

© 2017 Daniel Ronald Herber

ADVANCES IN COMBINED ARCHITECTURE, PLANT, AND CONTROL DESIGN

BY

DANIEL RONALD HERBER

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Systems and Entrepreneurial Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Doctoral Committee:

Assistant Professor James Allison
Professor Yuliy Baryshnikov
Associate Professor Carolyn Beck
Professor Harrison Kim

Abstract

The advancement of many engineering systems relies on novel design methodologies, design formulations, design representations, and other advancements. In this dissertation, we consider three broad design domains: architecture, plant, and control. These domains cover most of the potential design decision elements in an actively-controlled engineering system. In this dissertation, strategic aspects of this combined problem are addressed.

The task of representing and generating candidate architectures is addressed with methods developed based on colored graphs built by enumerating all perfect matchings of a specified catalog of components. The proposed approach captures all architectures under specific assumptions. General combined plant and control design (or co-design) problems are examined. Previous work in co-design theory imposed restrictions on the type of problems that could be posed. Here many of those restrictions are lifted. The problem formulations and optimality conditions for both the simultaneous and nested solution strategies are given along with a detailed discussion of the two methods. Direct transcription is also discussed as it enables the solution of general co-design problems by approximating the problem. Motivated primarily by the need for efficient methods to solve certain control problems that emerge using the nested co-design method, an automated problem generation procedure is developed to support easy specification of linear-quadratic dynamic optimization problems using direct transcription and quadratic programming. Pseudospectral and single-step methods (including the zero-order hold) are all implemented in this unified framework and comparisons are made.

Three detailed engineering design case studies are presented. The results from the enumeration and evaluation of all passive analog circuits with up to a certain num-

ber of components are used to synthesize low-pass filters and circuits that match a certain magnitude response. Advantages and limitations of enumerative approaches are highlighted in this case study, along with comparisons to circuits synthesized via evolutionary computation; many similarities are found in the topologies. The second case study tackles a complex co-design problem with the design of strain-actuated solar arrays for spacecraft precision pointing and jitter reduction. Nested co-design is utilized along with a linear-quadratic inner loop problem to obtain solutions efficiently. A simpler, scaled problem is analyzed to gain general insights into these results. This is accomplished with a unified theory of scaling in dynamic optimization. The final case study involves the design of active vehicle suspensions. All three design domains are considered in this problem. A class of architecture, plant, and control design problems which utilize linear physical elements is discussed. This problem class can be solved using the methods in this dissertation.

To Ashley.

Acknowledgments

First and foremost, it has been my honor to have been advised by Professor James T. Allison for the past six years. His enthusiasm and guidance have provided me with continued inspiration throughout my graduate studies. A fortuitous opportunity as an undergraduate grader was the start of a great relationship that defined my path and launched my successes. I would also like to thank my dissertation committee for their time and valuable feedback.

Most importantly, my wife, Ashley, has provided unwavering support over the past decade. She has always been right by my side through many years of schooling, offering encouragement and the *occasional* copyediting. I also want to thank all my family and friends for their support throughout this exciting and laborious journey.

I have also had the opportunity to work with many brilliant collaborators including Prof. Soon-Jo Chung, Jack Aldrich and Oscar Alvarez-Salazar from JPL, and Traci Spencer and Emily Horn from John Deere. There are many Engineering System Design Lab (ESDL) members that have contributed in numerous ways to my success as a graduate student including: Jeff Arena, Madhav Arora, Andrew Blanco, Christian Chilan, Anand Deshmukh, Tinghao Guo, Yong Hoon Lee, Shangting Li, Danny Lohan, Jason McDonald, Marlon Mitchell, Yashwanth Nakka, Albert Patterson, Satya Peddada, Lakshmi Rao, Vedant, and many others. I am grateful to have worked with you and I cherish our time together.

Some elements of this dissertation were produced as the result of collaborative efforts. Tinghao contributed to parts in Chapter 2 on architecture theory and the supporting code. Yong Hoon provided assistance in both the development of Chapter 5 on linear-quadratic dynamic optimization using direct transcription and the supporting code. The case study in Chapter 7 was a collaborative effort among a

number of coauthors with Christian as the primary lead and Yashwanth developing the beam theory model.

Finally, I would like to acknowledge support from Deere & Company, the Jet Propulsion Laboratory (JPL), the NSF center for Power Optimization of Electro-Thermal Systems (POETS), the Department of Industrial and Enterprise Systems Engineering, and the Mavis Future Faculty Fellow program.

Contents

List of Figures	x
List of Tables	xiv
List of Algorithms	xvi
List of Symbols	xvii
List of Abbreviations	xviii
List of General Notation	xix
Chapter 1 Introduction	1
1.1 Introductory Example	1
1.2 Three Design Domains	3
1.3 A Design Process for Complete Dynamic System Design	8
1.4 Solution Generation Challenges	9
1.5 Dissertation Overview	11
Chapter 2 Candidate Architectures through Enumeration	14
2.1 Candidate Architectures with Perfect Matchings	16
2.2 Candidate Graphs to Unique Useful Graphs	23
2.3 Tree Search Algorithm	30
2.4 Enumeration Case Studies	32
2.5 Discussion	40
2.6 Summary	42
Chapter 3 Co-Design: Combined Plant and Control Design	44
3.1 Problem Formulation	46
3.2 Necessary Conditions for Optimality	51
3.3 Practical Solution Considerations	56
3.4 Test Problems	61
3.5 Summary	66

Chapter 4 Scaling of Dynamic Optimization Formulations	68
4.1 Introduction	68
4.2 Theory of Scaling Dynamic Optimization Formulations	71
4.3 Motivating Examples	76
4.4 Summary	88
Chapter 5 Direct Transcription and Linear-Quadratic Dynamic Optimization.....	90
5.1 Introduction	90
5.2 Linear-Quadratic Dynamic Optimization	91
5.3 Approximate Solutions with Direct Transcription	97
5.4 Automated Problem Generation	106
5.5 Extensions	113
5.6 Numerical Examples	118
5.7 Future Work	131
5.8 Summary	135
Chapter 6 Case Study: Design of Passive Analog Circuits	137
6.1 Introduction	137
6.2 Enumeration-Based Synthesis Methodology	139
6.3 Examples	149
6.4 Discussion	160
6.5 Summary	163
Chapter 7 Case Study: Design of Strain-Actuated Solar Arrays	164
7.1 Introduction	164
7.2 Modeling of the Strain-Actuated Solar Arrays and Rigid Spacecraft Bus	167
7.3 Co-Design Problem Formulation	173
7.4 Analytical and Numerical Results for SASA System	178
7.5 Summary	188
Chapter 8 Case Study: Design of Vehicle Suspensions	191
8.1 Introduction	191
8.2 A Problem Class with Linear Physical Elements	193
8.3 Problem Formulation	197
8.4 Results	199
8.5 Summary	201
Chapter 9 Conclusions and Future Work	205
9.1 Summary	205
9.2 Contributions	207
9.3 Future Work	208

Appendix A Enhancements to the Perfect Matching-based Tree Algorithm for Generating Architectures	211
A.1 Overview	211
A.2 Replicate Ordering	212
A.3 Avoiding Loops	215
A.4 Avoiding Multi-Edges	219
A.5 Avoiding Line-Connectivity Constraints	221
A.6 Checking for Saturated Subgraphs	225
A.7 Enumerating Subcatalogs	230
A.8 Alternative Tree Traversal Strategies	234
A.9 Case Studies from Chapter 2	237
Appendix B Additional Architectures/Graphs	239
Appendix C Additional Material for Chapter 5	241
C.1 Algorithms in the Automated Problem Generation Procedure	241
C.2 Sparsity Patterns	249
C.3 Codes	252
C.4 Methods	255
Appendix D Summary of Available Code	263
Bibliography	264

List of Figures

1.1	Task description in the robotic manipulator example.	2
1.2	Some candidate robot manipulator architectures.....	2
1.3	Plant variables in the robotic manipulator example.	3
1.4	Joint control trajectories in the robotic manipulator example..	3
1.5	Proposed stages for complete dynamic system design.	8
1.6	Stage 1 details.	9
1.7	Dissertation content connections	13
2.1	Architectures represented as graphs.	15
2.2	Complete graphs on n vertices between 1 and 5.	18
2.3	Perfect matchings for K_2 , K_4 , and K_6	20
2.4	Comparison between a PM approach and adjacency matrix approach.	27
2.5	Two different type of isomorphisms.....	28
2.6	Tree structure for Case Study 1 using the basic tree search algorithm	31
2.7	G^P graphs for two examples.	33
2.8	Select interconnectivity graphs for Case Study 1.	33
2.9	Select connected ports and connected component graphs for Case Study 1. ..	34
2.10	All 16 unique graphs with no additional NSCs for Case Study 1.	35
2.11	All 5 unique graphs for Case Study 1 requiring a connected graph containing B and a specified number of unique edges.	35
2.12	All 12 unique graphs for Case Study 2 requiring all components to be connected and a specified number of unique edges.	36
2.13	Suspension architecture enumeration case study.	37
2.14	Suspension case study matrices for S_7 and the tree search algorithm.	39
2.15	Two architectures for the suspension case study.	40
3.1	Two co-design solution strategies	47
3.2	Scalar plant, scalar control problem (TP1) results	62
3.3	Co-design transfer problem (TP2) results.....	64
3.4	Simple SASA test problem (TP3) results	65
4.1	Spring-mass system	74
4.2	Relationships between the original and scaled problems	76

4.3	Illustrations of original and simplified strain-actuated solar array systems	77
4.4	Scaled and unscaled solutions for the simple SASA problem	82
4.5	Natural period vs. t_f for the simple SASA and original design study	83
4.6	Co-design transfer problem	84
4.7	Maximum absolute relative error vs. step size for defect constraints	87
5.1	Structure definitions	107
5.2	Overview of the automated problem generation procedure	110
5.3	Solution for Example 1	120
5.4	Numerical results for Example 1	121
5.5	Solution for Example 2	123
5.6	Numerical results for Example 2	124
5.7	Solutions for Example 3	125
5.8	Numerical results for Example 3	127
5.9	Solution for Example 4	128
5.10	Numerical results for Example 4	129
5.11	Solution for Example 5	131
6.1	Enumeration-based synthesis methodology	140
6.2	Different representations used for the same circuit	141
6.3	14 primitive circuits for the circuit structure space defined by Eqn. (6.1)	145
6.4	Subcircuits considered for generating practical circuits (series RLC subcircuits)	145
6.5	Two common template circuits	146
6.6	Performance vs. complexity for Frequency Response Matching example using set 1 along with select Pareto optimal circuits	151
6.7	Magnitude and errors over the desired frequency range using select circuits	152
6.8	Performance vs. complexity for Frequency Response Matching example using set 2	153
6.9	Performance vs. cumulative percentage for Frequency Response Matching example	153
6.10	Low-pass filter specifications	154
6.11	All feasible, minimum complexity circuits and attenuation responses for Low-Pass Filter Realizability task #1	157
6.12	Select feasible, minimum complexity circuits and attenuation responses for Low-Pass Filter Realizability task #2	158
6.13	Select feasible, minimum complexity circuits and attenuation responses for Low-Pass Filter Realizability task #3	159
6.14	Top two closest to be feasible circuits for Low-Pass Filter Realizability task #4	159
7.1	Illustration of the two modeling approaches used to gain design insights	167

7.2	Illustrations of various design representations for internally actuated array design problems for pointing.	169
7.3	Cross section of the actuated array, modeled as a composite beam.	171
7.4	Parametric study of maximum slewing angle with respect to slewing time.	180
7.5	Optimal array designs.	181
7.6	Bus angle and angular rate trajectories for select values of \bar{t}	182
7.7	Voltage history along the array for select values of \bar{t}	183
7.8	Array deflection profiles for select values of \bar{t}	185
7.9	Comparison between the first natural period of the optimal array designs and the slewing phase duration.	186
7.10	Scaled mode coefficient trajectories for select values of \bar{t}	187
7.11	Voltage trajectory metrics for select values of \bar{t}	190
8.1	Various vehicle suspension architectures.	192
8.2	The proposed trilevel solution strategy for combined architecture, plant, and control design.	196
8.3	Suspension architecture component catalog.	197
8.4	Optimal trajectories for the two passive suspensions.	202
8.5	Optimal trajectories for two suspensions.	203
8.6	Optimal trajectories for two suspensions including the current best architecture.	204
A.1	Example 1 for Algorithm A.2 (replicate ordering).	214
A.2	Example 2 for Algorithm A.2 (replicate ordering).	216
A.3	Illustration of a line-connectivity constraint.	221
A.4	Example 1 for Algorithm A.5 (line-connectivity constraints).	223
A.5	Example 2 for Algorithm A.5 (line-connectivity constraints).	225
A.6	Example 1 for Algorithm A.6 (saturated subgraphs).	228
A.7	Example 2 for Algorithm A.6 (saturated subgraphs).	229
A.8	Two tree traversal strategies.	235
A.9	Visualization of the impact of level-order isomorphism checking.	236
A.10	Parallelization example.	237
B.1	All ten minimum complexity topologies for Low-Pass Filter Realizability task #3.	239
B.2	All 274 graphs in Case Study 2 with no additional NSCs (gray hash indicates a multiedge).	240
C.1	Sparsity pattern of \mathbf{H} matrix for all considered methods except CQHS.	249
C.2	Sparsity pattern of the off-diagonal terms of $\{\mathbf{L}_{11}, \mathbf{L}_{12}, \mathbf{L}_{21}, \mathbf{L}_{22}\}$ in the \mathbf{H} matrix using the CQHS method.	250
C.3	Sparsity pattern of Mayer terms in \mathbf{H} matrix.	250

C.4	Sparsity pattern of \mathbf{A}_{e1} matrix for the defect constraints using a single-step method.....	251
C.5	Sparsity pattern of \mathbf{A}_{e1} matrix for the defect constraints using a pseudospectral method	251
C.6	LGL and CGL inner node locations from the roots of polynomials	257
C.7	ED, CGL, and LGL nodes for various values of N_t	258
C.8	Lagrange polynomial interpolation with ED, LGL, and CGL nodes	258
C.9	Convergence behavior for definite integral and derivative approximations using Lagrange interpolation with LGL nodes.....	259
C.10	Differentiation error using Lagrange interpolation with LGL nodes	260

List of Tables

6.1	Computational cost of Frequency Response Matching example	150
6.2	Performance comparison between best and arbitrary circuit transfer functions.	152
6.3	Computational cost of Low-Pass Filter Realizability example.	155
6.4	Low-Pass Filter Realizability specifications and results.	156
6.5	Task #1 feasible vs. total number of circuits for different complexity levels. .	157
7.1	Problem physical parameters.....	174
7.2	Summary of co-design problem formulation.	177
7.3	Summary of results for maximum slewing bounds using the PRBDM.	179
7.4	Geometric constraints for the co-design problem variations.	180
8.1	Some bond graph modeling analogies.	194
8.2	Co-design problem parameters.	199
8.3	Summary of the suspension design results.	200
A.1	Comparison (replicate ordering, Example 1).	215
A.2	Comparison (replicate ordering, Example 2).	215
A.3	Comparison (loops, Example 1).....	218
A.4	Comparison (loops, Example 2).....	218
A.5	Comparison (multi-edge).	221
A.6	Comparison (line-constraints, Example 1).	223
A.7	Comparison (line-constraints, Example 2).	224
A.8	Comparison (saturated subgraphs, Example 1).	228
A.9	Comparison (saturated subgraphs, Example 2).	229
A.10	Subcatalogs for Example 1.	233
A.11	Comparison (enumerating subcatalogs, Example 1).	233
A.12	Select subcatalogs for Example 2.	234
A.13	Comparison (enumerating subcatalogs, Example 2).	234
A.14	Example 1.	238
A.15	Comparison (Case Study 1).	238
A.16	Comparison (Case Study 2).	238
A.17	Comparison (Case Study 3).	238

C.1 Notation used in the algorithms.	241
---	-----

List of Algorithms

2.1	Creation of edge set for a specific perfect matching number.	22
2.2	Determination of the perfect matching number for a specific edge set.	23
2.3	Determination of the unique colored graphs given a set of colored graphs. ..	30
2.4	Basic tree search algorithm.	32
A.1	Original tree search algorithm.	212
A.2	Limit potential connections based on replicate ordering.	213
A.3	Limit potential connections based on loops.	217
A.4	Limit potential connections based on multi-edges.	219
A.5	Limit potential connections based on line-connectivity constraints.	222
A.6	Handle saturated subgraphs.	226
A.7	Generate set of unique, feasible graphs using subcatalogs.	232
C.1	Optimization variable index generating functions	242
C.2	Create Hessian.	242
C.3	Create sequences for Lagrange terms.	243
C.4	Create sequences for Mayer terms.	244
C.5	Create matrices for the defect constraints using a single-step method.	245
C.6	Create matrices for the defect constraints using a pseudospectral method. ...	246
C.7	Create matrices for the additional inequality (or equality) constraints.	247
C.8	Create sequences for path constraint terms.	248
C.9	Create sequences for boundary constraint terms.....	248

List of Symbols

Symbol	Description	Page
\mathbf{a}	set of additional general constraints, a are individual constraints	147
α	scale factor	72
A	adjacency matrix of graph G , a are individual vertices	16
\mathbf{A}	QP constraint linear term matrix	92
\mathbf{A}	state (or system) matrix	58
b	damping constant	173
β	translation factor	72
B	line-connectivity constraint	222
\mathbf{B}	QP constraint constant term matrix	92
\mathbf{B}	input matrix	58
c	QP objective constant term	92
$c(x)$	distributed proportionality coefficient between moment and voltage	172
C	colored label set representing distinct component types	17
C	capacitance	149
\mathbf{C}	output state matrix	115
\mathbf{C}	path constraints, C are individual path constraints	46
\mathbf{d}	disturbance	58
δ	array angle	172
$d(G)$	degree of a graph G	17
\mathbf{D}	differentiation matrix	100
\mathbf{D}	output control matrix	115
$\mathcal{D}(n)$	double factorial function, $(n - 1)!!$	19
$D(s)$	denominator of a transfer function	146
Δ	vector of time steps, $\Delta_k = t_{k+1} - t_k$	86
e	column vector of ones of appropriate length	18
ϵ	constraint tolerance	86
$\epsilon(x, t)$	array surface strain	175
e_r	relative error	87
E	edges in graph G , e are individual edges	16
E	error	147
\mathbf{E}	descriptor matrix	94

Symbol	Description	Page
$E(x)$	total Young's modulus of the composite array	168
\mathcal{E}	extremum function	114
f	function (derivative, map, etc.), f is a single function	14
f_p	passband frequency	156
f_s	stopband frequency	156
\mathbf{F}	discretized derivatives	98
\mathbf{F}	QP objective linear term	92
\mathbf{F}	outer-loop feasibility constraint	47
\mathcal{F}	feasible set	14
\mathbf{g}	inequality constraints, g are individual inequality constraints	47
$g(\omega, \mathbf{x})$	circuit model function	147
G	graph or transfer function of a complete circuit	16
\mathbf{G}	parameter matrix	94
\mathcal{G}	graph structure space	19
\mathbf{h}	equality constraints, h are individual equality constraints	92
\mathbf{h}	thicknesses of the PLS, h are individual thicknesses	171
h_n	neutral axis	171
H	Hamiltonian	52
\mathbf{H}	Hessian or QP objective quadratic term	92
I	induced region	48
\mathbf{I}	integral matrix	113
$I(x)$	total second moment of area of the composite array	168
J	mass moment of inertia	168
k	spring stiffness	77
K_n	complete graph of size n	18
K_p	passband gain	156
K_s	stopband gain	156
\mathbf{l}	matrix for linear Lagrange terms	94
\mathbf{l}	lower bounds on the optimization variables	147
ℓ	array segment boundaries	171
ℓ	array length	168
L	colors or labels in graph G , l are individual colors	17
L	inductance	156
\mathbf{L}	matrix for quadratic Lagrange terms	94
\mathcal{LB}	lower bound structure	108
\mathcal{L}	Lagrange term or running cost	46
m	mass	172
\mathbf{m}	matrix for linear Mayer terms	95
$m_R(x)$	mass per unit length of the composite array	168
M	potential inner-loop controls	48
M	mandatory component constraint	24

Symbol	Description	Page
M	matrix for quadratic Mayer terms	95
$M(x, t)$	moment applied on the solar array	168
\mathcal{M}	Mayer term or terminal cost	47
n	count of some quantity	16
n_a	number of constraints	148
n_c	number of components	146
n_p	number of poles	152
ν	multipliers for the boundary constraints	52
ν	single-step method constants	100
n_z	number of zeros	152
$N(s)$	numerator of a transfer function	146
O	output weighting matrix	115
O	optimality conditions	74
p	time-independent parameters (sometimes control parameters)	49
ρ	density	172
ρ	problem parameters	72
p_x	total number of ports for component-type x in the subcatalog	144
P	problem formulation	74
P	ports vector for each component type	17
\mathbf{P}	solution to algebraic Riccati equation	59
\mathcal{P}	problem class	195
$\mathcal{P}(n, m)$	edge set for the n th perfect matching of K_m	19
$\mathbf{q}(t)$	dynamic coefficients for control basis functions	170
r	spacecraft bus radius	168
r_{\max}	maximum rattlespace	198
r_k	individual residuals	147
R	vector indicating the number of replicates for each component type	17
R	resistance	149
R_{eff}	effective inertia ratio of the bus-array system	178
R_l	load resistance	146
R_s	source resistance	155
s	frequency	146
\mathbf{s}	sources	193
S	network structure constraint or scaling function	24
t	time continuum	46
t_m	time duration of the first phase	173
τ	tree (graph)	234
\mathbf{t}	discretized time continuum	57
θ	spacecraft bus angle	77
$\boldsymbol{\theta}$	single-step method variable weights	100
T_1	period of the first natural frequency	77

Symbol	Description	Page
$\mathcal{T}(n)$	number of permutations of an undirected adjacency matrix	26
\mathbf{u}	open-loop control variables, u are individual controls	8
\mathbf{u}	upper bounds on the optimization variables	147
U	unique connections constraint	217
\mathbf{U}	discretized controls	57
\mathcal{UB}	upper bound structure	108
v_s	source voltage	155
V	vertices in graph G , v are individual vertices	16
V	voltage	172
\mathbf{V}	output parameter matrix	115
w	weights (quadrature, multi-objective optimization, etc.)	49
w	array width	168
x	independent variable	72
x	spatial variable	169
\mathbf{x}	design variables	8
$\tilde{\mathbf{x}}$	expanded set of optimization variables	93
$\boldsymbol{\chi}$	multipliers for the outer-loop constraints	54
\mathbf{X}	set of QP optimization variables	92
\mathcal{X}	total angular momentum	173
y	dependent variable	72
\mathbf{y}	outputs	115
\mathbf{Y}	matrix for linear equality constraints	95
\mathcal{Y}	equality constraint structure	108
z_0	road profile	191
Ω	simultaneous method's feasible set (or constraint region)	48
Θ	augmented states	53
ϕ	boundary constraints, ϕ are individual boundary constraints	46
$\boldsymbol{\lambda}$	costates or multipliers for the state dynamics	52
μ	damping coefficient of the composite array	169
$\xi(x, t)$	solar array deflection	168
Ξ	discretized states	57
$\boldsymbol{\eta}(t)$	dynamic coefficients for Galerkin approximating functions	170
Ω	set of n_s frequency points	147
$\gamma(x)$	control basis functions	170
ψ	mode shapes, ψ are individual mode shapes	173
$\boldsymbol{\eta}$	multipliers for the outer-loop feasibility constraints	54
$\boldsymbol{\mu}$	multipliers for the path constraints	52
ψ	outer-loop objective function	47
Ψ	objective function	8
ω	frequency	147
$\boldsymbol{\omega}$	natural frequencies	173

Symbol	Description	Page
$\phi(x)$	Galerkin approximating functions	170
ξ	states, ξ are individual states	46
$\Phi(t, \tau)$	state-transition matrix	94
Z	matrix for linear inequality constraints	95
\mathcal{Z}	inequality constraint structure	108
ζ	defect constraints	57
Z	graph colorings or labels	17
<code>var</code>	variable in code	22

List of Abbreviations

Abbreviations	Description	Page
APGP	automated problem generation procedure	91
ARE	algebraic Riccati equation	59
BFS	breadth-first search	235
BVP	boundary value problem	56
CC	Clenshaw-Curtis	103
CCW	counter clockwise	168
CEF	composite Euler forward	104
CGL	Chebyshev-Gauss-Lobatto (nodes)	98
CLC	closed-loop control	6
CNC	component number constraint	23
CQHS	composite quadratic Hermite-Simpson	105
CTR	composite trapezoidal rule	104
DAE	differential-algebraic equation	56
DCC	direct connection constraint	23
DFS	depth-first search	235
DO	dynamic optimization	45
DT	direct transcription	46
ED	equidistant (nodes)	98
EF	Euler forward	100
En	enhancements included	211
FOC	fan out constraint	24
G	Gaussian	103
GNRS	graph numerical representation scheme	21
HS	Hermite-Simpson	100
ICC	indirect connection constraint	24
ISB	indexed stacked block	26
KKT	Karush-Kuhn-Tucker (conditions)	53
LGL	Legendre-Gauss-Lobatto (nodes)	98
LP	linear program	176
LPF	low-pass filter	154
LQDO	linear-quadratic dynamic optimization	90

Abbreviations	Description	Page
LQR	linear-quadratic regulator	49
LTI	linear time-invariant	94
LTV	linear time-varying	94
MNA	modified nodal analysis	10
MPC	model-predictive control	96
NG	nominal geometry	180
NLDO	nonlinear dynamic optimization	90
NLP	nonlinear program	46
NLS	nonlinear least-squares	147
NSC	network structure constraint	23
ODE	ordinary differential equation	71
OLC	open-loop control	6
Orig	original algorithm	211
PDE	partial differential equation	166
PEMA	piezoelectric material actuator	165
PLS	piecewise linear segments	180
PM	perfect matching	19
PMP	Pontryagin's minimum principle	52
PRBDM	pseudo-rigid body dynamic model	166
PS	pseudospectral	99
QCQP	quadratically-constrained quadratic program	135
QP	quadratic program	58
RK4	4th-order classical Runge-Kutta	100
SASA	strain-actuated solar array	164
SISO	single-input single-output (transfer function)	141
SS	single-step	99
TP	test problem	61
TR	trapezoidal	100
VL	variable length	180
ZOH	zero-order hold	102

List of General Notation

Notation	Description	Page
\circ	Hadamard product	18
\square	upper bound	92
$\underline{\square}$	lower bound	92
$ \square $	absolute value or cardinality of a set	18
$\bar{\square}$	scaled	72
$\dot{\square}$	time derivative	46
$\square!!$	double factorial	142
\square^C or \square_C	component	16
\square^P or \square_P	port	17
\square_0	initial	46
\square_a	architecture design	8
\square_b	base array material	171
\square_c	continuous (infinite-dimensional)	93
\square_c	control design	8
\square_d	design	194
\square_d	discrete (finite-dimensional)	93
\square_δ	array	172
\square_e	piezoelectric actuator material	171
\square_e	equality	92
\square_f	final	46
\square_i	inner loop	48
\square_i	inequality	92
\square_{\max}	maximum	77
\square_{\min}	minimum	143
\square_o	outer loop	47
\square_O	output	116
\square_p	parameter	98
\square_p	plant or physical design	8
\square_+	additional	113
\square_{PR}	pseudo-rigid body dynamic model	172
\square_R	reduced	25

Notation	Description	Page
\square_t	time discretization	57
\square_θ	spacecraft bus	168
\square_u	control	98
\square_ξ	states	98
\square^{CC}	connected components	22
\square^{CP}	connected ports	21
\square^\dagger	candidate	48
\square^I	interconnectivity	21
$\square^{(i)}$	<i>i</i> th candidate architecture	195
$\square^{\mathcal{P}}$	problem class	195
\square'	general derivative	79
\square^{QP}	quadratic program	94
\square^*	optimal	52
\square^T	matrix transpose	16

Chapter 1

Introduction¹

“Nevertheless, the design method is as inherent to the design process as the scientific method is to scientific exploration.”

R. J. McCrary [2, p. 12]

The advancement of many engineering systems relies on novel design methodologies, design formulations, and design representations among other parts. In this work, we consider three broad design domains: *architecture*, *plant*, and *control*. These domains cover most of the potential design elements of an engineering system (and in particular, of dynamic systems, or systems whose behavior evolves through time). Allowing design flexibility (where appropriate) in all three of these domains is the first step for formulating, solving, and understanding novel and innovative design problems. Furthermore, achieving design automation in these types of problems requires adequate theory and tools to support such considerable scope and complexity.

1.1 Introductory Example

We begin with an introductory design problem that includes decisions that can be classified under the architecture, plant, and control design domains. This short example will help build some initial intuition for the types of design problems considered here.

Consider a pick-and-place task shown in Fig. 1.1 (move an object from one place to another and return) that will be performed by a robotic manipulator (a device that can perform tasks without direct human interaction) [3]. There are a number of “robots” that can perform such a task. A few of these forms or architectures are shown in Fig. 1.2. Figure 1.2a displays a two-link serial architecture. An alternative architecture could have an additional link placed in series as is shown in Fig. 1.2b. Links can be combined in a variety of ways to create new

¹Elements of this chapter are based on work completed in Ref. [1].

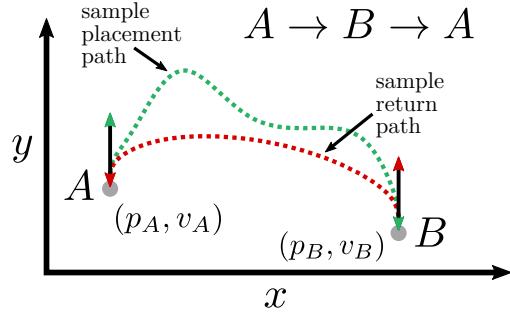


Figure 1.1: Task description in the robotic manipulator example.

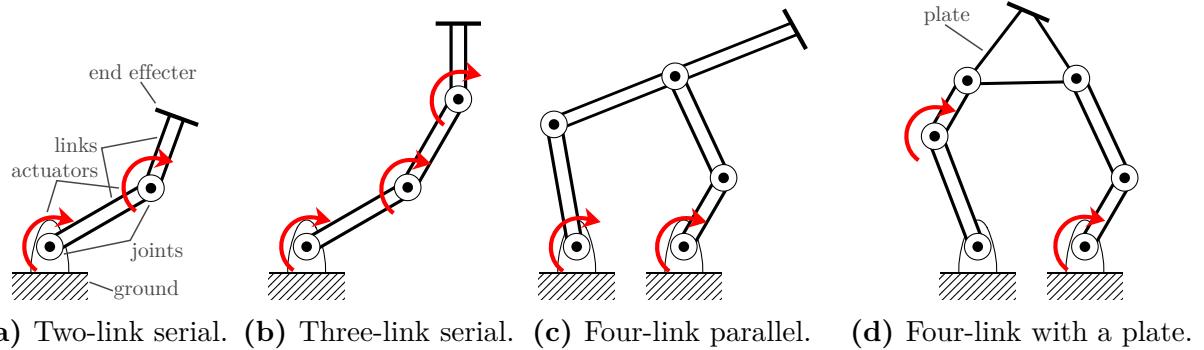


Figure 1.2: Some candidate robot manipulator architectures (red arrow indicates active joints).

architectures such as the four-link parallel manipulator in Fig. 1.2c. Additional components or building blocks can be added to the architectures such as the triangular plate in Fig. 1.2d. There are other decisions to be made with respect to the architecture including the joints that will be active (i.e., an actuator is present so a torque can be directly applied to the joint). The potential actively actuated joints are represented by red arrows in Fig. 1.2. From this discussion, the essence of the architecture design decisions is the selection of the components and their connectivity.

Some potential plant design variables for this problem are shown in Fig. 1.3. The link length, cross-section geometry, and ground spacing are all geometric plant variables. The spacing of the ground joints relative to point A in Fig. 1.1 could also be considered plant variables. All of these design variables are time-independent and related to the physical form of the manipulator. The distribution of the plant variables can vary depending on the architecture considered. If we use the two-link manipulator in Fig. 1.2a, then there will be only two length variables versus the four length variables associated with the four-link manipulator in Fig. 1.2c.

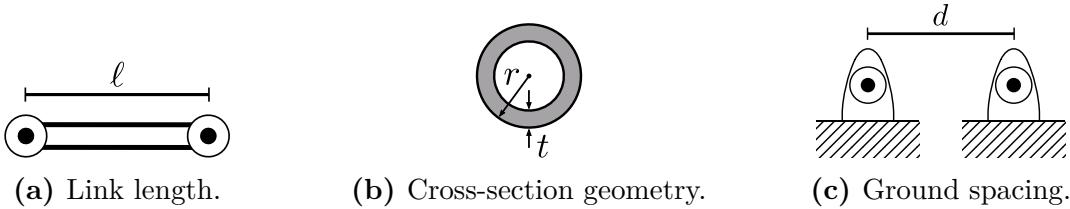


Figure 1.3: Plant variables in the robotic manipulator example.

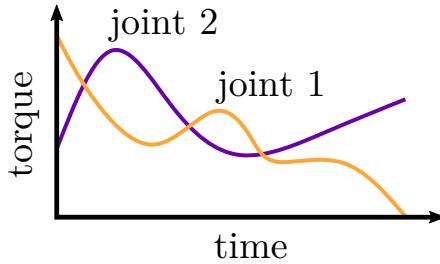


Figure 1.4: Joint control trajectories in the robotic manipulator example..

The control design could involve the specification of the torque trajectories at the active joints such that the manipulator performs the pick-and-place task shown in Fig. 1.4. These are time-varying signals that could be chosen such that the minimum amount of time or energy is used. The control design variables directly govern the behavior of the dynamic system.

It is not uncommon for each design domain to be treated separately. For example, the architecture may be assumed to be fixed while either the plant or control is developed. Recent work has considered combined plant and control design and this system-focused approach led to low energy consumption solutions [4]. Considering all three of the design domains could further lead to breakthroughs in performance and completely innovative systems. There are some potential challenges with solving this design problem including the determination of what architectures to consider, developing models for each considered architecture, and solving an appropriate optimization problem for each architecture to properly evaluate its performance.

1.2 Three Design Domains

In this section, we will discuss the classification of the three design domains. The purpose of these classifications is to help conceptualize what can we change and how can we handle these decisions in system-level design.

1.2.1 Architecture

In the engineering design context, architecture is defined as the elements contained within a system and their relationships [5]. An architecture design problem seeks to optimize some performance measure subject to the necessary constraints by determining both the distribution of the elements and their relationships in the system. A complementary use of the architecture representation is during the design process (see Sec. 1.3) where the elements and their relationships may change for reasons other than optimization of the system’s performance. The term “element” is a very generic but appropriate term as the elements in an architecture can be very different. Some dissimilar types of architecture elements include:

- Choosing between two elements with similar function but differing properties such as motor A vs. motor B from a manufacturer’s catalog
- Choosing elements with different functions such as a spring vs. a damper
- Choosing elements with different assumptions/forms such as an active vs. semi-active actuator, or feedback controller vs. open-loop controller
- Choosing elements with different levels of model fidelity such as a linear spring vs. a nonlinear, finite element-based spring

It is important to consider that not all architecture elements are meant to be strict design decisions. For example, seeking the best performing architecture by varying elements with similar function but different levels of model fidelity is not particularly useful. Therefore, a key point is that an architecture representation used during the design process does not need to be the final, realizable architecture to still be useful. There are advantages in using different levels of design representation which will be discussed more in Sec. 1.3. However, the broad definition of architecture used here encompasses the whole of an engineering system and its design representation.

Component is a common alternative term for element (and will be used frequently throughout this dissertation). Often, an architecture is made up of a particular set of components chosen from a set of discrete choices or a catalog. Components in the catalog may be homogeneous (all the same) or heterogeneous (different). The relationships between the elements of the architecture can be frequently captured by a graph [6]. These relationships, or connections, could be physics-based or general information sharing. From these definitions, we see that architecture design primarily involves the determination of discrete concepts (rather than the primarily continuous variables used in the remaining two design domains).

An alternative term for architecture is topology. Topology more commonly refers to the connections between similar elements, while architecture tends to focus on the specifics. For

example, consider a series-parallel topology for an electrical circuit. Traditionally, such a classification implies that replacing a particular component (e.g., swapping a capacitor for an inductor) in the circuit would still have the same topology. There is even a particular element type that forgoes the direct specification of the component known as the general impedance element. However, there are alternative definitions that would classify these as different topologies. Another domain where topology is frequently used is in structural optimization [7, 8]. In these types of architecture design problems, the elements are typically homogeneous (e.g., all are bars in the truss with the same type of area shape), so it does fit the definition of the connections between similar elements. Throughout this dissertation, both may still be used interchangeably even though this delineation between architecture and topology might be useful.

There are many architectures with heterogeneous components, such as electrical circuits [9–12], hybrid powertrains [13], vehicle suspensions [6], gear trains [14, 15], and biological networks [16] that are represented by colored graphs where the colors (or labels) are the component types. There are a large number of geometric architecture problems such as the design of trusses [7], heat spreaders [8], and soft robotics [17]. These problems sometimes have homogeneous elements, but a few have heterogeneous elements such as in soft robotics where the material at different locations can have a different behavior [17]. Sometimes, the architecture problem can be approximated with a continuum relaxation, such as with the solid isotropic material with penalization algorithm for structural optimization [7, 8]. However, the final solution still needs discrete values to define a realizable architecture.

1.2.2 Plant

The plant design is defined by variables that are generally regarded as time independent. Often this implies physical variables such as geometry, but may be other types such as spring stiffness (a lumped quantity). Sometimes the distinction is made when comparing between control and plant. Plant (sometimes called the artifact in this domain) are the quantities fixed during the control design. For example, the variables that can modify the dynamic equations without any control present would represent the plant.

Frequently, the variables are continuous scalars, but other types are possible. Examples of continuous scalars plant variables include the lengths of the links in a robotic manipulator [4] or the cross sectional area of truss elements [7]. The distributed thickness of a beam could be a part of the plant design (an infinite-dimensional plant variable) [18]. Discrete variables are also possible, such as the number of teeth in a gear [19].

1.2.3 Control

Control seeks to govern directly the behavior of a dynamic system, i.e., one which evolves through time. Unlike the other two design domains, control is frequently considered more adjustable for different tasks or conditions, i.e., one system may have different control schemes for different scenarios.

Generally speaking, there are two types of control paradigms: closed loop and open loop. In closed-loop control (CLC), there is some type of feedback mechanism where the current state of the system is considered when making decisions about the next action to take. In open-loop control (OLC), there is no feedback mechanism. With no feedback mechanism, open-loop control design variables are typically entire trajectories. An example of an OLC variable is the torque trajectory for a rotatory actuator in a robotic manipulator [4]. For closed-loop control, gains or other variables that modify aspects of the feedback mechanism could be considered the control design. An example of CLC is in the infinite-horizon linear-quadratic regulator with full state feedback [20].

1.2.4 Ambiguity in These Delineations

While descriptions were provided for each of the three design domains, there is still some ambiguity on how to provide the appropriate classifications. There are a number of potential reasons for this vagueness.

One of the primary reasons may simply be legacy design paradigms that treat certain parts of the design as separate [21]. System engineers may tackle the architecture design while the mechanical and control engineers deal with the plant and control designs, respectively (and what is in their domain is decided by some form of management). Therefore, the classification of the design variables may be along the lines of what is traditionally expected for the particular type of engineer to handle during the design process. In a system-level design approach, as is preferred in this work, this type of classification is not needed as all domains are considered uniformly.

Another type of delineation is based on variable types. Architecture design variables are discrete. The plant consists of continuous, time-independent variables and the control is infinite-dimensional trajectories. This classification scheme has the advantage of utilizing techniques specifically created to handle each of the variable types. However, the discussion above for each design domain demonstrated that the domains contain variables of variety types. In a similar way, the general theory or tools may be developed to handle certain parts

of the design domains which may provide their own classifications (e.g., multidisciplinary design frameworks [21]).

Another difficulty comes when the approximations or better representations of design variables in one domain are frequently classified in another domain. Consider the following examples where a design decision in one domain is approximated in another design domain:

- Architecture → plant: the solid isotropic material with penalization approximation in structural optimization [7, 8]
- Architecture → control: assuming a particular feedback controller form for the power take-off in wave energy conversion so that standard control-design techniques can be applied rather than selecting a specific power take-off architecture [22]
- Plant → architecture: using only preferred discrete component values in a circuit (e.g., E12 series) rather than continuous values for the inductance and capacitance [23]
- Plant → control: abstracting the power take-off system in a wave energy converter to an open-loop force trajectory rather than a physical-system with plant variables [24, 25]
- Control → plant: tuning a passive system with only physical components such as dampers and springs instead of a full-state feedback controller

In many of these cases, the designers may not put much importance in their classification but rather treat them as the necessary design variables for the problem at hand.

A final point is the concept of plant and control architectures. For new systems, it may be beneficial to split up the architecture decisions that are traditionally linked to the plant and control systems [1]. For example, candidate plant architectures for automotive hybrid powertrains include series, parallel, and power-split, among others [26]. For the power-split architecture (and other similar new architectures), the splitting of the power between the energy producing components is a necessary control decision [26]. The specific controller used to make this control decision is the control architecture and may be an open-loop trajectory, a basic feedback controller, or any number of alternatives depending on the stage in the design process or its performance.

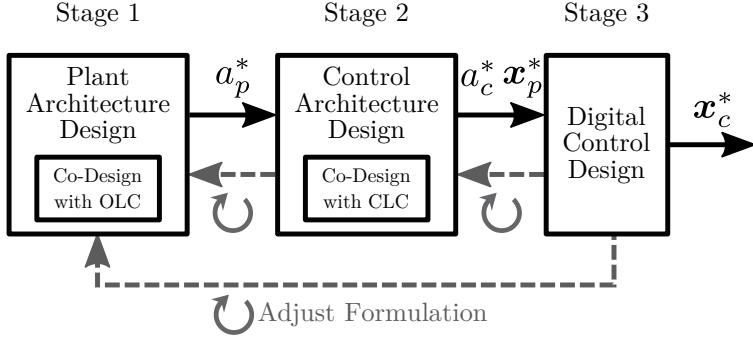


Figure 1.5: Proposed stages for complete dynamic system design.

1.3 A Design Process for Complete Dynamic System Design

Allowing flexibility in all design domains can create a number of challenges that can limit how effectively one arrives at a suitable solution. Here we adopt the design process proposed in Ref. [1] for complete dynamic system design that helps manage the complexity and uncertainty found in combined architecture, plant, and control design problems. It is a three stage process shown in Fig. 1.5. We denote the plant architecture as a_p and associated plant variables as \mathbf{x}_p . The control architecture is represented with a_c and control variables \mathbf{x}_c (and OLC variables as \mathbf{u}). The system-level objective function is represented by Ψ .

The first stage seeks to determine the optimal plant architecture by forgoing the specification of the control architecture. Using a specific control architecture limits creative plant design exploration at early design stages [1]. Instead, OLC is used for the control among other potential uses. OLC can replace some components or interfaces with optimal trajectories, reducing the design problem size while still providing an optimal solution [1, 25, 27]. In addition, OLC-based studies are particularly valuable at early design stages for gaining insights into upper system performance limits and dynamic behaviors and interactions that lead to system-optimal performance [4, 21, 28–31]. Furthermore, combined plant and control design, or co-design, will be used to determine the performance for each candidate plant architecture because it is a system-level optimization strategy [21, 32, 33] that will allow fair comparisons between candidate plant architectures [26].

The specifics for stage 1 are shown in Fig. 1.6. The inputs are the information about the system’s operating environment, objectives, constraints, and a catalog of candidate components. A nested optimization approach is used to handle the different variable types. Here the outer loop will change the discrete (plant) architecture design variables while the inner loop determines the optimal performance by solving the continuous co-design problem for

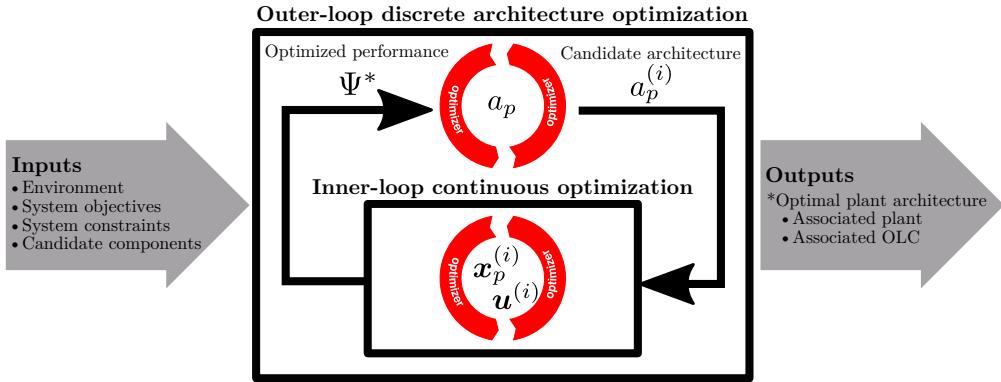


Figure 1.6: Stage 1 details.

the candidate architecture.

Stage 2 in Fig. 1.5 seeks to determine the best control architecture given the optimized plant architecture from stage 1. Similar to stage 1, a nested optimization approach can be used to vary the discrete control architecture decisions while the appropriate co-design problem is solved to determine the optimal performance for the candidate architecture. For example, we could consider a basic feedback, hybrid [34], or model predictive [35] controller architectures. The final stage, stage 3, seeks to determine the digital controller design given the plant and controller architectures from the previous stages [36].

It might be necessary to iterate between the stages if unforeseen issues appear. Information about these issues can be passed back to the previous stage and the problem formulation can be adjusted to address overlooked elements. Additionally, in each of the stages, a sequence of problems may be posed and solved, each one informed by the results of the previous problem, moving toward greater levels of system specificity.

The theory and studies in this dissertation will primarily focus on design problems in stage 1, but it is important to understand the motivations behind these studies and how they fit into the entire design process.

1.4 Solution Generation Challenges

There are a number of challenges associated with the design freedom found in combined architecture, plant, and control problems. The complexity of individual problems can be quite significant, sometimes even rendering the problem intractable, unless this complexity is handled appropriately. Efficiently automating the key tasks in the solution generation process is essential to arriving at desirable solutions in a practical manner. Here we highlight

three important generation tasks: candidate architecture generation, model generation, and optimization problem generation.

1.4.1 (Automated) Candidate Architecture Generation

Exploring different architectures requires an appropriate conceptual framework that allows for modifications to the appropriate elements in the architecture. A straightforward, commonly used representation is an adjacency matrix where the nonzero entries in the matrix represent connections between elements in the architecture. Candidate architectures could be a different set of nonzero values in the adjacency matrix. However, not all architecture representations are equally useful. Some might produce many infeasible systems or too many candidates. Others might produce many architectures with poor performance, or are not amendable to some optimization procedure. An example alternative representation framework was developed for electrical circuits where a sequence of low-level instructions are used to generate a circuit [37]. These instructions iteratively add new elements to the circuit in topologically different ways. A manual alternative would include experts proposing candidate architectures, which may be appropriate in some cases, but often may not support comprehensive design space exploration. In many traditional engineering design problems, the architecture is fixed so this abstract representation is not typically needed.

1.4.2 (Automated) Model Generation

Given some architecture specification (e.g., a graph), we need to create a suitable model (some representation of the architecture that can predict performance and identify if constraints are satisfied) for use in the optimization problems described in Sec. 1.3. Certain modeling methodologies support this task such as bond graph modeling [38] or block diagram-based modeling [39]. Other techniques have been developed for specific types of problems such as solid isotropic material with penalization for structural optimization [7] and modified nodal analysis (MNA) for electrical circuits [40]. Nevertheless, for many design problems, some investment will be needed to generate models efficiently and in an automated manner. In traditional engineering design problems, the model is fixed for the given architecture, but can vary based on the plant/control design variables.

1.4.3 (Automated) Optimization Problem Generation

For different candidate architectures of the same design problem, the optimization problem that predicts its performance may vary. If the model is different, there may be different plant and control variables. Different constraints may be present depending on the components in the architecture (e.g., we only need control actuator bounds if the component is present). Since all problem elements (e.g., number and types of variables, objective function form, constraints, model) could change between architectures, forming and solving the optimization problem automatically is important for solution efficiency. Understanding the general optimization problem structure for any candidate architecture is key so that it can be leveraged to find solutions faster and more robustly. For some types of optimization problems, such as ones with infinite-dimensional constraints and variables, additional work is needed to obtain (approximate) solutions. Frequently, only a single optimization problem form needs to be posed and solved in traditional engineering design problems.

1.5 Dissertation Overview

This introduction has provided a discussion on the combined architecture, plant, and control design problems including how this problem type can be used in the dynamic system design process and some potential challenges. Due to the diverse design domains considered in this dissertation, a variety of methodologies are presented to manage different elements in the design process. Chapters 2–5 focus on the development of these methodologies while Chapters 6–8 present detailed engineering design case studies that utilize the concepts. A focus in the initial chapters is on the generality of the proposed approaches. The frameworks presented are developed with specific consideration of what types of problems will fit under the proposed framework.

- Chapter 2 focuses on the task of representing and generating candidate architectures. The theory and algorithms developed in this chapter are applicable to architecture problems where the architecture is representable by a colored graph built from a catalog of components. A complete listing of all potential architectures under specific assumptions can be generated with this approach. A number of enhancements to the algorithms presented in this chapter are detailed in Appendix A.
- Chapter 3 focuses on a methodology for handling the other two design domains: combined plant and control design or co-design. The general co-design problem formulation

and optimality conditions are explained for both the simultaneous and nested solution strategies. Due to a number of challenges associated with the optimality conditions, practical solution considerations are discussed with a focus on the motivating reasons for using direct transcription in co-design.

- Chapter 4 concentrates on scaling in dynamic optimization (a general class of problems found in Chapter 3). The necessary theory for scaling dynamic optimization formulations is presented and a number of motivating examples are shown. Scaling can be used to help facilitate finding accurate, generalizable, and intuitive information. The unique structure of dynamic optimization suggests that scaling can be utilized in novel ways to provide better analysis and formulations more favorable for efficiently generating the solution. A simpler, scaled problem is used to understand the general trends found in the results of the case study in Chapter 7, along with minor roles in the other two case studies.
- Chapter 5 presents the direct transcription method for finding approximate solutions to dynamic optimization problems. The method discretizes (in time) the infinite-dimensional quantities and creates a mathematical program that can be solved. This method is an enabling method that allows solutions to be obtained for general co-design problems in Chapter 4. A bulk of the chapter is focused on solving a particular subclass: linear-quadratic dynamic optimization problems. These problems can be approximated with quadratic programs and be efficiently constructed and solved. Problems with this structure are particularly important when the three domains are considered because many control problems may need to be solved, and an inefficient method could hinder the ability to explore the solutions. The algorithms, sparsity patterns, and example codes are in Appendix C. The case studies in Chapter 7 and 8 utilize this theory and codes to solve the control subproblem.
- Chapter 6 presents an enumeration-based synthesis methodology for passive analog circuits by generating and evaluating all appropriate circuits. Two design problems are considered: frequency response matching and low-pass filter realizability. The circuit graphs (architectures) are generated using the methods described in Chapter 2. This problem contains both architecture and plant design decisions.
- Chapter 7 focuses on the design of strain-actuated solar arrays for spacecraft precision pointing and jitter reduction. This problem has a variety of challenging plant and control design variables. The nested co-design strategy described in Chapter 3 is

utilized, and the control subproblem is solved with the methods in Chapter 5. The primary example from Chapter 4 provides a number of insights into the results found in this case study.

- Chapter 8 is a case study that combines all three design domains with the design of vehicle suspensions. The work from Chapters 2–5 all contribute in addressing this complex design problem. This chapter also includes a discussion of a general class of architecture, plant, and control design problems that utilize linear physical elements.
- Chapter 9 concludes the dissertation with an overall summary of the key points and identifies future research directions.

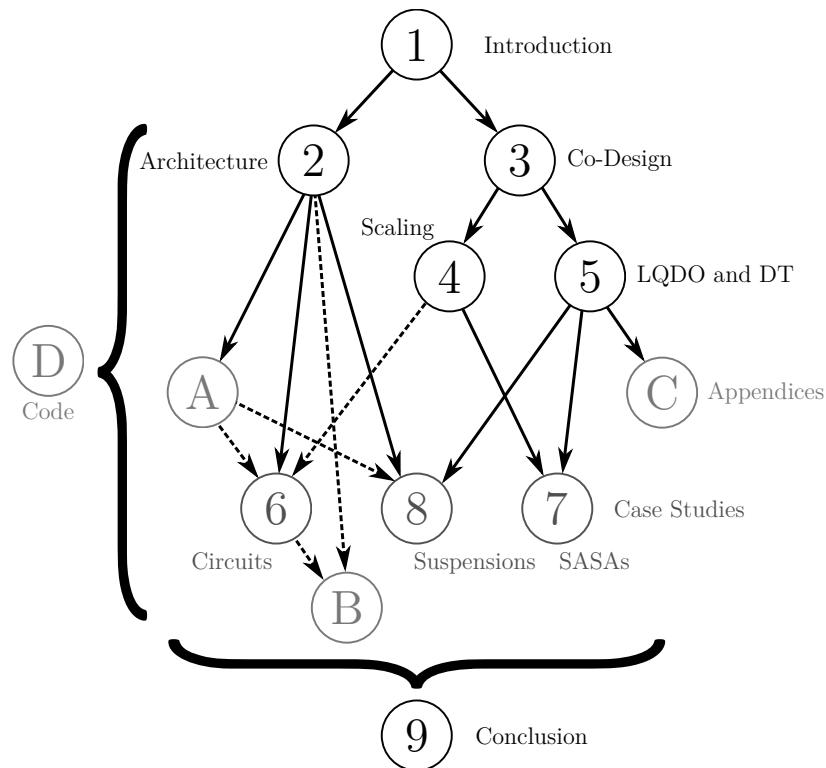


Figure 1.7: Dissertation content connections (dashed line indicates a narrower connection).

Chapter 2

Candidate Architectures through Enumeration²

“Central to design is the creative act. This is not to imply that all needs are met creatively. Some are met by found solutions, found in handbooks, catalogues, department stores, etc. However, if a need is met through design, then creativity is involved.”

R. A. Willem [42]

System architecture was defined as the elements or components contained within a system and their relationships in Sec. 1.2.1 [5, 43, 44]. Designing breakthrough engineering systems with new capabilities and new levels of performance requires innovations in system architecture [45, 46]. Engineers often rely on heuristics such as design by analogy [47] and intuition when considering system architecture, but this may result in fixation on example designs and stifle innovation [48].

Consider the following architecture design problem formulation:

$$\min_{\boldsymbol{x}_a} \Psi(\boldsymbol{x}_a) \quad (2.1a)$$

$$\text{subject to: } f_a(\boldsymbol{x}_a) = a \in \mathcal{F}_a \quad (2.1b)$$

where \boldsymbol{x}_a represents architecture design variables, $f_a(\boldsymbol{x}_a)$ is a mapping between the architecture design variables and the architecture a , and Ψ is a performance index. A fair comparison between architecture candidates in the set of feasible architectures \mathcal{F}_a requires knowledge of the best possible performance for each candidate architecture. This requires optimization with respect to *inner-loop* design variables such as plant and control design (see Chapter 3) [1, 32]. The number design variables, constraints, models, etc. can all change depending on the candidate architecture in the inner loop. Hence, formulating and solving architecture design problems can be challenging. Here we focus on a method that generates candidate architectures that may be used to determine the performance index with respect to an architecture-dependent inner-loop design problem.

²Elements of this chapter are based on work completed in Ref. [6, 41].

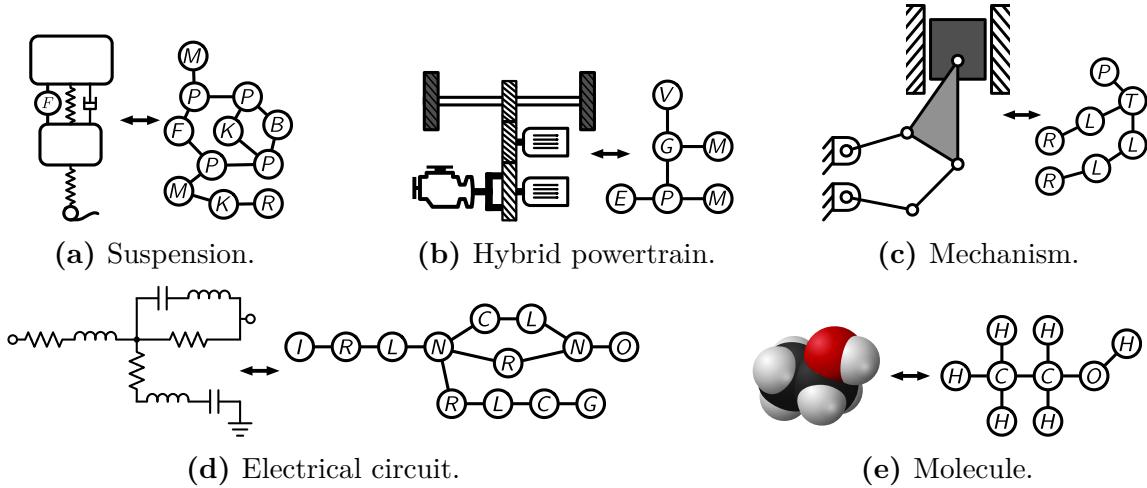


Figure 2.1: Architectures represented as graphs.

Many studies have concentrated on effective representation and generation methods, primarily based on graph representations of the system architecture (see Fig. 2.1 for some common engineering systems represented as graphs). There is a wide range of architecture design representations typically classified along the spectrum of continuum [49, 50] vs. discrete [51, 52] design domains and homogeneous [49, 50] vs. heterogeneous [51, 53] elements. The value of these methods often is to present new valid topologies to engineers for further evaluation (subjective or quantitative), helping to overcome design fixation. A popular class of methods for generating architecture candidates is generative representations [46, 49, 52–57]. This class covers a range of candidate architectures in an implicit form based on repeated application of rules that modify the graph. It has been recognized that generative approaches generate topologically simple designs, not covering the entire design space [51]. Furthermore, the design space is sensitive to design knowledge [47, 56] and rules [49, 57]. While these designs may satisfy functional requirements elegantly, generation of more elaborate architectures is needed in some cases.

It can be challenging to describe the design space of an architecture generation method, partially due to the combinatorial nature of architecture design problems. A better understanding of how certain rules restrict the design space can lead to better generative approaches, but this requires a complete design space to compare against. Furthermore, the ultimate goal is a set of all architectures that are feasible with respect to constraints [58] and that are unique [54]. Arriving at such a design space efficiently is a considerable challenge.

In this chapter, the design space is completely captured by a perfect matchings approach for a certain class of architecture design problems, more specifically, problems that are rep-

resented by undirected colored graphs under the component/port paradigm [43, 51, 59]. The proposed approach generates truly novel architectures (in fact all of them) but still leverages some of the natural constraints found in architecture design problems to reduce the number of graphs generated. This design space is found through an enumeration, i.e., a complete, ordered listing of all the items in the collection of feasible and unique architectures. This approach leads to a number of interesting insights into the fundamental nature of architecture design problems.

The remainder of the chapter is organized as follows. The next section outlines the some of the basic theory behind candidate architectures with perfect matchings. Network structure constraints and the colored graph isomorphism problem are then discussed, with the objective of achieving feasible unique architectures. Using the insights from the previous two sections, a tree search algorithm is developed that more efficiently covers the same design space. A number of case studies are then presented. Finally, a discussion is given of the results and how the proposed approaches can be used in current architecture design research.

2.1 Candidate Architectures with Perfect Matchings

First, some relevant graph theory background is given.

Definition 1 (Graph). *A graph is a pair $G = (V, E)$ of sets satisfying $E \subset [V]^2$ where the elements of V are the vertices and the elements of E are its edges.*

A *simple graph* is an unweighted, undirected graph containing no graph loops or multiple edges. The adjacency matrix of G is the $n \times n$ matrix $A = A(G)$ whose entries a_{ij} are given by:

$$a_{ij} = \begin{cases} 1 & \text{if the set } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

For a simple graph, the *adjacency matrix* must have 0s on the diagonal. For an undirected graph, the adjacency matrix is symmetric and if only a subset of the edges are present in E , the correct $A(G)$ can be constructed with $\text{sign}(A + A^T)$. The *connectivity matrix* of simple graph G can be found with:

$$A_C(n) = A^n \tag{2.2}$$

where the interpretation of $A_C(n)$ is for every nonzero entry, there exists at most n undirected

walks required to go from v_i to v_j , i.e., the pair of vertices are connected in some sense [60, p. 165]. We will assume that n is the same as the length of A giving all walks.

The degree of a vertex is the number of edges incident at the vertex and the average degree is $d(G) = 2|E|/|V|$ [61, p. 5]. A *matching* in a graph is a set of edges such that no two have a vertex in common and a perfect matching is a matching that covers every vertex [62, p. 255].

Definition 2 (Colored Graph). *A colored graph G is a three-tuple (V, E, L) where (V, E) specifies an undirected graph and $L = \{V_i\}_{i=1}^k$ is a partition of the vertices into color sets ($V_i \cap V_j = \emptyset$, $i \neq j$, and $(\cup_{i=1}^k V_i) = V$). For convenience, define $\text{color}(v) = i$ if $v \in V_i$.*

The graphs in Fig. 2.1 are colored graphs where each vertex represents a component. The colored labels indicate different component types. For example, in Fig. 2.1a, (K) represents a vertex with a coloring K indicating that it is a spring component type and that (B) represents a damper component type. These are termed *2-port* components since they can have up to 2 unique edges if ports are connected to a single additional vertex (this port notion is analogous to bond graph modeling [63]). However, there are fundamental limitations with this representation. For example, if the order that the ports of a component are connected to edges prescribed in A is important, then pure component graph representation is not sufficient for determining a unique architecture. Consider the planetary gear (P) in Fig. 2.1b. Since the planetary gear is represented by a single vertex, it is unclear which of the connected components $\{E, G, M\}$ is connected to the sun, ring, and carrier (common names for the planetary gear ports [13]). Permutations of this decision would result in different architectures but the same adjacency matrix. Another limitation is the unspecified nature of parallel connections when only component-level vertices are used. If the adjacency matrix specifies that K has three connections, it cannot be necessarily uniquely determined what connections are incident at each specific port of K . A better representation would determine unique graphs, motivating a pure ports graph representation of architectures.

2.1.1 Ports Graph

A port graph G^P is constructed from a three-tuple (C, R, P) :

- C is the colored label set representing distinct component types, whose size is denoted by n_C
- R is a column vector indicating the number of replicates for each component type
- P is a column vector indicating the number of ports for each component type

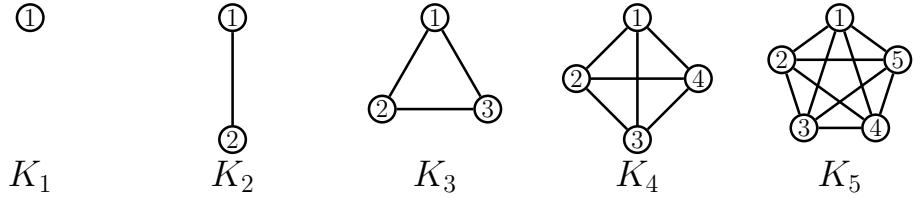


Figure 2.2: Complete graphs on n vertices between 1 and 5.

Using (C, R, P) we will create the three-tuple (V, E, L) that defines a proper colored graph (see Definition 2). The definition of an n -port component in this context is all n ports are completely connected to each other. Therefore each component can be considered a complete graph of its ports (see Fig. 2.2 for the first five complete graphs). The vertex and edge set for G^P is then defined as the union of these complete subgraphs:

$$(V, E)^P = \bigcup_{k=1}^{n_C} \bigcup_{j=1}^{R_k} K_{P_k} \quad (2.3)$$

where K_{P_k} is a complete graph of size P_k .

The complete label for each vertex is constructed from a naming scheme where the base is the colored label from C , the subscript is the replicate number, and the superscript is the port number. Then the set of colored labels for G^P can be constructed as:

$$L^P = \bigcup_{k=1}^{n_C} \bigcup_{j=1}^{R_k} \bigcup_{i=1}^{P_k} \{(C_k)_j^i\} \quad (2.4)$$

where each label is unique at this point.

There are a number of graph measures and metrics that can be computed for this class of graphs. First, the number of vertices is given by: $N_P = |V^P| = P^T R$, where $|\square|$ is the cardinality of a set. The number of edges in K_n is $n(n - 1)/2$ [60, p. 22], so we can easily calculate the number of edges in G^P as:

$$|E^P| = \frac{1}{2} (P \circ (P - 1))^T R \quad (2.5)$$

where \circ denotes the Hadamard product. The total number of components is: $N_C = e^T R$ where e is a column vector of ones of appropriate length.

2.1.2 Interconnectivity Graph

The essence of an architecture design problem is determining the relationships between ports. Therefore a natural question is: what are all the possible architectures? Subgraph enumeration provides a relevant framework for determining all possible graphs satisfying specified properties [62]. We now say that a candidate architecture in an architecture design space described by (C, R, P) has the following properties:

1. A set of components bounded by (C, R, P) (so $n_P \leq N_P$, where n_P is the total number of ports for the candidate architecture).
2. Each port in $(V^P, \{ \}, L^P)$, i.e., G^P without edges, is connected to another port (this implies n_P is even and is also known as a complete topology since there are no open ports [59]).

Now, a complete architecture design space described by (C, R, P) would be a set of candidate architectures that contain all possible valid subsets of components and all possible valid edge combinations between said subset of components. We will denote this graph structure space (a set of all graphs that fulfill a certain set of conditions) as \mathcal{G}_1 . We will now see that the enumeration based on perfect matchings (PMs) will correspond to the complete architecture design space.

Recall that a PM is a matching in which every vertex of the graph is incident to exactly one edge [62]; all PMs for K_2 , K_4 , and K_6 are shown in Fig. 2.3. Since a necessary condition for a PM is an even number of vertices, we will assume N_P is even (Sec. 2.2.1.1 discusses the implications of this restriction). The number of PMs for K_n can be calculated using the double factorial function:

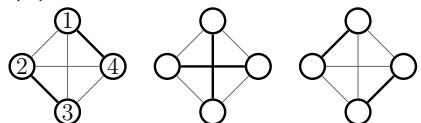
$$\mathcal{D}(n) = (n - 1)!! = (n - 1) \times (n - 3) \times \cdots \times 3 \times 1 \quad n \text{ even} \quad (2.6)$$

and the first several values of this function are $\mathcal{D}(2) = 1$, $\mathcal{D}(4) = 3$, $\mathcal{D}(6) = 15$, $\mathcal{D}(8) = 105$, $\mathcal{D}(10) = 945$, $\mathcal{D}(12) = 10,395$, $\mathcal{D}(14) = 135,135$, $\mathcal{D}(16) = 2,027,025$, $\mathcal{D}(18) = 34,459,425$ [64]. This function grows slower than the traditional factorial function since the even elements have been omitted. This result agrees quite well with the bound by Mittal and Frayman for a similar problem (the bound being on order of $\sqrt{N_P!}$) [43].

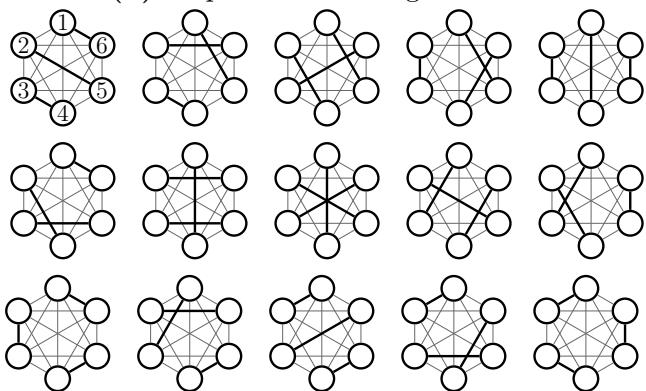
For n between 1 and $\mathcal{D}(N_P)$, $\mathcal{P}(n, N_P)$ denotes the edge set for the n th PM of K_{N_P} . The uniqueness of each PM can be ensured by ordering all edges with the first element being the larger vertex (in the sense of the index value). For example for the first graph in Fig. 2.3b, $\mathcal{P}(1, 4) = \{(4, 1), (3, 2)\} \neq \{(1, 4), (2, 3)\}$. It will be convenient to map the edge set to a vector where sequential pairs are a single edge (e.g., $\{(4, 1), (3, 2)\} \rightarrow [4 1 3 2]$). There are two interesting properties of the set of all PMs [62]:



(a) $1!!$ perfect matchings for K_2 .



(b) $3!!$ perfect matchings for K_4 .



(c) $5!!$ perfect matchings for K_6 .

Figure 2.3: Perfect matchings for K_2 , K_4 , and K_6 .

1. Enumerating all PMs of K_N results in a set of graphs where all possible edge combinations are present where each port is connected to exactly one other port.
2. The set of PMs graphs of K_N , contains all edge sets for K_{N-2} , where $N \geq 4$.

Now with these two properties in mind, consider any candidate architecture defined by the properties above. It has $n_P \leq N_P$ and n_P is even, so there exists a PM matching edge set in the set of PM graphs of K_{N_P} that contains the subset of components and the particular edge combination since all edge combinations of n_P are found in the set N_P . Therefore, because any candidate architecture can be found in the set of PM graphs of K_{N_P} , a PM approach captures the complete architecture design space.

A PM approach is a type of graph numerical representation scheme (GNRS) since there is a binary relation between \mathcal{G}_1 and $n \in [1, \mathcal{D}(N_p)]$. A PM approach is left-total and left-unique with respect to complete topologies of (C, R, P) (left implies a map from \mathcal{G}_1 to n and these are desired properties for a GNRS) [44]. Algorithm 2.1 is useful for this direction as it determines $\mathcal{P}(n, N_P)$ [62, 65]³. In addition, a PM approach is right-total and right-unique with respect to the same conditions (right implies a map from n to \mathcal{G}_1). Algorithm 2.2 is useful for the right direction as it determines $\mathcal{P}^{-1}(E)$ where E is a valid PM edge set. Based on these two efficient algorithms, a PM approach is *algorithmic* in both directions [44].

The interconnectivity graph G^I then is defined with V^P , L^P , and an edge set from the set of PMs:

$$E^I = \mathcal{P}(n, N_P) \quad \text{where } n \in [1, 2, \dots, \mathcal{D}(N_P)] \quad (2.7)$$

The number of edges is: $|E^I| = N_P/2$.

2.1.3 Connected Ports/Components Graph

The connected ports graph is the union of the ports graph and interconnectivity graph:

$$G^{CP} = G^P \cup G^I \quad (2.8)$$

The number of vertices is still N_p . There is possibility of multiple edges when combining the graphs since edges between the already connected ports of a component may be connected with a PM. We can simplify G^{CP} by combining all multiple edges into a single equivalent edge, thus creating a simple graph. Using this operation, the number of edges of G^{CP} can

³Ref. [65] contains MATLAB codes for Alg. 2.1 and more efficient recursive algorithm when all perfect matchings are required.

Algorithm 2.1: Creation of edge set for a specific perfect matching number.

Input : N – number of vertices (should be even)
 l – perfect matching number, integer between 1 and $(N - 1)!!$

Output: E – vector of edges in sequential pairs

```

1
2  $J \leftarrow [1, 3, 5, \dots, N - 1]$                                 /* odd numbers from 1 to  $N-1$  */
3  $P \leftarrow [1, \text{cumprod}(J)]$                                 /* cumulative double factorial */
4  $V \leftarrow [1, 2, \dots, N]$                                          /* create initial list of available vertices */
5 for  $j \leftarrow J$  do
6    $q \leftarrow (N + 1 - j)/2$                                          /* index for 2nd to last entry in P */
7    $i \leftarrow \text{ceil}(l/P(q))$                                        /* calculate smaller vertex index */
8    $E(j) \leftarrow v(\text{end})$                                          /* assign largest remaining value */
9   remove element  $V(\text{end})$                                          /* remove largest remaining value */
10   $E(j + 1) \leftarrow V(i)$                                          /* assign smaller selected value */
11  remove element  $V(i)$                                          /* remove the smaller selected value */
12   $i \leftarrow i - ((i - 1) \times P(q))$                                 /* get index in subgraph with 2 vertices removed */
13 end

```

be bounded by:

$$|E^P| \leq |E^{CP}| \leq |E^P| + |E^I| \quad (2.9)$$

since each edge of E^I could be a repeat of an edge in E^P . G^{CP} is a unique representation of an architecture since a PM is between specific ports and all components are fully connected subgraphs.

There are advantages of the component graph representation that should be utilized, such as a reduced number of vertices and edges (and not differentiating replicates). We now characterize *simple components* whose port ordering does not matter (e.g., a 2-port spring) and *structured components* where it does (e.g., a 3-port planetary gear). All simple components will be reduced to a single vertex and the appropriate edges will be created. The labels for simple components will be modified by removing both the superscript and subscript of L^{CP} . Structured components will only have their subscripts removed to maintain port discernibility. This graph representation is termed the connected component graph G^{CC} .

To get a better sense of the structure of G^{CC} consider all components to be simple components. Then the number of vertices is simply N_C . The number of edges of G^{CC} can be bounded by $0 \leq |E^{CC}| \leq |E^I|$ and the labels are:

$$L^{CC} = \bigcup_{k=1}^{n_C} \bigcup_{j=1}^{R_k} \{C_k\} \quad (2.10)$$

All graphs in the remaining sections are considered to be G^{CC} graphs unless otherwise noted.

Algorithm 2.2: Determination of the perfect matching number for a specific edge set.

Input : E – vector of edges in sequential pairs, should be properly ordered and even length

Output: l – perfect matching number, integer between 1 and $(N - 1)!!$

```

1  $N \leftarrow \text{length}(E)$                                      /* total number of vertices */
2  $P \leftarrow \text{reverse elements of cumprod}([1, 3, 5, \dots N - 3])$     /* array flip, cumulative double factorial */
3  $V \leftarrow [1, 2, \dots, N]$                                      /* create initial list of available vertices */
4  $l \leftarrow 1$                                                  /* initialize perfect matching index */
5 for  $j \leftarrow 1$  to  $N/2 - 1$  do
6    $q \leftarrow E(2j - 1)$                                          /* index of largest remaining vertex */
7    $V(q) \leftarrow 0$                                               /* zero entry, removing it */
8    $i \leftarrow \text{find index of } E(2j) \text{ in the vector of nonzero elements of } V$ 
9    $V(E(2j)) \leftarrow 0$                                          /* zero entry, removing it */
10   $l \leftarrow l + ((i - 1) \times P(j))$                            /* sum to get build up the index */
11 end

```

2.2 Candidate Graphs to Unique Useful Graphs

In the previous section an approach for enumerating architectures based on the consideration of every potential PM was outlined. However, there are a number of deficiencies in this set of architectures, including infeasible graphs based on practical constraints for a specific architecture design problem, and repeated graphs in the modeling sense. The following two subsections address architecture feasibility and uniqueness, respectively.

2.2.1 Network Structure Constraints

We define feasibility as a candidate architecture's satisfaction of all network structure constraints (NSCs) for a particular architecture design problem. Similar to the rules in generative design approaches, the creation of the set of network structure constraints for a particular architecture design problem is subject to the creativity and intuition of the designer. Some of these constraints may be fairly self-evident while others might be vague or contentious (Ref. [58] discusses these issues). Wyatt et al. describe four types of NSCs that are sufficient to define almost all aspects of realizability of an architecture and are summarized briefly (without edge coloring considerations) as [58]:

- Component number constraints (CNCs) prescribe how many components of a given type must be present
- Direct connection constraints (DCCs) prescribe which component types may be connected together by which connection types and cardinality of the connections

- Fan out constraints (FOCs) prescribe how many connections that components of a certain type must have in total
- Indirect connection constraints (ICCs) prescribe how many continuous paths there must be from every component of one type to every component of another type

Graph generating algorithms designed to always satisfy certain NSCs could be more useful since all of the generated graphs would be feasible with respect to those certain NSCs. Furthermore, graph generating algorithms can also be designed to produce graphs that have a higher proportion that satisfies certain NSCs than more naive approaches. Some of the NCSs that are not satisfied can be with edits to the graph. The only operation that will be considered here is the removal of vertices G^{CC} and the corresponding edges and labels as it will maintain certain properties of the graph structure space \mathcal{G}_1 . Other operations such as vertex insertion, edge insertion, or label substitution destroy the analysis of the design space coverage that is possible with an enumerative PM approach. Next, several common NSCs (denoted with S) are described along with the specifics of checking their satisfaction with available graph analysis tools.

S_1 Every graph must be a connected graph (ICC). A graph is termed connected if there is a path from any vertex to any other vertex in the graph [61, p. 18]. This can be checked with the connectivity matrix, $A_C(G)$, in Eqn. (2.2). If all entries in this matrix are not 1, then the graph is not connected.

S_2 Every graph can only have a maximum number of a given component type (CNC). This is defined by R in the architecture definition three-tuple so is naturally handled by a PM approach. An example: ‘Every suspension must have less than 3 springs’.

S_3 Every graph must have a specific number of certain component types (CNC). These mandatory components will be captured with a vector M of length n_C . The elements of M are binary with a 1 indicating all replicates of the component type must be present in the graph. An example: ‘Every hybrid powertrain must have an engine and a vehicle’.

S_4 Every graph must have specific component types connected to each other (ICC). This can be checked with the connectivity matrix in Eqn. (2.2). If nonzeros are not present at every location where a path must exist between component types, then the graph is infeasible. If we require S_1 and S_3 , then we can leverage the vector M in S_3 to satisfy both constraints by checking $A_C(G)$ such that all mandatory components are

connected to each other. An example: ‘Every hybrid powertrain must have an engine connected to a vehicle’.

- S_5 Every graph must have vertices whose number of unique edges is within a specific range (FOC). The values in P can define the upper bound for each vertex since components are defined by a certain number of ports. For even port numbered component types the lower bound is 0 and 1 for odd. This can be checked summing row-wise (or column-wise) the symmetric adjacency matrix $A(G)$ and comparing these sums to the appropriate index in P . This type of NSC is sometimes termed a degree-constrained subgraph problem [66, p. 217]. A PM approach naturally satisfies this constraint. An example: ‘Every spring must have between 0 and 2 unique edges’.
- S_6 Every graph must have vertices with a specified number of unique connections (FOC). This is a stronger form of S_5 where both the upper and lower bound can be determined by P and is sometimes termed a factors problem [66, p. 218]. An example: ‘Every spring must have exactly 2 unique edges’.
- S_7 Every graph must have edges between vertices that are feasible (DCC). We can specify that certain component types cannot be connected to other component types with a reduced potential adjacency matrix A_R . This $n_C \times n_C$ binary matrix will have 1 entries indicating a connection is feasible and 0 entries for infeasible. This constraint can be checked by verifying that each 1 in $A(G)$ has a corresponding 1 in the potential adjacency matrix. No self-loops in a specific component type can be enforced with a 0 at the appropriate location on the diagonal of A_R . A PM approach does not satisfy this constraint as all connections between ports are considered feasible. An example: ‘Every translational spring cannot be connected to any rotational damper’.

The ordering of the constraints matters if vertices are to be removed to satisfy certain constraints. The following procedure is assumed:

0. S_2 and S_5 naturally satisfied with a PM approach
1. Check S_3 and S_4 simultaneously using M since they can be checked without needing to remove the removable components
2. Remove components that don’t satisfy S_3 and S_4 ; thus satisfying S_1
3. Check S_6
4. Check S_7
5. Check any other constraints

The specific steps are only performed if the constraint is present in a specific architecture design problem. The ordering of S_6 , S_7 , and the other previously undefined constraints could be performed in an alternative order as long as they are checked after removable components are removed. This is so that a candidate architecture is not declared infeasible if only removed components and their connections violate the constraints.

The graph structure space defined as graphs that satisfy the present NSCs and (C, R, P) is denoted $\mathcal{G}_2 \subseteq \mathcal{G}_1$. The NSCs $\{S_1, S_3, S_4\}$ are assumed to be all present or none present to simplify the discussion as many common architecture design problems require all three. With the NSCs outlined, a *useful graph* is defined as one that is feasible with respect to the NSCs.

2.2.1.1 Comparison to Another Method

At this point, it is imperative to compare the PM approach to another graph numerical representation scheme that can be used for enumeration: indexed stacked blocks (ISBs) [44]. This scheme is far more general than the proposed PM approach as it allows for directed graphs, edge coloring, enumeration of potential colored label sets, and variable number of nodes. All permutations of the candidate adjacency matrices are considered. The graph structure space for the ISB approach is denoted \mathcal{G}_0 since $\mathcal{G}_1 \subseteq \mathcal{G}_0$. However with this generality comes an enormous space, potentially too large to be useful for certain problems.

We can analyze this statement by observing how the ISB block method handles some of the proposed NSCs. For a fair discussion, we should restrict the space to a certain block (fixed number of vertices and color label set ordering). Then both S_2 and S_7 can be naturally satisfied by removing the infeasible entries in the adjacency matrix. However, S_5 is not satisfied for large portions of \mathcal{G}_0 ; the degree of a vertex is not directly controlled. Once additional NSCs are added, the probability that an index results in a feasible graph might be so small that none are ever found.

To illustrate this consider the number of permutations of $A(G^{CC})$ with N_C components [44]:

$$\mathcal{T}(N_c) = 2^{N_C(N_C-1)/2} \quad (2.11)$$

with the first several values being $\mathcal{T}(1) = 1$, $\mathcal{T}(2) = 2$, $\mathcal{T}(3) = 8$, $\mathcal{T}(4) = 64$, $\mathcal{T}(5) = 1,024$, $\mathcal{T}(6) = 32,768$, $\mathcal{T}(7) = 2,097,152$, $\mathcal{T}(8) = 268,435,456$. Now consider the case when $N_P = 30$ and $N_C = 20$, then there are 6×10^{15} PMs versus 2×10^{57} adjacency matrix permutations. Both numbers are quite large but a clear combinatorial advantage is seen

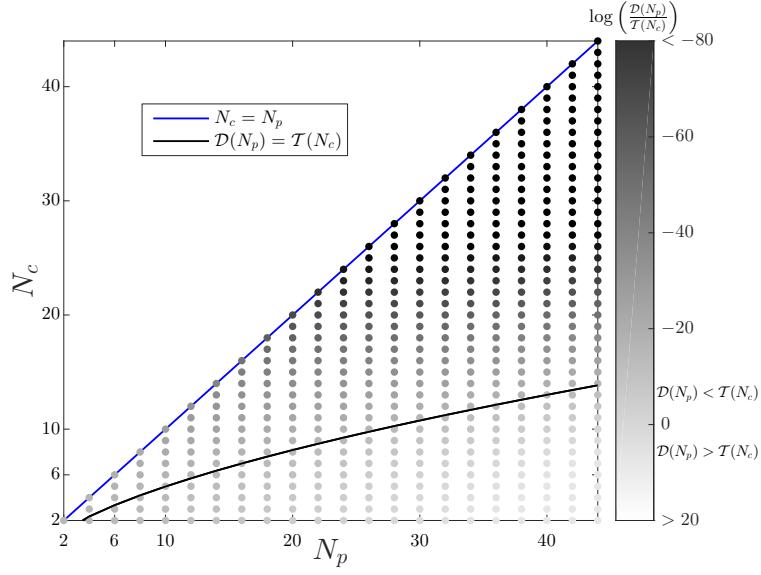


Figure 2.4: Comparison of number of graphs with PM approach and adjacency matrix approach.

with the PM approach (see Fig. 2.4). This will be exacerbated when structured components considered. However, since N_P and N_C can be different, there are some combinations where $T(N_C)$ is actually smaller than $D(N_P)$. This is shown in the figure with the curved line $D(N_P) = T(N_C)$. Most architecture design problems are above this line.

A PM approach can be seen as an alternative to permuting all possible adjacency matrices assuming the architecture design problem is based on (C, R, P) with NSC S_5 . The question then becomes does every port being filled as in a PM approach result in all architectures defined by a certain architecture design problem? Consider that we can always include 1-port components that represent empty connections, i.e., this component type implies that the vertex and edge can be removed from the graph without loss. We can control what components are allowed to have empty connections with S_7 . Certain NSC sets such as $\{S_1, S_3, S_4, S_6\}$ would also require every port to be filled.

2.2.2 Colored Graph Isomorphisms

If we have a list of useful graphs, how many of them are truly different? Determining if two graphs are *different* is known as the graph isomorphism problem. We define uniqueness among a set of architecture graphs to mean that no two candidate architectures are isomorphic.

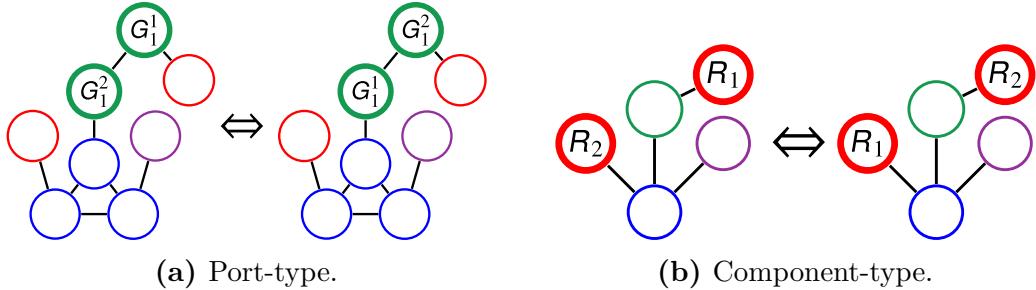


Figure 2.5: Two different type of isomorphisms.

Definition 3 (Isomorphism). Let $G = (V, E)$ and $G' = (V', E')$. We call G and G' isomorphic, and write $G \simeq G'$, if there exists a bijection $f : V \rightarrow V'$ with $(v_i, v_j) \in E \Leftrightarrow (f(v_i), f(v_j)) \in E'$ for all $v_i, v_j \in V$. The map f is called an isomorphism [61].

Definition 4 (Colored Graph Isomorphism). The colored graph isomorphism problem is to decide the existence of a color preserving isomorphism between a pair of colored graphs $G = (V, E, L)$ and $G' = (V', E', L')$, i.e., a mapping $f : V \rightarrow V'$ satisfying the following conditions:

1. f is an isomorphism by Definition 3.
2. $\text{color}(v) = \text{color}(f(v))$ for all $v \in V$.

We can better understand how the colored graph isomorphism problem affects the architecture design problem by looking at two different isomorphisms:

- *Port-type isomorphism* occurs when a component has ports that are indistinguishable in a modeling sense and can occur when using a ports representation. We have already termed such components as simple components. For example, consider a 2-port component that physically represents a mechanical translational spring. The two ports can be permuted and the resulting physical model will be equivalent. This demonstrated in Fig. 2.5a with the simple component type G . G^{CC} for the same graphs would be identical since the information about specific ports is lost. We leverage this fact to perform an initial port-type isomorphism filter to remove PMs that certainly have a port-type isomorphism. For a given simple n -port component, there are $n!$ ways to arrange the ports such that a port-type isomorphism occurs.
- *Component-type isomorphism* occurs when switching a pair of component type replicates preserves the graph. This type of isomorphism is present due to the arbitrary subscript numbers assigned to each vertex and is demonstrated in Fig. 2.5b. The 1-port component type R is permuted but since R_1 and R_2 are the same component type,

the graph remains the same (in the sense of Definition 4). For n replicates of a component type, there are $n!$ ways to arrange the components such that a component-type isomorphism occurs.

We now define the final graph structure space $\mathcal{G}_3 \subseteq \mathcal{G}_2$ representing all unique useful graphs. Assuming no NSCs except those naturally satisfied by a PM approach, we can discern a very rough lower bound on the size of this set with:

$$\mathcal{D}(N_P) \times \prod_{i=1}^{n_C} \frac{1}{R_i! \times (P_i!)^{R_i}} \leq |\mathcal{G}_3| \leq \mathcal{D}(N_P) \quad (2.12)$$

where this formula assumes all port-type and component-type isomorphisms that could occur, do occur in the set of PMs. Consider $(C, R, P) = (A, 6, 1)$, then Eqn. (2.12) provides a lower bound of 0.02 graphs, but we know there is exactly 1 unique graph.

Although the graph isomorphism problem is NP (nondeterministic polynomial time), there are many efficient practical algorithms [67]. Study of the graph isomorphism problem is an ongoing field and recent breakthroughs could lead to improved algorithms [68]. In this work, we utilize the python package igraph using the ISOMORPHIC_VF2 function [69] based on the VF2 algorithm [70] to solve the colored isomorphism problem.

Many architecture design studies ignore the isomorphism problem but presence of isomorphic graphs leads to the evaluation of non-unique options [54]. For certain problem sizes, the complexity of checking for isomorphisms may be much greater than generating and evaluating new, potentially non-unique graphs. But to understand the effect of problem definition, NSCs, and candidate graph generation algorithms on \mathcal{G}_3 requires the isomorphism checks, and can lead to insights into new algorithms that naturally avoid the isomorphism problem [71]. Other graph generation algorithms have been developed that avoid isomorphic graphs [72, 73]. Furthermore, for appropriately sized problems, the isomorphism check is computationally viable.

Algorithm 2.3 was developed to determine \mathcal{G}_3 given \mathcal{G}_2 . This algorithm checks a candidate graph against bins of already found unique graphs and stops checking if an isomorphism is found, making it parallelizable to a degree and removes unnecessary checks. There are a number of quick preliminary checks that can be done between two graphs, as necessary conditions for them to be isomorphic include having the same number of vertices, edges, and color label distributions.

Algorithm 2.3: Determination of the unique colored graphs given a set of colored graphs.

Input : Graphs – set of colored graphs
Nbin – number of bins (for parallel processing)

Output: UniqueGraphs – set of unique colored graphs

```

1 ind ← 1                                /* initialize index for total unique graphs */
2 bin(1).Graphs(1) ← Graphs(1)            /* first graph is always unique */
3 for i ← 2 to length(Graphs) do          /* check remaining graphs */
4   G1 ← Graphs(i)                         /* current graph to check */
5   for j ← 1 to min(Nbin, ind) do in parallel /* check against each nonempty bin */
6     k ← length(bin(j).Graphs)             /* unique graphs in bin */
7     IsoFlag ← 0                           /* initialize flag, 0 is not isomorphic */
8     while (k > 0) and (IsoFlag = 0) do    /* while graphs remain and isomorphism not found */
9       G2 ← bin(j).Graphs(k)                /* a unique graph */
10      if G1 and G2 pass preliminary isomorphism checks then
11        | IsoFlag ← isomorphic_vf2(G1, G2)  /* return 1 if G1 and G2 are isomorphic */
12      end
13      k ← k - 1                          /* decrease index since G2 checked */
14    end
15    results(j) ← IsoFlag                /* assign result for bin c */
16  end
17  if all elements of results are 0 then   /* if no isomorphisms */
18    J ← mod(ind, Nbin) + 1               /* index for next smallest bin */
19    bin(J).Graphs(end + 1) ← G1         /* assign to a bin */
20    ind ← ind + 1                      /* total unique graphs */
21  end
22 end
23 UniqueGraphs ← combine graphs in bin into a single set of graphs

```

2.3 Tree Search Algorithm

With a better understanding of the colored graph isomorphism problem in the context of architecture design, a tree search algorithm was developed to more efficiently enumerate a graph structure space that contains \mathcal{G}_3 . This algorithm is based on the idea that for simple components, the port ordering does not matter so we are free to always choose the first port of a component when making edges.

Algorithm 2.4 starts with a vector for length N_C where the entries are the number of ports for every component in G^{CC} . For example, if $P = [1 \ 2]$ and $R = [2 \ 3]$, then this vector would be $V = [1 \ 1 \ 2 \ 2 \ 2]$ and $cVf = [2 \ 3 \ 5 \ 7 \ 9]$. Recursion is then applied to enumerate all possible edge combinations where each recursive step adds an edge. The end result is a set of missorted PMs, i.e., the sequential pairs that define the edges need to be sorted such that they fit the definition of a PM in Sec. 2.1.2 (but no PM will occur twice and the

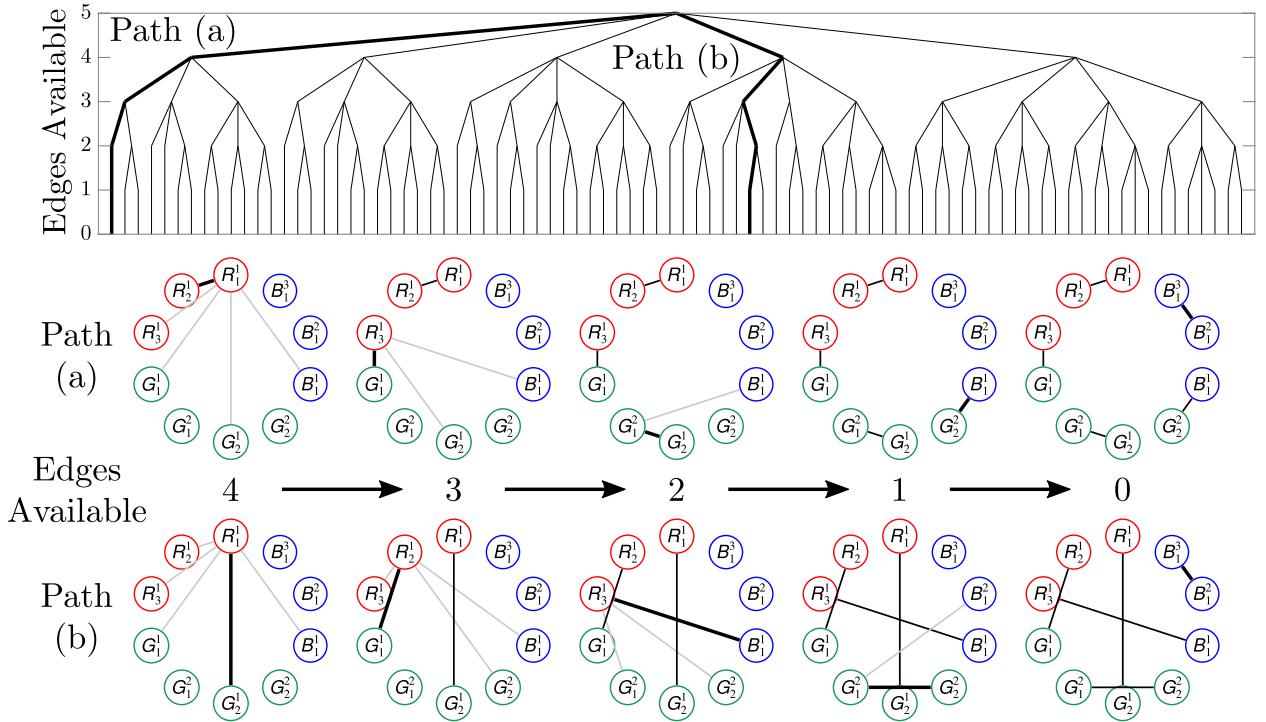


Figure 2.6: Tree structure for Case Study 1 using the basic tree search algorithm in Alg. 2.4.

property of naturally satisfying S_2 and S_5 is maintained). The end result is an algorithm that does not produce many PM graphs that would certainly have a port-type isomorphism. A visualization of the tree-like behavior is in Fig. 2.6 where each leaf in the tree is a new call of the algorithm and the branches are the loops through the possible remaining edges. Not all leaves have the same number of branches since components become completely connected at different times. Two paths are shown in the figure for a particular architecture design problem, each resulting in a different PM. Note that the thicker black lines indicate the chosen edge and the gray lines are the other potential edges for the particular leaf instance.

This approach is similar to a number of reported graph enumeration algorithms [59, 73–75]. The tree algorithm shares some similarities with deletion orderly algorithms [75]. Snavely and Papalambros developed a tree algorithm that only deals with structured components, which limits the design space by the number of components rather than the (C, R, P) notation in this work [59]. Faulon et al. enumerate molecules of a specific signature height with a recursive algorithm, but do not allow for components to be removed. Thus, they do not cover the same architecture design space [73]. Neither of these works make the connection between PM theory and architecture enumeration which provides a number of insights and

Algorithm 2.4: Basic tree search algorithm.

Input : V – vector of remaining ports for each component replicate
 E – vector of edges in sequential pairs, initially empty
 cVf – cumulative sum of the original V plus 1
 SavedGraphs – set of graphs, initially empty

Output: SavedGraphs – set of graphs

```

1  $iL \leftarrow \text{find}(V, \text{first})$                                 /* find first nonzero entry */
2  $L \leftarrow cVf(iL) - V(iL)$                                      /* left port */
3  $V(I(1)) \leftarrow V(iL) - 1$                                        /* remove port */
4  $I \leftarrow \text{find}(V)$                                          /* find nonzero entries */
5 for  $iR \leftarrow I$  do                                         /* loop through all nonzero entries */
6    $R \leftarrow cVf(iR) - V(iR)$                                      /* right port */
7    $E2 \leftarrow [E, L, R]$                                          /* combine left, right ports for an edge */
8    $V2 \leftarrow V$                                                  /* local remaining ports vector */
9    $V2(iR) \leftarrow V2(iR) - 1$                                       /* remove port (local copy) */
10  if any element of  $V2$  is nonzero then                         /* recursive call if any remaining vertices */
11    |  $\text{SavedGraphs} \leftarrow \text{Algorithm 2.4}$  with  $V2, E2, cVf, \text{SavedGraphs}$ 
12  else
13    |  $\text{SavedGraphs}\{\text{end} + 1\} \leftarrow E2$                       /* save missorted perfect matching */
14  end
15 end

```

practical functions.

