

# A Novel Augmented Graph Approach for Estimation in Localisation and Mapping

Paul Robert Thompson

A thesis submitted in fulfillment  
of the requirements for the degree of  
Doctor of Philosophy



Australian Centre for Field Robotics  
School of Aerospace, Mechanical and Mechatronic Engineering  
The University of Sydney

March, 2009

# Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the University or other institute of higher learning, except where due acknowledgement has been made in the text.

**Paul Robert Thompson**

March, 2009

# Abstract

Paul Robert Thompson  
The University of Sydney

Doctor of Philosophy  
March, 2009

## A Novel Augmented Graph Approach for Estimation in Localisation and Mapping

This thesis proposes the use of the augmented system form - a generalisation of the information form representing both observations and states. In conjunction with this, this thesis proposes a novel graph representation for the estimation problem together with a graph based linear direct solving algorithm.

The augmented system form is a mathematical description of the estimation problem showing the states and observations. The augmented system form allows a more general range of factorisation orders among the observations and states, which is essential for constraints and is beneficial for sparsity and numerical reasons.

The proposed graph structure is a novel sparse data structure providing more symmetric access and faster traversal and modification operations than the compressed-sparse-column (CSC) sparse matrix format. The graph structure was developed as a fundamental underlying structure for the formulation of sparse estimation problems. This graph-theoretic representation replaces conventional sparse matrix representations for the estimation states, observations and their interconnections.

This thesis contributes a new implementation of the indefinite LDL factorisation algorithm based entirely in the graph structure. This direct solving algorithm was developed in order to exploit the above new approaches of this thesis. The factorisation operations consist of accessing adjacencies and modifying the graph edges. The developed solving algorithm demonstrates the significant differences in the form and approach of the graph-embedded algorithm compared to a conventional matrix implementation.

The contributions proposed in this thesis improve estimation methods by providing novel mathematical data structures used to represent states, observations and the sparse links between them. These offer improved flexibility and capabilities which are exploited in the solving algorithm. The contributions constitute a new framework for the development of future online and incremental solving, data association and analysis algorithms for online, large scale localisation and mapping.

# Acknowledgements

First of all, I would like to thank my supervisor, Salah Sukkarieh and co-supervisor, Hugh Durrant-Whyte. It's been a long journey, and I thank you both for your patience, encouragement and trust in my direction.

I would like to thank my friends at the ACFR for being alongside me through this journey, particularly Dave, Jason, Mitch, Sharon, Toby and Stewart. Thank you to those who came before me for passing on your wisdom - Eric, Tim B, Ian, Fabio, Grover, Alex, Alex and Alexei. In turn, to those who are following - good luck.

Thank you to Jeremy, Ali, Esa and Tim H for the experiences on the Brumby, and for your consistently high standards which motivated me to do my best.

To Dad, David, Lainie and Lisa, Andrew and Lynda, thank you for everything. Thank you for providing a loving family home environment to retreat to, and for growing with me through this.

To Mum, I dedicate this to you - this thesis is for both of us. Thank you for inspiring me and sharing the experience with me. I miss you every day.

To my second family, Frank, Helen, Ross and Jacqui, thank you for your welcoming, kind friendship.

Thank you to James, Brooke, Ben, Katherine, Andy, for your friendship and support.

Finally, a special thank you to Marcelle for being my constant companion though this time and for always. I trust and value your encouragement, you know me better than anyone. I love you and I want you to know that I truly appreciate your amazing support and I am eagerly looking forward to everything we will do and share together in the future.

*tenpo li tawa la sona li kama*

*For Bev 1953-2008*

# Contents

<b>Declaration</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>x</b>
<b>List of Tables</b>	<b>xiii</b>
<b>List of Examples</b>	<b>xiv</b>
<b>Nomenclature</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Thesis Contributions . . . . .	3
1.2 Motivation for Approaches . . . . .	6
1.3 Motivating Problem . . . . .	8
1.4 Thesis Structure . . . . .	10
<b>2 Estimation In Localisation and Mapping</b>	<b>12</b>
2.1 Localisation and Mapping Literature . . . . .	12
2.1.1 Smoothing and Mapping (SAM) . . . . .	16
2.1.2 Viewpoint based SLAM . . . . .	18

2.1.3	SLAM Filtering . . . . .	20
2.2	Graphical Models Literature . . . . .	21
2.3	Assumptions and Context . . . . .	22
2.4	Solving Overview . . . . .	25
2.4.1	Step Based Approach . . . . .	27
2.4.2	Solving Linear Systems . . . . .	28
2.5	Summary . . . . .	30
2.6	Graph Notation . . . . .	31
<b>3</b>	<b>Augmented Methods in Estimation</b>	<b>33</b>
3.1	Introduction . . . . .	33
3.2	Augmenting Observations and Constraints . . . . .	37
3.2.1	Information Formulation . . . . .	38
3.2.2	Lagrangian Formulation . . . . .	40
3.2.3	Constraints . . . . .	46
3.2.4	Mixed Observations and Constraints . . . . .	52
3.2.5	Equivalence to the Information Form: Eliminating Observations	54
3.2.6	Literature - Augmented System Form . . . . .	57
3.2.7	Nonlinear Observations . . . . .	60
3.2.8	Properties of the Augmented System Form . . . . .	61
3.2.9	Regularisation of the Augmented System Form . . . . .	63
3.3	Augmenting Trajectory States . . . . .	65
3.3.1	Formation of the Trajectory States . . . . .	66
3.3.2	Discussion . . . . .	68
3.3.3	Equivalence . . . . .	70
3.4	Relation To Graphical Models . . . . .	73
3.4.1	Relation to Factor Graphs . . . . .	74
3.4.2	What are the systems <i>before</i> conditioning on the observations?	76
3.5	Insights for Data Fusion . . . . .	78

