustment of the partition of time subdomains also effects the gradient-type vector computed. However, in this case, there is not a significant difference in the objective function value history in the optimization process for the two kinds of different partition, see Figure D.4.

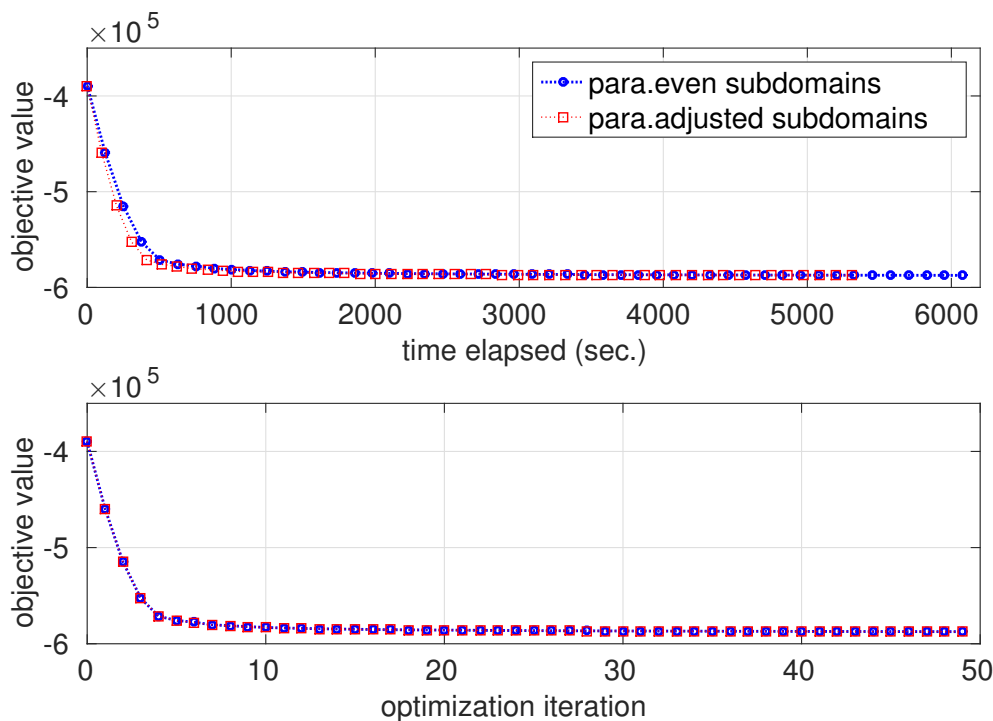


Figure D.4: Comparison of objective function value history between the parallel-in-time gradient-type method with different time subdomain partitions. The evenly split time subdomains are  $[0, 250]$ ,  $[250, 500]$ ,  $[500, 750]$ ,  $[750, 1000]$ ; The adjusted time subdomains are  $[0, 196]$ ,  $[196, 442]$ ,  $[442, 720]$ ,  $[720, 1000]$ . The first initialization step of a full gradient-sweep is the same for both cases and is not included in the plot. Model problem description in Section 6.3.2.2.

In conclusion, in this test problem, by adjusting the partition of the time subdomains according to the computation load, instead of number of time steps, a small portion of computation time (about 10% according to (D.0.1)) can be saved.



Above is the test result. I do not provide further discussion.