We can further improve on this algorithm by adding a single line between lines 3 and 4 that will result in graphs that always satisfy S_7 (feasible edge constraints). First, expand A_R such that its size is the same as G^{CC} where 0 entries still indicate infeasible edge constraints. The additional line would then be: $\text{Vallow} \leftarrow A(iL, :) \circ V$. By finding the nonzero entries of Vallow instead of V , we limit the for-loop to edges that are feasible. This has the intentional effect that certain branches of the tree will terminate before a feasible PM is found. Therefore when S_7 is present, we will utilize this ‘improved’ tree search algorithm to more efficiently enumerate \mathcal{G}_3 . Additional enhancements to the tree search algorithm are discussed in Appendix A as well as alternative tree traversal strategies.

2.4 Enumeration Case Studies

In this section, a number of case studies are provided to demonstrate the theoretical aspects of the previous sections⁴.

⁴Ref. [65] contains MATLAB codes that replicate the results from these enumeration case studies and can generate graphs for (C, R, P) architecture design problems.

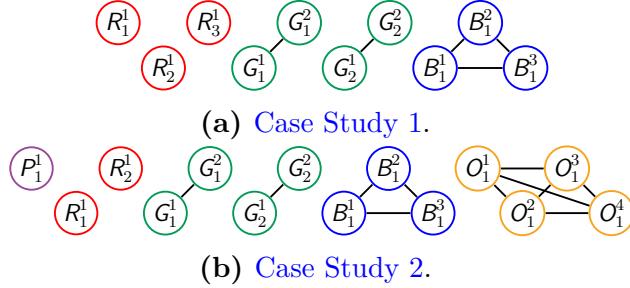


Figure 2.7: G^P graphs for two examples.

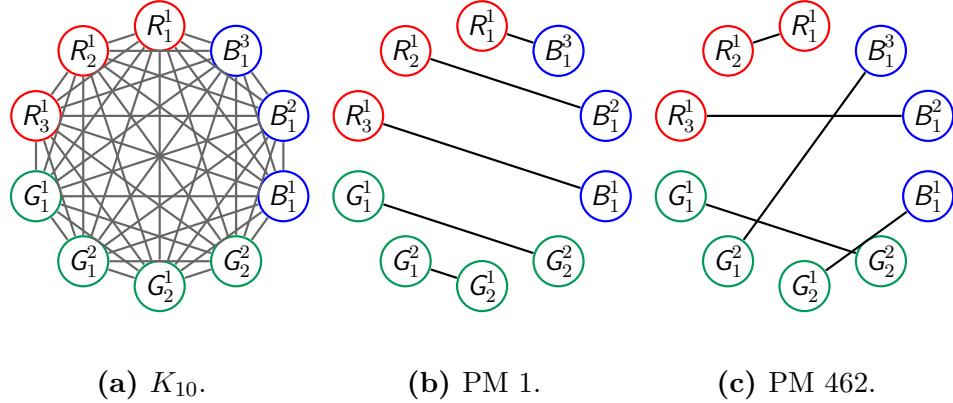


Figure 2.8: Select interconnectivity graphs for [Case Study 1](#).

2.4.1 Case Study 1

The base three-tuple is specified as:

$$C = \{R, G, B\}, \quad R = [3, 2, 1], \quad P = [1, 2, 3]$$

This example has three different simple component types that have multiple ports and replicates and is visualized in Fig. 2.7a. Then G^P is:

$$\begin{aligned} V^P &= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \\ E^P &= \{(4, 5), (6, 7), (8, 9), (8, 10), (9, 10)\} \\ L^P &= \{R_1^1, R_2^1, R_3^1, G_1^1, G_1^2, G_2^1, G_2^2, B_1^1, B_1^2, B_1^3\} \\ N_P &= 10, \quad N_C = 6, \quad 3.28 \leq |\mathcal{G}_3| \leq 945 \end{aligned}$$

In Fig. 2.8 we see G^I for two different PMs. Then in Fig. 2.9a we can see G^{CP} for PM 1. This can then be mapped to the equivalent G^{CC} shown in Fig. 2.9b. The basic tree search algorithm will have the same tree regardless of the NSCs and this tree is visualized in Fig. 2.6.

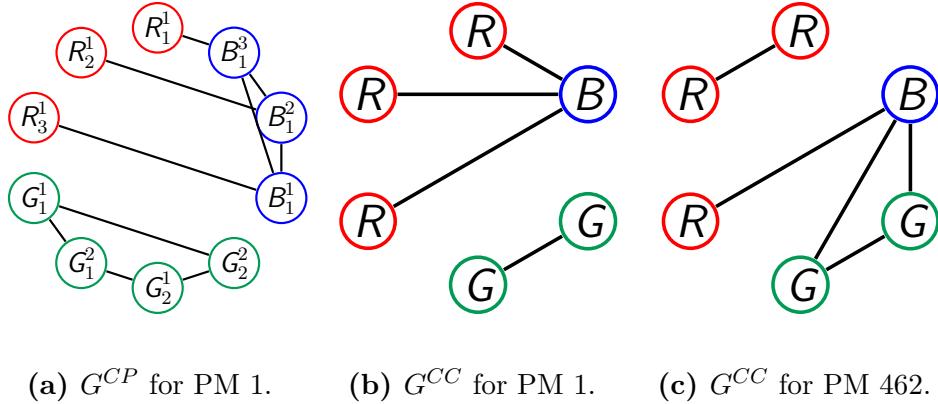


Figure 2.9: Select connected ports and connected component graphs for [Case Study 1](#).

Both graphs in Figs. 2.9b and 2.9c will have a topologically equivalent graph appear in the set of graphs generated by this algorithm. Now two different sets of NSCs will be discussed.

1. No additional NSCs. There are 32,768 adjacency matrices (\mathcal{G}_0); 945 PMs (\mathcal{G}_1); 86 candidate graphs with basic tree search algorithm; 77 remaining candidate graphs after initial port-type isomorphism filter; and 77 feasible graphs (\mathcal{G}_2). Finally, there are only 16 unique graphs (\mathcal{G}_3) consistent with Eqn. (2.12) and all shown in Fig. 2.10. 258 graph comparisons were needed and only 113 required a full isomorphism check.
2. NSCs S_1 , S_3 and S_4 with $M = [0, 0, 1]$, and S_6 with P as the number of unique edges. Same as 1 above until the feasibility checks and here there are only 23 feasible graphs due to the additional NSCs. Finally, there are only 5 unique graphs (\mathcal{G}_3) all shown in Fig. 2.11. Note that vertices not connected to (B) have been removed. 37 graph comparisons were needed and only 19 required a full isomorphism check. If $M = [1 1 1]$, then only 2 unique designs are possible.

2.4.2 Case Study 2

This example has five different component types that have multiple ports including multiple 1-port component types and is visualized in Fig. 2.7b. The base three-tuple is specified as:

$$C = \{P, R, G, B, O\}, \quad R = [1, 2, 2, 1, 1], \quad P = [1, 1, 2, 3, 4] \\ N_P = 14, \quad N_C = 7, \quad 58.7 \leq |\mathcal{G}_3| \leq 135135$$

1. No additional NSCs. 2,097,152 adjacency matrices (\mathcal{G}_0); 135,135 PMs (\mathcal{G}_1); 1,119 candidate graphs with basic tree search algorithm; 767 remaining candidate graphs after

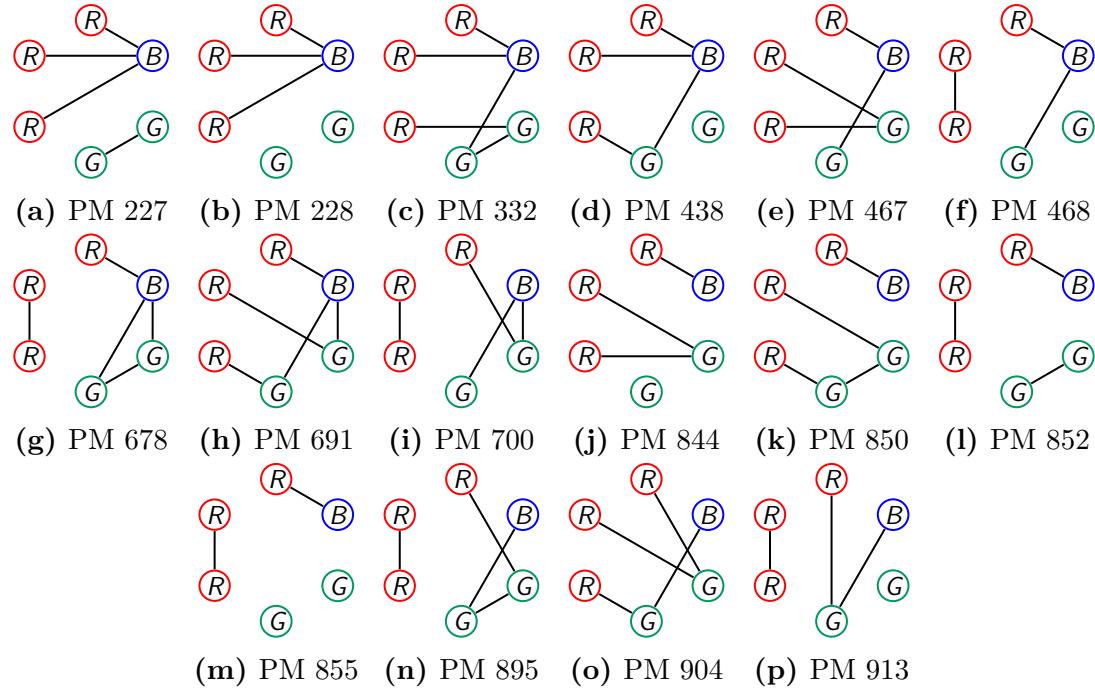


Figure 2.10: All 16 unique graphs with no additional NSCs for [Case Study 1](#).

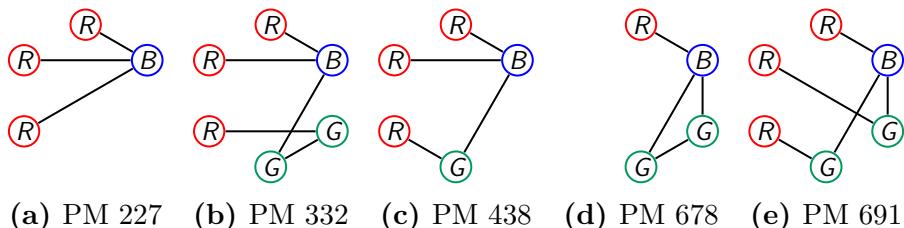


Figure 2.11: All 5 unique graphs for [Case Study 1](#) requiring a connected graph containing B and a specified number of unique edges.

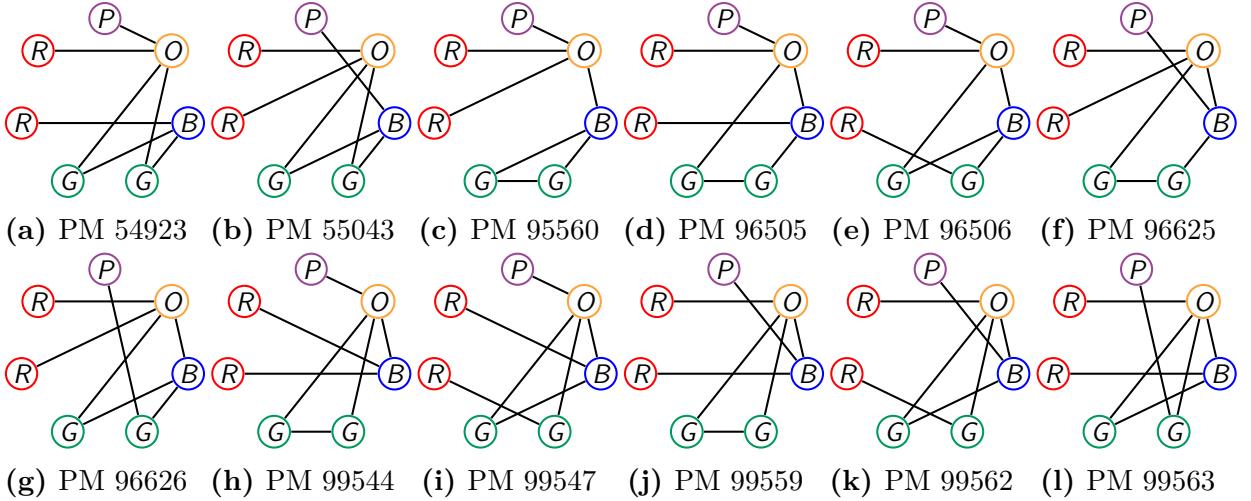


Figure 2.12: All 12 unique graphs for Case Study 2 requiring all components to be connected and a specified number of unique edges.

initial port-type isomorphism filter; 767 feasible graphs (\mathcal{G}_2); 274 unique graphs⁵ (\mathcal{G}_3). 41,036 graph comparisons were needed and only 11,828 required a full isomorphism check.

2. NSCs S_1 , S_3 and S_4 with $M = [1, 0, 0, 0, 0]$. Same as 1 above except there are only 137 unique graphs that contain P and are connected.
3. NSCs S_1 , S_3 and S_4 with $M = [1, 1, 1, 1, 1]$, and S_6 with P as the number of unique edges. Same as 1 above until the feasibility checks and here there are only 31 feasible graphs due to the additional NSCs. Finally, there are only 12 unique graphs all shown in Fig. 2.12. 102 graph comparisons were needed and all 102 required a full isomorphism check.
4. Same as 3 except with $C = \{P, R, G, B, O\}$ modified, $M = [1, 1, 1, 0, 1, 1]$, and appropriate changes to P and R . Therefore requiring at least one G to be present but otherwise the same. Now there are 34 feasible graphs and 14 are unique (more than 3 since this is a less constrained problem).

2.4.3 Case Study 3

A graph representing a quarter-car suspension was introduced in Fig. 2.1a; now we will seek graphs that have different topologies between the sprung (S) and unsprung (U) masses represented in Fig. 2.13. These graphs could then be used with design studies that eval-

⁵The 274 graphs are in Fig. B.2.

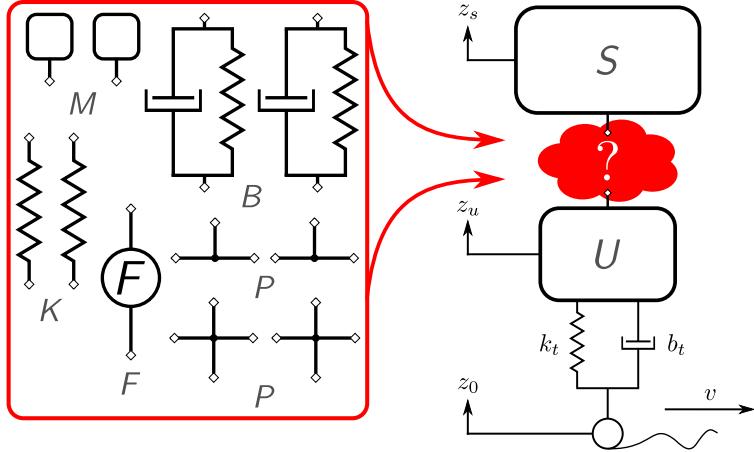


Figure 2.13: Suspension architecture enumeration case study.

uate the performance of a particular suspension architecture (see Ref. [27] for a design study on a particular architecture). The components considered will be additional masses, springs, dampers, a force actuator, and parallel connections (these are schematically shown in Fig. 2.13). The specific selection of (C, R, P) was chosen to be near the limits of what is currently possible with the proposed enumeration methods in this article.

Some additional assumptions are also made on the component definitions. First, B is a parallel damper and spring to ensure that there is a stable equilibrium point for the damper. Next, both 3- and 4-port parallel components are included to facilitate more efficient generation of the useful architectures. A 4-port parallel connection is equivalent to two 3-port parallel connections but the 4-port component provides structure that can be utilized with some specific NSCs. With the problem outlined, the base three-tuple is specified as:

$$C = \{S, U, M, K, B, F, P\}, \quad R = [1, 1, 2, 2, 2, 1, 2, 2]$$

$$P = [1, 1, 1, 2, 2, 2, 3, 4], \quad N_P = 28, \quad N_C = 13$$

$$1.01 \times 10^7 \leq |\mathcal{G}_3| \leq 2.13 \times 10^{14}$$

The NSCs for this case study are now listed with expanded details on reduced potential adjacency matrix A_R . A few of the constraints utilize insights from the physical modeling of the suspension architectures.

- S_1, S_3 and S_4 with $M = [1, 1, 0, 0, 0, 0, 0, 0]$ enforcing that both the sprung and unsprung masses must be connected and all components not connected to these two are removed.
- S_6 with P as the number of unique edges.
- $A_R(2, 1) = 0$ for S_7 to avoid a direct connection between the sprung and unsprung

masses as it would defeat the purpose of a suspension to isolate the masses. If this constraint was not added, 1/27 of graphs generated from a pure PM approach would contain this connection.

- $A_R(3,1) = A_R(3,2) = 0$ for S_7 since a feasible graph cannot have either S or U be connected to M as there would not be a path between between S and U . Therefore, no unique graphs are lost with this constraint. Rather, a more efficient enumeration results.
- $A_R(8,8) = A_R(7,7) = A_R(8,7) = 0$ for S_7 so no parallel components can be connected to each other. This greatly reduces the number of graphs generated by providing some specific structure on the number and type of parallel connections in the architectures.
- $A_R(4,4) = A_R(5,5) = 0$ for S_7 so no two K or B components can be connected in series since there are straightforward relationships to combine these series elements into a single equivalent component. By eliminating this type of connection when generating graphs, we have a substantially smaller number of graphs to evaluate.
- No parallel connection path can exist between a connected S or U as these masses would not be isolated. This is slightly different than the NSCs in Sec. 2.2.1 and is checked after S_7 .
- Cycles appear with single parallel components where the components in the cycle would not appear in the dynamic model based on the properties of a parallel connection (e.g., with the cycle $P \xleftarrow{K} B$, neither K nor B would appear in the dynamic model). Graphs with these cycles are declared unuseful since an isomorphic graph with the parallel cycle removed would already appear in the set of graphs generated by the tree algorithm.
- Many series connections between the 2-port components are interchangeable (e.g., $F-K$ and $K-F$ in series are physically equivalent). Therefore, these interchangeable series components are combined into a single equivalent component type (modifying the graph) and checked for isomorphisms. For the previously mentioned example, the equivalent component type would be FK based on alphabetical ordering of the original component labels.

The complete A_R is shown in Fig. 2.14a and its expansion to the potential adjacency matrix in Fig. 2.14b noting that both of these matrices are symmetric. Figure 2.14b has 1s on the diagonal since self-connections should be allowed so the desired graph structure space is covered. For example, we might want to consider graphs where all components are present except a single K component and this only can occur if the detached K is connected to itself (and later removed).

	<i>S</i>	<i>U</i>	<i>M</i>	<i>K</i>	<i>B</i>	<i>F</i>	<i>P</i>	<i>P</i>
<i>S</i>	1
<i>U</i>	0	1
<i>M</i>	0	0	1
<i>K</i>	1	1	1	0
<i>B</i>	1	1	1	1	0	.	.	.
<i>F</i>	1	1	1	1	1	1	.	.
<i>P</i>	1	1	1	1	1	1	0	.
<i>P</i>	1	1	1	1	1	1	0	0

(a) Reduced potential adjacency matrix A_R .

	<i>S</i>	<i>U</i>	<i>M</i>	<i>M</i>	<i>K</i>	<i>K</i>	<i>B</i>	<i>B</i>	<i>F</i>	<i>P</i>	<i>P</i>	<i>P</i>	<i>P</i>
<i>S</i>	1
<i>U</i>	0	1
<i>M</i>	0	0	1
<i>M</i>	0	0	1	1
<i>K</i>	1	1	1	1	1
<i>K</i>	1	1	1	1	0	1
<i>B</i>	1	1	1	1	1	1	1
<i>B</i>	1	1	1	1	1	1	0	1
<i>F</i>	1	1	1	1	1	1	1	1	1
<i>P</i>	1	1	1	1	1	1	1	1	1	1	.	.	.
<i>P</i>	1	1	1	1	1	1	1	1	1	0	1	.	.
<i>P</i>	1	1	1	1	1	1	1	1	1	0	0	1	.
<i>P</i>	1	1	1	1	1	1	1	1	1	0	0	0	1

(b) Potential adjacency matrix.

Figure 2.14: Suspension case study matrices for \mathcal{S}_7 and the tree search algorithm.

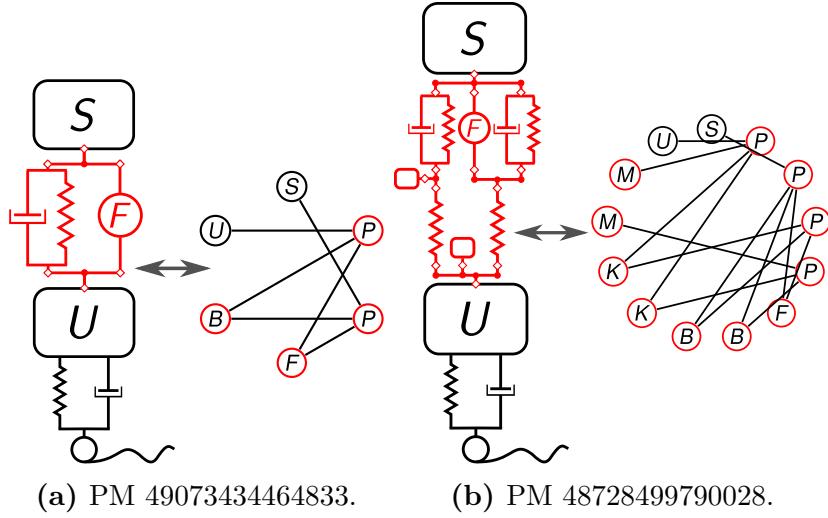


Figure 2.15: Two architectures for the suspension case study.

The results are presented in similar manner to the previous case studies: 4.7×10^{21} adjacency matrices⁶ (\mathcal{G}_0); 2.1×10^{14} PMs (\mathcal{G}_1); 1.6×10^8 candidate graphs generated through the basic tree search algorithm; 3.2×10^7 remaining candidate graphs after initial port-type isomorphism filter; 1.9×10^6 feasible graphs (\mathcal{G}_2); and 13,727 unique graphs (\mathcal{G}_3). 2×10^9 graph comparisons were needed and 3×10^7 required a full isomorphism check. Two unique architectures are shown in Fig. 2.15 with Fig. 2.15a being a common architecture [27].

2.5 Discussion

It is clear in the case studies that the number of unique designs is much smaller than the upper bounds given by either permutations of the adjacency matrix or a PM approach. We also can directly visualize the effect of adding specific NSCs. NSCs limited the architecture design space, but in a predictable manner. One such example was not allowing parallel components to be connected together in [Case Study 3](#). A 5-port parallel connection was no longer possible (i.e., a 3-port and 4-port parallel components connected), but this NSC greatly reduced the number of graphs generated excluding many infeasible graphs, such as ones where parallel connection paths existed between a connected S or U . Therefore, the

⁶This represents the fairest comparison as the proper permutations of the ports representation of the potential adjacency matrix in Fig. 2.14b without the parallel components. There are 5.4×10^8 adjacency matrices with a component representation but this suffers from the unknown parallel connection issue discussed in Sec. 2.1. There are 3.0×10^{23} adjacency matrices directly using the matrix in Fig. 2.14b.

addition of this NSC was a decision based on the tradeoff between coverage of the architecture design space and efficiency. [Case Study 3](#) also demonstrated that fairly large problem sizes can be enumerated with the improved tree search algorithm provided enough constraints are present. Also, all possible subgraphs that are connected and complete appear in the set of unique designs without any NSCs (e.g., all graphs in Fig. 2.11 appear as subgraphs in Fig. 2.10).

All reported unique solutions have a corresponding PM number. This number may not be unique since other PM numbers maybe isomorphic to the resulting G^{CC} . We see an example of two different PMs producing isomorphic graphs (PM 462 in Fig. 2.9c and PM 678 in Fig. 2.10g). While checking for isomorphisms can be computationally demanding, there typically is only a small subset of graphs that need the full isomorphism check as many comparisons fail with the simple checks and filters. Algorithm 2.3 can be useful to any architecture design problem no matter how the set of colored graphs is generated. Many of the results and algorithms assumed simple components, but structured components (i.e., a planetary gear) can be readily included. Replacing a simple component type with an equivalent structured component type would simply have the effect of increasing the number of unique designs.

The previous sections only considered enumeration constructing a specific graph structure space. However, many problems are too large for the proposed enumeration algorithms because computational limits (e.g., memory needed to store the graphs and computation time of Alg. 2.3) and evaluation limits (i.e., too many graphs are generated and we cannot evaluate all of them). Therefore, we need to consider methods that provide suitable *exploration* of the desired design space.

Both the pure PM approach and the tree search algorithms have nice properties such as the high likelihood of producing feasible, nonisomorphic graphs while not limiting the design space. A stochastic sampling of the unique integers between 1 and $\mathcal{D}(N_P)$ can produce any arbitrary number of PM graphs. However, more structured sampling approaches may be preferred. Consider the unique graphs in Fig. 2.10. We could have tested all PMs between 227 to 913 and found all unique graphs. A PM approach does exhibit some interesting similarly-preserving properties (e.g., the graphs for a given PM number and the next integer value have a high likelihood of containing similar edges). Further exploration of the structural properties of PM graphs could lead to better sampling techniques that still cover the desired design space.

We can further consider ways to structure the exploration space with the coupon collector's problem. This problem, stated in a form relevant to this article, is:

There are n unique graphs and at each trial, a new graph is chosen at random (with replacement). Let m be the number of trials. What is the expected number of trials such that all n unique graphs are selected?

The expected number of trials needed grows as $O(n \ln(n))$ [76]. Some of the assumptions in the problem are not directly satisfied such replacement and the probability distribution of choosing a particular unique graph but further study on the structure of \mathcal{G}_1 may yield exploration that ‘collects’ most of the unique graphs in a more efficient manner.

The tree search algorithms may also be used for exploration. On Line 5 of Alg. 2.3, we can randomly select an edge to add from \mathbf{l} instead of trying all possible edges. Therefore, the tree can be explored stochastically. Since the number of branches from a leaf varies, the probability of arriving at a certain final edge set is not equal (these probabilities can be calculated by assuming the tree is a Markov chain). Since the tree search algorithms cover the same desired graph structure space as the pure PM approach, can we selectively sample the tree and have some predictions on when all unique designs are found? These questions are left as future work items.

Finally, it is important to describe the specific uses of the proposed algorithms. They are suitable for problems that are represented by undirected colored graphs under the component/port paradigm [43, 51, 59]. Enumeration is appropriate for certain problem classes (primarily determined by size). It may also be appropriate for searching for all possible enhancing structures [51]. Enumeration has been useful for generating lists of organic molecules [73, 74, 77], finding all geometries of electrical circuits [78], identifying all biological network architectures that achieve specific behaviors [16], enumerating different gear trains [14, 15], and determining all hybrid powertrain configurations for a set list of components [13].

Exploration is suitable for sampling the design space for much larger problems [44]. These samples could be used as visualizations for expert evaluation or starting points for generative approaches. The unrestricted graphs from a PM approach could also be used in conjunction with feature extraction algorithms to develop generative rules that are not based solely on experience and intuition (where the features are subgraphs that provide desired benefits to the architecture) [79].

2.6 Summary

Architecture design is a challenging problem and this chapter presents some theory for generating candidate architectures with perfect matchings. A PM approach is a graph numerical

representation scheme that completely covers the design space that is needed in many architecture design problems. It ensures certain frequency and degree requirements are met on specific list of potential components. Enumeration of architecture design spaces can provide coverage and insights not currently possible with generative design approaches since enumeration approaches allow the designer to make specifications that they understand such as constraints and potential components, rather than rules for how things should be connected.

A number of general network structure constraints are fully outlined with the specifics of checking their satisfaction with available graph analysis tools. The colored graph isomorphism problem is discussed in great detail including the distinction between port-type and component-type isomorphisms. The limited number of full isomorphism checks and the efficiency of Alg. 2.3 demonstrate that larger than expected architecture design spaces can be obtained. A basic and improved tree search algorithm that avoids port-type isomorphisms was shown and is a primary example of how constraints can be naturally satisfied without reducing the design space.

The various case studies are initial insights into the true nature of the class of architecture problems studied in this dissertation. Consider again that there are only 12 unique graphs in Fig. 2.12 of 2,097,152 adjacency matrices and 135,135 PMs. For the suspension case study, a wide variety of network structure constraints were included based on natural requirements such as no direct connection between the sprung and unsprung masses. Other constraints were added to avoid duplicate dynamic models such as the avoidance of parallel cycles and series connections between the 2-port components. Moreover, constraints based on the experience and intuition of the designer were also included which limited the design space in a predictable manner such as the requirement that no parallel components can be connected together.

Future graph generation algorithms can use these insights to suitably address the unique challenges associated with architecture design problems. A number of directions are possible with the PM framework including deeper analysis of the structural properties of PM graphs, reduction of the number of graphs generated by the tree search algorithm, and the development of structured sampling approaches that result in nearly all unique graphs.

Chapter 3

Co-Design: Combined Plant and Control Design⁷

“If we want to solve problems effectively... we must keep in mind not only many features but also the influences among them. Complexity is the label we will give to the existence of many interdependent variables in a given system. The more variables and the greater their interdependence, the greater the system’s complexity. Great complexity places high demands on a planner’s capacity to gather information, integrate findings, and design effective actions.”

D. Dörner [80, p. 38]

Many dynamic engineering system design problems contain two groupings of the design variables: plant (or artifact) and control. For instance, consider again the system-level design of a robotic manipulator presented in Chapter 1. The plant design may comprise the geometric properties of the links and the control design may be embodied by the joint torque time trajectories for a specific task [4]. Many authors have shown the benefit of a combined strategy rather than a sequential approach [27, 33, 81–83]. With a sequential approach, the plant is optimized initially, followed by the controller [21, 83]. In this chapter, we focus on two solution strategies appropriate for combined plant and controller design or co-design: *simultaneous* and *nested*.

The simultaneous solution strategy optimizes both the plant and control variables in same optimization formulation. With the nested strategy, an outer optimization loop optimizes the plant design, and an inner optimization loop identifies the optimal control for each plant design tested by the outer loop [21, 83]. These two strategies are selected for a few reasons beyond them being the most studied and used in the literature. First, the simultaneous strategy is the most fundamental representation of an integrated design problem [84]. On the other hand, the nested strategy is a specific reorganization of the optimization problem based on the plant and control disciplines. Multidisciplinary design optimization (MDO)

⁷Elements of this chapter are based on work completed in Ref. [32].

is a field of research that investigates design methods for systems with multiple disciplines [21, 84]. However, if we are limited to single-system problems and implementations that do not partition the system across trajectories, there are a limited number of appropriate MDO methods suitable for co-design. One reason for this is that many MDO formulations were developed in a way that does not explicitly address the dynamic nature of general co-design problems [21]. This has led to partitioning schemes without full consideration of the coupling in the system. Allison and Nazari developed an augmented Lagrangian coordination method for co-design, but only accounted for unidirectional (plant) coupling [85]. The nested approach naturally handles single-system problems, bidirectional coupling, and the various trajectories in the problem. Another important motivating reason for reorganizing the original problem is to employ specialized optimization algorithms to solve specific subproblems [85, 86]. A number of efficient solution methods for specific problem forms can be utilized with the nested strategy. However, as it will be discussed throughout this article, both of these strategies have their drawbacks. Many of these issues are well known for simultaneous strategy, motivating the fields of MDO and distributed optimization [84]. The nested strategy is not amenable to coarse-grained parallelism and can be computationally intensive [85]. Additionally, it can have potential feasibility issues.

Here we consider co-design problems that are well-posed as dynamic optimization (DO) problems [21, 25, 83]. Co-design theory has focused primarily on specific DO formulations; as a result, only limited types of co-design problems fit into the existing frameworks. With a specific problem structure, a variety of algorithms have been developed to provide numerical solutions, and detailed analysis of the coupling and partitioning has been investigated [83, 87–93]. Such works have made considerable progress towards addressing specific challenges found in certain co-design problems. However, there has been less attention towards the general co-design problem (i.e., a co-design formulation with no restrictions) perhaps due to the lack of efficient solution techniques and continuing legacy design practices that treat certain problem elements as naturally separate [21].

Recently, numerical solution methods have been utilized to solve co-design problems that are not captured by previous problem definitions [21, 25]. Nontraditional problem formulation elements include system-level objective functions, general inequality path constraints, and general boundary conditions [4, 18, 24, 27, 33, 81, 82, 94]. With these nontraditional problem elements, much of the previous work in co-design theory does not apply. An essential element of engineering design optimization is an appropriate problem formulation. A thorough analysis of the general co-design formulation is needed to better allow designers to leverage the advantages intrinsic to the co-design methodology.

One of the numerical solution methods recently applied to co-design problems is known as direct transcription (DT), which approximates the infinite-dimensional DO problem with a finite nonlinear program (NLP) that can be solved with standard NLP solvers [27, 95–97]. DT has a number of favorable properties that supports the efficient generation of solutions to general co-design problems [21, 25, 27].

In this chapter, we will explore the general co-design problem with a focus on the simultaneous and nested solution strategies. Section 3.1 provides the formulations for the simultaneous and nested co-design solution strategies and Sec. 3.2 outlines the necessary conditions for optimality for both. Section 3.3 discusses some practical solution considerations relevant to the two strategies. Section 3.4 presents a number of test problems and finally, Sec. 3.5 offers a summary.

3.1 Problem Formulation

Here we present the two co-design strategies. First, we state a few assumptions. Initially, the time horizon $t_0 \leq t \leq t_f$ is assumed to be fixed. Section 3.2.4 discusses the inclusion of the horizon boundaries in the co-design problem formulations. Next, only general inequality constraints will directly appear in the formulations for conciseness of the formulations. Equality constraints are captured by two inequality constraints (i.e., $f(\mathbf{x}) = 0$ is equivalent to $f(\mathbf{x}) \leq 0$ and $-f(\mathbf{x}) \leq 0$).

3.1.1 Simultaneous Formulation

The general simultaneous co-design problem formulation contains all relevant objectives and constraints in a single optimization formulation. The DO problem formulation is:

$$\min_{\mathbf{x}_p, \mathbf{x}_c} \quad \Psi(\mathbf{x}_p, \mathbf{x}_c) = \int_{t_0}^{t_f} \mathcal{L}(t, \boldsymbol{\xi}, \mathbf{x}_c, \mathbf{x}_p) dt + \dots \quad (3.1a)$$

$$\mathcal{M}(\boldsymbol{\xi}(t_0), \boldsymbol{\xi}(t_f), \mathbf{x}_c, \mathbf{x}_p) \quad (3.1b)$$

$$\text{subject to: } \dot{\boldsymbol{\xi}} - \mathbf{f}(t, \boldsymbol{\xi}, \mathbf{x}_c, \mathbf{x}_p) = \mathbf{0} \quad (3.1c)$$

$$\mathbf{C}(t, \boldsymbol{\xi}, \mathbf{x}_c, \mathbf{x}_p) \leq \mathbf{0} \quad (3.1d)$$

$$\boldsymbol{\phi}(\boldsymbol{\xi}(t_0), \boldsymbol{\xi}(t_f), \mathbf{x}_c, \mathbf{x}_p) \leq \mathbf{0} \quad (3.1e)$$

where \mathbf{x}_p are the plant variables, \mathbf{x}_c are the control variables, t is the time continuum defined between t_0 and t_f , $\boldsymbol{\xi}$ are the states, Ψ is the objective function in Bolza form, \mathcal{L} is



(a) Simultaneous strategy. (b) Nested strategy.

Figure 3.1: Two co-design solution strategies where \mathcal{O} indicates an optimization problem.

the Lagrange (running cost) term, \mathcal{M} is the Mayer (terminal cost) term, Eqn. (3.1c) enforces the dynamics modeled as a first-order ordinary differential equation, Eqn. (3.1d) enforces any time-varying path constraints, and Eqn. (3.1e) enforces any time-dependent constraints. See Ref. [25] for a general discussion of this co-design formulation and the problem elements.

3.1.2 Nested Formulation

The alternative solution strategy to the simultaneous formulation in Prob. (3.1) is a nested one where an outer optimization loop optimizes the plant design, and an inner optimization loop identifies the optimal control for each plant design tested by the outer loop. Figure 3.1 illustrates the two strategies. This is a two-level optimization problem (a subclass of bilevel optimization) [98–100]. The outer-loop problem is defined as:

$$\min_{\mathbf{x}_p} \psi(\mathbf{x}_p) \quad (3.2a)$$

$$\text{subject to: } \phi_o(\mathbf{x}_p) \leq \mathbf{0} \quad (3.2b)$$

$$\mathbf{F}(\mathbf{x}_p) \leq \mathbf{0} \quad (3.2c)$$

where ϕ_o are the constraints in ϕ that only depend on the plant design, \mathbf{g}_o is the collection of the outer-loop constraints, and $\{\psi(\mathbf{x}_p), \mathbf{F}(\mathbf{x}_p)\}$ are two new problem formulation elements that will be outlined after the following inner-loop formulation:

$$\min_{\mathbf{x}_c} \Psi(\mathbf{x}_p^\dagger, \mathbf{x}_c) \quad (3.3a)$$

$$\text{subject to: } \left. \begin{array}{l} \dot{\boldsymbol{\xi}} - \mathbf{f}(t, \boldsymbol{\xi}, \mathbf{x}_c, \mathbf{x}_p^\dagger) = \mathbf{0} \\ \mathbf{C}(t, \boldsymbol{\xi}, \mathbf{x}_c, \mathbf{x}_p^\dagger) \leq \mathbf{0} \\ \boldsymbol{\phi}_i(\boldsymbol{\xi}(t_0), \boldsymbol{\xi}(t_f), \mathbf{x}_c, \mathbf{x}_p^\dagger) \leq \mathbf{0} \end{array} \right\} = \mathbf{g}_i(\cdot) \leq \mathbf{0} \quad (3.3b)$$

$$= \mathbf{g}_i(\cdot) \leq \mathbf{0} \quad (3.3c)$$

$$= \mathbf{g}_i(\cdot) \leq \mathbf{0} \quad (3.3d)$$

where \mathbf{x}_p^\dagger is a candidate plant design, $\boldsymbol{\phi}_i$ are the constraints of $\boldsymbol{\phi}$ not in $\boldsymbol{\phi}_o$, and \mathbf{g}_i is the collection of the inner-loop constraints. With both formulations presented, some additional concepts relevant to two-level optimization problems need to be discussed.

The simultaneous method's feasible set (or constraint region) is defined as:

$$\Omega = \{(\mathbf{x}_p, \mathbf{x}_c) : \mathbf{g}_o(\mathbf{x}_p, \mathbf{x}_c) \leq \mathbf{0}, \mathbf{g}_i(\mathbf{x}_p, \mathbf{x}_c) \leq \mathbf{0}\} \quad (3.4)$$

For each candidate \mathbf{x}_p^\dagger , the inner-loop feasible set is defined by:

$$\Omega(\mathbf{x}_p^\dagger) = \{\mathbf{x}_c : \mathbf{g}_i(\mathbf{x}_p^\dagger, \mathbf{x}_c) \leq \mathbf{0}\} \quad (3.5)$$

and then set of potential inner-loop optimal control designs is:

$$M(\mathbf{x}_p^\dagger) = \{\mathbf{x}_c : \mathbf{x}_c \in \arg \min \{\Psi(\mathbf{x}_p^\dagger, \mathbf{x}_c) : \mathbf{x}_c \in \Omega(\mathbf{x}_p^\dagger)\}\} \quad (3.6)$$

For a given \mathbf{x}_p , $M(\mathbf{x}_p)$ may be empty for some values of its argument, i.e., no feasible control design exists for the given \mathbf{x}_p . If $\mathbf{x}_c \in M(\mathbf{x}_p)$, then the optimal objective function value from the inner loop is:

$$\psi(\mathbf{x}_p) = \Psi(\mathbf{x}_p, \mathbf{x}_c) \quad (3.7)$$

which is used as the objective function of the outer loop, Eqn. (3.2a). Finally, the feasible set of the outer loop (known as the induced region) is:

$$I = \{(\mathbf{x}_p, \mathbf{x}_c) : (\mathbf{x}_p, \mathbf{x}_c) \in \Omega, \mathbf{x}_c \in M(\mathbf{x}_p)\} \quad (3.8)$$

and we note that Ω and I are not the same feasible region. This is the motivation behind the addition of Eqn. (3.2c). $\mathbf{F}(\mathbf{x}_p)$ is an additional constraint that may be added to ensure for all \mathbf{x}_p^\dagger , $\Omega(\mathbf{x}_p^\dagger)$ is nonempty, termed the outer-loop feasibility constraint. The two formulations are only equivalent if Ω remains unchanged. This constraint will be discussed more in the following discussion section.

3.1.3 Formulation Discussion

Here we discuss the specific problem formulation elements of both approaches, focusing on comparisons to the current co-design literature.

3.1.3.1 Control Design Variables

While plant variables will always be time-independent, control variables may be either time-independent or infinite-dimensional trajectories (with respect to time). We partition the control variables into two sets as:

$$\boldsymbol{x}_c = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{u} \end{bmatrix} \quad (3.9)$$

where \boldsymbol{p} are the control parameters and \boldsymbol{u} are the open-loop control (OLC) variables. The control parameters are static (time-independent) control variables (e.g., gains). The OLC variables are infinite-dimensional trajectories through time (e.g., joint torque time profile for a robotic manipulator). Both types of control design variables are found in the literature, although both types are not typically in the same design problem. Even if there are no OLC variables, the general co-design problem is still an infinite-dimensional problem due to the dynamic and path constraints. One very special case is when the optimal OLC can be defined with static gains such as with infinite-horizon linear-quadratic regulator (LQR) [20, 101, 102] discussed later in Sec. 3.3.2.2. See Refs. [33, 82, 90, 91, 101, 102] for examples with \boldsymbol{p} and Refs. [18, 24, 27, 81, 94] for examples with \boldsymbol{u} .

3.1.3.2 Objective Function

Even though the objective function is partitioned in Eqns. (3.1a) and (3.1b), converting between the Lagrange and Mayer terms is possible [20, 25]. Both forms are present to allow for more natural formulations of the problem (and to be consistent with the literature).

A common form of the co-design objective function is:

$$\min_{\boldsymbol{x}_p, \boldsymbol{x}_c} \Psi = w_p \Psi_p(\boldsymbol{x}_p) + w_c \Psi_c(\boldsymbol{x}_p, \boldsymbol{x}_c) \quad (3.10)$$

where $\{\Psi_p, \Psi_c\}$ are the plant and control objective functions and $\{w_p, w_c\}$ are objective weights [83]. While many co-design problems are appropriately partitioned with control and plant specific objective function terms [33, 87–89, 91–93, 102], others necessitate the general

objective form in Eqns. (3.1a)–(3.1b) [4, 18, 24, 33, 81]. These general objectives may only include one term or terms that are typically only classified as “control” objective terms.

3.1.3.3 Path and Boundary Constraints

Equations (3.1d) and (3.1e) could be further generalized into a single general constraint form, but the distinction is important for both the optimality conditions and the numerical approaches used to find solutions. The difference is primarily based on time dependence. Therefore, path and boundary constraints could be more appropriately named time-dependent and time-independent constraints.

Boundary constraints are common to many existing co-design formulations [21, 25, 83]. Traditional plant-only constraints such geometry or mass constraints are time-independent constraints [27, 92, 93, 102]. Many co-design formulations explicitly require initial condition constraints, but some co-design problems have unknown initial conditions or periodic constraints [24]. Another set of boundary constraints is the kinematic relationships in robotics [3]. These constraints might require an algebraic variable that depends on the states (e.g., the end position of the robotic manipulator depends on the state joint angles and geometric plant variables).

More recently, general path constraints have been included in co-design problems and are infinite-dimensional constraints [4, 21, 25, 27]. Many traditional engineering constraints can be formulated naturally as path constraints. States or outputs often need to be constrained between allowable bounds such as temperature, position, force, pressure, deflection, stress, power, etc. These examples are typically inequality path constraints but equality path constraints are also possible, such as an automobile following a prescribed drive cycle or ensuring the steady-state optimal tip-speed ratio is followed by a horizontal axis wind turbine [81]. The inclusion of path constraints is critical to the usefulness of many co-design studies. Fathy et al. included mixed control-plant constraints:

$$\mathbf{C}(t, \mathbf{u}(t), \mathbf{x}_p) \leq \mathbf{0} \quad (3.11)$$

but mixed state-control-plant constraints as in Eqn. (3.1d) are needed to represent many of the engineering constraints mentioned above [83].

3.1.3.4 Outer-Loop Feasibility Constraint

For the two solution strategies to be equivalent, $M(\mathbf{x}_p)$ must be nonempty for every candidate \mathbf{x}_p [83]. Introducing an appropriate outer-loop feasibility constraint is one technique to ensure this property is satisfied.

Consider again the design of a robotic manipulator. For certain geometric configurations, the kinematics restrictions of the manipulator may be such that it is unable to reach the prescribed target. To avoid a candidate \mathbf{x}_p that has no feasible control solution, we could add a reachability constraint to $\mathbf{F}(\mathbf{x}_p)$ [3, 4]. However, even with this additional constraint, mixed state-control-plant constraints may be present that cannot be satisfied with the given plant design, such as stress or deflections [4]. This example helps illustrate that for general co-design problems, these constraints can be quite difficult (or impossible) to determine.

An important type feasibility constraint is the classic controllability property applicable to linear systems theory [83]. Unfortunately, controllability is not sufficient to guarantee the nonemptiness of $\Omega(\mathbf{x}_p^\dagger)$ for general co-design problems. For example, if control bounds are provided, then it is not guaranteed that we can move between arbitrary initial and final positions in finite time.

It is also important to note that this outer-loop feasibility constraint may not strictly be required to find the system-optimal solution using the simultaneous strategy. Another strategy is to add a feasibility constraint that restricts Ω such that the desired property holds. This is not ideal since there will be no guarantee that the nested strategy will produce the system-level optimum, but may be a necessity for certain co-design problems. The implications of this will be discussed in the following sections.

3.2 Necessary Conditions for Optimality

In this section, we describe the necessary conditions for optimality for the two general co-design strategies from the previous section. Sufficient conditions will not be discussed directly, but are the Hamiltonian minimization conditions [20, 103, 104] or second-order conditions on the Lagrangian for finite-dimensional problems [105]. If a problem is singular, there may be additional sufficient conditions needed [103, 104]. We also assume that the simultaneous co-design problem is well-posed, i.e., there exists a solution. See Refs. [20, 103, 105] for discussions on constraint qualification, regularity, differentiability, and other relevant properties for a well-posed problem. We begin with the conditions for the OLC design only.

3.2.1 Open-Loop Control Design

The optimal OLC design problem formulation is:

$$\min_{\mathbf{u}} \quad \Psi = \int_{t_0}^{t_f} \mathcal{L}(t, \boldsymbol{\xi}, \mathbf{u}) dt + \mathcal{M}(\boldsymbol{\xi}(t_0), \boldsymbol{\xi}(t_f)) \quad (3.12a)$$

$$\text{subject to: } \dot{\boldsymbol{\xi}} - \mathbf{f}(t, \boldsymbol{\xi}, \mathbf{u}) = \mathbf{0} \quad (3.12b)$$

$$\mathbf{C}(t, \boldsymbol{\xi}, \mathbf{u}) \leq \mathbf{0} \quad (3.12c)$$

$$\boldsymbol{\phi}(\boldsymbol{\xi}(t_0), \boldsymbol{\xi}(t_f)) \leq \mathbf{0} \quad (3.12d)$$

which is a subformulation of Prob. (3.1). Similar to the Lagrangian in finite-dimensional optimization [20, 105], the infinite-dimensional constraints can be adjoined to the Lagrange term with time-varying Lagrange multipliers, creating the Hamiltonian of the problem:

$$H = \mathcal{L} + \boldsymbol{\lambda}^\top \mathbf{f} + \boldsymbol{\mu}^\top \mathbf{C} \quad (3.13)$$

where $\boldsymbol{\lambda}(t)$ are the costates (multipliers for the state dynamics) and $\boldsymbol{\mu}(t)$ are the multipliers for \mathbf{C} .

With this control-only formulation, we can directly apply Pontryagin's minimum principle (PMP) with path constraints [20, 103, 106] to arrive at the following necessary conditions for optimality (note, \square^* is used to denote an optimal value):

$$\dot{\boldsymbol{\lambda}}^* = - \left[\frac{\partial H}{\partial \boldsymbol{\xi}} \right]^* \quad (3.14a)$$

$$\mathbf{0} = \left[\frac{\partial H}{\partial \mathbf{u}} \right]^* \quad (3.14b)$$

$$0 = [\boldsymbol{\mu}^\top \mathbf{C}]^*, \quad 0 = [\boldsymbol{\nu}^\top \boldsymbol{\phi}]^* \quad (3.14c)$$

$$\boldsymbol{\mu}^* \geq \mathbf{0}, \quad \boldsymbol{\nu}^* \geq \mathbf{0} \quad (3.14d)$$

$$\mathbf{0} = \left[\boldsymbol{\lambda} + \frac{\partial \mathcal{M}}{\partial \boldsymbol{\xi}} + \boldsymbol{\nu}^\top \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{\xi}} \right]_{t_0}^*, \quad \mathbf{0} = \left[\boldsymbol{\lambda} - \frac{\partial \mathcal{M}}{\partial \boldsymbol{\xi}} - \boldsymbol{\nu}^\top \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{\xi}} \right]_{t_f}^* \quad (3.14e)$$

where $\boldsymbol{\nu}$ are the Lagrange multipliers for $\boldsymbol{\phi}$, Eqn. (3.14a) is the costate dynamics, Eqn. (3.14b) is the control stationarity condition, Eqns. (3.14c) are the complementary slackness conditions, Eqns. (3.14d) are the dual feasibility conditions, and Eqns. (3.14e) are the initial and final time transversality conditions. The conditions in Eqn. (3.14) are in addition to the constraints in Eqns. (3.12b)–(3.12d). The necessary conditions for both strategies can now be derived.

3.2.2 Co-design—Simultaneous Strategy

Here we will derive the simultaneous co-design optimality conditions only using PMP. Others have derived the optimality conditions for similar co-design formulations. See Ref. [107] for early work outside the co-design context. Fathy et al. utilized a combination of the Karush-Kuhn-Tucker (KKT) conditions and PMP [83].

Consider the following augmented state vector:

$$\Theta = \begin{bmatrix} \xi \\ \mathbf{x}_p \end{bmatrix}, \quad \dot{\Theta} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix} \quad (3.15)$$

with the replacement of all plant variables with $\mathbf{x}_p(t_0)$ and, therefore, no dependence on $\mathbf{x}_p(t_f)$. Now consider if unconstrained, the choice of initial and final states are, in effect, additional decision variables that can be modified to satisfy the optimality conditions.

The ξ -dependent terms in Eqn. (3.14) are Eqns. (3.14a) and (3.14e). Applying these conditions to the “state” plant variables, we have the following additional conditions:

$$-\dot{\lambda}_p^* = \left[\frac{\partial H}{\partial \mathbf{x}_p} \right]^* = \left[\frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} + \boldsymbol{\lambda}^\top \frac{\partial \mathbf{f}}{\partial \mathbf{x}_p} + \boldsymbol{\lambda}_p^\top \cancel{\frac{\partial \dot{\mathbf{x}}_p}{\partial \mathbf{x}_p}}^0 + \boldsymbol{\mu}^\top \frac{\partial \mathbf{C}}{\partial \mathbf{x}_p} \right]^* \quad (3.16a)$$

$$\mathbf{0} = \left[\boldsymbol{\lambda}_p + \frac{\partial \mathcal{M}}{\partial \mathbf{x}_p} + \boldsymbol{\nu}^\top \frac{\partial \phi}{\partial \mathbf{x}_p} \right]_{t_0}^* \quad (3.16b)$$

$$\mathbf{0} = \left[\boldsymbol{\lambda}_p - \cancel{\frac{\partial \mathcal{M}}{\partial \mathbf{x}_p}}^0 - \boldsymbol{\nu}^\top \cancel{\frac{\partial \phi}{\partial \mathbf{x}_p}}^0 \right]_{t_f}^* = \boldsymbol{\lambda}_p^*(t_f) \quad (3.16c)$$

Now using the first fundamental theorem of calculus, we can combine these equations into a single condition:

$$\boldsymbol{\lambda}_p^*(t_f) = \boldsymbol{\lambda}_p^*(t_0) + \int_{t_0}^{t_f} \dot{\boldsymbol{\lambda}}_p^* dt \quad (3.17a)$$

$$\mathbf{0} = \left[\frac{\partial \mathcal{M}}{\partial \mathbf{x}_p} + \boldsymbol{\nu}^\top \frac{\partial \phi}{\partial \mathbf{x}_p} \right]_{t_0}^* + \int_{t_0}^{t_f} \left[\frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} + \boldsymbol{\lambda}^\top \frac{\partial \mathbf{f}}{\partial \mathbf{x}_p} + \boldsymbol{\mu}^\top \frac{\partial \mathbf{C}}{\partial \mathbf{x}_p} \right]^* dt \quad (3.17b)$$

This condition is analogous to the condition found in other derivations of the optimality conditions [83, 107]. Note that the other conditions in Eqn. (3.14) remain unchanged. Therefore, the simultaneous co-design optimality conditions are a combination of Eqn. (3.14)

and Eqn. (3.17b):

$$\dot{\boldsymbol{\lambda}}^* = - \left[\frac{\partial H}{\partial \boldsymbol{\xi}} \right]^*, \quad \mathbf{0} = \left[\frac{\partial H}{\partial \mathbf{u}} \right]^* \quad (3.18a)$$

$$0 = [\boldsymbol{\mu}^\top \mathbf{C}]^*, \quad 0 = [\boldsymbol{\nu}^\top \boldsymbol{\phi}]^*, \quad \boldsymbol{\mu}^* \geq \mathbf{0}, \quad \boldsymbol{\nu}^* \geq \mathbf{0} \quad (3.18b)$$

$$\mathbf{0} = \left[\boldsymbol{\lambda} + \frac{\partial \mathcal{M}}{\partial \boldsymbol{\xi}} + \boldsymbol{\nu}^\top \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{\xi}} \right]_{t_0}^*, \quad \mathbf{0} = \left[\boldsymbol{\lambda} - \frac{\partial \mathcal{M}}{\partial \boldsymbol{\xi}} - \boldsymbol{\nu}^\top \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{\xi}} \right]_{t_f}^* \quad (3.18c)$$

$$\mathbf{0} = \left[\frac{\partial \mathcal{M}}{\partial \mathbf{x}_p} + \boldsymbol{\nu}^\top \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}_p} \right]_{t_0}^* + \int_{t_0}^{t_f} \left[\frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} + \boldsymbol{\lambda}^\top \frac{\partial \mathbf{f}}{\partial \mathbf{x}_p} + \boldsymbol{\mu}^\top \frac{\partial \mathbf{C}}{\partial \mathbf{x}_p} \right]^* dt \quad (3.18d)$$

3.2.3 Co-design—Nested Strategy

We now derive the necessary conditions for the nested strategy where the outer-loop problem is defined in Prob. (3.2) and the inner loop in Prob. (3.3). The inner-loop optimality conditions are defined in Eqn. (3.14) with only OLC variables. If the inner loop contains \mathbf{p} , then the necessary conditions are analogous to the simultaneous conditions in Eqn. (3.18) with \mathbf{p} replacing \mathbf{x}_p .

The outer-loop problem is finite-dimensional optimization since neither the dynamics nor path constraints are present. Therefore, we can directly utilize the KKT conditions [105]. The Lagrangian for the outer-loop problem is:

$$L = \psi + \boldsymbol{\chi}^\top \boldsymbol{\phi}_o + \boldsymbol{\eta}^\top \mathbf{F} \quad (3.19)$$

We first express the stationarity condition:

$$\mathbf{0} = \frac{dL}{d\mathbf{x}_p} = \frac{d\psi}{d\mathbf{x}_p} + \boldsymbol{\chi}^\top \frac{d\boldsymbol{\phi}_o}{d\mathbf{x}_p} + \boldsymbol{\eta}^\top \frac{d\mathbf{F}}{d\mathbf{x}_p} \quad (3.20a)$$

$$= \int_{t_0}^{t_f} \frac{d\mathcal{L}}{d\mathbf{x}_p} dt + \frac{d\mathcal{M}}{d\mathbf{x}_p} + \boldsymbol{\chi}^\top \frac{d\boldsymbol{\phi}_o}{d\mathbf{x}_p} + \boldsymbol{\eta}^\top \frac{d\mathbf{F}}{d\mathbf{x}_p} \quad (3.20b)$$

Now the total derivative terms can be explicated [108]:

$$\frac{d\mathcal{L}}{d\mathbf{x}_p} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}^*} \frac{d\boldsymbol{\xi}^*}{d\mathbf{x}_p} + \frac{\partial \mathcal{L}}{\partial \mathbf{u}^*} \frac{d\mathbf{u}^*}{d\mathbf{x}_p} + \frac{\partial \mathcal{L}}{\partial \mathbf{p}^*} \frac{d\mathbf{p}^*}{d\mathbf{x}_p} \quad (3.21a)$$

$$\frac{d\mathcal{M}}{d\mathbf{x}_p} = \frac{\partial \mathcal{M}}{\partial \mathbf{x}_p} + \left[\frac{\partial \mathcal{M}}{\partial \boldsymbol{\xi}^*} \frac{d\boldsymbol{\xi}^*}{d\mathbf{x}_p} \right]_{t_0} + \left[\frac{\partial \mathcal{M}}{\partial \boldsymbol{\xi}^*} \frac{d\boldsymbol{\xi}^*}{d\mathbf{x}_p} \right]_{t_f} + \frac{\partial \mathcal{M}}{\partial \mathbf{p}^*} \frac{d\mathbf{p}^*}{d\mathbf{x}_p} \quad (3.21b)$$

Here we see a number of coupled terms.

The complete set of necessary conditions for the outer loop are then:

$$\mathbf{0} = \frac{d\Psi}{d\mathbf{x}_p} + \boldsymbol{\chi}^\top \frac{d\phi_o}{d\mathbf{x}_p} + \boldsymbol{\eta}^\top \frac{dF}{d\mathbf{x}_p} \quad (3.22a)$$

$$\boldsymbol{\chi} \geq \mathbf{0}, \quad \boldsymbol{\eta} \geq \mathbf{0}, \quad \boldsymbol{\chi}^\top \boldsymbol{\phi}_o = \mathbf{0}, \quad \boldsymbol{\eta}^\top \mathbf{F} = \mathbf{0} \quad (3.22b)$$

which are a set of KKT conditions. The conditions in Eqn. (3.22) are in addition to the constraints Eqns. (3.2b) and (3.2c).

3.2.4 Free Time

The initial and final time values can be optimization variables as well [20, 25, 83, 103]. Optimal control problems frequently contain these terms and therefore, appropriate optimality conditions are provided. Subsequently, these additional optimization variables are typically included in the inner-loop problem of the nested co-design strategy [83]. However, some of the numerical methods used jointly with certain co-design strategies lose certain properties when the time horizon is not fixed. In these situations, perhaps, it is more beneficial to place these additional variables in the outer loop to maintain such properties.

The simultaneous co-design strategy will need one additional equation if t_f is free:

$$0 = \left[\frac{\partial \mathcal{M}}{\partial t_f} + \boldsymbol{\nu}^\top \frac{\partial \boldsymbol{\phi}}{\partial t_f} \right]^* + \left[\mathcal{L} + \boldsymbol{\lambda}^\top \mathbf{f} + \boldsymbol{\mu}^\top \mathbf{C} \right]_{t_f}^* \quad (3.23)$$

This is an additional transversality condition with the others in Eqn. (3.14e) [20, 103].

For a nested approach with t_f free in the outer loop, the additional stationary condition $dL/dt_f = 0$ would include:

$$\frac{d\mathcal{L}}{dt_f} = \frac{\partial \mathcal{L}}{\partial t_f} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}^*} \frac{d\boldsymbol{\xi}^*}{dt_f} + \frac{\partial \mathcal{L}}{\partial \mathbf{u}^*} \frac{d\mathbf{u}^*}{dt_f} + \frac{\partial \mathcal{L}}{\partial \mathbf{p}^*} \frac{d\mathbf{p}^*}{dt_f} \quad (3.24a)$$

$$\frac{d\mathcal{M}}{dt_f} = \frac{\partial \mathcal{M}}{\partial t_f} + \left[\frac{\partial \mathcal{M}}{\partial \boldsymbol{\xi}^*} \frac{d\boldsymbol{\xi}^*}{dt_f} \right]_{t_0} + \left[\frac{\partial \mathcal{M}}{\partial \boldsymbol{\xi}^*} \frac{d\boldsymbol{\xi}^*}{dt_f} \right]_{t_f} + \frac{\partial \mathcal{M}}{\partial \mathbf{p}^*} \frac{d\mathbf{p}^*}{dt_f} \quad (3.24b)$$

This completes the necessary conditions for optimality for the various problem formulations and solution strategies. Now some of the computational and practical aspects will be discussed.

3.3 Practical Solution Considerations

Obtaining solutions only using the optimality conditions in the previous section can be challenging. Practical formulation limitations, robustness, and computational efficiency also need to be considered when seeking solutions to co-design problems.

3.3.1 Obtaining Approximate Solutions

In the previous section, the finding an analytic solution that satisfies the optimality conditions is called an indirect method [95, 96]. Although this approach can lead to important insights into the structure of the solution, it can be quite challenging (or impossible) to solve these equations analytically. If complicated black box functions or table interpolation is used, analytic forms for their derivatives do not even exist. Therefore, numerical methods are often employed that provide approximate solutions to the original problem.

Numeric indirect methods derive explicitly the optimality conditions and then use discretization to form a boundary value problem (BVP) [96]. If inequality path constraints are present, multiple BVPs will need to be linked for each arc that a path constraint is active since a new set of differential algebraic equations will need to be solved. The number of constrained subarcs and the sequence of constrained/unconstrained arcs, however, are unknown a priori, so it is quite difficult or even impossible to construct the correct BVP.

Another issue with numeric indirect methods is standard solution procedures are not robust. An initial guess for the costates must be given, but these are not physical quantities, so there typically is no good way of providing a reasonable estimate [95]. Furthermore, even with a reasonable guess, the numerical solution of the costate dynamics in Eqn. (3.14a) can be very ill-conditioned [104].

An alternative is direct methods [95, 96]. Instead of stating the optimality conditions, the control and/or state are parametrized using function approximation and the objective function is approximated using numerical quadrature. This creates a discrete, finite-dimensional problem that then is optimized using large-scale NLP solvers [96].

The first class of direct methods is the sequential methods, which only parametrize the control. Given initial conditions and a set of control parameters, the differential-algebraic equation (DAE) model is solved through conventional DAE solvers (forward simulation) such as a Runge-Kutta method. Due to the use of conventional DAE solvers, this approach has the advantage of easily finding feasible solutions to the state equations, but needs to perform a full simulation for each perturbation in the optimization algorithm. Repeated

numerical integration of the DAE model, however, does not guarantee convergence with open-loop unstable systems [96] and the resulting solution can be very sensitive to the choice of control. This strategy is the easiest to construct out of all the direct methods since reliable and efficient codes for DAE and NLP solvers are naturally linked [96].

Sequential approaches typically produce low-accuracy solutions and are computationally inefficient. Many of these issues are due to the inability to handle path and boundary constraints efficiently. Since the states are calculated through a forward simulation, we must approximate numerically how local control perturbations will affect the global state trajectory. This leads to simultaneous approaches that forgo the nested analysis of sequential methods for a large set of constraints⁸.

3.3.1.1 Direct Transcription

The simultaneous approach, also known as direct transcription (DT), parametrizes both the state and control trajectories. Direct transcription is the focus in Chapter 5 so it is only discussed briefly in the context of its usefulness in solving co-design problems. Most DT methods are represented by the following NLP formulation:

$$\min_{\boldsymbol{U}, \boldsymbol{\Xi}, \boldsymbol{x}_p} \sum_{k=0}^{N_t} w_k \mathcal{L}(t_k, \boldsymbol{\xi}(t_k), \boldsymbol{u}(t_k), \boldsymbol{x}_p) + \mathcal{M}(\boldsymbol{\xi}(t_0), \boldsymbol{\xi}(t_f), \boldsymbol{x}_p) \quad (3.25a)$$

$$\text{subject to: } \boldsymbol{\zeta}(\boldsymbol{t}, \boldsymbol{\Xi}, \boldsymbol{U}, \boldsymbol{x}_p) = \mathbf{0} \quad (3.25b)$$

$$\boldsymbol{C}(\boldsymbol{t}, \boldsymbol{\Xi}, \boldsymbol{U}, \boldsymbol{x}_p) \leq \mathbf{0} \quad (3.25c)$$

$$\boldsymbol{\phi}(\boldsymbol{\xi}(t_0), \boldsymbol{\xi}(t_f), \boldsymbol{x}_p) \leq \mathbf{0} \quad (3.25d)$$

where $N_t + 1$ is the number of discrete time points and $\{\boldsymbol{t}, \boldsymbol{\Xi}, \boldsymbol{U}\}$ are the discretized forms of the time, states, and controls. The Lagrange term is approximated with numerical quadrature with weights w_k . The dynamic constraint is now enforced through a large number of equality constraints $\boldsymbol{\zeta}$, termed defect constraints [95–97]. Also, path constraints are now no more complicated than the dynamic constraints, a key advantage over other solution methods [96, 109].

This new large NLP formulation has a specific structure and sparsity pattern that can be exploited in NLP solvers to reduce total computational effort [96]. Simultaneous approaches

⁸The sequential and simultaneous methods in this section are entirely different than the sequential design method mentioned at the beginning of this chapter and the simultaneous co-design method [21, 27, 96]. The methods in this section find approximate solutions to DO problems. The design methods are alternative solution methods for problems with plant and control design variables. For example, one could pose a simultaneous co-design problem and solve it with a sequential method for dynamic optimization.

have been shown to have good convergence properties and handle unstable DAEs [110, 111]. Finally, these approaches have specific advantages for singular control problems and high-index path constraints [110]. This collection of desirable properties makes DT a strong candidate for finding solutions to general co-design problems.

General overviews of DT theory is available in Refs. [25, 27, 95–97, 110]. Please see Refs. [21, 25, 27] for further discussion on using DT to find solutions to co-design problems. In these co-design studies, DT was demonstrated to be a suitable method for finding solutions to more general co-design problems.

3.3.2 Specific Forms

In this chapter, we have presented the most general form of the co-design problem: both a nonlinear plant outer-loop and nonlinear control inner-loop. However, specific forms of the control subproblem have been utilized heavily in a variety of co-design studies. The primary motivation for these forms is the efficient generation of solutions for the inner-loop subproblem. These two forms are linear-quadratic dynamic optimization and LQR.

3.3.2.1 Linear-Quadratic Dynamic Optimization

Consider the following specific form of the control subproblem in Prob. (3.12):

$$\min_{\boldsymbol{u}} \quad \int_{t_0}^{t_f} \left(\begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{u} \end{bmatrix}^\top \begin{bmatrix} \mathbf{Q} & \mathbf{N} \\ \mathbf{N} & \mathbf{R} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{u} \end{bmatrix} + \begin{bmatrix} \mathbf{q} \\ \mathbf{r} \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{u} \end{bmatrix} \right) dt + \dots \quad (3.26a)$$

$$[\boldsymbol{\xi}^\top \mathbf{M} \boldsymbol{\xi} + \mathbf{m}^\top \boldsymbol{\xi}]_{t_0} + [\boldsymbol{\xi}^\top \mathbf{S} \boldsymbol{\xi} + \mathbf{s}^\top \boldsymbol{\xi}]_{t_f} \quad (3.26b)$$

$$\text{subject to: } \dot{\boldsymbol{\xi}} - (\mathbf{A}\boldsymbol{\xi} + \mathbf{B}\boldsymbol{u} + \mathbf{d}) = \mathbf{0} \quad (3.26c)$$

$$\mathbf{C}_1 \boldsymbol{\xi} + \mathbf{C}_2 \boldsymbol{u} - \mathbf{C}_3 \leq \mathbf{0} \quad (3.26d)$$

$$\boldsymbol{\phi}_1 \boldsymbol{\xi}(t_0) + \boldsymbol{\phi}_2 \boldsymbol{\xi}(t_f) - \boldsymbol{\phi}_3 \leq \mathbf{0} \quad (3.26e)$$

where t_f is finite and any of the matrices (with the exception of $\{\boldsymbol{\phi}, \mathbf{M}, \mathbf{m}, \mathbf{S}, \mathbf{s}\}$) can be time-varying. This particular linear-quadratic DO problem can be approximated as a finite-dimensional quadratic program (QP) using a DT method⁹. Both single-step and pseudospectral DT methods can generate sparse QPs for this problem form [25, 96]. Under

⁹A more general form of Prob. (3.26) is present in Chapter 5 along with the methods for generating the QP using DT.

certain conditions, the QP is convex and efficient to solve for even large systems and the global optimal solution for \mathbf{u} is guaranteed.

This form has been utilized in a number of nested co-design studies [18, 25]. The simultaneous strategy cannot be applied with this form since most (if not all) co-design dynamic constraints have a nonlinear dependence on the optimization variables [83]. For example consider the following bilinear dynamic constraint: $\dot{\xi} = -k\xi$, where k is a plant optimization variable. This dynamic constraint would need to be approximated with quadratic constraints.

3.3.2.2 Linear-Quadratic Regulator

Consider the following special form of the control-only subproblem: Prob. (3.26) containing only time-invariant matrices $\{\mathbf{Q} \succeq 0, \mathbf{R} \succ 0, \mathbf{A}, \mathbf{B}\}$, no path constraints, simple initial value constraints ($\boldsymbol{\xi}(t_0) = \boldsymbol{\xi}_0$), and is infinite-horizon ($t_f \rightarrow \infty$). The optimal OLC can then be readily computed with the solution of the algebraic Riccati equation (ARE):

$$\mathbf{P}\mathbf{A} + \mathbf{A}^\top\mathbf{P} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top\mathbf{P} + \mathbf{Q} = 0 \quad (3.27)$$

where \mathbf{P} is the unique, positive definite solution and \mathbf{u} is:

$$\mathbf{u}^* = -\mathbf{R}^{-1}\mathbf{B}^\top\mathbf{P}^*\boldsymbol{\xi}^* \quad (3.28)$$

This problem form is known as the infinite-horizon, continuous-time LQR [20, 101, 102], and is quite heavily utilized in co-design studies [33, 90, 91, 101, 102]. Solutions of this form can be computed efficiently and have a number of favorable properties such as a closed-form solution that can be used to create an equivalent simultaneous formulation [83]. There are additional forms (e.g., finite-time and time-varying versions) of this type of problem [20, 104] but the computation cost quickly becomes comparable with highly structured QPs using DT solution methods. However, this problem structure is simply unsatisfactory in addressing the requirements for some co-design problems (see Sec. 3.1.3).

3.3.3 Comparing the Strategies

A natural question is what solution technique should be used? This section will discuss some of the research around this question.

3.3.3.1 Computation Time

Here we parameterize the solution time for each method with:

$$\tau_s = T_0 + NT_i \quad (\text{simultaneous}) \quad (3.29\text{a})$$

$$\tau_n = T_0 + \bar{N} (\bar{T}_i + t_0 + nt_i) \quad (\text{nested}) \quad (3.29\text{b})$$

where N is the number of optimization algorithm iterations needed for convergence, T is a time variable, n is the number of inner-loop iterations, t is inner-loop time, $\bar{\square}$ is an outer-loop quantity, \square_0 is the initial time used to construct the optimization problem, and \square_i is the average iteration time.

Obviously, if $\bar{N} > N$ and $\bar{T}_i > T_i$, then the simultaneous approach is superior, but this is unlikely as the nested strategy reduces the number of variables and constraints in the outer-loop problem. This does highlight the computational goal of the nested approach. Ideally we seek a nested strategy that significantly reduces both the number of iterations and time required during each iteration in the outer-loop, while keeping the time required to solve the nested subproblem small.

Methods for reducing total computational expense for the general co-design problem with respect to either solution strategy are limited (other than favoring DT over other methods). One suggestion from Ref. [112] is to treat plant variables in the simultaneous approach with DT exactly as in Eqn. (3.15), where there is a plant variable at each time point and they are all constrained to be equal (which keeps Jacobian of the constraints and objective function sparse). The development of guidelines and methods to help address the computational cost for different classes of co-design problems is left as future work.

3.3.3.2 Literature-Based Recommendations

Reyer et al. stated that the size and complexity of the simultaneous problem formulation could make this strategy impractical [88]. A number of authors have agreed with this statement, i.e., the nested strategy was better suited for their particular problem [18, 81, 101, 113]. In the case of Refs. [101, 113], the studies utilize the LQR for an extremely efficient inner loop, i.e., $t_0 + nt_i$ was quite small. In Ref. [81], there was an extreme computational expense associated with changes in the plant variables (T_i was large), hence minimizing the number of function calls was paramount. In Ref. [18], there was both a high computational expense for modifying the plant design and the inner-loop could be solved with a QP.

There are a limited set of studies that claim the opposite. In Ref. [27], the authors

state that the simultaneous strategy is better, but the inner-loop was not solved with a QP even though it was in the Prob. (3.26) class, so it is hard to fairly compare the strategies. However, it is still hard to make general recommendations since it is not very common for co-design study to compare the tradeoffs between the simultaneous and nested strategies for their particular design problem. In most design studies, only one strategy is needed if it is producing the desired results.

There are some circumstances where the simultaneous strategy might be preferred. For some problems, it might be the only viable method. For example, it may be impossible to separate the domains if black-boxes are used. Also, consider the discussion of an empty Ω for a particular candidate plant design. In this situation, the nested strategy could struggle or fail to find the optimal solution. Another point is that the nested strategy does not follow the steepest descent direction in the simultaneous formulation (since the control variables are optimal for the control subproblem). A simultaneous approach may better utilize the coupling between the plant and control design variables for faster convergence.

It remains future work to determine if there are conditions such that a well-behaved simultaneous co-design formulation becomes more challenging to solve with the nested strategy.

3.4 Test Problems

In this section, three co-design test problems (abbreviated TP) are defined which highlight some of the general co-design theory concepts¹⁰. Please see also Refs. [4, 18, 24, 27, 33, 81, 82, 94] for examples of complex co-design problems that utilize nontraditional problem elements in the general co-design problem class.

3.4.1 Test Problem 1: Scalar Plant, Scalar Control

The first TP belongs to many early co-design formulations [83, 91] including infinite time horizon, optimization of control gains, no path constraints, and separate control and plant

¹⁰The MATLAB codes for these test problems is available in Ref. [114].

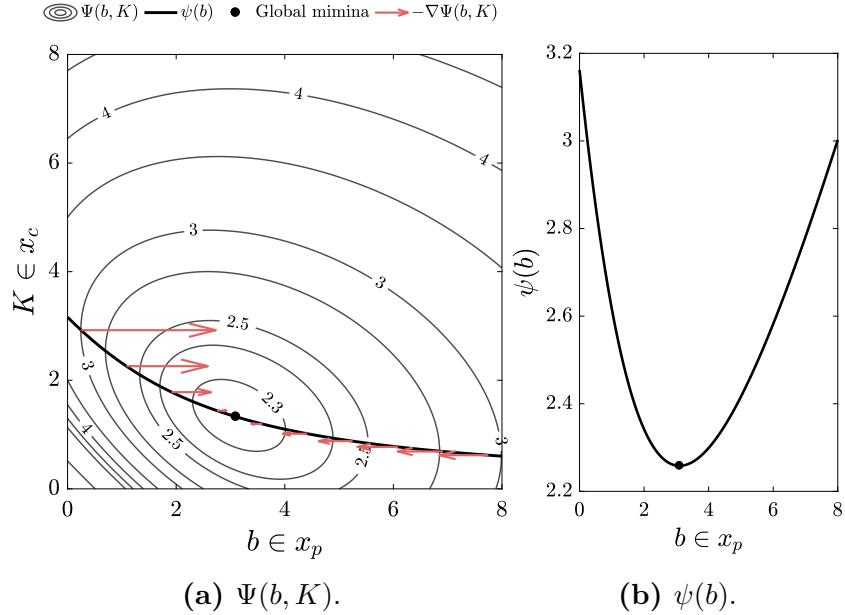


Figure 3.2: Scalar plant, scalar control problem (TP1) results with $q = 10$, $r = 1$, $w_c = 1$, and $w_p = 0.3$.

objectives as in Eqn. (3.10). The co-design problem is stated as:

$$\min_{b, K} \frac{w_c}{\xi_0^2} \int_0^\infty (q\xi^2 + ru^2) dt + w_p b \quad (3.30a)$$

$$\text{subject to: } \dot{\xi} = -b\xi + u \quad (3.30b)$$

$$\phi_1 := \xi(0) - \xi_0 = 0 \quad (3.30c)$$

$$\phi_2 := -b \leq 0, \phi_3 := -K \leq 0 \quad (3.30d)$$

where $u = -K\xi$, $b \in \mathbf{x}_p$, and $K \in \mathbf{x}_c$. The results are shown in Fig. 3.2 for one particular set of values of the problem parameters.

For this problem, the ARE defined in Eqn. (3.27) is a single quadratic equation: $0 = q - 2bP - P^2/r$. The solution to this equation defines the gains of a full-state feedback optimal control law and therefore can be used to obtain the potential inner-loop optimal control designs, $M(\mathbf{x}_p)$, in a nested formulation. This is visualized in Fig. 3.2a as the solid black curve. We note that this is a subspace of Ω , or simultaneous feasible set. Comparing Fig. 3.2a and Fig. 3.2b, we can visualize directly the difference between $\Psi(\mathbf{x}_p, \mathbf{x}_c)$ and $\psi(\mathbf{x}_p)$. We also note that the gradient of the nested solution trajectory in Fig. 3.2a is always orthogonal to the K -axis since it is the optimal gain value for a given b .

The coupling between the plant and control design variables is also present, noting the

“tilt” of the level sets. Therefore, a sequential method would not be guaranteed to arrive at the global optima (but an iterative sequential strategy would in TP1) [83]. In TP1, from all starting conditions, both the simultaneous and nested solution strategies arrive at the global optimum. Although this is an extremely simple co-design problem, it does help visualize a number of important concepts.

3.4.2 Test Problem 2: Co-Design Transfer

Consider the following simple co-design problem that seeks to move a second-order system from an arbitrary initial state to rest while minimizing control effort:

$$\min_{k,u(t)} \int_0^{t_f} u^2 dt \quad (3.31a)$$

$$\text{subject to: } \dot{\xi} = \begin{bmatrix} 0 & 1 \\ -k & 0 \end{bmatrix} \xi + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \quad (3.31b)$$

$$\phi_1 := \xi_1(0) - x_0 = 0, \quad \phi_2 := \xi_2(0) - v_0 = 0 \quad (3.31c)$$

$$\phi_3 := \xi_1(t_f) = 0, \quad \phi_4 := \xi_2(t_f) = 0 \quad (3.31d)$$

where $k \in \mathbf{x}_p$ and $u(t) \in \mathbf{x}_c$. The solution for $u^*(t, k)$ (i.e., the nested strategy) can be obtained by scaling the problem [115] into an equivalent DO problem in Ref. [104, pp. 166–167] (details of this procedure are in Sec. 4.3.2):

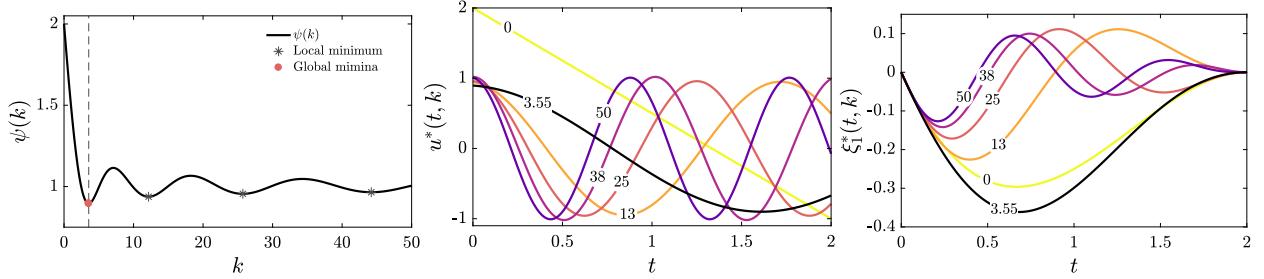
$$u^*(t, k) = -\frac{2k}{kt_f^2 - \sin^2(\sqrt{k}t_f)} \left(c_1(t, k)x_0 + c_2(t, k) \frac{v_0}{\sqrt{k}} \right) \quad (3.32a)$$

$$c_1(t, k) = \sin(\sqrt{k}(t_f - t)) \sin(\sqrt{k}t_f) - \sqrt{k}t_f \sin(\sqrt{k}t) \quad (3.32b)$$

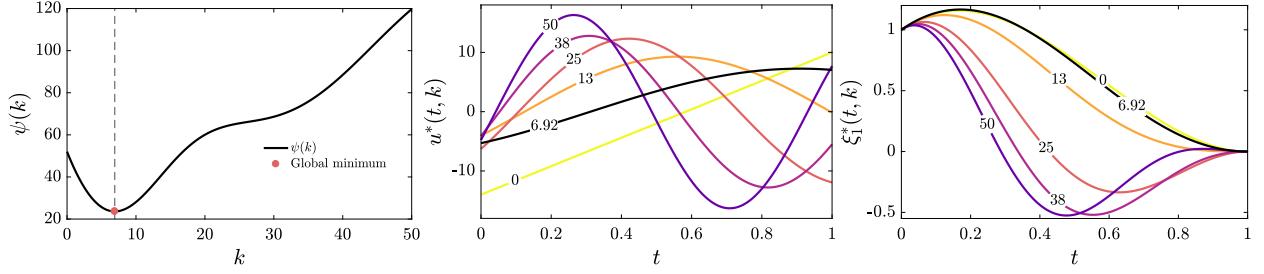
$$c_2(t, k) = -\cos(\sqrt{k}(t_f - t)) \sin(\sqrt{k}t_f) + \sqrt{k}t_f \cos(\sqrt{k}t) \quad (3.32c)$$

If $k = 0$, then the solution above is not valid and u^* is linear with respect to t . The original objective function can now be computed analytically by integrating the square of Eqn. (3.32). With this closed-form expression for the objective function, an optimality condition for k can be derived using Eqn. (3.20).

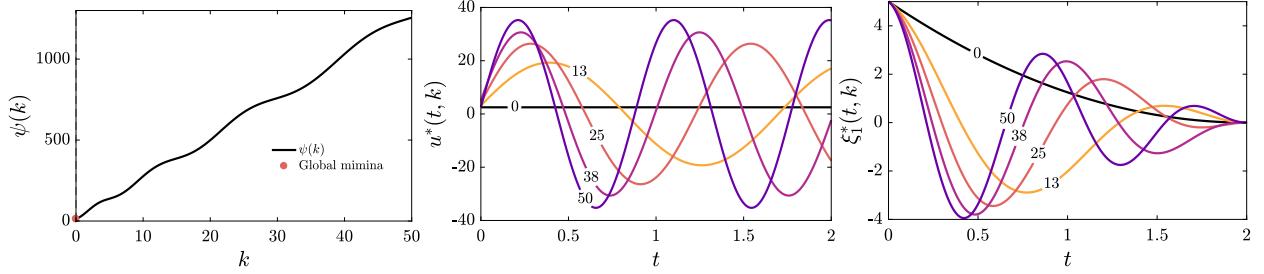
The results for this TP are shown in Fig. 3.3 for various values of the problem parameters. In Fig. 3.3a, there are a large number of local solutions of $\psi(k)$ and a clear global minimum at $k^* = 3.554$. In Fig. 3.3b, for the different values of the problem parameters, there is a single global minimum at $k^* = 6.924$. Finally in Fig. 3.3c, there is a global minimum with $k^* = 0$, i.e., the plant solution is degenerate.



(a) TP2 results demonstrating a large number of local solutions with $t_f = 2$, $x_0 = 0$, and $v_0 = -1$ (different values of k marked).



(b) TP2 results demonstrating single global minimum with $t_f = 1$, $x_0 = 1$, and $v_0 = 2$ (different values of k marked).



(c) TP2 results demonstrating degenerate plant solution with $t_f = 2$, $x_0 = 5$, and $v_0 = -5$ (different values of k marked).

Figure 3.3: Co-design transfer problem (TP2) results for various values of the problem parameters.

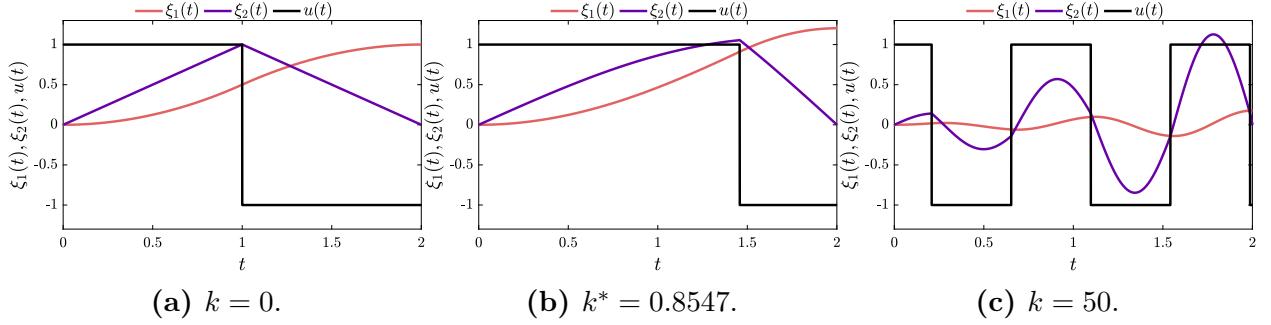


Figure 3.4: Simple SASA test problem (TP3) results with $J = 1$, $t_f = 2$, and $u_{\max} = 1$.

From these results, it is clear that caution should be used when claiming a global optimal co-design solution is found. Global search algorithms could improve the confidence of finding the true optimal solution such as a multistart approach or genetic algorithms [105]. The control solution in Eqn. (3.32) demonstrates the complicated nature of the analytical solutions for even the simplest of co-design problems. This problem may be a particularly useful TP as it is a well-posed co-design problem with a single system-level objective, general boundary conditions, and a closed-form OLC solution.

3.4.3 Test Problem 3: Simple SASA

The final TP is a co-design problem that was used directly in a detailed co-design study. The co-design study focused on developing a novel strain-actuated solar array (SASA) system for spacecraft pointing control and jitter reduction [18] (please see Sec. 4.3.1 for a detailed discussion of the connections to the detailed study in Chapter 7). Both the geometric properties of the solar array and OLC voltages along the array were the design variables. To help provide some insight into the results of the original design study, a much simpler, but still representative, co-design problem was proposed:

$$\min_{k,u(t)} \quad -\xi_1(t_f) \tag{3.33a}$$

$$\text{subject to: } \dot{\boldsymbol{\xi}} = \begin{bmatrix} 0 & 1 \\ -k/J & 0 \end{bmatrix} \boldsymbol{\xi} + \begin{bmatrix} 0 \\ 1/J \end{bmatrix} u \tag{3.33b}$$

$$\phi_1 := \xi_1(0) = 0, \quad \phi_2 := \xi_2(0) = 0 \tag{3.33c}$$

$$\phi_3 := \xi_2(t_f) = 0 \tag{3.33d}$$

$$C_1 := u - u_{\max} \leq 0, \quad C_2 := -u - u_{\max} \leq 0 \tag{3.33e}$$

where $k \in \mathbf{x}_p$ and $u(t) \in \mathbf{x}_c$. The results for this problem are shown in Fig. 3.4 for various values of k including $k^* = 0.8547$.

The OLC exhibits bang-bang behavior [20, 104], i.e., u is at either the minimum or maximum value. Using the control stationarity condition in Eqn. (3.14b), we can show directly that the Hamiltonian is minimized if u exhibits bang-bang behavior, but determining the locations of the switching is quite challenging. In a nested strategy, the number and location of the switches vary for different values of k (see Fig. 3.4c containing five switches while the optimal co-design solution only contains one switch in Fig. 3.4b). This is a direct example of the discussion in Sec. 3.3.1 motivating the using of DT in co-design (see Sec. 3.3.1.1). The nested co-design solutions were found using DT formulated as a QP with both the composite quadrature and defect constraints based on the trapezoidal rule [25, 95, 96].

TP3 is an attractive TP as the closed-form solution for the simultaneous strategy can be computed to a high degree of accuracy, and it exhibits challenging behavior associated with inequality path constraints.

3.5 Summary

In this chapter, general combined plant and controller design (or co-design) problems were examined. A large portion of existing co-design theory has focused on specific DO formulations, leaving many open questions for co-design studies that do not fit the previous definitions.

There are two basic co-design solution strategies: simultaneous and nested. The problem formulations for both strategies were presented and a discussion of the formulation elements was provided. The nested strategy was presented as two-level optimization problem including a characterization of the differences in the feasibility region between the two strategies. This motivated the outer-loop feasibility constraint as one approach for ensuring that for every candidate plant design, the control subproblem is well-posed.

The natural next step was the presentation of the optimality conditions for both methods. For the simultaneous strategy, the optimality conditions were derived using only Pontryagin's minimum principle and an augmented state vector. Due to a number of challenges associated with the optimality conditions, practical solution considerations were discussed with a focus the motivating reasons for using DT in co-design (e.g., the activity of inequality path constraints). Finally, three test problems were presented. These problems were fairly simple but highlighted a number of key concepts including coupling, the difference

between the feasible regions for each strategy, general boundary conditions, inequality path constraints, system-level objectives, and complexity of the closed-form solutions.

This chapter seeks to provide a foundation for additional advances in general co-design theory. The outer-loop feasibility constraint warrants further investigation. Better comparisons between the two strategies are needed, especially for co-design problems without a specific form of the inner loop. Developing methods for reducing total computational expense is also an important area such as the suggestion in Ref. [112] for treating plant variables like dummy states (or control) or alternative general strategies such decentralized optimization [85, 116]. Finally, development of test problems that are more realistic could help answer some of the open questions.

Chapter 4

Scaling of Dynamic Optimization Formulations¹¹

*“Behind complexity, there is always simplicity to be revealed.
Inside simplicity, there is always complexity to be discovered.”*

G. Yu [117]

4.1 Introduction

Dynamics play an increasingly important role in the advancement of many complex engineering systems [21]. The primary goal of design studies is to find solutions and gain a general understanding of the design trade-offs, building design knowledge for the particular system. Here we show how scaling can facilitate finding accurate, generalizable, and intuitive information for the design problem at hand.

At a basic level, scaling is simply the stretching, squeezing, and shifting of the problem elements and this can include the time continuum, design variables, constraints, objective function, etc. For example, if we have the inequality constraint $ax \leq b$ and $b > 0$, then we can arrive at an equivalent scaled constraint $\rho x \leq 1$ where $\rho = a/b$. The mechanics of scaling are fairly straightforward but proper utilization of scaling is heavily reliant on the creativity and intuition of the designer [118]. This barrier may be one of the reasons why scaling is often overlooked, but these manipulations can help define problem formulations that are 1) better suited for analysis and 2) more favorable for solution methods (e.g., higher quality solutions and faster convergence). In this chapter, we provide the necessary theory to scale dynamic optimization (DO) problems and some examples of how to use scaling in the context of a design study.

First, we review some of the previous uses of scaling focusing on examples relevant to obtaining solutions and the analysis of engineering design problems. Some of these examples

¹¹Elements of this chapter are based on work completed in Ref. [115].

are quite well-established, while others are rarely used. The context that all the examples provide is crucial for defining the most useful scaling procedure for a particular design problem. Some authors state that the importance of scaling can only be fully appreciated through examples [119].

4.1.1 Previous Uses of Scaling

One of the primary uses of scaling is to reduce the number of parameters in a set of equations [118–120]. Under certain conditions, algebraic manipulations can lead to a reduced set of parameters (e.g., consider the example above where now ρ is the only parameter). Buckingham's Pi theorem is one well-known method for reducing the number of parameters, which applies constraints on the mathematical interaction of the fundamental units of the system and does not necessarily need a specific mathematical expression of the system (e.g., the equations of motion) [118, 119, 121]. However, this approach does not necessarily leverage the equations of motion nor does it directly consider the numerical aspects of finding approximate solutions to the system of equations.

Another common use for scaling is to determine characteristic properties of the system. These properties may be scalars or functions. Characteristic scalars are well studied in many engineering domains such as fluid dynamics and heat transfer [120]. An example of such a scalar is the Reynolds number, a dimensionless constant that relates inertial forces to viscous forces within a fluid [120]. Other examples include intrinsic resonance frequency, length, damping, or time constant. Such characteristic properties can be conceptually easier to understand [118]. These properties may be well understood for classical domains but for others, these characteristic properties may be the key to building the required knowledge of the system.

In dynamic or spatially-defined systems, it can be advantageous to scale entire continuous functions. For example, for an initial value problem, characteristic solutions to the differential equation can be obtained that scale linearly with the initial condition [118]. Under certain conditions, even the solution to optimization problems can be scaled for different parameter values. There are some potentially restrictive issues with directly scaling optimal solutions that are discussed in Ref. [122].

The central tool in many design studies is design optimization. Frequently, the solution of a single optimization problem is not sufficient to address the complicated nature of a design activity. This typically leads to variations on the problem formulation elements (e.g., parametric sweeps of the problem's parameters). Both reducing the number of parameters and

obtaining scalable solutions can greatly improve this process.

Scaling can also be used to help decide if certain parts of a model or optimization formulation are small, and therefore negligible in a consistent manner [118, 123]. This can help with developing asymptotic solutions to differential equations, reveal multiple-time-scale structures, and simplify simulations, stability analysis, and controller design [123]. With respect to optimization formulations, scaled forms of the constraints can help determine if certain constraints are likely to be active or inactive based on the magnitude of their scaled forms [105].

One common motivation for scaling in optimization is to change the order of magnitude of the variables and constraints to be more favorable for computation [97, 105]. Large order of magnitude differences in either the variables or function values can produce ill-conditioned matrices such as Jacobians and Hessians; thus, algorithmic calculations may become unstable or inefficient [105]. Systematic preconditioning methods have been developed to scale special cases of problem elements such as linear constraints [124, 125]. This has been observed to be especially important in DO to ensure robustness and accuracy [95, 97]. Some automatic scaling procedures have been developed to help alleviate some of the computational issues associated with solving dynamic optimization problems [97].

Scaling has also been directly included in some design studies. In Ref. [122], the author develops a method for assessing the importance of scaling laws in design optimization and approximate similitude metrics for obtaining dynamically similar optimal solutions. There have been a number of interesting examples of controller design that utilizes scaling, including Refs. [126, 127], but are typically limited to linear feedback controllers. Additionally, in Ref. [18], the authors utilized scaling to understand the trends found in the solutions to a more complete optimization formulation (and this example will be discussed in detail in Sec. 4.3.1).

Some final uses include performing scaled tests [118, 120] and checking for dimensional homogeneity [120]. All of the examples provide a rich history of scaling in engineering activities.

4.1.2 Dynamic Optimization

In this chapter, we consider design optimization problems that are well-posed as the following DO problem:

$$\min_{\boldsymbol{u}, \boldsymbol{p}} \quad \int_{t_0}^{t_f} \mathcal{L}(t, \boldsymbol{\xi}, \boldsymbol{u}, \boldsymbol{p}) dt + \mathcal{M}(\boldsymbol{\xi}(t_0), \boldsymbol{\xi}(t_f), \boldsymbol{u}, \boldsymbol{p}) \quad (4.1a)$$

$$\text{subject to: } \dot{\boldsymbol{\xi}} - \boldsymbol{f}(t, \boldsymbol{\xi}, \boldsymbol{u}, \boldsymbol{p}) = \mathbf{0} \quad (4.1b)$$

$$\boldsymbol{C}(t, \boldsymbol{\xi}, \boldsymbol{u}, \boldsymbol{p}) \leq \mathbf{0} \quad (4.1c)$$

$$\boldsymbol{\phi}(\boldsymbol{\xi}(t_0), \boldsymbol{\xi}(t_f), \boldsymbol{p}) \leq \mathbf{0} \quad (4.1d)$$

where the optimization variables are \boldsymbol{p} (time-independent variables) and \boldsymbol{u} (open-loop control variables), t is the time continuum defined between t_0 and t_f , and $\boldsymbol{\xi}$ are the states. The objective function in Eqn. (4.1a) is in Bolza form where \mathcal{L} is the Lagrange (running cost) term and \mathcal{M} is the Mayer (terminal cost) term. Equation (4.1b) enforces the dynamics modeled as a first-order ordinary differential equation (ODE), Eqn. (4.1c) enforces any time-varying path constraints, and Eqn. (4.1d) enforces any time-independent constraints.

There are two approaches for finding solutions to Prob. (4.1). Indirect methods utilize the optimality conditions of the infinite-dimensional DO problem [95, 96, 104]. It can be quite challenging (or impossible) to solve these equations analytically. Therefore, numerical methods are often employed that provide approximate solutions to the original problem. The alternative solution methods are known as direct methods [95, 96]. Instead of stating the optimality conditions, the control and/or state are parametrized using function approximation and the objective function is approximated using numerical quadrature. This creates a discrete, finite-dimensional problem that then is optimized using large-scale nonlinear program (NLP) solvers [25, 95–97]. Scaling can be advantageous for both methods.

The optimality conditions will be important when scaling DO formulations and are the same optimality conditions for the simultaneous co-design method in Eqn. (3.18).

4.2 Theory of Scaling Dynamic Optimization Formulations

Now the theory of scaling DO formulations is presented with examples to help illustrate the concepts. The basics are described first, which are applicable to sets of differential-algebraic equations (DAEs). Second is scaling in the context of optimization formulations.