3.6	Residuals & Innovations . . . . .	81
3.6.1	Innovations . . . . .	82
3.6.2	Residuals . . . . .	83
3.6.3	Discussion . . . . .	85
3.6.4	Multiple Observation Terms . . . . .	87
3.6.5	Chi-Squared Degrees of Freedom . . . . .	89
3.6.6	Lagrange Multipliers for Measurement of Consistency . . . . .	91
3.6.7	Conclusion . . . . .	93
3.7	Benefits for Estimation . . . . .	93
3.7.1	Factorisation Ordering for Sparsity . . . . .	94
3.7.2	Factorisation Ordering for Numerical Stability . . . . .	118
3.7.3	Handling Nonlinear Observations . . . . .	122
3.7.4	Conclusion . . . . .	123
3.8	Future Research . . . . .	123
3.9	Chapter Conclusion . . . . .	123
<b>4</b>	<b>Graph Theoretic Representation</b>	<b>126</b>
4.1	Introduction . . . . .	126
4.2	Literature . . . . .	130
4.3	Graph Representation of Linear Systems . . . . .	133
4.3.1	Dense Vectors . . . . .	134
4.3.2	Matrix Entries . . . . .	136
4.3.3	Sparse Vectors . . . . .	138
4.3.4	Matrix Categories . . . . .	139
4.3.5	Discussion . . . . .	140
4.4	Graph Representation . . . . .	141
4.4.1	Edges and Loops . . . . .	142
4.4.2	Symmetric and Directed Edges . . . . .	142
4.4.3	Multiple Edge Sets . . . . .	144



4.4.4	Discussion and Conclusion . . . . .	145
4.5	Graph Representation Implementation . . . . .	146
4.5.1	Edges . . . . .	147
4.5.2	Loops . . . . .	148
4.5.3	Multiple Edge-Sets . . . . .	149
4.5.4	Vertices . . . . .	149
4.5.5	Graph . . . . .	151
4.5.6	Ordering Properties . . . . .	152
4.5.7	Examples . . . . .	153
4.6	Comparisons . . . . .	153
4.6.1	Qualitative Comparison . . . . .	153
4.6.2	Insertion Test . . . . .	156
4.6.3	Access Test . . . . .	163
4.7	Future Research . . . . .	167
4.8	Conclusion . . . . .	169
<b>5</b>	<b>Graph-Theoretic Solution Methods</b>	<b>171</b>
5.1	Introduction . . . . .	171
5.2	Symmetric LDL Factorisation Introduction . . . . .	172
5.2.1	LDL Factorisation, Mathematical Form . . . . .	176
5.2.2	LDL Factorisation, Dense Matrix Form . . . . .	181
5.3	Symmetric LDL Factorisation - Graph Form . . . . .	183
5.3.1	Graph Based Factorisation Steps . . . . .	183
5.3.2	Graph Based Linear Algebra Procedures . . . . .	189
5.4	Linear Systems Solve Using the LDL Factorisation . . . . .	195
5.4.1	Graph Based Block Diagonal Solve . . . . .	198
5.4.2	Graph Based Triangular Solve . . . . .	198
5.4.3	Graph Based Solve Implementation . . . . .	200
5.5	Reconstruction From LDL Factorisation . . . . .	207

5.6	Discussion . . . . .	209
5.6.1	Relation to the Junction Tree Algorithm . . . . .	209
5.7	Future Research . . . . .	211
5.7.1	Factorisation Approach . . . . .	211
5.7.2	Factorisation Ordering Choice . . . . .	212
5.8	Chapter Conclusion . . . . .	217
<b>6</b>	<b>Conclusion and Future Research</b>	<b>219</b>
6.1	Summary of Contributions . . . . .	219
6.2	Future Research . . . . .	221
6.2.1	Online Methods . . . . .	221
6.2.2	Iterative Methods . . . . .	223
6.2.3	Data Association Methods . . . . .	224
6.2.4	Decentralisation . . . . .	225
6.3	Conclusion . . . . .	226
<b>A</b>	<b>Augmented System Details</b>	<b>227</b>
A.1	Eliminating States . . . . .	227
A.2	Terms Relating to Residuals and Innovations . . . . .	230
A.3	Quadratic Forms and Mahalanobis Distances . . . . .	231
A.4	Linear Systems . . . . .	232
A.5	Proof of Equivalence of Residual and Innovation Distances . . . . .	233
	<b>Bibliography</b>	<b>245</b>