4.2.1 Scaling Basics

The first step when scaling a set of equations is to introduce a change of variables. Here we consider linear scaling with:

$$x = \alpha_x \bar{x} + \beta_x \quad (4.2a)$$

$$y(x) = \alpha_y \bar{y}(x) + \beta_y \quad (4.2b)$$

where x is an independent variable, y is a dependent variable, $\{\bar{x}, \bar{y}\}$ are the new dimensionless variables, and $\{\alpha_x, \beta_x, \alpha_y, \beta_y\}$ are the user-defined scaling constants. The only restriction on the scaling constants is $\alpha \neq 0$ to avoid an ill-defined mapping between the scaled and original variables. Other types of scaling are possible [128] but the linear scaling rule often proves to be suitable for DO.

To substitute higher-order derivatives properly, we need to consider the chain rule and linearity of differentiation [129]:

$$\frac{d^n y}{dx^n} = \frac{\alpha_y}{\alpha_x^n} \frac{d^n \bar{y}}{d\bar{x}^n} \quad (4.3)$$

where n is the order of the derivative. If integrals are present, we can use integration by substitution when changing the integral limits [129]:

$$\int_{x_0}^{x_f} f(x) dx = \alpha_x \int_{\bar{x}_0}^{\bar{x}_f} f(\alpha_x \bar{x} + \beta_x) d\bar{x} \quad (4.4)$$

where $f(x)$ is some function and the integration limits have been shifted from $\{x_0, x_f\}$ to $\{\bar{x}_0, \bar{x}_f\}$. With this small set of formulas, we can apply scaling to all the problem elements of Prob. (4.1).

The original system of DAEs can have problem parameters, denoted ρ . Through suitable choices of scaling variables, every system of DAEs with dimensional homogeneity (i.e., the dimensions on the left and right sides are the same) can be transformed into a dimensionless set of DAEs [118]. This will lead to the creation of dimensionless parameters, denoted $\bar{\rho}$, and will prove quite important as they are critical to many of the uses discussed in Sec. 4.1.1. These dimensionless quantities can be the same as the dimensionless quantities derived by using Buckingham's Pi theorem [118, 119, 121]. Here we assume that all equations, inequalities, and inequations have dimensional homogeneity.

4.2.1.1 Example: Spring-Mass System

To illustrate the concepts of the previous section, we apply scaling to a simple spring-mass system. The first step is to write down the equations and assumptions:

$$m\ddot{y}(t) + ky(t) = 0 \quad (4.5a)$$

$$y(t_0) = y_0, \quad \dot{y}(t_0) = v_0 \quad (4.5b)$$

$$m, k > 0, \quad y_0 \neq 0 \quad (4.5c)$$

In this system, the independent variable is t and the dependent variable is y . Now consider the following change of variables based on simple scaling in Eqn. (4.2):

$$t = \alpha_t \bar{t} + \beta_t, \quad y(t) = \alpha_y \bar{y}(\bar{t}) \quad (4.6)$$

The higher-order derivatives, using Eqn. (4.3), are then:

$$\frac{dy(t)}{dt} = \frac{\alpha_y}{\alpha_t} \frac{d\bar{y}(\bar{t})}{d\bar{t}}, \quad \frac{d^2y(t)}{dt^2} = \frac{\alpha_y}{\alpha_t^2} \frac{d^2\bar{y}(\bar{t})}{d\bar{t}^2} \quad (4.7)$$

We are free to choose the scaling constants so let's first look at the system of DAEs with the substitutions applied:

$$m \frac{\alpha_y}{\alpha_t^2} \frac{d^2\bar{y}(\bar{t})}{d\bar{t}^2} + k\alpha_y \bar{y}(\bar{t}) = 0 \quad (4.8a)$$

$$\alpha_y \bar{y} \left(\frac{t_0 - \beta_t}{\alpha_t} \right) = y_0, \quad \frac{\alpha_y}{\alpha_t} \frac{d\bar{y}}{d\bar{t}} \left(\frac{t_0 - \beta_t}{\alpha_t} \right) = v_0 \quad (4.8b)$$

We can perform some algebraic manipulations to make the left-hand side of the equations have unity coefficients:

$$\frac{d^2\bar{y}(\bar{t})}{d\bar{t}^2} = -\frac{k\alpha_t^2}{m} \bar{y}(\bar{t}) \quad (4.9a)$$

$$\bar{y} \left(\frac{t_0 - \beta_t}{\alpha_t} \right) = \frac{y_0}{\alpha_y}, \quad \frac{d\bar{y}}{d\bar{t}} \left(\frac{t_0 - \beta_t}{\alpha_t} \right) = \frac{\alpha_t v_0}{\alpha_y} \quad (4.9b)$$

Now, consider the following choice of the scaling constants:

$$\alpha_t = \sqrt{\frac{m}{k}}, \quad \beta_t = t_0, \quad \alpha_y = y_0 \quad (4.10)$$

Then the scaled system of DAEs is:

$$\frac{d^2\bar{y}(\bar{t})}{d\bar{t}^2} = -\bar{y}(\bar{t}) \quad (4.11a)$$

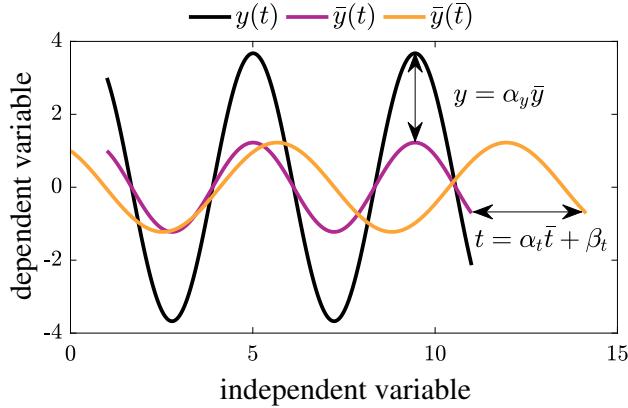


Figure 4.1: Spring-mass system with parameter values $k = 4$, $m = 2$, $y_0 = 3$, $v_0 = -3$, $t_0 = 1$, and $t_f = 11$.

$$\bar{y}(0) = 1, \quad \frac{d\bar{y}}{dt}(0) = \frac{v_0}{y_0} \sqrt{\frac{m}{k}} := \bar{\rho}_1 \quad (4.11b)$$

We see that this choice of constants results in a differential equation with unity coefficients. The original system had five parameters, but this scaled system now has only one, denoted $\bar{\rho}_1$, and is dimensionless. Furthermore, the initial position does not depend on any parameters. The scaling constant $\alpha_t = \sqrt{m/k}$ is typically referred to as the characteristic time constant for this system (also known as the reciprocal of the natural frequency).

Both the scaled and original solutions to this initial value problem are shown in Fig. 4.1. We see the stretching, squeezing, and shifting of the trajectory based on the scaling rules. The scaled trajectory might be more favorable to numerical approximation methods since the average magnitude of its derivative is closer to unity and consistent across the range of parameter values.

4.2.2 Scaling Dynamic Optimization Problems

We first denote the original problem formulation as P with optimality conditions O and optimal solution \mathbf{x}^* . Now the scaled formulation is denoted \bar{P} with optimality conditions \bar{O} and optimal solution $\bar{\mathbf{x}}^*$. For each problem, there may be problem parameters each denoted $\boldsymbol{\rho}$ and $\bar{\boldsymbol{\rho}}$, respectively. The optimization variables are related with the scaling function S : $\mathbf{x} = S(\bar{\mathbf{x}}, \boldsymbol{\rho})$ where for each optimization variable, a linear scaling law defined in Eqn. (4.2) exists that may depend on the problem parameters. Now for a given value of $\boldsymbol{\rho}$, the optimal solution to the two problems are clearly equivalent: if \mathbf{x}^* solves P , then $\bar{\mathbf{x}}^* = S^{-1}(\mathbf{x}^*, \boldsymbol{\rho})$ solves \bar{P} ; if $\bar{\mathbf{x}}^*$ solves \bar{P} , then $\mathbf{x}^* = S(\bar{\mathbf{x}}^*, \boldsymbol{\rho})$ solves P [128].

In some design studies, finding the optimal solution with respect to a single set of parameter values is sufficient to complete the design task. However, many require solutions that vary with respect to the parameters, i.e., $\mathbf{x}^*(\boldsymbol{\rho})$ for various values of $\boldsymbol{\rho}$. We can utilize the scaled problem in this task. If we are given two sets of parameters, $\boldsymbol{\rho}_1$ and $\boldsymbol{\rho}_2$, such that $\bar{\boldsymbol{\rho}}_1 = \bar{\boldsymbol{\rho}}_2$, then we can determine the optimal solution with respect to $\boldsymbol{\rho}_2$ utilizing the solution found with $\boldsymbol{\rho}_1$:

$$\mathbf{x}^*(\boldsymbol{\rho}_2) = S(\bar{\mathbf{x}}^*(\bar{\boldsymbol{\rho}}_1), \boldsymbol{\rho}_2) \quad (4.12)$$

The condition that $\bar{\boldsymbol{\rho}}_1 = \bar{\boldsymbol{\rho}}_2$ is potentially restrictive [122], but it also can be quite useful. This implies that we only need to generate solutions for all relevant values of $\bar{\boldsymbol{\rho}}$ rather than $\boldsymbol{\rho}$, which is favorable since typically $\bar{\boldsymbol{\rho}}$ contains fewer elements than the original $\boldsymbol{\rho}$.

Additional scaling might be possible depending on what can be discerned from the activity of the inequality constraints [105] but will be problem dependent. If certain constraints are found to be inactive for particular ranges of the problem parameters and the reduced form of the original optimization problem eliminates any dependence on a particular parameter, then solutions may be scaled irrespective of that parameter (as long as the inactivity holds in the original problem). In DO, there may be infinite-dimensional path constraints where determining the activity can be challenging (one of the primary motivations for direct methods [32]). This may lead to solutions that are widely different for minor changes in the problem parameters, so scaling must be done carefully (this is shown in the example in Sec. 4.3.1).

To determine how the optimality conditions are related between the original and scaled problems, additional scaling constants for the multipliers need to be introduced. Then the correct values for these scaling constants must be determined such that a map is known between O and \bar{O} .

The relationships between the optimization formulations, optimality conditions, and optimal solutions for both the original and scaled problems are shown in Fig. 4.2. These relations will be discussed in the context of different paths that may be taken to obtain solutions to a particular DO problem. Consider first the standard case where the original problem is proposed and then an optimal solution is found (e.g., using direct transcription (DT) and an NLP solver):

$$P \rightarrow \mathbf{x}^*$$

An alternative is to utilize the optimality conditions in Eqn. (3.18) to obtain an analytical

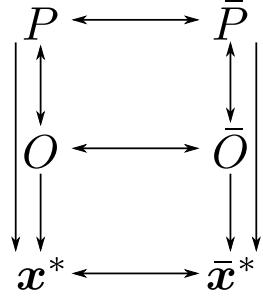


Figure 4.2: Relationships between the optimization formulations, optimality conditions, and optimal solutions for both the original and scaled problems.

solution or using a numeric indirect method:

$$P \rightarrow O \rightarrow \mathbf{x}^*$$

Now using scaling, we could instead transform to P to \bar{P} , and solve the scaled problem. Then the scaled solution can be mapped back to obtain the original problem's solution:

$$P \rightarrow \bar{P} \rightarrow \bar{\mathbf{x}}^* \rightarrow \mathbf{x}^*$$

The key here is that $\bar{\mathbf{x}}^*$ can be used to generate multiple \mathbf{x}^* using Eqn. (4.12). One final approach that may yield the most information from the scaling procedure is utilizing the optimality conditions:

$$P \rightarrow O \rightarrow \bar{O} \rightarrow \bar{P} \rightarrow \bar{\mathbf{x}}^* \rightarrow \mathbf{x}^*$$

With this approach, all of the suggestions in this section can be explored such as constraint activity.

The following section will use the theory in this section to illustrate scaling in DO with some motivating examples.

4.3 Motivating Examples

In this section, a number of examples of scaling in DO are presented. Some examples have direct application to existing design problems while others provide a more conceptual illustration. The examples in this section and the many uses described in Sec. 4.1.1 provide a strong foundation for using scaling in novel DO design problems.

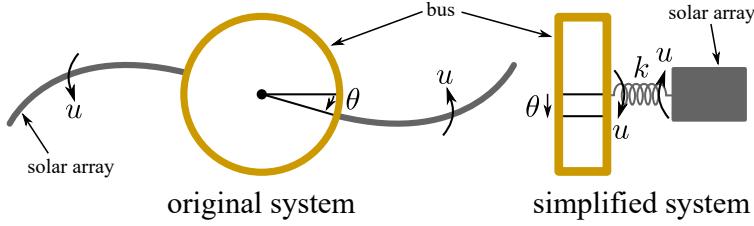


Figure 4.3: Illustrations of original and simplified strain-actuated solar array systems in Sec. 4.3.1.

4.3.1 Example 1: Simple SASA Problem

This first example was developed to better understand the observed trends from an existing design study (for full details see Chapter 7). The co-design study focused on developing a novel strain-actuated solar array (SASA) system for spacecraft pointing control and jitter reduction [18]. In this system, distributed piezoelectric actuators were used to strain the solar arrays, causing reactive forces that could be used to control the bus (spacecraft body) with higher precision, higher bandwidth, and reduced vibrations.

The observed trend was an optimal constant ratio between the natural period of the first mode, T_1 , and time allotted for performing the maneuver, t_f (which is visualized in Fig. 10 of Ref. [18]). There was much discussion on why this ratio seemed to be constant and if there was any significance to the value of this optimal ratio (approximately $T_1/t_f = 4.41$). To help provide some insight into these questions, a much simpler, but still representative, design problem was proposed. In Fig. 4.3, both the original and simplified SASA systems are visualized. The simplified system modeled many of the fundamental phenomena present in the original high-fidelity system using a small number of lumped parameters.

The problem formulation for the simplified system is:

$$\min_{k,u(t)} -\theta(t_f) \quad (4.13a)$$

$$\text{subject to: } J\ddot{\theta}(t) + k\theta(t) = u(t) \quad (4.13b)$$

$$\dot{\theta}(0) = \dot{\theta}(0) = 0 \quad (4.13c)$$

$$\dot{\theta}(t_f) = 0 \quad (4.13d)$$

$$|u(t)| \leq u_{\max} \quad (4.13e)$$

where θ is the relative displacement of the bus, J is related to the inertia ratio between the solar arrays and bus, u is an open-loop control moment applied to the solar array and is bounded by u_{\max} , and k is the stiffness in the solar array. The boundary constraints enforce

the system to start at rest with zero energy and end at rest. The optimization variables are both k and $u(t)$. Scaling can be applied to help analyze this DO problem.

4.3.1.1 Standard Form

First, we write the original DO problem in Eqn. (4.13) the standard form:

$$\min_{k,u(t)} -\xi_1(t_f) \quad (4.14a)$$

$$\text{subject to: } \dot{\boldsymbol{\xi}} = \begin{bmatrix} 0 & 1 \\ -k/J & 0 \end{bmatrix} \boldsymbol{\xi} + \begin{bmatrix} 0 \\ 1/J \end{bmatrix} u \quad (4.14b)$$

$$\phi_1 := \xi_1(0) = 0, \quad \phi_2 := \xi_2(0) = 0 \quad (4.14c)$$

$$\phi_3 := \xi_2(t_f) = 0 \quad (4.14d)$$

$$C_1 := u - u_{\max} \leq 0, \quad C_2 := -u - u_{\max} \leq 0 \quad (4.14e)$$

$$\text{where: } \xi_1 = \theta, \quad \xi_2 = \dot{\theta} \quad (4.14f)$$

The Hamiltonian is then:

$$H = \lambda_1 \xi_2 + \lambda_2 \left(-\frac{k}{J} \xi_1 + \frac{u}{J} \right) + \mu_1 (u - u_{\max}) + \mu_2 (-u - u_{\max}) \quad (4.15)$$

Now using the conditions in Eqn. (3.18), the additional conditions for optimality are:

$$\dot{\lambda}_1 = \frac{k}{J} \lambda_2, \quad \dot{\lambda}_2 = -\lambda_1 \quad (4.16a)$$

$$0 = \frac{\lambda_2}{J} + \mu_1 - \mu_2 \quad (4.16b)$$

$$0 = \mu_1 (u - u_{\max}), \quad \mu_1 \geq 0, \quad 0 = \mu_2 (-u - u_{\max}), \quad \mu_2 \geq 0 \quad (4.16c)$$

$$0 = \nu_1 \xi_1(0), \quad \nu_1 \neq 0, \quad 0 = \nu_2 \xi_2(0), \quad \nu_2 \neq 0 \quad (4.16d)$$

$$0 = \nu_3 \xi_2(t_f), \quad \nu_3 \neq 0 \quad (4.16e)$$

$$0 = \lambda_1(0) - 1 + \nu_1, \quad 0 = \lambda_2(0) + \nu_2 \quad (4.16f)$$

$$0 = \lambda_1(t_f), \quad 0 = \lambda_2(t_f) - \nu_3 \quad (4.16g)$$

$$0 = \int_0^{t_f} \lambda_2 \frac{\xi_1}{J} dt \quad (4.16h)$$

4.3.1.2 Scaled Form

Consider the following change of variables based on linear scaling described in Sec. 4.2.1:

$$t = \alpha_t \bar{t}, \quad u(t) = \alpha_u \bar{u}(t), \quad \theta(t) = \alpha_\theta \bar{\theta}(t) \quad (4.17)$$

Next, the differential equation in Eqn. (4.13b) is parameterized as:

$$J \frac{\alpha_\theta}{\alpha_t^2} \frac{d^2}{dt^2} \bar{\theta} + k \alpha_\theta \bar{\theta} = \alpha_u \bar{u} \Rightarrow \frac{d^2}{d\bar{t}^2} \bar{\theta} := \bar{\theta}'' = -k \frac{\alpha_t^2}{J} \bar{\theta} + \frac{\alpha_u \alpha_t^2}{J \alpha_\theta} \bar{u} \quad (4.18)$$

The three parameters in the problem are $\{J, u_{\max}, t_f\}$. There are three scaling constants to choose. The following points are considered when deciding how to scale the problem based on some previous intuition:

- Since all initial and final state conditions are zero, we should not start with α_θ
- The time horizon should be fixed, i.e., independent of t_f
- The magnitude of the control force should be unity to remove dependence on u_{\max}
- It is fine to have k multiplied by some constants since k is an optimization variable and is not directly constrained
- The natural period of this second-order system is $T = 2\pi\sqrt{J/k}$

With these considerations, we select the values of the scaling parameters as:

$$\alpha_t = \frac{t_f}{2\pi}, \quad \alpha_u = u_{\max}, \quad \alpha_\theta = \frac{u_{\max} t_f^2}{4\pi^2 J} \quad (4.19)$$

Note that the horizon is now fixed between 0 and 2π . Therefore, the scaled optimization problem is:

$$\min_{\bar{k}, \bar{u}(\bar{t})} - \frac{u_{\max} t_f^2}{4\pi^2 J} \bar{\theta}(2\pi) \quad (4.20a)$$

$$\text{subject to: } \bar{\theta}'' = -\frac{kt_f^2}{4\pi^2 J} \bar{\theta} + \bar{u} \quad (4.20b)$$

$$\bar{\theta}(0) = \bar{\theta}'(0) = 0 \quad (4.20c)$$

$$\bar{\theta}'(2\pi) = 0 \quad (4.20d)$$

$$|\bar{u}(\bar{t})| \leq 1 \quad (4.20e)$$

The scaled problem has two dimensionless quantities:

$$\bar{\rho}_1 = \frac{kt_f^2}{4\pi^2 J} \equiv \frac{t_f^2}{T^2}, \quad \bar{\rho}_2 = \frac{u_{\max} t_f^2}{4\pi^2 J} \quad (4.21)$$

First, we have $\sqrt{\bar{\rho}_1} = t_f/T$, the exact ratio we are investigating. We also have $\bar{\rho}_1$ as the only

part of the formulation that depends on k and since there are no constraints on k , we are effectively designing the quantity $\bar{\rho}_1$ directly. Second, $\bar{\rho}_2$ is a positive constant linear factor in the objective function, so it will not affect the nature of the solutions [128], and it can be removed temporarily from the scaled formulation. Therefore, the formulation is equivalent to the following DO problem in the standard form:

$$\min_{\bar{\rho}_1, \bar{u}(\bar{t})} -\bar{\xi}_1(2\pi) \quad (4.22a)$$

$$\text{subject to: } \dot{\bar{\xi}} = \begin{bmatrix} 0 & 1 \\ -\bar{\rho}_1 & 0 \end{bmatrix} \bar{\xi} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \bar{u} \quad (4.22b)$$

$$\phi_1 := \bar{\xi}_1(0) = 0, \quad \phi_2 := \bar{\xi}_2(0) = 0 \quad (4.22c)$$

$$\phi_3 := \bar{\xi}_2(2\pi) = 0 \quad (4.22d)$$

$$C_1 := \bar{u} - 1 \leq 0, \quad C_2 := -\bar{u} - 1 \leq 0 \quad (4.22e)$$

The dependence on the parameters $\boldsymbol{\rho} = \{J, u_{\max}, t_f\}$ has been removed completely; therefore, finding the single solution, $\bar{\boldsymbol{x}}^*$, to this scaled formulation will give all solutions for any valid $\boldsymbol{\rho}$! We have also derived relationships to indicate how different values for these parameters directly affect the solution, i.e., the function S .

4.3.1.3 Equivalence of the Optimality Conditions

It is still illustrative to show that there is a linear mapping between the O and \bar{O} for Probs. (4.14) and (4.22). The Hamiltonian for the scaled problem is:

$$\bar{H} = \bar{\lambda}_1 \bar{\xi}_2 + \bar{\lambda}_2 (-\bar{\rho}_1 \bar{\xi}_1 + \bar{u}) + \bar{\mu}_1 (\bar{u} - 1) + \bar{\mu}_2 (-\bar{u} - 1) \quad (4.23)$$

We expect some scaling of the multipliers between the two problems so we will define a scaling rule for each:

$$\begin{aligned} \lambda_1 &= \alpha_{\lambda_1} \bar{\lambda}_1, & \lambda_2 &= \alpha_{\lambda_2} \bar{\lambda}_2, & \mu_1 &= \alpha_{\mu_1} \bar{\mu}_1, & \mu_2 &= \alpha_{\mu_2} \bar{\mu}_2 \\ \nu_1 &= \alpha_{\nu_1} \bar{\nu}_1, & \nu_2 &= \alpha_{\nu_2} \bar{\nu}_2, & \nu_3 &= \alpha_{\nu_3} \bar{\nu}_3 \end{aligned} \quad (4.24)$$

To determine the proper values of the scaling constants, we need to substitute the change of variables into the optimality conditions in Eqn. (4.16):

$$\frac{\alpha_{\lambda_1}}{\alpha_t} \dot{\bar{\lambda}}_1 = \frac{k}{J} \alpha_{\lambda_2} \bar{\lambda}_2, \quad \frac{\alpha_{\lambda_2}}{\alpha_t} \dot{\bar{\lambda}}_2 = -\alpha_{\lambda_1} \bar{\lambda}_1 \quad (4.25a)$$

$$0 = \alpha_{\lambda_2} \frac{\bar{\lambda}_2}{J} + \alpha_{\mu_1} \bar{\mu}_1 - \alpha_{\mu_2} \bar{\mu}_2 \quad (4.25b)$$

$$0 = \alpha_{\mu_1} \bar{\mu}_1 (\alpha_u \bar{u} - u_{\max}), \quad \alpha_{\mu_1} \bar{\mu}_1 \geq 0 \quad (4.25c)$$

$$0 = \alpha_{\mu_2} \bar{\mu}_2 (-\alpha_u \bar{u} - u_{\max}), \quad \alpha_{\mu_2} \bar{\mu}_2 \geq 0 \quad (4.25d)$$

$$0 = \alpha_{\nu_1} \bar{\nu}_1 \alpha_\theta \bar{\xi}_1(0), \quad \alpha_{\nu_1} \bar{\nu}_1 \neq 0 \quad (4.25e)$$

$$0 = \alpha_{\nu_2} \bar{\nu}_2 \frac{\alpha_\theta}{\alpha_t} \bar{\xi}_2(0), \quad \alpha_{\nu_2} \bar{\nu}_2 \neq 0 \quad (4.25f)$$

$$0 = \alpha_{\nu_3} \bar{\nu}_3 \frac{\alpha_\theta}{\alpha_t} \bar{\xi}_2(2\pi), \quad \alpha_{\nu_3} \bar{\nu}_3 \neq 0 \quad (4.25g)$$

$$0 = \alpha_{\lambda_1} \bar{\lambda}_1(0) - 1 + \alpha_{\nu_1} \bar{\nu}_1, \quad 0 = \alpha_{\lambda_2} \bar{\lambda}_2(0) + \alpha_{\nu_2} \bar{\nu}_2 \quad (4.25h)$$

$$0 = \alpha_{\lambda_1} \bar{\lambda}_1(2\pi), \quad 0 = \alpha_{\lambda_2} \bar{\lambda}_2(2\pi) - \alpha_{\nu_3} \bar{\nu}_3 \quad (4.25i)$$

$$0 = \alpha_t \int_0^{2\pi} \alpha_{\lambda_2} \bar{\lambda}_2 \frac{\alpha_\theta \bar{\xi}_1}{J} d\tau \quad (4.25j)$$

With the substitution applied, Eqn. (4.25) above should match the optimality conditions for Prob. (4.22) exactly. This is accomplished with the following relationships:

$$\alpha_{\lambda_1} = \alpha_{\nu_1} = 1, \quad \alpha_{\lambda_2} = \alpha_{\nu_2} = \alpha_{\nu_3} = J\alpha_{\mu_2} = J\alpha_{\mu_2} = \alpha_t \quad (4.26)$$

Thus, a simple linear map exists between the optimality conditions between the original and scaled forms. The two Hamiltonians are related by¹²:

$$H = \alpha_\theta \alpha_t \bar{H} \quad (4.27)$$

4.3.1.4 Bounded Period

An additional inequality was necessary to explain further the observed results from the original study. This constraint was in the form of a bound on the period:

$$T = \frac{t_f}{\sqrt{\bar{\rho}_1}} \leq T_{\max} \quad (4.28)$$

which can be written as:

$$\frac{t_f^2}{T_{\max}^2} - \bar{\rho}_1 := \bar{\rho}_3 - \bar{\rho}_1 \leq 0 \quad (4.29)$$

where $\bar{\rho}_3$ is an additional dimensionless parameter. We can denote $\bar{\rho}_1^\dagger$ as the optimal value for $\bar{\rho}_1$ without the additional constraint in Eqn. (4.29). If $\bar{\rho}_3 < \bar{\rho}_1^\dagger$, then the constraint is inactive, and the previous solution is valid. However, if $\bar{\rho}_3 \geq \bar{\rho}_1^\dagger$, then the constraint is

¹²Using the objective $-\bar{\rho}_2 \bar{\theta}(2\pi)$, the relationship is $H = \alpha_\theta^2 \alpha_t \bar{H}$ with all multipliers containing an additional α_θ term.

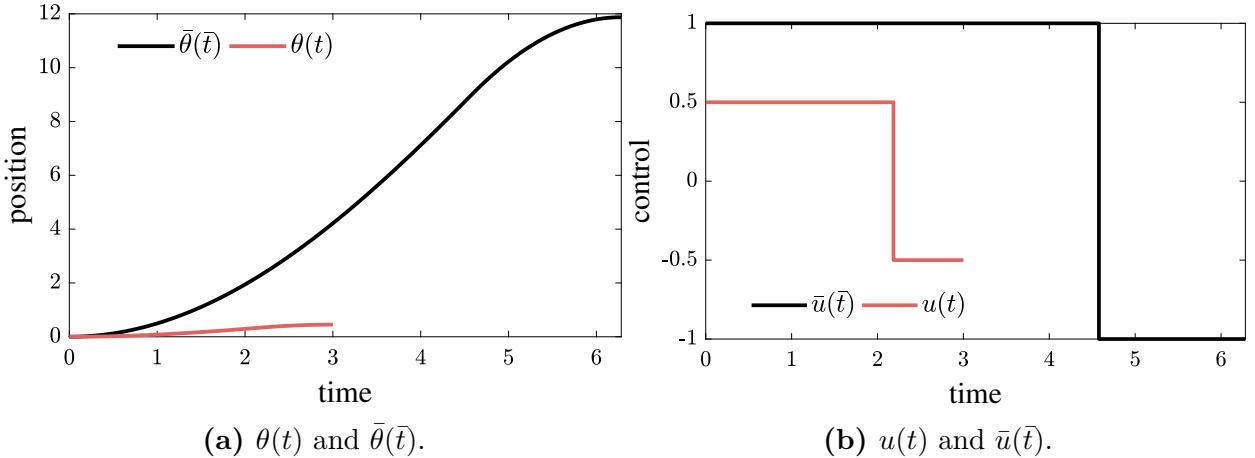


Figure 4.4: Scaled and unscaled solutions for the simple SASA problem with $u_{\max} = 0.5$, $t_f = 3$, $J = 3$.

active and we would need to find solutions for every required value of $\bar{\rho}_3$ since the form of the optimal control will vary for each value of $\bar{\rho}_3$. In this problem, the control remains bang-bang in nature but the number of switches increases and the switching locations vary (see Fig. 4.4b).

4.3.1.5 Solution

With the optimization problem, optimality conditions, and optimal solutions thoroughly characterized, we can compare this simple SASA problem to the original design study in Ref. [18]. Both the scaled and unscaled solutions for the simple SASA problem without the additional bound on the period being active are shown in Fig. 4.4. There are a number of similarities between the trajectories in Ref. [18] and in Fig. 4.4, including the bang-bang nature of the control when the period constraint is not active, and the general shape of the bus angle trajectories.

The primary comparison is Fig. 4.5, which includes the results from Fig. 10 of Ref. [18]. The figure plots the natural period of the first mode vs. t_f . There is an observed linear trend until a period limit seems to be reached. This is present for both design representations used: piecewise linear segments (PLS) and variable length (VL). Both of these representations can change the structural properties of the solar array. Although it is tough to see in the figure, there are coinciding data points at $t_f = 0.12$ s for both cases indicating that there is a similar optimal value for T_1 (and ratio).

Observing Fig. 4.5, there is a direct parallel between the data points and the simple SASA

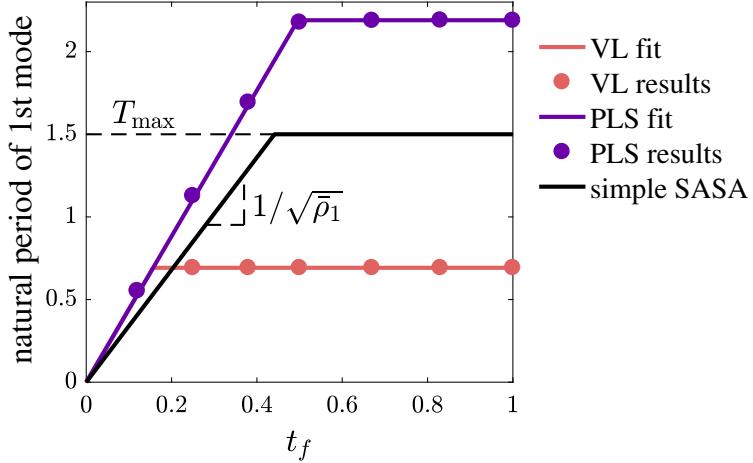


Figure 4.5: Natural period vs. t_f results for the simple SASA problem and original design study in Chapter 7.

problem. For the scaled problem, $\bar{\rho}_1^\dagger = 0.0866$ while the results from Ref. [18] indicate $\bar{\rho}_1$ should be approximately 0.0513. The difference may be attributed to the representation and constraints in the original design study. Initial discussions of the results tried to pin the ratio t_f/T_1 to 1/4 due to the periodic resonance of a simple beam. However, the simple SASA problem suggests that this ratio is simply an arbitrary constant dependent on the interaction between the bang-bang controller and dynamics. Continuing with the comparisons, the T_{\max} bound seems to equal about 2.19 s for the PLS case and 0.69 s for the VL case. If the physical-system constraints could be lifted, we have a reasonable prediction for how T_1 should vary.

This example illustrated how scaling can be utilized effectively to obtain insights in a design study.

4.3.2 Example 2: Co-Design Transfer Problem

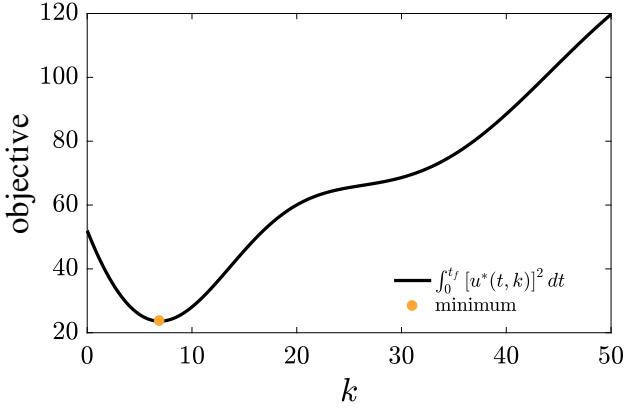
In this example, we will use scaling to transform a co-design problem into a form with a known solution¹³. Consider the following co-design problem:

$$\min_{k,u(t)} \int_0^{t_f} u^2 dt \quad (4.30a)$$

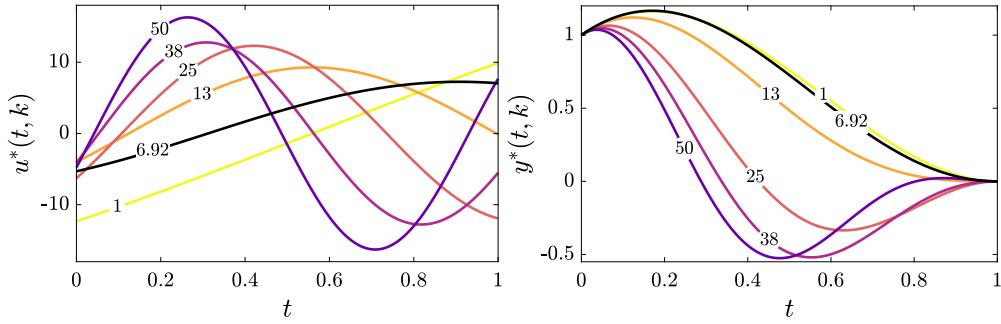
$$\text{subject to: } \ddot{y} = -ky + u \quad (4.30b)$$

$$y(0) = y_0, \dot{y}(0) = v_0, y(t_f) = 0, \dot{y}(t_f) = 0 \quad (4.30c)$$

¹³This problem was also used in Sec. 3.4.2.



(a) Objective function values.



(b) $u^*(t, k)$ for various values of k . (c) $y^*(t, k)$ for various values of k .

Figure 4.6: Co-design transfer problem with $t_f = 1$, $x_0 = 1$, and $v_0 = 2$ (with $k^* = 6.924$).

where k is the time-independent, physical-system design variable and $u(t)$ is the open-loop control design variable. Through scaling, we can transform Prob. (4.30) above into the problem in Ref. [104, pp. 166–167]. To accomplish this, consider the following change of variables:

$$t = \alpha_t \bar{t}, \quad u = \alpha_u \bar{u} \quad (4.31)$$

We choose the following values for the scaling constants:

$$\alpha_t = \frac{1}{\sqrt{k}}, \quad \alpha_u = k \quad (4.32)$$

where $k > 0$. The case for $k = 0$ needs to be handled separately. Substituting in these

scaling laws results in the following problem formulation:

$$\min_{k, \bar{u}(\bar{t})} k^{3/2} \int_0^{\bar{t}_f} \bar{u}^2 d\bar{t} \quad (4.33a)$$

$$\text{subject to: } y'' = -y + \bar{u} \quad (4.33b)$$

$$y(0) = y_0, \quad y'(0) = \bar{v}_0 \quad (4.33c)$$

$$y(\bar{t}_f) = 0, \quad y'(\bar{t}_f) = 0 \quad (4.33d)$$

$$\text{where: } \bar{t}_f = \sqrt{k} t_f, \bar{v}_0 = v_0 / \sqrt{k} \quad (4.33e)$$

The solution to Prob. (4.33) above is readily available in Ref. [104]. Using this result and applying the scaling rules in Eqn. (4.31), we obtain the following solution for $u(t)$ as a function of k :

$$\begin{aligned} u^*(t, k) &= -\frac{2k}{kt_f^2 - \sin^2(\sqrt{k}t_f)} \left(c_1(t, k)x_0 + c_2(t, k) \frac{v_0}{\sqrt{k}} \right) \\ c_1(t, k) &= \sin(\sqrt{k}(t_f - t)) \sin(\sqrt{k}t_f) - \sqrt{k}t_f \sin(\sqrt{k}t) \\ c_2(t, k) &= -\cos(\sqrt{k}(t_f - t)) \sin(\sqrt{k}t_f) + \sqrt{k}t_f \cos(\sqrt{k}t) \end{aligned} \quad (4.34)$$

For certain problem parameter values, the solution for various values of k is shown in Fig. 4.6. The original objective function can now be computed analytically with Eqn. (4.34). With this closed-form expression for the objective function, an optimality condition for k can be derived. This property makes this problem quite suitable for investigations between nested and simultaneous co-design methods and can be used to illustrate a number of concepts found in many co-design problems (such as open-loop control, multiple optima, degenerate plant designs, and a system-level objective function) [32].

For many DO problems, obtaining a closed-form solution for a portion of the design variables may not be possible. Observing different (but equivalent) forms of the same design optimization formulation can facilitate a better understanding of the design problem. In this example, we see that for larger values of k , the scaled problem has a larger time horizon but smaller initial velocity.

In addition, these different forms may be more suitable computation depending on the parameter values. In this example using the original Prob. (4.30) formulation, large values of k would result in highly oscillatory solutions. With the time-horizon fixed, large discretization errors would start appearing in the solutions with a fixed number of time points if a DT solution method was utilized. This helps motivate the final example.

4.3.3 Example 3: Direct Transcription Discretization

DT is a popular solution method for solving DO problems and is described in more detail in Chapter 5 [21, 27, 95, 96]. In this approach, the dynamics are approximated with a large number of equality constraints, termed defect constraints. Equality constraints in a NLP are typically of the form $h(\mathbf{x}) = 0$. However, most numerical algorithms implement this constraint type as two inequality constraints as:

$$-\epsilon \leq h(x) \leq \epsilon \quad (4.35)$$

where ϵ is a small tolerance number (e.g., the default tolerance in MATLAB's `fmincon` function is $\epsilon = 10^{-6}$ [130]). In this example, we will demonstrate how the absolute magnitudes of the state variables can greatly degrade direct transcription approximations.

Consider the simple scalar system:

$$\dot{y} = ay, \quad y(0) = y_0 \quad (4.36)$$

where the solution to this set of DAEs is $\hat{y}(t) = y_0 e^{at}$. We can introduce a change of variables as:

$$t = \alpha_t \bar{t}, \quad y = \alpha_y \bar{y}$$

where the scaling constants are $\alpha_t = |a|$ and $\alpha_y = y_0$, resulting in the following scaled system:

$$\bar{y}' = \bar{y}, \quad \bar{y}(0) = 1 \quad (4.37)$$

We can now see how the relative error is influenced by the different systems.

A basic single-step DT method is the trapezoidal rule and the defect constraint that ensures accurate dynamics has the following form for a scalar ODE:

$$0 = y_k - y_{k-1} - \frac{\Delta_k}{2} \left(f(t_{k-1}, y_{k-1}) + f(t_k, y_k) \right) \quad (4.38)$$

where y_k is the approximate value of $y(t)$ at t_k , $\Delta_k = t_k - t_{k-1}$ is the step-size parameter, and $f(t_k, y_k)$ is the value of the derivative function at t_k .

We will consider the initial step, i.e., $k = 1$. Then the DT constraint for Eqn. (4.36) is:

$$0 = y_1 - y_0 - \frac{\Delta_1}{2} (ay_0 + ay_1) \quad (4.39)$$

Using the tolerances in Eqn. (4.35), we can derive the minimum and maximum values for

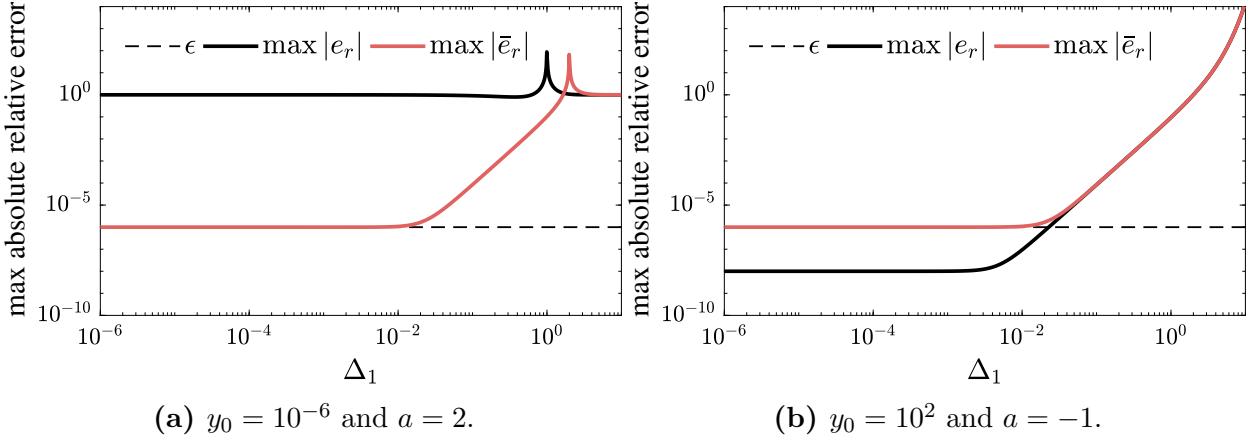


Figure 4.7: Maximum absolute relative error vs. step size for original and scaled defect constraints.

y_1 that still satisfy the constraint:

$$\frac{(1 + \Delta_1 a/2) y_0 - \epsilon}{(1 - \Delta_1 a/2)} \leq y_1 \leq \frac{(1 + \Delta_1 a/2) y_0 + \epsilon}{(1 - \Delta_1 a/2)} \quad (4.40)$$

Alternatively, the bounds for \bar{y}_1 are:

$$\frac{(1 + \text{sign}(a)\Delta_1/2) - \epsilon}{(1 - \text{sign}(a)\Delta_1/2)} \leq \bar{y}_1 \leq \frac{(1 + \text{sign}(a)\Delta_1/2) + \epsilon}{(1 - \text{sign}(a)\Delta_1/2)} \quad (4.41)$$

The relative error is defined as:

$$e_r = \frac{y_1 - \hat{y}_1}{\hat{y}_1} = \frac{y_1 - y_0 e^{at_1}}{y_0 e^{at_1}}, \quad \bar{e}_r = \frac{\bar{y}_1 - e^{\text{sign}(a)t_1}}{e^{\text{sign}(a)t_1}} \quad (4.42)$$

Now, consider the following values for the problem and DT parameters: $y_0 = 10^{-6}$, $a = 2$, $\epsilon = 10^{-6}$, $\Delta_1 = 10^{-3}$. For these values, we have the following relative error bounds:

$$-9.99 \times 10^{-1} \leq e_r \leq 9.99 \times 10^{-1} \quad (4.43a)$$

$$-9.99 \times 10^{-7} \leq \bar{e}_r \leq 9.99 \times 10^{-7} \quad (4.43b)$$

We see that the e_r is quite poor using the original system while \bar{e}_r is right around the expected ϵ tolerance. Consider one more choice of parameters: $y_0 = 10^3$, $a = -1$, $\epsilon = 10^{-6}$, $\Delta_1 = 10^{-3}$ and error bounds:

$$-1.08 \times 10^{-9} \leq e_r \leq 9.13 \times 10^{-10} \quad (4.44a)$$

$$-1.00 \times 10^{-6} \leq \bar{e}_r \leq 1.00 \times 10^{-6} \quad (4.44b)$$

Now we see that the relative error bounds are tighter than ϵ which could lead to numerical issues. The scaled form, however, is still right around the ϵ tolerance. The maximum absolute relative error vs. step size is shown in Fig. 4.7. From these plots, we see that for a sufficiently small step size, the scaled system has error bounds near ϵ .

Both of these parameter sets show the importance of scaling the dynamics to be near unity when using a DT implementation. Since DT implementations use a large number of linked defect constraints, the errors can compound, potentially producing highly inaccurate results. Additionally, scaling the time horizon provides a more uniform method for selecting Δ to achieve suitable accuracy without too many optimization variables (note that $\Delta_1 = 10^{-3}$ for \bar{e}_r is quite good in both plots). Finally, scaling the states to be near unity is also produces computationally favorable matrices (Hessian/Jacobians) and finite differencing [95, 97].

4.4 Summary

In this chapter, we explored scaling in DO with a particular focus on how to leverage scaling in design optimization. A review of the large amount of work around scaling was provided. The necessary theory for scaling DO formulations was presented and a number of novel motivating examples were provided. Scaling was shown to help facilitate finding accurate, generalizable, and intuitive information.

At a basic level, scaling is simply the stretching, squeezing, and shifting of the problem elements such as the time continuum, design variables, constraints, and objective function. The unique structure of DO suggests that scaling can be utilized in novel ways to provide better analysis and formulations more favorable for different solution methods. The mechanics of scaling are fairly straightforward but proper utilization of scaling is heavily reliant on the creativity and intuition of the designer. This nebulous qualification is the primary limitation on the use of scaling. However, scaling is also limited by the scaling law chosen, the structure of the problem, and the number of free parameters. Furthermore, scaling does not remove optimization; we still need to solve some form of the optimization problem. Improper use of scaling can lead to incorrect solutions/insights or amplification of numerical issues rather than mitigation.

Aiding the designer in constructing the appropriate scaled formulations needs compelling examples. In the simple SASA problem, scaling was used to understand observed results from more complete, higher-fidelity design study. The simpler scaled optimization problem and dimensionless variables provided a number of insights. In the second example, the so-

lution to a co-design problem was found by leveraging a scaled formulation with the known solution. Observing different (but equivalent) forms of the same design optimization formulation can facilitate a better understanding of the design problem. In the final example, the discretization error of one of the popular solution methods (namely direct transcription) for DO was explored. The scaled system showed much more favorable properties than the original. Additional work is needed to develop general guidelines to aid in balancing the many uses of scaling.

Chapter 5

Direct Transcription and Linear-Quadratic Dynamic Optimization¹⁴

“Since all the effects of Nature follow a certain law of maxima or minima, there is no doubt that, on the curved paths, which the bodies describe under the action of certain forces, some maximum or minimum property ought to obtain. What this property is, nevertheless, does not appear easy to define a priori by proceeding from the principles of metaphysics;”

L. Euler [132, p. 106]

Direct transcription is a solution strategy discussed briefly in Chapters 3 and 4 for finding approximate solutions to dynamic optimization problems. This chapter focuses on a particular subclass that is relevant to the two case studies in Chapters 7 and 8. Both use the nested co-design strategy from Chapter 3 so many control subproblems need to be solved and efficiency is paramount. The methods developed in this chapter provide a single unified description to automatically generate and solve this class of problems, even if there are different architectures with a varying number of states and controls.

5.1 Introduction

For more than half a century, dynamic optimization, or optimization with time-varying quantities, has played an integral role in the advancement of many designed systems, including applications in chemical engineering [96], aerospace engineering [95], wave energy conversion [133], and finance/economics [134]. Nonlinear dynamic optimization (NLDO) represents the most general class of problems [95, 96, 104]. A subclass of NLDO is linear-quadratic dynamic optimization (LQDO) where certain elements of the formulation are limited to quadratic and linear functions [20, 104, 135]. Frequently, LQDO formulations used in the

¹⁴Elements of this chapter are based on work completed in Ref. [131].

literature are only a subclass of the general LQDO problem. The key feature shared between the LQDO formulations is that solutions may be found via an appropriate quadratic program (QP), a particular class of finite-dimensional mathematical programs [128]. Here we present a unified framework for LQDO that can be solved as QPs.

In addition to providing a clear delineation of the general LQDO problem class, we develop an automated problem generation procedure (APGP) to form the QPs that represent LQDO problems. Here we define an APGP as a procedure in which, given a natural and manageable description of the problem, one can obtain a numerical solution with little or no user expertise. A key to minimizing the amount of knowledge needed by the user is an automated and efficient implementation of the various solution methods. Such procedures are available for DO (e.g., GPOPS-II [136], PSOPT [137], PROPT [138], SOS [139], and DIRCOL [140]) but typically are developed for the more general NLDO problems and therefore cannot effectively leverage the structure of LQDO. For the APGPs that handle LQDO problems, they are limited to specific solution methods and problem formulations (e.g., MPT3 [141] and MPC TOOLBOX [142]). Manual implementation of these solution methods is still quite prevalent in the literature, perhaps due to the lack of the necessary tools which sufficiently address the challenges of the particular class of problems. Additionally, a wide variety of competing solution methods exist, and comparisons between them cannot typically be performed efficiently (especially if a manual implementation is needed). These issues limit productivity and the general reach of LQDO, but can be addressed using an APGP under a unified framework for LQDO. In addition, this unified framework also provides additional insights into the LQDO problem class and led to additional developments in the solution methods.

The remainder of the chapter is structured as follows. Section 5.2 presents the general problem formulation for LQDO. Next, Sec. 5.2 discusses formulating LQDO problems as QPs with direct transcription methods. Section 5.4 details the APGP which takes a natural and manageable description of the problem and forms the QP. Section 5.5 outlines some extensions to the original LQDO problem formulation. Section 5.6 assesses the APGP with a number of numerical examples. Section 5.7 discusses various future work items.

5.2 Linear-Quadratic Dynamic Optimization

In this section, we begin by describing the general NLDO formulation and then a QP is defined. Then, with an assumed property of the solution method, we will characterize the LQDO formulation that supports solution via QPs.

5.2.1 General Nonlinear Dynamic Optimization

Dynamic-system design permits the optimization of: control trajectories, $\mathbf{u}(t)$; static parameters, \mathbf{p} ; and the time horizon defined by the boundary values t_0 and t_f . Written as an infinite-dimensional mathematical program, the NLDO formulation is:

$$\min_{\mathbf{u}(t), \mathbf{p}, t_0, t_f} \quad \Psi = \int_{t_0}^{t_f} \mathcal{L}(t, \boldsymbol{\xi}(t), \mathbf{u}(t), \mathbf{p}) dt + \mathcal{M}(\mathbf{p}, t_0, \boldsymbol{\xi}(t_0), t_f, \boldsymbol{\xi}(t_f)) \quad (5.1a)$$

$$\text{subject to: } \dot{\boldsymbol{\xi}}(t) - \mathbf{f}(t, \boldsymbol{\xi}(t), \mathbf{u}(t), \mathbf{p}) = \mathbf{0} \quad (5.1b)$$

$$\mathbf{h}(t, \boldsymbol{\xi}(t), \mathbf{u}(t), \mathbf{p}, t_0, \boldsymbol{\xi}(t_0), t_f, \boldsymbol{\xi}(t_f)) = \mathbf{0} \quad (5.1c)$$

$$\mathbf{g}(t, \boldsymbol{\xi}(t), \mathbf{u}(t), \mathbf{p}, t_0, \boldsymbol{\xi}(t_0), t_f, \boldsymbol{\xi}(t_f)) \leq \mathbf{0} \quad (5.1d)$$

where Eqn. (5.1a) defines the objective function with Lagrange \mathcal{L} and Mayer \mathcal{M} terms. Equation (5.1b) enforces the first-order ordinary differential equation (ODE) that describes the dynamic behavior of the states $\boldsymbol{\xi}(t)$. Equation (5.1c) enforces the algebraic equality constraints, and Eqn. (5.1d) enforces the algebraic inequality constraints. Most NLDO problems only contain certain elements of this general formulation. Many presentations of general NLDO reorganize Eqns. (5.1c)–(5.1d) by partitioning the constraints into those which depend on time-varying quantities and those which do not. The time-varying constraints are termed *path* constraints, and time-independent constraints termed *boundary* constraints (as was done in Chapters 3 and 4) [25, 95, 96]. Some formulations also include the states as optimization variables (primarily motivated by the eventual solution method and the states are still constrained by Eqn. (5.1b)).

5.2.2 Quadratic Program

In this chapter, we are concerned with a subclass of Prob. (5.1) that can be solved numerically as a finite-dimensional QP. A QP is defined as:

$$\min_{\mathbf{X}} \quad \frac{1}{2} \mathbf{X}^T \mathbf{H} \mathbf{X} + \mathbf{F}^T \mathbf{X} + c \quad (5.2a)$$

$$\text{subject to: } \mathbf{A}_e \mathbf{X} = \mathbf{B}_e \quad (5.2b)$$

$$\mathbf{A}_i \mathbf{X} \leq \mathbf{B}_i \quad (5.2c)$$

$$\underline{\mathbf{X}} \leq \mathbf{X} \leq \bar{\mathbf{X}} \quad (5.2d)$$

where \mathbf{X} is the set of optimization variables, and all other terms in the formulation are real-valued with no dependence on \mathbf{X} . \mathbf{H} is known as the Hessian matrix and is symmetric.

We note that the QP formulation in Prob. (5.2) is decidedly more structured than NLDO formulation in Prob. (5.1). The optimal solution to a quadratic program exists and is the global optimum if the objective is a convex quadratic function and the feasible set of Prob. (5.2) is nonempty [143]. There are a variety of efficient algorithms for solving QPs, including active-set [143], interior-point [144], and augmented Lagrangian [145] methods.

5.2.3 Linear-Quadratic Dynamic Optimization Problem Formulation

In this section, a subclass of Prob. (5.1) is defined that, when combined with a solution method, can generate a QP in the form of Prob. (5.2) that approximates the infinite-dimensional solution. There are two aspects that determine if a particular infinite-dimensional problem can be formulated as QP: 1) the structure of the objective function and constraints in Prob. (5.1), and 2) the solution method that approximates the infinite-dimensional problem as a finite one. To address the latter aspect, we will only consider order-maintaining methods. An order-maintaining method, for example, would have the following property: if a term is linear in the infinite-dimensional problem, then it will be approximated as a set of variables that appear linearly in the finite-dimensional problem. An equivalent property will need to be true for quadratic terms.

First, we denote the vector of optimization variables as:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{\xi} \\ \boldsymbol{p} \end{bmatrix} \quad (5.3)$$

Both elements of the time horizon are missing because they do not permit QPs (see Sec. 5.5.8). In addition, the states are included to support proper analysis of the problem class and the eventual solution methods. We also see $\boldsymbol{\xi}(t_0)$ and $\boldsymbol{\xi}(t_f)$ directly in the formulation, so we define an expanded set of optimization variables as:

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x}_c \\ \boldsymbol{x}_d \end{bmatrix} \text{ where: } \boldsymbol{x}_c = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{\xi} \end{bmatrix} := \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} \text{ and } \boldsymbol{x}_d = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{\xi}(t_0) \\ \boldsymbol{\xi}(t_f) \end{bmatrix} := \begin{bmatrix} \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \boldsymbol{x}_5 \end{bmatrix} \quad (5.4)$$

where \boldsymbol{x}_c and \boldsymbol{x}_d are collections of the continuous (infinite-dimensional) and discrete (finite-dimensional) optimization variables, and indexed variables \boldsymbol{x}_i are equivalent to the variable in the same row in the vector directly to the left in the same equation (e.g., $\boldsymbol{u} := \boldsymbol{x}_1$).

5.2.3.1 Linear Dynamics

A good place to start is with the dynamics expressed in Eqn. (5.1b) as it is a unique feature of DO. An appropriate choice for the state dynamics is a linear nonhomogeneous differential equation:

$$\mathbf{f} = \mathbf{f}^{QP} := \mathbf{A}(t)\boldsymbol{\xi}(t) + \mathbf{B}(t)\mathbf{u}(t) + \mathbf{G}(t)\mathbf{p} + \mathbf{d}(t) \quad (5.5)$$

where $\{\mathbf{A}, \mathbf{B}, \mathbf{G}\}$ are the linear time-varying (LTV) state/input/parameter matrices and $\mathbf{d}(t)$ is a disturbance.

Many authors consider different variations on Eqn. (5.5). The simplest form is that of a linear time-invariant (LTI) system: $\mathbf{f} = \mathbf{A}\boldsymbol{\xi} + \mathbf{B}\mathbf{u}$ [146, 147]. The next most common form is the LTV system without \mathbf{G} or \mathbf{d} [148]. The addition of the disturbance does appear in a number of formulations [104, 149–152]. No general formulations are seen which include \mathbf{p} , but this may be due to the limited number of problems where \mathbf{p} shows up linearly in \mathbf{f} . However, this demonstrates that even the simplest mixed parameter-control problems are not LQDO problems [25]. A more general form is $\mathbf{E}(t)\dot{\boldsymbol{\xi}} = \mathbf{f}^{QP}$, known as descriptor form [153, 154]. Here we will only consider that case where \mathbf{E} is nonsingular so it may be written as $\dot{\boldsymbol{\xi}} = \mathbf{E}^{-1}\mathbf{f}^{QP}$.

One may be tempted to integrate this differential equation and would arrive at the following:

$$\boldsymbol{\xi}(t) = \Phi(t, t_0)\boldsymbol{\xi}(t_0) + \int_{t_0}^t \Phi(t, \tau) (\mathbf{B}(\tau)\mathbf{u}(\tau) + \mathbf{G}(\tau)\mathbf{p} + \mathbf{d}(\tau)) d\tau \quad (5.6)$$

where $\Phi(t, \tau)$ is the state-transition matrix which describes the dynamics of the homogeneous system (i.e., the integral in Eqn. (5.6) is zero) [155]. The most general transition matrix is given by the Peano-Baker series. However, if \mathbf{A} is time-invariant, then the state-transition matrix is $e^{\mathbf{A}(t-\tau)}$. Some solution methods will utilize these properties, but in general, the differential equation is challenging to utilize directly.

5.2.3.2 Quadratic Objective Function

The QP objective function in Eqn. (5.2a) is a polynomial with degree two; therefore, we will choose a general quadratic cost functional containing appropriate elements of $\tilde{\mathbf{x}}$:

$$\mathcal{L} = \mathcal{L}^{QP} := \underbrace{\sum_{i=1}^5 \sum_{j=1}^5 \mathbf{x}_i^\top \mathbf{L}_{ij}(t) \mathbf{x}_j}_{H^{\mathcal{L}} = \tilde{\mathbf{x}}^\top \mathbf{L} \tilde{\mathbf{x}}} + \underbrace{\sum_{j=1}^5 \mathbf{l}_j^\top(t) \mathbf{x}_j}_{F^{\mathcal{L}} = \mathbf{l}^\top \tilde{\mathbf{x}}} + c^{\mathcal{L}}(t) \quad (5.7)$$

$$\mathcal{M} = \mathcal{M}^{QP} := \underbrace{\sum_{i=3}^5 \sum_{j=3}^5 \mathbf{x}_i^\top \mathbf{M}_{ij} \mathbf{x}_j}_{H^{\mathcal{M}} = \mathbf{x}_d^\top \mathbf{M} \mathbf{x}_d} + \underbrace{\sum_{j=3}^5 \mathbf{m}_j^\top \mathbf{x}_j}_{F^{\mathcal{M}} = \mathbf{m}^\top \mathbf{x}_d} + c^{\mathcal{M}} \quad (5.8)$$

where we require that $\mathbf{L}_{ij} = \mathbf{L}_{ji}^\top$ and $\mathbf{M}_{ij} = \mathbf{M}_{ji}^\top$ so that both are symmetric since \mathbf{H} needs to be symmetric. We note that the Mayer term only includes the discrete optimization variables because including any element of \mathbf{x}_c would not result in a scalar quantity¹⁵.

The Lagrange and Mayer terms in Eqns. (5.7)–(5.8) include all possible time-varying quadratic and linear terms in the most general expression. While most of the applications utilize a few select time-variant and time-invariant terms for their performance index [149, 150, 153, 154, 156–158], there are a few applications in the literature which include all the time-varying \mathbf{x}_c dependent terms in the linear and quadratic terms, but only includes $\xi(t_f)$ terms in linear and quadratic Mayer terms [148, 152, 159]. Constants in Lagrange and Mayer terms (c^L and c^M) are not widely used, but c^M term appears in a few references [153]. We also note that if there are no quadratic terms (i.e., $\mathbf{L} = \mathbf{0}$ and $\mathbf{M} = \mathbf{0}$), then the LQDO problem can be solved with linear programming (a special case of quadratic programming) [128].

5.2.3.3 Additional Linear Constraints

In most DO problems there are additional constraints that need to be enforced. In the context of LQDO, these can all be expressed as the following general linear equality and inequality constraint forms:

$$h = h^{QP} := \sum_{j=1}^5 \mathbf{Y}_j(t)^\top \mathbf{x}_j - \hat{Y}(t) = \mathbf{Y}^\top \tilde{\mathbf{x}} - \hat{Y} = 0 \quad (5.9)$$

$$g = g^{QP} := \sum_{j=1}^5 \mathbf{Z}_j(t)^\top \mathbf{x}_j - \hat{Z}(t) = \mathbf{Z}^\top \tilde{\mathbf{x}} - \hat{Z} \leq 0 \quad (5.10)$$

This representation of the constraints mixes the path and boundary definitions mentioned in Sec. 5.2.1. In Sec. 5.4.3.3 it will be shown that a straightforward procedure exists to determine which class (path or boundary) is most appropriate for a specified constraint.

As with the previous LQDO elements, there are a number of common constraint types seen in literature that are captured by these general linear constraints. There is a broad

¹⁵We can also note that since the time horizon is fixed in LQDO, \mathcal{M} is redundant since $\mathbf{M}_{ij}/(t_f - t_0)$ could be added to \mathbf{L}_{ij} for the appropriate indices. To facilitate more natural descriptions, all terms are considered.

class of boundary condition constraints that only contain the initial and final states [153] including prescribed initial conditions [104, 135, 146, 148, 152, 156, 159], prescribed final or terminal conditions [104, 146, 157, 158], and periodic conditions [25]. Initial and final state conditions need not be only equality constraints [157]. Mixed control-state constraints are also possible [104, 152, 153, 157, 159]. This type of constraint commonly includes only \mathbf{x}_c terms, but the whole collection $\tilde{\mathbf{x}}$ is possible and necessary such as when a parameter is used to represent maximum or absolute values (see Sec. 5.5.3 and Sec. 5.5.2).

There is a special subclass of Eqn. (5.10) which contains only single linear term:

$$\underline{\mathbf{x}}(t) - \mathbf{x}_j \leq \mathbf{0} \quad (5.11a)$$

$$\mathbf{x}_j - \bar{\mathbf{x}}(t) \leq \mathbf{0} \quad (5.11b)$$

This subclass is mentioned because it fits the form of Eqn. (5.2d) more so than either Eqn. (5.2b)–(5.2c). If $\underline{\mathbf{x}}$ and $\bar{\mathbf{x}}$ are not time-varying (see Ref. [146] where they can be time-varying), these constraints are commonly called saturation constraints or simple bounds [158].

5.2.3.4 Comparison to Similar Formulations

With all elements of the LQDO problem, we will briefly compare with some other common formulations found in the literature. Perhaps the most ubiquitous LQDO problem is the finite-horizon linear-quadratic regulator (LQR) problem [20]:

$$\min_{\mathbf{u}(t)} \quad \int_{t_0}^{t_f} (\boldsymbol{\xi}^T(t) \mathbf{L}_{22} \boldsymbol{\xi}(t) + \mathbf{u}^T(t) \mathbf{L}_{11} \mathbf{u}(t)) dt + \boldsymbol{\xi}^T(t_f) \mathbf{M}_{55} \boldsymbol{\xi}(t_f) \quad (5.12a)$$

$$\text{subject to: } \dot{\boldsymbol{\xi}}(t) - (\mathbf{A}\boldsymbol{\xi}(t) + \mathbf{B}\mathbf{u}(t)) = \mathbf{0} \quad (5.12b)$$

$$\boldsymbol{\xi}(t_0) - \boldsymbol{\xi}_0 = \mathbf{0} \quad (5.12c)$$

We note that the problem is time-invariant with no path constraints or mixed terms. This problem structure is still amenable to deriving a set of reasonably simple optimality conditions in the form of a boundary value problem [20, 104]. The optimality conditions for different variations on Prob. (5.12) have also been studied by Bryson and Ho [104]. These variations include an LTV system with $\boldsymbol{\xi}^T \mathbf{L}_{21} \mathbf{u}$ terms, exactly zero terminal error (i.e., $\boldsymbol{\xi}(t_f) = \mathbf{0}$), and the nonhomogeneous equation $\mathbf{f} = \mathbf{A}(t)\boldsymbol{\xi}(t) + \mathbf{B}(t)\mathbf{u}(t) + \mathbf{d}(t)$. Since these problem structures do not have path constraints, it is fairly straightforward to derive the optimality conditions.

Many model-predictive control (MPC) paradigms utilize Prob. (5.12) [160]. The following

additional constraints are also common in MPC formulations:

$$x(t) \in \mathcal{X}, \quad u(t) \in \mathcal{U}, \quad \forall t \geq 0 \quad (5.13)$$

where the sets \mathcal{X} and \mathcal{U} are polyhedra. These sets can be represented by sets of linear constraints of form Eqns. (5.9)–(5.10), or even as simple bounds (see Sec. 5.5.7). These polyhedra constraints are possible for more general forms of linear constraints, such as mixed input and state constraints [160].

An LQDO formulation with many similar elements is studied by Sideris and Rodriguez [159] with the notation slightly amended to be consistent with this work:

$$\min_{\mathbf{u}(t)} \int_{t_0}^{t_f} \left(\sum_{i=1}^2 \sum_{j=1}^2 \mathbf{x}_i^\top \mathbf{L}_{ij}(t) \mathbf{x}_j + \sum_{j=1}^2 \mathbf{l}_j^\top(t) \mathbf{x}_j \right) dt + \boldsymbol{\xi}^\top(t_f) \mathbf{M}_{55} \boldsymbol{\xi}(t_f) + \boldsymbol{\xi}^\top(t_f) \mathbf{m}_5 \quad (5.14a)$$

$$\text{subject to: } \dot{\boldsymbol{\xi}}(t) - (\mathbf{A}(t)\boldsymbol{\xi}(t) + \mathbf{B}(t)\mathbf{u}(t)) = \mathbf{0} \quad (5.14b)$$

$$\boldsymbol{\xi}(t_0) - \boldsymbol{\xi}_0 = \mathbf{0} \quad (5.14c)$$

$$h = \sum_{j=1}^2 \mathbf{Y}_j(t)^\top \mathbf{x}_j - \hat{Y}(t) = 0 \quad (5.14d)$$

Here we see an LTV system, an objective function with mixed and linear terms, and mixed linear equality conditions. However, there are no inequality constraints nor general boundary conditions. Han et al. [152] include inequality constraints and the disturbance, along with most of the problem elements in Prob. (5.14). From these select examples, we can see a diversity of formulations in the literature, and all fit with the given LQDO framework.

5.3 Approximate Solutions with Direct Transcription

The method used here to obtain approximate solutions to the LQDO problem is the previously mentioned direct transcription (DT) method. A collection of desirable properties make DT a strong candidate for use in solving the LQDO problem over other methods such as Pontryagin's maximum principle, numeric indirect methods, and sequential direct methods (see Sec. 3.3.1). Furthermore, DT methods frequently have the order-maintaining property previously described. After some preliminaries, a number of common DT methods are described in the context of their ability to approximate certain elements of LQDO problems.

5.3.1 Preliminaries

We start by creating a vector of discretized time values, \mathbf{t} , such that:

$$t_0 < t_1 < \cdots < t_{n_t-1} < t_{n_t} = t_f$$

where $N_t = n_t + 1$ is the number of discrete time points. These discrete values of t are also known as node points, and \mathbf{t} is termed the mesh [95, 96]. There are a variety of methods for determining the values of \mathbf{t} , but certain numerical schemes require a specific class of \mathbf{t} . The four mesh schemes considered here are arbitrary user-defined nodes, equidistant (ED) nodes, Legendre-Gauss-Lobatto (LGL) nodes [109, 161], and Chebyshev-Gauss-Lobatto (CGL) nodes (see Sec. C.4.1) [109, 162]. The time step for the k th segment is denoted $\Delta_k = t_{k+1} - t_k$, and the vector of time steps is Δ . The horizon length is denoted $\Delta = t_f - t_0$.

For clarity, function arguments may be shortened with the presumption that all time-varying quantities are evaluated at the specified time index. For example:

$$\mathbf{f}_k = \mathbf{f}(t_k) = \mathbf{f}(t_k, \boldsymbol{\xi}(t_k), \mathbf{u}(t_k), \mathbf{p})$$

Intermediary values of functions between node points will be required and are denoted with a bar:

$$\bar{\mathbf{f}}_k = \mathbf{f}(\bar{t}_k) = \mathbf{f}\left(\frac{t_k + t_{k+1}}{2}\right) \quad (5.15)$$

We define the following matrices that contain some discretized components of Prob. (5.1):

$$\begin{aligned} \Xi &= \begin{bmatrix} \boldsymbol{\xi}(t_0) \\ \vdots \\ \boldsymbol{\xi}(t_{n_t}) \end{bmatrix} = \begin{bmatrix} \xi_1(t_0) & \cdots & \xi_{n_\xi}(t_0) \\ \vdots & \ddots & \vdots \\ \xi_1(t_{n_t}) & \cdots & \xi_{n_\xi}(t_{n_t}) \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} \mathbf{u}(t_0) \\ \vdots \\ \mathbf{u}(t_{n_t}) \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1(t_0) & \cdots & \mathbf{u}_{n_u}(t_0) \\ \vdots & \ddots & \vdots \\ \mathbf{u}_1(t_{n_t}) & \cdots & \mathbf{u}_{n_u}(t_{n_t}) \end{bmatrix} \\ \mathbf{p} &= \begin{bmatrix} p_1 & \cdots & p_{n_p} \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} \mathbf{f}(t_0) \\ \vdots \\ \mathbf{f}(t_{n_t}) \end{bmatrix} = \begin{bmatrix} f_1(t_0) & \cdots & f_{n_\xi}(t_0) \\ \vdots & \ddots & \vdots \\ f_1(t_{n_t}) & \cdots & f_{n_\xi}(t_{n_t}) \end{bmatrix} \end{aligned} \quad (5.16)$$

where n_ξ is the number of states, n_u is the number of controls, and n_p is the number of parameters. Earlier both \mathbf{x} and \mathbf{X} were defined as the sets of infinite-dimensional and QP optimization variables, respectively. For convenience when describing the methods, x_i will denote a single variable (i.e., a single state, control, or parameter) and X_i will denote the vector containing its corresponding discretization described above. Finally, we will need to

concatenate matrix columns into a single column vector with the $\text{vec}(\cdot)$ function.

5.3.2 Defect Constraints

Following the same order as Sec. 5.2.3, we will begin with DT methods that approximate the linear dynamics in Eqn. (5.5). Defect constraints have the general form:

$$\zeta(\mathbf{t}, \Xi, \mathbf{U}, \mathbf{p}) = \mathbf{0} \quad (5.17)$$

There is a wide variety of methods to construct the defect constraints including pseudospectral [97, 109, 136, 162–164], multistage [97, 165–168], multistage with centers [169, 170], multistep [97, 165], central difference [137], zero-order hold [171, 172], general Hermite-differentiation [170, 173], block pulse function [174], Gegenbauer [175], and Fourier-Galerkin [176]. In this chapter, we focus on some of the most commonly used methods that are classified as either pseudospectral (PS) or single-step (SS) methods. It remains future work to integrate other DT methods into the general solution method if applicable (see Sec. 5.7.4).

5.3.2.1 Pseudospectral Methods

The PS methods used here are embodied as a literal form of Eqn. (5.1b), commonly called the differential form of the defect constraint [177]. An accurate representation of the dynamics is ensured by requiring the approximation for the state derivatives to be equal to the true derivative function values given by the dynamics in Eqn. (5.5). Our approximation for $\dot{\xi}$ will come from the derivative of an interpolating polynomial that represents ξ . The interpolating polynomial considered here is defined as:

$$\xi(t) \approx \mathbf{P}(t) = \sum_{k=0}^{n_t} \xi(t_k) \phi_k(t) \quad (5.18)$$

where ϕ_k are continuous basis polynomials that are typically constructed such that the interpolation property holds [137]. Therefore, the polynomial approximation is exact at the node points and an approximation at all other values of t . Efficiently calculating the derivative of this polynomial (among other considerations) typically requires a specific form of the mesh. Here, we utilize a scaled time horizon, $\tau \in [-1, 1]$, achieved through the following transformation:

$$\tau = \frac{2}{t_f - t_0} \mathbf{t} - \frac{t_f + t_0}{t_f - t_0} \quad (5.19)$$

This is a bijective function for $\mathbf{t} \rightarrow \boldsymbol{\tau}$ such that $\xi(t) = \xi(\tau)$ and $\dot{\xi}(t) = \frac{2}{\Delta} \frac{d}{d\tau} \xi(\tau)$. Therefore, the approximation of the state derivative at the node points is:

$$\dot{\xi}(t_k) \approx \frac{2}{\Delta} \frac{d}{d\tau} P(\tau_k) = \frac{2}{\Delta} \sum_{i=0}^{n_t} \mathbf{D}_{ki} \xi(t_i) \quad (5.20)$$

where the matrix \mathbf{D} is termed the differentiation matrix. This matrix depends only on the values of $\boldsymbol{\tau}$ and the type of interpolating polynomial, so we have an approximation for $\dot{\xi}$ that depends only on ξ . The defect constraints in matrix form are now constructed as the error between the derivative of \mathbf{P} and the true derivative function value:

$$\boldsymbol{\zeta} = \text{vec} \left(\frac{2}{\Delta} \mathbf{D} \mathbf{\Xi} - \mathbf{F} \right) = \mathbf{0} \quad (5.21)$$

which comprises the linear constraint we will use in Eqn. (5.2b).

The two particular PS schemes considered in this chapter are based on Lagrange basis polynomials. The first uses LGL nodes for $\boldsymbol{\tau}$ and is described in Refs. [109, 163, 164]. The second uses CGL nodes and is described in Refs. [109, 162, 164]. Section C.4.1 provides additional details on both PS schemes. Recent work has implemented PS methods as QPs for certain LQDO problems [178].

5.3.2.2 Single-Step Methods

This class of methods is constructed using information enclosed by t_k and t_{k+1} . The first fundamental theorem of calculus provides the general equation for this class of integration schemes:

$$\boldsymbol{\xi}(t_{k+1}) = \boldsymbol{\xi}(t_k) + \int_{t_k}^{t_{k+1}} \mathbf{f}(s, \boldsymbol{\xi}(s), \mathbf{u}(s), \mathbf{p}) ds \quad (5.22)$$

With our discretization scheme, the integral term only has values for state and control at the boundaries. Therefore, the SS methods considered here can be written in the following form:

$$\boldsymbol{\zeta}_k = \boldsymbol{\theta}_{1,k} \mathbf{u}_k^\top + \boldsymbol{\theta}_{2,k} \mathbf{u}_{k+1}^\top + \boldsymbol{\theta}_{3,k} \boldsymbol{\xi}_k^\top + \boldsymbol{\theta}_{4,k} \boldsymbol{\xi}_{k+1}^\top + \boldsymbol{\theta}_{5,k} \mathbf{p}^\top - \boldsymbol{\nu}_k = \mathbf{0} \quad (5.23)$$

where $k = 0, 1, \dots, n_t - 1$, and when combined into vector form, they form the linear constraint we will use in Eqn. (5.2b).

A popular family of SS methods is the Runge-Kutta method, which is a class of multistage methods [95]. Four common schemes are Euler forward (EF), trapezoidal (TR), Hermite-Simpson (HS), and 4th-order classical Runge-Kutta (RK4) (see Sec. C.4.2 for their general

formulas). Both EF and TR are commonly found in QP formulations [25, 95, 96] but in fact, HS and RK4 can be utilized to form QPs as well, specifically for the linear dynamic system of interest. The higher-order Runge-Kutta methods require stages at interior points on the time interval. For parity with EF and TR controls variables, we will assume the control is piecewise linear¹⁶:

$$\bar{\mathbf{u}}_k = \frac{\mathbf{u}_k + \mathbf{u}_{k+1}}{2} \quad (5.24)$$

We could have made the collection of $\bar{\mathbf{u}}_k$ as additional optimization variables, but the benefits of these higher-order methods will still be seen with the piecewise linear assumption. The Eqn. (5.23) coefficients for all four methods are then:

$$\text{Euler Forward (EF)} \left\{ \begin{array}{l} \theta_{1,k} = -\Delta_k \mathbf{B}_k \\ \theta_{2,k} = \mathbf{0} \\ \theta_{3,k} = -\mathbf{I} - \Delta_k \mathbf{A}_k \\ \theta_{4,k} = \mathbf{I} \\ \theta_{5,k} = -\Delta_k \mathbf{G}_k \\ \nu_k = \Delta_k \mathbf{d}_k \end{array} \right. \quad (5.25a)$$

$$\text{Trapezoidal Rule (TR)} \left\{ \begin{array}{l} \theta_{1,k} = -\frac{\Delta_k \mathbf{B}_k}{2} \\ \theta_{2,k} = -\frac{\Delta_k \mathbf{B}_{k+1}}{2} \\ \theta_{3,k} = -\mathbf{I} - \frac{\Delta_k \mathbf{A}_k}{2} \\ \theta_{4,k} = \mathbf{I} - \frac{\Delta_k \mathbf{A}_{k+1}}{2} \\ \theta_{5,k} = -\Delta_k \left(\frac{\mathbf{G}_k}{2} + \frac{\mathbf{G}_{k+1}}{2} \right) \\ \nu_k = \Delta_k \left(\frac{\mathbf{d}_k}{2} + \frac{\mathbf{d}_{k+1}}{2} \right) \end{array} \right. \quad (5.25b)$$

$$\text{Hermite-Simpson (HS)} \left\{ \begin{array}{l} \theta_{1,k} = -\Delta_k \left(\frac{\mathbf{B}_k}{6} + \frac{\bar{\mathbf{B}}_k}{3} + \frac{\Delta_k \bar{\mathbf{A}}_k \mathbf{B}_k}{12} \right) \\ \theta_{2,k} = -\Delta_k \left(\frac{\mathbf{B}_{k+1}}{6} + \frac{\bar{\mathbf{B}}_k}{3} - \frac{\Delta_k \bar{\mathbf{A}}_k \mathbf{B}_{k+1}}{12} \right) \\ \theta_{3,k} = -\mathbf{I} - \Delta_k \left(\frac{\mathbf{A}_k}{6} + \frac{\bar{\mathbf{A}}_k}{3} + \frac{\Delta_k \bar{\mathbf{A}}_k \mathbf{A}_k}{12} \right) \\ \theta_{4,k} = \mathbf{I} - \Delta_k \left(\frac{\mathbf{A}_{k+1}}{6} + \frac{\bar{\mathbf{A}}_k}{3} - \frac{\Delta_k \bar{\mathbf{A}}_k \mathbf{A}_{k+1}}{12} \right) \\ \theta_{5,k} = -\Delta_k \left(\frac{\mathbf{G}_k}{6} + \frac{2\bar{\mathbf{G}}_k}{3} + \frac{\mathbf{G}_{k+1}}{6} + \frac{\Delta_k \bar{\mathbf{A}}_k \mathbf{G}_k}{12} - \frac{\Delta_k \bar{\mathbf{A}}_k \mathbf{G}_{k+1}}{12} \right) \\ \nu_k = \Delta_k \left(\frac{\mathbf{d}_k}{6} + \frac{2\bar{\mathbf{d}}_k}{3} + \frac{\mathbf{d}_{k+1}}{6} + \frac{\Delta_k \bar{\mathbf{A}}_k \mathbf{d}_k}{12} - \frac{\Delta_k \bar{\mathbf{A}}_k \mathbf{d}_{k+1}}{12} \right) \end{array} \right. \quad (5.25c)$$

¹⁶Multistage methods with centers are also quite common and these methods use additional optimization variables for the controls in the interior of the interval [169, 170].

$$\left\{ \begin{array}{l}
\mathcal{F}_1(\mathbf{Q}) = \Delta_k \left(\frac{\bar{\mathbf{Q}}_k}{3} + \frac{\mathbf{Q}_k}{6} + \frac{\Delta_k \bar{\mathbf{A}}_k \mathbf{Q}_k}{6} + \frac{\Delta_k \bar{\mathbf{A}}_k \bar{\mathbf{Q}}_k}{12} + \frac{\Delta_k \mathbf{A}_{k+1} \bar{\mathbf{Q}}_k}{12} + \dots \right. \\
\quad \left. \frac{\Delta_k^2 \bar{\mathbf{A}}_k^2 \mathbf{Q}_k}{12} + \frac{\Delta_k^2 \mathbf{A}_{k+1} \bar{\mathbf{A}}_k \bar{\mathbf{Q}}_k}{24} + \frac{\Delta_k^3 \mathbf{A}_{k+1} \bar{\mathbf{A}}_k^2 \mathbf{Q}_k}{24} \right) \\
\mathcal{F}_2(\mathbf{Q}) = \Delta_k \left(\frac{\bar{\mathbf{Q}}_k}{3} + \frac{\mathbf{Q}_{k+1}}{6} + \frac{\Delta_k \bar{\mathbf{A}}_k \bar{\mathbf{Q}}_k}{12} + \frac{\Delta_k \mathbf{A}_{k+1} \bar{\mathbf{Q}}_k}{12} + \frac{\Delta_k^2 \mathbf{A}_{k+1} \bar{\mathbf{A}}_k \bar{\mathbf{Q}}_k}{24} \right) \\
\theta_{1,k} = -\mathcal{F}_1(\mathbf{B}) \\
\theta_{2,k} = -\mathcal{F}_2(\mathbf{B}) \\
\theta_{3,k} = -\mathbf{I} - \mathcal{F}_1(\mathbf{A}) - \mathcal{F}_2(\mathbf{A}) \\
\theta_{4,k} = \mathbf{I} \\
\theta_{5,k} = -\mathcal{F}_1(\mathbf{G}) - \mathcal{F}_2(\mathbf{G}) \\
\nu_k = \mathcal{F}_1(\mathbf{d}) + \mathcal{F}_2(\mathbf{d})
\end{array} \right. \quad (5.25d)$$

Classical
4th-order
Runge-Kutta
(RK4)

Another popular SS method is the zero-order hold (ZOH) [160]. This method assumes the control is piecewise constant, i.e., $\mathbf{u}(t) = \mathbf{u}(t_k)$ for $t \in [t_k, t_{k+1})$, allowing for $\mathbf{u}(t_k)$ to be brought outside of the integral in Eqn. (5.22) and no dependence on $\mathbf{u}(t_{k+1})$ in ζ_k . The Eqn. (5.23) coefficients for the ZOH method are then:

$$\left\{ \begin{array}{l}
\theta_{1,k} = - \int_{t_k}^{t_{k+1}} e^{\mathbf{A}(t_{k+1}-\tau)} \mathbf{B}(\tau) d\tau \\
\theta_{2,k} = \mathbf{0} \\
\theta_{3,k} = -e^{\mathbf{A}(t_{k+1}-t_k)} \\
\theta_{4,k} = \mathbf{I} \\
\theta_{5,k} = - \int_{t_k}^{t_{k+1}} e^{\mathbf{A}(t_{k+1}-\tau)} \mathbf{G}(\tau) d\tau \\
\nu_k = \int_{t_k}^{t_{k+1}} e^{\mathbf{A}(t_{k+1}-\tau)} \mathbf{d}(\tau) d\tau
\end{array} \right. \quad (5.26)$$

Zero-Order
Hold
(ZOH)

for the special case where \mathbf{A} is a time-invariant matrix. Therefore, the defect constraints are exact for the assumed control scheme. It is important to note that the control assumption is fundamentally different than what is used in the other SS and PS methods. This defect constraint method is frequently used in discrete-time problems and for model predictive control, typically as QPs [152, 160, 172]. To the author's knowledge, no software packages support easy comparison between the performance of the ZOH method and other continuous-time methods.

5.3.3 Objective Function Terms

Some of the general quadratic objective function terms in Eqns. (5.7)–(5.8) need to be approximated. The Mayer terms only depend on boundary values of the discretized components of Prob. (5.1), so it may be evaluated exactly by utilizing the appropriate optimization variables. The Lagrange terms require quadrature or approximate numerical evaluation of a definite integral [179]. Three different types of quadrature schemes are now discussed, and each is typically associated with a particular defect constraint approximation method.

5.3.3.1 Gaussian Quadrature

This type of quadrature scheme seeks to maximize the accuracy of the numerical integration by carefully choosing the node points in a specific time horizon (commonly, the previously mentioned scaled time horizon, $\tau \in [-1, 1]$). Then, the form of this quadrature scheme is:

$$\int_{-1}^1 \mathcal{L}(\tau) d\tau \approx \sum_{k=0}^{n_t} w_k \mathcal{L}(\tau_k) \quad (5.27)$$

where w_k are predetermined quadrature weights; note that $\mathcal{L}(\tau_k)$ appears linearly in the approximation. These weights depend only on the values of τ and the type of interpolating polynomial. For the PS method utilizing LGL nodes, these node points result in orthogonal collocation. Therefore, the quadrature approximation is extremely accurate, and is exact for polynomials up to degree $2n_t - 1$ [97]. For the values of these weights see Sec. C.4.1.1 or Refs. [109, 164].

5.3.3.2 Clenshaw-Curtis Quadrature

This quadrature scheme is specific for the PS method utilizing CGL nodes and has the same form as Eqn. (5.27). Although a Gaussian (G) quadrature scheme can be constructed for CGL nodes with Lagrange basis polynomials, it suffers from certain numerical issues [180]. Therefore, it is common to use Clenshaw-Curtis (CC) quadrature which is exactly accurate for polynomials up to degree n_t [180]. For the values of these weights see Sec. C.4.1.2 or Refs. [109, 180, 181].

5.3.3.3 Composite Quadrature

The final quadrature scheme will be motivated by the following transformation:

$$\dot{\xi}_0 = \mathcal{L}(t, \boldsymbol{\xi}, \mathbf{u}, \mathbf{p}) \text{ and } \xi_0(t_0) = 0, \text{ then } \int_{t_0}^{t_f} \mathcal{L}(t, \boldsymbol{\xi}, \mathbf{u}, \mathbf{p}) dt \equiv \xi_0(t_f) \quad (5.28)$$

This transformation adjoins an additional state variable ξ_0 with the dynamics equivalent to \mathcal{L} with an arbitrary initial value for the ODE [20]. Now the final value $\xi_0(t_f)$ can then be included as a Mayer term. We can subsequently apply the EF method to the ODE in Eqn. (5.28) and arrive at the following composite Euler forward (CEF) quadrature method:

$$\int_{t_0}^{t_f} \mathcal{L}(t) dt \approx \sum_{k=0}^{n_t-1} \Delta_k \mathcal{L}(t_k) \quad (5.29)$$

This is a composite quadrature method since we are using a set of points inside the time horizon to better approximate the definite integral [179]. Similar to Eqn. (5.27), the CEF method is linear in $\mathcal{L}(t_k)$. We can similarly derive the composite trapezoidal rule (CTR) using the TR method. Both CEF and CTR are quite commonly used with their corresponding defect constraint methods. In addition, the ZOH method typically utilizes CEF even though the controls are only specified as piecewise constant [160, 172].

Continuing with the same line of reasoning, we can use the HS method. The composite quadrature formula is then:

$$\int_{t_0}^{t_f} \mathcal{L}(t) dt \approx \frac{1}{6} \left(\Delta_0 \mathcal{L}_0 + \sum_{k=1}^{n_t-1} \left(4\Delta_k \bar{\mathcal{L}}_k + (\Delta_{k+1} - \Delta_k) \mathcal{L}_k \right) + \Delta_{n_t} \mathcal{L}_{n_t} \right) \quad (5.30)$$

Consequently, we need to approximate $\bar{\mathcal{L}}_k$. We can consider the integrand $\mathcal{L} = x_i(t)L(t)x_j(t)$ since \mathcal{L} has at a maximum quadratic terms. If x_i is a parameter or boundary value, then $x_i(\bar{t}_k)$ is the same throughout the time horizon. If x_i is a control, then Eqn. (5.24) defines piecewise linear controls. If x_i is a state, then the HS rule approximates $x_i(\bar{t}_k)$ as:

$$x_i(\bar{t}_k) = \frac{1}{2} (x_i(t_k) + x_i(t_{k+1})) + \frac{\Delta_k}{8} (f_i(t_k) - f_i(t_{k+1})) \quad (5.31)$$

However, there are some potential issues with directly using Eqn. (5.31). Unlike the CEF and CTR methods, the approximated state would create zeroth and first-order terms and formulas that depend on the particular class of x_i (these are not necessarily an erroneous approximation, but can complicate implementation). Here we pose an alternative implementation that still only includes second-order terms by assuming that the states are piecewise linear, same as the controls. In the context of Eqn. (5.31), this approximation becomes more accurate as Δ_k decreases and as $|f_i(t_k) - f_i(t_{k+1})|$ decreases. Now using the quadratic

integrand, the center term $\bar{\mathcal{L}}_k$ is:

$$\bar{\mathcal{L}}_k = x_i(\bar{t}_k)L(\bar{t}_k)x_j(\bar{t}_k) \quad (5.32a)$$

$$= \frac{\bar{L}_k}{4} (x_i(t_k)x_j(t_k) + x_i(t_k)x_j(t_{k+1}) + x_i(t_{k+1})x_j(t_k) + x_i(t_{k+1})x_j(t_{k+1})) \quad (5.32b)$$

We note that cross terms such as $x_i(t_k)x_j(t_{k+1})$ are present, so this is not a standard quadrature method [179]. Furthermore, since this requires an additional assumption on the states, we will term this quadrature method as the composite quadratic Hermite-Simpson (CQHS) method as it is primarily applicable to quadratic objective functions. To the author's knowledge, this method has not been utilized yet in the literature to construct the Hessian for LQDO problems; its accuracy will be discussed in Sec. 5.6. It remains future work to implement the QP form of the composite Hermite-Simpson method using Eqn. (5.31).

It is important to note the differences between the CEF, CTR, and CQHS methods by looking at the conditions for which exact integration occurs over each segment $[t_k, t_{k+1}]$:

CEF Exact if $\deg(x_i(t)L(t)x_j(t)) = 0$

CTR Exact if $\deg(x_i(t)L(t)x_j(t)) \leq 1$

CQHS Exact if $\deg(x_i(t)) \leq 1$, $\deg(x_j(t)) \leq 1$, and $\deg(x_i(t)L(t)x_j(t)) \leq 3$

where \deg yields the highest degree of its terms (with respect to t) when the polynomial is expressed in its canonical form consisting of a linear combination of monomials. So, comparatively, we see a higher-order of accuracy for CQHS. We can also try the same procedure using RK4, but this results in the same approximation scheme as the CQHS method since the linear approximation is used.

Finally, for quadratic terms with each term having time dependence, the composite quadrature schemes can be nicely summarized as indexed sparse matrices. These quadrature schemes with quadratic terms have the form:

$$\int_{t_0}^{t_f} x_i(t)L(t)x_j(t)dt \approx X_i^T H X_j \quad (5.33)$$

where H is a symmetric $N_t \times N_t$ matrix. Then each of the composite quadrature rules can be written as:

$$\text{CEF} \quad H(i, j) = \begin{cases} \Delta_i L_i & i = j \neq n_t \\ 0 & \text{otherwise} \end{cases} \quad (5.34a)$$

$$\text{CTR} \quad H(i, j) = \frac{1}{2} \begin{cases} \Delta_0 L_0 & i = j = 0 \\ \Delta_{n_t-1} L_{n_t} & i = j = n_t \\ \Delta_{i-1} L_i + \Delta_i L_i & i = j \neq \{0, n_t\} \\ 0 & \text{otherwise} \end{cases} \quad (5.34b)$$

$$\text{CQHS} \quad H(i, j) = \frac{1}{6} \begin{cases} \Delta_0 (L_0 + \bar{L}_0) & i = j = 0 \\ \Delta_{n_t-1} (\bar{L}_{n_t-1} + L_{n_t}) & i = j = n_t \\ \Delta_{i-1} \bar{L}_{i-1} & i - j = 1 \\ \Delta_i \bar{L}_i & i - j = -1 \\ \Delta_{i-1} (L_i + \bar{L}_{i-1}) + \Delta_i (\bar{L}_i + L_i) & i = j \neq \{0, n_t\} \\ 0 & \text{otherwise} \end{cases} \quad (5.34c)$$

These matrix formulas can also be utilized to derive the gradient and constant expressions for linear and constant objective function terms.

5.3.4 Additional Linear Constraints

The additional linear constraints in Eqns. (5.9)–(5.10) are handled by utilizing the discretized components of Prob. (5.1). Constraints with time dependence (i.e., path constraints) are approximated with a set of N_t constraints, one for each node point. However, the satisfaction of the path constraints at time points other than the node points is not guaranteed [96]. Boundary constraints only need one constraint for accurate representation as there is no issue with potential infeasibility between node points.

5.4 Automated Problem Generation

With the discussion of the general problem class for LQDO in Sec. 5.2 and DT methods in Sec. 5.3, we can now describe the automated problem generation procedure (APGP) for a variety of DT methods used to create a QP approximation. Unlike general NLDO, all elements of the approximated LQDO problem are associated with constant matrices in the QP, so there is no need for finite-differencing or automatic differentiation to compute the Hessian or Jacobian [97]. The matrices in the QP problem tend to be large sparse matrices. A sparse matrix is one in which many of the elements are zero [95]. In particular, the

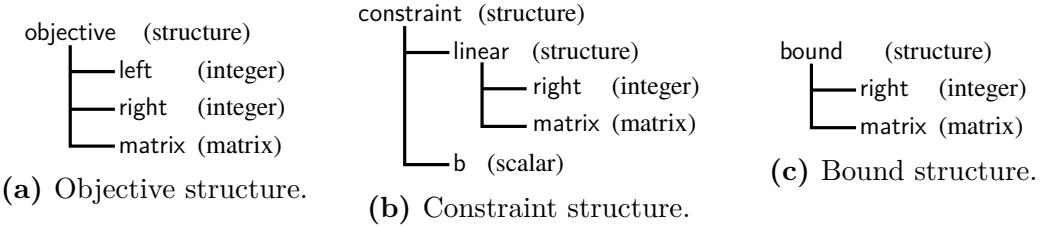


Figure 5.1: Structure definitions.

solution methods exhibit some form of a banded sparsity (and some localized dense blocks). From the given formulas, it is straightforward to determine the nonzero elements. This motivates the sequence-based approach used to generate the triples (row locations, column locations, and values) that define a sparse matrix.

5.4.1 Structure-Based Problem Definition

The first piece of the APGP is a natural description framework that describes LQDO problems. Here we use three different structure arrays to represent different problem elements, and they are outlined in a C-like notation in Fig. 5.1. These structures are an easy-to-use interface between the user and the APGP.

First, the structure definition in Fig. 5.1a captures all of the objective terms outlined in Eqns. (5.7)–(5.8). The objective structure is either \mathcal{L} and \mathcal{M} to account for Lagrange and Mayer terms, respectively. Each entry in the structure is another term in the objective function. The fields `left` and `right` can take on an integer value between 0 and 5, where 0 indicates a singleton dimension (useful for linear and constant terms) and the remaining values correspond to the index of the expanded set of optimization variables introduced in Eqn. (5.4). Finally, `matrix` is the potentially time-varying matrix of the appropriate size. To illustrate this notation, consider the following LQ objective function: $\int_{t_0}^{t_f} [\sin(t)u_1^2(t) + \xi_2(t_0)\xi_1(t)] dt + \xi_1(t_f) - 2\xi_2(t_f)$. The objective function can then be represented by the following¹⁷:

$$\int_{t_0}^{t_f} \sin(t)u_1^2(t)dt \iff \mathcal{L}(1).\text{left} = 1, \quad \mathcal{L}(1).\text{right} = 1, \quad \mathcal{L}(1).\text{matrix} = \sin(t) \quad (5.35a)$$

$$\int_{t_0}^{t_f} \xi_2(t_0)\xi_1(t)dt \iff \mathcal{L}(2).\text{left} = 4, \quad \mathcal{L}(2).\text{right} = 2, \quad \mathcal{L}(2).\text{matrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \quad (5.35b)$$

¹⁷This example assumes $n_\xi = 2$, $n_u = 1$.

$$\xi_1(t_f) - 2\xi_2(t_f) \iff \mathcal{M}\langle 1 \rangle.\text{left} = 0, \quad \mathcal{M}\langle 1 \rangle.\text{right} = 5, \quad \mathcal{M}\langle 1 \rangle.\text{matrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad (5.35c)$$

We now move onto the representation of the additional linear constraints in Eqns. (5.9)–(5.10). The **constraint** structure is either \mathcal{Y} or \mathcal{Z} to account for equality and inequality terms, respectively. The field **linear** can have multiple values to represent the summation needed for certain constraints. The fields **right** and **matrix** are analogous to their use in the objective function terms. The value for **b** is the potentially time-varying function. To illustrate this notation, consider the following linear constraints: $\xi_2(t_f) = 1$, $\xi_1(t) - 2\xi_2(t) \leq 0$, and $u_1(t) - p_1 \leq \sin(t)$. These linear constraints can then be represented by the following¹⁸:

$$\xi_2(t_f) = 1 \iff \begin{cases} \mathcal{Y}\langle 1 \rangle.\text{linear}\langle 1 \rangle.\text{right} = 5, & \mathcal{Y}\langle 1 \rangle.\text{linear}\langle 1 \rangle.\text{matrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ \mathcal{Y}\langle 1 \rangle.\text{b} = 1 \end{cases} \quad (5.36a)$$

$$\xi_1(t) - 2\xi_2(t) \leq 0 \iff \begin{cases} \mathcal{Z}\langle 1 \rangle.\text{linear}\langle 1 \rangle.\text{right} = 2, & \mathcal{Z}\langle 1 \rangle.\text{linear}\langle 1 \rangle.\text{matrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \\ \mathcal{Z}\langle 1 \rangle.\text{b} = 0 \end{cases} \quad (5.36b)$$

$$u_1(t) - p_1 \leq \sin(t) \iff \begin{cases} \mathcal{Z}\langle 2 \rangle.\text{linear}\langle 1 \rangle.\text{right} = 1, & \mathcal{Z}\langle 2 \rangle.\text{linear}\langle 1 \rangle.\text{matrix} = 1 \\ \mathcal{Z}\langle 2 \rangle.\text{linear}\langle 2 \rangle.\text{right} = 3, & \mathcal{Z}\langle 2 \rangle.\text{linear}\langle 2 \rangle.\text{matrix} = -1 \\ \mathcal{Z}\langle 2 \rangle.\text{b} = \sin(t) \end{cases} \quad (5.36c)$$

The **bound** structure is used to represent additional linear constraints that can be written as simple upper and lower bounds as in Eqn. (5.11). Then the structure is either \mathcal{UB} or \mathcal{LB} to account for upper and lower terms, respectively. The fields **right** and **matrix** are used analogously as the previous structures, with the exception that the values of **matrix** can be $\pm\infty$ to indicate no bounds when appropriate. To illustrate this notation, consider the following simple bounds: $u_1(t) \geq \sin(t)$ and $\xi_2(t) \leq \pi$. Then these bounds can be represented by the following¹⁹:

$$u_1(t) \geq \sin(t) \iff \mathcal{LB}\langle 1 \rangle.\text{right} = 1, \quad \mathcal{LB}\langle 1 \rangle.\text{matrix} = \sin(t) \quad (5.37a)$$

$$\xi_2(t) \leq \pi \iff \mathcal{UB}\langle 1 \rangle.\text{right} = 1, \quad \mathcal{UB}\langle 1 \rangle.\text{matrix} = \begin{bmatrix} \infty \\ \pi \end{bmatrix} \quad (5.37b)$$

¹⁸This example assumes $n_\xi = 2$, $n_u = 1$, $n_p = 1$.

¹⁹This example assumes $n_\xi = 2$, $n_u = 1$.

5.4.2 Procedure Overview

A schematic overview of the APGP is shown in Fig. 5.2. The user provides the problem structure using the notation defined in the previous section along with their choices for the mesh, quadrature, and defect constraints (these options are colored red). These options are related to the acronyms used in Sec. 5.3. We note that in addition to ED, LGL, and CGL mesh schemes, a user-defined mesh is also possible. First, the mesh is generated and a number of initialization tasks are performed. Next, the algorithms in the dashed box are presented in a modular format as the problem elements in LQDO are separate, and are utilized only when necessary as they approximate specific problem elements. Once all elements of the problem are approximated, the set of matrices that define the QP are passed to an appropriate QP solver to find the solution.

5.4.3 Algorithms

Here we briefly describe the algorithms with the full pseudocodes in Sec. C.1. The first algorithm outlines two functions used to get index sequence of the variable's location in both the continuous and discrete problems. The function GETCONTINDEX takes an integer between 0 and 5, used to denote a set of optimization variables in $\tilde{\mathbf{x}}$, and returns the sequence defining all the locations of that particular class of optimization variables in Eqn. (5.3). The second function, FINDQPINDEX, requires three inputs: 1) x is the specific number of the optimization variable that is selected, 2) $xtype$ is the same optimization variable classification as before, and 3) idx are the necessary indices to return. For example, if $n_u = 2$, $n_\xi = 3$, and $n_p = 1$, then x could be valued from 0 to 6 since there are 6 total optimization variables. Continuing with this example, if $x = 4$, $xtype = 2$, and $idx = 1$ to N_t , then we are requesting the indices of the discretization of $\xi_2(t)$, i.e., $\mathbf{X}(l) = \xi_2(t)$. Both of these functions will be useful when creating the objective function terms and additional linear constraints.

5.4.3.1 Objective Function Terms

There are three main algorithms are used to implement the five methods for approximating objective function terms in Sec. 5.3.3. The Lagrange terms are approximated by quadrature in Alg. C.3. The input is a structure \mathcal{L} of type **objective** and for each substructure, the relevant sequences are created. Both the GETCONTINDEX and GETQPINDEX are used to generate the appropriate row and column locations in the Hessian. While the CQHS method

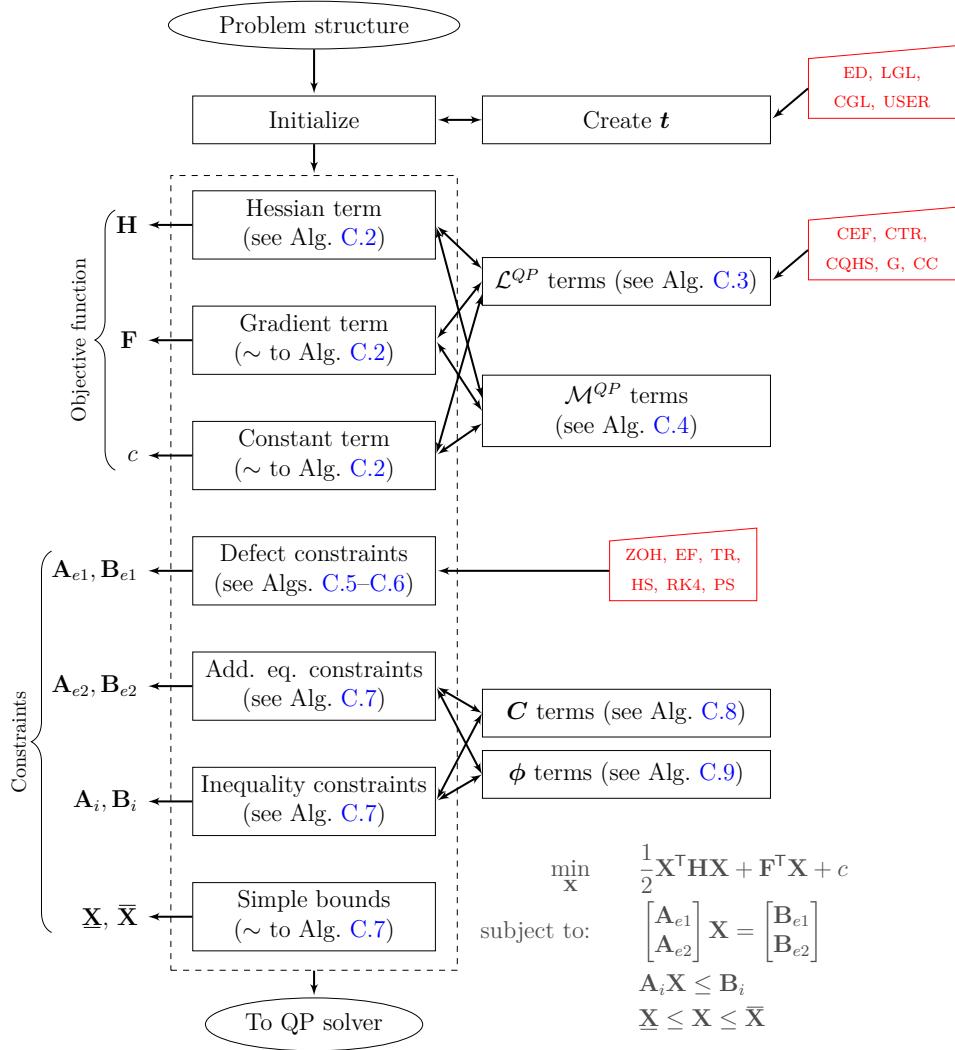


Figure 5.2: Overview of the automated problem generation procedure.

is shown specifically, the other quadrature schemes can readily be implemented with the same pseudocode with modifications to lines 25–26 pertaining to the values of the diagonal and off-diagonal entries. To visualize the process being used in Alg. C.3, the sparsity pattern for different \mathbf{L}_{ij} is shown in Fig. C.1. Each concatenation of \mathbf{l} on line 11 in the algorithm is the addition of a single diagonal in Fig. C.1b or column in Fig. C.1d.

Compact and efficient formulas for the quadrature methods are achieved by the creation of \mathbf{H} (and similar terms) and the use of the RSHIFT function. Consider the CTR method:

$$\begin{aligned}\mathbf{H} &= [\Delta_0 \quad \Delta_1 \quad \cdots \quad \Delta_{n_t-1} \quad 0] \\ \text{RSHIFT}(\mathbf{H}) &= [0 \quad \Delta_0 \quad \cdots \quad \Delta_{n_t-2} \quad \Delta_{n_t-1}] \\ \mathbf{Q} &= [\mathbf{A}(t_0) \quad \mathbf{A}(t_1) \quad \cdots \quad \mathbf{A}(t_{n_t-1}) \quad \mathbf{A}(t_{n_t})] \\ \text{CTR} &= [\Delta_0\mathbf{A}(t_0) \quad (\Delta_0 + \Delta_1)\mathbf{A}(t_1) \quad \cdots \quad (\Delta_{n_t-2} + \Delta_{n_t-1})\mathbf{A}(t_{n_t-1}) \quad \Delta_{n_t-1}\mathbf{A}(t_{n_t})]\end{aligned}\tag{5.38}$$

We see the formula $(\mathbf{H} \oplus \text{RSHIFT}(\mathbf{H})) \odot \mathbf{Q}/2$ produces the appropriate entries for the CTR method in Eqn. (5.34b).

The sequences for the Mayer terms are created with Alg. C.4, which is similar to the algorithm for the Lagrange terms. The approximation of Mayer terms is the same across the quadrature methods as previously mentioned.

To create the sparse Hessian matrix, Alg. C.2 takes the sequences from both Algs. C.3–C.4. The creation of the gradient and constant terms in Eqn. (5.2a) requires some minor modifications to Alg. C.2 but requires no modifications of Algs. C.3–C.4.

5.4.3.2 Defect Constraints

The algorithms for creating the defect constraints are shown in Alg. C.5 (SS methods) and Alg. C.6 (PS methods). We will first describe the approach used for the SS methods based on Eqns. (5.25)–(5.26). All required defect constraints for a particular state are generated inside the for-loop on line 3. In order, the appropriate entries in the matrix \mathbf{A}_{e1} are generated for the controls, states, and parameters based on their $\boldsymbol{\theta}$ term expressions (see Eqn. (5.23)). The sparsity pattern for this matrix is shown in Fig. C.4. The row and column locations are generated through the appropriate sequences based on the number of variables and node points. The values of the matrix entries are computed with element-wise formulas, similar to the approach used in the objective function terms (see Sec. 5.4.3.1). The indexing vector \mathbf{T} allows us to extract matrix values on different time grids shifted by one index which are present due to the shifting of k and $k+1$ values in the formulas. Using this approach, all

time-varying functions are evaluated only once on a particular time grid.

The `KRON` function[182] used on line 1 generates a matrix that efficiently implements \mathbf{I} in the equations (see its usage on lines 20–21). For the states and controls, values are calculated for the lower and upper diagonals in their respective blocks. For the controls, the lower diagonal values ($\boldsymbol{\theta}_1$ terms) are computed on line 10 and the upper diagonal ($\boldsymbol{\theta}_2$ terms) are computed on the following line. These diagonals are visualized in Figs. C.4c–d. Once the row, columns, and values for each state’s defect constraints are generated, they are combined into a single sparse matrix of size $n_\xi n_t \times n_X$. The disturbance in Eqn. (5.23) is the only part of the dynamics that appears in the \mathbf{B}_{e1} term. It is created in a similar fashion as the parameters (except there is only a single column in the resulting matrix). The algorithm presented specifically implements the TR method, but can be readily adapted to the other SS methods (the formulas for \mathbf{V} would need to be updated).

For PS methods (both with LGL or CGL nodes) in Alg. C.6, there is one more defect constraint per state than for an equivalent SS method. Overall, the procedure is very similar to the one for SS methods. The indexing vector \mathbf{T} is not needed since the matrix values only require the derivative function values at a single point in time in each row of the defect constraint (see the sparsity pattern in Figs. C.5b–d). Instead of \mathbf{K} , a PS method requires the differentiation matrix \mathbf{D} to be provided. This matrix is appropriately copied and shifted so that it coincides with the defect constraint rows and columns for the current state (see line 26). This is visualized with the block dense matrices in Fig. C.5a.

5.4.3.3 Additional Linear Constraints and Bounds

Both the additional inequality and equality constraints are created using Alg. C.7 taking in a structure of type `constraint`. For each substructure (i.e., for each different constraint), we need to determine if its type is either path or boundary. There are two conditions such that a constraint can be considered a path constraint: 1) any of the variable types in `right` are controls or states; 2) any of the matrices are time-varying (e.g., $\mathbf{Z}_3(t)$ or $\hat{\mathbf{Z}}(t)$).

If it is determined that the constraint is a path constraint, then Alg. C.8 is utilized to create the sparse matrix sequences. Otherwise, Alg. C.9 is utilized. These two algorithms are quite similar, the primary difference being how many constraints are created (N_t vs. one). Both utilize `GETCONTINDEX` and `GETQPIINDEX` in a similar fashion to the objective function terms in Sec. 5.4.3.1. After all sequences are created and combined, the sparse matrix is generated.

For simple bounds in Eqn. (5.11), the `bound` structure type is used. The same algorithms

are applicable but the entries are initialized as either $-\infty$ or ∞ depending on if it is a lower or upper bound. This ensures that unconstrained variables remain unconstrained.

5.5 Extensions

In this section, a number of extensions are described that still fit under the LQDO problem form using either some type of transformation or a simple application of the APGP.

5.5.1 Integral Constraints

Due to the equivalence between \mathcal{L} and \mathcal{M} , frequently the integral part of Eqn. (5.1a) is converted to an equivalent dynamic constraint [95, 104, 140]. Since the dynamic constraints are linear in LQDO, there can only be integral terms with a linear dependence. Therefore, we cannot utilize the transformation with quadratic objective terms and still have a QP. However we still can use this transformation if linear integral constraints are present. Consider the following inequality integral constraint [140]:

$$\int_{t_0}^{t_f} [\mathbf{I}^\top(t)\tilde{\mathbf{x}}] dt - \hat{I} \leq 0 \quad (5.39)$$

where $\mathbf{I}(t)$ is the integral matrix and \hat{I} is a scalar. We can add an equivalent state $\xi_{n_\xi+1} := \xi_+$ that has the same rate of change:

$$\dot{\xi}_+ = \mathbf{I}^\top(t)\tilde{\mathbf{x}} \quad (5.40)$$

Then we add the following initial value equality and final value inequality constraints:

$$\xi_+(t_0) = 0, \quad \xi_+(t_f) \leq \hat{I} \quad (5.41)$$

An similar procedure can be used for an equality constraint and additional states can be added for each linear integral constraint present in the problem.

5.5.2 Min-Max Objectives

A min-max objective function for Prob. (5.1) has the form [140]:

$$\Psi = \min_{\mathbf{x}} \max_{t \in [t_0, t_f]} \mathcal{E}(t, \boldsymbol{\xi}(t), \mathbf{u}(t), \mathbf{p}, t_0, \boldsymbol{\xi}(t_0), t_f, \boldsymbol{\xi}(t_f)) \quad (5.42)$$

where \mathcal{E} is the extremum function. We can introduce an additional parameter $p_{n_p+1} := p_+$ to transform this min-max problem into Mayer form by introducing an additional inequality constraint:

$$\mathcal{E}(t, \boldsymbol{\xi}(t), \mathbf{u}(t), \mathbf{p}, t_0, \boldsymbol{\xi}(t_0), t_f, \boldsymbol{\xi}(t_f)) - p_+ \leq 0 \quad (5.43)$$

and modifying the objective to:

$$\min_{\mathbf{x}, p_+} p_+ \quad (5.44)$$

This type of transformation is one of the common uses of parameters in LQDO problems assuming the extremum function is of the appropriate form. This appropriate form of \mathcal{E} (linear terms) was already discussed in Sec. 5.5.1 as we are transforming part of the objective to Mayer form. The structure of this transformation also allows for the extremum function to consist of multiple expressions such as the following:

$$\min_{\mathbf{x}} \max_{t \in [t_0, t_f]} \{u_1(t), \xi_1(t) + \xi_2(t)\} \quad (5.45)$$

which would be approximated with two additional constraints using the same parameter (i.e., $u_1(t) - p_+ \leq 0$ and $\xi_1(t) + \xi_2(t) - p_+ \leq 0$). The requirement only is that each expression in the maximization can be properly written as a linear inequality constraint bounded by the additional parameter.

5.5.3 Absolute Values in the Objective (and Constraints)

Consider the following finite-dimensional optimization problem:

$$\min_{\mathbf{z}_1, \mathbf{z}_2} |\mathcal{F}_1(\mathbf{z}_1)| + \mathcal{F}_2(\mathbf{z}_2) \quad (5.46a)$$

$$\text{subject to: } \mathbf{g}(\mathbf{z}_1, \mathbf{z}_2, |\mathcal{F}_1(\mathbf{z}_1)|) \leq \mathbf{0} \quad (5.46b)$$

Since the absolute value term is being minimized and contains a distinct set of the optimization variables, we can add an additional parameter $p_{n_p+1} := p_+$ and two inequality

constraints to arrive at the following equivalent problem:

$$\min_{\mathbf{z}_1, \mathbf{z}_2, p_+} p_+ + \mathcal{F}_2(\mathbf{z}_2) \quad (5.47a)$$

$$\text{subject to: } \mathbf{g}(\mathbf{z}_1, \mathbf{z}_2, p_+) \leq \mathbf{0} \quad (5.47b)$$

$$\mathcal{F}_1(\mathbf{z}_1) - p_+ \leq 0 \quad (5.47c)$$

$$-\mathcal{F}_1(\mathbf{z}_1) - p_+ \leq 0 \quad (5.47d)$$

An alternative reformulation utilizes the sum of two positive parameters for the absolute value. As with the previous two extensions, this extension introduces additional constraints that are linear with respect to the added parameter; so as long as \mathcal{F}_1 is linear, the additional constraints will be linear. For example, $\min |\xi_1(t_f) + \xi_2(t_f)|$. We can also have \mathcal{F}_1 in the Lagrange term, i.e., $\int_{t_0}^{t_f} |\mathcal{F}_1| dt$. Recall that the quadrature approximations for linear terms for all the methods in Sec. 5.3.3 are the sum of the products of a positive step size (or weight) and the value at every node point. Therefore, each point in the quadrature approximation is in the form of Eqn. (5.46a). However, we now need a parameter for each point in time to accurately capture that integral behavior. An example objective is $\min \int_{t_0}^{t_f} |u_1(t)| dt$.

Absolute value constraints with linear terms can readily be handled as well if they form a convex feasible region. For example, $|\xi_1(t) + \xi_2(t)| \geq 1$ would not be convex.

5.5.4 Output Tracking

Tracking a reference signal, either a set point or reference trajectories, can be a critical measure for control-system performance [104, 135]. Generally, the tracking error in the reference is computed against an output signal of the following form:

$$\mathbf{y}(t) = \mathbf{C}(t)\boldsymbol{\xi}(t) + \mathbf{D}(t)\mathbf{u}(t) + \mathbf{V}(t)\mathbf{p} + \mathbf{E}(t)\mathbf{d}(t) \quad (5.48)$$

where $\{\mathbf{C}, \mathbf{D}, \mathbf{V}, \mathbf{E}\}$ are the LTV output matrices. The output equation is similar to the dynamics in Eqn. (5.5), but no derivatives appear. Let us denote the reference signals as $\tilde{\mathbf{y}} = \tilde{\mathbf{y}}(t)$, then a suitable output tracking error quadratic objective function is:

$$\Psi = \int_{t_0}^{t_f} \left[(\mathbf{y} - \tilde{\mathbf{y}})^\top \mathbf{O}(t) (\mathbf{y} - \tilde{\mathbf{y}}) \right] dt \quad (5.49)$$

where \mathbf{O} is a symmetric weighting matrix. This objective function can be put into the same

form as Eqn. (5.7):

$$\Psi \equiv \int_{t_0}^{t_f} [\mathbf{x}^\top \mathbf{L}_O \mathbf{x} + \mathbf{l}_O^\top \mathbf{x} + c_O] dt \quad (5.50a)$$

$$\text{where: } \mathbf{L}_O = \begin{bmatrix} \mathbf{C}^\top \mathbf{O} \mathbf{C} & \mathbf{C}^\top \mathbf{O} \mathbf{D} & \mathbf{C}^\top \mathbf{O} \mathbf{V} \\ \mathbf{D}^\top \mathbf{O} \mathbf{C} & \mathbf{D}^\top \mathbf{O} \mathbf{D} & \mathbf{D}^\top \mathbf{O} \mathbf{V} \\ \mathbf{V}^\top \mathbf{O} \mathbf{C} & \mathbf{V}^\top \mathbf{O} \mathbf{D} & \mathbf{V}^\top \mathbf{O} \mathbf{V} \end{bmatrix} \quad (5.50b)$$

$$\mathbf{l}_O^\top = (2\tilde{\mathbf{y}}^\top \mathbf{O} + 2\mathbf{d}^\top \mathbf{E}^\top \mathbf{O}) \begin{bmatrix} \mathbf{C} \\ \mathbf{D} \\ \mathbf{V} \end{bmatrix}^\top, \quad c_O = \tilde{\mathbf{y}}^\top \mathbf{O} \tilde{\mathbf{y}} + \tilde{\mathbf{y}}^\top \mathbf{O} \mathbf{E} \mathbf{d} \quad (5.50c)$$

Therefore, output tracking can be easily incorporated into the previous quadratic objective. Bryson and Ho provide a boundary value problem (BVP) solution to a simpler output tracking problem [104].

5.5.5 Higher-Order Differential Equations

Equation (5.5) is a first-order linear differential equation. In general, there may be higher-order derivatives present. Consider the following third-order differential equation with $E_3(t) \neq 0$:

$$E_3(t) \ddot{\xi}_1 + E_2(t) \ddot{\xi}_1 + E_1(t) \dot{\xi}_1 = A(t) \xi_1 + B(t) u \quad (5.51)$$

To convert this into a first-order differential equation, we can add an additional state for each higher-order derivative present [140]. The dynamics of these state variables will be state of one order less. For the example above, we would have:

$$\begin{bmatrix} \dot{\xi}_1 \\ \dot{\xi}_2 \\ \dot{\xi}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ A/E_3 & -E_1/E_3 & -E_2/E_3 \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ B/E_3 \end{bmatrix} u \quad (5.52)$$

which is in the form of \mathbf{f}^{QP} and therefore, these types of higher-order differential equations can be readily handled by LQDO.

5.5.6 Control Rate Constraints

In some problems, we want to include the rate at which the control changes \dot{u} in our objective function or as a constraint such as [96]:

$$\dot{u}(t) \leq \dot{u}_{\max} \quad (5.53)$$

This situation can be handled in a similar manner as the higher-order differential equations in Sec. 5.5.5. We treat the control as another state variable $\xi_{n_\xi+1} := \xi_+ = u$ with dynamics of $\dot{\xi}_+ = \dot{u}$. Now \dot{u} is the independent input to the system. Both u and \dot{u} can be naturally handled using path constraints, objective function terms, etc. This transformation also works with higher-order control derivatives such as \ddot{u} .

5.5.7 Polyhedra Constraints

A common constraint in convex optimization is that the optimization variables must lie in some convex region. In general, this region cannot be directly handled in LQDO, but an approximation can be constructed with a polyhedron using the convex hull of a finite set of points on the region's boundary. This polyhedron is characterized by a set of linear inequality constraints and any point that is found to be feasible in the polyhedron would be feasible in the original region. An example is $u_1^2(t) + u_2^2(t) \leq 1$, which is an elliptical region that can be approximated (see Ref. [183]). Each additional linear constraint would be either a boundary or path constraint, so they would greatly add to the total number of constraints. Polyhedral regions are also used in piecewise affine systems (systems with different dynamics depending on the specific location in the state-input space) [160].

5.5.8 Bilevel Optimization and Minimum Time Problems

Bilevel optimization is where one optimization problem is embedded (nested) within another [98]. In some problems, the embedded problem has the form of LQDO, such as some nested co-design problem formulations as was discussed in Chapter 3 [18, 32]. For example, the outer loop may consist of variables that modify the matrices such as \mathbf{A} and \mathbf{B} (see Chapters 7 and 8 for examples that use LQDO and the APGP on a problem with this property).

Minimum time problems directly include t_0 and t_f in the objective function. For example, if $\mathcal{M} = t_f$, then it might seem possible to treat t_f as an optimization variable and have a linear term in the objective function. However, treating t_f as an optimization variable

results in nonlinear defect constraints and \mathcal{L} approximations. Observing Eqn. (5.21) and Eqn. (5.25), we see direct dependence on Δ in the defect constraint formulas. Therefore, minimum time problems *cannot* be solved directly with a QP. One solution approach for minimum time problems that still uses the LQDO framework is to solve a bilevel (nested) optimization problem. The outer loop solves the single variable problem optimization for t_f with the QP formulation (for a fixed value of t_f) as the inner-loop solution. If we expand to quadratically-constrained QPs (see Sec. 5.7.7), we can directly represent some minimum time problems.

5.5.9 Use in Nonlinear Optimal Control Problems

In some problems, there might be a quadratic objective but nonlinear dynamics or a nonlinear objective and linear dynamics. If certain problem elements fit the LQDO form, then the appropriate matrices can be generated (recall Fig. 5.2 where the algorithms for generating the QP matrices are modular). These matrices then can be combined with the nonlinear elements of the problem and solved with nonlinear programming solvers. Many of these solvers have special categorizations in order to efficiently leverage the problem structure such as for linear equality or inequality constraints.

5.6 Numerical Examples

In this section, we will use five numerical examples to demonstrate the efficacy of the proposed APGP. The first four examples have known solutions that can be used to compute the optimal trajectories and objective function value to high precision. Therefore, we can directly compare the different solution methods based on their deviation from the known solution. The errors are reported as local maximum/minimum values (i.e., as a forward-looking moving maximum/minimum) in order to discuss the converge behavior independent of fortuitous meshes (e.g., the nodes points happen to be exactly at the time value where a path constraint should change activity). In addition to comparing the absolute error, we will look at the time to create the QP with the APGP and the total QP time (creation plus solver time). Other performance metrics include local and global error, robustness to initial guess²⁰, and problem size or memory needed [170, 184]. Please see Refs. [133, 170, 184] for numerical

²⁰The chosen solver does not require an initial guess.

comparisons between some of the DT methods. It remains future work to perform these additional comparisons.

All tests were performed on a personal computer with an i7-6800K at 3.8 GHz (up to 12 threads available), 32 GB DDR4 3200 MHz RAM, WINDOWS 10 64-bit, and MATLAB 2017a. The QP solver used was the standard solver QUADPROG in MATLAB using the **interior-point-convex** algorithm [185]. All tolerances were set to 10ϵ , where ϵ is the machine precision number, in order to obtain the best solution possible for a particular QP. The complete set of codes from the APGP and examples are available at Ref. [186].

5.6.1 Example 1

5.6.1.1 Description

For the first example, we will consider the problem on pp. 166–167 from Ref. [104]:

$$\min_{u(t)} \frac{1}{2} \int_0^{t_f} u^2 dt \quad (5.54a)$$

$$\text{subject to: } \dot{\xi} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \xi + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \quad (5.54b)$$

$$\xi_1(0) = x_0, \quad \xi_2(0) = v_0, \quad \xi_1(t_f) = 0, \quad \xi_2(t_f) = 0 \quad (5.54c)$$

Although there are no path constraints, both the initial and final state values are fully constrained. As a result, this problem does not fit many traditional LQDO problem definitions such as Prob. (5.12). The structure-based problem description for this example is:

$$\mathcal{L}\langle 1 \rangle.\text{left} = 1, \quad \mathcal{L}\langle 1 \rangle.\text{right} = 1, \quad \mathcal{L}\langle 1 \rangle.\text{matrix} = 1/2 \quad (5.55a)$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (5.55b)$$

$$\mathcal{LB}\langle 1 \rangle.\text{right} = 4, \quad \mathcal{LB}\langle 1 \rangle.\text{matrix} = \begin{bmatrix} x_0 & v_0 \end{bmatrix}^\top \quad (5.55c)$$

$$\mathcal{LB}\langle 2 \rangle.\text{right} = 5, \quad \mathcal{LB}\langle 2 \rangle.\text{matrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}^\top \quad (5.55d)$$

$$\mathcal{UB}\langle 1 \rangle.\text{right} = 4, \quad \mathcal{UB}\langle 1 \rangle.\text{matrix} = \begin{bmatrix} x_0 & v_0 \end{bmatrix}^\top \quad (5.55e)$$

$$\mathcal{UB}\langle 2 \rangle.\text{right} = 5, \quad \mathcal{UB}\langle 2 \rangle.\text{matrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}^\top \quad (5.55f)$$

The MATLAB code is in Sec. C.3.1.

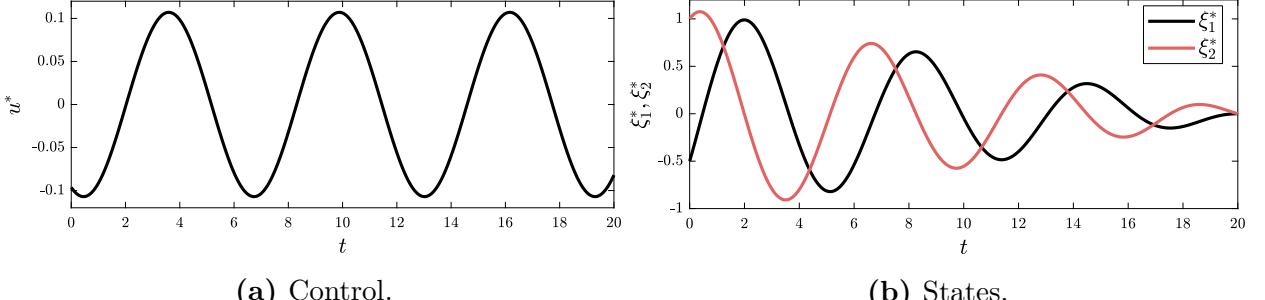


Figure 5.3: Solution for Example 1.

5.6.1.2 Solution

It can be shown that the control trajectory that minimizes the objective while satisfying the constraints is:

$$u^*(t) = -\frac{2}{t_f^2 - \sin^2(t_f)} \begin{bmatrix} x_0 \\ v_0 \end{bmatrix}^\top \begin{bmatrix} \sin(t_f - t) \sin(t_f) - t_f \sin(t) \\ -\cos(t_f - t) \sin(t_f) + t_f \cos(t) \end{bmatrix} \quad (5.56)$$

with an optimal objective function value of:

$$\Psi^* = \frac{t_f (v_0^2 + x_0^2) + 2t_f^2 v_0 x_0 - \cos(t_f) \sin(t_f) (v_0^2 - x_0^2)}{t_f^2 - \sin(t_f)^2} - 2v_0 x_0 \quad (5.57)$$

The problem parameters used are $t \in [0, 20]$, $x_0 = -1/2$, and $v_0 = 1$. With these parameter values, $\Psi^* = 0.059842$. The optimal trajectories for both the control and states is shown in Fig. 5.3.

5.6.1.3 Numerical Results

The convergence results for the eight tested schemes are shown in Fig. 5.4a (objective) and Fig. 5.4b (controls). The best scheme in terms of overall convergence rate was LGL-PS-G (7), and it is nearly linear. However, once the scheme's accuracy was near machine epsilon, an accuracy threshold was reached and even started to slowly diverge (perhaps due to small errors in the calculation of the differentiation matrix, weights, etc.). The next best scheme was CGL-PS-CC (8). It seemed to have a similar convergence rate as the other PS-based scheme, but it eventually achieves a sublinear rate of convergence until it reaches the precision threshold (for the objective value).

The SS-based schemes now remain. The best was ED-HS-CQHS (5), although ED-RK4-CQHS (6) was only slightly worse. These are the two schemes that utilize the proposed CQHS

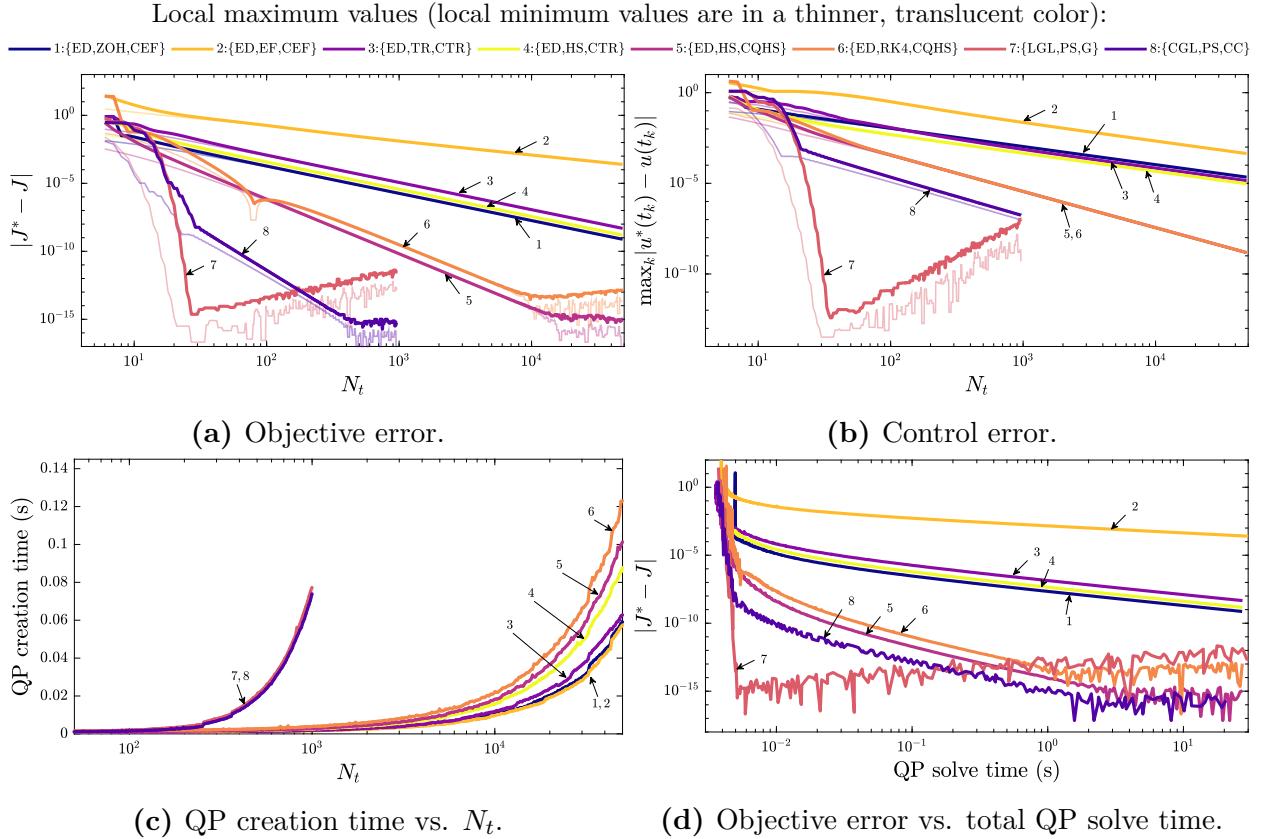


Figure 5.4: Numerical results for Example 1.

quadrature method. The convergence rate seemed to be sublinear, and an objective value accuracy threshold was reached for both schemes. The control error between these schemes was nearly identical. Next, perhaps surprisingly, was ED-ZOH-CEF (1). Even though this scheme assumed piecewise constant control, it performed better than some of the more classical SS-based schemes. Some of this accuracy may be due to the exact approximation of the objective function (i.e., only u^2 terms). ED-HS-CTR (4) was slightly better than ED-TR-CTR (3), indicating that the higher-order HS method did indeed provide additional accuracy for the same number of nodes. Finally, ED-ZOH-CEF (1) was the worst scheme tested.

The time to create the QP vs. N_t for each scheme is shown in Fig. 5.4c. Here we see two distinct groups: one for the PS-based schemes, and one for the SS-based schemes. The PS-based schemes take a longer amount of time for a specific N_t because the sparse matrices are much denser (cf. Fig. C.5 and Fig. C.4). The primary cost is the construction of the sparse matrices from the sequences. The SS-based schemes do vary in their creation time

with the schemes, with more matrix calculations and denser defect constraint matrices being slower to create. Therefore, we observe that (1) is the fastest and (6) is the slowest. All creation times for this problem are under 0.13 s, even for larger N_t .

A fairer comparison between the schemes considers the tradeoffs between accuracy and total solve time. The time to create and solve the QP vs. the error in the objective function is shown in Fig. 5.4d. The ordering is generally the same as the error plots, and (7) is clearly the preferred scheme for this problem. Schemes (5) and (6) are slightly more attractive as the computation time for a given error is only slightly slower than the PS-based schemes.

5.6.2 Example 2

5.6.2.1 Description

For the second example, we will consider the problem on pp. 120–122 from Ref. [104] and in Ref. [187]:

$$\min_u \quad \frac{1}{2} \int_0^1 u^2 dt \quad (5.58a)$$

$$\text{subject to: } \dot{\boldsymbol{\xi}} = \begin{bmatrix} \xi_2 \\ u \end{bmatrix} \quad (5.58b)$$

$$\xi_1(0) = 0, \quad \xi_1(1) = 0, \quad \xi_2(0) = 1, \quad \xi_2(1) = -1 \quad (5.58c)$$

$$\xi_1(t) \leq 1/9 \quad (5.58d)$$

This problem is similar to [Example 1](#) but, now has a path constraint. The structure-based problem description for this example is:

$$\mathcal{L}\langle 1 \rangle.\text{left} = 1, \quad \mathcal{L}\langle 1 \rangle.\text{right} = 1, \quad \mathcal{L}\langle 1 \rangle.\text{matrix} = 1/2 \quad (5.59a)$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (5.59b)$$

$$\mathcal{LB}\langle 1 \rangle.\text{right} = 4, \quad \mathcal{LB}\langle 1 \rangle.\text{matrix} = \begin{bmatrix} 0 & 1 \end{bmatrix}^\top \quad (5.59c)$$

$$\mathcal{LB}\langle 2 \rangle.\text{right} = 5, \quad \mathcal{LB}\langle 2 \rangle.\text{matrix} = \begin{bmatrix} 0 & -1 \end{bmatrix}^\top \quad (5.59d)$$

$$\mathcal{UB}\langle 1 \rangle.\text{right} = 4, \quad \mathcal{UB}\langle 1 \rangle.\text{matrix} = \begin{bmatrix} 0 & 1 \end{bmatrix}^\top \quad (5.59e)$$

$$\mathcal{UB}\langle 2 \rangle.\text{right} = 5, \quad \mathcal{UB}\langle 2 \rangle.\text{matrix} = \begin{bmatrix} 0 & -1 \end{bmatrix}^\top \quad (5.59f)$$

$$\mathcal{UB}\langle 3 \rangle.\text{right} = 2, \quad \mathcal{UB}\langle 3 \rangle.\text{matrix} = [\ell \quad \infty]^\top \quad (5.59g)$$

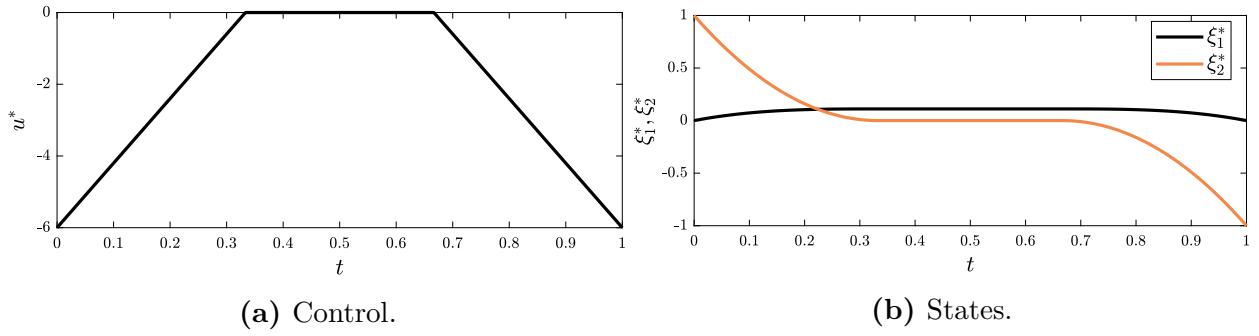


Figure 5.5: Solution for Example 2.

The MATLAB code is in Sec. C.3.2.

5.6.2.2 Solution

It can be shown that the control trajectory that minimizes the objective while satisfying the constraints when $0 < \ell \leq 1/6$ is:

$$u^*(t) = \begin{cases} -\frac{2}{3\ell} \left(1 - \frac{t}{3\ell}\right) & 0 \leq t < 3\ell \\ 0 & 3\ell \leq t < 1 - 3\ell \\ -\frac{2}{3\ell} \left(1 - \frac{1-t}{3\ell}\right) & 0 \leq t \leq 3\ell \end{cases} \quad (5.60)$$

with an optimal objective function value of:

$$\Psi^* = \frac{4}{9\ell} \quad (5.61)$$

A problem parameter value of $\ell = 1/9$ will be used, and with this value, $\Psi^* = 4$. The optimal trajectories for both the control and states are shown in Fig. 5.5.

5.6.2.3 Numerical Results

The convergence results for the eight tested schemes are shown in Fig. 5.6. All schemes, including the PS-based schemes, achieve similar sublinear convergence rates, and this is due to the path constraint. For the ED meshes, if $N_t - 1$ was a multiple of 3, then nodes values of $1/3$ and $2/3$ would be directly included in the mesh. Otherwise, the locations where the path constraint changes activity (in the true solution) would not be included. Therefore, there is slow convergence due to errors around these points. However, when $N_t = 4$ with ED-HS-CQHS (5) or ED-RK4-CQHS (6), all relevant quantities (states, control, and objective

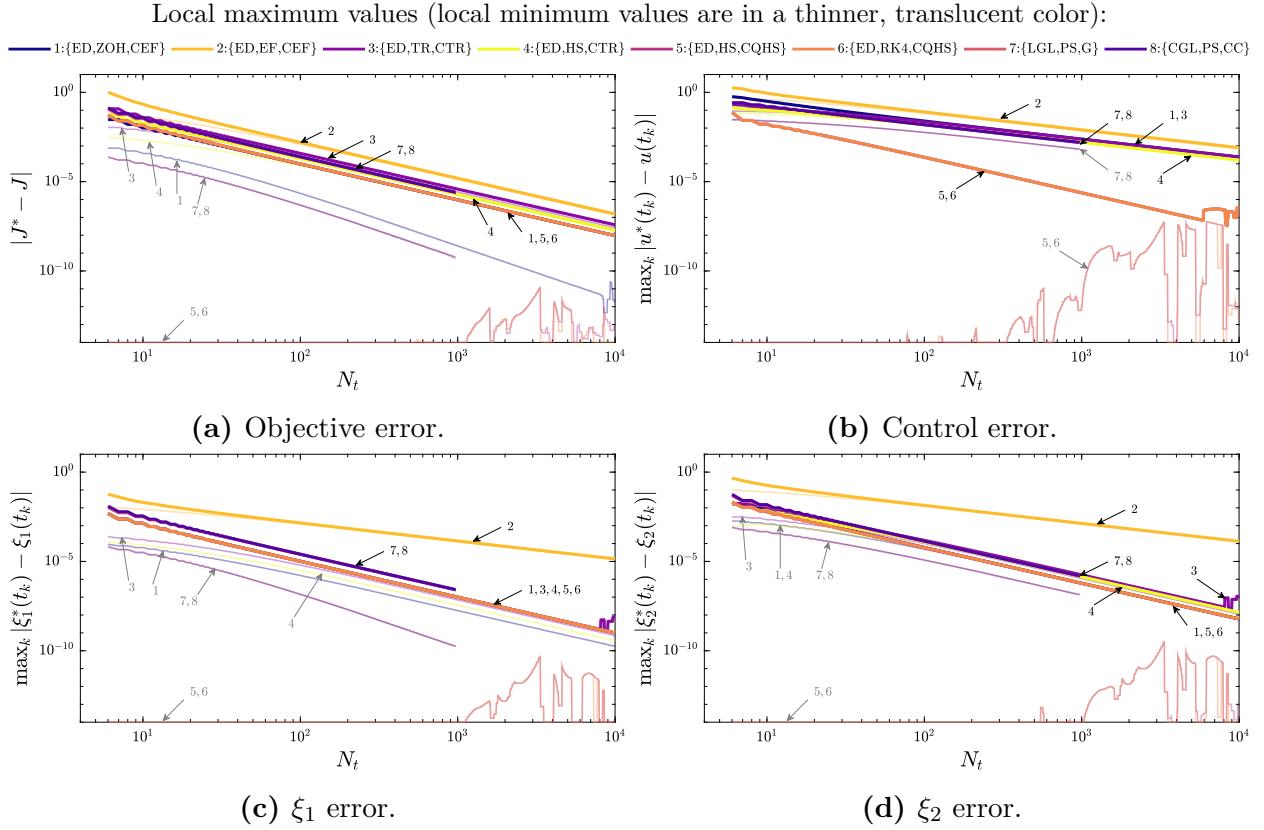


Figure 5.6: Numerical results for [Example 2](#).

value) are accurate within the machine precision! Such accuracy with a minimal number of node points was previously only possible with specifically constructed multiple-interval PS methods [109]. Since the optimal control policy is piecewise linear, the CQHS method is exactly accurate for u^2 terms. Furthermore, the states are piecewise quadratic and cubic and both the HS and RK4 methods exactly approximate these dynamics. Therefore, schemes (5,6) exactly represent the original problem.

Ignoring the favorable meshes and looking at the local worst errors, we still see (5,6) performing the best along with ED-ZOH-CEF (1) (except with respect to the controls). These are followed closely by the other methods except for ED-EF-CEF (2), which was the worst method again. These results for the PS-based schemes demonstrate the potentially issue with using a single global polynomial when path constraints are present [188, 189]. Multiple-interval PS methods would be more suitable for this type of problem (see Sec. 5.7.1).

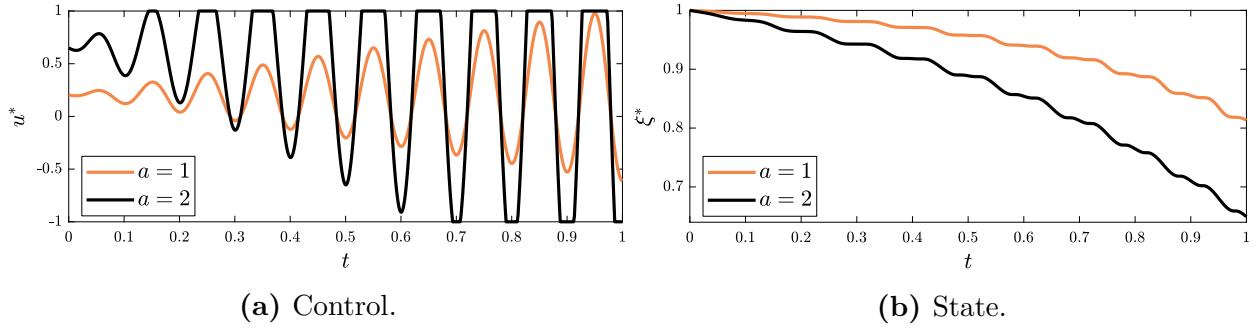


Figure 5.7: Solutions for Example 3.

5.6.3 Example 3

5.6.3.1 Description

For the third example, we will consider the problem on pp. 109–110 of Ref. [104]:

$$\min_{u(t)} \quad \frac{a^2}{2}[\xi(t_f)]^2 + \frac{1}{2} \int_0^{t_f} u^2 dt \quad (5.62a)$$

$$\text{subject to: } \dot{\xi} = b(t)u \quad (5.62b)$$

$$\xi(0) = \xi_0 \quad (5.62c)$$

$$|u(t)| \leq 1 \quad (5.62d)$$

This problem has time-varying matrices, path constraints, and both Lagrange and Mayer terms. The structure-based problem description for this example is:

$$\mathcal{M}\langle 1 \rangle.\text{left} = 5, \quad \mathcal{M}\langle 1 \rangle.\text{right} = 5, \quad \mathcal{M}\langle 1 \rangle.\text{matrix} = a^2/2 \quad (5.63a)$$

$$\mathcal{L}\langle 1 \rangle.\text{left} = 1, \quad \mathcal{L}\langle 1 \rangle.\text{right} = 1, \quad \mathcal{L}\langle 1 \rangle.\text{matrix} = 1/2 \quad (5.63b)$$

$$A = 0, \quad B = b(t) \quad (5.63c)$$

$$\mathcal{UB}\langle 1 \rangle.\text{right} = 4, \quad \mathcal{UB}\langle 1 \rangle.\text{matrix} = \xi_0, \quad \mathcal{LB}\langle 1 \rangle.\text{right} = 4, \quad \mathcal{LB}\langle 1 \rangle.\text{matrix} = \xi_0 \quad (5.63d)$$

$$\mathcal{UB}\langle 2 \rangle.\text{right} = 1, \quad \mathcal{UB}\langle 2 \rangle.\text{matrix} = 1, \quad \mathcal{LB}\langle 2 \rangle.\text{right} = 1, \quad \mathcal{LB}\langle 2 \rangle.\text{matrix} = -1 \quad (5.63e)$$

The MATLAB code is in Sec. C.3.3.

5.6.3.2 Solution

The optimal control is:

$$u^*(t) = -\text{sat} \left[a^2 b(t) \xi(t_f) \right] \quad (5.64)$$

where $\xi(t_f)$ is computed from the implication equation:

$$\xi(t_f) = \xi_0 - \int_0^{t_f} b(t) \text{sat} \left[a^2 b(t) \xi(t_f) \right] dt \quad (5.65)$$

The problem parameters used are $t_f = 1$, $\xi_0 = 1$, and $b(t) = t \cos(20\pi t) - 1/4$. Both $a = 1$ and $a = 2$ will be tested. For $a = 1$, $\Psi^* = 0.406759$ and $\xi^*(t_f) = 0.813517$. For $a = 2$, $\Psi^* = 1.150647$ and $\xi^*(t_f) = 0.649528$. The optimal trajectories for control and state for both values of a are shown in Fig. 5.7. With $a = 1$, the path constraints are never active, but with $a = 2$, the path constraints switch activity frequently.

5.6.3.3 Numerical Results

The convergence results for the eight tested schemes and both values of a are shown in Fig. 5.8. When $a = 1$ (where the path constraints are not active), the results are somewhat similar to Example 1. The best methods are the two PS-based schemes (7,8), but unlike the previous example, the gap between them is negligible. Although the SS-based schemes are converging to the true solution, the ranking of the schemes depends highly on the relative preference between the objective, control, or state errors. The two CQHS-based schemes have lower state error, but higher control error. ED-TR-CTR (3) has much lower control error, but higher state error. With respect to the objective error, (3) begins with lower error but there is a transition point around $N_t = 200$ where (5,6) exhibit less error. We also see ED-HS-CTR (4) performing worse than (3), differing from the previous examples.

With $a = 2$, the results are quite different with more sporadic convergence behavior (especially for the control). We observe that (3) is now the best scheme. Combining the results from both values of a , (3) appears to be the best option if high accuracy is required in both problem versions. These unexpected results might be explained by the fact that there is no \mathbf{A} in this example (a fairly uncommon property in LQDO). As a result, this example may prove to be a useful, challenging test problem in the future. As a result, this example may prove to be a useful, challenging test problem in the future. Fully understanding these results is left as future work.

Local maximum values (local minimum values are in a thinner, translucent color):

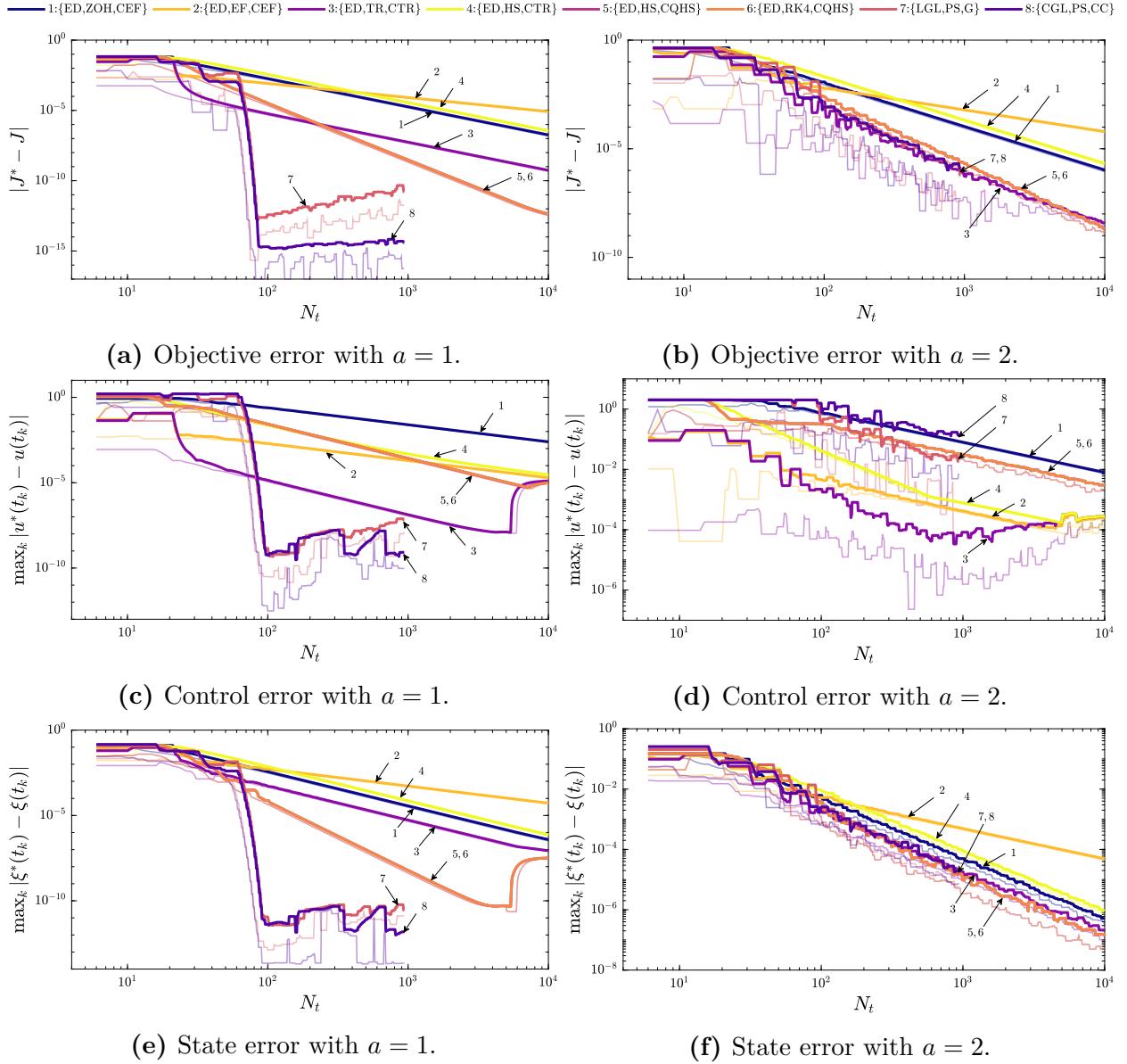


Figure 5.8: Numerical results for [Example 3](#).

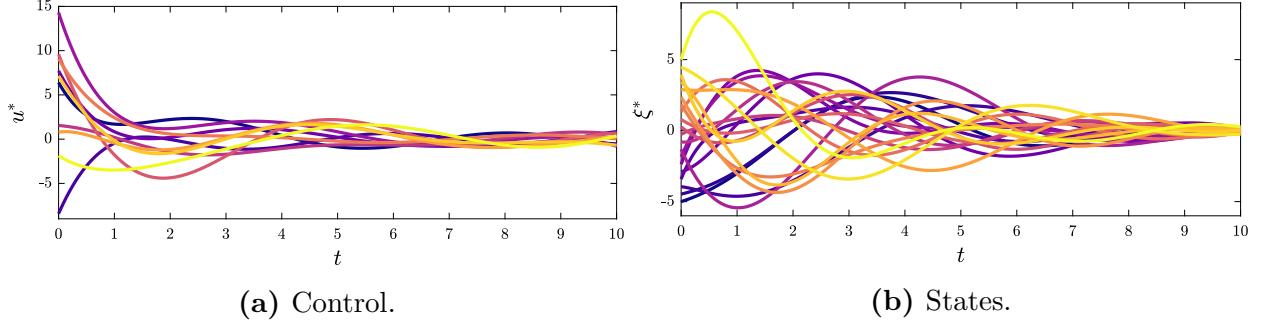


Figure 5.9: Solution for Example 4.

5.6.4 Example 4

5.6.4.1 Description

For the fourth example, we will consider the finite-horizon LQR problem in Prob. (5.12) [20, 104]:

$$\min_{\mathbf{u}(t)} \quad [\boldsymbol{\xi}^\top \mathbf{M} \boldsymbol{\xi}]_{t=t_f} + \int_{t_0}^{t_f} [\boldsymbol{\xi}^\top \mathbf{Q} \boldsymbol{\xi} + \mathbf{u}^\top \mathbf{R} \mathbf{u}] dt \quad (5.66a)$$

$$\text{subject to: } \dot{\boldsymbol{\xi}} = \mathbf{A}\boldsymbol{\xi} + \mathbf{B}\mathbf{u} \quad (5.66b)$$

$$\boldsymbol{\xi}(t_0) = \boldsymbol{\xi}_0 \quad (5.66c)$$

where \mathbf{M} and \mathbf{Q} are symmetric positive semidefinite and \mathbf{R} is symmetric positive definite. The structure-based problem description for this example is:

$$\mathcal{M}\langle 1 \rangle.\text{left} = 5, \quad \mathcal{M}\langle 1 \rangle.\text{right} = 5, \quad \mathcal{M}\langle 1 \rangle.\text{matrix} = \mathbf{M} \quad (5.67a)$$

$$\mathcal{L}\langle 1 \rangle.\text{left} = 2, \quad \mathcal{L}\langle 1 \rangle.\text{right} = 2, \quad \mathcal{L}\langle 1 \rangle.\text{matrix} = \mathbf{Q} \quad (5.67b)$$

$$\mathcal{L}\langle 2 \rangle.\text{left} = 1, \quad \mathcal{L}\langle 2 \rangle.\text{right} = 1, \quad \mathcal{L}\langle 2 \rangle.\text{matrix} = \mathbf{R} \quad (5.67c)$$

$$\mathcal{UB}\langle 1 \rangle.\text{right} = 4, \quad \mathcal{UB}\langle 1 \rangle.\text{matrix} = \boldsymbol{\xi}_0, \quad \mathcal{LB}\langle 1 \rangle.\text{right} = 4, \quad \mathcal{LB}\langle 1 \rangle.\text{matrix} = \boldsymbol{\xi}_0 \quad (5.67d)$$

The MATLAB code is in Sec. C.3.4.

5.6.4.2 Solution

The optimal control has the following form [20, 104]:

$$\mathbf{u}^* = -\mathbf{R}^{-1} \mathbf{B}^\top \mathbf{P} \boldsymbol{\xi} \quad (5.68)$$

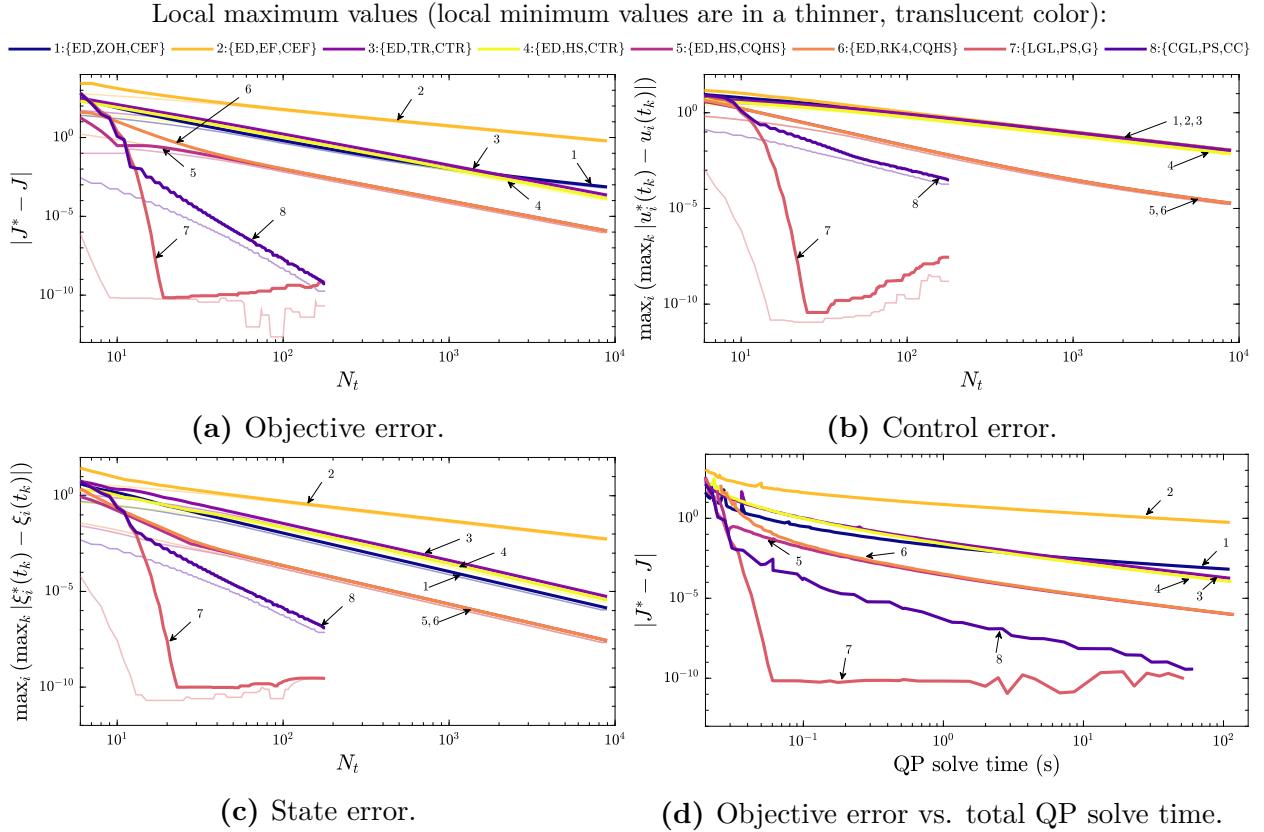


Figure 5.10: Numerical results for [Example 4](#).

where \mathbf{P} is symmetric positive semidefinite matrix that is a solution to the following differential equation and boundary condition:

$$\dot{\mathbf{P}} = -\mathbf{Q} - \mathbf{A}^\top \mathbf{P} - \mathbf{P} \mathbf{A} + \mathbf{P} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^\top \mathbf{P}, \quad \mathbf{P}(t_f) = \mathbf{M} \quad (5.69)$$

The specific problem parameters are shown in Sec. C.3.4 ($n_\xi = 20$ and $n_u = 10$ with generated matrices). The optimal trajectories for controls and states are shown in Fig. 5.9, and were determined by numerically solving the BVP problem with a relative error tolerance at 10^{-10} .

5.6.4.3 Numerical results

The convergence results for the eight tested schemes are shown in Fig. 5.10. The numerical results for this example are quite similar to [Example 1](#) (see Fig. 5.4). The PS-based methods (7,8) performed the best, followed by the CQHS-based methods (5,6). Then was (1,3,4) and finally (1) was the worst again. The primary discussion point for this example is the efficiency

at which the LQR problem was solved. The objective error vs. total QP solve time is shown in Fig. 5.10d. LGL-PS-G (7) with $N_t = 23$ took only 0.2 s to create and solve the QP with an accuracy in states, controls, and objective value at the tolerance used (10^{-10}) when generating the BVP solution in Sec. 5.6.4.2. These results demonstrate that DT approximations of the finite-horizon LQR problem can be a competitive solution strategy.

5.6.5 Example 5

5.6.5.1 Description

The final example is a constructed problem that will help demonstrate some of the problem elements and extensions in Sec. 5.5 not seen in the previous examples. The infinite-dimensional problem formulation is:

$$\min_{\mathbf{u}(t)} \quad \int_0^1 \left[u_1^2/10 + u_2^2/10 + u_1 \xi_1 + u_1 \xi_2 + 5 (\xi_2 - g(t))^2 \right] dt + \max_{0 \leq t \leq 1} \xi_3(t) \quad (5.70a)$$

$$\text{subject to: } \dot{\boldsymbol{\xi}} = \begin{bmatrix} -1 & 2 & 0 \\ 3 & -4 & 0 \\ 1 & 2 & -1 \end{bmatrix} \boldsymbol{\xi} + \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1/20 \end{bmatrix} \mathbf{u} \quad (5.70b)$$

$$\xi_1(0) = 2, \quad \xi_3(0) = 1/2 \quad (5.70c)$$

$$\xi_2(0) - \xi_2(1) = 0 \quad (5.70d)$$

$$\int_0^1 \xi_1(t) dt = 0 \quad (5.70e)$$

$$-\xi_1(t) + u_2(t)/12 \leq 0 \quad (5.70f)$$

$$\xi_2(t) \leq g(t) \quad (5.70g)$$

$$|u_2| \leq 10 \quad (5.70h)$$

where $5(\xi_2 - g(t))^2$ is an output tracking term (resulting in time-varying quadratic, linear, and constant objective function terms, see Sec. 5.5.4), $\max \xi_3(t)$ is a min-max objective term (that will be approximated with a parameter, see Sec. 5.5.2), $\xi_2(0) - \xi_2(1) = 0$ is a periodic constraint (that will be implemented as a linear equality constraint), $\int_0^1 \xi_1(t) dt = 0$ is an integral constraint (which will be approximated with an additional state, see Sec. 5.5.1), $-\xi_1(t) + u_2(t)/12 \leq 0$ is a mixed control-state path constraint, $\xi_2(t) \leq g(t)$ is a time-varying simple bound, and $|u_2| \leq 10$ is a linear absolute value constraint (that will be implemented with two constraints, see Sec. 5.5.3). For brevity, the structure-based implementation is only

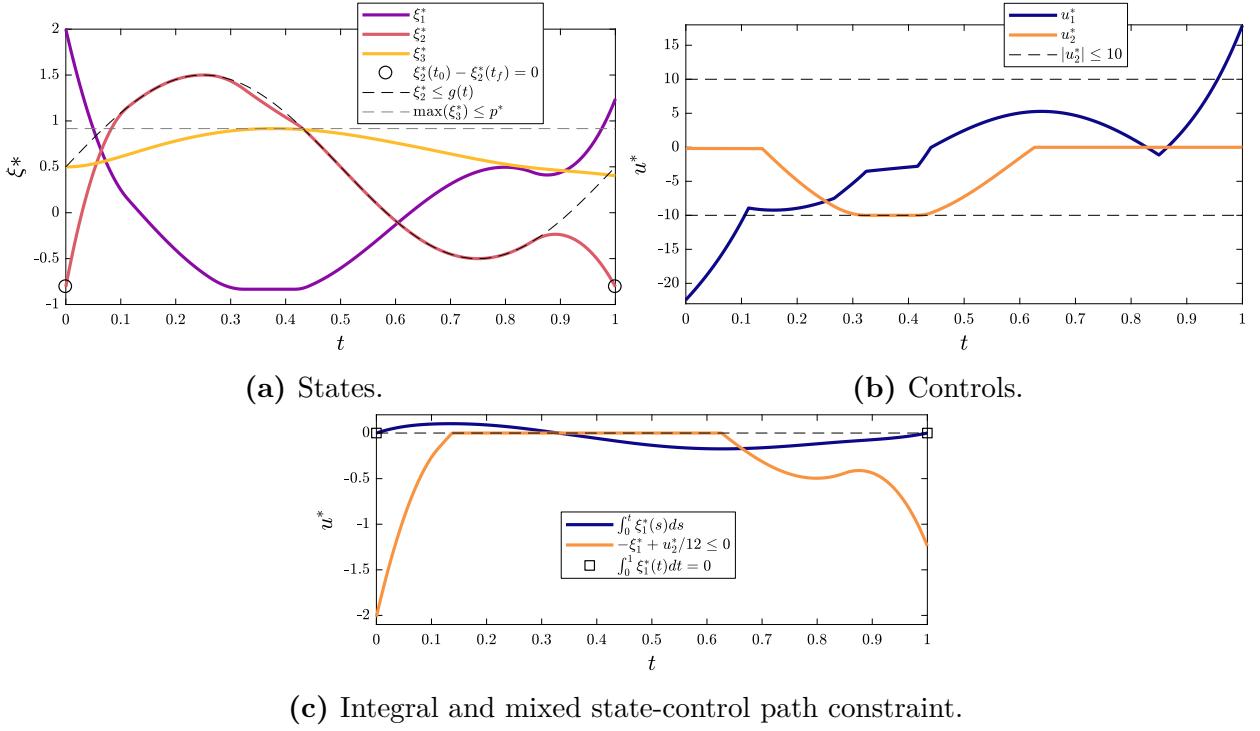


Figure 5.11: Solution for Example 5.

shown in the MATLAB code in Sec. C.3.5.

5.6.5.2 Solution

This example, like many LQDO problems, does not have a straightforward solution, so we can instead use DT to obtain an approximate solution. Here we set $g(t) = \sin(2\pi t) + 1/2$. The optimal trajectories for the states and controls are shown in Fig. 5.11 along with the trajectories for the integral and mixed control-state constraints. This solution was found using ED-HS-CQHS (5) with 5,000 node points and $\Psi^* = 6.368153$. All constraints are satisfied, and all the path constraints enter and exit activity during the time horizon.

5.7 Future Work

The proposed APGP and unified LQDO problem description can provide a strong basis for a number of future developments that could improve the effectiveness and the user experience.

5.7.1 Multiple-Interval Pseudospectral Methods and Multiphase Problems

The current implementation of the PS methods utilizes a single interpolating polynomial over the entire time horizon to approximate the states and controls. Since the interpolating polynomials are continuous basis functions, approximated quantities at the time values between node points may contain significant inaccuracy with any of discontinuity or non-smoothness imposed in the optimal function shape. In addition, since the particular PS method requires a specific form of the mesh (e.g., LGL or CGL meshes), which is stretched in the middle range, having enough resolution at certain location enforces excessive resolution in both end regions.

These problems could be addressed by implementing multiple intervals in our computational domain [109, 136, 190]. Each interval in the time horizon would contain an interpolating polynomial and continuity between the states would be ensured with the following continuity constraint:

$$\boldsymbol{\xi}^{(i-1)}(t_f^{(i-1)}) = \boldsymbol{\xi}^{(i)}(t_0^{(i)}), \quad \text{where: } t_f^{(i-1)} = t_0^{(i)} \quad (5.71)$$

which is a linear equality constraint; thus, is possible in LQDO. This multiple-interval approach can also be implemented along with the *hp*-adaptive mesh refinement technique described the following section [190–192].

Conceptually similar, multiphase problems could be proposed [136]. In a multiphase problem, various continuity constraints (which need not be the same form as Eqn. (5.71)) are present to ensure consistency across the phases. The problem elements between the phases may be different (e.g., different constraints present or the dynamics change). Unlike the multiple-interval approximation method, multiphase problems can be solved with all the methods listed here.

5.7.2 Mesh Refinement

The accuracy assessments in Sec. 5.6 were performed on problems with known exact solutions. However, for many LQDO problems, such solutions are unavailable (and is a primary reason for using DT). Resolution of mesh nodes and location of each node affect the overall solution accuracy and mesh refinement (iteratively solving the problem on a specific mesh and then updating the mesh with changes in the mesh resolution and node locations) is a viable technique for obtaining high accuracy solutions to continuous-time dynamic optimization problems [169, 190]. Such techniques are particularly useful when there is discontinuity

or non-smoothness in the solution (e.g., when path constraints change activity).

There are two common types of mesh refinement: h - and p -adaptive methods [193]. For the SS methods, the h -adaptive methods could be employed to minimize the local and global discretization error [169, 194, 195]. For the multiple-interval PS methods, the combined hp - or ph -adaptive methods could be applied to get the benefit of spectral convergence within local time interval while accommodating discontinuities between neighboring time intervals [189, 190, 192, 196]. Since mesh refinement is performed offline (not when solving the QP), it can readily be incorporated into the APGP.

5.7.3 Costate Approximation and First-Order Necessary Conditions

Alternative methods to DT are the indirect methods which derive a set of first-order necessary conditions for optimality for the DO problem (e.g., these conditions were used to derive the BVP in [Example 4](#)). Even though there are a number of challenges associated with using indirect methods [25, 95, 96, 104], it still can be useful to utilize the concepts from the indirect methods. For example, the costate variables are the time-varying multipliers associated with the dynamics in the augmented Hamiltonian. Similar to the states and controls, these have optimal values as well. Errors in the costates can be assessed in a similar manner to states and controls to help determine the convergence properties of each of the methods. In addition, computing the Hamiltonian given the DT solution can help verify the quality of the solution.

Determining the values of the costates and other multipliers is done through a mapping from the Karush-Kuhn-Tucker (KKT) multipliers of the finite-dimensional optimization problem [111, 177]. Estimations of costate have been studied for finite and infinite time horizons with direct optimal control methods [111, 177, 197, 198]. Approximating costates with discontinuous trajectories requires a jump condition between discontinued segments [197, 198] and is especially important when the multiple intervals are considered for problems with discontinuous dynamic behavior, described in Sec. [5.7.1](#). Various dynamic optimization software tools have provided these estimates [136, 137]. Computing these estimates for LQDO would be no different than the general NLDO problems and might, in fact, be simpler due to the structured form of the objective and dynamics.

5.7.4 Additional Methods

A large number of methods to construct the defect constraints were listed in Sec. [5.3.2](#). Only a small subset of these available methods have been implemented, and it remains future work

to implement (as long as they are order-maintaining methods) and assess the effectiveness of these methods in LQDO. Different quadrature schemes may also be considered, such as the true HS quadrature scheme that uses Eqn. (5.31).

5.7.5 Customized QP Solvers

The numerical results in Sec. 5.6 were found using one solver available in MATLAB. There are a number of other QP solvers available (e.g., `OOQP` [199] and `CVX` [200]) that may be more suitable for this class of problems (both in computational efficiency and convergence). An effective solver should be able to handle large matrices, the specific sparsity patterns, and ill-conditioned matrices. Perhaps a custom QP solver could be created that handles this type of ill-conditioned problems [201–204]. Some recent QP methods have used LQDO problems with specific DT methods as one of their test problems [203, 204].

Some of these numerical difficulties were illustrated in the results using the LGL-PS-G(7) scheme. This scheme typically exhibited a tendency of divergence after exponential convergence up to a specific polynomial order. A rapid growth of condition number along with a growth of the differential operator order causes this instability, which makes the problem nonconvex and may sacrifice the expected spectral accuracy of the solution [205, 206].

5.7.6 Scaling

It has been established that properly-scaled optimization problems are easier to solve than a poorly-scaled problem [97]. In general, this is embodied by the constraints and optimization variables being close to $\mathcal{O}(1)$ [207]. Affine transformations of both the optimization variables and time horizon can be utilized to accomplish this task [97, 115]. An example of this type of transformation is the similarity transform applied to linear systems [155]. The constraints can be directly scaled by the row norms of the matrices [97]. Systematic preconditioning methods have been developed for linear constraints [124, 125]. Properly-scaled defect constraints are particularly important to ensure the accuracy of the approximation [115]. Using these concepts, among other techniques, automatic scaling procedures have been developed for DO problems [97].

5.7.7 Quadratically-Constrained Quadratic Programs

A more general class of optimization problems that includes QPs as a special case are quadratically-constrained quadratic programs (QCQPs) [128]. The objective is the same as in Eqn. (5.2a), but we now include constraints with up to quadratic dependence:

$$\frac{1}{2} \mathbf{X}^T \mathbf{P}_e \mathbf{X} + \mathbf{A}_e \mathbf{X} = \mathbf{B}_e \quad (5.72a)$$

$$\frac{1}{2} \mathbf{X}^T \mathbf{P}_i \mathbf{X} + \mathbf{A}_i \mathbf{X} \leq \mathbf{B}_i \quad (5.72b)$$

In general, a QCQP is an NP-hard problem [208]. However, if all inequality constraints are convex and only linear equality constraints are present, then the QCQP can be solved with semidefinite programming or second-order cone programming [128]. Modifying the algorithms in Sec. 5.4 to generate QCQPs and understanding the structural properties of the generated matrices is future work. There are many interesting items that could be represented with QCQPs, including quadratic terms in the dynamics [209], MTPs, simple co-design problems (e.g., $\dot{\xi} = k\xi$, where k is also an optimization variable) [32], power and energy terms (e.g., ξu) [25, 133], and the conversion of quadratic Lagrange terms to Mayer form as quadratic equality constraints (discussed in Sec. 5.5.1).

5.8 Summary

In this chapter, a unified framework for solving general linear-quadratic dynamic optimization (LQDO) problems was proposed. This class of dynamic optimization problems contains problem elements such as a linear nonhomogeneous differential equation, quadratic objective function terms, and additional linear constraints where the optimization variables include the controls, states, parameters, initial state values, and final states values.

A class of numerical methods known as direct transcription (DT) was utilized to find approximate solutions to the LQDO problem. The DT methods parameterize both the state and control trajectories and include them as optimization variables. A large number of equality constraints (termed defect constraints) are used to ensure feasible dynamics. Both pseudospectral (PS) and single-step (SS) methods are utilized to construct the defect constraints. A variety of SS methods are implemented including Euler forward (EF), trapezoidal rule (TR), Hermite-Simpson (HS), 4th-order classical Runge-Kutta (RK4), and zero-order hold (ZOH). HS and RK4 are frequently utilized on NLDO problems, but rarely with LQDO. ZOH is a commonly used method, particularly within the MPC framework. A number of

quadrature schemes are implemented including a new composite quadratic Hermite-Simpson (CQHS) method. This method is derived in a similar manner as the composite HS method, but uses linear interpolation between node points for each term in the quadratic objective function.

An automated problem generation procedure (APGP) is fully outlined that makes it relatively straightforward to obtain a DT solution to an LQDO problem. A structure-based scheme is used to represent the problem. The algorithms for efficiently generating the sequences that define the sparse matrices is also described. Full MATLAB codes are available at Ref. [186]. Five examples are shown to demonstrate the efficacy of the APGP. Including a variety of DT methods allowed for direct comparisons between the methods. The PS-based methods had extremely fast convergence in problems with no path constraints, and had a smooth optimal solution in general. When the nonsmoothness was present in the optimal solutions, the higher-order SS methods performed better, with the HS and RK4 being the best. The CQHS-based methods generally performed as good as or better than the other SS methods, demonstrating the relative effectiveness of the new quadrature scheme.

A number of extensions are described including integral constraints, min-max objectives, absolute values, output tracking, control rate constraints, and polyhedra constraints, demonstrating that a diverse set of problems fit under the LQDO framework. There are a number of methods and features that can be implemented in the future including multiple-interval PS methods, multiphase problems, mesh refinement, costate approximation, additional defect and quadrature methods, customized QP solvers, scaling, and quadratically-constrained QPs.

Chapter 6

Case Study: Design of Passive Analog Circuits²¹

“It will be remarked that no attention is paid to the actual values of the resistances, but only to the forms in which they can be combined. The enumeration of the forms of combinations of a given number of resistances is of considerable interest.”

P. A. Macmahon [9]

6.1 Introduction

Synthesis of analog electric circuits is a complex and resource-intensive task. Circuit-level synthesis involves two major attributes: 1) the topology (i.e., what components are present and how they are connected or the circuit structure) and 2) sizing (i.e., the selection of the component values). Both attributes are required for a fully-defined circuit. While there exist many computer-aided tools for performing network analysis and simulation [210] as well as circuit sizing [211], there is still a need for high-quality synthesis methods that perform well on a variety of circuit synthesis problems.

Any synthesis method must navigate the immense potential design space present in circuit synthesis design problems [212]. Often this vast design space leads to the reliance on experts and domain-specific knowledge [23]. Alternatively, formal design methodologies have been developed for specific problem classes (e.g., linear frequency-domain filters) [213]. While these human-focused and specific formal approaches can be suitable for some applications, they often fail at finding desirable solutions for unfamiliar and complex design problems. To address some of these issues, there has been considerable research to try to leverage existing design knowledge by formally incorporating heuristics [214] and knowledge bases [215] into various tools.

²¹Elements of this chapter are based on work completed in Ref. [12].

More recent efforts have moved toward methods with minimal initial design knowledge [23, 37, 210, 211, 216]. Methods that adopt this principle can better generate truly novel topologies and nonexperts can more easily parse the required inputs for the automation tool [37, 210, 211, 213, 217]. Both evolutionary computation [23, 37, 210, 211, 213, 216–218] and simulated annealing [219] are concepts that have been utilized to generate and select fully-defined circuits with minimal knowledge. Evolutionary-based methods, in particular, have received significant attention due to their relative success at balancing between minimal initial knowledge, design space freedom, and computational expense [211].

In evolutionary computation, an initial population (set of candidate solutions) is generated and iteratively updated through metaheuristics and stochastic operators [220]. While these approaches can produce ‘semi-optimal’ solutions [211], they still have some drawbacks. First, the underlying algorithms have a number of parameters that must be selected (e.g., crossover probability and population size) [37, 210, 217, 218, 220]. Often values are given, but the sensitivity of these parameters is not thoroughly investigated, or better yet, optimal recommendations. Secondly, a poor initial population can result in delayed convergence [211]. Since these approaches are inherently stochastic, robustness is an important issue. Even with a favorable initial population, using standard metaheuristics can be inefficient (although some recent work has started to address this issue with custom genetic algorithms [211]). It has also been observed that overly complex solutions (i.e., too many components) are often selected [210, 213, 217]. In general, a target response is more easily satisfied with a complex circuit than a simple one [210]. Finally, the global solution is not guaranteed, and it is hard to determine how far we are from the global optimum. Addressing these shortcomings could lead to a truly efficient and effective synthesis tool.

All of the previously mentioned synthesis methods have aimed to navigate the vast design space by selectively sampling from the complete design space. An infrequently utilized approach is to actually test all topologies. In an enumerative synthesis approach, we generate and evaluate a complete listing of all possible circuit topologies under certain specifications. Then we can simply select the best one with confidence in its optimality. The obvious reason to not test all topologies is the rate at which such a complete listing grows with the number of candidate system components. Here we make a case for the proposed enumeration-based synthesis methodology to solve certain types of synthesis problems, as well as to generate knowledge that could aid more general and scalable solution of synthesis problems.

Enumeration has been utilized previously for electrical circuit design. Macmahon, among others, have considered the enumeration of series-parallel networks [9–11]. Foster provided the enumeration of geometrical circuits classified according to their nullity and rank [78].

Bruccoleri et al. systematically generated all the wide-band amplifier circuits with two MOS transistors [221, 222]. Enumeration has also been useful in other synthesis problems such hybrid powertrains [13], gear trains [15], and biological networks [16].

Before the enumeration-based methodology is described, it is important to visit the main motivations for this work, including the questions:

- Under what specifications/complexity is enumeration still feasible?
- Have previous methods found the “best” circuit? How close was the found circuit to the “best” circuit?
- What are the properties of the complete circuit structure space (e.g., how many unique circuits are there with up to a certain number of components)?
- Are stochastically sampled (rather than enumerative) topologies generated with the proposed approach more useful for initial populations due to their structural properties?
- Can a complete listing of circuits (both good and bad) be used to adapt an evolutionary approach to better address the synthesis task, perhaps through online learning or new metaheuristics?

These points will be addressed in Sec. 6.4.

The remainder of the chapter is organized as follows. Section 6.2 describes the proposed enumeration-based synthesis methodology. Section 6.3 provides some examples for both frequency response and low-pass filter synthesis problems. Section 6.4 is a discussion of the results and methodology. Section 6.5 provides the conclusions.

6.2 Enumeration-Based Synthesis Methodology

In this section, the procedure used to synthesize passive analog circuits is described. At the core of this approach is evaluating a set of circuits generated through an enumerative procedure. Figure 6.1 illustrates the overall flow of the enumeration-based synthesis framework.

6.2.1 Representing Circuits as Colored Graphs

Fundamental to this approach is the representation of circuits as colored graphs. A colored (or labeled) graph is an extension of an undirected graph with an additional set defining

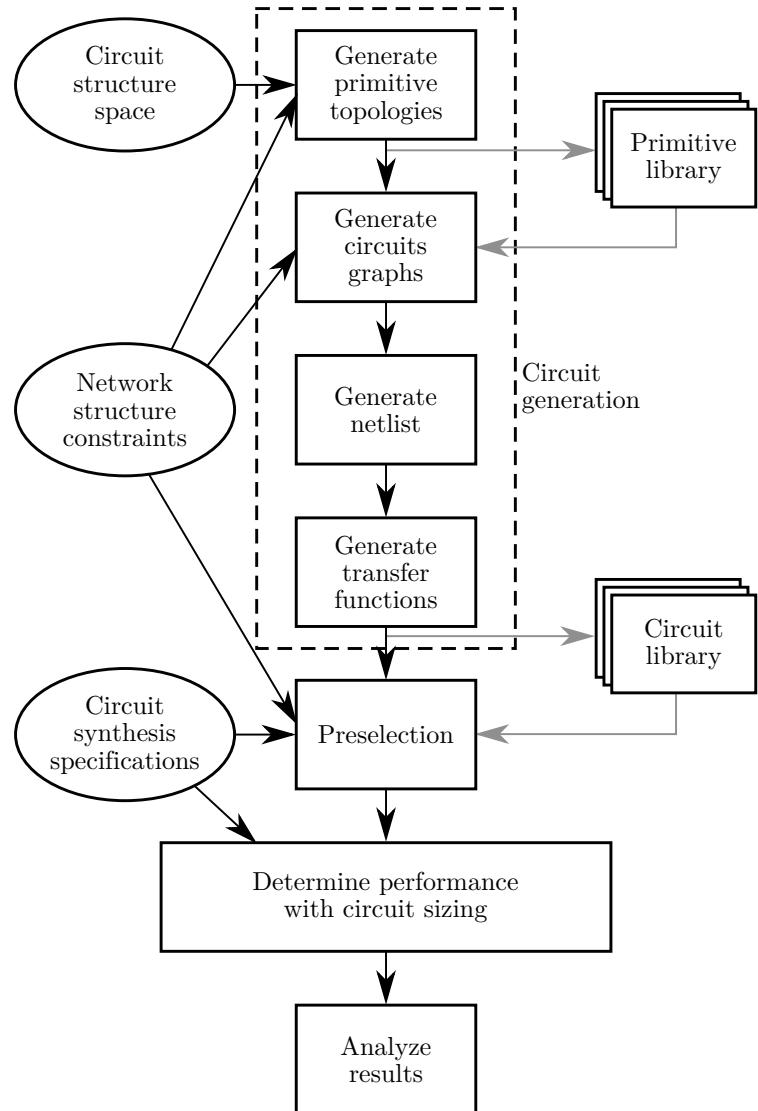
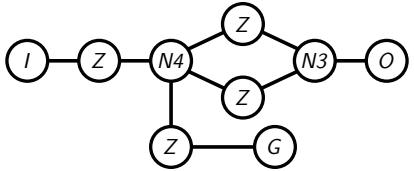
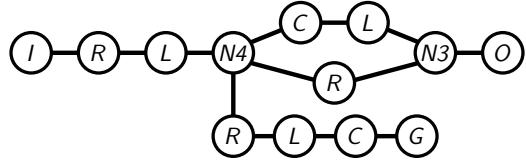


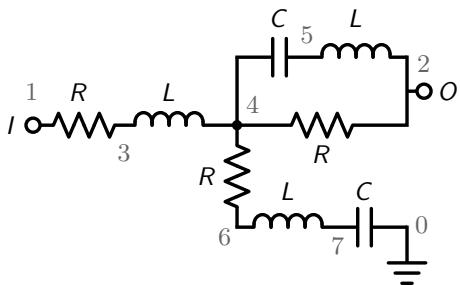
Figure 6.1: Enumeration-based synthesis methodology.



(a) Primitive circuit.



(b) Practical circuit.



(c) Circuit schematic.

7	0	<i>C</i>
7	6	<i>L</i>
6	4	<i>R</i>
5	4	<i>C</i>
5	2	<i>L</i>
4	3	<i>L</i>
4	2	<i>R</i>
3	1	<i>R</i>

(d) Netlist.

Figure 6.2: Different representations used for the same circuit.

the vertex or edge colors [223]. Colored graphs in this chapter will always be vertex-colored graphs. A properly defined colored graph will capture the topology of an electrical circuit.

In a vertex-colored graph, circuit components such as resistors, capacitors, inductors, etc. can be represented as different vertex colors as is shown in Fig. 6.2b. For example, every resistor in a circuit is labeled with *R* and is indistinguishable from one another. If we are only concerned with the topology of the circuit network, not the specific components used, then we can use general impedance colors *Z* (see Fig. 6.2a for an example). We will use the terms *primitive circuit* if only general impedance elements are used and *practical circuit* if specific components are present (such as the standard two-ports passive elements). In this chapter, we will only consider two-port impedance elements, although the extension to multi-port impedance elements is possible.

In addition to the impedance elements, additional voltage nodes in the circuit [40] will be labeled with *N*. These are 0-junctions in bond graph modeling where the voltage is constant [38]. Voltage nodes are further categorized by their number of connections (e.g., 3-port voltage node vs. 4-port voltage node). Two *n*-port voltage nodes are then considered indistinguishable. The distribution of voltage nodes in the circuit will vary (in Fig. 6.2a, there is one 3-port and one 4-port *N*). Similarly, the ground voltage node is labeled with *G*.

One advantage of a colored graph representation is colors can be used to represent a variety of concepts. Many synthesis problems directly involve the input/output behavior of the synthesized circuit (e.g., single-input single-output (SISO) transfer function) [211]. We

can use different colors to represent to location of the input and output nodes in the circuit (see Fig. 6.2a with colors I and O).

To summarize, circuits defined by colored graphs will contain colors representing two-port impedance elements, a variety of n -port voltage nodes, the ground node, the input node, and the output node. With all relevant information included in the colored graphs, we can determine if two colored graphs are unique with respect to permutations of the vertices of the graphs [6, 224]. This is known as the colored graph isomorphism problem and is important to handle so that redundant circuits are not reused.

There are other graph representations of circuits. A more common representation is an edge-colored graph with all vertices representing voltage nodes and edge colors representing different impedance types [23]. Vertex coloring will be needed if the input and output nodes need to be identified. The motivations behind the chosen representation is the ability to define intuitively the set of circuits that will be enumerated and leverage existing work in enumerating colored graphs [6]. These properties will become apparent in the following sections.

6.2.2 Generating Primitive Circuits

Enumeration of the primitive circuits is performed using a perfect matching-inspired algorithm where all ports of every component are connected to exactly one other port [6]. In the worst case scenario, the growth of the number of graphs is $(N - 1)!!$ where N is the total number of ports and $!!$ represents the double factorial function. However, this bound is extremely conservative as many of the generated graphs are isomorphic (not unique) or do not satisfy network structure constraints (NSCs). Satisfaction of all NSCs defines the feasibility for a particular graph. Many enhancements to the original algorithm have been made in Ref. [41], further leveraging the structure of the enumeration task and allowing reasonably large graph structure spaces to be enumerated. Here we will refer to the graph structure space covered by this approach as the circuit structure space²².

The required information for the enumeration approach is the following designer-defined elements:

- C is the colored label sequence representing distinct component types
- P is a vector indicating the number of ports for each component type

²²For more details on the enumeration approach used see Chapter 2 and Appendix A.

- R_{\min} is a vector indicating the minimum number of replicates for each component type
- $R_{glsfoo[noindex]max}$ is a vector indicating the maximum number of replicates for each component type

where we define R as a matrix containing both R_{\min} and R_{\max} . For a circuit synthesis task, we could choose these elements as:

$$C = \{I, O, G, Z, N3, N4\}, \quad P = [1 1 1 2 3 4] \quad (6.1a)$$

$$R_{\min} = [1 1 0 1 0 0], \quad R_{\max} = [1 1 1 3 2 1] \quad (6.1b)$$

where we have included the input node, output node, ground node, general two-port impedance elements, 3-port voltage nodes, and 4-port voltage nodes. The input/output nodes and at least one impedance element are mandatory in any feasible graph. In a feasible circuit, there can be up to three impedance elements, two 3-port voltage nodes, and one 4-port voltage node. The choice of these elements is a major design decision and will be discussed further along with the examples presented in Sec. 6.3.

The addition of network structure constraints not only improves the quality of the generated graphs, but also decreases the computation time required for enumeration [6, 41]. The first NSC is the requirement that every feasible graph has no loops or multi-edges. Next, we require that each graph is a connected graph so the input/output are guaranteed to be connected and there are no isolated components.

The next set of NSCs limit the structure of the graph's adjacency matrix by ensuring zeros in certain locations:

- No ports from $\{I, O, G\}$ can be connected. If any of these are connected directly, then the transfer function for the circuit will be trivial and useless.
- No ports from $\{Z\}$ can be connected. This would create series general impedance which is undesired during the generation of primitive circuits.
- No ports from $\{N3, N4, \dots, Nx\}$ can be connected. This simply creates larger voltage nodes and we already restricted the maximum port size.

A final set of NSCs are some line-connectivity constraints [41]. Each constraint is specified as a triple of integers: $[\#1, \#2, \#3]$ where each triple is interpreted as: if $\#1$ and $\#2$ are connected, don't ever connect $\#2$ to $\#3$. For example, if we have $[1, 5, 2]$ and use Eqn. (6.1), then we are enforcing if I is connected to $N3$, then $N3$ should not be connected to O . If we had $I - N3 - O$ present in a graph, the transfer function between the input and output

would be unity. To prevent such situations, we have that no voltage node (Nx) should be connected directly between any of the 1-port components $\{I, O, G\}$.

Based on these NSCs, we can filter out subcatalogs of (C, P, R) that we know will not contain any feasible graphs; thus, reducing the computational expense when generating primitive circuits. A subcatalog is a set of component replicates bounded by (C, P, R) [41]. Since no voltage node can be connected to another, their ports must be connected to the other component types. The same is true for $\{I, O, G\}$ and $\{Z\}$. The 1-port components may be connected to either Z or Nx , resulting in two possible cases: one where all 1-port components are connected to Z , and another only to Nx . Then a necessary condition for a feasible graph to exist in a certain subcatalog under the NSCs is:

$$p_N < p_Z + (p_I + p_O + p_G) \wedge p_N > p_Z - (p_I + p_O + p_G) \quad (6.2)$$

where p_x indicates the total number of ports for component-type x in the subcatalog. For example for Eqn. (6.1), the subcatalog $R_1 = R_{\max}$ is infeasible due to the left condition ($10 \not\prec 9$), and the subcatalog $R_2 = [1 \ 1 \ 0 \ 3 \ 1 \ 0]$ is infeasible due to the right condition ($3 \not\succ 4$).

With the desired circuit structure space fully defined, we use enumeration to generate all possible unique graphs that satisfy the requirements above (see Ref. [65] for the MATLAB code for enumeration). This set of graphs is termed the *primitive library* and can be saved for future reuse. For Eqn. (6.1) with all the previously mentioned NSCs, there are 14 unique, feasible primitive graphs. Each one of these primitive circuits is shown in Fig. 6.3. Here we see many common topologies including parallel, series-parallel, L-section, and T-section. *All* topologies that can be constructed with respect to (C, P, R) and the NSCs will be present.

6.2.3 Generating Practical Circuits

For every primitive circuit in the primitive library, we want to transform the general impedance elements into practical components. The designer must specify the potential subcircuits for these general impedance elements. Here we will consider subcircuits defined by series connections between $\{R, L, C\}$ components, visualized in Fig. 6.4. This set of subcircuits was chosen since the primitive graphs capture different “parallel” topologies and these subcircuits contain different “series” variations.

It is important to understand the growth during this step so if there are N potential subcircuits and n general impedance components in the primitive circuit, then there are $(2^N - 1)^n$ practical circuits. If $N = 7$ and $n = 4$, then there are 2,401 practical circuits for

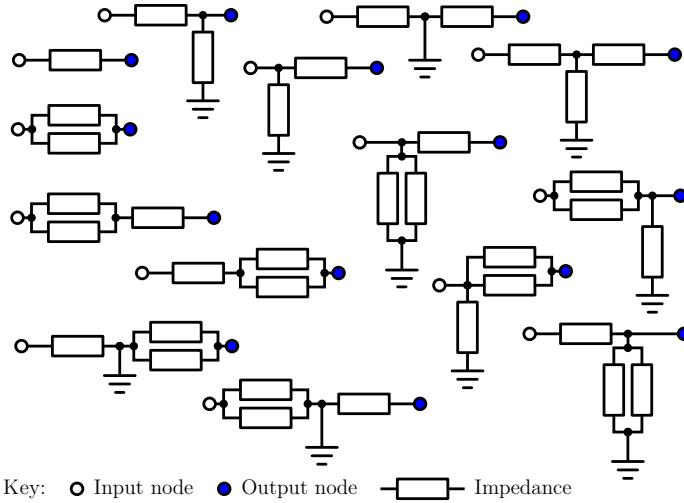


Figure 6.3: 14 primitive circuits for the circuit structure space defined by Eqn. (6.1).

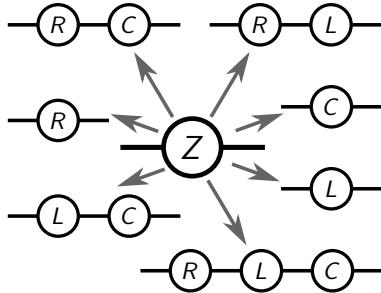


Figure 6.4: Subcircuits considered for generating practical circuits (series RLC subcircuits).

a single primitive circuit. These permutations do not guarantee that each practical circuit is unique, so isomorphic checks must be performed. Note that a practical circuit generated from primitive circuit A will not be isomorphic to a practical circuit generated from primitive circuit B , so we only need to compare practical graphs from the same primitive circuit.

6.2.4 Model Construction

In many synthesis problems, there is a predefined *template circuit* (or embryonic circuit) that contains some fixed circuit elements and their relation to the input node, output node, and ground [23, 37, 211, 217, 218]. Two common template circuits are shown in Fig. 6.5. A *complete circuit* combines a practical circuit and the template circuit.