# List of Figures

1.1	Thesis outline (subsets of this thesis) . . . . .	2
1.2	Thesis outlook (supersets of this thesis) . . . . .	2
2.1	SLAM frameworks . . . . .	14
2.2	SLAM frameworks (detail) . . . . .	15
2.3	Major components of the optimisation algorithm. . . . .	27
2.4	Graph notation for example systems . . . . .	32
3.1	Two views of contours of the Lagrangian surface. The solution in $\mathbf{x}$ and $\boldsymbol{\nu}$ (circled) is the stationary point on the Lagrangian. The two dark lines indicate solutions to the partial derivatives $\nabla_{\boldsymbol{\nu}}L = \mathbf{0}$ (concave up) and $\nabla_{\mathbf{x}}L = \mathbf{0}$ (concave down). The quadratic in the $(\mathbf{x}, L)$ space is the projection of the line $\nabla_{\boldsymbol{\nu}}L = \mathbf{0}$ into $\mathbf{x}$ , which is the quadratic cost function $F(\mathbf{x})$ . . . . .	45
3.2	The ranges 0 to $\infty$ for covariance and information forms . . . . .	51
3.3	Schematic illustration of the augmented system form and the information form. . . . .	56
3.4	A set of equivalences between graph concepts and linear systems in estimation . . . . .	76
3.5	The augmented and information forms before observation-conditioning	77
3.6	Illustration of the innovation and residual terms . . . . .	81
3.7	Illustration of the <i>residual</i> approach for multiple observations . . . .	88
3.8	A multiple-residual case arising from a trajectory smoothing structure	89
3.9	Alternative system forms and solving approaches . . . . .	95

3.10	Number of nonzeros in the augmented form and the information form for various $N_{\text{state}}$ , in the case of a large observation degree & small state degree. . . . .	98
3.11	Number of nonzeros in the <b>L</b> factor for various ordering approaches, in the case of a large observation degree & small state degree. . . . .	99
3.12	A large observation degree, small state degree example, systems <b>A</b> and <b>Y</b> <sup>+</sup> . . . . .	100
3.13	A large observation degree, small state degree example showing the <b>L</b> for the alternative orderings. . . . .	101
3.14	Large state degree, small observation degree - L sparsity vs. $N_{\text{state}}$ . .	104
3.15	Large state degree, small observation degree - L sparsity (orderings) .	105
3.16	A large state degree, small observation degree example, systems <b>A</b> and <b>Y</b> <sup>+</sup> . . . . .	106
3.17	A large state degree, small observation degree example showing the <b>L</b> for the alternative orderings. . . . .	107
3.18	Structure of states and observations for a dynamic system example. .	109
3.19	Example factorisation ordering . . . . .	115
3.20	Typical fragments of the factorisation ordering generated by <code>colpermamd(A)</code> .116	
4.1	Graph representation of symmetric linear systems . . . . .	128
4.2	An example triangular square linear system <b>L</b> shown in both matrix and graph forms. . . . .	129
4.4	Matrix and graph equivalents for a dense vector . . . . .	134
4.3	Squareness and symmetry ambiguity of matrices resolved in the graph form . . . . .	135
4.5	Vector oriented and vertex oriented data storage schemes. . . . .	136
4.6	Matrix and graph equivalents for scalar matrix entries, for symmetric (undirected), unsymmetric (directed) and diagonal entries. . . . .	137
4.7	Sparse vector representations . . . . .	138
4.8	Matrix and graph equivalents for a block diagonal matrix . . . . .	140
4.9	The <b>edge</b> data structure . . . . .	148
4.10	The <b>loop</b> data structure. . . . .	148
4.11	The <b>graph</b> containment of the <b>vertex</b> objects. . . . .	157

4.12	The <b>graph</b> containment of the <b>edge</b> objects. . . . .	158
4.13	The <b>vertex</b> containment of the <b>edge</b> objects . . . . .	159
4.14	The CSC matrix format . . . . .	160
4.15	Insertion time versus the number of shifted entries for the CSC matrix format and the graph format . . . . .	162
4.16	Graph and Matrix representations of a linear chain for the traversal test.	164
4.17	Traversal time versus chain length . . . . .	166
5.1	Graph based LDL factorisation example . . . . .	184
5.2	scalar outer product, graph form . . . . .	193
5.3	Off-diagonal outer product, graph form . . . . .	197
5.4	In vs. out edges for triangular (acyclic) systems . . . . .	201
5.5	Acyclic graph root and leaf boundaries. . . . .	206

# List of Tables

3.2	Number of nonzeros in the augmented form and the information form, in the case of a large observation degree & small state degree. . . . .	98
3.3	Number of nonzeros in the $\mathbf{L}$ factor for various ordering approaches, in the case of a large observation degree & small state degree. . . . .	99
3.4	Number of nonzeros in the augmented form and the information form, in the case of a large state degree & small observation degree. . . . .	104
3.5	Number of nonzeros in the $\mathbf{L}$ factor for various ordering approaches, in the case of a large state degree & small observation degree. . . . .	105
3.6	Summary of dimensions of observations and their linked states . . . .	111
3.7	Summary of dimensions of states and their linked observations . . . .	111
3.8	Augmented system $\mathbf{L}$ factor sparsity for various ordering algorithms .	112
3.9	Sparsity of the un-factorised augmented and information form systems.	113
3.10	Triangular Factor Sparsity . . . . .	114