For a complete circuit, a model is constructed by obtaining the transfer function between

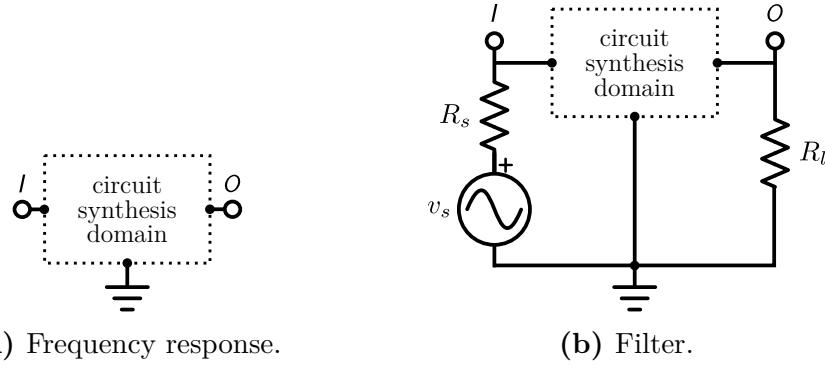


Figure 6.5: Two common template circuits.

the input and output nodes, denoted G . This transfer function will depend on the frequency s and parameters from the template circuit ρ such as R_l in Fig. 6.5b. Additionally, optimization variables \mathbf{x} for the circuit are the coefficients for the two-port elements. The distribution of the optimization variables will vary depending on the practical circuit.

To generate $G(s, \rho, \mathbf{x})$, we use SCAM [225], a MATLAB tool that derives and solves circuit equations symbolically using the modified nodal analysis method [40]. This tool requires a netlist representation of the circuit, which is a set of triples that define the interconnections of circuit elements relative to numbered voltage nodes (see Fig. 6.2d for an example). It is fairly straightforward to generate the netlist from the colored graph representation. The only exception is when voltage nodes are connected to any of the 1-port elements since these components are conceptually voltage nodes (and therefore are collapsed to a single voltage node).

The collection of all complete circuits and their models is termed the *practical circuit library*, and can be saved for future reuse. Using the primitive library in the previous section and the subcircuits from Fig. 6.4, there are 207 unique complete circuits.

6.2.5 Preselection

Preselection is an optional step that can be performed before the complete circuits are optimized for the particular synthesis task. The designer can limit the circuits that are evaluated based on their preferences/constraints. Some examples include:

- Bounds on the degrees of the polynomials $N(s)$ and $D(s)$, where $G(s) = N(s)/D(s)$
- Bounds on the total number of components, n_c , sometimes considered a complexity measure [217]

- Bounds on the distribution of the components (e.g., limit to a maximum of three resistors and two capacitors)
- Bounds on the total number of circuits to evaluate, potentially selecting a fixed number of random circuits (although the coverage properties would no longer be present)

These preferences could be utilized to explore lower-complexity circuits first, checking whether higher complexity circuits are needed to satisfy the requirements of the synthesis task. Since this approach uses a predefined library of circuits, these preferences/constraints can be handled readily.

6.2.6 Evaluation

With the desired set of complete circuits, we now need to determine how well the circuits satisfy a given synthesis problem. Here we consider synthesis tasks that seek to minimize the error between desired transfer function properties and the circuit's physical response. This involves sizing the circuit optimization variables \mathbf{x} . Consider a set of n_s frequency points, denoted Ω , with desired values defined by $f(\omega_k)$. Using a model function $g(\omega_k, \mathbf{x})$, the synthesis task is posed as the following curving fitting problem:

$$\min_{\mathbf{x}} E = \sum |r_k(\mathbf{x})|^2 \quad (6.3a)$$

$$\text{subject to: } \mathbf{a}(\mathbf{x}) \leq \mathbf{0} \quad (6.3b)$$

$$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \quad (6.3c)$$

$$\text{where: } r_k = g(\omega_k, \mathbf{x}) - f(\omega_k) \quad (6.3d)$$

where E is the error, r_k are the individual residuals, \mathbf{a} is the set of additional general constraints, and $\{\mathbf{l}, \mathbf{u}\}$ are the lower and upper bounds on the optimization variables, respectively. If there are no additional general constraints, then this is a standard nonlinear least-squares (NLS) problem and suitable solvers are utilized [226]. Otherwise, general nonlinear program (NLP) solvers will be used [130].

One suitable residual function for matching a circuit transfer function to a desired magnitude response is:

$$r_k = \log|G(\omega_k, \mathbf{x})| - \log|F(\omega_k)| = \log \left| \frac{G(\omega_k, \mathbf{x})}{F(\omega_k)} \right|, \quad k = 1, \dots, n_s \quad (6.4)$$

i.e., we want to minimize the pointwise decibel error (ignoring the constant for simplicity since it does not effect the optimal curve fit). Alternative residual functions have been suggested

in the literature. Staying within the NLS framework, $r_k = G - F$ or $r_k = N - FD$ [227]. Others utilized normalized versions [211, 216] or the sum of the absolute errors [217]. The chosen function has been shown to produce solutions similar to the other residual functions but frequently is much smoother, improving convergence [228].

Another useful residual function is when only feasibility is sought, i.e., find a suitable \mathbf{x} such that Eqns. (6.3b)–(6.3c) are satisfied. This feasibility problem can be posed as a NLS using the following residual function:

$$r_k = \max(0, a_k(\mathbf{x})), \quad k = 1, \dots, n_a \quad (6.5)$$

where n_a is the number of constraints. This results in a common penalty method used with NLP [229].

Some studies also introduce heuristic penalty functions to help penalize higher-complexity circuits [210, 213]. Complexity penalization is not needed in the optimization formulation as it can be readily computed offline because the distribution of circuit elements is not changing, as is the case in an evolutionary approach.

Two other aspects of the evaluation procedure are a possible weight function and linear frequency scaling. A weight function may be defined as $E = \sum w(\omega_k) |r_k(\mathbf{x})|^2$ where w is the weight function that may be used to give less attention to high frequencies, provide greater weight to specific regions of interest, or combine multiobjectives [211, 216]. We will also use simple linear scaling of the frequency, i.e., $s = \alpha \bar{s}$ where $\alpha > 0$. This is done for numerical stability reasons and we will use the following recommendation from Ref. [230]: $\alpha = (\min(\Omega) + \max(\Omega))/2$.

Due to the complexity of the proposed nonlinear program, it may be challenging to find a feasible solution or solutions near the global optimum. To help remedy this issue, a multi-start approach is utilized [231]. Here a large number of stratified random samples (100,000) were initially tested and the best five were used as initial points in independent optimization runs.

When the performance of all the circuits has been evaluated, the results can be analyzed. Unlike other synthesis approaches, there are potentially a large number of feasible circuits to choose from. Having many alternatives is a significant advantage. Tradeoffs in complexity vs. performance can readily be observed. For example, we can find the lowest complexity feasible circuit. Alternatively, if no feasible circuits are found, we have gained some insights into the types of circuits that will be required to satisfy the synthesis task. These post-processing analysis tasks will be discussed more in the [Discussion](#) section.

6.3 Examples

Two different kinds of passive analog synthesis problems (frequency response matching and low-pass filter realizability) have been chosen as the design examples. Both of these problem types have been studied previously, so we can readily make comparisons to previous results. All runs were performed on a single computer with an i7-6800K at 3.8 GHz (up to 12 threads available), 32 GB DDR4 3200 MHz RAM, WINDOWS 10 64-bit, and MATLAB 2017a. All aspects of the synthesis tool are coded in the MATLAB language²³.

6.3.1 Frequency Response Matching

6.3.1.1 Synthesis Task

The first example is based on Ref. [210]. Here we wish to synthesize circuits that match the following frequency response:

$$|F(j\omega)| = \sqrt{\frac{2\pi}{10\omega}} \quad (6.6)$$

over the frequency range:

$$0.2 \leq \frac{\omega}{2\pi} \leq 5$$

with 500 logarithmically spaced evaluation points. Since we have a desired magnitude response, not an explicit transfer function, traditional design methods cannot accommodate this synthesis task [210].

We will consider two sets of simple bounds on the coefficients of the resistors and capacitors:

$$\begin{aligned} \text{(set 1)} \quad R &\in [10^{-2}, 10^0] \Omega, & C &\in [10^{-2}, 10^0] F \\ \text{(set 2)} \quad R &\in [10^{-2}, 10^5] \Omega, & C &\in [10^{-10}, 10^0] F \end{aligned}$$

and there are no additional general constraints. The residual function is the pointwise decibel error in Eqn. (6.4).

²³The synthesis codes are available at Ref. [232].

6.3.1.2 Circuit Structure Space

The desired circuit structure space is “all topologies that have up to 6 impedance subcircuits and a required connection to the ground.” Such a space is captured by:

$$C = \{I, O, G, Z, N3, N4, N5, N6, N7, N8, N9\} \quad (6.7a)$$

$$P = [1 \ 1 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9] \quad (6.7b)$$

$$R_{\min} = [1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \quad (6.7c)$$

$$R_{\max} = [1 \ 1 \ 1 \ 6 \ 5 \ 3 \ 3 \ 2 \ 1 \ 1 \ 1] \quad (6.7d)$$

and includes all the NSCs from Sec. 6.2.2. Only nine different orders of voltage nodes are used, as any higher-order nodes would not produce feasible topologies. The specific values of R_{\max} are the maximum number of replicates needed to still capture all possible feasible subcatalogs with respect to Eqn. (6.2). The subcircuits for generating practical circuits will be series connections between $\{R, C\}$ components to replicate the elements available in Ref. [210]. The primitive circuit library then has 393 unique graphs, and these graphs are expanded to 104,235 unique practical circuit graphs (see Table 6.1).

	Step	t (s)	N
Circuit generation	Primitive circuits	78	393
	Practical circuits	131	104,235
	Transfer functions	14,379	43,249
Evaluation	Ref. [210], set 1	29,303	43,249
	Ref. [210], set 2	28,739	43,249

Table 6.1: Computational cost of Frequency Response Matching example.

This synthesis task will utilize the template circuit in Fig. 6.5a. Of the 104,235 practical circuit graphs, 43,249 were found to have unique transfer functions. This difference is attributed to cases where two circuits that have the same transfer function but different colored graph representations. For example, in some practical circuits, components do not show up in the transfer function (due to a functional short between the input and output nodes). In these cases, the lower complexity circuit is always kept. With the desired circuit library, we can now perform the sizing task to determine which circuits better satisfy the requirements of the synthesis task.

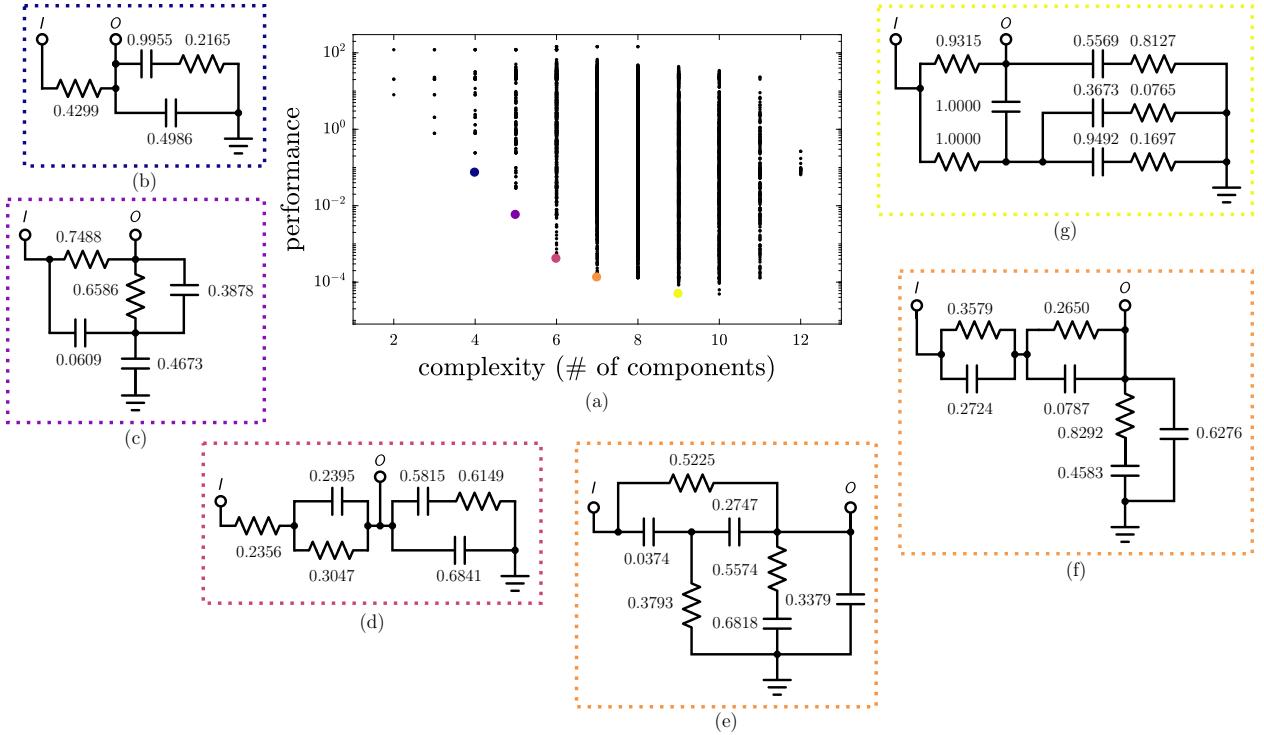
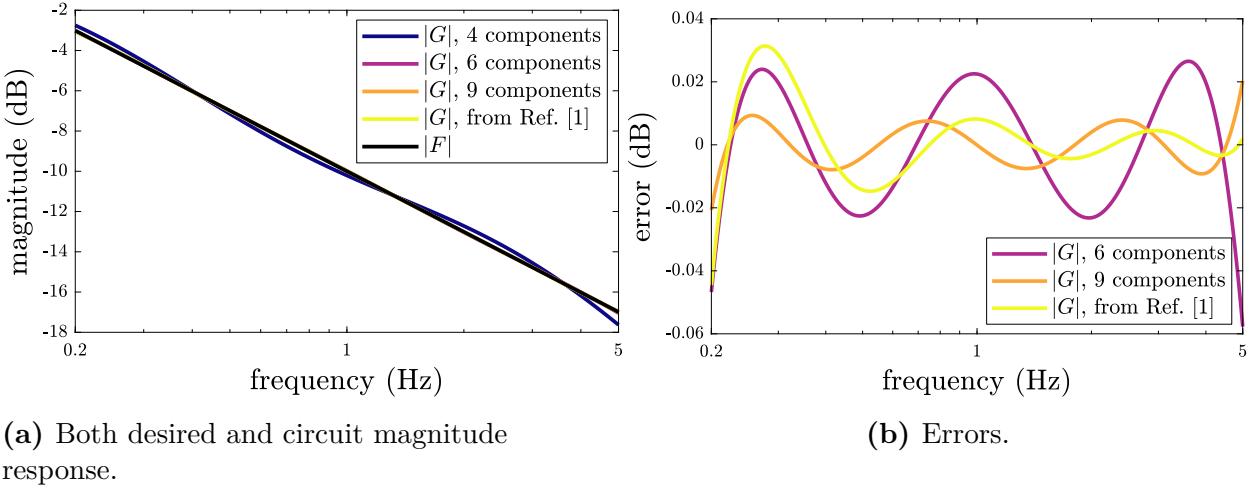


Figure 6.6: Performance vs. complexity (# of components) for Frequency Response Matching example using set 1 along with select Pareto optimal circuits (units are Ω and F).

6.3.1.3 Results

The results for set 1 are summarized in Fig. 6.6a where the performance for every circuit is shown, stratified by the complexity (or number of components). Select Pareto-optimal (performance cannot be improved without an increase in complexity) circuits are also shown in Figs. 6.6a–g. The desired magnitude response can be seen in Fig. 6.7a along with $|G|$ for select circuits in Fig. 6.6.

In fact, the included circuits are not unique Pareto-optimal circuits. For complexity levels of $\{4, 5, 6, 7, 9\}$, there are $\{5, 22, 57, 200, 2\}$ different circuits that produce the same level of performance. This is due to each having nearly identical transfer functions using different circuit topologies. For a specified number of poles and zeros, there is a lower bound on the performance (nonzero in this task) defined by a transfer function where the coefficients are tuned directly (sometimes called complex-curve fitting [227]). Therefore, a set of circuits perform as well as possible under the pole/zero limitations defined by the circuit structure space and complexity level. For example, consider the circuit in Fig. 6.6b where G contains 2 zeros and 3 poles, and produces a performance level of 7.3×10^{-2} . If we instead designed the polynomial coefficients directly, the lower bound on the performance is 2.9×10^{-2} ; which is



(a) Both desired and circuit magnitude response.

(b) Errors.

Figure 6.7: Magnitude and errors over the desired frequency range using select circuits from Fig. 6.6.

similar to the synthesized circuit but smaller. See Table 6.2 for this comparison at additional complexity levels and note the trend in the optimal orders of n_z and n_p .

Performance					
n_c	n_z^*	n_p^*	Best circuit G	Best G	
4	2	3	7.3×10^{-2}	2.9×10^{-2}	
5	3	3	5.7×10^{-3}	3.4×10^{-3}	
6	3	4	4.1×10^{-4}	4.0×10^{-4}	
7	4	4	1.3×10^{-4}	4.8×10^{-5}	
8	4	5	1.3×10^{-4}	5.6×10^{-6}	
9	5	5	4.8×10^{-5}	6.6×10^{-7}	

Table 6.2: Performance level comparison between best circuit transfer function and best arbitrary transfer function for various complexity levels.

In a similar manner as set 1, the results for set 2 are summarized in Fig. 6.8. In this set of results, an interesting pattern emerges, namely more discrete groupings of the performance levels and similarly valued groups are shared between different complexity levels. With increased flexibility in the values of the passive component's coefficients, transfer functions with similar properties (e.g., the same order for n_z and n_p) are attracted to similar performance levels during sizing. This is further illustrated in Fig. 6.9 where the performance is plotted by cumulative percentage of circuits that have at least the given performance level. For set 1 we see little pattern in the performance curve but with set 2, we see more discrete performance levels. In this figure we also see more circuits achieving a specified level of

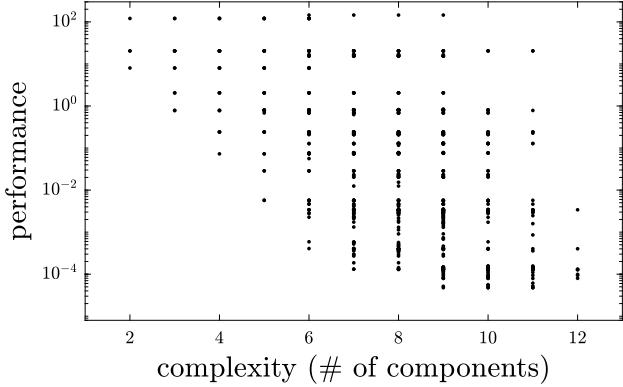


Figure 6.8: Performance vs. complexity (# of components) for Frequency Response Matching example using set 2.

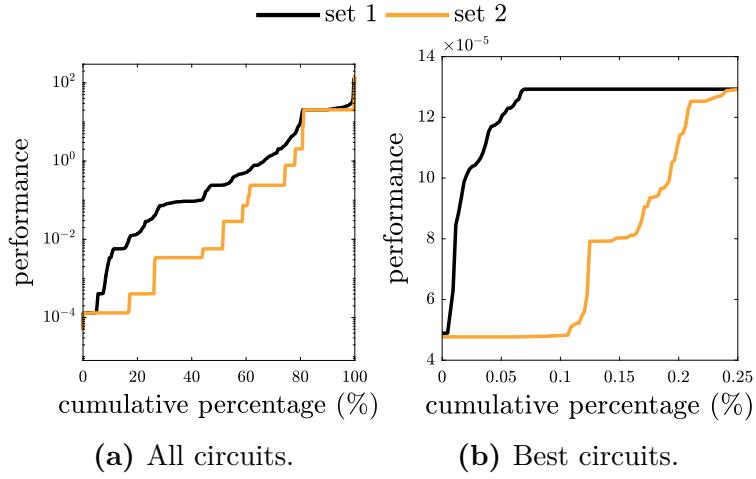


Figure 6.9: Performance vs. cumulative percentage for Frequency Response Matching example.

performance with the increased variable ranges in set 2 (i.e., the curve for set 2 is always below set 1). In addition, many more circuits achieve the best performance level, but there is only a minor reduction of the objective compared to circuits sized using set 1.

A circuit of particular interest is the one found in Fig. 6.6e, as it has the same topology as the circuit reported in Ref. [210]. Since the topology and sizing tasks were performed simultaneously using an evolutionary approach in Ref. [210], it could have been challenging to converge to the local optimum (something gradient-based methods do well). However, it is extremely impressive that the synthesized circuit in Ref. [210] was close to the Pareto frontier (same topology, slightly worse performance level at 3.0×10^{-4} with the error visualized in Fig. 6.7b).

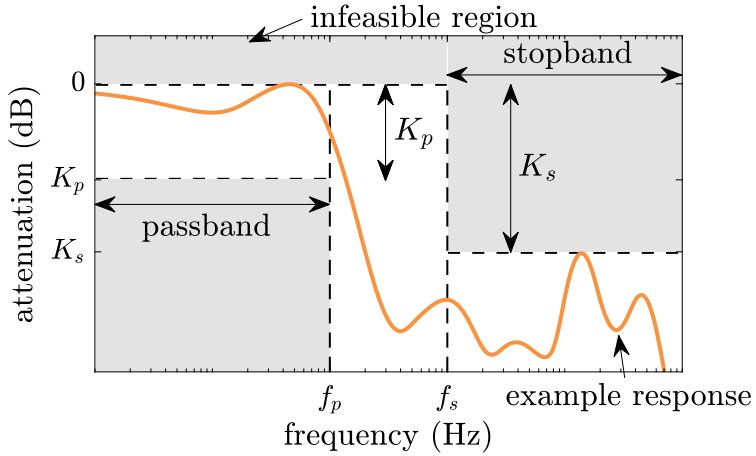


Figure 6.10: Low-pass filter specifications.

While the circuit in Fig. 6.6g has the best performance level (under the enumerated circuit structure space), it would be up to the designer to decide if the increase in complexity is worth the performance improvement. Having a large number of options provides the designer with additional flexibility when selecting a final circuit.

6.3.2 Low-Pass Filter Realizability

6.3.2.1 Synthesis Task

The second example is the synthesis of a low-pass filter (LPF). A LPF attenuates signals above a certain frequency and passes all other signals. The design specification of a LPF is shown in Fig. 6.10. There are many classical synthesis methods for LPFs such as Butterworth, Chebyshev I, Chebyshev II, Elliptic, Legendre-Papoulis, etc. [233]. LPFs are also frequently used as examples for evolutionary-based synthesis methods [23, 37, 211, 213, 217, 218, 234]. Here we will try to synthesize four LPFs with their specifications and variable bounds shown in Table 6.4.

6.3.2.2 Circuit Structure Space

The desired circuit structure space is “all topologies that have up to 7 passive components with an optional connection to the ground”. Such a space is captured by:

$$C = \{I, O, G, Z, N3, N4, N5, N6, N7, N8, N9\} \quad (6.8a)$$

$$P = [1 \ 1 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9] \quad (6.8b)$$

$$R_{\min} = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \quad (6.8c)$$

$$R_{\max} = [1 \ 1 \ 1 \ 7 \ 5 \ 4 \ 3 \ 2 \ 2 \ 2 \ 1] \quad (6.8d)$$

and includes all the NSCs from Sec. 6.2.2. This is very similar to Eqn. (6.7), except Z now has up to seven replicates, G is optional, and some of the Nx maximum replicate numbers have changed based on what is needed for Eqn. (6.2). The subcircuits for generating practical circuits will be series connections between $\{L, C\}$ components to replicate the elements typically used in LPFs. The primitive circuit library then has 5,300 unique graphs and these graphs are expanded to 1,804,496 unique practical circuit graphs (see Table 6.3).

	Step	t (s)	N
Circuit generation	Primitive circuits	2,694	5,300
	Practical circuits	6,330	1,804,496
	Transfer functions	20,460	123,156
Evaluation	Task #1	235,810	123,156
	Task #2	80,788	38,172
	Task #3	83,231	38,172
	Task #4	606	281

Table 6.3: Computational cost of Low-Pass Filter Realizability example.

This synthesis task will utilize the template circuit in Fig. 6.5b with parameters $\{v_s, R_s, R_l\}$ valued at $\{2V, 1k\Omega, 1k\Omega\}$. Of the 1,804,496 practical circuit graphs, 123,156 were found to have less than 7 components and remain unique transfer functions. The practical circuit graph representation can be used directly to count the number of components before the transfer function is constructed.

6.3.2.3 Results

The results for this example are summarized in Table 6.4 with the number of synthesized feasible circuits, the percentage of circuit topologies that were feasible after sizing, and the

#	c.f.	f_p (Hz)	f_s (Hz)	K_p (dB)	K_s (dB)	L bounds (H)	C bounds (F)
1	[37, 217]	925	3200	3.01	22.0	[0.1m, 1.5]	[0.1p, 200 μ]
2	[23]	1000	2000	1.00	60.0	[0.01m, 10]	[0.1p, 100 μ]
3	[211]	800	2000	0.60	68.0	[0.1m, 1]	[100p, 1 μ]
4	[37, 217]	1000	2000	0.01	63.5	[0.1m, 1.5]	[0.1p, 200 μ]

(a) Specifications.

#	# feasible	% feasible	min n_c
1	38172	30.99	3
2	280	0.23	6
3	197	0.16	6
4	0	0	> 7

(b) Results.

Table 6.4: Summary of Low-Pass Filter Realizability specifications and results.

minimum number of components needed to satisfy the synthesis specifications. The tasks had increasingly more difficult to satisfy specifications.

The first study is based on examples in Refs. [37, 217], and the specifications were chosen such that they could be satisfied using a third-order Butterworth filter [37]. This implies that there should be at least one three-component topology that satisfies the specifications (ignoring the variable bounds). In fact, there are six different three-component topologies that satisfy the requirements, all shown in Figs. 6.11a–6.11f. The circuits in Fig. 6.11a and Fig. 6.11b are topologically similar to one another, as well as Fig. 6.11c and Fig. 6.11d. The other two are well known topologies; Fig. 6.11e is a π -section, and Fig. 6.11f is a T-section. Their attenuation responses are shown in Fig. 6.11g; all responses are within the specifications.

The best circuit for task #1 found in Ref. [217] is Fig. 6.11b; thus, their evolutionary approach did find one of the minimum-complexity circuits. The best circuit from Ref. [37] had seven components; it is not reported as it is not a minimum-complexity topology. Since enumeration was used, we can look at the likelihood that certain topologies would have been feasible. For topologies with up to seven components, 30.99% of the topologies are feasible (see Table 6.5 for the percentage for each complexity level).

The specifications for task #2 are more stringent. Because of this, we can take the 38,204 feasible circuits from task #1 as the starting set of circuits since any circuit that is not feasible in task #1 will not be feasible in task #2. From this, only 280 circuits are found to be feasible with the minimum n_c being six components. Two of the eleven minimum-

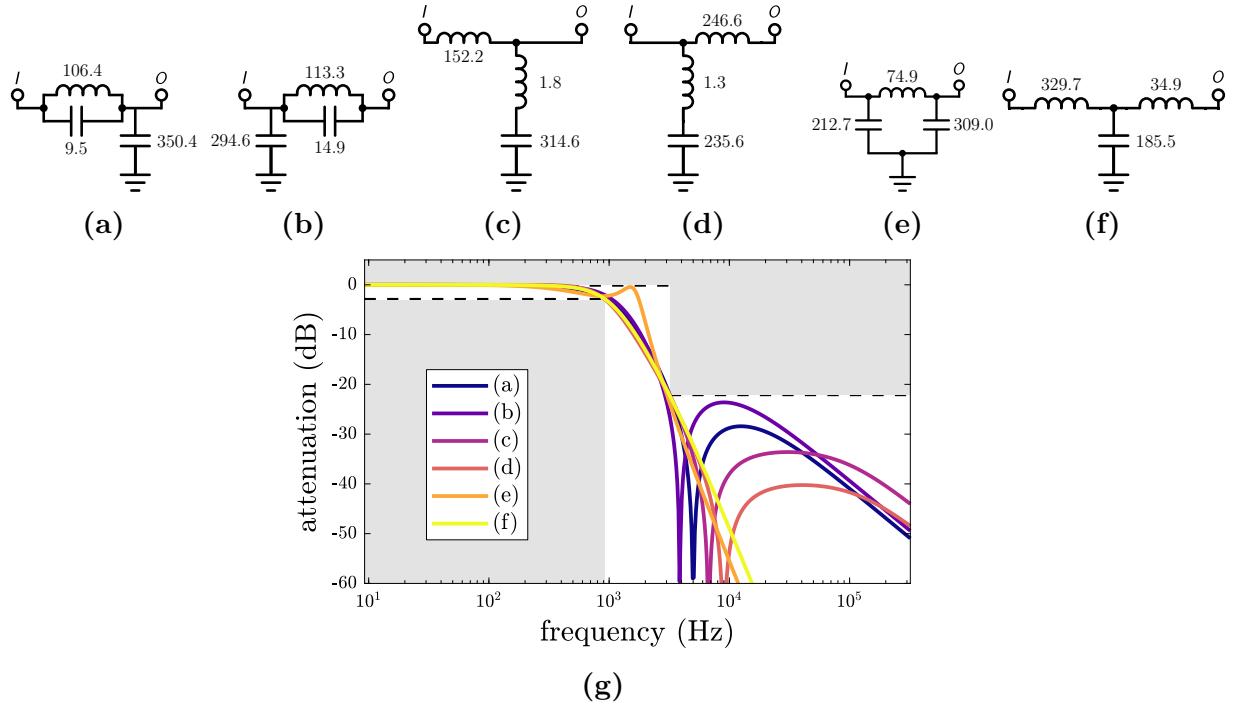


Figure 6.11: All feasible, minimum complexity circuits and attenuation responses for [Low-Pass Filter Realizability](#) task #1 (units are mH and nF).

n_c	Circuits		
	Feasible	Total	%
1	0	2	0.0
2	0	12	0.0
3	6	60	10.0
4	62	338	18.3
5	534	2,192	24.4
6	4,240	14,685	28.9
7	33,330	105,867	31.5

Table 6.5: Task #1 feasible vs. total number of circuits for different complexity levels.

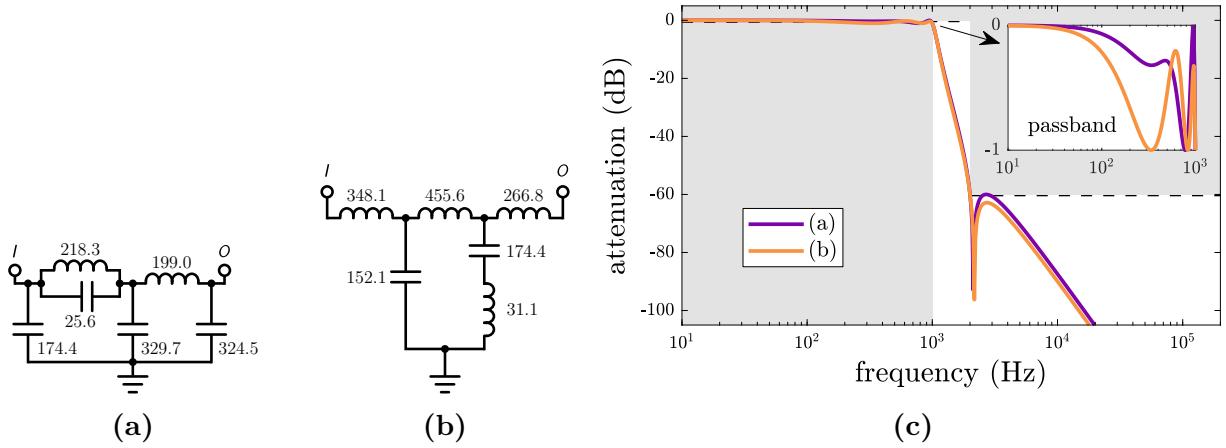


Figure 6.12: Select feasible, minimum complexity circuits and attenuation responses for Low-Pass Filter Realizability task #2 (units are mH and nF).

complexity circuits are shown in Figs. 6.12a–6.13b. Their attenuation responses are shown in Fig. 6.12c (note the magnified region to highlight constraint satisfaction in the passband).

In Ref. [23], the reported circuit had eight components. A direct comparison is not a fair assessment, as their study limited the preferred (discrete) component values (E12 series). However, we can see that Fig. 6.12a is the same as their reported topology when C4 and C5 are removed. An alternative complexity metric is the number of inductors as inductors can be bulky, heavy, and expensive compared to capacitors [233]. Under this metric, Fig. 6.12a is the minimum inductor solution (with two alternatives). A similar discussion can be had with the task #1 results, where only one inductor is needed in three of the circuits, such as in Fig. 6.11a.

The next task had slightly more stringent K_p and K_s limits, but the transition region was slightly larger. Furthermore, the variables bounds are the tightest of the four tasks. Only 197 topologies were found to be feasible, which is less than with task #2. In this task, six components was the minimum number required, and four of the ten minimum complexity topologies are shown in Figs. 6.13a–6.13d (all ten circuits are shown in Fig. B.1). The minimum inductor topology is illustrated in Fig. 6.13a. Their attenuation responses are shown in Fig. 6.13e, and all responses are within the specifications. The circuit in Fig. 6.13c is the same topology found in Ref. [211], which is another case of an evolutionary approach arriving at a minimum complexity circuit.

Task #4 had the toughest specifications, primarily due to $K_p = 0.01$. Since the requirements were equal to or more stringent than those in task #2, only the 280 feasible circuits from task #2 were tested, greatly reducing the computational cost. None of the topologies

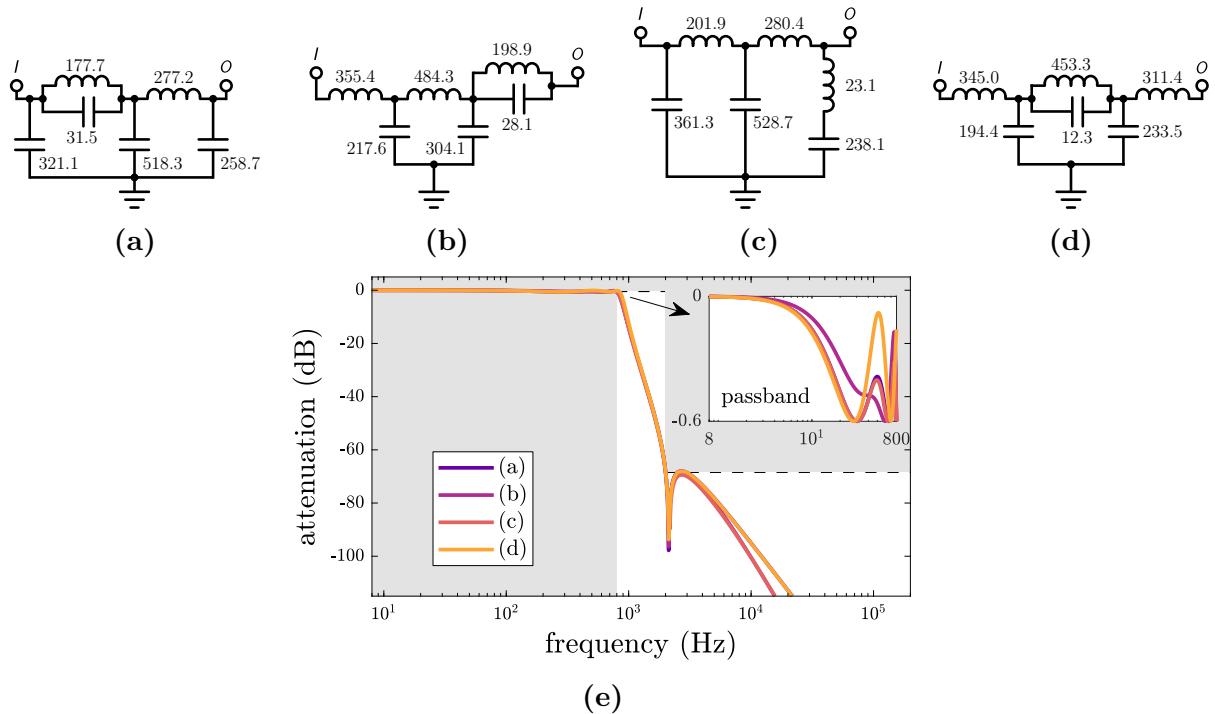


Figure 6.13: Select feasible, minimum complexity circuits and attenuation responses for Low-Pass Filter Realizability task #3 (units are mH and nF).

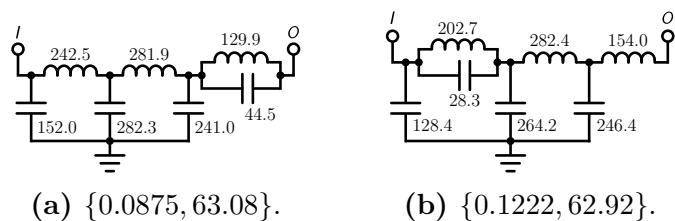


Figure 6.14: Top two closest to be feasible circuits for Low-Pass Filter Realizability task #4 and realized gains $\{K_p, K_s\}$ with respect to $f_p = 1000$ Hz and $f_s = 2000$ Hz (units are mH and nF).

produced a feasible circuit, so we come to the conclusion that at least eight components are needed. In Ref. [37], 21 components were needed, but in Ref. [217], only 12 components. Therefore, we now know that between 8 and 12 components are required to produce a feasible design.

The top two circuits (in terms of performance) are shown in Fig. 6.14. As expected, they both include seven components, which was the limit. The topologies are similar to minimum complexity topologies found in the previous tasks. Fig. 6.14a is similar to Fig. 6.13b, and Fig. 6.14b is similar to Fig. 6.13a. For the best circuit, the gains $\{K_p, K_s\}$ were found to be $\{0.0875, 63.08\}$, which are a substantial improvement compared to task #2, but still do not meet this task's stringent specifications.

6.4 Discussion

In this section we address the motivating questions introduced in Sec. 6.1, and summarize what we have learned about the enumeration-based synthesis methodology.

Both of examples in Sec. 6.3 show that enumeration is feasible for certain commonly-used synthesis problems. LPF task #4, on the other hand, demonstrates that sufficiently demanding synthesis problems can be a challenge to solve with enumeration, primarily due to the required computational cost. Even though no acceptable solution was found in task #4, valuable insights were gained and a number of good circuits were found that nearly satisfy the specifications. One way these nearly-feasible designs could be useful is as data for evolutionary computing methods that can leverage known good topologies to improve effectiveness [217].

The primary drawback of an enumeration-based synthesis methodology is simply the computational cost and its exponential growth with system size. Most of the resource-intensive tasks can be performed in parallel, such as generating the transfer functions and evaluating the circuits. All reported costs were for a single machine, but with increasing computational resources, larger and more complex synthesis problems can be solved within an acceptable duration using enumeration than in the past. We acknowledge that while enumeration strategies have clear limits, the methods presented here, based on perfect matching and efficient algorithms that eliminate isomorphisms and topologies that are infeasible with respect to NSCs, can enumerate all unique, feasible topologies for larger synthesis problems than those solved previously via enumeration. Furthermore, reusability is inherent to the proposed methodology, a property not typically associated with evolutionary approaches. Only

a single set of circuits and associated transfer functions was generated for all the tasks in the [Low-Pass Filter Realizability](#) example. In addition, we can use results from a previous synthesis task to evaluate a different synthesis study more efficiently, without comprising the enumerative properties of the approach. This was demonstrated in tasks $\#1 \rightarrow \#2 \rightarrow \#4$, where only the previous task's feasible circuits were used, greatly reducing the computational cost. So even if the initial investment is high, the data and knowledge gained can be used for similar design tasks.

Three circuit structure spaces were discussed in this chapter, providing insights into circuit properties that satisfy (C, P, R) specifications. The growth for Eqn. (6.8) is shown in Table 6.5. Using an exponential regression on this data, we can predict the number of circuits for larger values of n_c . From this, for eight and nine components, we predict $\sim 540,320$ and $\sim 3,274,300$ circuits, respectively.

Another compelling result was the observation that a number of previous studies using evolutionary-based approaches did find at least one minimum complexity or Pareto-optimal circuit. Principally, the evolutionary approaches can arrive at such a circuit with a lower computational cost compared to an enumerative approach. However, typically only a single solution is presented with an evolutionary approach (or multiple runs are needed), but since all circuits are evaluated in an enumerative approach, a large set of alternatives and a larger set of insights is available to the designer. In the [Frequency Response Matching](#) example, the Pareto-optimal topology in Fig. 6.6e is the same as the circuit reported in Ref. [210]. In the [Low-Pass Filter Realizability](#) example, the minimum complexity topology in Fig. 6.13c is the same as the circuit reported in Ref. [211]. However, some previous studies produced overly-complex solutions [37].

Another relevant point of discussion is whether circuit sizing is performed simultaneously with or nested within the topology exploration task. Many previous approaches utilize simultaneous sizing and synthesis [37, 211]. An alternative—employed here, in Refs. [213, 217], and on the first generation in Ref. [211]—is to nest the optimal circuit component sizing task within synthesis by solving the sizing problem for each candidate topology. Here the sizing problem is solved only for unique, feasible topologies. The nested approach can converge much faster and involve fewer objective function evaluations [213]. This work also demonstrates that only a few circuits need to be sized if there is a high probability of a feasible topology. For a 99.94% chance of finding a feasible topology, only 20 circuits in task $\#1$ need to be tested, or 3,200 circuits for task $\#2$ in the LPF example. Enumeration using a nested approach looks even more favorable for certain problems if only *one* satisfactory circuit is desirable. Table 6.5 is direct evidence that it is easier to satisfy specifications with

more components [210].

Another use for the circuit generation framework in Sec. 6.2 is to provide diverse starting circuits for other synthesis approaches. The perfect-matching graph generation approach can be modified to produce stochastically sampled graphs, even for catalogs where enumeration is impractical [6]. Many authors state the importance of the initial set of circuits, including properties such as diversity and feasibility that can be assessed using the proposed enumeration method [23, 211]. The circuits generated from Ref. [65] are extremely diverse and likely feasible (e.g., there will never be an unconnected branch) but stay within a prescribed circuit structure space (such as a maximum number of components). Additionally, isomorphic topologies are minimized (or completely removed if checked directly) when compared to circuit sets generated from only random connections or other similar graph generation methods.

Looking forward, enumeration-based approaches could be leveraged to improve existing methods or help define new ones. Enumeration-based results provide a wealth of information for a particular synthesis problem. Using a set of results, feature extraction algorithms could be used to develop different metaheuristics that are based less on legacy forms or intuition (descriptive knowledge), but actual results for the problem at hand [79] to provide more normative synthesis strategies and accelerate the creation, analysis, and understanding of unprecedented systems. Enumeration could also be used in a multi-step process where smaller catalogs are initially explicated. From these results, potentially useful subcircuits could be identified and be used as components in a new circuit structure space or as in done in Fig. 6.4.

Recent work has employed machine learning strategies to scale to synthesis problems larger than the training data set generated using enumeration [235], for the simpler case of non-colored graphs. Extending the use of data from enumeration to larger synthesis problems described by colored graphs is an important topic for future work.

Finally, there are a number of general improvements that could be made to the proposed methodology. Continuous values for \mathbf{x} were assumed, but there are standardized preferred component values (such as E12 series) that are frequently desirable [23]. Solving the sizing task using a nested approach will require a different optimization technique than NLS (such as a genetic algorithm [217]). Further improvements to the graph generation code and the sizing procedure could expand the scope of problems for which enumeration is practical. Topologies including active and multi-port components are possible under the graph generating code, but modifications to the model generation and evaluation for these synthesis problems would be needed.

6.5 Summary

Here an enumeration-based synthesis methodology for passive analog circuits is described. In the enumerative approach, all circuit topologies under certain graph structure specifications are generated and tested. With a complete set of results, the most desirable circuit can be selected with guarantees on its optimality. This methodology requires minimal initial knowledge, maintains complete design space coverage, and produces reusable information. Both presented examples (frequency response matching and low-pass filter realizability) demonstrated that enumeration is feasible for certain commonly-used synthesis problems, but is also a challenge to use for sufficiently demanding synthesis tasks. The results are compared to existing approaches and show that some evolutionary approaches have produced minimum complexity or Pareto-optimal topologies. Future work is to adapt the approach for additional synthesis tasks and better understand the types of synthesis problems that enumeration is an appropriate design methodology.

Chapter 7

Case Study: Design of Strain-Actuated Solar Arrays²⁴

“What we usually consider are impossible are simply engineering problems... there’s no law of physics preventing them.”

M. Kaku [236]

7.1 Introduction

Advancements in spacecraft technology accelerate discovery in Earth and space sciences; faster reorientation and ultra-quiet jitter-free operation for space observatories and optical links have the potential to transform the rate and quality of data obtained for scientific investigation [237, 238]. Scientific needs drive exceptionally stringent spacecraft pointing and control requirements, which in turn demand new strategies for space vehicle design and control [239, 240]. The strategy proposed here uses existing appendages (solar arrays) with distributed actuation to achieve high-precision attitude control. Strain-actuated solar arrays (SASAs), which employ distributed piezoelectric material actuators, provide high accuracy and bandwidth for spacecraft attitude control, thereby supporting quiet operation for high-precision scientific instruments. Additionally, the dual use of the same spacecraft component, i.e., solar arrays, for power generation and precision attitude control reduces payload delivery costs.

A unique capability of the proposed SASA pointing architecture is to perform attitude slewing maneuvers in addition to suppressing structural vibrations. Although the current bending limit of the arrays bounds the magnitude of the attitude maneuvers to the order of milli-radians or arc minutes, these advancements are important for high precision pointing. After the pointing target has been acquired, the SASA control system performs small-scale reorientations and pointing stability in the presence of jitter disturbances.

²⁴Elements of this chapter are based on work completed in Ref. [18].

Strain-actuated solar arrays for precision pointing will require the arrays to behave more like a flexible structure than a rigid one. Space structures by necessity are extremely lightweight and flexible, but vibrations from reaction wheel assemblies, reorientation maneuvers, and other disturbances degrade performance. There has been previous work to handle this issue, including the extensively-studied topic of control-structure interaction [240, 241]. However, most of these works have led to design guidelines based on the primary goal of damping out vibrations [242, 243]. In contrast, the primary goal here is to control the spacecraft orientation. In the integrated design and control study presented here we seek to utilize the flexible body dynamics to our advantage to provide new levels of system performance.

Piezoelectric material actuators (PEMAs) [244–246] are a proven solution for distributed actuation [247, 248]. Applying voltage across PEMAs bonded to or embedded within structures induces strain, causing the structure to deform (bend, twist, elongate, or contract depending on design). PEMA-based intelligent structures outperform conventional point-actuated structures [240, 249], in terms of mass, fast dynamic response, and high precision [250]. Although piezoelectric actuators have been used for structural active damping, they have not been used for slewing control of structures due to their small stroke. The proposed SASA architecture, however, does not slew the array structure about a revolute joint, but bends it to slew the bus using conservation of angular momentum. In this way, the SASA system implements the novel functionality of small-scale attitude control. Since there are small strain limits on array bending, the small PEMA stroke does not limit this application. Furthermore, the use of piezoelectric actuators allows for quiet operation for sensitive instruments.

The distributed actuation of intelligent structures provides tremendous design flexibility. This opens up new opportunities for system performance, but also increases design difficulty [250, 251]. The co-design work presented here considers not only the design of the actuator system, but also of a distributed structure for optimal active performance. Furthermore, the use of open-loop distributed control allows us to obtain insights for actuator placement and to investigate limits of performance.

In most previous co-design studies, the physical aspects of the system design have been managed in a very simplified manner. For example, physical system (plant) design decisions have often been limited to actuator placement [252, 253]. Many co-design studies have used simplified plant models [4, 249, 254, 255] that do not support exploration of changes to distributed geometric structural design, preventing full exploitation of the design synergy between structural tailoring and distributed control system design. A more ideal co-design

method supports changes to distributed structural properties (e.g., changing structural shape affects how inertial and stiffness properties vary spatially). Structural tailoring coupled with control design has long been recognized as an important, yet formidable problem [101]. Although there are examples of tailoring passive system dynamics to work optimally with active control using lumped plant stiffness, damping, and mass parameters as design variables [254], these methods cannot be extended to distributed parameter systems.

To summarize, much is known regarding the design of control systems and actuators for intelligent structures, but only if the structural design is held fixed. A few examples of fully-integrated design exist, but only with simplified treatment of structural design. Since the proposed SASA pointing architecture involves inherent dynamic coupling between control actuation and flexible structural dynamics, it is a good case for a co-design study. In this work, distributed geometry—specifically, distributed array structure thickness—is optimized simultaneously with distributed moment control of the array structure.

An initial study of the SASA concept was performed previously, focusing on attitude control, to demonstrate its feasibility. It was shown that the spacecraft bus orientation can be controlled by the appropriate bending of the arrays using a pseudo-rigid body dynamic model (PRBDM). A more recent study has further developed realistic feedback control systems suitable for SASA architectures [256–258]. An earlier version [259] of this work introduced the use of Euler-Bernoulli beam theory for a physically consistent description of the array dynamics, and the nonlinear partial differential equation (PDE) model is implemented using Galerkin approximating functions [260–262]. In the model, piezoelectric actuation is represented by a distributed moment on the array. The model also accounts for the elastic and inertial properties of the actuators. In the spacecraft slewing and pointing maneuvers considered here, the array bending displacement is small. This enables reasonable accuracy when using a linearized bus-array model. Based on this linear model, we present parametric studies that 1) help determine performance limits, and 2) provide insight into the resulting array designs.

The use of open-loop controls and distributed optimization parameters, e.g., voltage and array thickness, allows for solutions that make limited assumptions on the control or physical architecture. This aids in the exploration of ultimate system performance limits [27]. Although there may be practical constraints for feedback control system implementation, the resulting co-design solutions can provide important insights into how to design the physical array structure such that it performs optimally as an actively controlled system, capitalizing on synergy between physical- and control-system design [1].

The primary objectives of this work are to demonstrate the feasibility of the SASA attitude

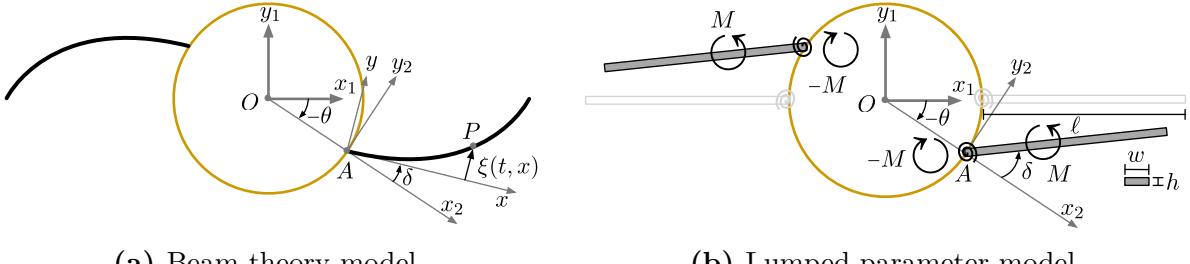


Figure 7.1: Illustration of the two modeling approaches used to gain design insights.

control architecture on a representative spacecraft system, to determine the optimal designs of the distributed array structure and controls, and to reveal qualitative design insights for intelligent structures with distributed geometric design.

This chapter is organized as follows. The models for the bus-array system, distributed composite array structure, distributed control, and PRBDM are presented in Sec. 7.2. The formulation for the combined design of the distributed array structure and distributed control is presented in Sec. 7.3. Analytical results based on PRBDM theory and numerical results of the co-design studies are discussed in Sec. 7.4. Results include the analysis of the optimal design tradeoff for the array structure, the optimal placement of segmented piezoelectric actuators, and parametric studies on passive damping and jitter disturbance.

7.2 Modeling of the Strain-Actuated Solar Arrays and Rigid Spacecraft Bus

The modeling approach used here is based on the recent work on aircraft dynamics with flexible, articulated wings [263] (see Ref. [264] for details). The spacecraft motion is modeled as an ordinary differential equation (ODE) of a simple cylinder, and the solar array structure is modeled as a PDE of a composite beam with thickness that can vary along its length.

7.2.1 Partial Differential Equation Model

Here we assume that actuation is effected only through solar array strain actuators that produce strain at the solar array structure surface, resulting in array bending and a distributed moment due to strain actuator surface forces. The strain actuators do not interact with anything external to the spacecraft system, so the total system momentum must be

conserved. Therefore, for a generally counter clockwise (CCW) movement of the solar array, the bus (θ) will rotate in the opposing CW direction allowing for attitude changes. This is evident in both the illustration of the beam theory coordinate system in Fig. 7.1a and its comparable PRBDM lumped parameter model in Fig. 7.1b.

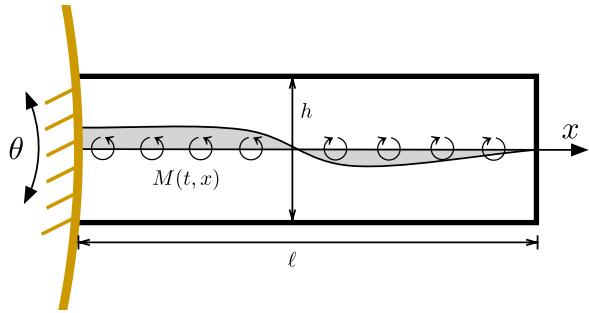
7.2.1.1 Coupled ODE-PDE Dynamic Model

The coordinate systems used for the derivation of the Lagrangian of the system are shown in Fig. 7.1a. The model has two arrays with asymmetric actuation. Let the radius of the spacecraft body be r , and the spacecraft body rotation angle about origin O be θ . In deriving the equations of motion, it is assumed that the deflections $\xi(x, t)$ due to bending are small and the beam has no longitudinal velocity.

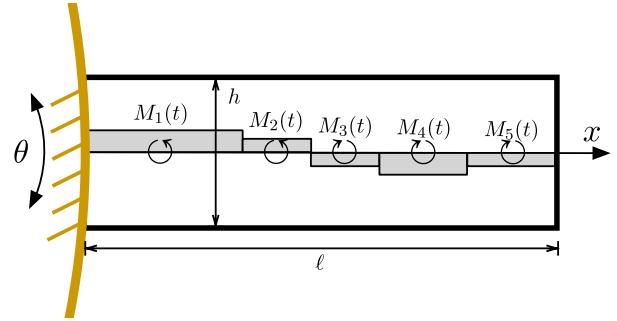
The mass moment of inertia of the spacecraft bus is J_θ . The total length and width of the solar array are represented by ℓ and w . The mass per unit length of the composite beam is denoted $m_R(x)$ and the total rigidity is $E(x)I(x)$. The further details of the structural model are discussed in Sec. 7.2.1.3.

The moment applied on the array is $M(x, t)$ over the locations where a piezoelectric actuator is bonded; a small actuation gap (0.5 cm) was applied at the root and the tip to satisfy the boundary conditions. Using the explicit generalization for the hybrid coordinate systems approach, the equations of motion were derived in Refs. [18, 264]. The applied boundary conditions specify zero deflection and slope of the deflected beam at the root, and no external force or moment at the free end.

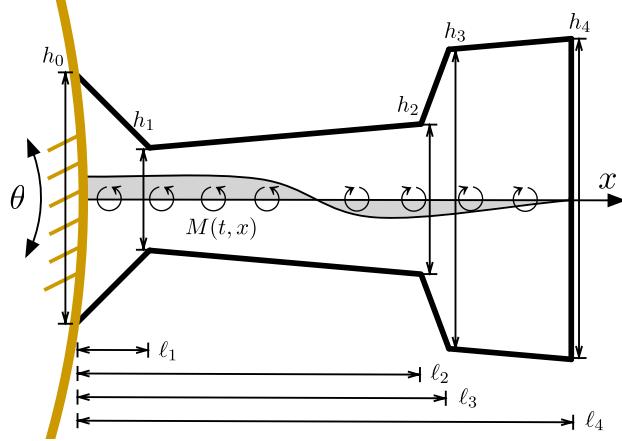
The proposed SASA architecture is envisioned for high-precision pointing control. To maintain the structural integrity of the arrays and to take into account actuator limitations, bounds are defined for the array strain and control magnitude, which in turn limit array displacements to small values. Therefore, a linearized bus-array system can still predict the dynamics with sufficient accuracy (see Ref. [18] for details). While the linear model does not approximate large array displacements accurately, displacements in our tests are small, and linearization makes the integrated structural and control optimization problem more tractable. This allows to conduct various parametric studies, which support the focus of this work on design methods and design insight. The following linearized equations are now



(a) Continuously distributed internal moment on a uniform thickness array.



(b) Piecewise constant distributed internal moment on a uniform thickness array.



(c) Piecewise linear distributed array thickness.

Figure 7.2: Illustrations of various design representations for internally actuated array design problems for pointing.

used, and the boundary conditions remain the same:

$$\int_0^\ell [\mathbf{M}_{sl}] \begin{bmatrix} \ddot{\theta} \\ \ddot{\xi} \end{bmatrix} dx + \begin{bmatrix} 0 \\ \int_0^\ell (2EI\xi'' + 2\mu EI\dot{\xi}'')'' dx \end{bmatrix} = \begin{bmatrix} d \\ \int_0^\ell 2M'' dx \end{bmatrix} \quad (7.1)$$

where: $[\mathbf{M}_{sl}] = \begin{bmatrix} m_{11}(\xi) & m_{12} \\ m_{12} & m_{22} \end{bmatrix} = \begin{bmatrix} (J_\theta/\ell + 2(m_R(x+r)^2)) & 2m_R(x+r) \\ 2m_R(x+r) & 2m_R \end{bmatrix}$

The term μ is used to model the structural damping in the solar array. A disturbance $d(t)$ acts on the bus as a torque input.

7.2.1.2 Galerkin Formulation

To approximate numerically the PDE in Eqn. (7.1), we use a Galerkin formulation [261–263]. A linear combination of approximating functions is used to represent the array dynamic state [261] and distributed moment. These functions are chosen such that they satisfy the boundary conditions of the array, i.e., the fixed-free condition. The j th approximating functions used for spatially distributed deflection and moment representations, respectively, are defined as [261]:

$$\phi_j(x) = 1 - \cos\left(\frac{j\pi x}{\ell}\right) + \frac{1}{2}(-1)^{j+1}\left(\frac{j\pi x}{\ell}\right)^2, \quad \gamma_j(x) = \phi_j(x) + x^j + 1 \quad (7.2)$$

The array deflection and distributed moment are then approximated as:

$$\xi(x, t) = \boldsymbol{\phi}(x)^T \boldsymbol{\eta}(t), \quad M(x, t) = \boldsymbol{\gamma}(x)^T \mathbf{q}(t) \quad (7.3)$$

For the co-design studies here, four approximating functions are used. This approximation parameterizes the control as a spatially-varying distributed moment, but the actual control input on a piezoelectric segment is normally a uniform voltage [265]. Comparing both of these representations in Figs. 7.2a and 7.2b, we may think of the spatially-varying distributed moment as the limiting case of the piecewise uniform moment (segment length approaching zero). The applicability of this approximation to a real implementable physical system will be discussed in Sec. 7.4.4.

A system of ODEs that approximate the PDE given above is derived [264] by minimizing weighted residual of the ξ dynamics. Using the above formulation and defining additional matrices we obtain:

$$[\mathbf{M}_g] \begin{bmatrix} \ddot{\theta} \\ \ddot{\boldsymbol{\eta}} \end{bmatrix} + \begin{bmatrix} 0 \\ 2[e](\boldsymbol{\eta} + \mu\dot{\boldsymbol{\eta}}) \end{bmatrix} = \begin{bmatrix} d \\ \int_0^\ell 2\boldsymbol{\phi} M'' dx \end{bmatrix} \quad (7.4)$$

where: $[\mathbf{M}_g] = \begin{bmatrix} J_\theta + 2 \int_0^\ell m_R (x+r)^2 dx & 2[\mathbf{B}] \\ 2[\mathbf{B}]^T & 2[\mathbf{A}] \end{bmatrix}$

$$[\mathbf{A}] = \int_0^\ell m_R \boldsymbol{\phi} \boldsymbol{\phi}^T dx, \quad [\mathbf{B}] = \int_0^\ell m_R (x+r) \boldsymbol{\phi}^T dx, \quad [e] = \int_0^\ell \boldsymbol{\phi} (EI \boldsymbol{\phi}'^T)'' dx$$

7.2.1.3 Structural Model of Composite Array

The structural geometry of the array is also designed with the distributed moment. In this work, the length of the array and the distributed thickness are optimized. The spatially varying array thickness design is represented using piecewise linear segments. The length

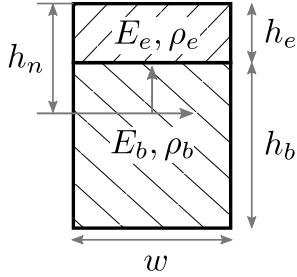


Figure 7.3: Cross section of the actuated array, modeled as a composite beam.

of the array is divided into multiple segments as shown in Fig. 7.2c. Segment lengths and slopes can be changed. The distributed array thickness design is parameterized using the absolute locations of the segment boundaries (quantified by the vector ℓ), and the thickness at the segment boundaries (quantified by the vector h). On the segment j , the thickness varies linearly with respect to x as follows:

$$h_j(x) = (h_{j+1} - h_j) \frac{x - \ell_j}{\ell_{j+1} - \ell_j} + h_j, \quad x \in [\ell_j, \ell_{j+1}] \quad (7.5)$$

The array is laminated with a layer of piezoelectric material on the top surface which acts as an actuator and has the same width as the beam. It is also assumed that the entire top surface of the array is covered with a piezoelectric layer of constant thickness h_e . The neutral axis of the composite beam is not at the center due to the inhomogeneous structural properties. Consider the cross section of the composite beam shown in Fig. 7.3. The distance from the neutral axis and the top surface of the array with piezoelectric material is h_n . The thickness of the base array and the piezoelectric layer are $h_b(x)$ and h_e , respectively. Note that $h_b(x)$ can vary spatially. The neutral axis location h_n , for each position $0 \leq x \leq \ell$, can be obtained by balancing the forces across the cross section and solving for h_n :

$$h_n = \frac{0.5E_e h_e^2 + E_b h_b (0.5h_b + h_e)}{E_e h_e + E_b h_b} \quad (7.6)$$

The area moments of inertia of the array, I_b , and the piezoelectric layer, I_e , about the neutral axis are:

$$I_b = \frac{wh_b^3}{12} + wh_b (h_e + 0.5h_b - h_n)^2, \quad I_e = \frac{wh_e^3}{12} + wh_e (h_n - 0.5h_e)^2 \quad (7.7)$$

The total array rigidity is given by:

$$EI = E_b I_b + E_e I_e \quad (7.8)$$

The mass per unit length of the composite array is $m_R(x) = m_{Rb}(x) + m_{Re}(x)$, where $m_{Rb}(x)$

and $m_{Re}(x)$ are the mass per unit length of the base array and the piezoelectric material, respectively. The application of a voltage V across the piezoelectric layer induces a moment M . This moment, due to only internal actuation, can be calculated by applying force balance across the cross section of the composite array [264]:

$$M(x, t) = c(x)V(x, t), \quad (7.9)$$

$$\text{where: } c(x) = \frac{d_{31}E_b^2wh_b^3E_eh_b(h_b + h_e)}{2(E_b^2h_b^4 + 4E_bE_eh_b^3h_e + 6E_bE_eh_b^2h_e^2 + 4E_bE_eh_bh_e^3 + E_e^2h_e^4)}$$

and d_{31} is the ratio of the strain and the electric field applied across the piezoelectric layer when all the external forces are held constant [265]. Observe that the moment is proportional to the applied voltage.

7.2.2 Pseudo-Rigid Body Dynamic Model

A PRBDM was developed for the spacecraft system for the purpose of performing additional numerical studies that complement those based on the PDE model, including studies that yield qualitative insights that are difficult to obtain via the more sophisticated PDE model. The flexible solar arrays were modeled both with single and multi-link approximations. The single link model is presented here, where each array is modeled as a single rigid link connected to the spacecraft body via a revolute joint and a torsional spring (see Fig. 7.1b). This is a lumped compliance approximation of the distributed compliance of the actual solar array. However, these models only describe the behavior at a component level rather than the specific point-to-point variations, while Euler-Bernoulli beam in Sec. 7.2.1 does capture these variations.

Applying the Euler-Lagrange equation to the system in Fig. 7.1b we arrive at the following equations of motion:

$$\mathbf{M}_{\text{PR}} \begin{bmatrix} \ddot{\theta} \\ \ddot{\delta} \end{bmatrix} + \mathbf{B}_{\text{PR}} \begin{bmatrix} \dot{\theta} \\ \dot{\delta} \end{bmatrix} + \mathbf{K}_{\text{PR}} \begin{bmatrix} \theta \\ \delta \end{bmatrix} = \boldsymbol{\tau} \quad (7.10)$$

$$\text{where: } \mathbf{M}_{\text{PR}} = \begin{bmatrix} J_\theta + 2J_\delta + 2mr^2 + \frac{1}{2}m\ell^2 + 2\ell mr \cos(\delta) & 2J_\delta + mrl \cos(\delta) + \frac{1}{2}m\ell^2 \\ 2J_\delta + mrl \cos(\delta) + \frac{1}{2}m\ell^2 & 2J_\delta + \frac{1}{2}m\ell^2 \end{bmatrix}$$

$$\mathbf{B}_{\text{PR}} = \begin{bmatrix} -2\ell mr\dot{\delta} \sin(\delta) & -\ell mr\dot{\delta} \sin(\delta) \\ \ell mr\dot{\theta} \sin(\delta) & 2b \end{bmatrix}, \quad \mathbf{K}_{\text{PR}} = \begin{bmatrix} 0 & 0 \\ 0 & 2k \end{bmatrix}, \quad \boldsymbol{\tau} = \begin{bmatrix} d \\ 2M \end{bmatrix}$$

$$m = \rho\ell wh, \quad J_\delta = \frac{1}{12}m(\ell^2 + h^2)$$

and where k is the torsional spring stiffness and b is the damping constant at the revolute joints. Solving the eigenvalue problem ($\mathbf{M}_{\text{PR}}^{-1}\mathbf{K}_{\text{PR}} = \boldsymbol{\omega}^2$) gives the natural frequencies of the system:

$$\omega_1^2 = 0 \quad (7.12a)$$

$$\omega_2^2 = k \frac{2m\ell^2 + 8m\ell r \cos(\delta) + 8mr^2 + 4J_\theta + 8J_\delta}{-2\ell^2 m^2 r^2 \cos(\delta)^2 + 2\ell^2 m^2 r^2 + J_\theta \ell^2 m + 8J_\delta mr^2 + 4J_\theta J_\delta} = \frac{k}{J_{\text{eff}}(\delta, \ell, h, w)} \quad (7.12b)$$

One of the eigenfrequencies is zero since the system permits a rigid body mode. The two mode shapes are:

$$\boldsymbol{\psi} = [\boldsymbol{\psi}_1 \ \boldsymbol{\psi}_2] = \begin{bmatrix} 1 & -\frac{m\ell^2 + 2m\ell r \cos(\delta) + 4J_\delta}{m\ell^2 + 4m\ell r \cos(\delta) + 4mr^2 + 2J_\theta + 4J_\delta} \\ 0 & 1 \end{bmatrix} \quad (7.13)$$

We note that the nonrigid mode eigenfrequency (ω_2) and eigenvector ($\boldsymbol{\psi}_2$) are not constant but depend on the path of the array. The total angular momentum of the system is:

$$\mathcal{X} = \left(\frac{1}{2} \ell^2 m + 2\ell m r \cos(\delta) + 2mr^2 + J_\theta + 2J_\delta \right) \dot{\theta} + \left(\frac{1}{2} \ell^2 m + \ell m r \cos(\delta) + 2J_\delta \right) \dot{\delta} \quad (7.14)$$

Since internal moments cannot change the total angular momentum of the system, the only mode that is present in the absence of external moments is the momentum conserving mode $\boldsymbol{\psi}_2$.

7.3 Co-Design Problem Formulation

The objective of the co-design study is to provide insights into how the actively-controlled solar array should be designed to optimize the performance in terms of attitude slewing and jitter reduction. A balanced co-design approach is utilized where physical-system (geometric specification of the solar array) and control-system (open-loop voltage trajectories) design are considered in an equally comprehensive manner [27]. A general simultaneous co-design formulation (with a fixed time horizon) is in Prob. (3.1). Here the physical system design \mathbf{x}_p is parameterized by \mathbf{h} and ℓ with eight distinct linear segments. The control system design variable is defined here as $\mathbf{x}_c := \mathbf{q}(t)$; these control trajectories are used to compute $M(x, t)$ and $V(x, t)$.

The spacecraft control task is divided into two consecutive phases; t_m is the time duration of the first phase. The first phase (slewing) concentrates on rotating the bus from an initial angular displacement $\theta(t_0)$ back to $\theta = 0$ at time t_m . In the second phase (pointing) the

(a) Bus parameters.		(b) Array and piezoelectric material parameters.	
Parameter	Value	Parameter	Value
J_θ	372.49 kg m^2	ℓ_{nominal}	1.575 m
r	1.02 m	h_{nominal}	0.018 m
		w_{nominal}	1.862 m
		h_p	$2 \times 10^{-4} \text{ m}$
		E_b	1.57 GPa
		ρ_b	332.03 kg/m^3
		E_p	62 GPa
		ρ_p	7800 kg/m^3
		μ	10^{-4} s

Table 7.1: Problem physical parameters.

bus is held inertially fixed for precision pointing. Any vibrations generated during slewing must be damped out during the pointing phase. The objective function is to maximize the slewing angle, demonstrating the maximum capability of the SASA attitude control system:

$$\Psi = -\theta(t_0) \quad (7.15a)$$

where $t \in [t_0, t_f] = [0 \text{ s}, t_m + 1 \text{ s}]$ and t_m is solved at various values between 0.12 s and 4 s. This parametric sweep on t_m helps quantify the tradeoff between the competing objectives of slew angle maximization and slew time minimization. The dynamic constraint uses the linear ODE defined in Eqn. (7.4) with 4 approximating functions. The physical parameters for the bus, array, and piezoelectric layer are shown in Table 7.1. A disturbance moment $d(t)$ on the bus is present during the slewing and pointing phases. It consists of a jitter component (e.g., vibrations similar to those that arise from moving parts such as pumps) and a bias component (e.g., due to solar radiation pressure, atmospheric drag):

$$d(t) = 10^{-4} \text{ Nm} + 2 \times 10^{-3} \sin(50t) \text{ Nm} \quad (7.15b)$$

The initial configuration is stationary with an initial bus orientation $\theta_0 = \theta(t_0) \neq 0$. The initial states²⁵ are:

$$\xi(t_0) = \begin{bmatrix} \theta(t_0) \\ \dot{\theta}(t_0) \\ \boldsymbol{\eta}(t_0) \\ \dot{\boldsymbol{\eta}}(t_0) \end{bmatrix} = \begin{bmatrix} \theta_0 \\ 0 \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \quad (7.15c)$$

²⁵The states ξ include the deflection ξ of the array in Fig. 7.1a through $\boldsymbol{\eta}$.

The bus angle and angular rate are constrained to 0 during the pointing phase. This simultaneously meets the pointing task and eliminates jitter if a feasible solution is found:

$$|\theta(t)| = 0, \quad t \in [t_m, t_f] \quad (7.15d)$$

$$|\dot{\theta}(t)| = 0, \quad t \in [t_m, t_f] \quad (7.15e)$$

Numerical experiments indicate these constraints can be satisfied in all but very unusual cases. PEMA actuation magnitude is constrained to satisfy maximum voltage restrictions:

$$|V(x, t)| = \left| \frac{M(x, t)}{c(x)} \right| \leq 300 \text{ V} \quad (7.15f)$$

The array surface strain is constrained to be less than 0.1% to maintain structural integrity:

$$|\epsilon(x, t)| = |(h_b(x) + h_e - h_n(x)) \xi''(t)| \leq 10^{-3} \quad (7.15g)$$

Traditional silicon-based solar cells can withstand strain levels on the order of 0.1% [266]. However, recent advances allow the manufacturing of flexible solar cells that can achieve strain levels on the order of 10% [267], as well as fiber-shaped solar cells that can be woven into textiles [268].

The total array length, linear array segment lengths, and array thickness values have manufacturing and operational constraints:

$$\begin{aligned} 0.5 \text{ m} &\leq \ell_n \leq 2.5 \text{ m} \\ 0.05 \text{ m} &\leq \ell_{i+1} - \ell_i \leq 1 \text{ m} \quad i = 0, 1, \dots, n \end{aligned} \quad (7.15h)$$

$$0.009 \text{ m} \leq h_i \leq 0.055 \text{ m}$$

The array volume is constrained to be less than the nominal value in order to avoid increasing the payload mass and delivery costs. This is proportional to the array structure cross-sectional area:

$$\sum_{i=1}^n \frac{w}{2} (h_{i-1} + h_i) (\ell_i - \ell_{i-1}) \leq 0.054 \text{ m}^3 \quad (7.15i)$$

Array planform area is constrained to the nominal value to maintain the same level of power generation:

$$\ell_n w = 2.932 \text{ m}^2 \quad (7.15j)$$

Direct orientation of the solar arrays towards the sun requires the attitude rotation axis to be normal to the sun-spacecraft vector. Solar power generation is a function of array

area and orientation (and other factors). The planform area constraint is intended to be large enough to ensure adequate overall power generation even when arrays are not oriented directly toward the sun. A more sophisticated and accurate approach would model power generation directly across a range of maneuvers and insolation conditions. An investigation of the tradeoff between power generation and attitude control is a topic for future work.

This completes the exposition of the co-design problem formulation; objective and constraint functions are summarized in Table 7.2. Note that some optimization variables (states and controls) are linear while other variables (plant parameters) are nonlinear in the co-design formulation. A traditional approach to solve this type of problems is to use nested co-design [27, 32, 83]. This approach consists of an outer-loop that finds the optimal plant parameters, while the inner-loop finds the optimal states and controls histories for each point in the plant parameter space sampled by the outer-loop. In this way, the inner-loop can return the cost of a particular plant design to the outer-loop. Since the the optimal control problem in the inner-loop is linear, it can be formulated and solved as a linear program (LP) using a direct transcription method [25, 95, 96, 269]. Direct transcription has been used traditionally in trajectory optimization [95, 270, 271]. The custom transcription code in Chapter 5 was used to transform the infinite dimensional optimal control problem into a finite dimensional optimization problem. The structure-based description is in the following section.

A feasible solution for the inner-loop problem is globally-optimal because it is formulated as an LP program. However, the nonlinear dependencies on the plant parameters in the outer-loop require special attention to ensure that the global optimum is found. The outer loop is implemented using MATLAB PATTERNSEARCH [272], while the inner-loop is implemented using MATLAB QUADPROG [185]. MATLAB PATTERNSEARCH was configured to use complete search (Latin hypercube sampling) and polling options to help find the global solution.

In an earlier version of this work [259], the nonlinear dynamics were directly used. Therefore, the methods from Chapter 5 were not applicable. A nonlinear dynamic optimization software package was instead used to solve the problem using the simultaneous co-design method from Chapter 3. However, there were a number of problems with this strategy. The computation time was extremely large, making it impossible to perform tests on a variety of starting points or problem parameters, so there was less confidence in the global optimality of the solution and less information gained. One reason for this large computation time was the fact that the plant-dependent quantities needed to be updated frequently when the simultaneous approach was used. The nested approach, on the other hand, typically tests a smaller number of candidate plant designs. Additionally, the control was not guaranteed to be the global optimum, as is the case here.

Name	Eqn. #	Linear w.r.t.				Level
		ξ	x_c	x_p		
Max. Slew Amount	Eqn. (7.15a)	Yes				Both
Dynamics	Eqn. (7.4) & Eqn. (7.15b)	Yes	Yes	No	Inner	
Initial Conditions	Eqn. (7.15c)	Yes				Inner
Pointing	Eqn. (7.15d) & Eqn. (7.15e)	Yes				Inner
Voltage Limits	Eqn. (7.15f)		Yes	No	Inner	
Strain Limits	Eqn. (7.15g)	Yes		No	Inner	
Geometry Bounds	Eqn. (7.15h)			Yes	Outer	
Array Volume	Eqn. (7.15i)			No	Outer	
Planform Area	Eqn. (7.15j)			No	Outer	

Table 7.2: Summary of co-design problem formulation.

The results for a number of minor variations of this formulation will be discussed next after a short study on the fundamental limits of a slewing maneuver with SASA utilizing a reduced form of this co-design formulation and the PRBDM model.

7.3.1 Inner-Loop Structure-Based Description

Here we present the structure-based description for the LQDO problem. The approach from Chapter 5 is utilized twice, once for each phase. Then the LPs created for each phase are linked with linear continuity constraints (see Sec. 5.7.1) for the states.

The maximum slew objective in Eqn. (7.15a) is implemented with a single Mayer term:

$$\mathcal{M}\langle 1 \rangle.\text{left} = 0, \quad \mathcal{M}\langle 1 \rangle.\text{right} = 4, \quad \mathcal{M}\langle 1 \rangle.\text{matrix} = [-1 \ 0] \quad (7.16)$$

The initial conditions in Eqn. (7.15c) are implemented with simple bounds in the first phase:

$$\mathcal{UB}\langle \cdot \rangle.\text{right} = 2, \quad \mathcal{UB}\langle \cdot \rangle.\text{matrix} = [\infty \ 0]^T \quad (7.17a)$$

$$\mathcal{LB}\langle \cdot \rangle.\text{right} = 2, \quad \mathcal{LB}\langle \cdot \rangle.\text{matrix} = [-\infty \ 0]^T \quad (7.17b)$$

The bus angle and angular rate equality constraints in Eqns. (7.15d)–(7.15e) are implemented with simple upper and lower bounds only in the second phase:

$$\mathcal{UB}\langle \cdot \rangle.\text{right} = 2, \quad \mathcal{UB}\langle \cdot \rangle.\text{matrix} = [0 \ 0 \ \infty]^T \quad (7.18a)$$

$$\mathcal{LB}\langle \cdot \rangle.\text{right} = 2, \quad \mathcal{UB}\langle \cdot \rangle.\text{matrix} = [0 \ 0 \ -\infty]^T \quad (7.18b)$$

The absolute value limits on the voltage in Eqn. (7.15f) are implemented with linear inequal-

ity at each test point x_i in both phases:

$$\mathcal{Z}(\cdot).linear\langle 1 \rangle.right = 1, \quad \mathcal{Z}(\cdot).linear\langle 1 \rangle.matrix = \gamma(x_i), \quad \mathcal{Z}(\cdot).b = V_{\max}c(x_i) \quad (7.19a)$$

$$\mathcal{Z}(\cdot).linear\langle 1 \rangle.right = 1, \quad \mathcal{Z}(\cdot).linear\langle 1 \rangle.matrix = -\gamma(x_i), \quad \mathcal{Z}(\cdot).b = V_{\max}c(x_i) \quad (7.19b)$$

The absolute value limits on the strain in Eqn. (7.15g) are implemented with linear inequality at each test point x_i in both phases:

$$Z = \begin{bmatrix} 0 & 0 & (h_b(x_i) + h_e - h_n(x_i))\phi''(x_i) & \mathbf{0} \end{bmatrix}^T \quad (7.20a)$$

$$\mathcal{Z}(\cdot).linear\langle 1 \rangle.right = 2, \quad \mathcal{Z}(\cdot).linear\langle 1 \rangle.matrix = Z, \quad \mathcal{Z}(\cdot).b = \epsilon_{\max} \quad (7.20b)$$

$$\mathcal{Z}(\cdot).linear\langle 1 \rangle.right = 2, \quad \mathcal{Z}(\cdot).linear\langle 1 \rangle.matrix = -Z, \quad \mathcal{Z}(\cdot).b = \epsilon_{\max} \quad (7.20c)$$

7.4 Analytical and Numerical Results for SASA System

7.4.1 Maximum Slewing Bounds using the PRBDM

The momentum of the PRBDM system is given in Eqn. (7.14), and it must be conserved if no external disturbance acts on the spacecraft ($d \equiv 0$). Assuming zero initial momentum, we can integrate the momentum equation to determine the relationship between θ and δ :

$$\begin{aligned} 0 &= \left(\frac{1}{2}\ell^2m + 2\ell mr \cos(\delta) + 2mr^2 + I_\theta + 2I_\delta \right) \dot{\theta} + \left(\frac{1}{2}\ell^2m + \ell mr \cos(\delta) + 2I_\delta \right) \dot{\delta} \\ &:= I_1(\ell, h, w, \delta)\dot{\theta} + I_2(\ell, h, w, \delta)\dot{\delta} \\ \dot{\theta} &= -\frac{I_2(\delta, \cdot)}{I_1(\delta, \cdot)}\dot{\delta} \\ \theta(t_f) - \theta(t_0) &= - \int_{t_0}^{t_f} \frac{I_2(\delta, \cdot)}{I_1(\delta, \cdot)} \dot{\delta} dt \end{aligned} \quad (7.21)$$

The question we are trying to answer requires an upper bound on $|\theta(t_f) - \theta(t_0)|$. We can find a reasonable upper bound by determining the maximum value of the integrals. Recall that m and I_δ are dependent on the geometric physical design variables. Since the geometric variables are positive and it is reasonable to assume $\cos(\delta) > 0$ (panel angle must be smaller than $|\delta| < \pi/2$), then we see that the following difference is strictly positive: $I_1 - I_2 = \ell mr \cos(\delta) + 2mr^2 + I_\theta > 0$. Therefore, the effective inertia ratio, $R_{\text{eff}} := I_2/I_1$, must be between 0 and 1.

If $|\delta(t)| \leq \delta_{\max}$ is small, then R_{eff} is nearly time-independent. Observe also that the

Study	$R_{\text{eff},\max}$	$R_{\text{eff},\max}\delta_{\max}$	Actual
Nominal Geometry	0.12	0.0108 rad (0.62°)	> 0.0063 rad (0.36°)
Maximal ℓ	0.22	0.0301 rad (1.73°)	> 0.0176 rad (1.01°)

Table 7.3: Summary of results for maximum slewing bounds using the PRBDM.

effective inertia ratio is maximized when $\delta = 0$. Therefore, we will use this maximal value, denoted $R_{\text{eff},\max}$, as a time-independent value to arrive at the following inequality:

$$\begin{aligned} |\theta(t_f) - \theta(t_0)| &\leq R_{\text{eff},\max} \left| - \int_{t_0}^{t_f} \dot{\delta} dt \right| \\ &\leq R_{\text{eff},\max} |\delta(t_0) - \delta(t_f)| \end{aligned} \quad (7.22)$$

Assuming $\delta(t_0) = 0$, $\theta(t_f) = 0$, and that the prescribed bound on δ is hit at t_f , then we have:

$$|\theta_0| \leq R_{\text{eff},\max} \delta_{\max} \leq \delta_{\max} \quad (7.23)$$

Therefore we expect the maximum change for the bus angle to be bounded above by the allowable change in panel angle using only internal actuation of the solar array. This implies that if only milli-radian deflections of the array are feasible, we can only achieve, at most, milli-radian changes in bus orientation. Additional novel solutions for SASA such as revolute joints that allow for large changes in the effective inertia ratio without violating conservation of momentum may extend this limit.

A comparable δ_{\max} condition for a continuous array is a strain bound. If we want the largest value for δ , analogously the strain will be at its maximum possible value at each point on the arrays. Using the constant thickness solar array in Fig. 7.1b, we can calculate this bound as $\delta_{\max} = (\ell/h)\epsilon_{\max}$. With this relationship, we can calculate the predicted maximum slewing bound. Two cases are shown in Table 7.3: nominal geometry and maximum allowable value for ℓ . The latter case achieves the maximum value of the slew bound since both $R_{\text{eff},\max}$ and δ_{\max} are maximized. We can also determine the actual maximum slew bound using the optimal control problem in Sec. 7.3 (i.e., fixed geometry) without the voltage constraint. This result is also shown in Table 7.3 and indeed the bound is verified. The bound is not tight because the optimal voltage trajectories did not strain all of the array to the prescribed bound but only most of the array; thus, this assumption was only partially valid. In the next section, the study will consider the other design constraints and allow piecewise-linear array thickness changes. The bounds in this section do not consider all the design constraints and therefore are only approximate indications of the maximum possible slewing performance.

Variation	Array Property		
	Length (m)	Thickness (m)	Volume (m^3)
NG	1.575	0.018	0.054
VL	$0.5 \leq \ell \leq 2.5$	0.018	0.054
PLS	$0.5 \leq \ell \leq 2.5$	$0.009 \leq h(x) \leq 0.055$	≤ 0.054

Table 7.4: Geometric constraints for the co-design problem variations.

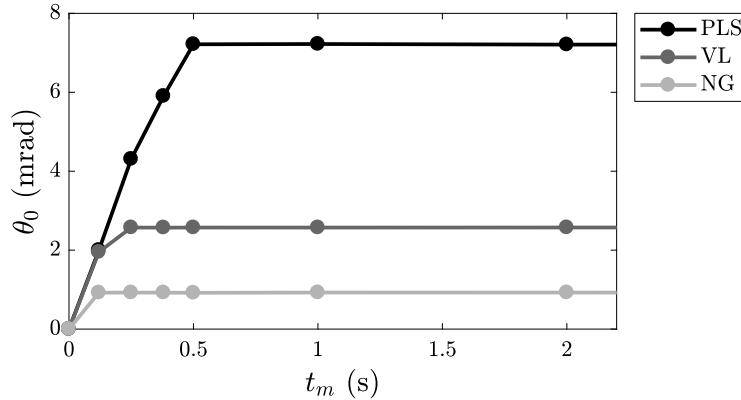
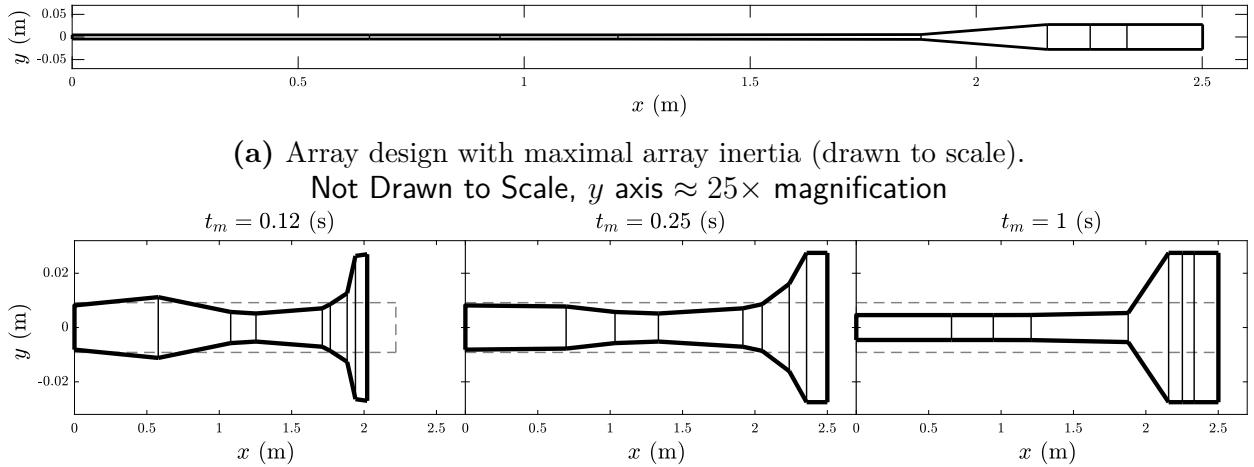


Figure 7.4: Parametric study of maximum slewing angle with respect to slewing time.

7.4.2 Maximum Slewing Bounds using the Co-Design Problem Formulation

In this section we study three variations of geometric design representation in the co-design problem introduced in Sec. 7.3. These are denoted nominal geometry (NG), variable length (VL), and piecewise linear segments (PLS). The geometric constraints for each case are summarized in Table 7.4. The NG case does not involve physical-system design since it is fixed and is only included in this study as a performance baseline. For the VL case, ℓ is the sole physical-system design variable (see Fig. 7.2a). Finally, the PLS case varies total array length, segment lengths, and spatially-varying thickness to modify distributed geometric design of the array structure (see Fig. 7.2c). The array volume (or mass) in the PLS case can be smaller than the nominal, whereas in the VL case, the volume is fixed to the nominal value of the NG case. Since the array planform area is constrained to a nominal value in each case, the array width is determined by the array length. Several values of the slewing time, t_m , between 0.12 s and 4 s were studied to investigate the relationship between slew time, the maximum slew angle, and corresponding optimal array designs. The final time of the simulation is given by $t_f = t_m + 1$.

The maximum slewing results for each of the three variations are summarized in Fig. 7.4. As expected, the NG case achieved the smallest maximum slewing angle (0.9 mrad) for all



(b) Array designs for both Variable Length (dashed) and Piecewise Linear Segments (solid) studies.

Figure 7.5: Optimal array designs.

the tested slewing times. This result indicates that the performance level desired may not be achievable through control design alone. Furthermore, the peak maximum achievable slewing angles for the VL and PLS cases (2.6 mrad and 7.2 mrad, respectively) are consistent with the results of Sec. 7.4.1. The optimal array designs are shown in Fig. 7.5b. We observed that the optimal PLS geometries for slewing times ≥ 0.5 s were similar. In addition, optimal VL designs for slewing times ≥ 0.25 s are equal. The optimal trajectories for the bus angle and angular rate for the PLS and VL cases are shown in Fig. 7.6. These figures show that, through only internal actuation of the solar arrays, the slew maneuvers were performed and then the bus was held fixed (i.e., $\theta \equiv 0$ and $\dot{\theta} = 0$) for 1.0 sec all while managing jitter.

Note that for the VL case with slewing times of 0.25 sec and 1.0 sec, the bus angle trajectories are different, but the same slewing angle is achieved. Additional control-design-only problem formulations were conducted with t_m up to 30 s and with the array design fixed as the optimal design from the 4 s slew time PLS solution. It was found that the maximum slewing angle remained equal to 7.2 mrad, indicating that a fundamental limit was reached. The limiting factor preventing larger slew angles here is the actuator voltage limit, as opposed to momentum limitations as detailed in Sec. 7.4.1. Figure 7.7 illustrates that the actuator voltage is saturated during the pointing phase.

The array displacements for the PLS and VL cases are shown in Fig. 7.8. Results from slewing and pointing phases are shown separately. Conservation of angular momentum with negligible material damping in the bus-array system provides a natural explanation of these numerical results. For example, rotating the bus in the clockwise direction requires the array

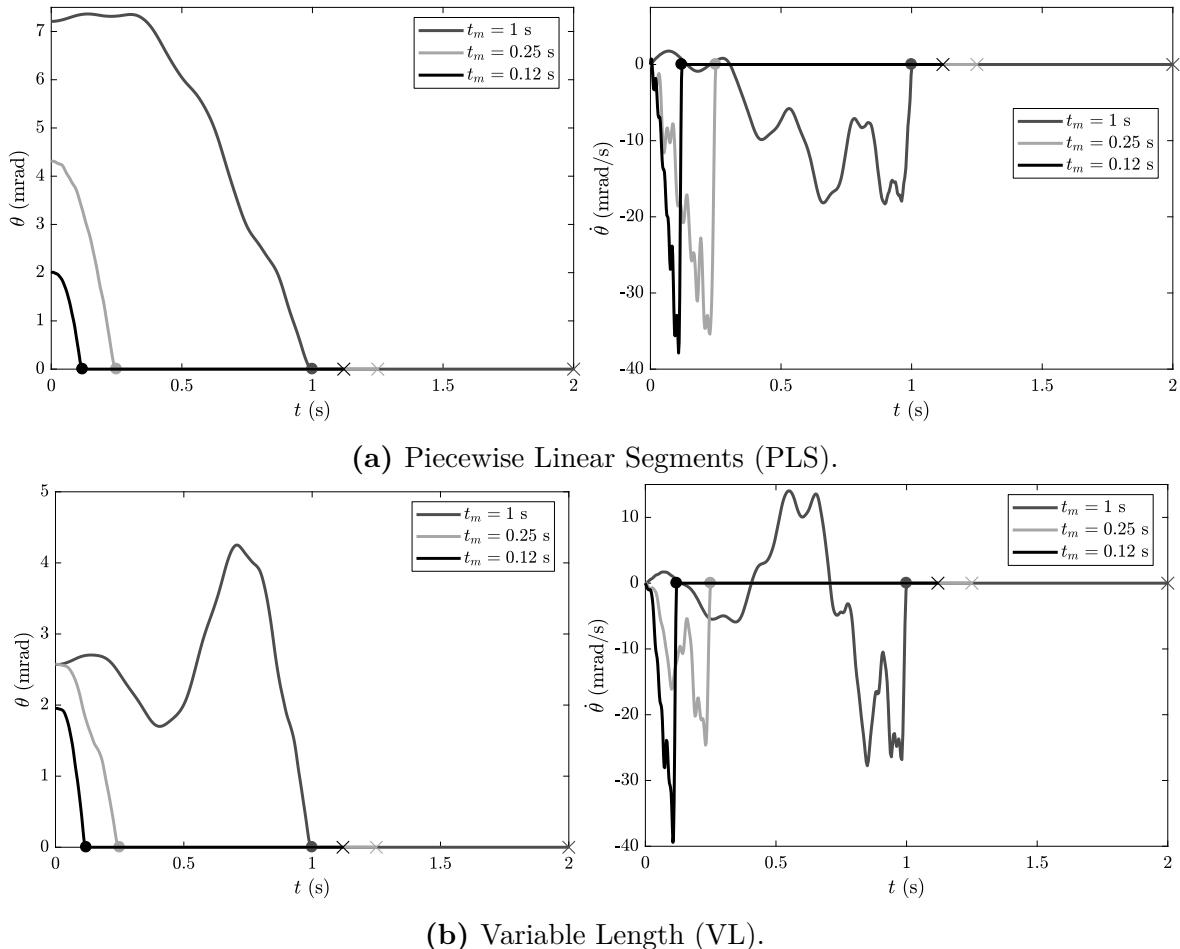
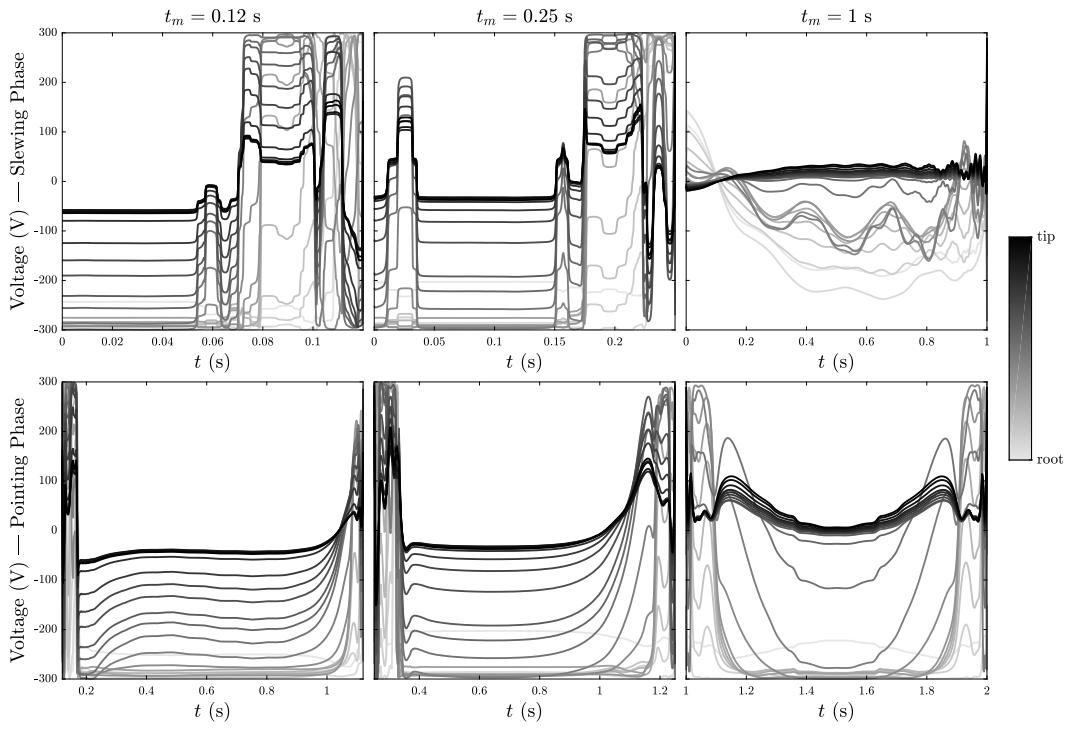
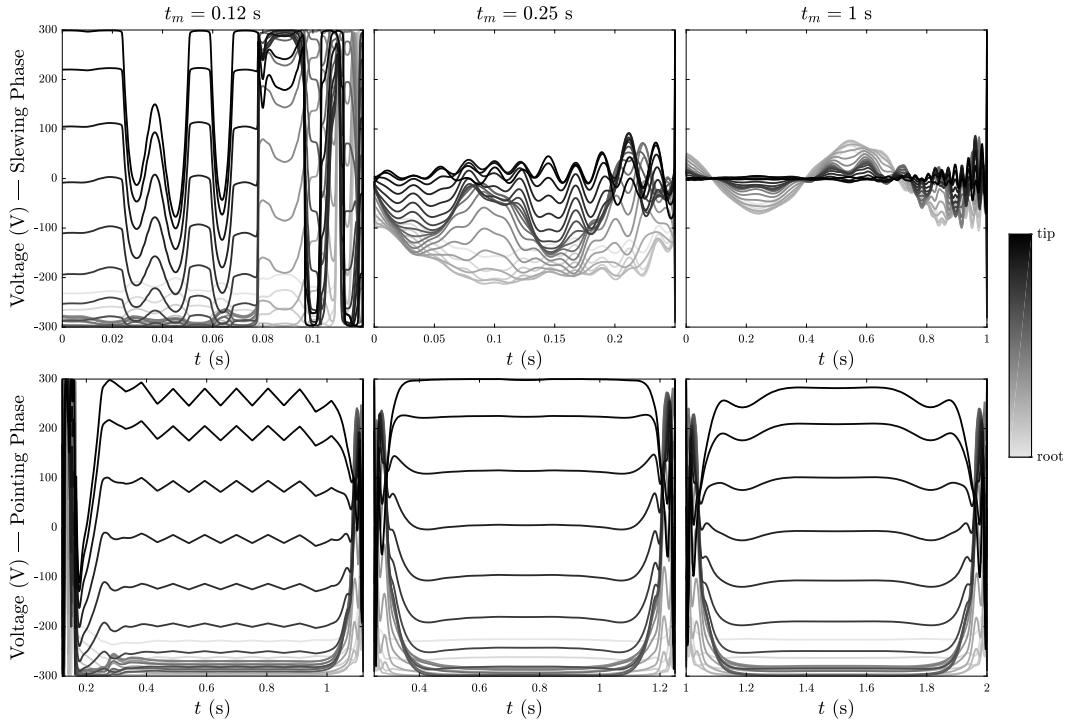


Figure 7.6: Bus angle and angular rate trajectories for select values of \bar{t} .



(a) Piecewise Linear Segments (PLS).



(b) Variable Length (VL).

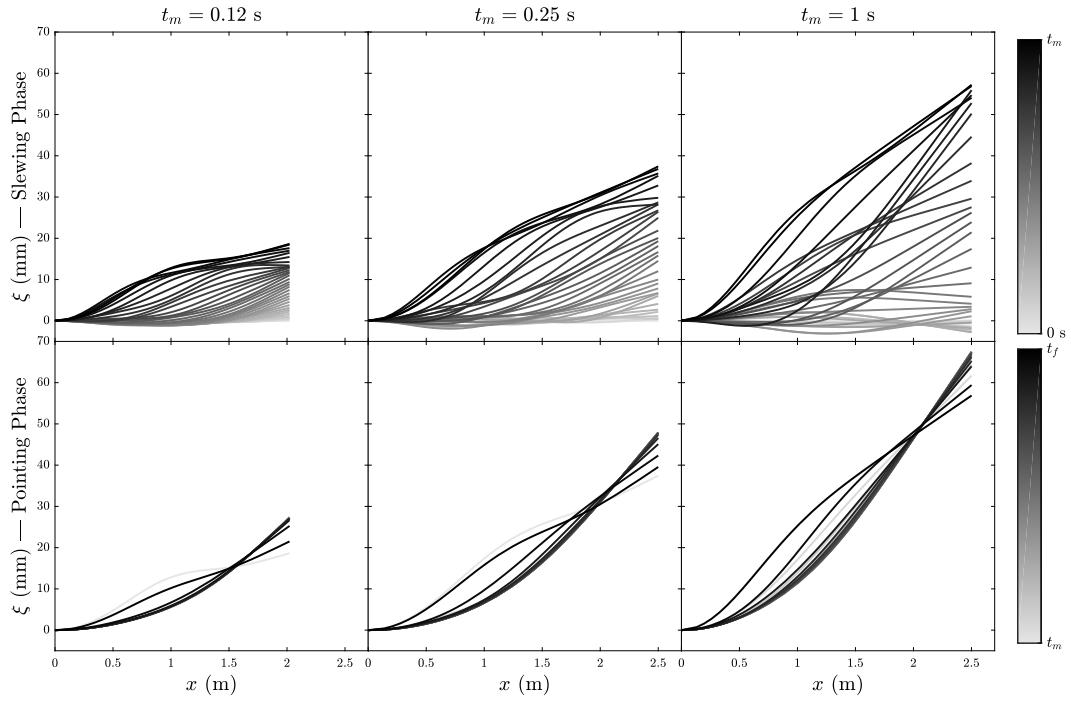
Figure 7.7: Voltage history along the array for select values of \bar{t} .

to displace in the opposite direction (counter-clockwise) (see Fig. 7.1a). This is particularly evident in the VL case with slewing time of 1.0 sec, which shows the effect of CW and CCW array displacements on the bus angle trajectory in Figs. 7.8b and 7.6b.

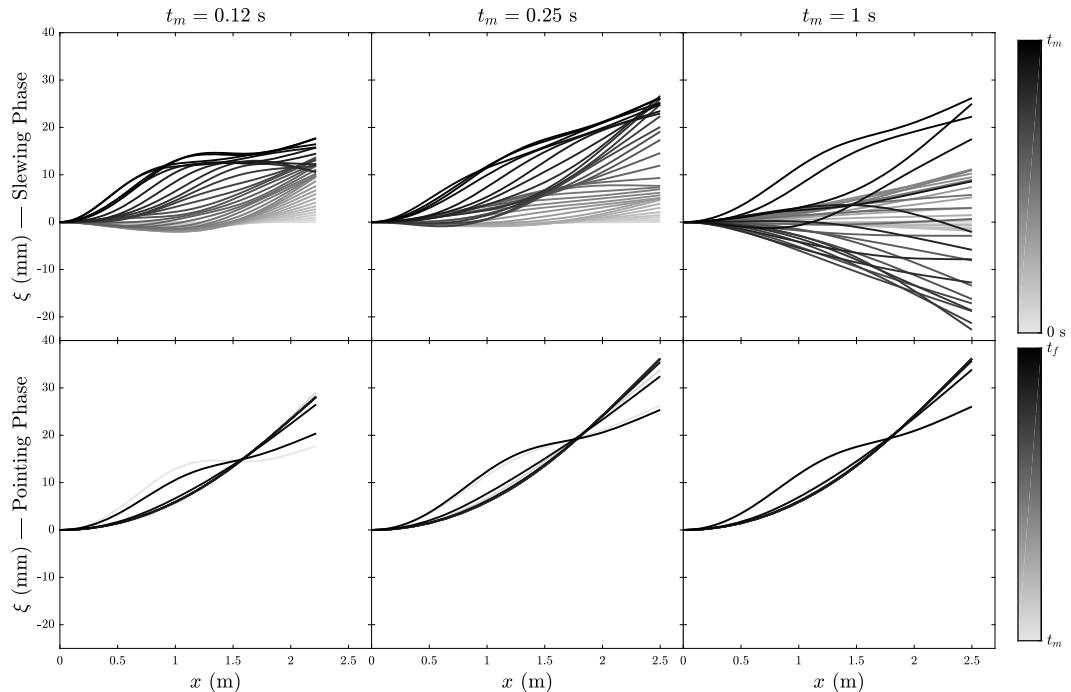
Figure 7.5b shows the array physical design evolution with respect to the slewing time. For longer slewing times (≥ 0.5 s for the PLS case and ≥ 0.25 s for the VL case), the optimal array design maximizes the effective inertia ratio between the bus and the arrays subject to the given constraints. This array shape also corresponds to the maximum slew limits seen in the parametric study confirming the general result shown in Sec. 7.4.1. With this observation, we could define a cost proxy function for maximizing array inertia when we have long slewing times [4]. With shorter slewing times, however, we see more complex designs that do not maximize inertia. Since the analysis using the PRBDM model did not take into account the control system, we need an alternate explanation for these optimal arrays designs. To this end, an integrated analysis is performed that considers the synergy between natural passive dynamics (natural modes of the array, no control) and active (controlled) dynamics.

7.4.3 Optimal Design Tradeoff for Array Structure

The periods of the first natural modes for the array designs of the PLS and VL cases are shown in Fig. 7.9 (the NG case is not considered as it does not involve any modifications of the physical array design). The dashed line is a reference that indicates whether one quarter of the period of the first natural frequency, denoted $T_1/4$, is longer (above) or shorter (below) than the given slewing time. We use the $T_1/4$ line as a reference but the true relationship for this particular co-design formulation appears to be closer to $T_1/4.2$, and likely varies slightly based on problem parameters. Consider now the array design that maximizes the array moment of inertia, subject to the given constraints, for a given problem variation with a particular value for T_1 . If this particular value for $T_1/4$ is shorter than the slewing time, the maximum inertia design will be optimal and the optimal controller will utilize primarily the passive dynamics of the first structural mode to achieve an optimal slewing maneuver (and use higher-order modes partially). If $T_1/4$ is longer than the given slewing time, the maximal inertia array design can only make partial use of the first mode dynamics in the slewing direction during the given slewing horizon. In other words, the array dynamics now need to be faster to work synergistically with the active controller when the slewing time is reduced, and the relationship between t_m and T_1 is approximately linear due to simple scaling of the problem based on the time horizon (see Sec. 4.3.1 for the scaled problem



(a) Piecewise Linear Segments (PLS).



(b) Variable Length (VL).

Figure 7.8: Array deflection profiles for select values of \bar{t} .

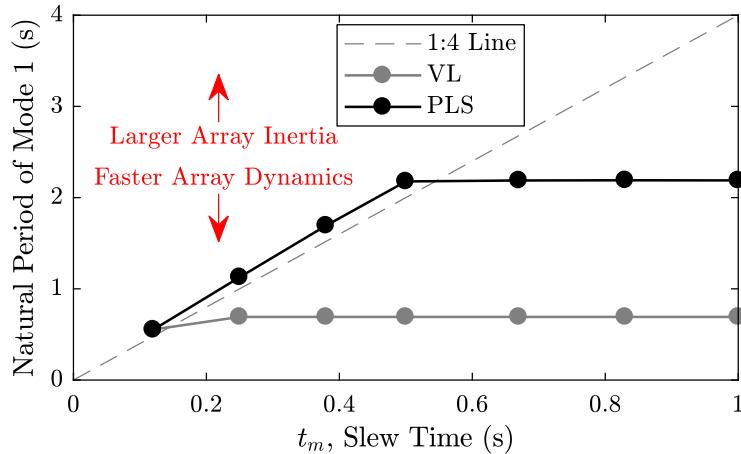
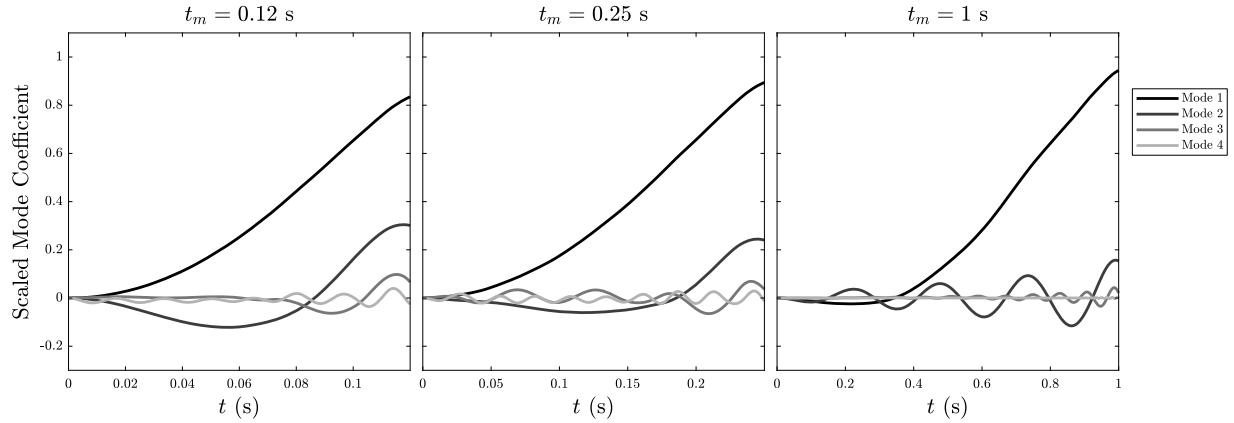


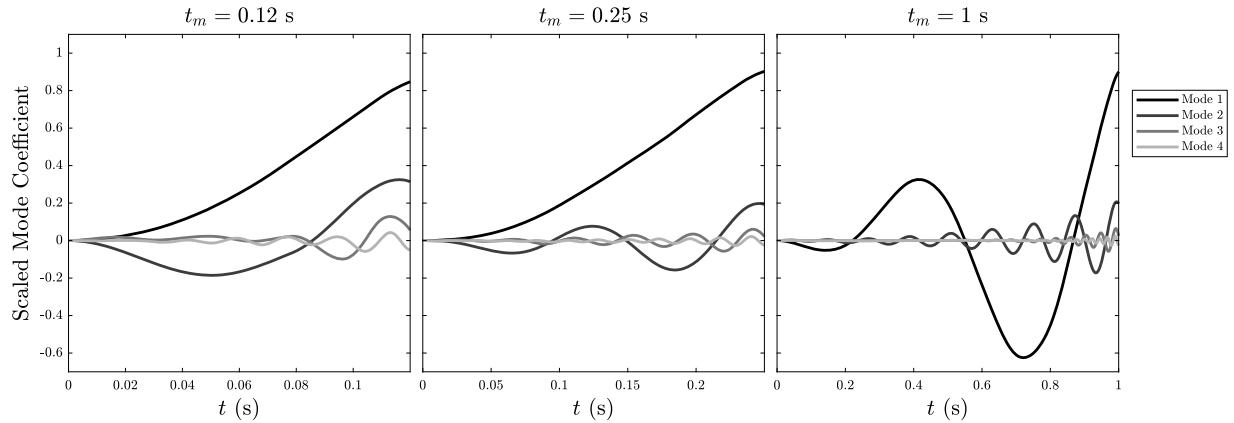
Figure 7.9: Comparison between the first natural period of the optimal array designs and the slewing phase duration.

and a discussion on this finding). Co-design studies are ideal for determining this tradeoff since allowing simultaneous structural and control design freedom provides access to higher performance levels through synergistic structural and control design tailoring without major assumptions, i.e., the parameters for the array structure and open-loop control design are distributed. These results also reveal that the proxy objective function of maximizing inertia is not accurate for faster slew times.

In the cases where the optimal tradeoff is active, the optimal control trajectories include a bang-bang control near the root during the slewing phase, and the optimal array structure changes according to the given design freedom in Table 7.4. For the VL case, the only mechanism available for changing inertia and the first natural frequency is to adjust ℓ , explaining the observed shorter array when $t_m = 0.12$ s (see Fig. 7.5b). For the PLS case, the inertia and the first natural frequency are changed by redistributing the mass and/or reducing the array length to utilize more fully a combination of the array's elastic and inertial properties. Observe that the array mass is not reduced, i.e., the mass constraint in Eqn. (7.15i) is active. In addition to using the first natural mode, results indicate that the optimal solutions with shorter t_m also tend to leverage the use of the second natural mode (refer to the increase in the coefficient of the second mode in Fig. 7.10). Tailoring of the structural design to best use the second mode is evident by the placement of a segment with local minimum thickness near the midpoint of the array length in Fig. 7.5b. The additional structural design freedom provided by the PLS formulation vs. the VL formulation demonstrates the ability of a co-design formulation with more plant design freedom to tailor the passive dynamics of the system to achieve better performance [4].



(a) Piecewise Linear Segments (PLS) design results.



(b) Variable Length (VL) design results.

Figure 7.10: Scaled mode coefficient trajectories for select values of \bar{t} .

7.4.4 Optimal Placement of Segmented Piezoelectric Actuators

A continuously variable, spatially distributed voltage is not a physically realizable actuation strategy, but these optimal trajectories provide insights into performance limits, as well as how a physically realizable strain actuation system should be designed. Continuous voltage variation can be approximated using several piezoelectric segments as shown in Fig. 7.2b, where a constant voltage $V_i(t)$ is applied to each segment. An analysis of optimal voltage trajectories for the PLS and VL cases was performed to provide insight into actuator placement. Figure 7.11 illustrates for each spatial position along the array 1) the maximum voltage amplitude across all time, 2) the mean voltage amplitude during slewing, and 3) the mean voltage amplitude during pointing. The maximum allowable voltage magnitude is 300 V. For small t_m , the actuators are nearly saturated during the slewing phase, and the voltage limit is reached at some point during the maneuver across most of the array. When

the slewing times are longer, the lower average voltage magnitude during the slewing phase is due to the use of modal resonance.

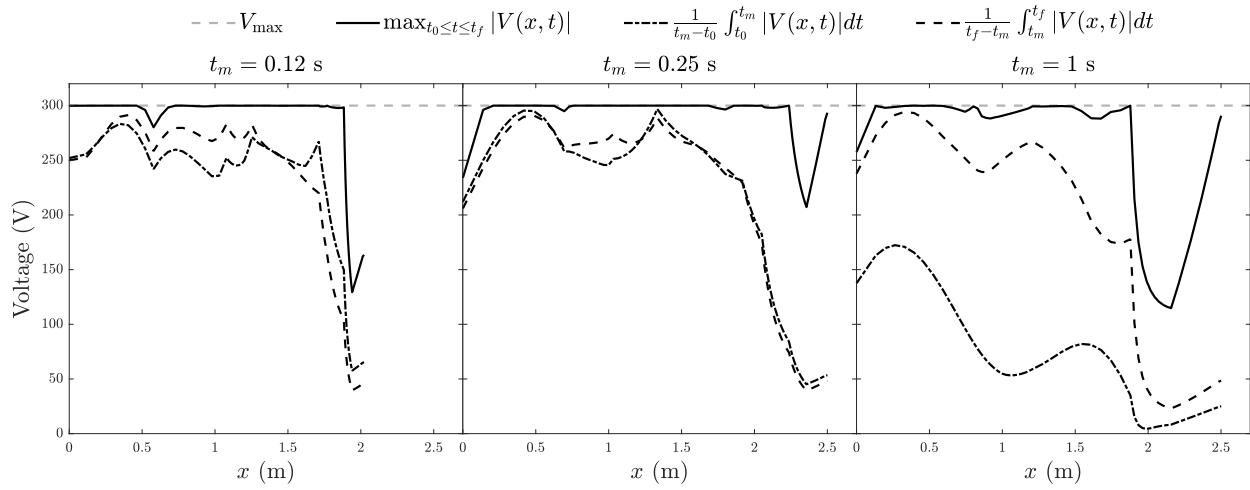
We see that a physical implementation would benefit from placing piezoelectric segments over most of the array area with the exception of the tip. Since each piezoelectric segment can only be actuated with a voltage that is constant in space (not in time), a large number of individual segments translates into more degrees of freedom for the control, which in turn can allow higher performance. However, the voltage metrics suggest that at a minimum two piezoelectric segments should be placed at the two points of local maximum average voltage (which are located near the root and the length midpoint), to take advantage of the natural passive dynamics. These locations are near the critical points of the shapes of the first and second natural modes, which are the dominant modes during the slewing phase as shown in Fig. 7.10. The scaled mode coefficients in this figure indicate the relative contribution of each mode in their linearly combined effect on the array deflection. Future studies can include model actuation using individual piezoelectric segments with constant spatial voltage and limited length to determine their optimal placement location.

7.5 Summary

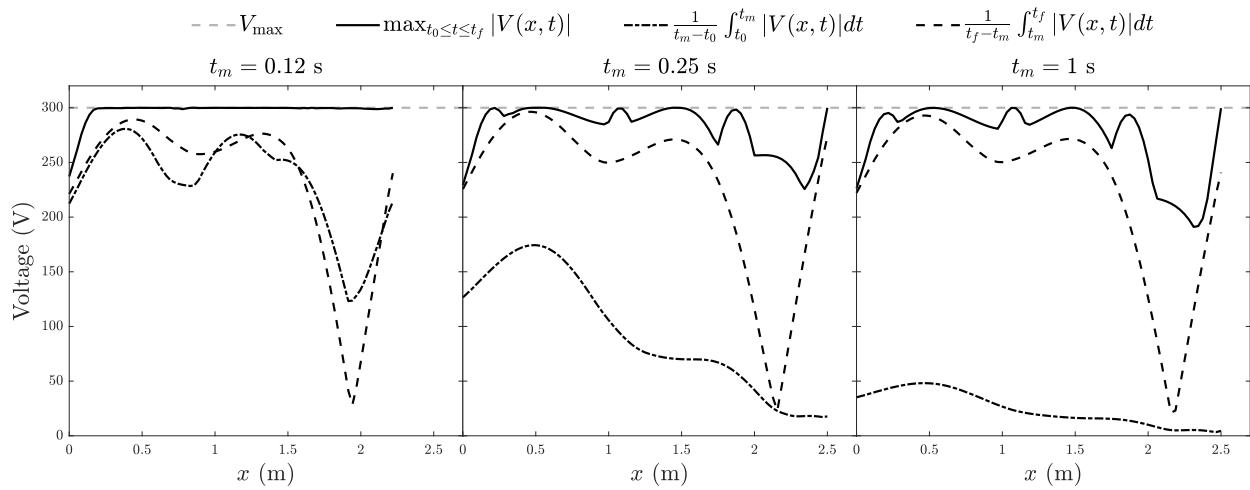
In this chapter, we investigated the integrated structural and control system design of a strain-actuated solar array for spacecraft pointing control and jitter reduction. Slew maneuvers on the order of milli-radians or arc minutes have been achieved in simulations for a representative spacecraft system without increasing the total array mass or reducing the array planform area. A parametric study was conducted with different levels of design freedom and slewing times. Results show that separately designing the control system or the structural system alone cannot achieve the higher performance levels that are possible through the proposed combined design of the structural and control systems. Furthermore, adding degrees of freedom to the structural design—specifically, distributed geometric design—improved performance further by tailoring the passive dynamics of the array with the active controller. This study also indicated the relative effectiveness of the nested co-design strategy over the simultaneous one for certain design problems.

Since the SASA system is based on internal actuation, the angular momentum of the bus-array system must be conserved in the absence of material or joint damping. A desired bus rotation requires array deflection in the opposite direction. Results showed that in addition to accomplishing the required slewing and pointing maneuvers, the optimal array design

is driven by the interaction between active and natural passive dynamics. Conservation analysis indicates that increasing the array moment of inertia helps improve the maximum slewing angle. An array design with maximum mass moment of inertia subject to the given constraints will be optimal if the slewing horizon is larger than a problem-dependent scaling of the first natural mode period of the array (approximately one quarter of this period), and the optimal controller will use modal resonance for an efficient slewing maneuver. For faster slew maneuvers, however, structural dynamics analysis reveals that it is beneficial to choose a tailored design that reduces inertia somewhat, but provides faster passive dynamics that interact with active control to increase the maximum slew angle. The optimal design here occurs when approximately one quarter of the period of the first natural mode is equal to the slewing time. This allows the passive dynamics to contribute to the maximization of the displacement of the first natural mode in the given slewing horizon, which in turn maximizes the slewing angle. In these cases, the resulting control design includes a bang-bang control near the root during the slewing phase. Results also show that the dynamic behavior of the array may be approximated by a PRBDM system with rigid links and joints. This connection helped provide qualitative insights into the design and behavior of intelligent structures with distributed actuation.



(a) Piecewise Linear Segments (PLS) design results.



(b) Variable Length (VL) design results.

Figure 7.11: Voltage trajectory metrics for select values of \bar{t} .

Chapter 8

Case Study: Design of Vehicle Suspensions

“Systems engineering should be, first and foremost, a state of mind and an attitude taken when dealing with complexity.”

J.-L. Wippler [273, p. 208]

8.1 Introduction

The design of vehicle suspensions has been of considerable interest ever since the invention of the automobile. A suspension transfers forces in the system to provide a smooth ride for the passenger and good handling characteristics among other objectives. Fundamentally, it is a type of vibration isolator [274–276]. Different types of suspensions can typically be classified into the three categories of passive [274–279], semi-active [1, 276, 280, 281], or active [27, 33, 275, 276, 282, 283] depending on the external energy flow into the system. There has been considerable research interest in analyzing and optimizing all types of suspensions, particularly utilizing tools from dynamics and controls. However, much of this research has focused on a select few canonical suspensions such as the ones shown in Figs. 8.1a–8.1d. Here we will consider architecture changes in the suspensions, i.e., different components connected in new ways, such as the suspensions shown in Figs. 8.1e–8.1f. In all the suspensions, an unsprung mass U is connected to the road profile z_0 , and there is some mechanical path between the sprung mass S and U .

In addition to architecture design changes, both the physical and control system designs will also be considered. This leads to a complex, challenging design problem that is typically not treated in a systematic fashion in engineering design practice. Here we will describe a particular problem class of combined architecture, plant, and control problems that can leverage a number of previously developed tools to provide solutions to this design problem. Due to the breadth of this design study, many simplifying assumptions must be made to

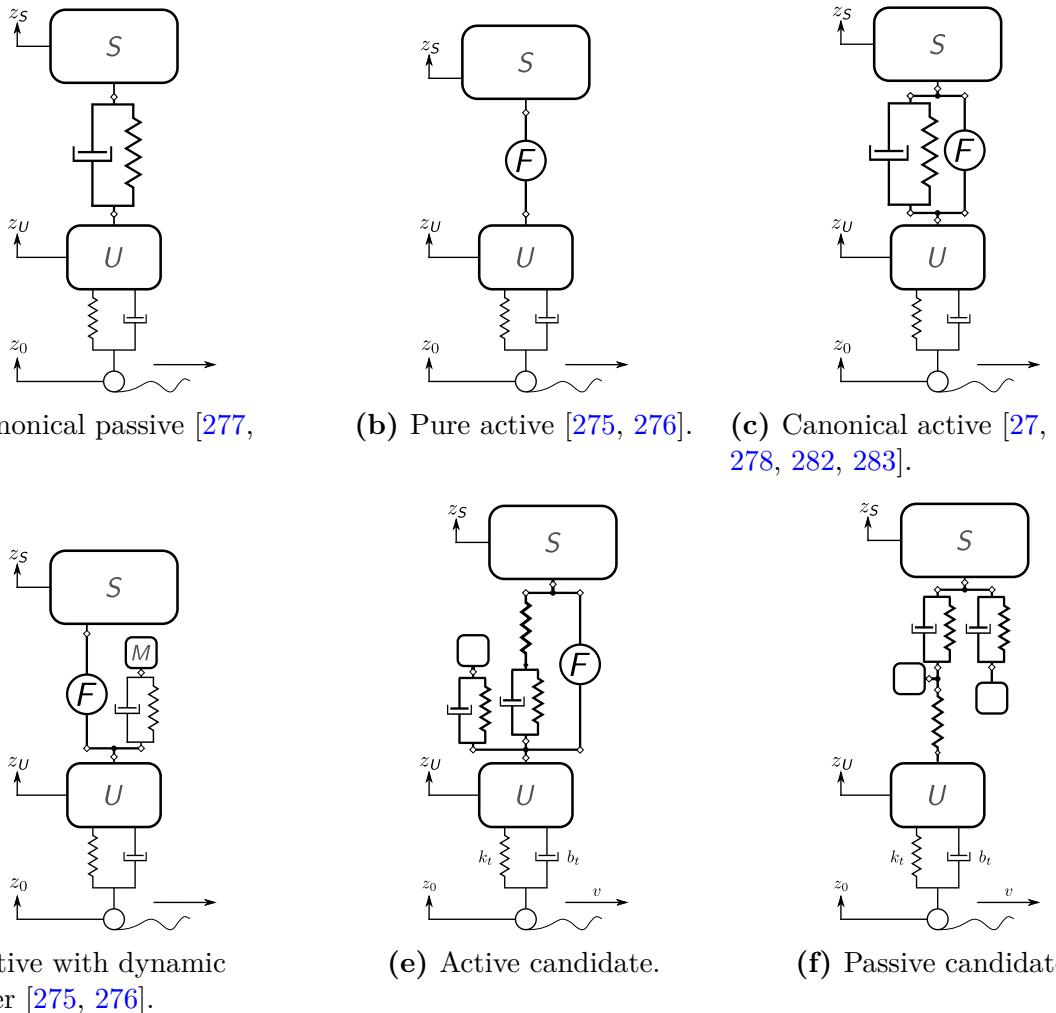


Figure 8.1: Various vehicle suspension architectures.

keep the problem tractable, but still makes it possible to reveal various design insights. It has been observed by experts in the field that even simple suspension models and studies have had a “profound impact on the practical implementations some 15–20 years later [276]”. In addition, suspension design problems have proven to be an interesting area for research in combined plant and control design or co-design [27, 33, 282]. It is important to note that the purpose of the early-stage studies proposed in this chapter is not to supplant the detailed, rigorous, robust previous research, but rather seeks to identify new architectures that could be investigated in the same level of detail that the few canonical architectures have received.

The remainder of this chapter is organized as follows. Section 8.2 outlines the considered problem class with linear physical elements. Section 8.3 provides the combined architecture, plant, and control vehicle suspension design problem formulation. Section 8.4 presents the results of this case study.

8.2 A Problem Class with Linear Physical Elements

8.2.1 Linear Physical Elements

A useful framework for describing linear physical elements is bond graph modeling with power port nodes (or simply power nodes) [38]. Power nodes are characterized by constitutive parameters and follow some constitutive relation (typically a fundamental physical law). They can be classified as source nodes (S_e , S_f), storage nodes (C , I), resistive nodes (R), reversible transducers (TF , GY), and junction nodes (θ , 1). Some analogies for these power nodes in different energy domains are in Table 8.1. An example of a TF (transformer) is a lever, gear, or hydraulic cylinder. The GY (gyrator) typically describes the conversion between energy domains, such as with an electrical DC motor or mass accelerometer. The θ -junction is analogous to Kirchhoff’s current law and the 1 -junction is analogous to the voltage law in electrical systems. For more details on bond graph modeling, see Refs. [38, 284, 285]

The key property for systems represented by bond graphs with linear time-invariant (LTI) elements is that the equations of motion can be represented as a linear descriptor model. If we denote the set of all constitutive parameters for a particular bond graph as \mathbf{p} , the model is of the form:

$$\mathbf{E}(\mathbf{p})\dot{\boldsymbol{\xi}} = \mathbf{A}(\mathbf{p})\boldsymbol{\xi} + \mathbf{B}(\mathbf{p})\mathbf{s} \quad (8.1)$$

Linear Mechanical				
Label	Intuitive	Topology Preserving	Electrical	
S_e	Force	Velocity	Voltage	
S_f	Velocity	Force	Current	
C	$1/K$	M	C	
I	M	$1/K$	I	
R	B	$1/B$	R	

Table 8.1: Some bond graph modeling analogies.

where ξ are the states and s are the sources. The matrix E is invertible if there are no algebraic loops in the system [38, 286]. Here we assume that all algebraic loops are appropriately removed (e.g., see Ref. [286]) so that we have an explicit first-order ordinary differential equation (ODE) with only $\{A, B\}$.

The architecture design decisions will include what power nodes to include in the system and their connections. The constitutive parameters will be the plant design variables (but the plant design could consist of more realistic variables such as the geometry of spring with a mapping back to the appropriate constitutive parameter). The control decisions will come in the form of certain source types. The sources may also be used to add various disturbances to the system.

8.2.2 Problem Class Definition

We would like to solve architecture design problems of the following form:

$$\min_{\mathbf{x}_a} \Psi_a(\mathbf{x}_a) + \Psi_d(f_p(\mathbf{x}_a), f_c(\mathbf{x}_a)) \quad (8.2a)$$

$$\text{subject to: } f_a(\mathbf{x}_a) = a \in \mathcal{F}_a \quad (8.2b)$$

where \mathbf{x}_a represents architecture design variables, $f_a(\mathbf{x}_a)$ is a mapping between the architecture design variables and the architecture a , and \mathcal{F}_a is the feasible set of architectures. Ψ_a is the architecture-only objective function (such as a complexity metric that counts the number of additional components [12]), while Ψ_d is the general design objective function that includes dependence on the plant and control design. This dependence is represented by the mapping functions f_p and f_c between the architecture and the plant \mathbf{x}_p and open-loop control (OLC) \mathbf{u} design variables.

A fair comparison between architecture candidates requires knowledge of the best possible performance for each candidate architecture. To determine the value of Ψ_d we must solve a

suitable co-design problem. This problem has the following form:

$$\min_{\boldsymbol{x}_p^{(i)}, \boldsymbol{u}^{(i)}} \quad \Psi_d = \int_{t_0}^{t_f} \mathcal{L}^{\mathcal{P}}(t, \boldsymbol{y}, \boldsymbol{x}_p^{(i)}) dt + \mathcal{M}^{\mathcal{P}}(\boldsymbol{y}(t_0), \boldsymbol{y}(t_f), \boldsymbol{x}_p^{(i)}) \quad (8.3a)$$

$$\text{subject to: } [\dot{\boldsymbol{\xi}} = \boldsymbol{f}^{\mathcal{P}}(t, \boldsymbol{\xi}, \boldsymbol{u}, \boldsymbol{x}_p)]^{(i)} \quad (8.3b)$$

$$\boldsymbol{h}^{\mathcal{P}}(t, \boldsymbol{y}, \boldsymbol{x}_p^{(i)}) = \mathbf{0} \quad (8.3c)$$

$$\boldsymbol{g}^{\mathcal{P}}(t, \boldsymbol{y}, \boldsymbol{x}_p^{(i)}) \leq \mathbf{0} \quad (8.3d)$$

$$\text{where: } \boldsymbol{y} = \boldsymbol{y}^{\mathcal{P}}(t, \boldsymbol{\xi}^{(i)}, \boldsymbol{u}^{(i)}, \boldsymbol{x}_p^{(i)}) \quad (8.3e)$$

where $\square^{(i)}$ indicates a problem formulation element appropriate for the i th candidate architecture from Prob. (8.2), t is the time continuum between t_0 and t_f , and \boldsymbol{y} are the (architecture-independent) outputs. The problem elements $\{\mathcal{L}, \mathcal{M}, \boldsymbol{f}, \boldsymbol{h}, \boldsymbol{g}\}$ represent the Lagrange term, Mayer term, dynamics, equality constraints, and inequality constraints.

The subscript \mathcal{P} indicates a particular problem class that the problem elements must be in. Here we will consider a class whose specific structure can lead to efficient solution strategies, but still covers the combined architecture, plant, and control design problems of interest. Consider a function, $f^{\mathcal{P}}$, in the problem class \mathcal{P} with type f (e.g., the type might be Lagrange term or inequality constraints). Then $f^{\mathcal{P}}$ must be a function where for fixed values of $\boldsymbol{x}_p^{(i)}$, $f^{\mathcal{P}}$ has an equivalent linear-quadratic dynamic optimization (LQDO) problem element described in Chapter 5. With this specification of \mathcal{P} , Prob. (8.3) is a strong candidate for the nested co-design solution strategy for the reasons discussed in Chapter 3, namely an efficient inner-loop strategy for LQDO. This leads to the following trilevel solution approach.

8.2.3 A Trilevel Solution Approach

The trilevel solution approach used here is illustrated in Fig. 8.2. Each one of the three levels is now described.

8.2.3.1 Architecture Design: Level a

The topmost level is responsible for taking the problem definition, a user-defined component catalog, and network structure constraints and providing candidate architectures for the other levels. Many architectures can be represented by colored graphs as the nodes in this representation scheme can be used to represent a variety of concepts. Therefore, the approach

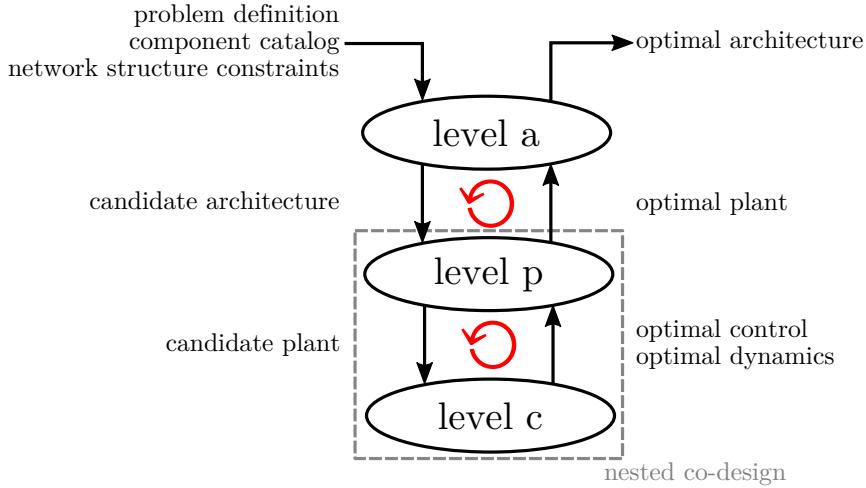


Figure 8.2: The proposed trilevel solution strategy for combined architecture, plant, and control design.

used here will be the perfect matching-based algorithm for generating all architectures in Chapter 2. The different labels in the graphs will correspond to the power nodes in Sec. 8.2.1.

8.2.3.2 Plant Design: Level p

The next level takes the candidate architecture and performs the outer-loop co-design tasks for the plant design [32]. The appropriate optimization problem needs to be automatically created and solved. Since these types of problems can be highly nonconvex (see Chapter 3), global search algorithms are utilized to help improve the confidence of finding the true optimal solution (in this chapter, a multistart approach is used, but an alternative is a genetic algorithm [105, 220, 231]).

An automated model generation procedure (see Sec. 1.4.2) was developed to take the generated graphs and produce the appropriate (linear) model. This procedure utilizes the SIMULINK/SIMSCAPE modeling environment to generate the appropriate diagram. Each time a plant variable is updated, the model is regenerated through a linearization procedure. This is a relatively expensive operation; a better method would generate an analytical representation of $\mathbf{A}(\mathbf{x}_p)$ and the other matrices so that it only needs to be performed once per candidate architecture. Generating these equations is a task for future work.

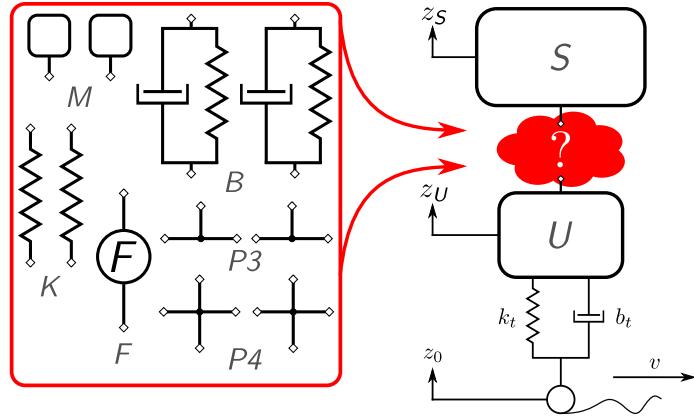


Figure 8.3: Suspension architecture component catalog.

8.2.3.3 Control Design: Level c

The deepest level takes the candidate plant and model to formulate the appropriate LQDO problem in Chapter 5 [131]. The structured-based description makes it relatively straightforward to handle a varying number of states and controls along with the output definitions. Since this inner-loop co-design problem has a special structure, it can be solved with low computational expense and is guaranteed to be the global optimal control [32]. We also note that the number of times level c is solved is much greater than level p, which is much greater than level a.

8.3 Problem Formulation

In this section, the problem formulation for the combined architecture, plant, and control design of a quarter-car vehicle suspension is described.

8.3.1 Architecture Specification

The component catalog and NSCs will be the same as the case study in Sec. 2.4.3. The (C, R, P) specification is:

$$C = \{S, U, M, K, B, F, P3, P4\} \quad (8.4a)$$

$$R = [1, 1, 2, 2, 2, 1, 2, 2] \quad (8.4b)$$

$$P = [1, 1, 1, 2, 2, 2, 3, 4] \quad (8.4c)$$

The catalog is represented in Fig. 8.3. Each of the component types fits in the bond graph modeling paradigm: $\{S, U, M\}$ are I -type storage nodes, K is a C -type storage node, B is a subsystem containing a spring and damper in parallel (see Fig. 8.3), and F is an Se -type effort source. The remaining component types represent 1 -junctions with a differing connection numbers.

The architecture-only objective function term is the sum of the additional physical components (i.e., everything but S , U , and Px):

$$\Psi_a = w_a (n_M + n_K + n_B + n_F) \quad (8.5)$$

where w_a is the weighting coefficient. This just one metric for complexity that we can use to look at tradeoffs between complexity and performance.

8.3.2 Co-Design Problem

Three outputs will be needed to capture the co-design problem formulation:

$$\mathbf{y} = \begin{bmatrix} z_U \\ \ddot{z}_S \\ z_S \\ u \end{bmatrix} \quad (8.6)$$

namely the unsprung mass position, sprung mass acceleration, unsprung mass position, and control. The co-design objective function is the sum of several performance metrics:

$$\Psi_d = \int_{t_0}^{t_f} \left(w_1 (y_1 - z_0)^2 + w_2 y_2^2 + w_3 y_4^2 \right) dt \quad (8.7a)$$

where the term $w_1 (y_1 - z_0)^2$ captures the handling objective, $w_2 y_2^2$ represents the passenger comfort objective, and $w_3 y_4^2$ control effort objective (see Refs. [27, 33, 282]).

Next, the states of the system are initialized to their zero equilibrium position with the following simple bound constraint:

$$\boldsymbol{\xi}^{(i)}(t_0) = \mathbf{0} \quad (8.7b)$$

To ensure that the separation between the sprung and unsprung masses remains tolerable, the following rattlespace constraint is necessary [27, 33, 276, 277, 283]:

$$|y_3 - y_1| \leq r_{\max} \quad (8.7c)$$

Parameter	Value	Parameter	Value
w_1	10^5	k_t	232×10^3 N/m
w_2	0.5	b_t	0 Ns/m
w_3	10^{-5}	r_{\max}	0.03 m
t_0	0 s	t_f	9 s
m_{\min}	0.001 kg	m_{\max}	5 kg
b_{\min}	10^3 Ns/m	b_{\max}	10^6 Ns/m
k_{\min}	10^3 N/m	k_{\max}	10^7 N/m
m_U	65 kg	m_S	325 kg

Table 8.2: Co-design problem parameters.

This constraint can be converted into two linear constraints as is shown in Sec. 5.5.3. The rattlespace constraint is commonly included in the objective function but is more appropriately included as a constraint. The LQDO problem class can readily handle linear inequality constraints unlike other solution strategies [27, 131].

All the previous constraints are necessary for the inner-loop co-design problem. The outer-loop specific plant constraints are simple bounds on the linear coefficients:

$$m_{\min} \leq \mathbf{x}_m^{(i)} \leq m_{\max} \quad (8.7d)$$

$$b_{\min} \leq \mathbf{x}_b^{(i)} \leq b_{\max} \quad (8.7e)$$

$$k_{\min} \leq \mathbf{x}_k^{(i)} \leq k_{\max} \quad (8.7f)$$

where the subscripts $\{m, b, k\}$ indicate the additional mass, damper, and spring plant variables for the candidate architecture.

The problem parameters used in this study are shown in Table 8.2 (many of the parameters are based the study in Ref. [27]). A rough road input is used from Refs. [27, 279].

8.4 Results

In this section, we summarize the results of the vehicle suspension case study. We utilize the code from Ref. [186] to solve the control subproblem. The defect constraints are formed using the trapezoidal (TR) and the chosen quadrature the composite quadratic Hermite-Simpson (CQHS) method (see Chapter 5). It was determined that 2000 time points were needed to approximate sufficiently the problem. The results for the first four architectures in Fig. 8.1 are presented in Table 8.3, along with the other two novel candidate suspensions. A variety of stochastically generated suspensions were evaluated in the graph structure space defined

(a) Maximum control and optimal plant variables.

#	Figure	Ψ_a/w_a	Ψ_d	$w_1(y_1 - z_0)^2$	$w_2y_2^2$	$w_3y_4^2$
1	Fig. 8.1a	2	10.96	6.60	4.36	0.00
2	Fig. 8.1b	1	7.79	3.10	1.51	3.18
3	Fig. 8.1c	3	7.52	3.25	1.99	2.28
4	Fig. 8.1d	4	7.79	3.09	1.51	3.19
5	Fig. 8.1e	7	6.58	2.48	2.43	1.68
6	Fig. 8.1f	7	9.69	5.27	4.42	0.00

(b) Maximum control and optimal plant variables.

#	Figure	$\max u $	k_1	k_2	k_3	b_1	b_2	m_1	m_2
1	Fig. 8.1a	0	1.77E4	—	—	1.88E3	—	—	—
2	Fig. 8.1b	598	—	—	—	—	—	—	—
3	Fig. 8.1c	634	1.47E4	—	—	1.00E3	—	—	—
4	Fig. 8.1d	611	6.89E6	—	—	6.04E5	—	1.02E-3	—
5	Fig. 8.1e	478	9.55E4	6.67E3	8.83E4	1.46E4	2.01E3	3.11E-3	—
6	Fig. 8.1f	0	7.28E4	8.54E3	2.51E5	1.00E3	2.27E3	3.21E0	1.10E0

Table 8.3: Summary of the suspension design results.

by Prob. (8.4), and the two reported novel architectures were the among the best performing for an active or passive suspension system.

As expected, the worst-performing suspension of the six in Fig. 8.1 was the canonical passive design (some of the optimal position trajectories and the rattlespace are shown in Fig. 8.4a). Here we see fairly large fluctuations in z_S , and the rattlespace constraint is satisfied. The alternative passive suspension in Fig. 8.1f (results in Fig. 8.4b) achieved a 12% reduction in the objective function. The primary improvement was in the handling objective. However, this architecture is more complex with seven additional components compared the original two.

The results for the pure active suspension are shown in Fig. 8.5a. Compared to the passive suspensions, the performance index is significantly lower, demonstrating the potential value of an active component. Since there are no passive components naturally keeping the sprung mass near the equilibrium position in this architecture, we see a drift in the sprung mass position (but the rattlespace constraint is still satisfied). The canonical active suspension in Fig. 8.1c does not have this issue. From the results in Fig. 8.5b, we see a very different profile for z_S . Since this architecture has some plant design flexibility, we expected an improvement in performance over the pure active design. A 3.5% decrease in the objective function value is observed indicating that the addition of the two passive components does result in a minor

improvement in performance. The distribution of the individual objective function terms in Eqn. (8.7a) is quite different between the two suspensions.

The addition of the dynamic absorber to the pure active suspension in Fig. 8.1d is supposed to improve handling without compromising the comfort objective [276]. However, the results in Fig. 8.6a indicate no performance benefit for this architecture change with respect to the specific problem parameters used in this study. This is most readily observed with the value of the additional mass near the lower bound of 0.001 kg (effectively removing it from the system). It is also the only architecture where the rattlespace constraint is active at some point during the time horizon. The final architecture had the best overall performance with a 13% reduction compared to the canonical active suspension. The results are shown in Fig. 8.4b, and this design had the smallest maximum control effort. Once again, the dynamic absorber subcomponent (now attached to the sprung mass) is ineffective with a mass value near the lower bound. This design did need seven additional components to achieve this performance improvement.

8.5 Summary

The results in this case study demonstrate that changes to the vehicle suspension architecture can result in improved performance. The purpose of these early-stage studies is to identify new architectures that could be investigated in the same level of detail that the few canonical architectures have received. This case study utilized a newly developed paradigm for combined architecture, plant, and control design that can be applied to systems with linear physical elements.

It remains future work to evaluate the entire set of possible 13,727 unique suspension architectures from Prob. (8.4) [6]. In addition, there are a number of improvements that can be made to the problem formulation. Multiple road inputs should be considered simultaneously to give a better representation of all the environments the suspension will need to function in. Frequency domain properties, such as suspension quality spectral density and control energy spectral density, could also be utilized for a more effective problem formulation [33].

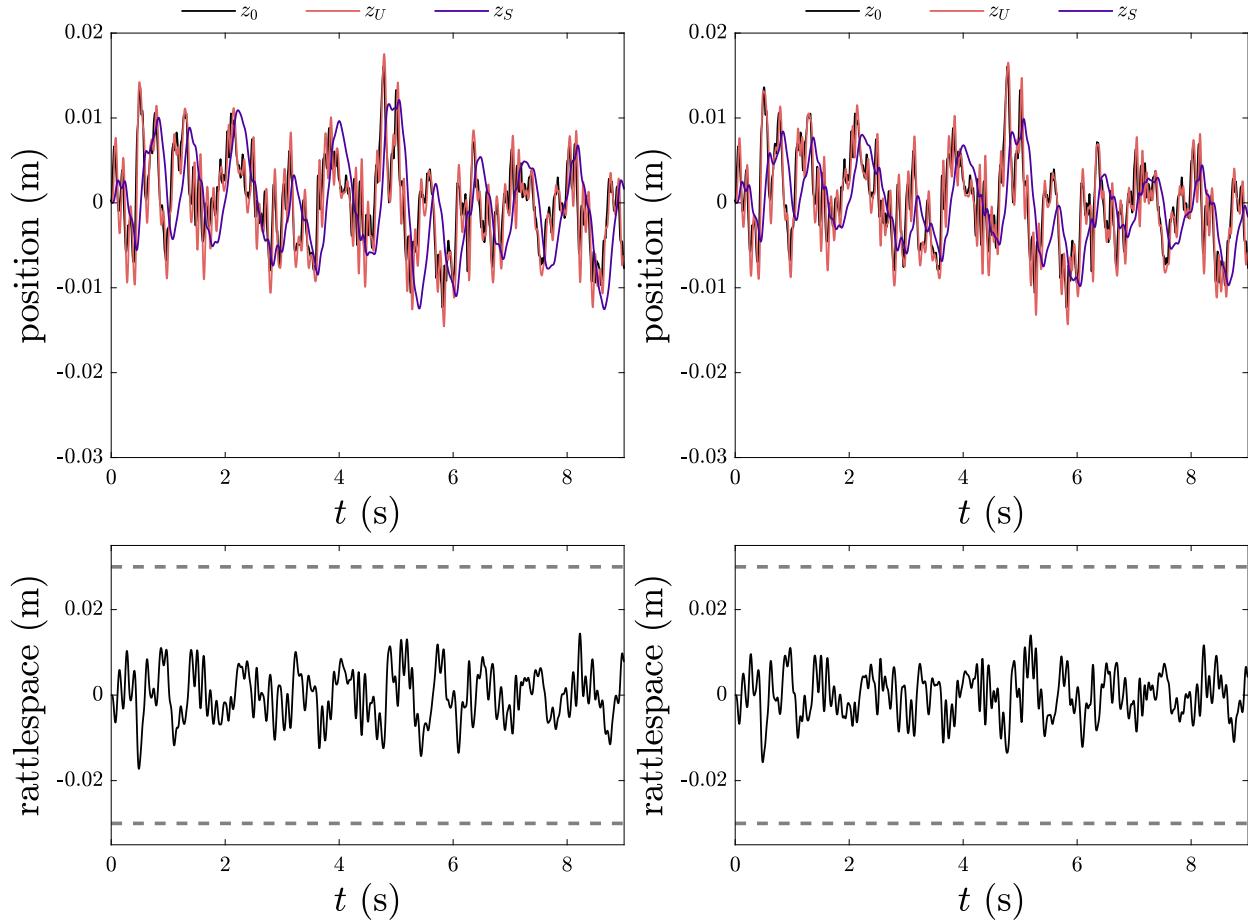


Figure 8.4: Optimal trajectories for the two passive suspensions.

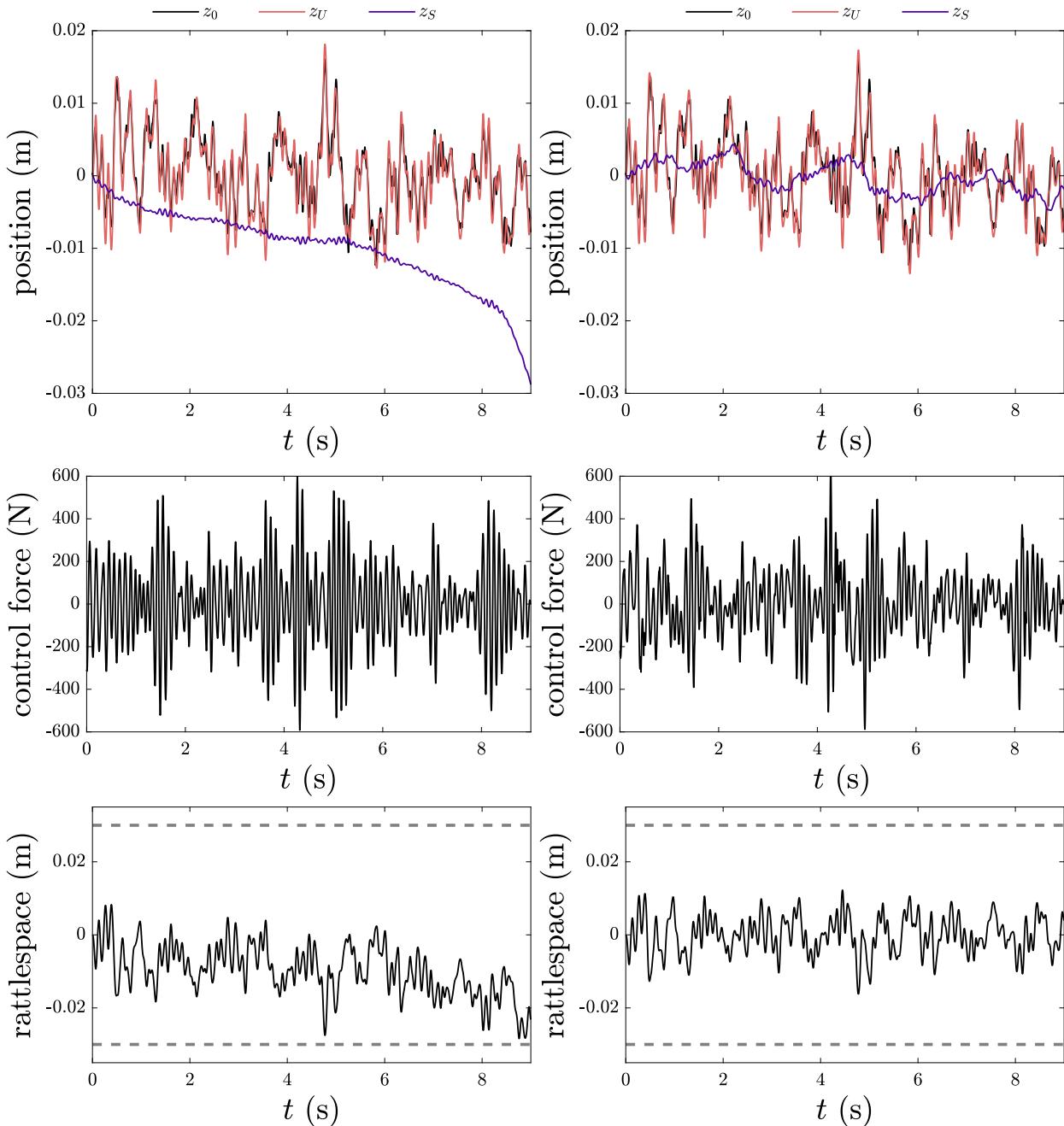
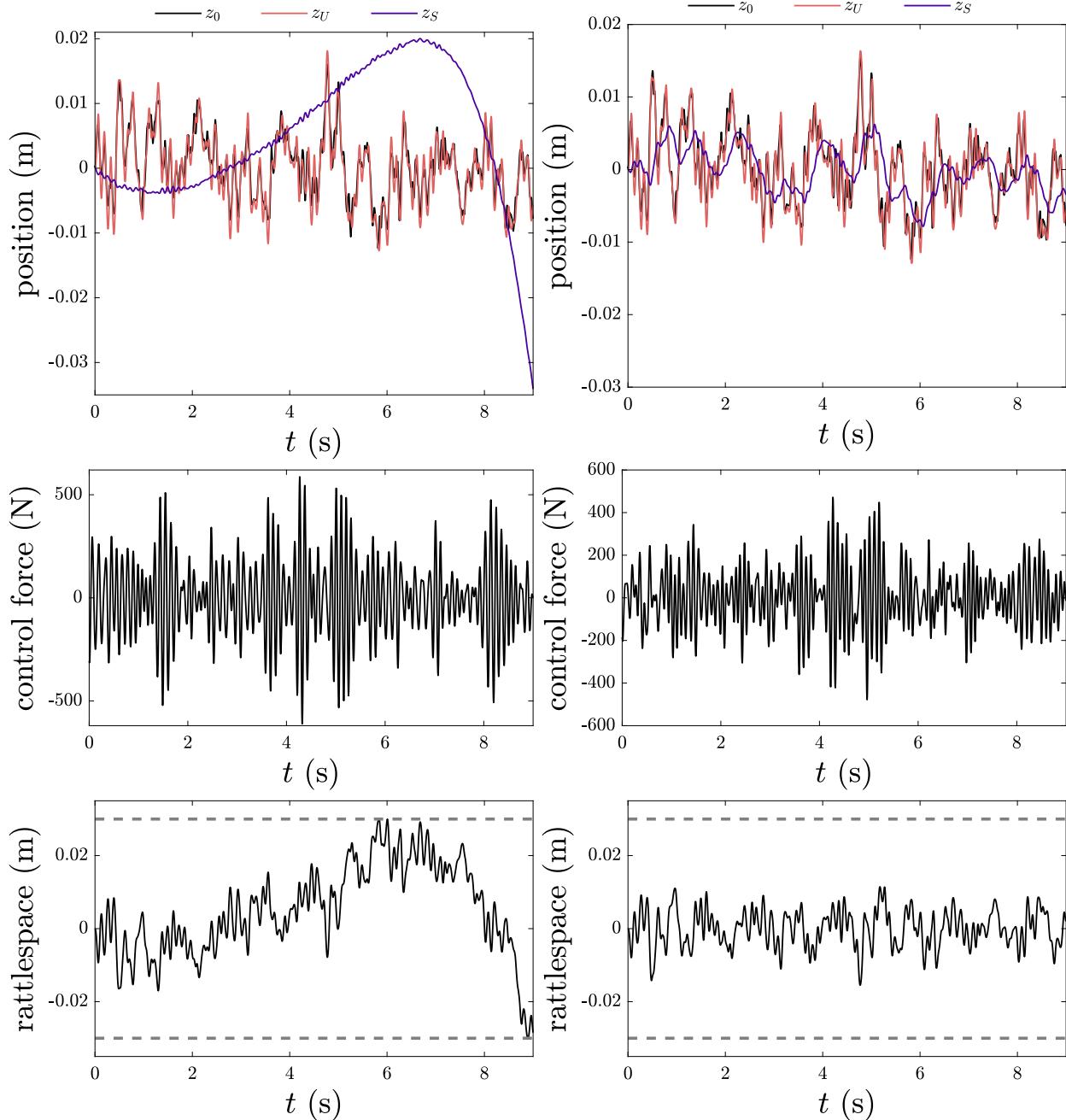


Figure 8.5: Optimal trajectories for two suspensions.



(a) Active with dynamic absorber in Fig. 8.1d.

(b) Active candidate in Fig. 8.1e.

Figure 8.6: Optimal trajectories for two suspensions including the current best architecture.

Chapter 9

Conclusions and Future Work

“Design, on the other hand, is concerned with how things ought to be...”

H. A. Simon [287, p. 114]

9.1 Summary

The design of actively-controlled, dynamic engineering systems is a grand task. While there are a number of approaches that can be used to address these design problems, a formal systematic design automation approach can lead to innovations in many different areas. In this dissertation, three design domain classifications were explored: architecture, plant, and control. The necessary theory and tools were developed to handle various aspects of this integrated design paradigm and a number of case studies were provided to illustrate the proposed design process.

Chapter 2 focused on the task of representing and generating (all) candidate architectures for a particular architecture problem class define by colored graphs built from a catalog of components. The growth rate of the complete listing was shown to be bounded by the double factorial function on the number of ports, but the practical examples with suitable NSCs demonstrated the number of unique, feasible architectures is frequently manageable.

The next chapter focused the general co-design problem, or combined plant and control design. The dynamic optimization problem formulation and optimality conditions for both the simultaneous and nested solution strategies were presented. The test problems in this chapter highlighted a number of key concepts including coupling, the difference between the feasible regions for each strategy, general boundary conditions, inequality path constraints, system-level objectives, the complexity of the closed-form solutions, and nonconvexity. Due to a number of challenges associated with the optimality conditions, practical solution considerations were discussed with a focus on the motivating reasons for using DT in co-design.

An investigation was done with scaling in dynamic optimization problems to help manage the complexity and develop design insights. The mechanics of scaling are fairly straightforward, but proper utilization of scaling relies heavily on the creativity and intuition of the designer. In the simple SASA problem, scaling was used to understand observed results from more complete, higher-fidelity design study in Chapter 7.

A bulk of the next chapter focused on solving a particular subclass of DO, namely LQDO problems. These problems can be approximated with quadratic programs and be constructed and solved efficiently. A class of numerical methods known as DT was utilized to find approximate solutions to the LQDO problem. Including a variety of DT methods allowed for direct comparisons between the methods. The PS-based methods had extremely fast convergence in problems with no path constraints and that were generally smooth. When nonsmoothness was present in the optimal solutions, the higher-order SS methods performed better, with the HS and RK4 being the best. The CQHS-based methods generally performed as well as or better than the other SS methods, demonstrating the relative effectiveness of the new quadrature scheme.

The first detailed case study undertook the design of passive analog circuits through the enumeration of all relevant circuits generated using the methods presented in Chapter 2. Both presented examples (frequency response matching and low-pass filter realizability) demonstrated that enumeration is feasible for certain commonly-used synthesis problems, but is also a challenge to use for sufficiently demanding synthesis tasks. The results were compared to existing approaches. Enumeration showed that some evolutionary approaches have produced minimum complexity or Pareto-optimal topologies. In addition, the results provided initial insights into the computational expense required to solve architecture design problems with enumeration.

The second case study tackled a sophisticated co-design problem with the design of SASAs for spacecraft precision pointing and jitter reduction. Single-axis slew maneuvers of 7.2 milliradians were achieved for a representative spacecraft model without increasing array mass or reducing array planform area. From additional tradeoff studies, a design criteria was revealed for the array structure and control strategy based on the optimal design tradeoff between large array inertia and fast structural dynamics. This study also indicated the relative effectiveness of the nested co-design strategy over the simultaneous one for certain design problems.

The final case study considered the design of vehicle suspensions with design decisions in all three domains. It was shown that changes in the suspension architecture can result in improvements to the suspension quality when compared to the few canonical representations.

A problem class with combined architecture, plant, and control design using linear physical elements was presented. This class can be solved using the methods in this dissertation, but special attention is still needed to keep the problems manageable. The sentiment remains true for any design problem that attempts to determine the optimal architecture, plant, and control.

9.2 Contributions

1. A method for enumerating all architectures represented by colored graphs under certain assumptions was developed. This perfect matching-based approach also included a number of enhancements which cover the same set of graphs more efficiently. It was shown that reasonably large problems can be enumerated when NSCs and isomorphism checking are included when compared to other approaches such as adjacency matrix enumeration. The examples and case studies demonstrated that this approach can provide architectures that are useful in various engineering design studies.
2. Previous work in co-design theory imposed restrictions on the type of problems that could be posed. The work in this dissertation lifted many of those restrictions. The problem formulations and optimality conditions for both the simultaneous and nested solution strategies are given, along with a general discussion on the practical solution strategies. Three test problems were developed to illustrate the differences between the two co-design strategies.
3. A unified approach to the scaling of dynamic optimization formulations was developed with a particular focus on how to leverage scaling in design studies. The necessary theory for scaling dynamic optimization formulations was presented, and a number of motivating examples were shown.
4. A unified framework for solving linear-quadratic dynamic optimization problems was developed. The considered problem class is very general, covering many previously studied linear-quadratic problems. An automated problem generation procedure is developed that generates the matrices for the quadratic program given a natural structure-based description. A new composite quadratic Hermite-Simpson method that uses linear interpolation between node points for each term in the quadratic objective function was developed as one available quadrature scheme.

5. Several engineering design examples were presented. Each one of the case studies handles a complex, relevant design problem with some combination of architecture, plant, and control design decisions. The code for most of the content in this dissertation is made available in an effort to make these contributions and examples available for replication and as a foundation for future work (see Appendix D).

9.3 Future Work

- **Design Process for Complete Dynamic System Design**—Many of the contributions in this dissertation support various aspects of the complete dynamic system design process described in Sec. 1.3. However, there are still many gaps that limit the influence of the proposed design process. The two case studies that included control are stage 1 studies. To create a realizable system, the control architecture (both continuous and digital) need to be developed in a satisfactory way, as well as using sufficiently accurate models and robust problem formulations. Additional theory and tools are needed to address the remaining design tasks such as bridging the gap between open-loop and closed-loop control [1]. Furthermore, compelling examples are needed to provide testimony for the benefits of the prosed approach.
- **Effective Utilization of Enumerative Methods in Architecture Design**—The perfect matching-based enumerative scheme in Chapter 2 allowed for the generation of all architectures under certain assumptions. However, any enumerative method (with some open-ended nature) will reach a point where too many architectures need to be generated and tested, as was shown in the simple examples and the case studies. Therefore, determining when enumeration is appropriate is an important task in any architecture design problem. Developing new NSCs that capture real feasibility requirements can help push this limit. An alternative is to utilize enumeration in novel ways with strategies that better scale with larger architecture problem sizes, such as evolutionary computation or machine learning. Recent work has employed machine learning strategies to scale to synthesis problems larger than the training data set generated using an enumeration [235] of spatially defined, uncolored graphs. Extending the use of data from enumeration to larger synthesis problems described by general colored graphs is an important topic for future work.
- **Choosing between Nested and Simultaneous Co-Design Solution Strate-**

gies—The discussion on the two co-design solution strategies in Chapter 3 was lacking complete information on how to choose between either strategy (or another strategy appropriate for co-design problems). Additional work is needed to provide clear guidelines with supporting evidence. The development of appropriate test problems could aid in this endeavor. General reductions in the computational expense of either strategy will also be important so that fair comparisons can be made.

- **Future Work in Linear-Quadratic Dynamic Optimization**—In Sec. 5.7, a number of future work items for the automated problem generation procedure for solving linear-quadratic dynamic optimization problems were outlined. This included multiple-interval PS methods, multiphase problems, mesh refinement, costate approximation, additional defect and quadrature methods, customized QP solvers, scaling, and quadratically-constrained QPs programs. Implementing these can improve the LQDO by providing additional problem types and better quality solutions.
- **Combined Architecture, Plant, and Control Design with Linear Physical Elements**—The problem class presented in Chapter 8 is lacking theory and tools in some areas. An efficient and automated method for generating the state-space equations as an analytical function of the plant design variables should greatly reduce the computational expense over linearization of a block-diagram model. This may be possible with advanced bond graph modeling tools [288, 289], or the conversion of the system into an equivalent electrical circuit [290]. Understanding the different problem elements that can be effectively handled would be useful such as frequency domain constraints or certain types of plant constraints. Techniques to filter out poor performing topologies before the full inner-loop co-design problem is solved may be possible.
- **Further Development of the Case Studies**—The three case studies in this dissertation were constructed to provide the initial design solutions and insights needed to handle the complex nature of their design problems. However, additional work is needed so these applications do not remain only in the early stages of development. Although seemingly closest to realizability, the design of passive analog circuits should include layout design and restriction of using the preferred component values. The design of SASAs has different elements that can be improved. A continuously distributed internal moment is hard to realize on actual hardware, so piecewise constant actuation is preferred. Determining design guidelines for this control architecture, such as actuator placement and a suitable (closed-loop) control system that can provide the

performance and robustness required for space missions, is essential. The final vehicle suspension case study used simplified plant models and a single problem formulation with one road profile. Furthermore, open-loop control was utilized, so a closed-loop controller should be developed. The usefulness of the optimal designs at this stage is limited until more suitable design problems are solved. Insights from the simpler studies can provide an initial basis for the comprehensive studies.

Appendix A

Enhancements to the Perfect Matching-based Tree Algorithm for Generating Architectures²⁶

A.1 Overview

In Chapter 2, a tree search algorithm was developed to generate a set of colored graphs covering the graph structure space defined by (C, R, P) and various additional network structure constraints. This algorithm is shown in Alg. A.1 with some minor changes to the variable names and the feasible edge improvement directly included. In this appendix, a number of enhancements are proposed to either more efficiently cover the same graph structure space or allow additional network structure constraints to be defined.

Each one of the enhancements will be discussed in the following format. First, the theory behind the enhancement will be presented with a focus on showing the desired graph structure space is still covered. Second, the implementation of the enhancement will be discussed with pseudocode. Finally, some examples are provided comparing the original algorithm to the enhancement. Both visualizations or other aides and computational tests are provided²⁷. *En* is an abbreviation for enhancements included, *Orig* is an abbreviation for original algorithm (see Alg. A.1), and *12T* is an abbreviation for 12 threads.

There are six enhancements proposed in this report: replicate ordering, avoiding loops, avoiding multi-edges, avoiding line-connectivity constraints, checking for saturated subgraphs, and enumerating subcatalogs. The final section discusses the effect of the enhancements on the case studies in Chapter 2.

²⁶Elements of this chapter are based on work completed in Ref. [41].

²⁷A tests were performed on a personal computer with an i7-6800K at 3.8 GHz (up to 12 threads available), 32 GB DDR4 3200 MHz RAM, WINDOWS 10 64-bit, and MATLAB 2017a.

Algorithm A.1: Original tree search algorithm.

Input : V – vector of remaining ports for each component replicate
 E – vector of edges in sequential pairs, initially empty

A – expanded potential adjacency matrix
 cVf – cumulative sum of the original V plus 1
 SavedGraphs – set of graphs, initially empty

Output: SavedGraphs – set of graphs

```

1  $iL \leftarrow \text{find}(V, \text{first})$                                 /* find first nonzero entry */
2  $L \leftarrow cVf(iL) - V(iL)$                                      /* left port */
3  $V(iL) \leftarrow V(iL) - 1$                                          /* remove port */
4  $V_{\text{allow}} \leftarrow V \circ A(iL,:)$                                /* zero infeasible edges */
5  $I \leftarrow \text{find}(V_{\text{allow}})$                                      /* find nonzero entries */
6 for  $iR \leftarrow I$  do                                              /* loop through all nonzero entries */
7    $R \leftarrow cVf(iR) - V(iR)$                                          /* right port */
8    $E2 \leftarrow [E, L, R]$                                             /* combine left, right ports for an edge */
9    $V2 \leftarrow V$                                                        /* local remaining ports vector */
10   $V2(iR) \leftarrow V2(iR) - 1$                                          /* remove port (local copy) */
11   $A2 \leftarrow A$                                                        /* local expanded potential adjacency matrix */
12  if any element of  $V2$  is nonzero then                                /* recursive call if any remaining vertices */
13    |  $\text{SavedGraphs} \leftarrow \text{Algorithm A.1 (tree)} \text{ with } V2, E2, A2, cVf, \text{SavedGraphs}$ 
14  else
15    |  $\text{SavedGraphs}\{\text{end} + 1\} \leftarrow E2$                            /* save missorted perfect matching */
16  end
17 end

```

A.2 Replicate Ordering

This enhancement is based on replicate ordering and is similar to the port-ordering modification that the original tree algorithm is based on [6]. Now, we eliminate some component-type isomorphisms during the graph generation process.

A.2.1 Theory

Consider a single component type and its set of N replicates. Now, consider a single replicate numbered n and $n \neq 1$. During an iteration of the tree algorithm (Alg. A.1), a single edge is added. If replicate $n - 1$ still has no ports connected, then adding this edge to n will produce a graph isomorphic to the graph created when adding the edge to $n - 1$. This claim is based on the component-type isomorphism issue where we have two identical replicates currently with no edges. Therefore, we only need to allow a connection to $n - 1$ and not to n , and the tree algorithm will continue generating the desired graph structure space. When $n = 1$, an edge will always be allowed since there is no other replicate to compare against.

This enhancement is general since it does not rely on any network structure constraints (NSCs) being present.

A.2.2 Implementation

Algorithm A.2: Limit potential connections based on replicate ordering.

Input : V – vector of remaining ports for each component replicate

Vf – initial vector of ports available for each component replicate

$iInitRep$ – indices of the initial replicate for each component type

Output: $Vordering$ – binary vector where 1 indicates an edge is possible, 0 if it is not

```

1  $Vordering \leftarrow circshift(V, [1]) \neq Vf$            /* check if left neighbor has a connection */
2  $Vordering(iInitRep) \leftarrow 1$                    /* initial replicates are always 1 */

```

Algorithm A.2 is the pseudocode for the implementation of this enhancement. The implementation is centered around the creation of a binary vector ($Vordering$) with length equal to the total number of components where unity indicates a connection is allowed and zero indicates it is not. This enhancement is implemented efficiently with the $circshift(v, k)$ function. This function circularly shifts the elements in array v by k positions [291]. Based on the discussion above, we want to determine if $n - 1$ has been connected to any other vertex. Using the initial vector of ports available for each component replicate (Vf), we $circshift$ the vector of remaining ports for each component replicate (V) by one position to the right and compare these vectors (see line 1). A pair that is not equal indicates $n - 1$ has at least one edge, so we now need to allow connections to n .

The procedure above will produce incorrect results for the initial replicates. However, we have that these initial replicates should never have their connection disallowed so we simply ensure that their index in $Vordering$ is always unity (see line 2). The indices of the initial replicate for each component type ($iInitRep$) is calculated before the tree algorithm is called since it does not change throughout the graph generation procedure.

This enhancement is inserted between lines 3 and 4 of Alg. A.1 and line 4 is changed to:

$$Vallow = V \circ A(iL,:) \circ Vordering \quad (\text{A.1})$$

Now, $Vordering$ can disallow connections in the same way as $A(iL,:)$.

A.2.3 Examples

A.2.3.1 Example 1

The base three-tuple and NSCs for this example are specified as:

$$C = \{X\}, \quad R = [4], \quad P = [2], \quad \text{no additional NSCs} \quad (\text{A.2})$$

The second and third inputs for Algorithm A.2 are:

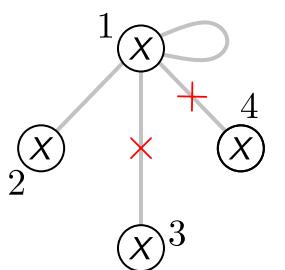
$$Vf = [2 2 2 2], \quad iInitRep = [1] \quad (\text{A.3})$$

We will consider two different V , one at the initial iteration of the tree algorithm and one at some intermediate iteration. Figure A.1a goes through the operations in Algorithm A.2 for the different V . The example iterations are also visualized in Figs. A.1b and A.1c.

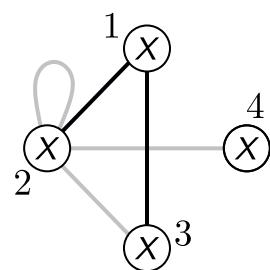
Table A.1 compares the original algorithm with the enhancement for this example. There is a reduction in candidate graphs generated while the number of unique graphs remains the same.

	Initial	Intermediate
V	[1 2 2 2]	[0 0 1 2]
$\text{circshift}(V[0 1])$	[2 1 2 2]	[2 0 0 1]
Vf	[2 2 2 2]	[2 2 2 2]
$\text{circshift}(V[0 1]) \neq Vf$	[0 1 0 0]	[0 1 1 1]
$V\text{ordering}(iInitRep) \leftarrow 1$	[1 1 0 0]	[1 1 1 1]
Removed/total branches	2/4	0/2

(a) Algorithm operations.



(b) Initial iteration.



(c) Intermediate iteration.

Figure A.1: Example 1 for Algorithm A.2 (replicate ordering).

	Orig	En	Orig/En
Candidate Graphs	26	8	3.25
Unique Graphs	5	5	1
Generation Time (s)	0.0052	0.0033	1.58
Total Time (s)	0.0095	0.0060	1.58

Table A.1: Comparison (replicate ordering, [Example 1](#)).

A.2.3.2 Example 2

The base three-tuple and NSCs for this example are specified as:

$$C = \{W, X, Y, Z\}, \quad R = [3 \ 4 \ 2 \ 1], \quad P = [1 \ 2 \ 2 \ 3], \quad \text{no additional NSCs} \quad (\text{A.4})$$

The second and third inputs for Algorithm [A.2](#) are:

$$Vf = [1 \ 1 \ 1 \ 2 \ 2 \ 2 \ 2 \ 2 \ 3], \quad initRep = [1 \ 4 \ 8 \ 10] \quad (\text{A.5})$$

We will consider two different V , one at the initial iteration of the tree algorithm and one at some intermediate iteration. Figure [A.2a](#) goes through the operations in Algorithm [A.2](#) for the different V . The example iterations are also visualized in Figs. [A.2b](#) and [A.2c](#).

Table [A.2](#) compares the original algorithm with the enhancement for this example. There is a reduction in candidate graphs generated while the number of unique graphs remains the same.

	Orig	En	Orig/En
Candidate Graphs	456766	14359	31.8
Unique Graphs	1657	1657	1
Generation Time (s)	16.26	0.69	23.6
Total Time (s)	3577	145	24.7

Table A.2: Comparison (replicate ordering, [Example 2](#)).

A.3 Avoiding Loops

Under specific NSCs, we can exclude loops during the graph generation process. A loop is an edge that connects a vertex to itself [\[61, p. 25\]](#).

	Initial	Intermediate
V	$[0 \ 1 \ 1 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2 \ 3]$	$[0 \ 0 \ 0 \ 0 \ 1 \ 2 \ 2 \ 1 \ 2 \ 3]$
$\text{circshift}(V[0 \ 1])$	$[3 \ 0 \ 1 \ 1 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2]$	$[3 \ 0 \ 0 \ 0 \ 0 \ 1 \ 2 \ 2 \ 1 \ 2]$
V_f	$[1 \ 1 \ 1 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2 \ 3]$	$[1 \ 1 \ 1 \ 2 \ 2 \ 2 \ 2 \ 2 \ 2 \ 3]$
$\text{circshift}(V[0 \ 1]) \neq V_f$	$[1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1]$	$[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1]$
$V\text{ordering}(i\text{InitRep}) \leftarrow 1$	$[1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1]$	$[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1]$
Removed/total branches	$5/9$	$1/6$

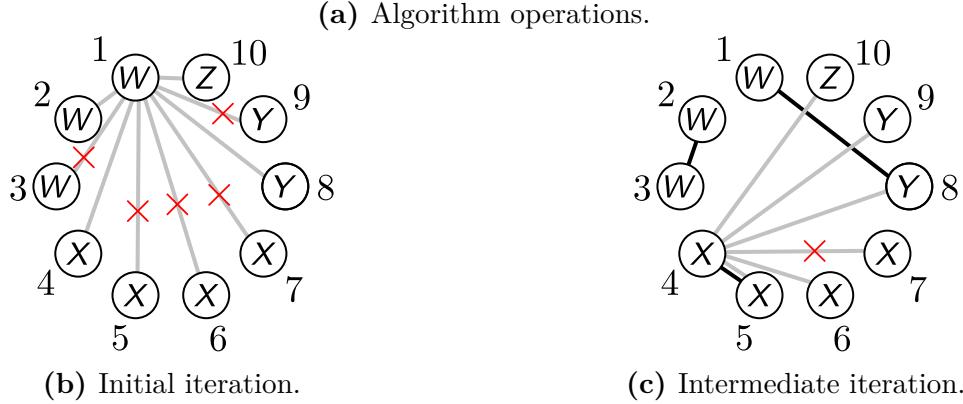


Figure A.2: Example 2 for Algorithm A.2 (replicate ordering).

A.3.1 Theory

Consider a component type that is both mandatory (S_3) and each edge is required to be unique (S_6). Then a feasible graph cannot have any loops for this component type since the number of edges would not be unique. We cannot make the same assumption if a component type needs to have each edge be unique but is not a mandatory component. Consider a two-port, nonmandatory component type with one replicate. Now, if we want a connected graph with all components except this specific two-port component, then a loop is required since each port must be filled. Therefore, this enhancement can be implemented with NAND logic where only a mandatory component type with each connection required to be unique is excluded from having loops.

This is not a general enhancement since it requires specific NSCs.

A.3.2 Implementation

Algorithm A.3 is the pseudocode for the implementation of this enhancement. The implementation is centered around modifying the expanded potential adjacency matrix (A) before the graph generation algorithm is called. The total number of components is found and

Algorithm A.3: Limit potential connections based on loops.

Input : A – expanded potential adjacency matrix
 M – vector indicating if a component replicate is mandatory
 U – vector indicating if a component replicate requires unique connections

Output: A – expanded potential adjacency matrix

```

1 if any( $M \wedge U$ ) then                                /* some loops should be excluded */
2    $N \leftarrow \text{length}(M)$                          /* total number of component replicates */
3    $i\text{Diag} \leftarrow [1 : N + 1 : N^2]$            /* indices for the diagonal elements */
4    $A(i\text{Diag}) \leftarrow !(M \wedge U)$             /* assign NAND between  $M$  and  $U$  to the diagonal */
5 end

```

then the linear index values for the diagonal of expanded potential adjacency matrix are computed. Finally, in line 4, NAND logical operator between M and U is assigned to the diagonal of A .

A.3.3 Examples

A.3.3.1 Example 1

The base three-tuple and NSCs for this example are specified as:

$$C = \{X, Y\}, \quad R = [2 \ 2], \quad P = [2 \ 2], \quad S_3 \text{ with } M = [0 \ 1], \quad S_6 \text{ with } U = [1 \ 1] \quad (\text{A.6})$$

We will consider an all unity A but this could be any expanded potential adjacency matrix. The second and third inputs to Alg. A.3 are:

$$M = [0 \ 0 \ 1 \ 1], \quad U = [1 \ 1 \ 1 \ 1], \quad !(M \wedge U) = [1 \ 1 \ 0 \ 0] \quad (\text{A.7})$$

Now, Alg. A.3 modifies the expanded potential adjacency matrix as:

$$A = \begin{array}{c|cccc} & X & X & Y & Y \\ \begin{matrix} X \\ X \\ Y \\ Y \end{matrix} & \begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{matrix} & \xrightarrow{A(i\text{Diag}) \leftarrow !(M \wedge U)} & \begin{matrix} X & X & Y & Y \\ \begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{matrix} \end{matrix} \end{array} \quad (\text{A.8})$$

where Y is mandatory and unique connections are required so corresponding diagonal entries in A were zeroed. Table A.3 compares the original algorithm with the enhancement for this example. There is a reduction in candidate graphs generated while the number of unique graphs remains the same.

	Orig	En	Orig/En
Candidate Graphs	26	16	1.63
Unique Graphs	3	3	1
Generation Time (s)	0.0024	0.0022	1.09
Total Time (s)	0.0086	0.0070	1.23

Table A.3: Comparison (loops, [Example 1](#)).

A.3.3.2 Example 2

The base three-tuple and NSCs for this example are specified as:

$$C = \{X, Y\}, \quad R = [2 \ 2], \quad P = [2 \ 2], \quad S_3 \text{ with } M = [1 \ 1], \quad S_6 \text{ with } U = [1 \ 1] \quad (\text{A.9})$$

We will consider an all unity A but this could be any expanded potential adjacency matrix. The second input to Alg. [A.3](#) is:

$$M = [1 \ 1 \ 1 \ 1], \quad U = [1 \ 1 \ 1 \ 1], \quad !(M \wedge U) = [0 \ 0 \ 0 \ 0] \quad (\text{A.10})$$

Now, Alg. [A.3](#) modifies the expanded potential adjacency matrix as:

$$A = \begin{array}{cc} & \begin{matrix} X & X & Y & Y \end{matrix} \\ \begin{matrix} X \\ X \\ Y \\ Y \end{matrix} & \left[\begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{matrix} \right] \xrightarrow{A(iDiag) \leftarrow !(M \wedge U)} \begin{matrix} X & X & Y & Y \end{matrix} \\ & \left[\begin{matrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{matrix} \right] \end{array} \quad (\text{A.11})$$

Since both component types are mandatory and unique connections are required, the entire diagonal is zeroed. Table [A.4](#) compares the original algorithm with the enhancement for this example. There is a reduction in candidate graphs generated while the number of unique graphs remains the same.

	Orig	En	Orig/En
Candidate Graphs	26	11	2.36
Unique Graphs	2	2	1
Generation Time (s)	0.0024	0.0022	1.09
Total Time (s)	0.0086	0.0060	1.43

Table A.4: Comparison (loops, [Example 2](#)).

A.4 Avoiding Multi-Edges

Under specific NSCs, we can exclude multi-edges during the graph generation process. A multi-edge is two or more edges that are incident to the same two vertices [61, p. 25].

A.4.1 Theory

Consider when each edge is required to be unique (S_6). Due to the sequential nature of the tree algorithm, a single edge must be added between two components before a second edge is added; thus, creating a multi-edge. Therefore, when the first edge is added between two components, we can utilize the expanded potential adjacency matrix to disallow any further connections between the components. Since a feasible graph would not have any multi-edges when each edge is required to be unique, the tree algorithm with this enhancement will continue generating the desired graph structure space.

This enhancement should not be applied on loops since loops are occasionally needed to remove components (see Sec. A.3 for the handling of loops). Also, this is not a general enhancement since it requires specific NSCs.

A.4.2 Implementation

Algorithm A.4: Limit potential connections based on multi-edges.

Input : A – expanded potential adjacency matrix
U – vector indicating if a component replicate requires unique connections
iR – component index for right port
iL – component index for left port

Output: A – expanded potential adjacency matrix

```
1 if U(iL) ∨ U(iR) then          /* either component requires unique connections */
2   |  if iR ≠ iL then           /* don't do for self loops */
3     |    A(iR, iL) ← 0         /* limit this connection */
4     |    A(iL, iR) ← 0         /* limit this connection */
5   end
6 end
```

Algorithm A.4 is the pseudocode for the implementation of this enhancement. The implementation is centered around modifying the expanded potential adjacency matrix (A) when a new edge is created. This enhancement is only called if either component replicate in the edge requires unique connections. Additionally, this enhancement is only called if the edge is

not a loop (see Sec. A.3 for the handling of loops). If both of these conditions are met, then the corresponding entries in the expanded potential adjacency matrix are zeroed in lines 3 and 4.

This enhancement is inserted between lines 11 and 12 of Alg. A.1 using the local copy A2.

A.4.3 Example

The base three-tuple and NSCs for this example are specified as:

$$C = \{W, X, Y, Z\}, \quad R = [1 1 1 1], \quad P = [3 3 3 3], \quad S_6 \text{ with } U = [1 1 1 1] \quad (\text{A.12})$$

Consider one path during the graph generation process when the reduced potential adjacency matrix is initially all ones:

$$\begin{array}{c}
 \begin{array}{cccc} W & X & Y & Z \end{array} \\
 \begin{array}{c} W \\ X \\ Y \\ Z \end{array} \left[\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{array} \right] \xrightarrow{} \begin{array}{cccc} W & X & Y & Z \end{array} \\
 \begin{array}{c} X \\ Y \\ Z \end{array} \left[\begin{array}{cccc} 1 & \mathbf{0} & 1 & 1 \\ \mathbf{0} & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{array} \right] \xrightarrow{} \begin{array}{cccc} W & X & Y & Z \end{array} \\
 \begin{array}{c} X \\ Y \\ Z \end{array} \left[\begin{array}{cccc} 1 & 0 & \mathbf{0} & 1 \\ 0 & 1 & 1 & 1 \\ \mathbf{0} & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{array} \right] \xrightarrow{} \\
 \underbrace{\hspace{10em}}_{\text{Iter. 0, } V = [3 3 3 3]} \quad \underbrace{\hspace{10em}}_{\text{Iter. 1, } V = [\bar{2} \bar{2} 3 3]} \quad \underbrace{\hspace{10em}}_{\text{Iter. 2, } V = [\bar{1} \cancel{2} \bar{2} 3]} \\
 \begin{array}{c} W \\ X \\ Y \\ Z \end{array} \left[\begin{array}{cccc} 1 & 0 & 0 & \mathbf{0} \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ \mathbf{0} & 1 & 1 & 1 \end{array} \right] \xrightarrow{} \begin{array}{cccc} W & X & Y & Z \end{array} \\
 \begin{array}{c} X \\ Y \\ Z \end{array} \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & \mathbf{0} & 1 \\ 0 & \mathbf{0} & 1 & 1 \\ 0 & 1 & 1 & 1 \end{array} \right] \xrightarrow{} \begin{array}{cccc} W & X & Y & Z \end{array} \\
 \begin{array}{c} X \\ Y \\ Z \end{array} \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \mathbf{0} \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{array} \right] \\
 \underbrace{\hspace{10em}}_{\text{Iter. 3, } V = [\bar{0} \cancel{2} \cancel{2} \bar{2}]} \quad \underbrace{\hspace{10em}}_{\text{Iter. 4, } V = [0 \bar{1} \bar{1} 2]} \quad \underbrace{\hspace{10em}}_{\text{Iter. 5, } V = [0 \bar{0} \cancel{1} \bar{1}]}
 \end{array}$$

where the matrix above represents \mathbf{A} , \square indicates this component was selected for an edge, and $\cancel{\square}$ indicates the connection was disallowed. Each iteration added a pair of zeros to the potential adjacency matrix.

Table A.5 compares the original algorithm with the enhancement for this example. There is a reduction in candidate graphs generated while the number of unique graphs remains the same.

	Orig	En	Orig/En
Candidate Graphs	211	46	4.59
Unique Graphs	1	1	1
Generation Time (s)	0.0086	0.0037	2.32
Total Time (s)	0.015	0.0070	2.14

Table A.5: Comparison (multi-edge).

A.5 Avoiding Line-Connectivity Constraints

On line 4 of Alg. A.1, we utilize the expanded potential adjacency matrix (\mathbf{A}) to disallow connections between components. This is also phrased as every graph must have edges between vertices that are feasible. These are termed vertex-connectivity constraints, denoted S_7 . A similar type of NSC can be included between the lines of the graph, termed line-connectivity constraints.

A.5.1 Theory

The line graph of a graph G is the graph with the edges of G as its vertices, and where two edges of G are adjacent in the line graph if and only if they are incident in G [60, p. 10]. Consider the graph in Fig. A.3a and its corresponding line graph in Fig. A.3b with three component types. If component type 1 is connected to component type 2, we can specify if a connection between component types 2 and 3 is allowed. This is equivalent to specifying if line type (1, 2) can be connected to line type (2, 3).

This enhancement is a new type of NSC and is designated S_8 .



Figure A.3: Illustration of a line-connectivity constraint.

A.5.2 Implementation

Since this enhancement is a new type of NSC, it is specified before the graph generation procedure. For each line-connectivity constraint, a triple of integers is supplied defining the component types in Fig. A.3a. They are supplied in increasing order, i.e., $[#1, #2, #3]$.

Therefore, each triple is interpreted as: if #1 and #2 are connected, don't ever connect #2 to #3. These triples help construct the reduced 3-D array with line-connectivity constraint information, B . They are indexed in reverse order to facilitate extracting column vectors, i.e., $B(\#3, \#2, \#1) = 0$. Given a set of triples, a function creates the expanded $N \times N \times N$ matrix where N is the total number of component replicates where a zero indicates a connection is not allowed and one indicates it is allowed.

Algorithm A.5: Limit potential connections based on line-connectivity constraints.

Input : A – expanded potential adjacency matrix
 iR – component index for right port
 iL – component index for left port
 B – 3-D array with line-connectivity constraint information

Output: A – expanded potential adjacency matrix

```

1 if there are any line-connectivity constraints then
2    $A(:, iR) \leftarrow A(:, iR) \circ B(:, iR, iL)$            /* potentially limit connections */
3    $A(:, iL) \leftarrow A(:, iL) \circ B(:, iL, iR)$            /* potentially limit connections */
4    $A([iR, iL], :) \leftarrow A(:, [iR, iL])^T$           /* make symmetric */
5 end
```

Algorithm A.5 is the pseudocode for the implementation of this enhancement. This enhancement is only called if there are any line-connectivity constraints, S_8 . First using the component index of the right port, we extract a vector from the 3-D array with line-connectivity constraint information (B) when $\#1 = iL$ and $\#2 = iR$. This vector is multiplied element-wise with the correct row of the expanded potential adjacency matrix (A), potentially limiting connections. This step then performed again switching the roles of component indices, i.e., $\#1 = iR$ and $\#2 = iL$. Finally, the changes to A are applied to the symmetric location, ensuring A remains symmetric.

This enhancement is inserted between lines 11 and 12 of Alg. A.1 using the local copy A2.

A.5.3 Examples

A.5.3.1 Example 1

The base three-tuple and NSCs for this example are specified as:

$$C = \{X, Y, Z\}, \quad R = [1 \ 2 \ 2], \quad P = [2 \ 2 \ 2], \quad S_8(1) = [1, 2, 3] \quad (\text{A.13})$$

The reduced (B) and expanded (B) 3-D arrays with line-connectivity constraint information are:

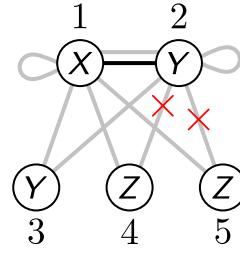
$$B(:,:,1) = \begin{matrix} X & Y & Z \\ X & \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \\ Y & \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \\ Z & \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \end{matrix}, \quad B(:,:,1) = \begin{matrix} X & Y & Y & Z & Z \\ X & \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ Y & \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ Y & \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ Z & \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \end{bmatrix} \\ Z & \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \end{bmatrix} \end{matrix} \quad (\text{A.14})$$

Figure A.4a goes through the operations in Algorithm A.5 for a certain line type. The limiting of potential edges is visualized in Fig. A.4b.

Table A.6 compares the original algorithm with the enhancement for this example. There is a reduction in candidate graphs generated while the number of unique graphs remains the same.

	Line Type 1
iL	1
iR	2
$B(:, iR, iL)$	$[1 \ 1 \ 1 \ 0 \ 0]^T$
$B(:, iL, iR)$	$[1 \ 1 \ 1 \ 1 \ 1]^T$

(a) Algorithm operations.



(b) Line Type 1.

Figure A.4: Example 1 for Algorithm A.5 (line-connectivity constraints).

	Orig	En	Orig/En
Candidate Graphs	146	98	1.49
Unique Graphs	26	26	1
Generation Time (s)	0.004	0.005	0.80
Total Time (s)	0.036	0.034	1.06

Table A.6: Comparison (line-constraints, Example 1).

A.5.3.2 Example 2

The base three-tuple and NSCs for this example are specified as:

$$\begin{aligned} C &= \{X, Y, Z\}, \quad R = [2 \ 2 \ 3], \quad P = [2 \ 2 \ 2] \\ S_8(1) &= [1, 2, 2], \quad S_8(2) = [2, 1, 2], \quad S_8(3) = [3, 3, 3] \end{aligned} \quad (\text{A.15})$$

The reduced 3-D array with line-connectivity constraint information (B) is:

$$\begin{aligned}
 B(:,:,1) &= \begin{matrix} X & Y & Z \\ X & \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, & B(:,:,2) = \begin{matrix} X & Y & Z \\ Y & \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \\ Z & \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{matrix} \\
 B(:,:,3) &= \begin{matrix} X & Y & Z \\ X & \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \\ Y & \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \\ Z & \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{matrix}
 \end{aligned} \tag{A.16}$$

The expanded 3-D arrays with line-connectivity constraint information (B) for $S_8(3)$ are:

$$B(:,:,5) = B(:,:,6) = B(:,:,7) = \begin{matrix} X & X & Y & Y & Z & Z & Z \\ X & \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ X & \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ Y & \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ Y & \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ Z & \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \\ Z & \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \tag{A.17}$$

Figure A.5a goes through the operations in Algorithm A.5 for two line types. The limiting of potential edges is visualized in Figs. A.5b and A.5c.

Table A.7 compares the original algorithm with the enhancement for this example. There is a reduction in candidate graphs generated while the number of unique graphs remains the same.

	Orig	En	Orig/En
Candidate Graphs	8316	5120	1.62
Unique Graphs	119	119	1
Generation Time (s)	0.184	0.183	1.01
Total Time (s)	2.361	2.096	1.13

Table A.7: Comparison (line-constraints, Example 2).

	Line Type 1	Line Type 2
iL	1	5
iR	3	6
$B(:, iR, iL)$	$[1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1]^T$	$[1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0]^T$
$B(:, iL, iR)$	$[1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1]^T$	$[1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0]^T$

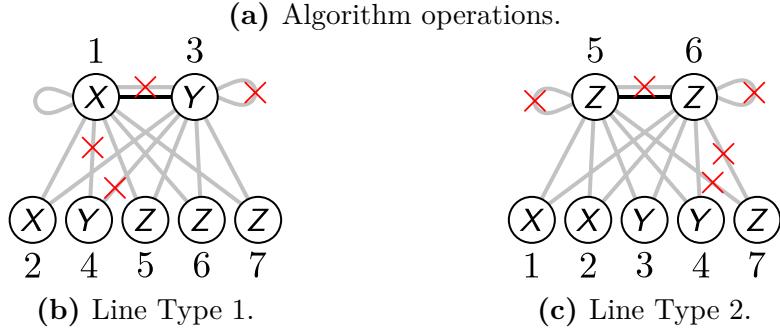


Figure A.5: Example 2 for Algorithm A.5 (line-connectivity constraints).

A.6 Checking for Saturated Subgraphs

This enhancement is based on the work in Ref. [73] for enumerating molecules. In their work, all atoms are mandatory in the graph. Therefore, the detection of a saturated subgraph before all atoms have been connected indicates the candidate graph will be infeasible and can be discarded.

A.6.1 Theory

A subgraph of a graph G is another graph formed from a subset of the vertices and edges of G [61, p. 3]. A saturated subgraph is a subgraph with no empty ports and may contain multiple connected subgraphs [73]. Since a saturated subgraph has no empty ports, no components other than the components currently in this subgraph will be connected to this subgraph during further iterations of the graph generation procedure. If we determine that the current iteration of the tree algorithm has created a saturated subgraph, then there are three scenarios to consider.

First, if all mandatory components are contained in the saturated subgraph, then no additional iterations are needed. Since all components not connected to a mandatory component will be removed, all components not currently in the saturated subgraph will be removed. Therefore, the topology of the remaining components is negligible. Since the topology is neg-

ligible, we can assign an arbitrary topology to the remaining components, save the graph, and terminate the iteration.