# List of Examples

3.1	The elimination of constraint-deviation bias using equality constraints	47
3.2	The case of zero eigenvalues due to over-defined constraints . . . . .	62
3.3	The regularisation of over-defined constraints . . . . .	64
3.4	Sparsity factorisation ordering in a large observation degree case . . .	97
3.5	Sparsity factorisation orderings in a large state degree case . . . . .	103
3.6	Sparsity factorisation in a localisation and mapping example . . . . .	109
3.7	Numerical stability of the augmented system versus information form near constraints . . . . .	119
3.8	Numerical stability of differing factorisation orderings near constraints	120
5.1	The requirement for 2 by 2 block factorisation steps. . . . .	179
5.2	$\mathbf{L}$ versus $\mathbf{L}^T$ , forward versus backward directions for directed-acyclic (triangular) solving . . . . .	199

# Nomenclature

## Typefaces

$\mathbf{A}, \mathbf{Y}$	Matrices
$\mathbf{x}, \mathbf{b}$	vectors
$x, v$	scalars
<code>add_edge</code>	Code and pseudo code
$\mathcal{A}, \mathcal{L}, \mathcal{D}$	graph edge-sets
$\mathcal{G}$	graphs

## Notation

$\mathbf{x}$	Any state
$\nu$	Any observation or constraint Lagrange multiplier
$\mathbf{h}(\mathbf{x})$	A (nonlinear) observation function of state $\mathbf{x}$
$\mathbf{z}$	The obtained observation value
$\mathbf{h}(\mathbf{x}) - \mathbf{z}$	The residual of the observation, evaluated at $\mathbf{x}$
$\Delta\mathbf{x}$	An increment or step to state $\mathbf{x}$
$\mathbf{H}$	The observation Jacobian (either linear or a particular linearisation of a nonlinear $\mathbf{h}$ )
$\mathbf{H}^T$	The $\mathbf{H}$ matrix transposed
$\mathbf{Y}^+$	The posterior value of $\mathbf{Y}$
$E[\mathbf{v}]$	Expectation of $\mathbf{v}$
$a \rightarrow b$	Replacement of $a$ into $b$
$\mathbf{x}_e$	Solution (final estimate) of $\mathbf{x}$
$\hat{\mathbf{x}}$	Mean of a prior estimate of $\mathbf{x}$
(obs,states)	Concatenation of the sequence of observations followed by states



**Abbreviations**

SLAM	Simultaneous Localisation and Mapping
DoF	Degrees of Freedom
nnz	Number of nonzeros
CSC	Compressed-Sparse-Column (sparse matrix format)
CSR	Compressed-Sparse-Row (sparse matrix format)
MAP	Maximum-a-posteriori (estimate)
PDF	Probability density function

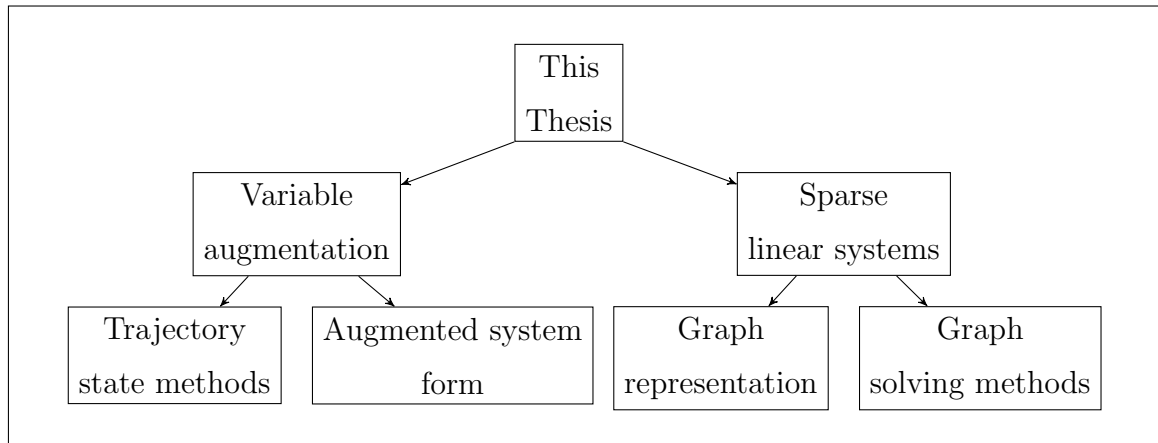
# Chapter 1

## Introduction

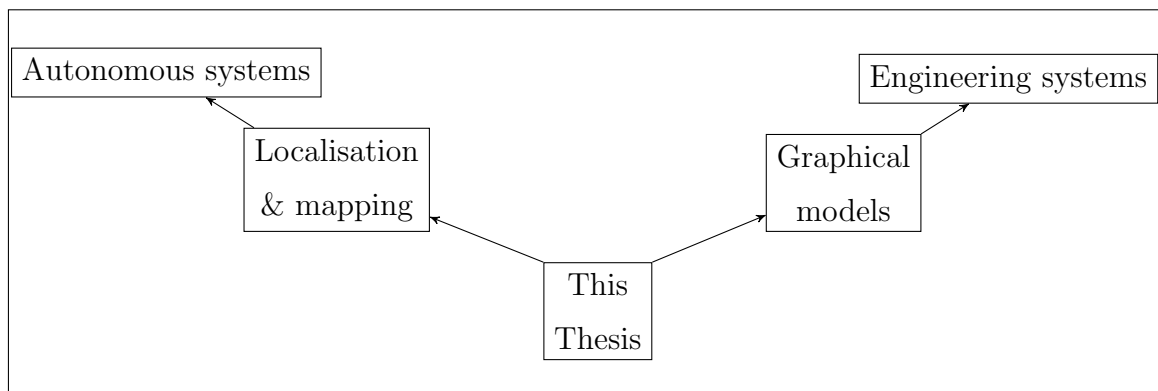
This thesis contributes innovative mathematical structures and approaches for the formulation and solution of estimation problems. These consist of: the augmented system form; the graph representation of estimation problems and linear systems; and the graph embedded solving of linear systems.

The approaches presented in this thesis have improved capabilities and flexibility over their conventional alternatives. The approaches are more general but mathematically equivalent alternatives to existing methods. By offering novel and general alternatives to fundamental underlying tools, this thesis contributes towards the bottom-up improvement of estimation solving methods. In particular, this thesis contributes detailed linear and nonlinear systems structures and associated data structures. These are used to represent observations, states and the links between them. This thesis also contributes novel approaches to a solving algorithm which operates within those structures. The topics contributed by this thesis operate in a complementary manner with each other, while still being separately and individually beneficial.

This thesis is aimed at improvements in nonlinear, high dimensional estimation problems in localisation and mapping. Further potential applications exist in a wider variety of estimation problems and related high-dimensional solving or optimisation problems.



**Figure 1.1:** Thesis outline (subsets of this thesis). This thesis includes topics relating to the augmentation of additional variables, including trajectory states and observations, sparse linear systems including a novel graph representation and associated solving algorithms.



**Figure 1.2:** Thesis outlook (supersets of this thesis). Applications and extensions to this thesis lie in localisation & mapping and graphical models, for example. These in turn relate to autonomous systems and engineering systems more generally.

## 1.1 Thesis Contributions

The principal contributions of this thesis are as follows:

### 1. Augmented Methods in Estimation

This thesis proposes and develops an estimation approach consisting of the augmentation of observations and constraints, in addition to the states. This augmented system method is based upon the trajectory state augmentation approach, since the benefits of retaining the *observations* rely on retaining their related *states*.

The augmentation of observations makes the augmented form more general than the information form, since it describes the dual system of both the states and the observations & constraints. The observations and constraints are augmented as dual variables, rather than marginalised into the states.

This thesis proposes that the augmented system form is a more general starting point for estimation algorithms than the information form. The conventional approach of solving the information form can equivalently be recovered by eliminating the *observations* first. In addition, a wider range of elimination orders can be obtained by eliminating observations and states in a mixed order, including the simultaneous elimination of pairs of variables in the observations and/or states. This flexible elimination is essential in the presence of constraints, and improves the numerical conditioning in cases of *near constraint* tight observations. This flexible elimination is also beneficial for sparsity and numerical reasons, depending on specific numerical and graph structure properties of the states and observations.

The augmented form is a mathematical description of the estimation problem showing explicitly and separately the states and observations together with a cross-coupling interaction. The augmented system form describes the full structure of the sparse estimation problem as a mathematically solvable system. The augmentation of observations exposes their Jacobians directly, allowing simple linearisation changes, whereas in the information form the Jacobians

are mixed in together and expressed on the states. This formulation approach brings insights into data fusion and estimation systems by explicitly showing the interaction of observations and states via Lagrange multipliers.

## 2. **A novel graph-theoretic structure for sparse estimation problems**

This thesis contributes a new structure for representing estimation problems. This structure is a graph based structure, focusing on the representation of objects and the links between them, rather than using conventional vector and matrix semantics. The structure represents estimation problem states, nonlinear observation terms and their linearisation but also focuses on linear systems generally. This structure offers improved capabilities and efficiency of storage, access and online modification.

This thesis contributes a novel approach for the mapping of vectors and matrices into graph vertices and edges as an explicit structure for runtime operations. In particular, this thesis contributes a graph structure distinguishing loops, symmetric and directed edges, and containing multiple *edge sets* which are all motivated from the requirements for representing linear systems.

The structure introduced in this thesis departs from conventional vector and matrix semantics. This thesis proposes a new interface based on object access rather than integer indexing. This subtle change actually has a significant effect on the arrangement of algorithms.

This thesis contributes a practical implementation of the graph based structure for linear systems. This thesis compares the graph structure implementation against a conventional sparse matrix format for insertion and traversal operations, showing significant benefits to performance.

## 3. **Estimation direct solving algorithm in the graph structure**

This thesis proposes a novel graph based implementation of the LDL direct factorisation and solving algorithm. This algorithm exploits the graph embedded representation of the linear systems to allow greater flexibility and capabilities in the factorisation, particularly regarding the factorisation ordering. The new

graph data structure opens up the development of a new variety of estimation theoretic and linear algebraic tools which may be useful in future for faster solving and online modification algorithms.

The relationships between these contributions are as follows:

### **Augmented system form & Graph structure**

The augmented system form provides a mathematical system representing both observations and states, while the graph structure provides a fundamental tool for representing sparse relationships between variables generally. The graph structure helps by efficiently representing the observations and states, and the sparse observation Jacobians which link them. The graph structure complements the benefits of the augmented system form by allowing fast augmentation linking and access operations and a decoupling of algorithmic orderings with storage orderings.

### **Augmented system form & Graph solving**

While the augmented system form provides the mathematical system, the graph solving algorithm describes how to solve it. In particular, the graph solving algorithm supports the solution of indefinite linear systems, which occur as a result of using the augmented system form. The graph solving algorithm complements the augmented system form by allowing flexible factorisation orderings.