The second scenario is if none of the mandatory components are in the saturated subgraph. This provides no additional information so we allow the current iteration to continue. The final scenario is when some, but not all, mandatory components are in the saturated subgraph. Since at least one pair of mandatory components will not be connected, this graph will be infeasible. Therefore, we can terminate this iteration without saving the graph.

This is not a general enhancement since it requires specific NSCs, namely at least one mandatory component (S_3).

A.6.2 Implementation

Algorithm A.6: Handle saturated subgraphs.

Input : V – vector of remaining ports for each component replicate
 E – vector of edges in sequential pairs
 Vf – initial vector of ports available for each component replicate
 M – vector indicating if all replicates of the component type must be present
 cVf – cumulative sum of the original V plus 1
 SavedGraphs – set of graphs

Output: SavedGraphs – set of graphs

```

1  if there are any necessary components then
2     $i\text{NonSat} \leftarrow \text{find}(V)$                                 /* find the nonsaturated components */
3    if  $V(i\text{NonSat}) = Vf(i\text{NonSat})$  then                      /* check for saturated subgraph */
4       $n\text{Uncon} \leftarrow \text{sum}(M(i\text{NonSat}))$                   /* # of mandatory comps not in saturated subgraph */
5      if  $n\text{Uncon} = 0$  then                                         /* all mandatory components are in saturated subgraph */
6        for  $j \leftarrow 1$  to  $\text{sum}(V)$  do                               /* add remaining edges in arbitrary order */
7           $k \leftarrow \text{find}(V, 1)$                                          /* find first nonzero entry */
8           $LR \leftarrow cVf(k) - V(k)$ 
9           $V(k) \leftarrow V(k) - 1$                                          /* remove port */
10          $E \leftarrow [E, LR]$                                          /* add port */
11      end
12       $\text{SavedGraphs}\{\text{end} + 1\} \leftarrow E$                          /* missorted perfect matching */
13      continue                                                       /* stop iteration, graph has been added */
14    else if  $n\text{Uncon} = \text{sum}(M)$  then                          /* no mandatory comps are in saturated subgraph */
15      /* continue with this iteration */
16    else                                                               /* some but not all mandatory components are in saturated subgraph */
17      continue                                                       /* stop iteration, this graph is infeasible */
18    end
19  end

```

Algorithm A.6 is the pseudocode for the implementation of this enhancement. This en-

hancement is only called if there are some mandatory components (S_3). First, the unsaturated components are found by checking the vector of remaining ports for each component replicate, \mathbf{V} . A component is saturated if all ports are filled. To determine if the current graph is a saturated subgraph, we compare the remaining ports of the unsaturated components to the original number of ports available (see line 3).

If the current graph is indeed a saturated subgraph, then we compute the number of mandatory components not in the saturated subgraph, n_{Uncon} . If all mandatory components are in the saturated subgraph, then this graph is feasible. We assign an arbitrary ordering to the remaining components, save the graph, and terminate the iteration (see lines 6 to 13). If no mandatory components are in the saturated subgraph, then we allow the current iteration to continue (see line 14). Finally, if some but not all mandatory components are in the saturated subgraph, we stop the iteration since the graph is infeasible (see lines 15 and 16).

This enhancement is inserted between lines 11 and 12 of Alg. A.1 since the current edge needs to be added but before the recursion call. With this enhancement, the `else` statement on line 14 of Alg. A.1 will never be reached if there are any mandatory components. The `if` condition is only untrue when a saturated subgraph is present (every component's port being filled).

A.6.3 Examples

A.6.3.1 Example 1

The base three-tuple and NSCs for this example are specified as:

$$C = \{X, Y\}, \quad R = [2 \ 3], \quad P = [2 \ 2], \quad S_3 \text{ with } M = [1 \ 0] \quad (\text{A.18})$$

Some of the other inputs are then:

$$\mathbf{Vf} = [2 \ 2 \ 2 \ 2 \ 2], \quad \mathbf{M} = [1 \ 1 \ 0 \ 0 \ 0] \quad (\text{A.19})$$

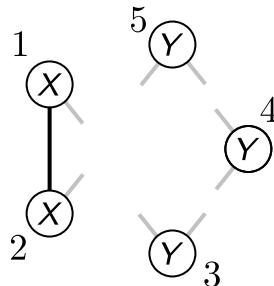
We will consider two different \mathbf{V} , one after an initial edge is added and one at some intermediate iteration. Figure A.6a goes through the operations in Algorithm A.6 for the different \mathbf{V} . These example iterations are also visualized in Figs. A.6b and A.6c.

Table A.8 compares the original algorithm with the enhancement for this example. There is a reduction in candidate graphs generated while the number of unique graphs remains the

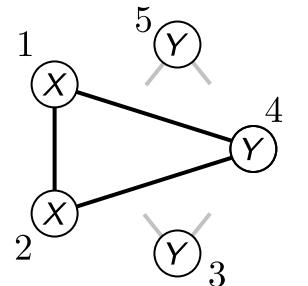
same.

	Iteration	Initial	Intermediate
V	[1 1 2 2 2]	[0 0 2 0 2]	
$iNonSat \leftarrow \text{find}(V)$	[1 2 3 4 5]	[3 5]	
$V(iNonSat)$	[1 1 2 2 2]	[2 2]	
$Vf(iNonSat)$	[2 2 2 2 2]	[2 2]	
$V(iNonSat) = Vf(iNonSat)$	False	True	
$M(iNonSat)$	—	[0 0]	
$nUncon \leftarrow \text{sum}(M(iNonSat))$	—	0	
Feasible	—	Yes	

(a) Algorithm operations.



(b) Initial iteration.



(c) Intermediate iteration.

Figure A.6: Example 1 for Algorithm A.6 (saturated subgraphs).

	Orig	En	Orig/En
Candidate Graphs	146	91	1.60
Unique Graphs	6	6	1
Generation Time (s)	0.0056	0.0046	1.22
Total Time (s)	0.023	0.015	1.53

Table A.8: Comparison (saturated subgraphs, Example 1).

A.6.3.2 Example 2

The base three-tuple and NSCs for this example are specified as:

$$C = \{X\}, \quad R = [9], \quad P = [2], \quad S_3 \text{ with } M = [1] \quad (\text{A.20})$$

Some of the other inputs are then:

$$Vf = [2 2 2 2 2 2 2 2 2], \quad M = [1 1 1 1 1 1 1 1 1] \quad (\text{A.21})$$

We will consider two different V , one after an initial edge is added and one at some intermediate iteration. Figure A.7a goes through the operations in Algorithm A.6 for the different V . These example iterations are also visualized in Figs. A.7b and A.7c.

Table A.9 compares the original algorithm with the enhancement for this example. There is a reduction in candidate graphs generated while the number of unique graphs remains the same.

	Intermediate 1	Intermediate 2
V	[0 0 0 1 2 1 2 2 2]	[0 0 0 0 2 0 2 2 2]
$iNonSat \leftarrow \text{find}(V)$	[4 5 6 7 8 9]	[5 7 8 9]
$V(iNonSat)$	[1 2 1 2 2 2]	[2 2 2 2]
$Vf(iNonSat)$	[2 2 2 2 2]	[2 2 2 2]
$V(iNonSat) = Vf(iNonSat)$	False	True
$M(iNonSat)$	—	[1 1 1 1]
$nUncon \leftarrow \text{sum}(M(iNonSat))$	—	4
Feasible	—	No

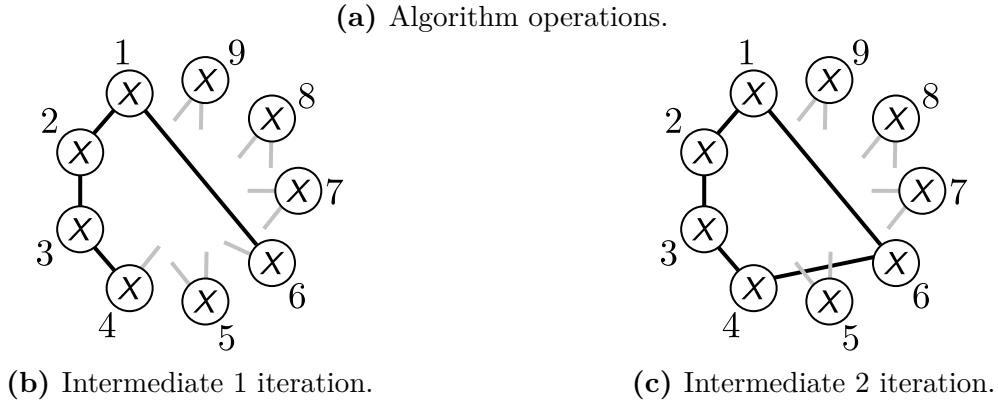


Figure A.7: Example 2 for Algorithm A.6 (saturated subgraphs).

	Orig	En	Orig/En
Candidate Graphs	852316	460872	1.85
Unique Graphs	1	1	1
Generation Time (s)	29.83	19.23	1.55
Total Time (s)	58.57	33.11	1.77

Table A.9: Comparison (saturated subgraphs, Example 2).

A.7 Enumerating Subcatalogs

Leveraging some of the properties of the graph structure space and some NSCs, we can break the graph generation procedure into subtasks that more efficiently generate the same graph structure space.

A.7.1 Theory

A candidate architecture in an architecture design space described by (C, R, P) has the following properties [6]:

1. A set of component replicates bounded by (C, R, P) . This set of component replicates is termed a subcatalog of (C, R, P) .
2. Each port in $(V^P, \{\}, L^P)$, i.e., G^P without edges, is connected to another port (this implies an even number of ports).

The original tree algorithm in Alg. A.1 generated all candidate architectures in this architecture design space since the set of perfect matchings (PMs) graphs of K_N contains all edge sets for K_{N-2} , where $N \geq 4$ [6]. Instead of relying on this property, an alternative would be an enumeration of all possible subcatalogs of (C, R, P) . This property is no longer strictly needed since the edge sets of the PMs of K_N for each subcatalog is enough to generate all the desired graphs. However, this approach on its own provides no general improvements to the graph generation procedure.

If we require every graph to be a connected graph (S_1), we can enforce the following: A feasible graph for a specific subcatalog must have every component replicate connected (i.e., all the replicates are mandatory). This is due to the tree algorithm being utilized on every subcatalog. Enumerating subcatalogs only provides general improvements to the graph generation procedure if the property that all replicates are mandatory in each subcatalog is effectively utilized.

Two of the previously discussed enhancements utilize this property: 1) checking for saturated subgraphs in Sec. A.6 and 2) avoiding loops in Sec. A.3. A greater proportion of mandatory component types improves the effectiveness of these enhancements. Another benefit is isomorphism checks only need to be performed between graphs in their respective subcatalog. Since every component type is mandatory in the subcatalog, the colored label sets will be different between graphs in different subcatalogs. Therefore, graphs from different subcatalogs are definitely not isomorphic. This reduces the number of isomorphism

checks and allows for further parallelization. Finally, the generation of graphs for each subcatalog can be performed in parallel. However, each subcatalog will take varying amounts of time to complete so the benefit will vary depending on the particular (C, R, P) and NSCs. The original tree algorithm does not leverage parallelization during the graph generation procedure.

This enhancement also allows for an improved representation of the number of replicates for each component type. Instead of the original vector R , where each entry was the maximum number of replicates for the specific component type, minimum and maximum values can be specified. The maximum values, denoted R_{\max} , is equivalent to the previous R . The minimum values, denoted R_{\min} , can naturally capture mandatory components and nonzero lower bounds. Every nonzero element of R_{\min} indicates a mandatory component type. If R_{\min} and R_{\max} for a component type is 2 and 5, then there must be between 2 and 5 replicates in a feasible graph.

The set of subcatalogs contains all possible combinations of integers values for each component type bounded by R_{\min} and R_{\max} . Therefore, the total number of subcatalogs is:

$$N_{\text{subcatalogs}} = \prod_{k=1}^{|R_{\max}|} [R_{\max}(k) - R_{\min}(k) + 1] \quad (\text{A.22})$$

Some of these subcatalogs may be invalid, e.g., the subcatalog has an odd number of ports or is empty.

A.7.2 Implementation

Algorithm A.7 is the pseudocode for the implementation of this enhancement. This enhancement is only called if we require every graph to be a connected graph (S_1)

First, the potential number of replicates for each component type is stored in a cell array, Rlist . All subcatalogs are then generated with `ndgrid` using Rlist . This function creates a rectangular grid in N-D space [292]. Next, the subcatalogs are filtered for subcatalogs with an odd number of ports or are empty. Additional user-specified filters can all be applied here. Once all the filters have been applied, the number of subcatalogs is calculated. The final step before generating the graphs is to create a local copy of the network structure constraints since they are modified for each subcatalog.

Generating graphs for each subcatalog can now be performed in parallel (see line 8). Before a subcatalog is used, a number of items need to be updated to properly define the subcatalog. We find the locations of the nonzero replicates on line 10. Then the colored

Algorithm A.7: Generate set of unique, feasible graphs using subcatalogs.

Input : Rmin – vector indicating min number of replicates for each component type
 Rmax – vector indicating the max number of replicates for each component type
 C – colored label set
 P – ports vector
 NSC – structure for the network structure constraints

Output: FinalGraphs – set of unique, feasible graphs

```

1 for k ← 1 to length(Rmax) do                                /* for each component type */
2   | Rlist{k} ← Rmin(k) : 1 : Rmax(k)                         /* list of potential number of replicates */
3 end
4 Subcatalogs ← matrix form of the cell array from ndgrid(Rlist)      /* generate subcatalogs */
5 Subcatalogs ← filter Subcatalogs (empty, odd port, custom filters)
6 Nsubcatalogs ← number of rows in Subcatalogs                      /* number of subcatalogs */
7 nsc ← NSC                                                        /* local NSC structure */
8 for k ← 1 to Nsubcatalogs do in parallel
9   | r ← Subcatalogs(k,:)                                         /* extract R vector for this subcatalog test */
10  | I ← r ≠ 0                                                 /* nonzero replicate locations */
11  | c ← C(I)                                                 /* extract colored labels */
12  | r ← r(I)                                                 /* extract replicates vector */
13  | p ← P(I)                                                 /* extract ports vector */
14  | nsc.U ← NSC.U(I)                                         /* extract unique connections vector */
15  | nsc.A ← NSC.A(I,:)                                       /* extract reduced potential adjacency matrix */
16  | nsc.A(:,I) ← NSC.A(:,I)                                   /* symmetric */
17  | nsc.B ← extract appropriate line-connectivity triples using NSC.B and I
18  | nsc.M ← ones(size(r))                                     /* all component types are mandatory */
19  | Graphs{k} ← generate feasible graphs for this subcatalog using c, r, p, and nsc
20 end
21 for k ← 1 to Nsubcatalogs do in parallel                      /* obtain unique graphs */
22   | Graphs{k} ← determine set of unique graphs in Graphs{k}
23 end
24 FinalGraphs ← combine unique graphs from each subcatalog using Graphs

```

labels, replicates vector, and ports vector are updated for this subcatalog, only including component types with at least one replicate. In addition, both the unique connections vector, reduced potential adjacency matrix, and line-connectivity triples need to be updated to include only the relevant constraints for this subcatalog. Some component types may not be present in a particular subcatalog, so any NSCs with these component types are not needed. Since all component types are mandatory in this approach, we set each component type as mandatory on line 18. Finally, we generate feasible graphs for this subcatalog using the same method used for a single catalog.

After the feasible graphs have been found, each subcatalog can be analyzed for the set of unique graphs. Again, this task can be performed in parallel as each subcatalog is independent. The final step is to combine all the unique graphs into a single set.

A.7.3 Examples

These examples are tested using this enhancement and the handling of saturated subgraphs in Sec. A.6.

A.7.3.1 Example 1

The base three-tuple and NSCs for this example are specified as:

$$C = \{X, Y\}, \quad R_{\min} = [1 \ 0], \quad R_{\max} = [1 \ 8], \quad P = [2 \ 2] \quad (\text{A.23})$$

All 9 subcatalogs for this example are shown in Table A.10.

Table A.11 compares the original algorithm with the enhancement for this example. There is a reduction in feasible graphs generated while the number of unique graphs remains the same. The enhancement with 12 threads (12T) and 1 thread (1T) available is shown.

#	r	c	p	Feasible Graphs
1	[1]	{X}	[2]	1
2	[1 1]	{X, Y}	[2 2]	1
3	[1 2]	{X, Y}	[2 2]	1
4	[1 3]	{X, Y}	[2 2]	3
5	[1 4]	{X, Y}	[2 2]	12
6	[1 5]	{X, Y}	[2 2]	60
7	[1 6]	{X, Y}	[2 2]	360
8	[1 7]	{X, Y}	[2 2]	2520
9	[1 8]	{X, Y}	[2 2]	20160

Table A.10: Subcatalogs for Example 1.

	Orig	En	En (12T)	Orig/En	Orig/En (12T)
Feasible Graphs	96940	23118	23118	4.19	4.19
Unique Graphs	9	9	9	1	1
Generation Time (s)	29.857	27.385	26.371	1.09	1.13
Total Time (s)	39.604	29.943	29.721	1.32	1.33

Table A.11: Comparison (enumerating subcatalogs, Example 1).

A.7.3.2 Example 2

The base three-tuple and NSCs for this example are specified as:

$$C = \{W, X, Y, Z\}, \quad R_{\min} = [1 0 0 0], \quad R_{\max} = [1 2 3 3], \quad P = [1 1 2 3] \quad (\text{A.24})$$

A select number of the 48 subcatalogs for this example are shown in Table A.12.

Table A.13 compares the original algorithm with the enhancement for this example. There is a reduction of feasible graphs generated while the number of unique graphs remains the same. The enhancement with 12 threads (12T) available is also shown.

#	r	c	p	Feasible Graphs
1	[1 0 0 0]	{W}	[1]	– (odd)
2	[1 1 0 0]	{W, X}	[1 1]	1
3	[1 2 0 0]	{W, X}	[1 1]	– (odd)
4	[1 0 1 0]	{W, Y}	[1 2]	– (odd)
5	[1 1 1 0]	{W, X, Y}	[1 1 2]	1
⋮	⋮	⋮	⋮	⋮
44	[1 1 2 3]	{W, X, Y, Z}	[1 1 2 3]	– (odd)
45	[1 2 2 3]	{W, X, Y, Z}	[1 1 2 3]	1548
46	[1 0 3 3]	{W, Y, Z}	[1 2 3]	1683
47	[1 1 3 3]	{W, X, Y, Z}	[1 1 2 3]	– (odd)
48	[1 2 3 3]	{W, X, Y, Z}	[1 1 2 3]	11844

Table A.12: Select subcatalogs for Example 2.

	Orig	En	En (12T)	Orig/En	Orig/En (12T)
Feasible Graphs	45015	16235	16235	2.77	2.77
Unique Graphs	489	489	489	1	1
Generation Time (s)	8.940	12.903	11.216	0.69	0.80
Total Time (s)	60.977	49.285	44.291	1.24	1.38

Table A.13: Comparison (enumerating subcatalogs, Example 2).

A.8 Alternative Tree Traversal Strategies

Visualized in Fig. 2.6, the main algorithm in Alg. A.1 for enumerating the graph structure space of interest is functionally equivalent to visiting all nodes in a tree, denoted τ . Here we

will further characterize the tree structure and alternative strategies for traversing it. All the enhancements discussed in this appendix can be readily incorporated into the alternative tree traversal strategies discussed here.

A *tree* is an undirected graph in which any two vertices are connected by a unique path [66, p. 27]. A *rooted tree* is a tree in which one vertex has been designated the root [61, p. 13]. A *directed rooted tree* is a rooted tree where the edges are assigned a natural orientation, either away from or towards the root [66, p. 29]. Algorithm A.1 traverses a *directed rooted tree*. Here the root of τ is G^P without edges (or a graph with all the ports, see Sec. 2.1) and every vertex in τ represents some undirected labeled graph. The edges in τ are naturally directed away from the root because the algorithm produces new graphs in this direction by adding edges. Each directed edge (parent \rightarrow child) in τ represents the addition of a single edge to the parent graph to create the child graph. The height of a vertex in a rooted tree is the length of the longest downward path to the vertex from the root. Then the height of τ is equal to half the number of ports (or the number of edges needed to create a perfect matching). Only the set of vertices in τ with maximal height comprise the graph structure space of interest.

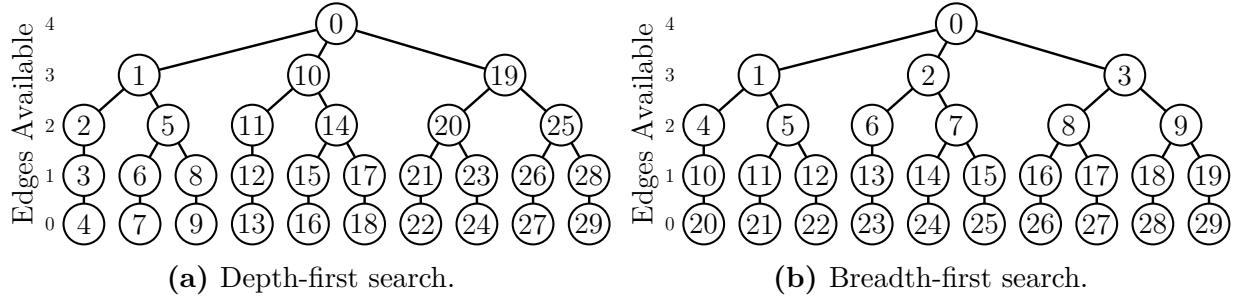


Figure A.8: Two tree traversal strategies (numbers indicate order the vertices are visited).

A.8.1 Depth-First vs. Breadth-First Search

Section 2.3 was titled *Tree Search Algorithm*, but we can be more descriptive of the particular algorithm implementation. There are two basic methods of tree traversal (the process of visiting each vertex in a directed rooted tree): depth-first search (DFS) or breadth-first search (BFS) [293, 294]. The primary difference the two methods is the order in which the vertices are explored [293]. DFS explores a particular path in the tree to the maximum height possible before backtracking and continuing down an alternative, unexplored path [294]. This process is visualized in Fig. A.8a. There are both stack-based and recursive

implementations of the DFS [293, pp. 169–172]. From these definitions, Alg. A.1 can be classified as a recursive DFS algorithm. While the current implementation works fairly well, the other tree traversal method, BFS, may be better suited for enumerating the graph structure space of interest.

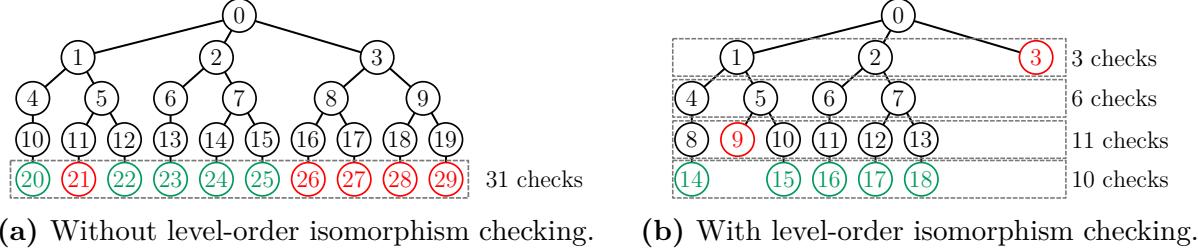


Figure A.9: Visualization of the impact of level-order isomorphism checking during the graph generation process for $(C, R, P) = (\{G, B\}, [1 \ 2], [2 \ 3])$.

A BFS method traverses the tree by visiting each vertex in a particular level first before moving to larger levels through the use of a queue [293, 294]. Levels are defined by sets of vertices with the same height. This process is visualized in Fig. A.8b and note the difference between DFS. The potential advantage of a BFS implementation would be the ability to include isomorphism checking at each level. In the current implementation, this is not possible so (potentially) many intermediate graphs that are isomorphic to other intermediate graphs are enumerated. By identifying isomorphic intermediate graphs, we can remove these vertices (and their subtree) from the tree traversal process. Thus, there would be a reduction the number of graphs generated while covering the same graph structure space. However, there is a considerable computational cost associated with checking if a set of graphs is isomorphic (see Sec. A.9), so there may be cases where the overall computational expense is larger with a BFS implementation with level-order isomorphism checking.

Consider the example in Fig. A.9 comparing the BFS method with and without level-order isomorphism checking. The desired set of unique graphs (colored green) is covered by both approaches, but different trees are traversed. There is a reduction from a total of 29 generated graphs to 18, but the number of graph comparisons needed only decreases from 31 to 30. With the overhead associated with calling the isomorphism checking function, level-order isomorphism checking may actually increase computation time. This is even more likely with the enhancements included because some of the vertices would be removed faster through the enhancements rather than direct isomorphism checking. It is future work to both implement this enhancement and determine its impact on the overall computational expense.

A.8.2 Parallelized Tree Traversal

We can further leverage our knowledge of the tree structure by parallelizing the traversal process. There has been considerable work in parallelizing various graph algorithms [295, 296]. The enhancement in Sec. A.7 for enumerating subcatalogs was a type of parallelized tree traversal, but it is more or less unpredictable in how it partitions the original tree (which was acceptable since it covers the same graph structure space). There are additional parallel traversal strategies that could be implemented in conjunction with the other enhancements such as the one below.

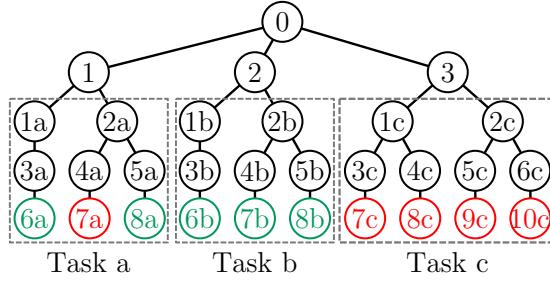


Figure A.10: Parallelization example.

Consider a tree with N levels. We can proceed as normal on a single worker using the BFS method up to level $n \leq N$. At this point, the task of traversing the subtree of each vertex in level n is a parallelizable task (assuming no level-order isomorphism checking). Level-order isomorphism checking could still be utilized in each of the tasks, but the collected graphs from each of the tasks would still need to be checked for uniqueness. An example of this approach is shown in Fig. A.10 with three tasks and assuming perfect parallelization of the generation task, a reduction from 29 to 13 effective algorithm calls (3 plus the maximum of the tasks).

A.9 Case Studies from Chapter 2

In Figs. A.15–A.17, the results from the case studies in Chapter 2 are compared with the enhancements in this chapter. All enhancements are present in the comparisons. No parallel computing was used in Case Study 1/2 and in Case Study 3, 12 threads were used for any parallel computing tasks. The PYTHON isomorphism checking method was used.

	Example 1			Example 2		
	Orig	En	Orig/En	Orig	En	Orig/En
Candidate Graphs	86	41	2.10	—	—	—
Feasible Graphs	77	39	1.97	23	11	2.09
Unique Graphs	16	16	1	5	5	1
Generation Time (s)	0.010	0.006	1.67	0.013	0.009	1.44
Total Time (s)	0.039	0.022	1.77	0.189	0.176	1.07

Table A.15: Comparison (Case Study 1).

(a) Examples 1 and 2.

	Example 1			Example 2		
	Orig	En	Orig/En	Orig	En	Orig/En
Candidate Graphs	1119	633	1.77	—	—	—
Feasible Graphs	767	442	1.74	767	212	3.62
Unique Graphs	274	274	1	140	140	1
Generation Time (s)	0.082	0.051	1.61	0.111	0.107	1.04
Total Time (s)	1.514	1.211	1.25	0.352	0.322	1.09

(b) Examples 3 and 4.

	Example 1			Example 2		
	Orig	En	Orig/En	Orig	En	Orig/En
Feasible Graphs	31	22	1.41	34	25	1.36
Unique Graphs	12	12	1	14	14	1
Generation Time (s)	0.137	0.008	17.13	0.134	0.009	14.89
Total Time (s)	0.156	0.024	6.50	0.153	0.027	5.67

Table A.16: Comparison (Case Study 2).

(a) Formulations changes with enhancements.

$$R_{\min} = [1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0], \quad R_{\max} = [1 \ 1 \ 2 \ 2 \ 2 \ 1 \ 2 \ 2]$$

$$S_8(1) = [1, 7, 2], \quad S_8(2) = [2, 7, 1], \quad S_8(3) = [1, 8, 2], \quad S_8(4) = [2, 8, 1]$$

(b) Results.

	Orig	En	Orig/En
Feasible Graphs	1943862	48408	40.16
Unique Graphs	13727	13774	0.997
Generation Time (s)	10872.7	251.8	43.18
Total Time (s)	17903.2	688.1	26.02

Table A.17: Comparison (Case Study 3).

Appendix B

Additional Architectures/Graphs

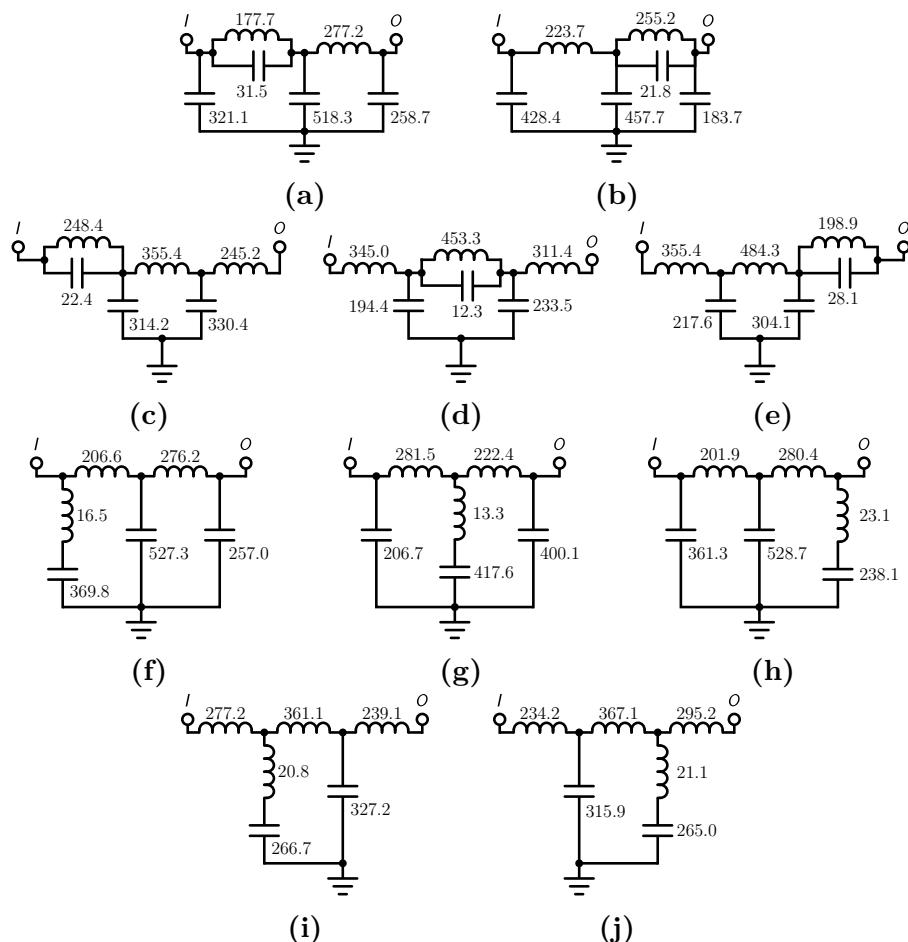


Figure B.1: All ten minimum complexity topologies for [Low-Pass Filter Realizability](#) task #3 (units are mH and nF).



Figure B.2: All 274 graphs in [Case Study 2](#) with no additional NSCs (gray hash indicates a multiedge).

Appendix C

Additional Material for Chapter 5²⁸

C.1 Algorithms in the Automated Problem Generation Procedure

Some notation used in the algorithms is shown in Table C.1.

Notation	Description
n_X	number total QP variables, i.e., $N_t(n_u + n_\xi) + n_p$
\emptyset	empty sequence
\oplus, \ominus, \odot	elementwise summation, difference, and product
1 to n	sequence defined by $[1, 2, \dots, n]$ where n is an integer
$S\langle i \rangle$	i th element of the sequence S
$ S $	number of elements in S (length or cardinality)
$\text{BLKDIAG}(A, B, \dots)$	construct block diagonal matrix from input arguments [297]
$\text{COMBINE}(S, R)$	concatenation of sequences S and R
$\text{EYE}(n)$	$n \times n$ identity matrix
$\text{KRON}(A, B)$	Kronecker tensor product of matrices A and B [182]
$\text{ONES}(n_1, n_2)$	$n_1 \times n_2$ matrix of ones
$\text{REPMAT}(A, n_1, n_2)$	repeat copies of A with n_1 row-wise and n_2 column-wise copies
$\text{RSHIFT}(S)$	circularly shifts the elements in S by 1 position to the right
$\text{SPARSE}(i, j, v, n, m)$	sparse matrix $A(i, j) = v$ with size n by m
$\text{ZEROS}(n_1, n_2)$	$n_1 \times n_2$ matrix of zeros

Table C.1: Notation used in the algorithms.

²⁸Elements of this chapter are based on work completed in Refs. [109].

Algorithm C.1: Optimization variable index generating functions (both continuous and discrete problems).

```

1 function GETCONTINDEX(xtype) /* index sequence of variable locations in continuous problem */
2   switch xtype do
3     case 0 or empty do
4       | X ← 0                                     /* singleton dimension */
5     case 1 do
6       | X ← 1 to  $n_u$                            /* controls */
7     case 2 or 4 or 5 do
8       | X ←  $n_u + 1$  to  $n_u + n_\xi$            /* states, initial states, and final states */
9     case 3 do
10      | X ←  $n_u + n_\xi + 1$  to  $n_u + n_\xi + n_p$  /* parameters */
11    end
12  end
13  return X                                     /* sequence of variable locations */
14 end
15 function GETQPINDEX(x, xtype, idx)           /* index sequence of variable locations in QP */
16   switch xtype do
17     case 0 or empty do
18       | l ← ONES(1,  $N_t$ )                     /* singleton dimension */
19     case 1 or 2 do
20       | l ← ( $x - 1$ )  $N_t + 1$  to  $xN_t$         /* states or controls */
21     case 3 do
22       | l ← ( $x + n_u n_t + n_\xi n_t$ ) ONES(1,  $N_t$ ) /* parameters */
23     case 4 do
24       | l ← (( $x - 1$ )  $N_t + 1$ ) ONES(1,  $N_t$ ) /* initial states */
25     case 5 do
26       | l ←  $xN_t$  ONES(1,  $N_t$ )             /* final states */
27     end
28   end
29   l ← l(idx)                                /* extract necessary indices */
30   return l                                     /* sequence of variable locations */
31 end

```

Algorithm C.2: Create Hessian.

Require: structures \mathcal{L} and \mathcal{M} of type objective

```

1 [IL, JL, VL] ← Algorithm C.3 using  $\mathcal{L}$                                /* Lagrange term sequences */
2 [IM, JM, VM] ← Algorithm C.4 using  $\mathcal{M}$                          /* Mayer term sequences */
3 l ← COMBINE(IL, IM)
4 J ← COMBINE(JL, JM)
5 V ← COMBINE(VL, VM)
6 H ← SPARSE(l, J, V,  $n_X$ ,  $n_X$ )                                     /* sparse matrix with  $n_X$  by  $n_X$  size */
7 H ← H + HT                                         /* make symmetric for  $\mathbf{X}^T \mathbf{H} \mathbf{X}/2$  form */
8 return H                                              /* Hessian */

```

Algorithm C.3: Create sequences for Lagrange terms.

Require: structure \mathcal{L} of type objective

```

1  $\emptyset \leftarrow I, J, H, Q, Qmid$                                 /* initialize sequences */
2 for  $k \leftarrow 1$  to  $|\mathcal{L}|$  do
3    $L \leftarrow \mathcal{L}(k)$                                          /* obtain current substructure */
4    $A \leftarrow L.\text{matrix}$                                        /* obtain current submatrix */
5    $R \leftarrow \text{GETCONTINDEX}(L.\text{left})$                       /* rows (continuous), see Alg. C.1 */
6    $C \leftarrow \text{GETCONTINDEX}(L.\text{right})$                      /* columns (continuous), see Alg. C.1 */
7   for  $i \leftarrow 1$  to  $|R|$  do
8     for  $j \leftarrow 1$  to  $|C|$  do
9        $r \leftarrow \text{GETQPINDEX}(R(i), L.\text{left}, 1 \text{ to } N_t)$       /* rows (QP), see Alg. C.1 */
10       $c \leftarrow \text{GETQPINDEX}(C(j), L.\text{right}, 1 \text{ to } N_t)$         /* columns (QP), see Alg. C.1 */
11       $I \leftarrow \text{COMBINE}(I, r)$                                          /* rows in main diagonal */
12       $J \leftarrow \text{COMBINE}(J, c)$                                          /* columns in main diagonal */
13       $H \leftarrow \text{COMBINE}(H, \Delta, 0)$                                      /* vector of time steps */
14       $Q \leftarrow \text{COMBINE}(Q, \text{evaluate } A(i,j) \text{ on } t)$ 
15       $Qmid \leftarrow \text{COMBINE}(Qmid, \text{evaluate } A(i,j) \text{ on } \bar{t}, 0)$ 
16    end
17  end
18 end
19  $IU \leftarrow \text{remove every } N_t \text{ element from } I \text{ starting with element } N_t$       /* rows in upper diagonal */
20  $IL \leftarrow \text{remove every } N_t \text{ element from } I \text{ starting with element } 1$           /* rows in lower diagonal */
21  $JU \leftarrow \text{remove every } N_t \text{ element from } J \text{ starting with element } 1$           /* columns in upper diagonal */
22  $JL \leftarrow \text{remove every } N_t \text{ element from } J \text{ starting with element } N_t$         /* columns in lower diagonal */
23  $Hoff \leftarrow \text{remove every } N_t \text{ element from } H \text{ starting with element } N_t$       /* remove the added zeros */
24  $Qoff \leftarrow \text{remove every } N_t \text{ element from } Qmid \text{ starting with element } N_t$     /* remove the added zeros */
25  $V \leftarrow (H \oplus \text{RSHIFT}(H)) \odot Q/6 \oplus (H \odot Qbar/6) \oplus \text{RSHIFT}(H \odot Qbar/6)$ 
26  $Voff \leftarrow Hoff \odot Qoff/6$ 
27  $\mathcal{I} \leftarrow \text{COMBINE}(\mathcal{I}, I, IU, IL)$ 
28  $\mathcal{J} \leftarrow \text{COMBINE}(\mathcal{J}, J, JU, JL)$ 
29  $\mathcal{V} \leftarrow \text{COMBINE}(\mathcal{V}, V, Voff, Voff)$ 
30 return  $\mathcal{I}, \mathcal{J}, \mathcal{V}$                                               /* sequences that define Lagrange terms in H */

```

method	V (line 25)	$Voff$ (line 26)	
/* CEF CTR G & CC	$H \odot Q$ $(H \oplus \text{RSHIFT}(H)) \odot Q/2$ $(\Delta/2)W \odot Q$	\emptyset \emptyset \emptyset	*/

/* replace line 13 with $W \leftarrow \text{COMBINE}W, w$ (see Eqn. (5.27)) */

Algorithm C.4: Create sequences for Mayer terms.

Require: structure \mathcal{M} of type objective

```
1  $\emptyset \leftarrow \mathcal{I}, \mathcal{J}, \mathcal{V}$                                 /* initialize sequences */
2 for  $k \leftarrow 1$  to  $|\mathcal{M}|$  do
3    $M \leftarrow \mathcal{M}\langle k \rangle$                                 /* obtain current substructure */
4    $A \leftarrow M.\text{matrix}$                                      /* obtain current submatrix */
5    $R \leftarrow \text{GETCONTINDEX}(M.\text{left})$                   /* rows (continuous), see Alg. C.1 */
6    $C \leftarrow \text{GETCONTINDEX}(M.\text{right})$                  /* columns (continuous), see Alg. C.1 */
7   for  $i \leftarrow 1$  to  $|R|$  do
8     for  $j \leftarrow 1$  to  $|C|$  do
9        $r \leftarrow \text{GETQPIDEX}(R\langle i \rangle, M.\text{left}, 1)$     /* rows (QP), see Alg. C.1 */
10       $c \leftarrow \text{GETQPIDEX}(C\langle j \rangle, M.\text{right}, 1)$   /* columns (QP), see Alg. C.1 */
11       $\mathcal{I} \leftarrow \text{COMBINE}(\mathcal{I}, r)$ 
12       $\mathcal{J} \leftarrow \text{COMBINE}(\mathcal{J}, c)$ 
13       $\mathcal{V} \leftarrow \text{COMBINE}(\mathcal{V}, A\langle i, j \rangle)$ 
14    end
15  end
16 end
17 return  $\mathcal{I}, \mathcal{J}, \mathcal{V}$                                 /* sequences that define Mayer terms in  $H$  */
```

Algorithm C.5: Create \mathbf{A}_{e1} and \mathbf{B}_{e1} matrices for the defect constraints using a single-step method.

```

Require:  $A, B, G$                                 /* matrices that define the dynamics in Eqn. (5.5) */
1  $K \leftarrow \text{KRON}(\text{EYE}(n_\xi), \text{ONES}(n_t, 1))$ 
2  $\emptyset \leftarrow \mathcal{I}, \mathcal{J}, \mathcal{V}$           /* initialize sequences */
3 for  $i \leftarrow 1$  to  $n_\xi$  do
4    $\text{DefectIdx} \leftarrow (i - 1)n_t + 1$  to  $int_t$            /* current defect constraint row locations */
5    $I \leftarrow \text{REPMAT}(\text{DefectIdx}, 1, n_u)$              /* current defect constraint row indices */
6    $J \leftarrow 1$  to  $n_u N_t$  but remove every  $N_t$           /* optimization variable column indices */
7    $T \leftarrow 1$  to  $n_u N_t$  but remove every  $N_t$           /* time indexing vector */
8    $H \leftarrow \text{REPMAT}(\Delta, n_u, 1)$                      /* vector of time steps */
9    $B \leftarrow \text{evaluate } B\langle i, : \rangle \text{ on } t$ 
10   $V3 \leftarrow -H/2 \odot B\langle T \rangle$                       /*  $\theta_1$  for the TR method */
11   $V4 \leftarrow -H/2 \odot B\langle T+1 \rangle$                    /*  $\theta_2$  for the TR method */
12   $\mathcal{I} \leftarrow \text{COMBINE}(\mathcal{I}, I, I)$ 
13   $\mathcal{J} \leftarrow \text{COMBINE}(\mathcal{J}, J, J+1)$ 
14   $\mathcal{V} \leftarrow \text{COMBINE}(\mathcal{V}, V3, V4)$ 
15   $I \leftarrow \text{REPMAT}(\text{DefectIdx}, 1, n_\xi)$              /* current defect constraint row indices */
16   $J \leftarrow n_u N_t + 1$  to  $(n_u + n_\xi)N_t$  but remove every  $N_t$  /* optimization variable column indices */
17   $T \leftarrow 1$  to  $n_\xi N_t$  but remove every  $N_t$           /* time indexing vector */
18   $H \leftarrow \text{REPMAT}(\Delta, n_\xi, 1)$                      /* vector of time steps */
19   $A \leftarrow \text{evaluate } A\langle i, : \rangle \text{ on } t$ 
20   $V1 \leftarrow -K\langle :, i \rangle \ominus H/2 \odot A\langle T \rangle$     /*  $\theta_3$  for the TR method */
21   $V2 \leftarrow K\langle :, i \rangle \ominus H/2 \odot A\langle T+1 \rangle$  /*  $\theta_4$  for the TR method */
22   $\mathcal{I} \leftarrow \text{COMBINE}(\mathcal{I}, I, I)$ 
23   $\mathcal{J} \leftarrow \text{COMBINE}(\mathcal{J}, J, J+1)$ 
24   $\mathcal{V} \leftarrow \text{COMBINE}(\mathcal{V}, V1, V2)$ 
25   $I \leftarrow \text{REPMAT}(\text{DefectIdx}, 1, n_p)$              /* current defect constraint row indices */
26   $J \leftarrow \text{KRON}(N_t(n_u + n_\xi) + (1 \text{ to } n_p), \text{ONES}(1, n_t))$  /* optimization variable column indices */
27   $T \leftarrow 1$  to  $n_p N_t$  but remove every  $N_t$           /* time indexing vector */
28   $H \leftarrow \text{REPMAT}(\Delta, n_\xi, 1)$                      /* vector of time steps */
29   $G \leftarrow \text{evaluate } G\langle i, : \rangle \text{ on } t$ 
30   $V \leftarrow -H/2 \odot (G\langle T \rangle \oplus G\langle T+1 \rangle)$     /*  $\theta_5$  for the TR method */
31   $\mathcal{I} \leftarrow \text{COMBINE}(\mathcal{I}, I) \text{ and } \mathcal{J} \leftarrow \text{COMBINE}(\mathcal{J}, J) \text{ and } \mathcal{V} \leftarrow \text{COMBINE}(\mathcal{V}, V)$ 
32 end
33  $\mathbf{A}_{e1} \leftarrow \text{SPARSE}(\mathcal{I}, \mathcal{J}, \mathcal{V}, n_\xi n_t, n_X)$       /* sparse matrix */
34  $\emptyset \leftarrow \mathcal{I}, \mathcal{V}$           /* initialize sequences */
35 for  $i \leftarrow 1$  to  $n_\xi$  do
36    $I \leftarrow (i - 1)n_t + 1$  to  $int_t$            /* current defect constraint row indices */
37    $T \leftarrow 1$  to  $n_t$                          /* time indexing vector */
38    $d \leftarrow \text{evaluate } d\langle i \rangle \text{ on } t$ 
39    $V \leftarrow H/2 \odot (d\langle T \rangle \oplus d\langle T+1 \rangle)$     /*  $\nu$  for the TR method */
40    $\mathcal{I} \leftarrow \text{COMBINE}(\mathcal{I}, I) \text{ and } \mathcal{V} \leftarrow \text{COMBINE}(\mathcal{V}, V)$ 
41 end
42  $\mathbf{B}_{e1} \leftarrow \text{SPARSE}(\mathcal{I}, 1, \mathcal{V}, n_\xi n_t, 1)$       /* sparse matrix */
43 return  $\mathbf{A}_{e1}, \mathbf{B}_{e1}$           /* matrices */

```

Algorithm C.6: Create \mathbf{A}_{e1} and \mathbf{B}_{e1} matrices for the defect constraints using a pseudospectral method.

```

Require:  $A, B, G, D$            /* matrices that define the dynamics in Eqn. (5.5) and differentiation
   matrix in Eqn. (5.20) */
1  $\emptyset \leftarrow \mathcal{I}, \mathcal{J}, \mathcal{V}$                                 /* initialize sequences */
2 for  $i \leftarrow 1$  to  $n_\xi$  do
3    $\text{DefectIdx} \leftarrow (i - 1)N_t + 1$  to  $iN_t$           /* current defect constraint row locations */
   /* ▷ controls —————— */
4    $I \leftarrow \text{REPMAT}(\text{DefectIdx}, 1, n_u)$              /* current defect constraint row indices */
5    $J \leftarrow 1$  to  $n_u N_t$                                /* current optimization variable column indices */
6    $B \leftarrow \text{evaluate } B\langle i, : \rangle \text{ on } t$ 
7    $V \leftarrow -(\Delta/2)B$ 
8    $\mathcal{I} \leftarrow \text{COMBINE}(\mathcal{I}, I)$ 
9    $\mathcal{J} \leftarrow \text{COMBINE}(\mathcal{J}, J)$ 
10   $\mathcal{V} \leftarrow \text{COMBINE}(\mathcal{V}, V)$ 
    /* ▷ states —————— */
11   $I \leftarrow \text{REPMAT}(\text{DefectIdx}, 1, n_u)$              /* current defect constraint row indices */
12   $J \leftarrow n_u N_t + 1$  to  $(n_u + n_\xi)N_t$           /* current optimization variable column indices */
13   $A \leftarrow \text{evaluate } A\langle i, : \rangle \text{ on } t$ 
14   $V \leftarrow -(\Delta/2)A$ 
15   $\mathcal{I} \leftarrow \text{COMBINE}(\mathcal{I}, I)$ 
16   $\mathcal{J} \leftarrow \text{COMBINE}(\mathcal{J}, J)$ 
17   $\mathcal{V} \leftarrow \text{COMBINE}(\mathcal{V}, V)$ 
    /* ▷ parameters —————— */
18   $I \leftarrow \text{REPMAT}(\text{DefectIdx}, 1, n_p)$              /* current defect constraint row indices */
19   $J \leftarrow \text{KRON}(N_t(n_u + n_\xi) + (1 \text{ to } n_p), \text{ONES}(1, N_t))$       /* current optimization variable column
   indices */
20   $G \leftarrow \text{evaluate } G\langle i, : \rangle \text{ on } t$ 
21   $V \leftarrow -(\Delta/2)G$ 
22   $\mathcal{I} \leftarrow \text{COMBINE}(\mathcal{I}, I)$ 
23   $\mathcal{J} \leftarrow \text{COMBINE}(\mathcal{J}, J)$ 
24   $\mathcal{V} \leftarrow \text{COMBINE}(\mathcal{V}, V)$ 
25 end
26  $D \leftarrow \text{COMBINE}(\text{ZEROS}(n_\xi N_t, n_u N_t), \text{BLKDIAG}(D, \dots, D), \text{ZEROS}(n_\xi N_t, n_p))$  /*  $n_\xi$  copies of  $D$  to
   BLKDIAG */
27  $\mathbf{A}_{e1} \leftarrow D \oplus \text{SPARSE}(\mathcal{I}, \mathcal{J}, \mathcal{V}, n_\xi N_t, n_x)$            /* sparse matrix */
28  $\emptyset \leftarrow \mathcal{I}, \mathcal{V}$                                 /* initialize sequences */
29 for  $i \leftarrow 1$  to  $n_\xi$  do
30    $I \leftarrow (i - 1)N_t + 1$  to  $iN_t$           /* current defect constraint row indices */
31    $d \leftarrow \text{evaluate } d\langle i \rangle \text{ on } t$ 
32    $V \leftarrow (\Delta/2)d$                          /*  $\nu$  for the TR method */
33    $\mathcal{I} \leftarrow \text{COMBINE}(\mathcal{I}, I)$ 
34    $\mathcal{V} \leftarrow \text{COMBINE}(\mathcal{V}, V)$ 
35 end
36  $\mathbf{B}_{e1} \leftarrow \text{SPARSE}(\mathcal{I}, 1, \mathcal{V}, n_\xi N_t, 1)$            /* sparse matrix */
37 return  $\mathbf{A}_{e1}, \mathbf{B}_{e1}$                                 /* matrices */

```

Algorithm C.7: Create matrices for the additional inequality (or equality) constraints.

Require: structure \mathcal{Z} of type constraint

```

1  $\emptyset \leftarrow \mathcal{I}_A, \mathcal{I}_B, \mathcal{J}_A, \mathcal{V}_A, \mathcal{V}_B$                                 /* initialize sequences */
2 for  $i \leftarrow 1$  to  $|\mathcal{Z}|$  do
3    $Z \leftarrow \mathcal{Z}\langle i \rangle$                                                  /* obtain current substructure */
4    $\text{PathFlag} \leftarrow 0$                                          /* initialize as a boundary constraint */
5   for  $j \leftarrow 1$  to  $|\mathcal{Y}|$  do
6     if  $Z.\text{linear}\langle j \rangle.\text{right} < 3$  then
7       |  $\text{PathFlag} \leftarrow 0$                                          /* set as a path constraint */
8     else if  $Z.\text{linear}\langle j \rangle.\text{matrix is time varying}$  then
9       |  $\text{PathFlag} \leftarrow 0$                                          /* set as a path constraint */
10      end
11      if  $Z.b$  is time varying then
12        |  $\text{PathFlag} \leftarrow 0$                                          /* set as a path constraint */
13      end
14    end
15     $N \leftarrow |\mathcal{I}_B|$                                          /* current number of constraints */
16    if  $\text{PathFlag} = 1$  then
17      |  $[I, J, V, B] \leftarrow \text{Algorithm C.8 using } Z$           /* path constraint */
18      |  $\mathcal{I}_B \leftarrow \text{COMBINE}(\mathcal{I}_B, N + (1 \text{ to } N_t))$  /*  $N_t$  constraints */
19    else
20      |  $[I, J, V, B] \leftarrow \text{Algorithm C.9 using } Z$           /* boundary constraint */
21      |  $\mathcal{I}_B \leftarrow \text{COMBINE}(\mathcal{I}_B, N + 1)$                 /* single constraint */
22    end
23     $\mathcal{I}_A \leftarrow \text{COMBINE}(\mathcal{I}_A, I + N)$ 
24     $\mathcal{J}_A \leftarrow \text{COMBINE}(\mathcal{J}_A, J)$ 
25     $\mathcal{V}_A \leftarrow \text{COMBINE}(\mathcal{V}_A, V)$ 
26     $\mathcal{V}_B \leftarrow \text{COMBINE}(\mathcal{V}_B, B)$ 
27 end
28  $\mathbf{A}_i \leftarrow \text{SPARSE}(\mathcal{I}_A, \mathcal{J}_A, \mathcal{V}_A, |\mathcal{I}_B|, n_X)$           /* sparse matrix */
29  $\mathbf{B}_i \leftarrow \text{SPARSE}(\mathcal{I}_B, 1, \mathcal{V}_B, |\mathcal{I}_B|, 1)$                 /* sparse matrix */
30 return  $\mathbf{A}_i, \mathbf{B}_i$                                               /* matrices */

```

Algorithm C.8: Create sequences for path constraint terms.

Require: structure YZ of type constraint

```
1  $\emptyset \leftarrow \mathcal{I}, \mathcal{J}, \mathcal{V}$                                 /* initialize sequences */
2 for  $j \leftarrow 1$  to  $|YZ.linear|$  do
3    $A \leftarrow YZ.linear[j].matrix$                                /* obtain current submatrix */
4    $C \leftarrow GETCONTINDEX(YZ.linear[j].right)$                 /* columns (continuous), see Alg. C.1 */
5   for  $i \leftarrow 1$  to  $|C|$  do
6      $r \leftarrow 1$  to  $N_t$                                      /* add  $N_t$  rows for a path constraint */
7      $c \leftarrow GETQPIINDEX(C[i], YZ.linear[j].right, 1$  to  $N_t)$       /* columns (QP), see Alg. C.1 */
8      $\mathcal{I} \leftarrow COMBINE(\mathcal{I}, r)$ 
9      $\mathcal{J} \leftarrow COMBINE(\mathcal{J}, c)$ 
10     $\mathcal{V} \leftarrow COMBINE(\mathcal{V}, \text{evaluate } A[i] \text{ on } t)$ 
11  end
12 end
13  $\mathcal{B} \leftarrow \text{evaluate } YZ.b \text{ on } t$ 
14 return  $\mathcal{I}, \mathcal{J}, \mathcal{V}, \mathcal{B}$                                 /* sequences for path constraint terms */
```

Algorithm C.9: Create sequences for boundary constraint terms.

Require: structure YZ of type constraint

```
1  $\emptyset \leftarrow \mathcal{I}, \mathcal{V}$                                 /* initialize sequences */
2 for  $j \leftarrow 1$  to  $|YZ.linear|$  do
3    $A \leftarrow YZ.linear[j].matrix$                                /* obtain current submatrix */
4    $C \leftarrow GETCONTINDEX(YZ.linear[j].right)$                 /* columns (continuous), see Alg. C.1 */
5   for  $i \leftarrow 1$  to  $|C|$  do
6      $r \leftarrow 1$                                               /* add 1 row for a boundary constraint */
7      $c \leftarrow GETQPIINDEX(C[i], YZ.linear[j].right, 1)$       /* columns (QP), see Alg. C.1 */
8      $\mathcal{I} \leftarrow COMBINE(\mathcal{I}, r)$ 
9      $\mathcal{J} \leftarrow COMBINE(\mathcal{J}, c)$ 
10     $\mathcal{V} \leftarrow COMBINE(\mathcal{V}, A[i])$ 
11  end
12 end
13  $\mathcal{B} \leftarrow YZ.b$ 
14 return  $\mathcal{I}, \mathcal{J}, \mathcal{V}, \mathcal{B}$                                 /* sequences for boundary constraint terms */
```

C.2 Sparsity Patterns

In this section, a number of sparsity patterns (a visualization of the potential nonzero entries in a matrix) are shown for the QP matrices (see Fig. 5.2). The symbol \circ indicates a potential nonzero entry in the matrix and \bullet indices a potential nonzero entry if we only look at the specific term mentioned. In all the figures, $\{n_u, n_\xi, n_p, N_t\}$ are $\{1, 2, 2, 7\}$.

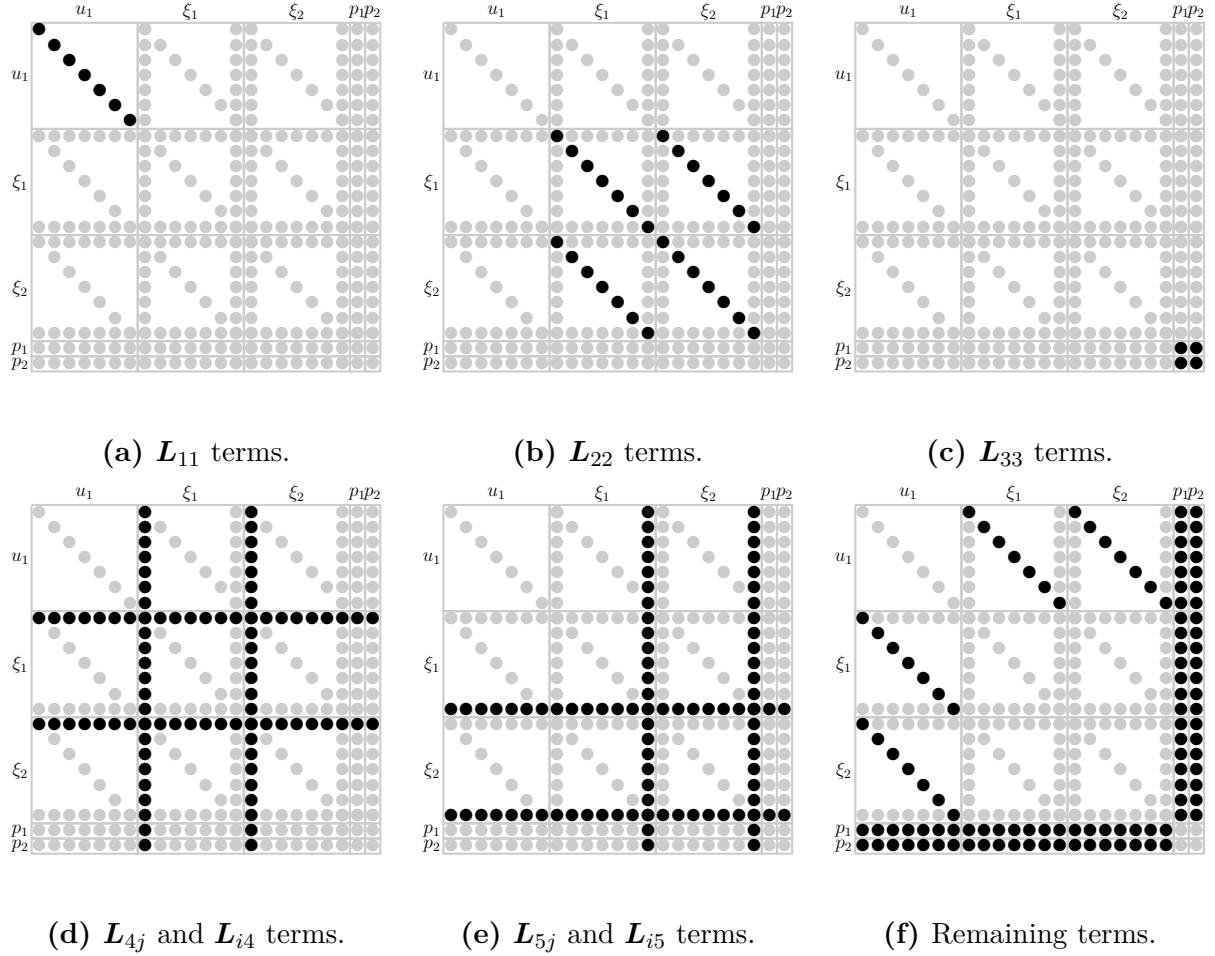


Figure C.1: Sparsity pattern of \mathbf{H} matrix for all considered methods except CQHS.

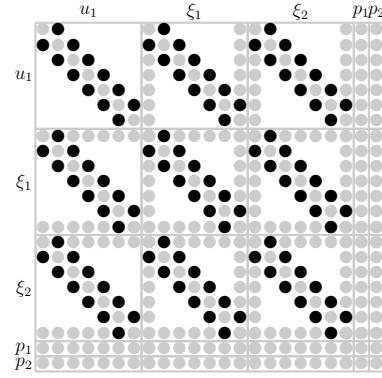


Figure C.2: Sparsity pattern of the off-diagonal terms of $\{\mathbf{L}_{11}, \mathbf{L}_{12}, \mathbf{L}_{21}, \mathbf{L}_{22}\}$ in the \mathbf{H} matrix using the CQHS method.

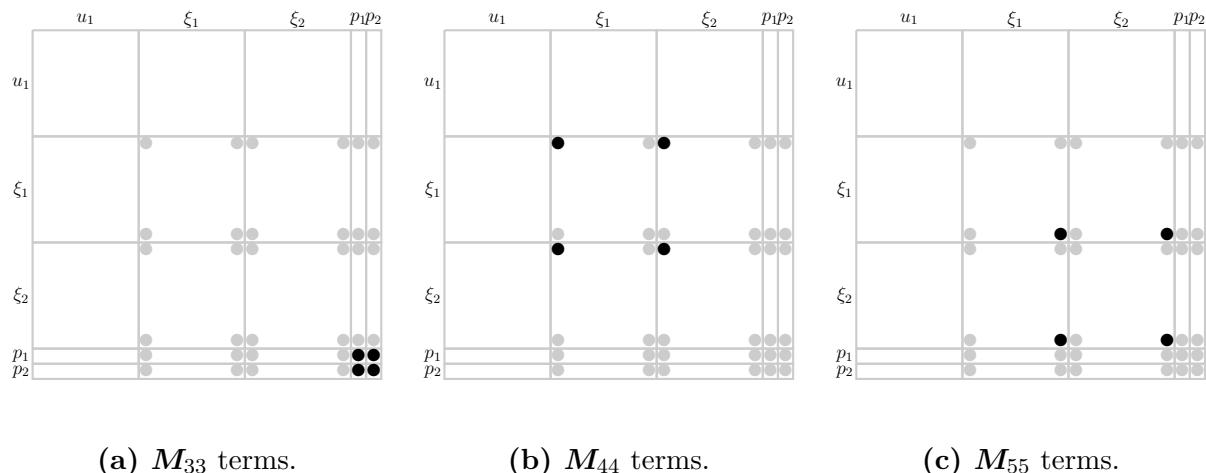


Figure C.3: Sparsity pattern of Mayer terms in \mathbf{H} matrix.

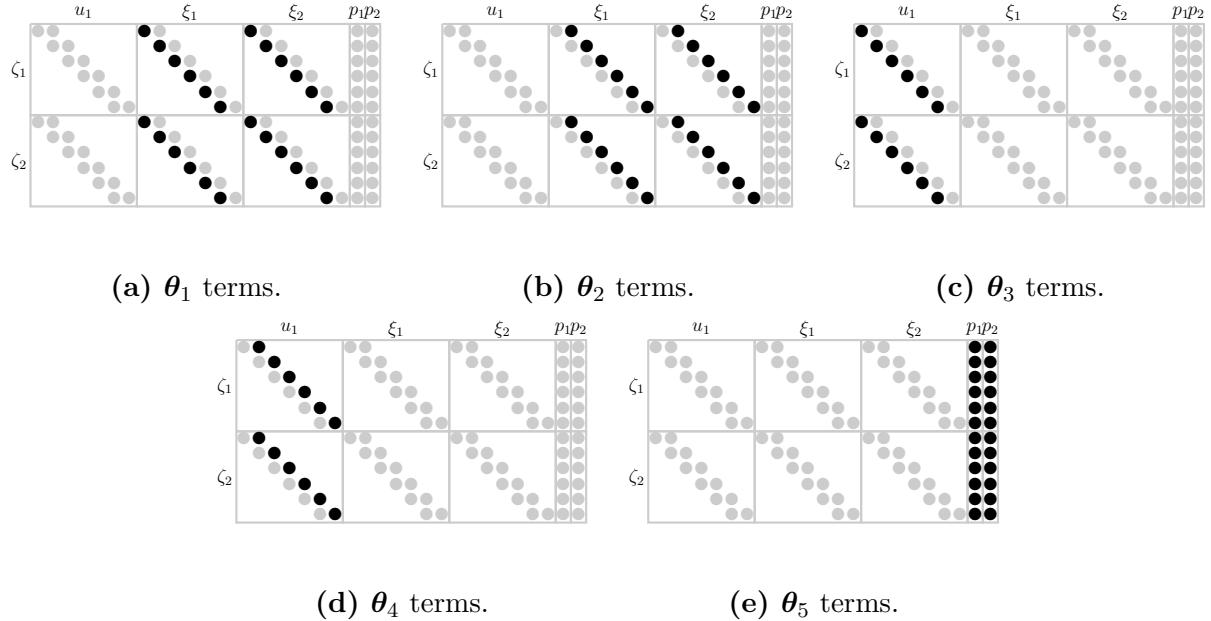


Figure C.4: Sparsity pattern of \mathbf{A}_{e1} matrix for the defect constraints using a single-step method.

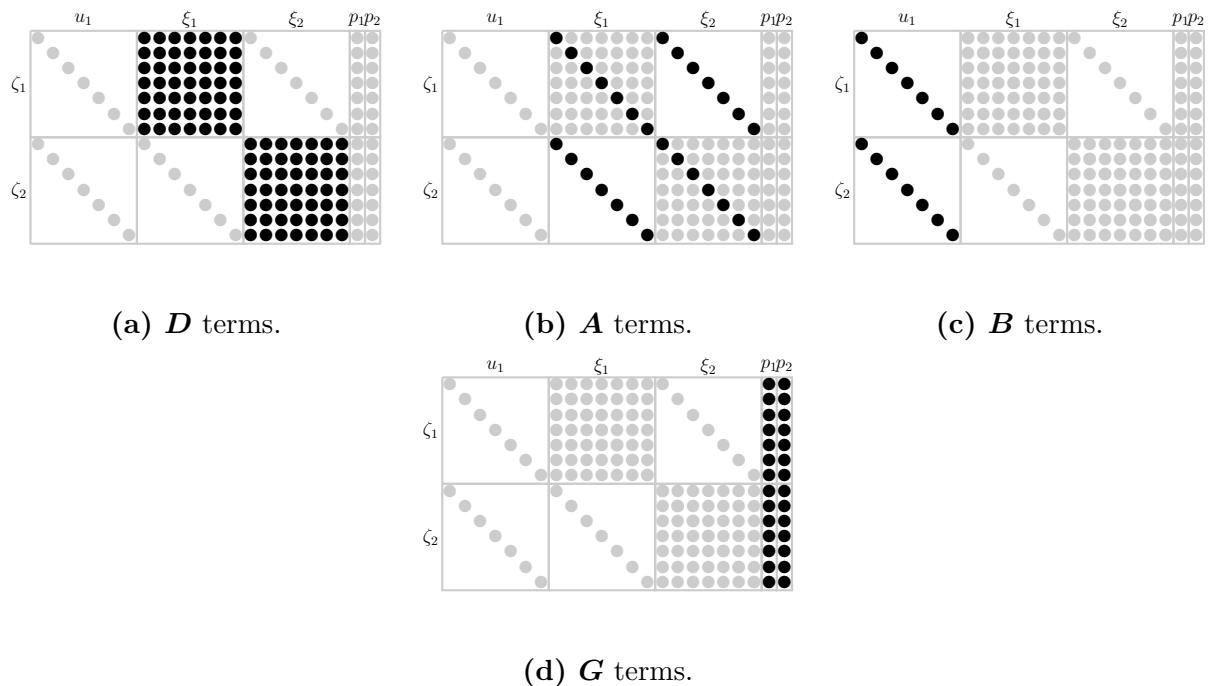


Figure C.5: Sparsity pattern of \mathbf{A}_{e1} matrix for the defect constraints using a pseudospectral method.

C.3 Codes

C.3.1 MATLAB Code for Example 1

```
% problem parameters
p.t0 = 0; p.tf = 20; % time horizon
p.x0 = -0.5; p.v0 = 1;
% system dynamics
A = [0 1;-1 0]; B = [0;1];
% Lagrange term
L(1).left = 1; L(1).right = 1; L(1).matrix = 1/2; % 1/2*u^2
% simple bounds
UB(1).right = 4; UB(1).matrix = [p.x0;p.v0]; % initial states
LB(1).right = 4; LB(1).matrix = [p.x0;p.v0];
UB(2).right = 5; UB(2).matrix = [0;0]; % final states
LB(2).right = 5; LB(2).matrix = [0;0];
% combine structures
setup.A = A; setup.B = B; setup.L = L; setup.UB = UB; setup.LB = LB; setup.p = p;
% solve
[T,U,Y,P,F,p,opts] = DTQP_solve(setup,[]);
```

C.3.2 MATLAB Code for Example 2

```
% problem parameters
p.t0 = 0; p.tf = 1; % time horizon
p.ell = 1/9;
% system dynamics
A = [0 1;0 0]; B = [0;1];
% Lagrange term
L(1).left = 1; L(1).right = 1; L(1).matrix = 1/2; % 1/2*u^2
% simple bounds
UB(1).right = 4; UB(1).matrix = [0;1]; % initial states
LB(1).right = 4; LB(1).matrix = [0;1];
UB(2).right = 5; UB(2).matrix = [0;-1]; % final states
LB(2).right = 5; LB(2).matrix = [0;-1];
UB(3).right = 2; UB(3).matrix = [p.ell;Inf]; % states
% combine structures
setup.A = A; setup.B = B; setup.L = L; setup.UB = UB; setup.LB = LB; setup.p = p;
% solve
[T,U,Y,P,F,p,opts] = DTQP_solve(setup,[]);
```

C.3.3 MATLAB Code for Example 3

```
% problem parameters
p.t0 = 0; p.tf = 1; % time horizon
p.x0 = 1; p.a = 2;
g = @(t) t.*cos(20*pi*t) - 1/4;
% system dynamics
A = 0; B{1,1} = g;
% Lagrange term
L(1).left = 1; L(1).right = 1; L(1).matrix = 1/2; % 1/2*u^2
% Mayer term
M(1).left = 5; M(1).right = 5; M(1).matrix = p.a^2/2; % a^2/2*x^2
% simple bounds
UB(1).right = 4; UB(1).matrix = p.x0; % initial state
LB(1).right = 4; LB(1).matrix = p.x0;
UB(2).right = 1; UB(2).matrix = 1; % control
LB(2).right = 1; LB(2).matrix = -1;
% combine structures
setup.A=A; setup.B=B; setup.L=L; setup.M=M; setup.UB=UB; setup.LB=LB; setup.p=p;
% solve
[T,U,Y,P,F,p,opts] = DTQP_solve(setup,[]);
```

C.3.4 MATLAB Code for Example 4

```
rng(393872382) % specific random seed
% problem parameters
p.ns = 20; p.nu = 10; % number of states and controls
p.t0 = 0; p.tf = 10; % time horizon
p.x0 = linspace(-5,5,p.ns)'; % initial states
% system dynamics
A = sprand(p.ns,p.ns,0.5,1); B = sprand(p.ns,p.nu,1,1);
% Lagrange term
Qi = sprand(p.ns,p.ns,0.2); Qi = (Qi*Qi')/100;
L(1).left = 1; L(1).right = 1; L(1).matrix = eye(p.nu); % u'*R*u
L(2).left = 2; L(2).right = 2; L(2).matrix = Qi; % x'*Q*x
% Mayer term
M(1).left = 5; M(1).right = 5; M(1).matrix = 10*eye(p.ns); % x^T*M*x
% initial states, simple bounds
UB(1).right = 4; UB(1).matrix = p.x0; % initial states
LB(1).right = 4; LB(1).matrix = p.x0;
% combine structures
setup.A=A; setup.B=B; setup.L=L; setup.M=M; setup.UB=UB; setup.LB=LB; setup.p=p;
% solve
[T,U,Y,P,F,p,opts] = DTQP_solve(setup,[]);
```

C.3.5 MATLAB Code for Example 5

```
% problem parameters
g = @(t) sin(2*pi*t) + 0.5;
p.t0 = 0; p.tf = 1; % time horizon
% system dynamics
A = [-1,2,0,0;3,-4,0,0;1,2,-1,0;1,0,0,0]; B = [1,0;-1,0,0,1/20;0,0]; G = zeros(4,1);
% Lagrange term
L(1).left = 1; L(1).right = 1; L(1).matrix = eye(2)/10; % u1^2 + u2^2
L(2).left = 1; L(2).right = 2; L(2).matrix = [1,1,0,0;0,0,0,0]; % u1*y1 + u1*y2
L(3).left = 2; L(3).right = 2; L(3).matrix = zeros(4); L(3).matrix(2,2) = 5; % 5*y2^2
L(4).left = 0; L(4).right = 2; L(4).matrix = {0,@(t) -5*2*g(t),0,0}; % -5*2*g*y2
L(5).left = 0; L(5).right = 0; L(5).matrix{1} = @(t) 5*(g(t)).^2; % 5*g^2
% Mayer term
M(1).left = 0; M(1).right = 3; M(1).matrix = 1; % p1
% y2(t0)-y2(tf) = 0, equality constraint
Y(1).linear(1).right = 4; Y(1).linear(1).matrix = [0;1;0;0]; % y2(t0)
Y(1).linear(2).right = 5; Y(1).linear(2).matrix = [0;-1;0;0]; % -y2(tf)
Y(1).b = 0;
% -y1 + u2/12 < 0, inequality constraint
Z(1).linear(1).right = 2; Z(1).linear(1).matrix = [-1;0;0;0]; % -y1
Z(1).linear(2).right = 1; Z(1).linear(2).matrix = [0;1/12]; % u2/12
Z(1).b = 0;
% y3 < p, inequality constraint
Z(2).linear(1).right = 2; Z(2).linear(1).matrix = [0;0;1;0]; % y3
Z(2).linear(2).right = 3; Z(2).linear(2).matrix = -1; % -p1
Z(2).b = 0;
% initial states, simple bounds
UB(1).right = 4; UB(1).matrix = [2;inf;0.5;0];
LB(1).right = 4; LB(1).matrix = [2;-inf;0.5;0];
% final states, simple bounds
UB(2).right = 5; UB(2).matrix = [inf;inf;inf;0];
LB(2).right = 5; LB(2).matrix = -[inf;inf;inf;0];
% abs(u2) < 10, simple bounds
UB(3).right = 1; UB(3).matrix = [inf;10];
LB(3).right = 1; LB(3).matrix = -[inf;10];
% y2 < g(t), simple bounds
UB(4).right = 2; UB(4).matrix= {inf;@(t) g(t);inf;inf};
% combine structures
setup.A = A; setup.B = B; setup.G = G; setup.L = L; setup.M = M;
setup.Y = Y; setup.Z = Z; setup.UB = UB; setup.LB = LB; setup.p = p;
% solve
[T,U,Y,P,F,p,opts] = DTQP_solve(setup,[]);
```

C.4 Methods

C.4.1 Pseudospectral Methods

This section outlines how to determine the nodes, differentiation matrix, and quadrature weights for use in pseudospectral methods.

C.4.1.1 Legendre Pseudospectral Method with LGL Nodes

Let $L_N(\tau)$ denote the Legendre polynomial of order N , which may be generated from:

$$L_N(\tau) = \frac{1}{2^N N!} \frac{d^N}{d\tau^N} (\tau^2 - 1)^N \quad (\text{C.1})$$

The Lagrange-Gauss-Lobatto (LGL) nodes are defined as:

$$\tau_k = \begin{cases} -1 & \text{if } k = 0 \\ \text{kth root of } \dot{L}_{N_t}(\tau) & \text{if } k = \{1, 2, \dots, N_t - 1\} \\ 1 & \text{if } k = N_t \end{cases} \quad (\text{C.2})$$

where $\dot{L}_N = \frac{dL_N}{d\tau}$. We note that the nodes are always between $[-1, 1]$ and contain both endpoints (Ref. [298] codes from Ref. [161]).

We define the basis polynomials needed in Eqn. (5.18) for the Legendre-based method as Lagrange basis polynomials:

$$\phi_k(\tau) = \prod_{i=0, i \neq k}^{N_t} \frac{\tau - \tau_i}{\tau_k - \tau_i} \quad (\text{C.3})$$

With LGL nodes, $\phi_k(\tau)$ can be written in the following alternative form [163]:

$$\phi_k(\tau) = \frac{1}{N_t(N_t + 1)L_{N_t}(\tau_k)} \frac{(\tau^2 - 1)\dot{L}_{N_t}(\tau)}{\tau - \tau_k} \quad (\text{C.4})$$

The differentiation matrix needed in Eqn. (5.20) for the Legendre-based method is:

$$D_{ki} = \begin{cases} \frac{L_{N_t}(\tau_k)}{L_{N_t}(\tau_i)} \frac{1}{\tau_k - \tau_i} & \text{if } k \neq i \\ N_t(N_t + 1)/4 & \text{if } k = i = 0 \\ -N_t(N_t + 1)/4 & \text{if } k = i = N_t \\ 0 & \text{otherwise} \end{cases} \quad (\text{C.5})$$

Further numerical enhancements can be made to improve stability in the presence of rounding errors (expression from Ref. [163], Ref. [298] codes from Ref. [161]).

The quadrature weights w_k needed in Eqn. (5.27) for the Legendre-based method are:

$$w_k = \frac{2}{N_t(N_t + 1)} \frac{1}{(L_{N_t}(\tau_k))^2}, \quad k = \{0, 1, \dots, N_t\} \quad (\text{C.6})$$

These are Gaussian quadrature weights that are exactly accurate for polynomials of degree up to degree $2N_t - 1$ (expression from Ref. [164], Ref. [298] codes from Ref. [161]).

C.4.1.2 Chebyshev Pseudospectral Method with CGL Nodes

Let $T_N(\tau)$ denote the Chebyshev polynomial of order N , which may be generated from:

$$T_N = \cos(N \cos^{-1}(\tau)) \quad (\text{C.7})$$

The Chebyshev-Gauss-Lobatto (CGL) nodes are defined as the roots of $\dot{T}_{N_t} = \frac{dT_{N_t}}{d\tau}$ and the additional endpoints. All CGL nodes can be computed conveniently by:

$$\tau_k = -\cos\left(\frac{\pi k}{N_t}\right) \quad k = \{0, 1, \dots, N_t\} \quad (\text{C.8})$$

We note that the nodes are always between $[-1, 1]$ and contain both endpoints.

We define the basis polynomials needed in Eqn. (5.18) for the Chebyshev-based method as Lagrange basis polynomials previously defined in Eqn. (C.3). With CGL nodes, $\phi_k(\tau)$ can be written in the following alternative form [162]:

$$\phi_k(\tau) = \frac{(-1)^{k+1}}{N_t^2 a_k} \frac{(1 - \tau^2) \dot{T}_{N_t}(\tau)}{\tau - \tau_k} \quad \text{where: } a_k = \begin{cases} 2 & \text{if } k = \{0, N_t\} \\ 1 & \text{otherwise} \end{cases} \quad (\text{C.9})$$

The differentiation matrix needed in Eqn. (5.20) for the Chebyshev-based method is:

$$D_{ki} = \begin{cases} \frac{a_k}{a_i} \frac{(-1)^{k+i}}{(\tau_k - \tau_i)} & \text{if } k \neq i \\ -\frac{\tau_k}{2(1 - \tau_k^2)} & \text{if } 1 \leq k = i \leq N_t - 1 \\ \frac{2N_t^2 + 1}{6} & \text{if } k = i = 0 \\ -\frac{2N_t^2 + 1}{6} & \text{if } k = i = N_t \end{cases} \quad (\text{C.10})$$

Further numerical enhancements can be made to improve stability in the presence of rounding errors (expression from Ref. [162], Ref. [298] codes from [299, p. 54]).

The quadrature weights w_k needed in Eqn. (5.27) for the Chebyshev-based method are:

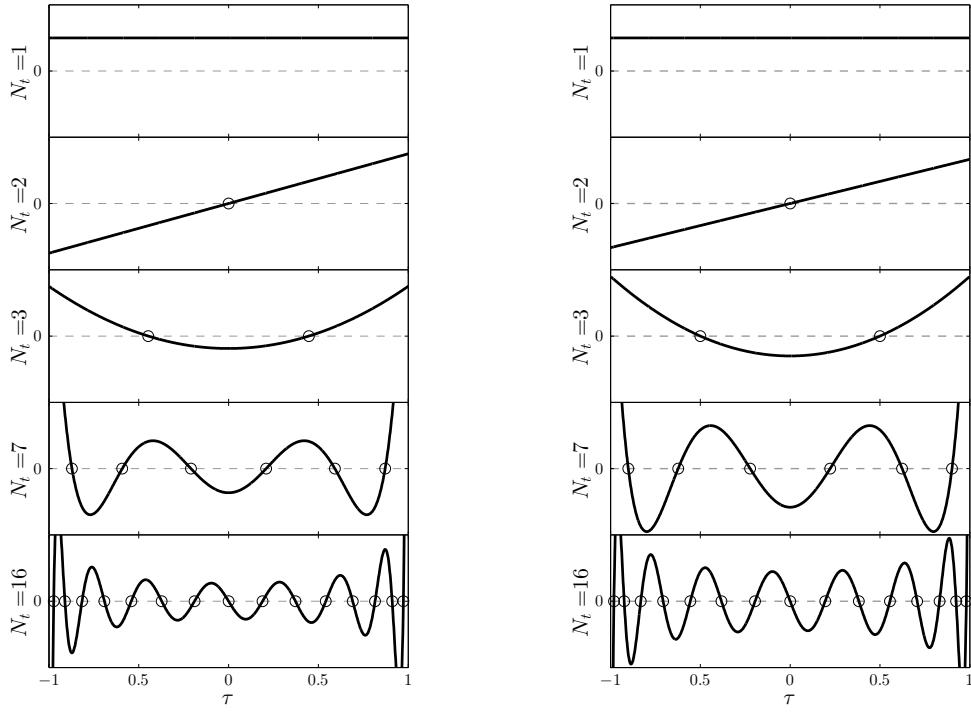
$$w_k = \frac{c_k}{N_t} \left(1 - \sum_{j=1}^{\lfloor N_t/2 \rfloor} \frac{b_j}{4j^2 - 1} \cos(2j\tau_k) \right) \quad (\text{C.11})$$

where: $b_j = \begin{cases} 1 & \text{if } j = N_t/2 \\ 2 & \text{if } j < N_t/2 \end{cases}$, $c_k = \begin{cases} 1 & \text{if } k = \{0, N_t\} \\ 2 & \text{otherwise} \end{cases}$

These are Clenshaw-Curtis quadrature weights that are exactly accurate for polynomials of degree up to degree N_t (expression from Ref. [300], Ref. [298] codes from Ref. [299, p. 128]).

C.4.1.3 Visualizations for Specific Pseudospectral Implementations

This section contains a number of visualizations devoted to explaining the various aspects of the specific pseudospectral methods outlined in Secs. C.4.1.1 and C.4.1.2. Many of the figures are similar to the ones found in Refs. [137, 163] so please refer to them for further analysis.



(a) Inner LGL nodes based on $\dot{L}_{N_t}(\tau)$. (b) Inner CGL nodes based on $\dot{T}_{N_t}(\tau)$.

Figure C.6: LGL and CGL inner node locations from the roots of polynomials.

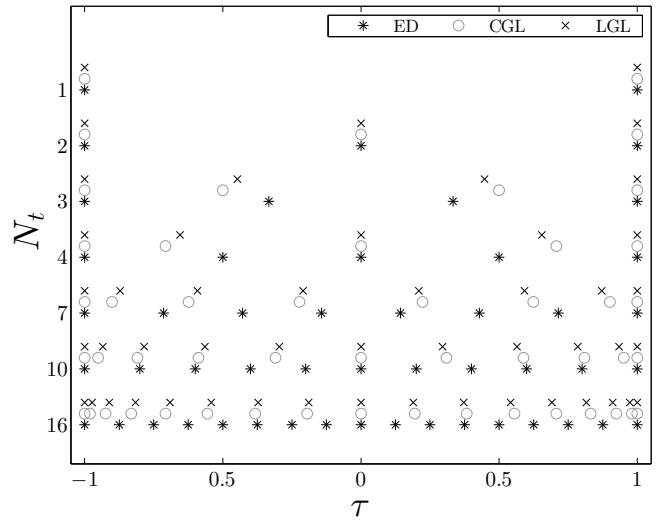


Figure C.7: ED, CGL, and LGL nodes for various values of N_t .

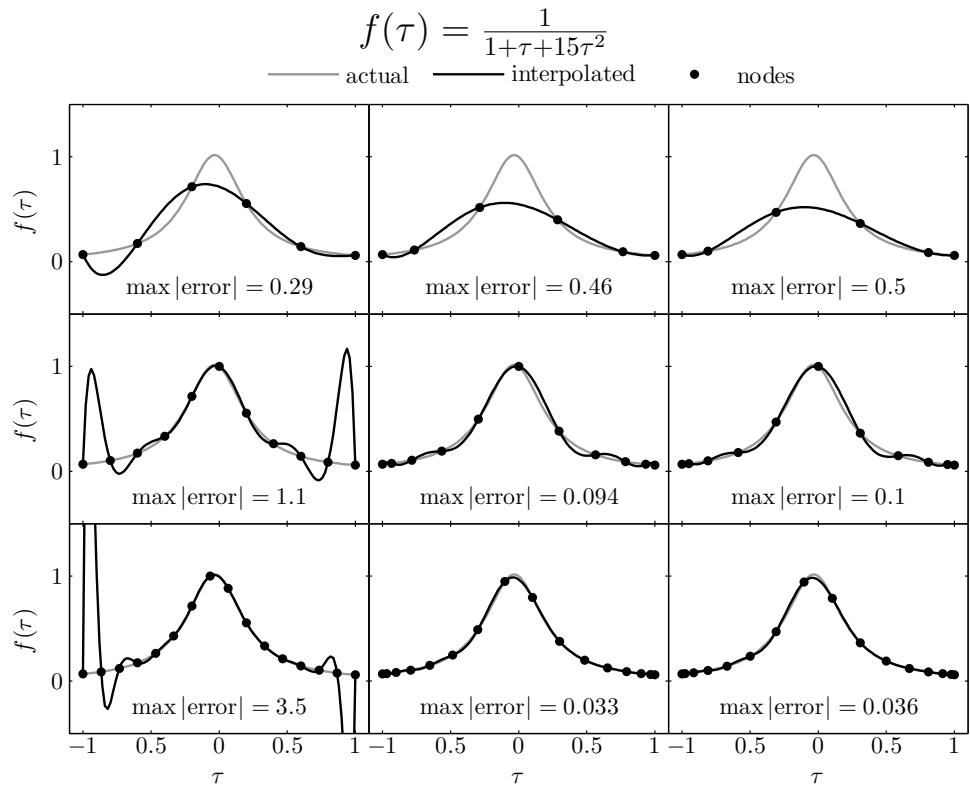
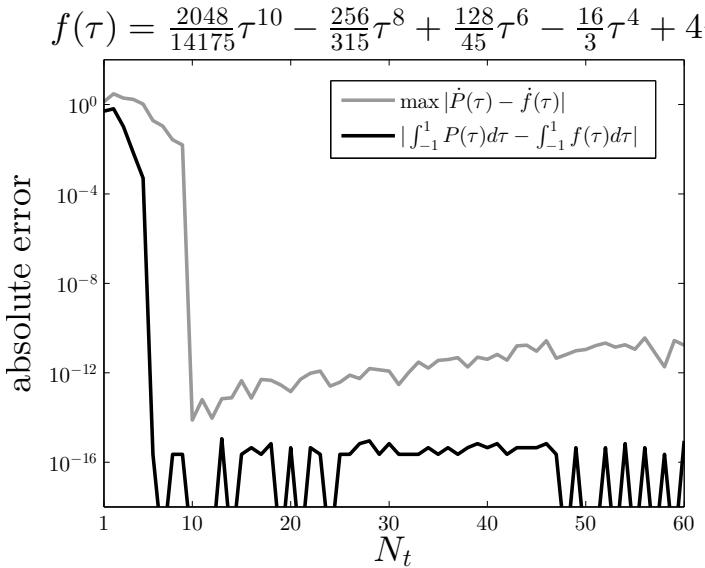
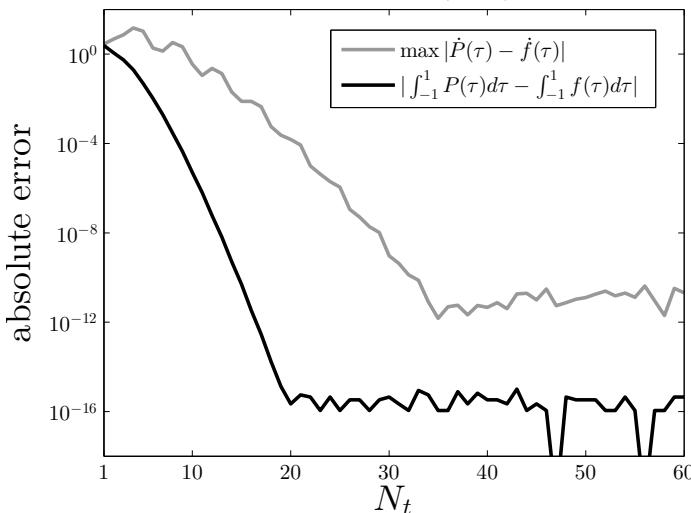


Figure C.8: Lagrange polynomial interpolation with ED, LGL, and CGL nodes for various values of N_t .



(a) Polynomial example.

$$f(\tau) = \sin(5\tau^2)$$



(b) Nonpolynomial example.

Figure C.9: Convergence behavior for definite integral and derivative approximations using Lagrange interpolation with LGL nodes.

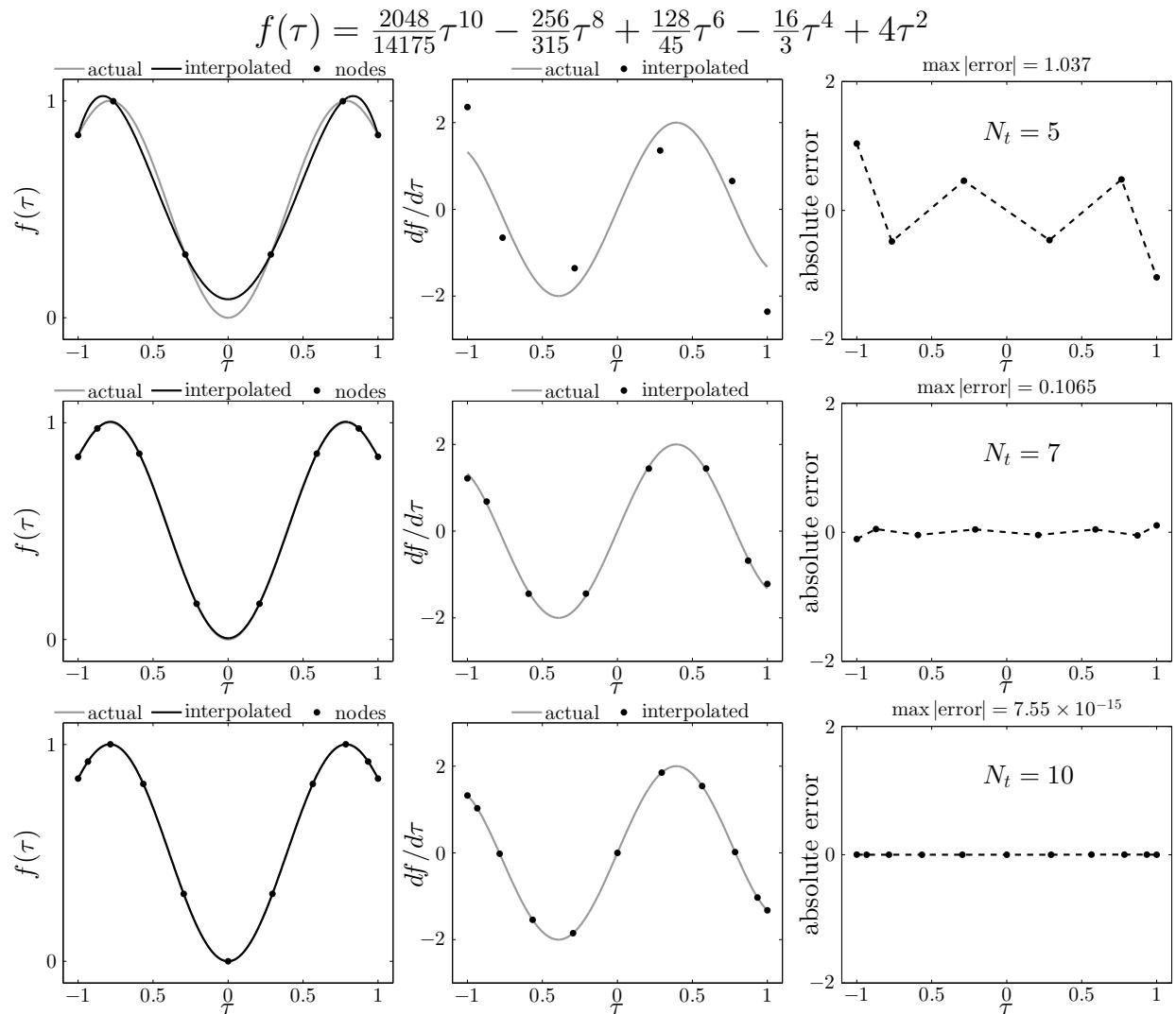


Figure C.10: Differentiation error using Lagrange interpolation with LGL nodes for various values of N_t .

C.4.2 Single-Step Methods

Here we provide the general forms of the single-step (SS) methods in Sec. 5.3.2.2.

C.4.2.1 Euler Forward

The Euler forward method is an explicit first-order method:

$$\zeta(t_k) = \xi(t_{k+1}) - \xi(t_k) - \Delta_k \mathbf{f}(t_k) \quad (\text{C.12})$$

C.4.2.2 Trapezoidal Rule

The trapezoidal rule is an implicit second-order method:

$$\zeta(t_k) = \xi(t_{k+1}) - \xi(t_k) - \frac{\Delta_k}{2} (\mathbf{f}(t_k) + \mathbf{f}(t_{k+1})) \quad (\text{C.13})$$

C.4.2.3 Hermite-Simpson

The Hermite-Simpson is an implicit third-order method:

$$\zeta(t_k) = \xi(t_{k+1}) - \xi(t_k) - \frac{\Delta_k}{6} (\mathbf{f}(t_k) + 4\mathbf{f}(\bar{t}_k) + \mathbf{f}(t_{k+1})) \quad (\text{C.14a})$$

$$\mathbf{f}(\bar{t}_k) = \mathbf{f}(\bar{t}_k, \xi(\bar{t}_k), \mathbf{u}(\bar{t}_k), \mathbf{p}) \quad (\text{C.14b})$$

$$\xi(\bar{t}_k) = \frac{1}{2} (\xi(t_{k+1}) + \xi(t_k)) + \frac{\Delta_k}{8} (\mathbf{f}(t_k) - \mathbf{f}(t_{k+1})) \quad (\text{C.14c})$$

$$\mathbf{u}(\bar{t}_k) = \frac{1}{2} (\mathbf{u}(t_{k+1}) + \mathbf{u}(t_k)) \quad (\text{C.14d})$$

(*Derivation of Defect Constraints in LQDO*) Recall that the state dynamics in LQDO is a linear nonhomogeneous differential equation:

$$\mathbf{f}(t_k) = \mathbf{A}_k \xi_k + \mathbf{B}_k \mathbf{u}_k + \mathbf{G}_k \mathbf{p} + \mathbf{d}_k \quad (\text{C.15})$$

The intermediate state value using Eqn. (C.14c) is approximated as:

$$\begin{aligned} \xi(\bar{t}_k) &= \frac{1}{2} (\xi_{k+1} + \xi_k) + \frac{\Delta_k}{8} (\mathbf{A}_k \xi_k + \mathbf{B}_k \mathbf{u}_k + \mathbf{G}_k \mathbf{p} + \mathbf{d}_k + \dots \\ &\quad - \mathbf{A}_{k+1} \xi_{k+1} - \mathbf{B}_{k+1} \mathbf{u}_{k+1} - \mathbf{G}_{k+1} \mathbf{p} - \mathbf{d}_{k+1}) \end{aligned} \quad (\text{C.16})$$

The intermediate derivative function value in Eqn. (C.14b) is approximated with:

$$\mathbf{f}(\bar{t}_k) = \bar{\mathbf{A}}_k \boldsymbol{\xi}(\bar{t}_k) + \bar{\mathbf{B}}_k \mathbf{u}(\bar{t}_k) + \bar{\mathbf{G}}_k \mathbf{p} + \bar{\mathbf{d}}_k \quad (\text{C.17a})$$

$$\begin{aligned} &= \left(\frac{\bar{\mathbf{B}}_k}{2} + \frac{\Delta_k}{8} \bar{\mathbf{A}}_k \mathbf{B}_k \right) \mathbf{u}_k + \left(\frac{\bar{\mathbf{B}}_k}{2} - \frac{\Delta_k}{8} \bar{\mathbf{A}}_k \mathbf{B}_{k+1} \right) \mathbf{u}_{k+1} + \dots \\ &\quad \left(\frac{\bar{\mathbf{A}}_k}{2} + \frac{\Delta_k}{8} \bar{\mathbf{A}}_k \mathbf{A}_k \right) \boldsymbol{\xi}_k + \left(\frac{\bar{\mathbf{A}}_k}{2} - \frac{\Delta_k}{8} \bar{\mathbf{A}}_k \mathbf{A}_{k+1} \right) \boldsymbol{\xi}_{k+1} + \dots \\ &\quad \left(\bar{\mathbf{G}}_k + \frac{\Delta_k}{8} \bar{\mathbf{A}}_k \mathbf{G}_k - \frac{\Delta_k}{8} \bar{\mathbf{A}}_k \mathbf{G}_{k+1} \right) \mathbf{p} + \left(\bar{\mathbf{d}}_k + \frac{\Delta_k}{8} \bar{\mathbf{A}}_k \mathbf{d}_k - \frac{\Delta_k}{8} \bar{\mathbf{A}}_k \mathbf{d}_{k+1} \right) \end{aligned} \quad (\text{C.17b})$$

Combining this with the HS defect constraint formula in Eqn. (C.14a):

$$\zeta(t_k) = \boldsymbol{\xi}(t_{k+1}) - \boldsymbol{\xi}(t_k) - \frac{\Delta_k}{6} (\mathbf{f}(t_k) + 4\mathbf{f}(\bar{t}_k) + \mathbf{f}(t_{k+1})) \quad (\text{C.18a})$$

$$\begin{aligned} &= \left(-\frac{\Delta_k}{6} \mathbf{B}_k - \frac{\Delta_k}{3} \bar{\mathbf{B}}_k - \frac{\Delta_k^2}{12} \bar{\mathbf{A}}_k \mathbf{B}_k \right) \mathbf{u}_k + \left(-\frac{\Delta_k}{6} \mathbf{B}_{k+1} - \frac{\Delta_k}{3} \bar{\mathbf{B}}_k + \frac{\Delta_k^2}{12} \bar{\mathbf{A}}_k \mathbf{B}_{k+1} \right) \mathbf{u}_{k+1} + \dots \\ &\quad \left(-\mathbf{I} - \frac{\Delta_k}{3} \bar{\mathbf{A}}_k - \frac{\Delta_k^2}{12} \bar{\mathbf{A}}_k \mathbf{A}_k - \frac{\Delta_k}{6} \mathbf{A}_k \right) \boldsymbol{\xi}_k + \left(\mathbf{I} - \frac{\Delta_k}{3} \bar{\mathbf{A}}_k + \frac{\Delta_k^2}{12} \bar{\mathbf{A}}_k \mathbf{A}_{k+1} - \frac{\Delta_k}{6} \mathbf{A}_{k+1} \right) \boldsymbol{\xi}_{k+1} + \dots \\ &\quad \left(-\frac{\Delta_k}{6} \mathbf{G}_k - \frac{2\Delta_k}{3} \bar{\mathbf{G}}_k - \frac{\Delta_k^2}{12} \bar{\mathbf{A}}_k \mathbf{G}_k + \frac{\Delta_k^2}{12} \bar{\mathbf{A}}_k \mathbf{G}_{k+1} - \frac{\Delta_k}{6} \mathbf{G}_{k+1} \right) \mathbf{p} + \dots \\ &\quad \left(-\frac{\Delta_k}{6} \mathbf{d}_k - \frac{2\Delta_k}{3} \bar{\mathbf{d}}_k - \frac{\Delta_k^2}{12} \bar{\mathbf{A}}_k \mathbf{d}_k + \frac{\Delta_k^2}{12} \bar{\mathbf{A}}_k \mathbf{d}_{k+1} - \frac{\Delta_k}{6} \mathbf{d}_{k+1} \right) \end{aligned} \quad (\text{C.18b})$$

which is the same equation as the SS step formula in Eqn. (5.23) with the coefficients for the HS method in Eqn. (5.25c).

C.4.2.4 Classical 4th-Order Runge-Kutta

This is another popular explicit scheme given by:

$$\zeta(t_k) = \boldsymbol{\xi}(t_{k+1}) - \boldsymbol{\xi}(t_k) - \frac{\Delta_k}{6} (\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \quad (\text{C.19a})$$

$$\mathbf{k}_1 = \mathbf{f}(t_k) \quad (\text{C.19b})$$

$$\mathbf{k}_2 = \mathbf{f} \left(\bar{t}_k, \boldsymbol{\xi}(t_k) + \frac{\Delta_k}{2} \mathbf{k}_1, \mathbf{u}(\bar{t}_k), \mathbf{p} \right) \quad (\text{C.19c})$$

$$\mathbf{k}_3 = \mathbf{f} \left(\bar{t}_k, \boldsymbol{\xi}(t_k) + \frac{\Delta_k}{2} \mathbf{k}_2, \mathbf{u}(\bar{t}_k), \mathbf{p} \right) \quad (\text{C.19d})$$

$$\mathbf{k}_4 = \mathbf{f} (t_{k+1}, \boldsymbol{\xi}(t_k) + \Delta_k \mathbf{k}_3, \mathbf{u}(t_{k+1}), \mathbf{p}) \quad (\text{C.19e})$$

$$\mathbf{u}(\bar{t}_k) = \frac{1}{2} (\mathbf{u}(t_{k+1}) + \mathbf{u}(t_k)) \quad (\text{C.19f})$$

Appendix D

Summary of Available Code

Here we summarize the available code developed as a part of this dissertation. All code is primarily written in the MATLAB language.

- (PM ARCHITECTURES PROJECT) Ref.[65] contains the code for Chapter 2 and Appendix A. It generates the set of unique useful graphs with a perfect matching-based approach. Ref. [301] is used in Ref. [65] and contains a recursive algorithm for the $(n - 1)!!$ perfect matchings of K_n and incomplete listings for large n .
- (CO-DESIGN EXAMPLES REPOSITORY) Ref. [114] contains the code for the examples in Chapters 3, 4, and 7 including the SASA case study.
- (DT QP PROJECT) Ref. [186] contains the code for Chapter 5, the automated problem generation for linear-quadratic dynamic optimization using direct transcription and quadratic programming. Refs. [298, 302] are useful teaching aids for direct methods of optimal control.
- (PM CIRCUITS) Ref. [232] contains the code for Chapter 6, the passive analog circuits case study including the automated model generator for creating the transfer function given the graph and automated optimization problem generation for the sizing task.
- (PM SUSPENSIONS) Ref. [303] contains the code for Chapter 8 including the automated model generator for creating the linearized state-space system given the graph and automated optimization problem generation utilizing Ref. [186].

Bibliography

- [1] A. P. Deshmukh, D. R. Herber, and J. T. Allison, “Bridging the Gap between Open-Loop and Closed-Loop Control in Co-Design: A Framework for Complete Optimal Plant and Control Architecture Design,” in *American Control Conference*, Chicago, IL, USA, Jul. 2015, pp. 4916–4922. doi: [10.1109/ACC.2015.7172104](https://doi.org/10.1109/ACC.2015.7172104) (see pp. 1, 7, 8, 14, 166, 191, 208)
- [2] R. J. McCrary, “The Design Method in Practice,” in *The Design Method*, S. A. Gregory, Ed., 1st ed. Springer, 1966, pp. 11–18. doi: [10.1007/978-1-4899-6331-4_1](https://doi.org/10.1007/978-1-4899-6331-4_1) (see p. 1)
- [3] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*, 1st ed. Wiley, 2005, isbn: 978-0471649908 (see pp. 1, 50, 51)
- [4] J. T. Allison, “Plant-Limited Co-Design of an Energy-Efficient Counterbalanced Robotic Manipulator,” *Journal of Mechanical Design*, vol. 135, no. 10, p. 101 003, Aug. 2013. doi: [10.1115/1.4024978](https://doi.org/10.1115/1.4024978) (see pp. 3, 5, 6, 8, 44, 45, 50, 51, 61, 165, 184, 186)
- [5] E. Crawley, O. d. Weck, S. Eppinger, *et al.*, “The Influence of Architecture in Engineering Systems,” Massachusetts Institute of Technology, Engineering Systems Monograph, Mar. 2004 (see pp. 4, 14)
- [6] D. R. Herber, T. Guo, and J. T. Allison, “Enumeration of Architectures with Perfect Matchings,” *Journal of Mechanical Design*, vol. 139, no. 5, p. 051 403, May 2017. doi: [10.1115/1.4036132](https://doi.org/10.1115/1.4036132) (see pp. 4, 5, 14, 142, 143, 162, 201, 212, 230)
- [7] M. P. Bendsøe and O. Sigmund, *Topology Optimization*. Springer, 2004, isbn: 978-3642076985. doi: [10.1007/978-3-662-05086-6](https://doi.org/10.1007/978-3-662-05086-6) (see pp. 5, 7, 10)
- [8] D. J. Lohan, E. M. Dede, and J. T. Allison, “Topology Optimization for Heat Conduction Using Generative Design Algorithms,” *Structural and Multidisciplinary Optimization*, vol. 55, no. 3, pp. 1063–1077, Aug. 2016. doi: [10.1007/s00158-016-1563-6](https://doi.org/10.1007/s00158-016-1563-6) (see pp. 5, 7)
- [9] P. A. Macmahon, “The Combinations of Resistances,” *Discrete Applied Mathematics*, vol. 54, no. 2–3, pp. 225–228, Oct. 1994, Reprinted from The Electrician, 1892. doi: [10.1016/0166-218X\(94\)90024-8](https://doi.org/10.1016/0166-218X(94)90024-8) (see pp. 5, 137, 138)
- [10] Z. A. Lomnicki, “Two-Terminal Series-Parallel Networks,” *Advances in Applied Probability*, vol. 4, no. 1, pp. 109–150, Apr. 1972. doi: [10.2307/1425808](https://doi.org/10.2307/1425808) (see pp. 5, 138)
- [11] Y. Isokawa, “Series-Parallel Circuits and Continued Fractions,” *Applied Mathematical Sciences*, vol. 10, no. 27, pp. 1321–1331, 2016. doi: [10.12988/ams.2016.63103](https://doi.org/10.12988/ams.2016.63103) (see pp. 5, 138)
- [12] D. R. Herber and J. T. Allison, “Passive Analog Circuit Synthesis through Enumeration,” *to be submitted*, (see pp. 5, 137, 194)
- [13] A. E. Bayrak, Y. Ren, and P. Y. Papalambros, “Topology Generation for Hybrid Electric Vehicle Architecture Design,” *Journal of Mechanical Design*, vol. 138, no. 8, p. 081 401, Jun. 2016. doi: [10.1115/1.4033656](https://doi.org/10.1115/1.4033656) (see pp. 5, 17, 42, 139)

- [14] E. Pennestri and P. P. Valentini, "Kinematics and Enumeration of Combined Harmonic Drive Gearings," *Journal of Mechanical Design*, vol. 137, no. 12, p. 122303, Dec. 2015. doi: [10.1115/1.4031590](https://doi.org/10.1115/1.4031590) (see pp. 5, 42)
- [15] J. M. del Castillo, "Enumeration of 1-DOF Planetary Gear Train Graphs Based on Functional Constraints," *Journal of Mechanical Design*, vol. 124, no. 4, pp. 723–732, Dec. 2002. doi: [10.1115/1.1514663](https://doi.org/10.1115/1.1514663) (see pp. 5, 42, 139)
- [16] W. Ma, A. Trusina, H. El-Samad, *et al.*, "Defining Network Topologies That Can Achieve Biochemical Adaptation," *Cell*, vol. 138, no. 4, pp. 760–773, Aug. 2009. doi: [10.1016/j.cell.2009.06.013](https://doi.org/10.1016/j.cell.2009.06.013) (see pp. 5, 42, 139)
- [17] N. Cheney, R. MacCurdy, J. Clune, *et al.*, "Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Powerful Generative Encoding," in *Genetic and Evolutionary Computation Conference*, Amsterdam, The Netherlands, Jul. 2013 (see p. 5)
- [18] C. M. Chilan, D. R. Herber, Y. K. Nakka, *et al.*, "Co-Design of Strain-Actuated Solar Arrays for Spacecraft Precision Pointing and Jitter Reduction," *AIAA Journal*, vol. 55, no. 9, pp. 3180–3195, Sep. 2017. doi: [10.2514/1.J055748](https://doi.org/10.2514/1.J055748) (see pp. 5, 45, 49, 50, 59–61, 65, 70, 77, 82, 83, 117, 164, 168)
- [19] R. G. Budynas and K. J. Nisbett, *Shigley's Mechanical Engineering Design*, 10th ed. McGraw-Hill, 2014, isbn: 978-0073398204 (see p. 5)
- [20] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*, 1st ed. Princeton University Press, 2012, isbn: 978-0691151878 (see pp. 6, 49, 51, 52, 55, 59, 66, 90, 96, 104, 128)
- [21] J. T. Allison and D. R. Herber, "Multidisciplinary Design Optimization of Dynamic Engineering Systems," *AIAA Journal*, vol. 52, no. 4, pp. 691–710, Apr. 2014. doi: [10.2514/1.J052182](https://doi.org/10.2514/1.J052182) (see pp. 6–8, 44–46, 50, 57, 58, 68, 86)
- [22] J. Falnes, *Ocean Waves and Oscillating Systems*, 1st ed. Cambridge University Press, 2002, isbn: 978-0521782111 (see p. 7)
- [23] Z. Gan, Z. Yang, T. Shang, *et al.*, "Automated Synthesis of Passive Analog Filters Using Graph Representation," *Expert Systems with Applications*, vol. 37, no. 3, pp. 1887–1898, Mar. 2010. doi: [10.1016/j.eswa.2009.07.013](https://doi.org/10.1016/j.eswa.2009.07.013) (see pp. 7, 137, 138, 142, 145, 154, 156, 158, 162)
- [24] D. R. Herber and J. T. Allison, "Wave Energy Extraction Maximization in Irregular Ocean Waves Using Pseudospectral Methods," in *International Design Engineering Technical Conferences*, Portland, OR, USA, Aug. 2013. doi: [10.1115/DETC2013-12600](https://doi.org/10.1115/DETC2013-12600) (see pp. 7, 45, 49, 50, 61)
- [25] D. R. Herber, "Dynamic System Design Optimization of Wave Energy Converters Utilizing Direct Transcription," Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, May 2014. url: <http://hdl.handle.net/2142/49463> (see pp. 7, 8, 45–47, 49, 50, 55, 58, 59, 66, 71, 92, 94, 96, 101, 133, 135, 176)
- [26] A. Bayrak, "Topology Considerations in Hybrid Electric Vehicle Powertrain Architecture Design," PhD dissertation, The University of Michigan, Ann Arbor, MI, USA, May 2015. url: <http://hdl.handle.net/2027.42/111412> (see pp. 7, 8)
- [27] J. T. Allison, T. Guo, and Z. Han, "Co-Design of an Active Suspension Using Simultaneous Dynamic Optimization," *Journal of Mechanical Design*, vol. 136, no. 8, p. 081003, Jun. 2014. doi: [10.1115/1.4027335](https://doi.org/10.1115/1.4027335) (see pp. 8, 37, 40, 44–46, 49, 50, 57, 58, 60, 61, 86, 166, 173, 176, 191–193, 198, 199)
- [28] H. Son and K.-M. Lee, "Open-Loop Controller Design and Dynamic Characteristics of a Spherical Wheel Motor," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 10, pp. 3475–3482, Oct. 2010. doi: [10.1109/TIE.2009.2039454](https://doi.org/10.1109/TIE.2009.2039454) (see p. 8)

- [29] S. Schaal and C. G. Atkeson, “Open Loop Stable Control Strategies for Robot Juggling,” in *IEEE International Conference on Robotics and Automation*, Atlanta, GA, USA, May 1993. doi: [10.1109/robot.1993.292260](https://doi.org/10.1109/robot.1993.292260) (see p. 8)
- [30] S. van Mourik, H. Zwart, and K. J. Keesman, “Integrated Open Loop Control and Design of a Food Storage Room,” *Biosystems Engineering*, vol. 104, no. 4, pp. 493–502, Dec. 2009. doi: [10.1016/j.biosystemseng.2009.09.010](https://doi.org/10.1016/j.biosystemseng.2009.09.010) (see p. 8)
- [31] M. Karkee and B. L. Steward, “Study of the Open and Closed Loop Characteristics of a Tractor and a Single Axle Towed Implement System,” *Journal of Terramechanics*, vol. 47, no. 6, pp. 379–393, Dec. 2010. doi: [10.1016/j.jterra.2010.05.005](https://doi.org/10.1016/j.jterra.2010.05.005) (see p. 8)
- [32] D. R. Herber and J. T. Allison, “Nested and Simultaneous Solution Strategies for General Combined Plant and Controller Design Problems,” in *ASME International Design Engineering Technical Conferences*, Cleveland, OH, USA, Aug. 2017 (see pp. 8, 14, 44, 75, 85, 117, 135, 176, 196, 197)
- [33] H. K. Fathy, P. Y. Papalambros, A. G. Ulsoy, *et al.*, “Nested Plant/Controller Optimization with Application to Combined Passive/Active Automotive Suspensions,” in *American Control Conference*, vol. 4, Denver, CO, USA, Jun. 2003, pp. 3375–3380. doi: [10.1109/ACC.2003.1244053](https://doi.org/10.1109/ACC.2003.1244053) (see pp. 8, 44, 45, 49, 50, 59, 61, 191–193, 198, 201)
- [34] J. Lygeros, S. Sastry, and C. Tomlin, *Hybrid Systems: Modeling, Analysis and Control*, Dec. 2008 (see p. 9)
- [35] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*, 1st ed. Cambridge University Press, 2017, isbn: 978-1107652873 (see p. 9)
- [36] I. D. Landau and Z. Gianluca, *Digital Control Systems*. Springer, 2006, isbn: 978-1846280559. doi: [10.1007/978-1-84628-056-6](https://doi.org/10.1007/978-1-84628-056-6) (see p. 9)
- [37] J. D. Lohn and S. P. Colombano, “A Circuit Representation Technique for Automated Circuit Design,” *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 3, pp. 205–219, Sep. 1999. doi: [10.1109/4235.788491](https://doi.org/10.1109/4235.788491) (see pp. 10, 138, 145, 154, 156, 160, 161)
- [38] W. Borutzky, *Bond Graph Methodology*, 1st ed. Springer, 2010, isbn: 978-1848828827. doi: [10.1007/978-1-84882-882-7](https://doi.org/10.1007/978-1-84882-882-7) (see pp. 10, 141, 193, 194)
- [39] The MathWorks. Simulink: Simulation and Model-Based Design, url: <https://www.mathworks.com/help/simulink/index.html> (visited on 10/17/2016) (see p. 10)
- [40] C.-W. Ho, A. Ruehli, and P. Brennan, “The Modified Nodal Approach to Network Analysis,” *IEEE Transactions on Circuits and Systems*, vol. 22, no. 6, pp. 504–509, Jun. 1975. doi: [10.1109/tcs.1975.1084079](https://doi.org/10.1109/tcs.1975.1084079) (see pp. 10, 141, 146)
- [41] D. R. Herber and J. T. Allison, “Enhancements to the Perfect Matching-based Tree Algorithm for Generating Architectures,” Engineering System Design Lab, Tech. Rep. preprint-v2, 2017. url: <http://systemdesign.illinois.edu/publications/Her17d.pdf> (see pp. 14, 142–144, 211)
- [42] R. A. Willem, “Design and Science,” *Design Studies*, vol. 11, no. 1, pp. 43–47, Jan. 1990. doi: [10.1016/0142-694X\(90\)90013-3](https://doi.org/10.1016/0142-694X(90)90013-3) (see p. 14)
- [43] S. Mittal and F. Frayman, “Towards a Generic Model of Configuration Tasks,” in *International Joint Conference on Artificial Intelligence*, Detroit, MI, USA, Aug. 1989, pp. 1395–1401 (see pp. 14, 16, 19, 42)
- [44] D. F. Wyatt, D. C. Wynn, and P. J. Clarkson, “A Scheme for Numerical Representation of Graph Structures in Engineering Design,” *Journal of Mechanical Design*, vol. 136, no. 1, p. 011010, Jan. 2014. doi: [10.1115/1.4025961](https://doi.org/10.1115/1.4025961) (see pp. 14, 21, 26, 42)

- [45] J. Cagan, M. I. Campbell, S. Finger, *et al.*, “A Framework for Computational Design Synthesis: Model and Applications,” *Journal of Computing and Information Science in Engineering*, vol. 5, no. 3, pp. 171–181, Sep. 2005. doi: [10.1115/1.2013289](https://doi.org/10.1115/1.2013289) (see p. 14)
- [46] A. Chakrabarti, K. Shea, R. Stone, *et al.*, “Computer-Based Design Synthesis Research: An Overview,” *Journal of Computing and Information Science in Engineering*, vol. 11, no. 2, p. 021003, Jun. 2011. doi: [10.1115/1.3593409](https://doi.org/10.1115/1.3593409) (see pp. 14, 15)
- [47] J. Chan, K. Fu, C. Schunn, *et al.*, “On the Benefits and Pitfalls of Analogies for Innovative Design: Ideation Performance Based on Analogical Distance, Commonness, and Modality of Examples,” *Journal of Mechanical Design*, vol. 133, no. 8, p. 081004, Aug. 2011. doi: [10.1115/1.4004396](https://doi.org/10.1115/1.4004396) (see pp. 14, 15)
- [48] J. S. Linsey, I. Tseng, K. Fu, *et al.*, “A Study of Design Fixation, Its Mitigation and Perception in Engineering Design Faculty,” *Journal of Mechanical Design*, vol. 132, no. 4, p. 041003, Apr. 2010. doi: [10.1115/1.4001110](https://doi.org/10.1115/1.4001110) (see p. 14)
- [49] A. Hooshmand, M. I. Campbell, and K. Shea, “Steps in Transforming Shapes Generated With Generative Design Into Simulation Models,” in *International Design Engineering Technical Conferences*, vol. 3, Chicago, IL, USA, Aug. 2012, pp. 883–892. doi: [10.1115/DETC2012-71056](https://doi.org/10.1115/DETC2012-71056) (see p. 15)
- [50] A. Khetan, D. J. Lohan, and J. T. Allison, “Managing Variable-dimension Structural Optimization Problems Using Generative Algorithms,” *Structural and Multidisciplinary Optimization*, vol. 52, no. 4, pp. 695–715, Jun. 2015. doi: [10.1007/s00158-015-1262-8](https://doi.org/10.1007/s00158-015-1262-8) (see p. 15)
- [51] C. Münzer, B. Helms, and K. Shea, “Automatically Transforming Object-Oriented Graph-Based Representations Into Boolean Satisfiability Problems for Computational Design Synthesis,” *Journal of Mechanical Design*, vol. 135, no. 10, p. 101001, Jul. 2013. doi: [10.1115/1.4024850](https://doi.org/10.1115/1.4024850) (see pp. 15, 16, 42)
- [52] T. Guo, “Design of Genetic Regulatory Networks,” Master’s thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 2014. url: <http://hdl.handle.net/2142/49667> (see p. 15)
- [53] L. C. Schmidt and J. Cagan, “GGREADA: A Graph Grammar-based Machine Design Algorithm,” *Research in Engineering Design*, vol. 9, no. 4, pp. 195–213, Dec. 1997. doi: [10.1007/BF01589682](https://doi.org/10.1007/BF01589682) (see p. 15)
- [54] L. C. Schmidt, H. Shetty, and S. C. Chase, “A Graph Grammar Approach for Structure Synthesis of Mechanisms,” *Journal of Mechanical Design*, vol. 122, no. 4, pp. 371–376, Dec. 2000. doi: [10.1115/1.1315299](https://doi.org/10.1115/1.1315299) (see pp. 15, 29)
- [55] G. S. Hornby, H. Lipson, and J. B. Pollack, “Generative Representations for the Automated Design of Modular Physical Robots,” *IEEE Transactions on Robotics and Automation*, vol. 19, no. 4, pp. 703–719, Aug. 2003. doi: [10.1109/TRA.2003.814502](https://doi.org/10.1109/TRA.2003.814502) (see p. 15)
- [56] C. R. Bryant, D. A. McAdams, R. B. Stone, *et al.*, “A Computational Technique for Concept Generation,” in *International Design Engineering Technical Conferences*, vol. 5a, Long Beach, CA, USA, Sep. 2005, pp. 267–276. doi: [10.1115/DETC2005-85323](https://doi.org/10.1115/DETC2005-85323) (see p. 15)
- [57] A. C. Starling and K. Shea, “A Parallel Grammar for Simulation-Driven Mechanical Design Synthesis,” in *International Design Engineering Technical Conferences*, vol. 2, Long Beach, CA, USA, Sep. 2005, pp. 427–436. doi: [10.1115/DETC2005-85414](https://doi.org/10.1115/DETC2005-85414) (see p. 15)
- [58] D. F. Wyatt, D. C. Wynn, J. P. Jarrett, *et al.*, “Supporting Product Architecture Design Using Computational Design Synthesis with Network Structure Constraints,” *Research in Engineering Design*, vol. 23, no. 1, pp. 17–52, Apr. 2012. doi: [10.1007/s00163-011-0112-y](https://doi.org/10.1007/s00163-011-0112-y) (see pp. 15, 23)
- [59] G. L. Snavely and P. Y. Papalambros, “Abstraction as a Configuration Design Methodology,” in *Advances in Design Automation*, vol. 65, Albuquerque, NM, USA, Sep. 1993, pp. 297–305 (see pp. 16, 19, 31, 42)

- [60] C. Godsil and G. Royle, *Algebraic Graph Theory*, 1st ed. Springer, 2001, isbn: 978-1461301639. doi: [10.1007/978-1-4613-0163-9](https://doi.org/10.1007/978-1-4613-0163-9) (see pp. 17, 18, 221)
- [61] R. Diestel, *Graph Theory*, 2nd ed. Springer, 2000, isbn: 978-0387950141 (see pp. 17, 24, 28, 215, 219, 225, 235)
- [62] F. J. Rispoli, *Applications of Discrete Mathematics; ch. Applications of Subgraph Enumeration*, updated edition. McGraw-Hill, 2007 (see pp. 17, 19, 21)
- [63] Z. Wu, M. I. Campbell, and B. R. Fernández, “Bond Graph Based Automated Modeling for Computer-Aided Design of Dynamic Systems,” *Journal of Mechanical Design*, vol. 130, no. 4, p. 041102, Apr. 2008. doi: [10.1115/1.2885180](https://doi.org/10.1115/1.2885180) (see p. 17)
- [64] N. J. A. Sloane. (Nov. 3, 2017). Sequence A001147, The On-Line Encyclopedia of Integer Sequences, url: <https://oeis.org/A001147> (see p. 19)
- [65] D. R. Herber, T. Guo, and J. T. Allison. PM Architectures Project, GitHub, url: <https://github.com/danielrherber/pm-architectures-project> (see pp. 21, 32, 144, 162, 263)
- [66] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, 1st ed. Holt, Rinehart and Winston, 1976, isbn: 978-0486414539 (see pp. 25, 235)
- [67] B. D. McKay and A. Piperno, “Practical Graph Isomorphism, II,” *Journal of Symbolic Computation*, vol. 60, pp. 94–112, Jan. 2014. doi: [10.1016/j.jsc.2013.09.003](https://doi.org/10.1016/j.jsc.2013.09.003) (see p. 29)
- [68] L. Babai. (2016). Graph Isomorphism in Quasipolynomial Time. version 2, url: <https://arxiv.org/abs/1512.03547> (see p. 29)
- [69] G. Csárdi and T. Nepusz, “The igraph Software Package for Complex Network Research,” *InterJournal*, vol. Complex Systems, p. 1695, 2006. url: <http://igraph.org> (see p. 29)
- [70] L. P. Cordella, P. Foggia, C. Sansone, *et al.*, “An Improved Algorithm for Matching Large Graphs,” in *IAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition*, Ischia, Italy, May 2001, pp. 149–159 (see p. 29)
- [71] C. Königseder and K. Shea, “Comparing Strategies for Topologic and Parametric Rule Application in Automated Computational Design Synthesis,” *Journal of Mechanical Design*, vol. 138, no. 1, p. 011102, Jan. 2016. doi: [10.1115/1.4031714](https://doi.org/10.1115/1.4031714) (see p. 29)
- [72] R. C. Read, “Every One a Winner or How to Avoid Isomorphism Search When Cataloguing Combinatorial Configurations,” *Annals of Discrete Mathematics*, vol. 2, pp. 107–120, 1978. doi: [10.1016/S0167-5060\(08\)70325-X](https://doi.org/10.1016/S0167-5060(08)70325-X) (see p. 29)
- [73] J.-L. Faulon, C. J. Churchwell, and D. P. Visco Jr., “The Signature Molecular Descriptor. 2. Enumerating Molecules from Their Extended Valence Sequences,” *Journal of Chemical Information and Modeling*, vol. 43, no. 3, pp. 721–734, Mar. 2003. doi: [10.1021/ci020346o](https://doi.org/10.1021/ci020346o) (see pp. 29, 31, 42, 225)
- [74] R. E. Carhart, D. H. Smith, H. Brown, *et al.*, “Applications of Artificial Intelligence for Chemical Inference. XVII. Approach to Computer-Assisted Elucidation of Molecular Structure,” *Journal of the American Chemical Society*, vol. 97, no. 20, pp. 5755–5762, Oct. 1975. doi: [10.1021/ja00853a021](https://doi.org/10.1021/ja00853a021) (see pp. 31, 42)
- [75] C. J. Colbourn and R. C. Read, “Orderly Algorithms for Generating Restricted Classes of Graphs,” *Journal of Graph Theory*, vol. 3, no. 2, pp. 187–195, Jun. 1979. doi: [10.1002/jgt.3190030210](https://doi.org/10.1002/jgt.3190030210) (see p. 31)
- [76] P. Flajolet, D. Gardy, and L. Thimonier, “Birthday Paradox, Coupon Collectors, Caching Algorithms and Self-organizing Search,” *Discrete Applied Mathematics*, vol. 39, no. 3, pp. 207–229, Nov. 1992. doi: [10.1016/0166-218X\(92\)90177-C](https://doi.org/10.1016/0166-218X(92)90177-C) (see p. 42)

- [77] L. Ruddigkeit, R. v. Deursen, L. C. Blum, *et al.*, “Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17,” *Journal of Chemical Information and Modeling*, vol. 52, no. 11, pp. 2864–2875, Nov. 2012. doi: [10.1021/ci300415d](https://doi.org/10.1021/ci300415d) (see p. 42)
- [78] R. M. Foster, “Geometrical Circuits of Electrical Networks,” *Electrical Engineering*, vol. 51, no. 1, pp. 309–317, Jan. 1932. doi: [10.1109/EE.1932.6429606](https://doi.org/10.1109/EE.1932.6429606) (see pp. 42, 138)
- [79] M. Berlingario, F. Bonchi, B. Bringmann, *et al.*, “Mining Graph Evolution Rules,” in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2009, vol. 5781, pp. 115–130. doi: [10.1007/978-3-642-04180-8_25](https://doi.org/10.1007/978-3-642-04180-8_25) (see pp. 42, 162)
- [80] D. Dörner, *The Logic Of Failure: Recognizing And Avoiding Error In Complex Situations*, revised edition. Basic Books, 1997, isbn: 978-0201479485 (see p. 44)
- [81] A. P. Deshmukh and J. T. Allison, “Multidisciplinary Dynamic Optimization of Horizontal Axis Wind Turbine Design,” *Structural and Multidisciplinary Optimization*, vol. 53, no. 1, pp. 15–27, Jan. 2016. doi: [10.1007/s00158-015-1308-y](https://doi.org/10.1007/s00158-015-1308-y) (see pp. 44, 45, 49, 50, 60, 61)
- [82] H.-S. Yan and G.-J. Yan, “Integrated Control and Mechanism Design for the Variable Input-Speed Servo Four-bar Linkages,” *Mechatronics*, vol. 19, no. 2, pp. 274–285, Mar. 2009. doi: [10.1016/j.mechatronics.2008.07.008](https://doi.org/10.1016/j.mechatronics.2008.07.008) (see pp. 44, 45, 49, 61)
- [83] H. K. Fathy, J. A. Reyer, P. Y. Papalambros, *et al.*, “On the Coupling between the Plant and Controller Optimization Problems,” in *American Control Conference*, vol. 3, Arlington, VA, USA, Jun. 2001, pp. 1864–1869. doi: [10.1109/ACC.2001.946008](https://doi.org/10.1109/ACC.2001.946008) (see pp. 44, 45, 49–51, 53, 55, 59, 61, 63, 176)
- [84] J. R.R. A. Martins and A. B. Lambe, “Multidisciplinary Design Optimization: A Survey of Architectures,” *AIAA Journal*, vol. 51, no. 9, pp. 2049–2075, Sep. 2013. doi: [10.2514/1.j051895](https://doi.org/10.2514/1.j051895) (see pp. 44, 45)
- [85] J. T. Allison and S. Nazari, “Combined Plant and Controller Design Using Decomposition-Based Design Optimization and the Minimum Principle,” in *ASME International Design Engineering Technical Conferences*, Montreal, Canada, Aug. 2010, pp. 765–774. doi: [10.1115/DETC2010-28887](https://doi.org/10.1115/DETC2010-28887) (see pp. 45, 67)
- [86] A. Kusiak and N. Larson, “Decomposition and Representation Methods in Mechanical Design,” *Journal of Mechanical Design*, vol. 117, no. B, pp. 17–24, Jun. 1995. doi: [10.1115/1.2836453](https://doi.org/10.1115/1.2836453) (see p. 45)
- [87] B. D. Frischknecht, D. L. Peters, and P. Y. Papalambros, “Pareto Set Analysis: Local Measures of Objective Coupling in Multiobjective Design Optimization,” *Structural and Multidisciplinary Optimization*, vol. 43, no. 5, pp. 617–630, Nov. 2011. doi: [10.1007/s00158-010-0599-2](https://doi.org/10.1007/s00158-010-0599-2) (see pp. 45, 49)
- [88] J. A. Reyer, H. K. Fathy, P. Y. Papalambros, *et al.*, “Comparison of Combined Embodiment Design and Control Optimization Strategies Using Optimality Conditions,” in *Design Engineering Technical Conference*, Pittsburgh, PA, USA, Sep. 2001 (see pp. 45, 49, 60)
- [89] A. L. Hale, R. J. Lisowski, and W. E. Dahl, “Optimal Simultaneous Structural and Control Design of Maneuvering Flexible Spacecraft,” *AIAA Journal of Guidance, Control, and Dynamics*, vol. 8, no. 1, pp. 86–93, Jan. 1985. doi: [10.2514/3.19939](https://doi.org/10.2514/3.19939) (see pp. 45, 49)
- [90] F. Eastep, N. Khot, and R. Grandhi, “Improving the Active Vibrational Control of Large Space Structures through Structural Modifications,” *Acta Astronautica*, vol. 15, no. 6–7, pp. 383–389, 1987. doi: [10.1016/0094-5765\(87\)90174-3](https://doi.org/10.1016/0094-5765(87)90174-3) (see pp. 45, 49, 59)
- [91] M. Sunar and S. S. Rao, “Simultaneous Passive and Active Control Design of Structures Using Multiobjective Optimization Strategies,” *Computers & Structures*, vol. 48, no. 5, pp. 913–924, Sep. 1993. doi: [10.1016/0045-7949\(93\)90513-D](https://doi.org/10.1016/0045-7949(93)90513-D) (see pp. 45, 49, 59, 61)

- [92] D. L. Peters, P. Y. Papalambros, and A. G. Ulsoy, “Control Proxy Functions for Sequential Design and Control Optimization,” *Journal of Mechanical Design*, vol. 133, no. 9, p. 091007, Sep. 2011. doi: [10.1115/1.4004792](https://doi.org/10.1115/1.4004792) (see pp. 45, 49, 50)
- [93] D. L. Peters, P. Y. Papalambros, and A. G. Ulsoy, “On Measures of Coupling Between the Artifact and Controller Optimal Design Problems,” in *International Design Engineering Technical Conferences*, vol. 2, San Diego, CA, USA, 2009, pp. 1363–1372. doi: [10.1115/DETC2009-86868](https://doi.org/10.1115/DETC2009-86868) (see pp. 45, 49, 50)
- [94] S. Maraniello and R. Palacios, “Optimal Vibration Control and Co-Design of Very Flexible Actuated Structures,” *Journal of Sound and Vibration*, vol. 377, pp. 1–21, Sep. 2016. doi: [10.1016/j.jsv.2016.05.018](https://doi.org/10.1016/j.jsv.2016.05.018) (see pp. 45, 49, 61)
- [95] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, 2nd ed. SIAM, Jan. 2010, isbn: 978-0898716887. doi: [10.1137/1.9780898718577](https://doi.org/10.1137/1.9780898718577) (see pp. 46, 56–58, 66, 70, 71, 86, 88, 90, 92, 98, 100, 101, 106, 113, 133, 176)
- [96] L. T. Biegler, *Nonlinear Programming*, 2nd ed. SIAM, Jan. 2010, isbn: 978-0898717020. doi: [10.1137/1.9780898719383](https://doi.org/10.1137/1.9780898719383) (see pp. 46, 56–58, 66, 71, 86, 90, 92, 98, 101, 106, 117, 133, 176)
- [97] A. V. Rao, “Survey of Numerical Methods for Optimal Control,” *Advances in the Astronautical Sciences*, vol. 135, no. 1, pp. 497–528, 2010 (see pp. 46, 57, 58, 70, 71, 88, 99, 103, 106, 134)
- [98] B. Colson, P. Marcotte, and G. Savard, “An Overview of Bilevel Optimization,” *Annals of Operations Research*, vol. 153, no. 1, pp. 235–256, Sep. 2007. doi: [10.1007/s10479-007-0176-2](https://doi.org/10.1007/s10479-007-0176-2) (see pp. 47, 117)
- [99] L. N. Vicente and P. H. Calamai, “Bilevel and Multilevel Programming: A Bibliography Review,” *Journal of Global Optimization*, vol. 5, no. 3, pp. 291–306, 1994. doi: [10.1007/BF01096458](https://doi.org/10.1007/BF01096458) (see p. 47)
- [100] T. Tanino and T. Ogawa, “An Algorithm for Solving Two-level Convex Optimization Problems,” *International Journal of Systems Science*, vol. 15, no. 2, pp. 163–174, 1984. doi: [10.1080/00207728408926552](https://doi.org/10.1080/00207728408926552) (see p. 47)
- [101] W. K. Belvin and K. C. Park, “Structural Tailoring and Feedback Control Synthesis: An Interdisciplinary Approach,” *Journal of Guidance, Control, and Dynamics*, vol. 13, no. 3, pp. 424–429, May 1990. doi: [10.2514/3.25354](https://doi.org/10.2514/3.25354) (see pp. 49, 59, 60, 166)
- [102] S. S. Rao, “Combined Structural and Control Optimization of Flexible Structures,” *Engineering Optimization*, vol. 13, no. 1, pp. 1–16, Jan. 1988. doi: [10.1080/03052158808940943](https://doi.org/10.1080/03052158808940943) (see pp. 49, 50, 59)
- [103] B. Chachuat. (2007). Nonlinear and Dynamic Optimization: From Theory to Practice. version IC-32: winter semester 2006/2007, Automatic Control Laboratory, url: [https://infoscience.epfl.ch/record/111939/files/Chachuat_07\(IC32\).pdf](https://infoscience.epfl.ch/record/111939/files/Chachuat_07(IC32).pdf) (see pp. 51, 52, 55)
- [104] A. E. Bryson Jr. and Y.-C. Ho, *Applied Optimal Control*, revised edition. Taylor & Francis, 1975, isbn: 978-0891162285 (see pp. 51, 56, 59, 63, 66, 71, 84, 85, 90, 94, 96, 113, 115, 116, 119, 122, 125, 128, 133)
- [105] P. Y. Papalambros and D. J. Wilde, *Principles of Optimal Design*, 3rd ed. Cambridge University Press, 2017, isbn: 978-1107132672 (see pp. 51, 52, 54, 65, 70, 75, 196)
- [106] L. S. Pontryagin, *The Mathematical Theory of Optimal Processes*. Interscience, 1962 (see p. 52)
- [107] J. Doležal, “On the Solution of Optimal Control Problems Involving Parameters and General Boundary Conditions,” *Kybernetika*, vol. 17, no. 1, pp. 71–81, 1981 (see p. 53)
- [108] E. W. Weisstein. Total Derivative, MathWorld—A Wolfram Web Resource, url: <http://mathworld.wolfram.com/TotalDerivative.html> (visited on 12/06/2016) (see p. 54)

- [109] D. R. Herber, “Basic Implementation of Multiple-Interval Pseudospectral Methods to Solve Optimal Control Problems,” Engineering System Design Lab, University of Illinois at Urbana-Champaign, Urbana, IL, USA, Tech. Rep. UIUC-ESDL-2015-01, 2015. url: <http://hdl.handle.net/2142/77888> (see pp. 57, 98–100, 103, 124, 132, 241)
- [110] L. T. Biegler, “An Overview of Simultaneous Strategies for Dynamic Optimization,” *Chemical Engineering and Processing: Process Intensification*, vol. 46, no. 11, pp. 1043–1053, Nov. 2007. doi: [10.1016/j.cep.2006.06.021](https://doi.org/10.1016/j.cep.2006.06.021) (see p. 58)
- [111] D. Garg, “Advances in Global Pseudospectral Methods for Optimal Control,” PhD dissertation, University of Florida, Gainesville, FL, USA, 2011. url: <http://ufdc.ufl.edu/UFE0043196/00001> (see pp. 58, 133)
- [112] M. P. Kelly. (2015). Transcription Methods for Trajectory Optimization, Cornell University, url: http://www.matthewpeterkelly.com/research/MattKelly__Transcription_Methods_for_Trajectory_Optimization.pdf (see pp. 60, 67)
- [113] J. Onoda and R. T. Haftka, “An Approach to Structure/Control Simultaneous Optimization for Large Flexible Spacecraft,” *AIAA Journal*, vol. 25, no. 8, pp. 1133–1138, Aug. 1987. doi: [10.2514/3.9754](https://doi.org/10.2514/3.9754) (see p. 60)
- [114] D. R. Herber and J. T. Allison. Co-Design Examples Repository, GitHub, url: <https://github.com/danielrherber/co-design-examples-repository> (see pp. 61, 263)
- [115] D. R. Herber and J. T. Allison, “Unified Scaling of Dynamic Optimization Design Formulations,” in *ASME International Design Engineering Technical Conferences*, Cleveland, OH, USA, Aug. 2017 (see pp. 63, 68, 134)
- [116] T. Liu, S. Azarm, and N. Chopra, “On Decentralized Optimization for a Class of Multisubsystem Codesign Problems,” *Journal of Mechanical Design*, vol. 139, no. 12, p. 121404, Oct. 2017. doi: [10.1115/1.4037893](https://doi.org/10.1115/1.4037893) (see p. 67)
- [117] G. Yu. (2004). Syllabus for Algorithm Design and Implementations (see p. 68)
- [118] M. H. Holmes, *Introduction to the Foundations of Applied Mathematics*, 1st ed. Springer, 2009, isbn: 978-0387877495. doi: [10.1007/978-0-387-87765-5](https://doi.org/10.1007/978-0-387-87765-5) (see pp. 68–70, 72)
- [119] E. v. Groesen and J. Molenaar, “Dimensional Analysis and Scaling,” in *Continuum Modeling in the Physical Sciences*. SIAM, 2007, pp. 1–29, isbn: 978-0898716252 (see pp. 69, 72)
- [120] Y. A. Çengel and J. M. Cimbala, *Fluid Mechanics: Fundamentals and Applications*, 1st ed. McGraw-Hill, 2006, isbn: 0072472367, isbn: 978-0073380322 (see pp. 69, 70)
- [121] E. Buckingham, “On Physically Similar Systems; Illustrations of the Use of Dimensional Equations,” *Physical Review*, vol. 4, no. 4, pp. 345–376, Oct. 1914. doi: [10.1103/PhysRev.4.345](https://doi.org/10.1103/PhysRev.4.345) (see pp. 69, 72)
- [122] B. Kittirungsi, “A Scaling Methodology for Dynamic Systems: Quantification of Approximate Similitude and Use in Multiobjective Design,” PhD dissertation, The University of Michigan, Ann Arbor, MI, USA, May 2008. url: <http://hdl.handle.net/2027.42/58383> (see pp. 69, 70, 75)
- [123] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Prentice Hall, May 2002, isbn: 0130673897, isbn: 978-0130673893 (see p. 70)
- [124] L. Bergamaschi, J. Gondzio, and G. Zilli, “Preconditioning Indefinite Systems in Interior Point Methods for Optimization,” *Computational Optimization and Applications*, vol. 28, no. 2, pp. 149–171, Jul. 2004. doi: [10.1023/B:COAP.0000026882.34332.1b](https://doi.org/10.1023/B:COAP.0000026882.34332.1b) (see pp. 70, 134)
- [125] M. Benzi, “Preconditioning Techniques for Large Linear Systems: A Survey,” *Journal of Computational Physics*, vol. 182, no. 2, pp. 418–477, Nov. 2002. doi: [10.1006/jcph.2002.7176](https://doi.org/10.1006/jcph.2002.7176) (see pp. 70, 134)

- [126] M. Ghanekar, D. Wang, and G. Heppler, “Scaling Laws for Linear Controllers of Flexible Link Manipulators Characterized by Nondimensional Groups,” *IEEE Transactions on Robotics and Automation*, vol. 13, no. 1, pp. 117–127, Feb. 1997. doi: [10.1109/70.554352](https://doi.org/10.1109/70.554352) (see p. 70)
- [127] S. Brennan and A. Alleyne, “Robust Scalable Vehicle Control via Non-Dimensional Vehicle Dynamics,” *Vehicle System Dynamics*, vol. 36, no. 4-5, pp. 255–277, Nov. 2001. doi: [10.1076/vesd.36.4.255.3551](https://doi.org/10.1076/vesd.36.4.255.3551) (see p. 70)
- [128] S. Boyd and L. Vandenberghe, *Convex Optimization*, 7th ed. Cambridge University Press, 2009, isbn: 978-0521833783 (see pp. 72, 74, 80, 91, 95, 135)
- [129] G. Strang, *Calculus*, 1st ed. Wellesley-Cambridge Press, 1991 (see p. 72)
- [130] The MathWorks. fmincon: Find Minimum of Constrained Nonlinear Multivariable Function, url: <https://www.mathworks.com/help/optim/ug/fmincon.html> (visited on 10/16/2016) (see pp. 86, 147)
- [131] D. R. Herber, Y. H. Lee, and J. T. Allison, “Unified Framework for Solving General Linear-Quadratic Dynamic Optimization Problems Utilizing Direct Transcription and Quadratic Programming,” *to be submitted*, (see pp. 90, 197, 199)
- [132] H. H. Goldstine, *A History of the Calculus of Variations from the 17th through the 19th Century*, 1st ed. Springer, 1980, isbn: 978-1461381082. doi: [10.1007/978-1-4613-8106-8](https://doi.org/10.1007/978-1-4613-8106-8) (see p. 90)
- [133] N. Faedo, S. Olaya, and J. V. Ringwood, “Optimal Control, MPC and MPC-like Algorithms for Wave Energy Systems: An Overview,” *IFAC Journal of Systems and Control*, vol. 1, pp. 37–56, Sep. 2017. doi: [10.1016/j.ifacsc.2017.07.001](https://doi.org/10.1016/j.ifacsc.2017.07.001) (see pp. 90, 118, 135)
- [134] *Optimal Control and Dynamic Games*, ed. by C. Deissenberg and R. F. Hartl. Springer, 2005. doi: [10.1007/b136166](https://doi.org/10.1007/b136166) (see p. 90)
- [135] B. D. O. Anderson and J. B. Moore, *Optimal Control: Linear Quadratic Methods*. Dover, 2007, isbn: 978-0486457666 (see pp. 90, 96, 115)
- [136] M. A. Patterson and A. V. Rao, “GPOPS-II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming,” *ACM Transactions on Mathematical Software*, vol. 41, no. 1, pp. 1–37, Oct. 2014. doi: [10.1145/2558904](https://doi.org/10.1145/2558904) (see pp. 91, 99, 132, 133)
- [137] V. M. Becerra, *PSOPT Optimal Control Solver User Manual*, release 4 build 2015, 2015. url: https://github.com/PSOPT/psopt/blob/master/PSOPT/doc/PSOPT_Manual_R4.pdf (visited on 10/12/2017) (see pp. 91, 99, 133, 257)
- [138] P. E. Rutquist and M. M. Edvall, *PROPT - Matlab Optimal Control Software*, Pullman, WA, USA: TOMLAB Optimization, Apr. 2010 (see p. 91)
- [139] J. T. Betts, *Sparse Optimization Suite (SOS)*, Release 2015.11, 2015 (see p. 91)
- [140] O. von Stryk, *User’s Guide for DIRCOL: A Direct Collocation Method for the Numerical Solution of Optimal Control Problems*, version 2.1 edition, Technische Universität Darmstadt, Nov. 1999 (see pp. 91, 113, 116)
- [141] M. Herceg, M. Kvasnica, C. N. Jones, *et al.*, “Multi-Parametric Toolbox 3.0,” in *European Control Conference*, Zürich, Switzerland, Jul. 2013, pp. 502–510. url: <http://control.ee.ethz.ch/~mpt> (see p. 91)
- [142] The MathWorks. Model Predictive Control Toolbox: Design and Simulate Model Predictive Controllers, url: <https://www.mathworks.com/help/mpc/index.html> (visited on 10/02/2017) (see p. 91)

- [143] J.-S. Pang, “Methods for Quadratic Programming: A Survey,” *Computers & Chemical Engineering*, vol. 7, no. 5, pp. 583–594, Jan. 1983. doi: [10.1016/0098-1354\(83\)80004-0](https://doi.org/10.1016/0098-1354(83)80004-0) (see p. 93)
- [144] A. Altman and J. Gondzio, “Regularized Symmetric Indefinite Systems in Interior Point Methods for Linear and Quadratic Optimization,” *Optimization Methods and Software*, vol. 11, no. 1–4, pp. 275–302, Jan. 1999. doi: [10.1080/10556789908805754](https://doi.org/10.1080/10556789908805754) (see p. 93)
- [145] F. Delbos and J. C. Gilbert, “Global Linear Convergence of an Augmented Lagrangian Algorithm to Solve Convex Quadratic Optimization Problems,” *Journal of Convex Analysis*, vol. 12, no. 1, pp. 45–69, 2005 (see p. 93)
- [146] G. Lack and M. Enns, “Optimal Control Trajectories with Minimax Objective Functions by Linear Programming,” *IEEE Transactions on Automatic Control*, vol. 12, no. 6, pp. 749–752, Dec. 1967. doi: [10.1109/tac.1967.1098752](https://doi.org/10.1109/tac.1967.1098752) (see pp. 94, 96)
- [147] A. Sala, “Improving Performance Under Sampling-Rate Variations via Generalized Hold Functions,” *IEEE Transactions on Control Systems Technology*, vol. 15, no. 4, pp. 794–797, Jul. 2007. doi: [10.1109/tcst.2006.890302](https://doi.org/10.1109/tcst.2006.890302) (see p. 94)
- [148] S.-K. Wang and M. L. Nagurka, “Linear Quadratic Optimal Control Design Using Chebyshev-Based State Parameterization,” Carnegie Mellon University, Tech. Rep., 1992 (see pp. 94–96)
- [149] R. D. Hampton, C. R. Knospe, and M. A. Townsend, “A Practical Solution to the Deterministic Nonhomogeneous LQR Problem,” *Journal of Dynamic Systems, Measurement, and Control*, vol. 118, no. 2, p. 354, Jun. 1996. doi: [10.1115/1.2802329](https://doi.org/10.1115/1.2802329) (see pp. 94, 95)
- [150] M. Popescu, “Fundamental Solution for Linear Two-point Boundary Value Problem,” *Journal of Applied Mathematics and Computing*, vol. 31, no. 1–2, pp. 385–394, Sep. 2009. doi: [10.1007/s12190-008-0219-0](https://doi.org/10.1007/s12190-008-0219-0) (see pp. 94, 95)
- [151] G. Li, “Nonlinear Model Predictive Control of a Wave Energy Converter Based on Differential Flatness Parameterisation,” *International Journal of Control*, vol. 90, no. 1, pp. 68–77, Sep. 2015. doi: [10.1080/00207179.2015.1088173](https://doi.org/10.1080/00207179.2015.1088173) (see p. 94)
- [152] L. Han, M. K. Camlibel, J.-S. Pang, *et al.*, “A Unified Numerical Scheme for Linear-Quadratic Optimal Control Problems with Joint Control and State Constraints,” *Optimization Methods and Software*, vol. 27, no. 4–5, pp. 761–799, Oct. 2012. doi: [10.1080/10556788.2011.593624](https://doi.org/10.1080/10556788.2011.593624) (see pp. 94–97, 102)
- [153] M. Gerdts, “A Survey on Optimal Control Problems with Differential-Algebraic Equations,” in *Surveys in Differential-Algebraic Equations II*, Springer, 2015, pp. 103–161, isbn: 978-3319110493. doi: [10.1007/978-3-319-11050-9_3](https://doi.org/10.1007/978-3-319-11050-9_3) (see pp. 94–96)
- [154] S. L. Campbell and P. Kunkel, “On the Numerical Treatment of Linear–Quadratic Optimal Control Problems for General Linear Time-Varying Differential-Algebraic Equations,” *Journal of Computational and Applied Mathematics*, vol. 242, pp. 213–231, Apr. 2013. doi: [10.1016/j.cam.2012.10.011](https://doi.org/10.1016/j.cam.2012.10.011) (see pp. 94, 95)
- [155] C.-T. Chen, *Linear System Theory and Design*, 3rd ed. Oxford University Press, 1999, isbn: 978-0195392074 (see pp. 94, 134)
- [156] G.-Y. Tang, Y.-D. Zhao, and B.-L. Zhang, “Optimal Output Tracking Control for Nonlinear Systems Via Successive Approximation Approach,” *Nonlinear Analysis: Theory, Methods & Applications*, vol. 66, no. 6, pp. 1365–1377, Mar. 2007. doi: [10.1016/j.na.2006.01.021](https://doi.org/10.1016/j.na.2006.01.021) (see pp. 95, 96)
- [157] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan, “An FPGA Implementation of a Sparse Quadratic Programming Solver for Constrained Predictive Control,” in *International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, USA: ACM Press, 2011. doi: [10.1145/1950413.1950454](https://doi.org/10.1145/1950413.1950454) (see pp. 95, 96)

- [158] G. Bashein, “A Simplex Algorithm for On-line Computation of Time Optimal Controls,” *IEEE Transactions on Automatic Control*, vol. 16, no. 5, pp. 479–482, Oct. 1971. doi: [10.1109/tac.1971.1099776](https://doi.org/10.1109/tac.1971.1099776) (see pp. 95, 96)
- [159] A. Sideris and L. A. Rodriguez, “A Riccati Approach to Equality Constrained Linear Quadratic Optimal Control,” in *American Control Conference*, Baltimore, MD, USA, 2010, pp. 5167–5172. doi: [10.1109/acc.2010.5530688](https://doi.org/10.1109/acc.2010.5530688) (see pp. 95–97)
- [160] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*, 1st ed. Cambridge University Press, 2017, isbn: 978-1107016880 (see pp. 96, 97, 102, 104, 117)
- [161] J. Shen, T. Tang, and L.-L. Wang, *Spectral Methods: Algorithms, Analysis and Applications*, 1st ed. Springer, 2011, isbn: 978-3540710400. doi: [10.1007/978-3-540-71041-7](https://doi.org/10.1007/978-3-540-71041-7) (see pp. 98, 255, 256)
- [162] F. Fahroo and I. M. Ross, “Direct Trajectory Optimization by a Chebyshev Pseudospectral Method,” *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 160–166, 2002. doi: [10.2514/2.4862](https://doi.org/10.2514/2.4862) (see pp. 98–100, 256)
- [163] V. M. Becerra and R. K. H. Galvão, “Um Tutorial Sobre Métodos Pseudo-Espectrais Para Controle Ótimo Computacional,” *Sba: Controle & Automação Sociedade Brasileira de Automatica*, vol. 21, no. 3, pp. 224–244, Jun. 2010. doi: [10.1590/s0103-17592010000300002](https://doi.org/10.1590/s0103-17592010000300002) (see pp. 99, 100, 255–257)
- [164] F. Fahroo and I. M. Ross, “Advances in Pseudospectral Methods for Optimal Control,” in *AIAA Guidance, Navigation and Control Conference and Exhibit*, Honolulu, HI, USA, Aug. 2008. doi: [10.2514/6.2008-7309](https://doi.org/10.2514/6.2008-7309) (see pp. 99, 100, 103, 256)
- [165] J. T. Betts, “Survey of Numerical Methods for Trajectory Optimization,” *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, Mar. 1998. doi: [10.2514/2.4231](https://doi.org/10.2514/2.4231) (see p. 99)
- [166] M. Bittner, “Utilization of Problem and Dynamic Characteristics for Solving Large Scale Optimal Control Problems,” PhD dissertation, Technical University of Munich, Munich, Germany, Apr. 2017. url: <http://nbn-resolving.de/urn/resolver.pl?urn=nbn:de:bvb:91-diss-20170511-1343164-1-1> (see p. 99)
- [167] D. Pardo, L. Moller, M. Neunert, *et al.*, “Evaluating Direct Transcription and Nonlinear Optimization Methods for Robot Motion Planning,” *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 946–953, Jul. 2016. doi: [10.1109/lra.2016.2527062](https://doi.org/10.1109/lra.2016.2527062) (see p. 99)
- [168] D. Sonawane, M. Pathak, and V. R. Subramanian, “Convergence Rates for Direct Transcription of Optimal Control Problems Using Second Derivative Methods,” in *American Control Conference*, Boston, MA, USA: IEEE, Jul. 2016, pp. 215–220. doi: [10.1109/acc.2016.7524918](https://doi.org/10.1109/acc.2016.7524918) (see p. 99)
- [169] J. T. Betts and W. P. Huffman, “Mesh Refinement in Direct Transcription Methods for Optimal Control,” *Optimal Control Applications and Methods*, vol. 19, no. 1, pp. 1–21, Jan. 1998. doi: [10.1002/\(sici\)1099-1514\(199801/02\)19:1<1::aid-oca616>3.0.co;2-q](https://doi.org/10.1002/(sici)1099-1514(199801/02)19:1<1::aid-oca616>3.0.co;2-q) (see pp. 99, 101, 132, 133)
- [170] P. Williams, “A Comparison of Differentiation and Integration Based Direct Transcription Methods,” in *AAS/AIAA Space Flight Mechanics Meetings*, vol. 120, Copper Mountain, CO, USA, Jan. 2005, pp. 389–408 (see pp. 99, 101, 118)
- [171] R. Amrit, J. B. Rawlings, and L. T. Biegler, “Optimizing Process Economics Online Using Model Predictive Control,” *Computers & Chemical Engineering*, vol. 58, pp. 334–343, Nov. 2013. doi: [10.1016/j.compchemeng.2013.07.015](https://doi.org/10.1016/j.compchemeng.2013.07.015) (see p. 99)
- [172] J. Hals, J. Falnes, and T. Moan, “Constrained Optimal Control of a Heaving Buoy Wave-Energy Converter,” *Journal of Offshore Mechanics and Arctic Engineering*, vol. 133, no. 1, p. 011401, Feb. 2011. doi: [10.1115/1.4001431](https://doi.org/10.1115/1.4001431) (see pp. 99, 102, 104)

- [173] P. Williams, “Hermite-Legendre-Gauss-Lobatto Direct Transcription in Trajectory Optimization,” *Journal of Guidance, Control, and Dynamics*, vol. 32, no. 4, pp. 1392–1395, Jul. 2009. doi: [10.2514/1.42731](https://doi.org/10.2514/1.42731) (see p. 99)
- [174] C. Hwang, D.-H. Shih, and F.-C. Kung, “Use of Block-Pulse Functions in the Optimal Control of Deterministic Systems,” *International Journal of Control*, vol. 44, no. 2, pp. 343–349, Aug. 1986. doi: [10.1080/00207178608933603](https://doi.org/10.1080/00207178608933603) (see p. 99)
- [175] K. T. Elgindy and K. A. Smith-Miles, “Fast, Accurate, and Small-Scale Direct Trajectory Optimization Using a Gegenbauer Transcription Method,” *Journal of Computational and Applied Mathematics*, vol. 251, pp. 93–116, Oct. 2013. doi: [10.1016/j.cam.2013.03.032](https://doi.org/10.1016/j.cam.2013.03.032) (see p. 99)
- [176] G. Bacelli and J. V. Ringwood, “Numerical Optimal Control of Wave Energy Converters,” *IEEE Transactions on Sustainable Energy*, vol. 6, no. 2, pp. 294–302, Apr. 2015. doi: [10.1109/tste.2014.2371536](https://doi.org/10.1109/tste.2014.2371536) (see p. 99)
- [177] C. C. Françolin, D. A. Benson, W. W. Hager, *et al.*, “Costate Approximation in Optimal Control Using Integral Gaussian Quadrature Orthogonal Collocation Methods,” *Optimal Control Applications and Methods*, vol. 36, no. 4, pp. 381–397, Feb. 2014. doi: [10.1002/oca.2112](https://doi.org/10.1002/oca.2112) (see pp. 99, 133)
- [178] P. Williams, “Application of Pseudospectral Methods for Receding Horizon Control,” *Journal of Guidance, Control, and Dynamics*, vol. 27, no. 2, pp. 310–314, Mar. 2004. doi: [10.2514/1.5118](https://doi.org/10.2514/1.5118) (see p. 100)
- [179] M. T. Heath, *Scientific Computing: An Introductory Survey*, 2nd ed. McGraw Hill, 2002, isbn: 978-0072399103 (see pp. 103–105)
- [180] L. N. Trefethen, “Is Gauss Quadrature Better than Clenshaw–Curtis?” *SIAM Review*, vol. 50, no. 1, pp. 67–87, Jan. 2008. doi: [10.1137/060659831](https://doi.org/10.1137/060659831) (see p. 103)
- [181] Q. Gong, I. M. Ross, and F. Fahroo, “Costate Computation by a Chebyshev Pseudospectral Method,” *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 2, pp. 623–628, Mar. 2010. doi: [10.2514/1.45154](https://doi.org/10.2514/1.45154) (see p. 103)
- [182] The MathWorks. kron: Kronecker Tensor Product, url: <https://www.mathworks.com/help/matlab/ref/kron.html> (see pp. 112, 241)
- [183] D. R. Herber. Approximating an Elliptical Region with Linear Constraints, Mathworks File Exchange, url: <https://www.mathworks.com/matlabcentral/fileexchange/56519> (visited on 10/12/2017) (see p. 117)
- [184] Q. Wang and J. S. Arora, “Several Simultaneous Formulations for Transient Dynamic Response Optimization: An Evaluation,” *International Journal for Numerical Methods in Engineering*, vol. 80, no. 5, pp. 631–650, Oct. 2009. doi: [10.1002/nme.2655](https://doi.org/10.1002/nme.2655) (see p. 118)
- [185] The MathWorks. quadprog: Quadratic Programming, url: <https://www.mathworks.com/help/optim/ug/quadprog.html> (visited on 10/03/2017) (see pp. 119, 176)
- [186] D. R. Herber, Y. H. Lee, and J. T. Allison. DT QP Project, GitHub, url: <https://github.com/danielrherber/dt-qp-project> (see pp. 119, 136, 199, 263)
- [187] A. E. Bryson, W. F. Denham, and S. E. Dreyfus, “Optimal Programming Problems with Inequality Constraints I: Necessary Conditions for Extremal Solutions,” *AIAA Journal*, vol. 1, no. 11, pp. 2544–2550, Nov. 1963. doi: [10.2514/3.2107](https://doi.org/10.2514/3.2107) (see p. 122)
- [188] C. L. Darby, “hp-Pseudospectral Method for Solving Continuous-Time Nonlinear Optimal Control Problems,” PhD dissertation, University of Florida, Gainesville, FL, USA, 2009. url: <http://ufdc.ufl.edu/UFE0042778/00001> (see p. 124)

- [189] C. L. Darby, W. W. Hager, and A. V. Rao, “An hp -Adaptive Pseudospectral Method for Solving Optimal Control Problems,” *Optimal Control Applications and Methods*, vol. 32, no. 4, pp. 476–502, 2011. doi: [10.1002/oca.957](https://doi.org/10.1002/oca.957) (see pp. 124, 133)
- [190] C. Darby and A. Rao, “A State Approximation-Based Mesh Refinement Algorithm for Solving Optimal Control Problems Using Pseudospectral Methods,” in *AIAA Guidance, Navigation, and Control Conference*, AIAA 2009-5791, Chicago, IL, USA: American Institute of Aeronautics and Astronautics, Aug. 2009. doi: [10.2514/6.2009-5791](https://doi.org/10.2514/6.2009-5791) (see pp. 132, 133)
- [191] Q. Gong, F. Fahroo, and I. M. Ross, “Spectral Algorithm for Pseudospectral Methods in Optimal Control,” *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 3, pp. 460–471, May 2008. doi: [10.2514/1.32908](https://doi.org/10.2514/1.32908) (see p. 132)
- [192] T. Fujikawa and T. Tsuchiya, “Enhanced Mesh Refinement in Numerical Optimal Control Using Pseudospectral Methods,” *SICE Journal of Control, Measurement, and System Integration*, vol. 7, no. 3, pp. 159–167, May 2014. doi: [10.9746/jcmsi.7.159](https://doi.org/10.9746/jcmsi.7.159) (see pp. 132, 133)
- [193] J. Zhao and S. Li, “Modified Multiresolution Technique for Mesh Refinement in Numerical Optimal Control,” *Journal of Guidance, Control, and Dynamics*, 2017. doi: [10.2514/1.g002796](https://doi.org/10.2514/1.g002796) (see p. 133)
- [194] J. T. Betts, N. Biehn, S. L. Campbell, et al., “Compensating for Order Variation in Mesh Refinement for Direct Transcription Methods,” *Journal of Computational and Applied Mathematics*, vol. 125, no. 1–2, pp. 147–158, Dec. 2000. doi: [10.1016/s0377-0427\(00\)00465-9](https://doi.org/10.1016/s0377-0427(00)00465-9) (see p. 133)
- [195] S. Jain and P. Tsotras, “Trajectory Optimization Using Multiresolution Techniques,” *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 5, pp. 1424–1436, Sep. 2008. doi: [10.2514/1.32220](https://doi.org/10.2514/1.32220) (see p. 133)
- [196] M. A. Patterson, W. W. Hager, and A. V. Rao, “A ph Mesh Refinement Method for Optimal Control,” *Optimal Control Applications and Methods*, vol. 36, no. 4, pp. 398–421, Jul. 2015. doi: [10.1002/oca.2114](https://doi.org/10.1002/oca.2114) (see p. 133)
- [197] C. L. Darby, D. Garg, and A. V. Rao, “Costate Estimation using Multiple-Interval Pseudospectral Methods,” *Journal of Spacecraft and Rockets*, vol. 48, no. 5, pp. 856–866, Sep. 2011. doi: [10.2514/1.a32040](https://doi.org/10.2514/1.a32040) (see p. 133)
- [198] M. Schori, T. J. Boehme, T. Jeinsch, et al., “Costate Approximation from Direct Methods for Switched Systems with State Jumps,” in *European Control Conference*, 7330583, Linz, Austria, Jul. 2015. doi: [10.1109/ecc.2015.7330583](https://doi.org/10.1109/ecc.2015.7330583) (see p. 133)
- [199] E. M. Gertz and S. J. Wright, “Object-Oriented Software for Quadratic Programming,” *ACM Transactions on Mathematical Software*, vol. 29, no. 1, pp. 58–81, Mar. 2003. doi: [10.1145/641876.641880](https://doi.org/10.1145/641876.641880) (see p. 134)
- [200] M. C. Grant and S. P. Boyd, “Graph Implementations for Nonsmooth Convex Programs,” in *Lecture Notes in Control and Information Sciences*, ed. by V. Blondel, S. Boyd, and H. Kimura. Springer, 2008, vol. 371, pp. 95–110. doi: [10.1007/978-1-84800-155-8_7](https://doi.org/10.1007/978-1-84800-155-8_7) (see p. 134)
- [201] J. J. Torsti and A. M. Aurela, “A Fast Quadratic Programming Method for Solving Ill-Conditioned Systems of Equations,” *Journal of Mathematical Analysis and Applications*, vol. 38, no. 1, pp. 193–204, Apr. 1972. doi: [10.1016/0022-247x\(72\)90127-8](https://doi.org/10.1016/0022-247x(72)90127-8) (see p. 134)
- [202] N. I. M. Gould, “Iterative Methods for Ill-Conditioned Linear Systems from Optimization,” in *Applied Optimization*. Springer, 2000, vol. 36, pp. 123–141. doi: [10.1007/978-1-4757-3226-9_7](https://doi.org/10.1007/978-1-4757-3226-9_7) (see p. 134)
- [203] B. O’Donoghue, G. Stathopoulos, and S. Boyd, “A Splitting Method for Optimal Control,” *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2432–2442, Nov. 2013. doi: [10.1109/tcst.2012.2231960](https://doi.org/10.1109/tcst.2012.2231960) (see p. 134)

- [204] E. Ghadimi, A. Teixeira, I. Shames, *et al.*, “Optimal Parameter Selection for the Alternating Direction Method of Multipliers (ADMM): Quadratic Problems,” *IEEE Transactions on Automatic Control*, vol. 60, no. 3, pp. 644–658, Mar. 2015. doi: [10.1109/tac.2014.2354892](https://doi.org/10.1109/tac.2014.2354892) (see p. 134)
- [205] L.-L. Wang, M. D. Samson, and X. Zhao, “A Well-Conditioned Collocation Method Using a Pseudospectral Integration Matrix,” *SIAM Journal on Scientific Computing*, vol. 36, no. 3, A907–A929, 2014. doi: [10.1137/130922409](https://doi.org/10.1137/130922409) (see p. 134)
- [206] L. N. Trefethen and M. R. Trummer, “An Instability Phenomenon in Spectral Methods,” *SIAM Journal on Numerical Analysis*, vol. 24, no. 5, pp. 1008–1023, 1987. doi: [10.1137/0724066](https://doi.org/10.1137/0724066) (see p. 134)
- [207] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. Academic Press, 1981, isbn: 978-0122839528 (see p. 134)
- [208] A. M.-C. So, “Semidefinite Optimization Applications,” in *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, 2011. doi: [10.1002/9780470400531.eorms0755](https://doi.org/10.1002/9780470400531.eorms0755) (see p. 135)
- [209] C. Sun and R. Dai. (2016). An Iterative Method for Nonconvex Quadratically Constrained Quadratic Programs. Submitted to IEEE Transactions on Automatic Control, url: <https://arxiv.org/abs/1609.02609> (see p. 135)
- [210] J. B. Grimbleby, “Automatic Analogue Network Synthesis Using Genetic Algorithms,” in *Genetic Algorithms in Engineering Systems: Innovations and Applications*, IET, Sep. 1995. doi: [10.1049/cp:19951024](https://doi.org/10.1049/cp:19951024) (see pp. 137, 138, 148–150, 153, 161, 162)
- [211] A. Das and R. Vemuri, “An Automated Passive Analog Circuit Synthesis Framework using Genetic Algorithms,” in *Computer Society Annual Symposium on VLSI*, IEEE, 2007. doi: [10.1109/isvlsi.2007.22](https://doi.org/10.1109/isvlsi.2007.22) (see pp. 137, 138, 141, 145, 148, 154, 156, 158, 161, 162)
- [212] O. Mitea, M. Meissner, L. Hedrich, *et al.*, “Automated Constraint-Driven Topology Synthesis for Analog Circuits,” in *Design, Automation & Test in Europe Conference & Exhibition*, IEEE, 2011. doi: [10.1109/date.2011.5763264](https://doi.org/10.1109/date.2011.5763264) (see p. 137)
- [213] J. B. Grimbleby, “Automatic Analogue Circuit Synthesis Using Genetic Algorithms,” *IEE Proceedings - Circuits, Devices and Systems*, vol. 147, no. 6, pp. 319–323, Dec. 2000. doi: [10.1049/ip-cds:20000770](https://doi.org/10.1049/ip-cds:20000770) (see pp. 137, 138, 148, 154, 161)
- [214] G. Sussman and R. Stallman, “Heuristic Techniques in Computer-Aided Circuit Analysis,” *IEEE Transactions on Circuits and Systems*, vol. 22, no. 11, pp. 857–865, Nov. 1975. doi: [10.1109/tcs.1975.1083985](https://doi.org/10.1109/tcs.1975.1083985) (see p. 137)
- [215] R. Harjani, R. A. Rutenbar, and L. R. Carley, “OASYS: A Framework for Analog Circuit Synthesis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 12, pp. 1247–1266, Dec. 1989. doi: [10.1109/43.44506](https://doi.org/10.1109/43.44506) (see p. 137)
- [216] A. Das and R. Vemuri, “Topology Synthesis of Analog Circuits Based on Adaptively Generated Building Blocks,” in *Design Automation Conference*, Anaheim, CA, USA, Jun. 2008 (see pp. 138, 148)
- [217] C. Goh and Y. Li, “GA Automated Design and Synthesis of Analog Circuits with Practical Constraints,” in *Congress on Evolutionary Computation*, IEEE, May 2001. doi: [10.1109/cec.2001.934386](https://doi.org/10.1109/cec.2001.934386) (see pp. 138, 145, 146, 148, 154, 156, 160–162)
- [218] J. R. Koza, F. H. Bennett, D. Andre, *et al.*, “Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 2, pp. 109–128, Jul. 1997. doi: [10.1109/4235.687879](https://doi.org/10.1109/4235.687879) (see pp. 138, 145, 154)

- [219] E. S. Ochotta, R. A. Rutenbar, and L. R. Carley, “Synthesis of High-performance Analog Circuits in ASTRX/OBLX,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 3, pp. 273–294, Mar. 1996. doi: [10.1109/43.489099](https://doi.org/10.1109/43.489099) (see p. 138)
- [220] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, 1st ed. Springer, 2003, isbn: 978-3642072857. doi: [10.1007/978-3-662-05094-1](https://doi.org/10.1007/978-3-662-05094-1) (see pp. 138, 196)
- [221] F. Brucolieri, E. A. M. Klumperink, and B. Nauta, “Generating All Two-MOS-transistor Amplifiers Leads to New Wide-band LNAs,” *IEEE Journal of Solid-State Circuits*, vol. 36, no. 7, pp. 1032–1040, Jul. 2001. doi: [10.1109/4.933458](https://doi.org/10.1109/4.933458) (see p. 139)
- [222] E. A. M. Klumperink, “Transconductance Based CMOS Circuits: Circuit Generation, Classification and Analysis,” PhD dissertation, Universiteit Twente, Enschede, Netherlands, Mar. 1997 (see p. 139)
- [223] *Handbook of Graph Theory*, 2nd ed., ed. by J. L. Gross, J. Yellen, and P. Zhang. CRC Press, 2014, isbn: 978-1439880180 (see p. 141)
- [224] M. Meissner, O. Mitea, L. Luy, *et al.*, “Fast Isomorphism Testing for a Graph-based Analog Circuit Synthesis Framework,” in *Design, Automation & Test in Europe Conference & Exhibition*, Mar. 2012. doi: [10.1109/date.2012.6176570](https://doi.org/10.1109/date.2012.6176570) (see p. 142)
- [225] E. Cheever. Symbolic Circuit Analysis in Matlab, Mathworks File Exchange, url: <https://www.mathworks.com/matlabcentral/fileexchange/3443> (see p. 146)
- [226] The MathWorks. lsqnonlin: Solve Nonlinear Least-Squares (Nonlinear Data-Fitting) Problems, url: <https://www.mathworks.com/help/optim/ug/lsqnonlin.html> (see p. 147)
- [227] E. C. Levy, “Complex-Curve Fitting,” *IRE Transactions on Automatic Control*, vol. AC-4, no. 1, pp. 37–43, 1959. doi: [10.1109/tac.1959.6429401](https://doi.org/10.1109/tac.1959.6429401) (see pp. 148, 151)
- [228] J. R. F. Arruda, “Objective Functions for the Nonlinear Curve Fit of Frequency Response Functions,” *AIAA Journal*, vol. 30, no. 3, pp. 855–857, Mar. 1992. doi: [10.2514/3.11001](https://doi.org/10.2514/3.11001) (see p. 148)
- [229] J. W. Chinneck, *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, 1st ed. Springer, 2008, isbn: 978-0387749310. doi: [10.1007/978-0-387-74932-7](https://doi.org/10.1007/978-0-387-74932-7) (see p. 148)
- [230] R. Pintelon, P. Guillaume, Y. Rolain, *et al.*, “Parametric Identification of Transfer Functions in the Frequency Domain-A Survey,” *IEEE Transactions on Automatic Control*, vol. 39, no. 11, pp. 2245–2260, 1994. doi: [10.1109/9.333769](https://doi.org/10.1109/9.333769) (see p. 148)
- [231] R. Martí, “Multi-Start Methods,” in *Handbook of Metaheuristics*, G. F. and K. G. A., Eds. Springer, 2003, vol. 57, pp. 355–368, isbn: 978-1402072635. doi: [10.1007/0-306-48056-5_12](https://doi.org/10.1007/0-306-48056-5_12) (see pp. 148, 196)
- [232] D. R. Herber. PM Circuits, GitHub, url: <https://github.com/danielrherber/pm-circuits> (see pp. 149, 263)
- [233] L. Wanhammar, *Analog Filters using MATLAB*, 1st ed. Springer, 2009, isbn: 978-0387927664. doi: [10.1007/978-0-387-92767-1](https://doi.org/10.1007/978-0-387-92767-1) (see pp. 154, 158)
- [234] J. R. Koza, F. H. Bennett III, D. Andre, *et al.*, “Synthesis of Topology and Sizing of Analog Electrical Circuits by Means of Genetic Programming,” *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2-4, pp. 459–482, Jun. 2000. doi: [10.1016/s0045-7825\(99\)00397-7](https://doi.org/10.1016/s0045-7825(99)00397-7) (see p. 154)
- [235] T. Guo, D. J. Lohan, R. Cang, *et al.*, “An Indirect Design Representation for Topology Optimization Using Variational Autoencoder and Style Transfer,” in *AIAA 2018 Science and Technology Forum and Exposition*, to appear, Kissimmee, FL, USA, Jan. 2018 (see pp. 162, 208)
- [236] A. Jha. (Jun. 2009). Science Weekly with Michio Kaku: Impossibility is Relative, The Guardian, url: <https://www.theguardian.com/science/audio/2009/jun/11/michio-kaku-physics-impossible> (visited on 10/30/2017) (see p. 164)

- [237] S. Arnon, S. Rotman, and N. S. Kopeika, "Optimum Transmitter Optics Aperture for Satellite Optical Communication," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 34, no. 2, pp. 590–596, Apr. 1998. doi: [10.1109/7.670339](https://doi.org/10.1109/7.670339) (see p. 164)
- [238] A. H. de Ruiter, C. Damaren, and J. R. Forbes, *Spacecraft Dynamics and Control: An Introduction*, 1st ed. John Wiley & Sons, 2013, isbn: 978-1118342367 (see p. 164)
- [239] S. W. Sirlin, "Vibration Isolation for Spacecraft Using the Piezoelectric Polymer PVF₂," *The Journal of the Acoustical Society of America*, vol. 82, no. S13, Nov. 1987. doi: [10.1121/1.2024666](https://doi.org/10.1121/1.2024666) (see p. 164)
- [240] S. S. Rao and M. Sunar, "Piezoelectricity and Its Use in Disturbance Sensing and Control of Flexible Structures: A Survey," *Applied Mechanics Reviews*, vol. 47, no. 7, pp. 113–123, Apr. 1994. doi: [10.1115/1.3111074](https://doi.org/10.1115/1.3111074) (see pp. 164, 165)
- [241] P. H. Meckl and R. Kinzel, "Robust Motion Control of Flexible Systems Using Feedforward Forcing Functions," *IEEE Transactions on Control Systems Technology*, vol. 2, no. 3, pp. 245–254, Sep. 1994. doi: [10.1109/87.317981](https://doi.org/10.1109/87.317981) (see p. 165)
- [242] E. F. Crawley and J. D. Luis, "Use of Piezoelectric Actuators as Elements of Intelligent Structures," *AIAA Journal*, vol. 25, no. 10, pp. 1373–1385, Oct. 1987. doi: [10.2514/3.9792](https://doi.org/10.2514/3.9792) (see p. 165)
- [243] R. A. Manning, "Optimum Design of Intelligent Truss Structures," in *Structures, Structural Dynamics, and Materials Conference, Structures, Structural Dynamics, and Materials*, Baltimore, MA, USA, Apr. 1991, pp. 528–533. doi: [10.2514/6.1991-1158](https://doi.org/10.2514/6.1991-1158) (see p. 165)
- [244] J. Pan, C. H. Hansen, and S. D. Snyder, "A Study of the Response of a Simply Supported Beam to Excitation by a Piezoelectric Actuator," *Journal of Intelligent Material Systems and Structures*, vol. 3, no. 1, pp. 3–16, Jan. 1992. doi: [10.1177/1045389X9200300101](https://doi.org/10.1177/1045389X9200300101) (see p. 165)
- [245] J. E. Hubbard Jr. and S. E. Burke, "Distributed Transducer Design for Intelligent Structural Components," in *Intelligent Structural Systems*, H. S. Tzou and G. L. Anderson, Eds., 1st ed. Springer, 1992, vol. 13, pp. 305–324, isbn: 978-9048141920. doi: [10.1007/978-94-017-1903-2_8](https://doi.org/10.1007/978-94-017-1903-2_8) (see p. 165)
- [246] W. Hwang and H. C. Park, "Finite Element Modeling of Piezoelectric Sensors and Actuators," *AIAA Journal*, vol. 31, no. 5, pp. 930–937, May 1993. doi: [10.2514/3.11707](https://doi.org/10.2514/3.11707) (see p. 165)
- [247] O. S. Alvarez-Salazar and K. Iliff, "Destabilizing Effects of Rate Feedback on Strain Actuated Beams," *Journal of Sound and Vibration*, vol. 221, no. 2, pp. 289–307, Mar. 1999. doi: [10.1006/jsvi.1998.2010](https://doi.org/10.1006/jsvi.1998.2010) (see p. 165)
- [248] T. Bailey and J. E. Hubbard Jr., "Distributed Piezoelectric-Polymer Active Vibration Control of a Cantilever Beam," *Journal of Guidance, Control, and Dynamics*, vol. 8, no. 5, pp. 605–611, Sep. 1985. doi: [10.2514/3.20029](https://doi.org/10.2514/3.20029) (see p. 165)
- [249] E. K. Dimitriadis, C. R. Fuller, and C. A. Rogers, "Piezoelectric Actuators for Distributed Vibration Excitation of Thin Plates," *Journal of Vibration and Acoustics*, vol. 113, no. 1, pp. 100–107, Jan. 1991. doi: [10.1115/1.2930143](https://doi.org/10.1115/1.2930143) (see p. 165)
- [250] J. L. Pinkerton, A.-M. R. McGowan, R. W. Moses, *et al.*, "Controlled Aeroelastic Response and Airfoil Shaping Using Adaptive Materials and Integrated Systems," in *Symposium on Smart Structures and Materials*, San Diego, CA, USA, Feb. 1996, pp. 166–177 (see p. 165)
- [251] R. M. Fowler, "Investigation of Compliant Space Mechanisms with Application to the Design of a Large-Displacement Monolithic Compliant Rotational Hinge," Master's thesis, Brigham Young University, Provo, UT, USA, Aug. 2012. url: <http://hdl.lib.byu.edu/1877/etd5391> (see p. 165)
- [252] J. L. Fanson and T. K. Caughey, "Positive Position Feedback Control for Large Space Structures," *AIAA Journal*, vol. 28, no. 4, pp. 717–724, Apr. 1990. doi: [10.2514/3.10451](https://doi.org/10.2514/3.10451) (see p. 165)

- [253] W. P. Li and H. Huang, “Integrated Optimization of Actuator Placement and Vibration Control for Piezoelectric Adaptive Trusses,” *Journal of Sound and Vibration*, vol. 332, no. 1, pp. 17–32, Jan. 2013. doi: [10.1016/j.jsv.2012.08.005](https://doi.org/10.1016/j.jsv.2012.08.005) (see p. 165)
- [254] M. J. Smith, K. M. Grigoriadis, and R. E. Skelton, “Optimal Mix of Passive and Active Control in Structures,” *Journal of Guidance, Control, and Dynamics*, vol. 15, no. 4, pp. 912–919, 1992. doi: [10.2514/3.20924](https://doi.org/10.2514/3.20924) (see pp. 165, 166)
- [255] R. V. Grandhi, “Structural and Control Optimization of Space Structures,” *Computers & Structures*, vol. 31, no. 2, pp. 139–150, 1989. doi: [10.1016/0045-7949\(89\)90222-8](https://doi.org/10.1016/0045-7949(89)90222-8) (see p. 165)
- [256] D. R. Herber, J. W. McDonald, O. S. Alvarez-Salazar, *et al.*, “Reducing Spacecraft Jitter During Satellite Reorientation Maneuvers via Solar Array Dynamics,” in *AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Atlanta, GA, USA, Jun. 2014. doi: [10.2514/6.2014-3278](https://doi.org/10.2514/6.2014-3278) (see p. 166)
- [257] O. S. Alvarez-Salazar, J. B. Aldrich, N. Filipe, *et al.*, “Strain Actuated Solar Arrays for Precision Pointing of Spacecraft,” in *AAS Guidance, Navigation, and Control Conference*, Breckenridge, CO, USA, Feb. 2016 (see p. 166)
- [258] Y.-Q. Yu, L. L. Howell, C. Lusk, *et al.*, “Dynamic Modeling of Compliant Mechanisms Based on the Pseudo-Rigid-Body Model,” *Journal of Mechanical Design*, vol. 127, no. 4, pp. 760–765, Aug. 2005. doi: [10.1115/1.1900750](https://doi.org/10.1115/1.1900750) (see p. 166)
- [259] C. M. Chilan, D. R. Herber, Y. K. Nakka, *et al.*, “Co-Design of Strain-Actuated Solar Arrays for Precision Pointing and Jitter Reduction,” in *Science and Technology Forum and Exposition*, San Diego, CA, USA, Jan. 2016. doi: [10.2514/6.2016-0162](https://doi.org/10.2514/6.2016-0162) (see pp. 166, 176)
- [260] R. L. Bisplinghoff, H. Ashley, and R. L. Halfman, *Aeroelasticity*, 1st ed. Dover, 1996, isbn: 978-0486691893 (see p. 166)
- [261] J. L. Junkins and Y. Kim, *Introduction to Dynamics and Control of Flexible Structures*, 1st ed. AIAA, 1993, isbn: 9781563470547. doi: [10.2514/4.862076](https://doi.org/10.2514/4.862076) (see pp. 166, 170)
- [262] A. A. Paranjape, J. Guan, S.-J. Chung, *et al.*, “PDE Boundary Control for Flexible Articulated Wings on a Robotic Aircraft,” *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 625–640, Jun. 2013. doi: [10.1109/TRO.2013.2240711](https://doi.org/10.1109/TRO.2013.2240711) (see pp. 166, 170)
- [263] A. A. Paranjape, S.-J. Chung, H. H. Hilton, *et al.*, “Dynamics and Performance of Tailless Micro Aerial Vehicle with Flexible Articulated Wings,” *AIAA Journal*, vol. 50, no. 5, pp. 1177–1188, May 2012. doi: [10.2514/1.J051447](https://doi.org/10.2514/1.J051447) (see pp. 167, 170)
- [264] Y. K. Nakka, S.-J. Chung, J. T. Allison, *et al.*, “Nonlinear ODE-PDE Control of Strain Actuated Solar Arrays for High Precision Spacecraft Attitude Control,” *Journal of Guidance, Control, and Dynamics (to be submitted)*, (see pp. 167, 168, 170, 172)
- [265] S. O. M. Moheimani and A. J. Fleming, *Piezoelectric Transducers for Vibration Control and Damping*, 1st ed. Springer, 2006, isbn: 978-1846283314. doi: [10.1007/1-84628-332-9](https://doi.org/10.1007/1-84628-332-9) (see pp. 170, 172)
- [266] J. Lindmayer and W. C.Y., “Development of a High Efficiency Thin Silicon Solar Cell,” Solarex Corporation, Rockville, MD, USA, Tech. Rep. NASA-CR-157078, SX/105/F, Sep. 1977 (see p. 175)
- [267] J. Lee, J. Wu, M. Shi, *et al.*, “Stretchable GaAs Photovoltaics With Designs That Enable High Areal Coverage,” *Advanced Materials*, vol. 23, no. 8, pp. 986–991, Feb. 2011. doi: [10.1002/adma.201003961](https://doi.org/10.1002/adma.201003961) (see p. 175)
- [268] L. Qiu, S. He, J. Yang, *et al.*, “Fiber-Shaped Perovskite Solar Cells With High Power Conversion Efficiency,” *Small*, vol. 12, no. 18, pp. 2419–2424, May 2016. doi: [10.1002/smll.201600326](https://doi.org/10.1002/smll.201600326) (see p. 175)

- [269] P. J. Enright and B. A. Conway, “Discrete Approximations to Optimal Trajectories Using Direct Transcription and Nonlinear Programming,” *Journal of Guidance, Control, and Dynamics*, vol. 15, no. 4, pp. 994–1002, 1992. doi: [10.2514/3.20934](https://doi.org/10.2514/3.20934) (see p. 176)
- [270] C. M. Chilan and B. A. Conway, “Automated Design of Multiphase Space Missions Using Hybrid Optimal Control,” *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 5, pp. 1410–1424, Sep. 2013. doi: [10.2514/1.58766](https://doi.org/10.2514/1.58766) (see p. 176)
- [271] D. Morgan, S.-J. Chung, and F. Y. Hadaegh, “Model Predictive Control of Swarms of Spacecraft Using Sequential Convex Programming,” *Journal of Guidance, Control, and Dynamics*, vol. 37, no. 6, pp. 1725–1740, Nov. 2014. doi: [10.2514/1.G000218](https://doi.org/10.2514/1.G000218) (see p. 176)
- [272] The MathWorks. patternsearch: Find Minimum of Function Using Pattern Search, url: <https://www.mathworks.com/help/gads/patternsearch.html> (visited on 10/29/2017) (see p. 176)
- [273] D. Luzeaux, J.-R. Ruault, and J.-L. Wippler, Eds., *Large Scale Complex Systems and Systems of Systems Engineering: Case Studies*. Wiley, 2013, isbn: 978-1848212534. doi: [10.1002/9781118601495](https://doi.org/10.1002/9781118601495) (see p. 191)
- [274] P. Kasturi and P. Dupont, “Constrained Optimal Control of Vibration Dampers,” *Journal of Sound and Vibration*, vol. 215, no. 3, pp. 499–509, Aug. 1998. doi: [10.1006/jsvi.1998.1661](https://doi.org/10.1006/jsvi.1998.1661) (see p. 191)
- [275] D. Hrovat, “Survey of Advanced Suspension Developments and Related Optimal Control Applications,” *Automatica*, vol. 33, no. 10, pp. 1781–1817, Oct. 1997. doi: [10.1016/S0005-1098\(97\)00101-5](https://doi.org/10.1016/S0005-1098(97)00101-5) (see pp. 191, 192)
- [276] ——, “Applications of Optimal Control to Advanced Automotive Suspension Design,” *Journal of Dynamic Systems, Measurement, and Control*, vol. 115, no. 2B, pp. 328–342, Jun. 1993. doi: [10.1115/1.2899073](https://doi.org/10.1115/1.2899073) (see pp. 191–193, 198, 201)
- [277] M. Gobbi and G. Mastinu, “Analytical Description and Optimization of the Dynamic Behaviour of Passively Suspended Road Vehicles,” *Journal of Sound and Vibration*, vol. 245, no. 3, pp. 457–481, Aug. 2001. doi: [10.1006/jsvi.2001.3591](https://doi.org/10.1006/jsvi.2001.3591) (see pp. 191, 192, 198)
- [278] Y. He and J. McPhee, “Multidisciplinary Design Optimization of Mechatronic Vehicles with Active Suspensions,” *Journal of Sound and Vibration*, vol. 283, no. 1-2, pp. 217–241, May 2005. doi: [10.1016/j.jsv.2004.04.027](https://doi.org/10.1016/j.jsv.2004.04.027) (see pp. 191, 192)
- [279] J. T. Allison, “Optimal Partitioning and Coordination Decisions in Decomposition-based Design Optimization,” PhD dissertation, The University of Michigan, Ann Arbor, MI, USA, May 2008. url: <http://hdl.handle.net/2027.42/58449> (see pp. 191, 192, 199)
- [280] G. Koch and T. Kloiber, “Driving State Adaptive Control of an Active Vehicle Suspension System,” *IEEE Transactions on Control Systems Technology*, vol. 22, no. 1, pp. 44–57, Jan. 2014. doi: [10.1109/TCST.2013.2240455](https://doi.org/10.1109/TCST.2013.2240455) (see p. 191)
- [281] A. Bourmistrova, I. Storey, and A. Subic, “Multiobjective Optimisation of Active and Semi-Active Suspension Systems with Application of Evolutionary Algorithm,” in *International Conference on Modelling and Simulation*, Melbourne, Australia, Dec. 2005, pp. 1217–1223 (see p. 191)
- [282] S. F. Alyaqout, P. Y. Papalambros, and A. G. Ulsoy, “Combined Design and Robust Control of a Vehicle Passive/Active Suspension,” in *European Control Conference*, Kos, Greece, Jul. 2007, pp. 1264–1270 (see pp. 191–193, 198)
- [283] A. G. Ulsoy, D. Hrovat, and T. Tseng, “Stability Robustness of LQ and LQG Active Suspensions,” *Journal of Dynamic Systems, Measurement, and Control*, vol. 116, no. 1, pp. 123–131, Mar. 1994. doi: [10.1115/1.2900666](https://doi.org/10.1115/1.2900666) (see pp. 191, 192, 198)
- [284] J. A. Kypuros, *System Dynamics and Control with Bond Graph Modeling*. CRC Press, 2013, isbn: 978-1466560758 (see p. 193)

- [285] D. C. Karnopp, D. L. Margolis, and R. C. Rosenberg, *System Dynamics: Modeling, Simulation, and Control of Mechatronic Systems*, 5th ed. Wiley, 2012, isbn: 978-0470889084 (see p. 193)
- [286] G. Gonzalez and R. Galindo, “Removing the Algebraic Loops of a Bond Graph Model,” *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 222, no. 6, pp. 543–556, Sep. 2008. doi: [10.1243/09596518jsce559](https://doi.org/10.1243/09596518jsce559) (see p. 194)
- [287] H. A. Simon, *The Sciences of the Artificial*, 3rd ed. The MIT Press, 1996, isbn: 978-0262691918 (see p. 205)
- [288] J. J. Granda and J. Reus, “New Developments in Bond Graph Modeling Software Tools: The Computer Aided Modeling Program Camp-G and MATLAB,” in *IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, FL, USA, Oct. 1997. doi: [10.1109/ICSMC.1997.638215](https://doi.org/10.1109/ICSMC.1997.638215) (see p. 209)
- [289] C. Kleijn, M. A. Groothuis, and H. G. Differ, *20-sim 4.6 Reference Manual*, Controllab Products B.V., 2017 (see p. 209)
- [290] C. Sullivan. (2004). System Analogies. Handouts from ENGS 22, Systems, Dartmouth College, url: http://www.dartmouth.edu/~sullivan/22files/System_analogy_all.pdf (visited on 10/31/2017) (see p. 209)
- [291] The MathWorks. circshift, url: <https://www.mathworks.com/help/matlab/ref/circshift.html> (visited on 04/03/2017) (see p. 213)
- [292] ——, ndgrid, url: <https://www.mathworks.com/help/matlab/ref/ndgrid.html> (visited on 04/10/2017) (see p. 231)
- [293] S. S. Skiena, *The Algorithm Design Manual*, 2nd ed. Springer, 2008, isbn: 978-1848000698. doi: [10.1007/978-1-84800-070-4](https://doi.org/10.1007/978-1-84800-070-4) (see pp. 235, 236)
- [294] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *et al.*, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009, isbn: 978-0262033848 (see pp. 235, 236)
- [295] M. J. Quinn and N. Deo, “Parallel Graph Algorithms,” *Computing Surveys*, vol. 16, no. 3, pp. 319–348, Sep. 1984. doi: [10.1145/2514.2515](https://doi.org/10.1145/2514.2515) (see p. 237)
- [296] E. Rehbati and D. G. Corneil, “Parallel Computations in Graph Theory,” *SIAM Journal on Computing*, vol. 7, no. 2, pp. 230–237, May 1978. doi: [10.1137/0207020](https://doi.org/10.1137/0207020) (see p. 237)
- [297] The MathWorks. blkdiag: Construct Block Diagonal Matrix from Input Arguments, url: <https://www.mathworks.com/help/matlab/ref/blkdiag.html> (visited on 10/03/2017) (see p. 241)
- [298] D. R. Herber. Basic Implementation of Multiple-Interval Pseudospectral Methods to Solve Optimal Control Problems, GitHub, url: <https://github.com/danielrherber/basic-multiple-interval-pseudospectral> (see pp. 255–257, 263)
- [299] L. N. Trefethen, *Spectral Methods in MATLAB*, 1st ed. SIAM, 2000, isbn: 978-0898714654. doi: [10.1137/1.9780898719598](https://doi.org/10.1137/1.9780898719598) (see pp. 256, 257)
- [300] J. Waldvogel, “Fast Construction of the Fejér and Clenshaw–Curtis Quadrature Rules,” *BIT Numerical Mathematics*, vol. 46, no. 1, pp. 195–202, Mar. 2006. doi: [10.1007/s10543-006-0045-4](https://doi.org/10.1007/s10543-006-0045-4) (see p. 257)
- [301] D. R. Herber. Perfect Matchings of a Complete Graph, GitHub, url: <https://github.com/danielrherber/perfect-matchings-of-a-complete-graph> (see p. 263)
- [302] ——, Optimal Control Direct Method Examples, GitHub, url: <https://github.com/danielrherber/optimal-control-direct-method-examples> (see p. 263)
- [303] ——, PM Suspensions, GitHub, url: <https://github.com/danielrherber/pm-suspensions> (see p. 263)