### **Graph structure & Graph solving**

The graph structure provides the representation of the problem and the tools for manipulation, while the graph solving algorithm utilises the graph structure tools as the manipulations in the solving algorithm. The graph solving algorithm exploits the benefits of the graph structure in terms of easy and fast insertions and adjacency accesses. The graph solving algorithm operates with the novel facilities of the graph structure approach, especially in terms of indexing and access properties.

## 1.2 Motivation for Approaches

This thesis presents an interlinked set of approaches which derive from an investigation into alternative structures for the formulation and solving methods in estimation.

The methods were motivated by the trajectory state or delayed state paradigm for estimation [20, 24, 26, 46, 70, 71]. It was desirable to be able to rapidly insert states and observations into the representation and perform online modification solving. Considerations for the online modification were motivated by iterative methods. The concept of this was to traverse through the structure of the observations and states, starting from the point of insertion of new observations, where the residuals and solution were disturbed, and terminating several steps later, where the solution and residuals would be less affected.

This motivated the idea to use a graph representation to manage the structure of the observations and their connections to the states of the estimation problem. This included the idea that the observation Jacobian matrix would be easily stored on the graph edges between the observations and the states. Indeed, a key insight was that any sparse linear system could be encoded onto a graph between variables of the system. An essential aspect was that the graph structure would be a core part of the implementation, rather than a matrix based implementation, as well as being an important theoretical tool.

The graph representation had mathematical appeal over the alternative sparse matrix representation. The graph representation offered explicit encoding of the sparsity structure and fast access to adjacent variables. The graph representation would operate without a row or column orientation preference, giving it a symmetry advantage over a matrix representation.

The graph representation presented considerable conceptual hurdles in relation to how it would relate mathematically to the estimation problem and how it would relate to a direct solving process. The representation initially considered was a bipartite directed representation consisting of edges pointing from states to observations, and with distinct treatments for observations and constraints. However, there were some unresolved

questions in this representation. The primary focus of the bipartite graph was on the representation of  $\mathbf{H}$ , the Jacobian of the observations with respect to the states.  $\mathbf{H}$  is fundamentally rectangular and directed. It was then unclear how to represent  $\mathbf{Y}$ , the prior information matrix, which is fundamentally square and symmetric. Also, in the bipartite directed representation it was not clear what mathematical system or solving process corresponded to the joint system of observations and states.

Iterative methods were considered for the solution process. In particular, the conjugate-gradient method for the normal equations (CGNR) [61] was considered. The CGNR performs matrix-vector multiplication in the normal equations  $((\mathbf{H}^T \mathbf{H}) \mathbf{x})$  in the form  $\mathbf{H}^T(\mathbf{H}\mathbf{x})$ . The directed bipartite edges were well suited to these forward and transposed matrix-vector multiplications required for the iterative method.

However, this approach is based on the information form (normal equations) rather than solving a joint system in the observations and constraints.

It was also important to consider direct solving methods, due to the need to be able to perform marginalisation and obtain individual or small joint covariances, for data association purposes, and to relate the developed methods back to existing estimation methods. It was not clear how the bipartite graph representation approach would support the factorisations required for direct solving. One consideration was to introduce new pseudo-observation vertices corresponding to each row in the factorisation. The QR factorisation of the observation Jacobian was considered, based on [10] and [20] however, there were difficulties adapting this to constrained systems.

The augmented system form, from both the least squares [10] and equality constraint literature[12], provided a mathematical model for an estimation problem formulation consisting of explicit separation of the observations and states. This justified and motivated alterations to the graph embedded representation. Instead of the bipartite graph linking states to observations or constraints, the augmented system form described a symmetric graph. The augmented system form described how to uniformly represent each of the terms  $\mathbf{H}$  and  $\mathbf{Y}$ , as undirected edges. ( $\mathbf{H}$  is represented by *undirected* edges because it is an off-diagonal between the states and observations in the augmented system, which is *symmetric*). The augmented form also allowed



for generalised observations & constraints with a single, unified treatment via the observation/constraint uncertainty covariance,  $\mathbf{R}$ . The augmented system also emerged during this thesis as a fundamental underlying mathematical system for the estimation problem.

The factorisation process was then able to be clearly understood as the LDL factorisation of the augmented system, which is a single symmetric system. The graph concept was extended in order to be able to simultaneously represent both the undirected, symmetric systems of the estimation problem (augmented form) and the unsymmetric, triangular, directed-acyclic systems of the linear system factorisation. This required extensions to the graph concept, guided by the mathematics of the linear systems forms which were required.

Finally, an initial set of direct solving algorithms based entirely in the new graph representation were developed. These adapt existing algorithms into the new representation to explore the consequences of the new representation and the augmented system form. The direct solving algorithms presented are the beginning of new alternative methods which exploit the graph representation.

### 1.3 Motivating Problem

This thesis was motivated by the problem of online, joint estimation of external feature mapping and vehicle localisation. This mapping and localisation problem was considered in the context of multiple unmanned aerial vehicles operating jointly on the mapping and localisation task. The sensors driving the estimation task were considered to be primarily vision and inertial sensing, which further motivated the inclusion of various sensor calibration and bias parameters.

This application problem is subject to various fundamental challenges:

- The system models, such as the observation and prediction models, are inherently nonlinear. Nonlinearity impairs the ability of the system to predict the variation

of the models at a distant point in state space, given analysed information from a present point in state space. This reduces the ability of the estimator to accurately determine the effects of adjustments to the state estimates. This in turn means that adjustments must be made more carefully, checking validity and re-computing linearisation values.

- The observation and prediction models in this application context are typically *partial rank* models. A partial rank observation model refers to an observation which provides fewer observation outputs than the number of input states. Consequently, the observation *cannot be inverted* to obtain estimates of the states. For example, the vision sensing provides a projective, bearing-only observation of only two dimensions, despite the state consisting of many degrees of freedom in the observer, camera and feature.

This thesis addresses partial rank models by relying exclusively on the one direction in which models *can always* be used: the computation of projected observations from states (rather than the inversion of observations back into state estimates). Furthermore, sufficiently general linear algebra operations or precautions are required when handling general rank models, since various terms may appear invertible.

- The states used in the application consist of mixed static and dynamic states. The vehicle position, velocity and attitude states are highly dynamic, whereas the feature states are modelled as stationary, static states. (Sensor calibration and bias parameters might be modelled either way depending on application domain choices). Furthermore, the static and dynamic states are coupled, since the mapping task requires the vehicle to observe each of the map features.

From an estimation point of view, the mixture of static and dynamic states leads to a problem structure consisting of various *chain-like* and *looped* structures. Furthermore, this structure varies continuously in an unpredictable manner. The chain-like aspect derives from discretisation of continuous dynamics, and this evolves with a steady chain-like structure. Looped structures form when earlier

features are re-observed at a later time.

These high dimensional, sparse interlinking structures motivated the developments in this thesis.

- The estimation problem derives from online systems which continually input sensor and prediction observations. Therefore, the estimation problem is changing and growing continually. The problem of *online estimation* motivated the developments in this thesis.

## 1.4 Thesis Structure

Chapter 2 gives an overview of the estimation process, indicating where the contributions of this thesis fit in.

Chapter 3 describes the augmented system method. The concept and theory of the approach is described and its relation to existing methods in estimation is shown. The augmented form is a key tool for solving equality constrained problems. The augmented form is an important generalisation of the information form which offers benefits in the full representation of the estimation problem including the observations and states. The augmented system form can be factorised using a mixed ordering in the observation and state variables, allowing improved sparsity and numerical stability properties.

Chapter 4 proposes a novel graph embedded representation of the estimation problem and associated sparse linear systems. The theory and implementation of the graph representation are described. Numerical evaluations compare the proposed graph representation of linear systems against the conventional compressed-sparse-column (CSC) matrix representation. Compared to the matrix representation, the graph representation allows constant time insertion and removal of edges or vertices, allows fast & constant time access to adjacent variables, and decouples the factorisation ordering from the underlying storage. The proposed graph representation also incorporates

novel graph representation elements motivated by the need to represent sparse linear systems.

Chapter 5 presents a graph based direct solving algorithm for the solution of estimation problems. This algorithm exploits the graph embedded representation of the linear systems. The fast insertion and adjacency capabilities of the graph embedded representation alter the complexity of sparse direct solving algorithms. This allows the solving algorithm to use a more flexible factorisation ordering approach, determined mid-factorisation.

Chapter 6 concludes the thesis and outlines areas for future research.

# Chapter 2

## Estimation In Localisation and Mapping

This chapter gives a broad overview of the overall estimation processes to indicate where the contributions of this thesis reside within the context of the estimation process.

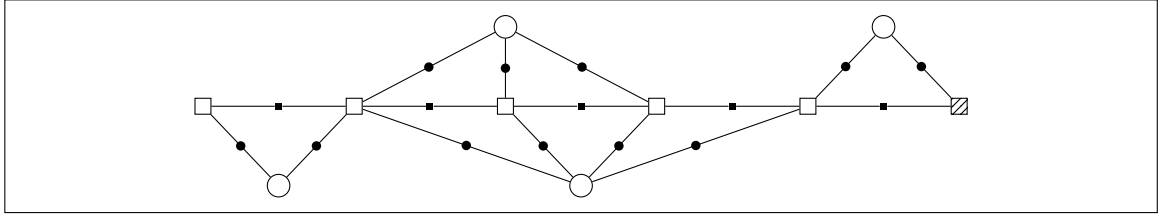
The methods of this thesis are broadly based on prior approaches in probabilistic estimation [8, 50], estimation in simultaneous localisation and mapping [20, 70, 71], bundle adjustment [72], numerical optimisation [12, 54], numerical linear algebra [10, 37] and graphical models [40, 41, 44, 56]

### 2.1 Localisation and Mapping Literature

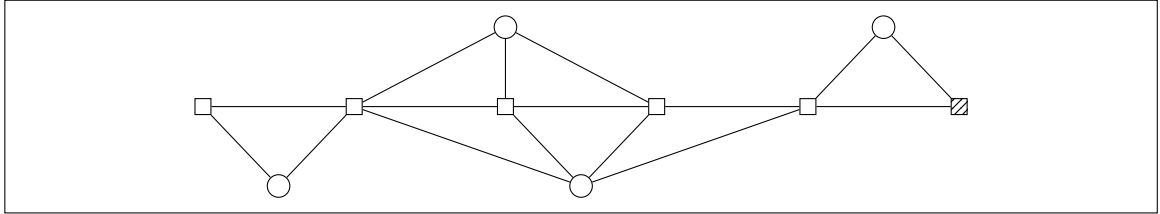
There are several primary approaches in the literature on localisation and mapping which will be considered in this thesis. These are illustrated in figures 2.1 and 2.2. Further surveys of the literature in SLAM are given in [7, 23, 70].

- Augmented system form. (Proposed in this thesis). The estimated variables are the vehicle pose trajectory states, static feature map states and the observation Lagrange multipliers.

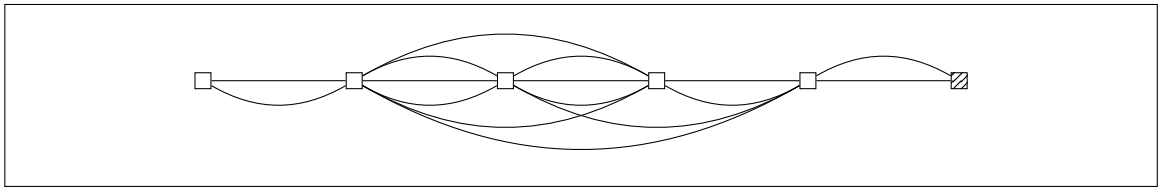
- Pose trajectory and map form (smoothing and mapping or SAM). The estimated variables are the vehicle pose trajectory states and the static feature map states.
- Pose-only trajectory form (viewpoint based SLAM). The estimated variables are the vehicle pose trajectory. Note that the terminology “trajectory state methods” used in this thesis covers both the trajectory-only estimation (viewpoint based SLAM) and the trajectory plus map estimation approaches (smoothing and mapping).
- Filtering form. The estimated variables are the present vehicle pose state and the static feature map states.



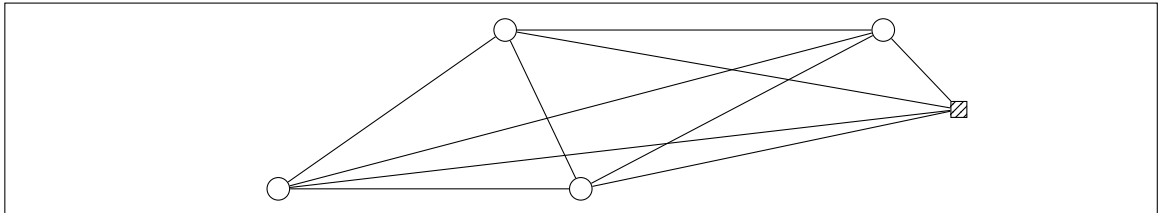
(a) The augmented observations, trajectory and map form of this thesis. The estimated variables are: vehicle pose trajectory states, the static feature map states, plus the observations.



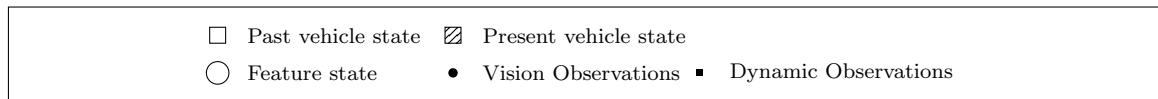
(b) The SLAM trajectory and map form (smoothing and mapping or SAM). The estimated variables are: vehicle pose trajectory states and the static feature map states. This is obtained from 2.1a by eliminating the observation variables.



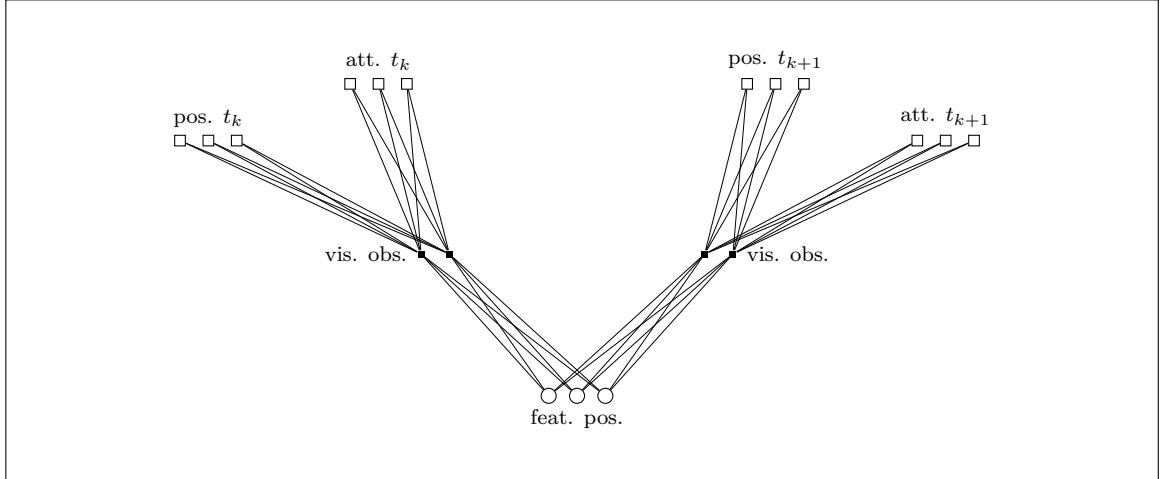
(c) The SLAM pose trajectory only form (viewpoint based SLAM). The estimated variables are: vehicle pose trajectory states. This is obtained from 2.1b by eliminating the feature states.



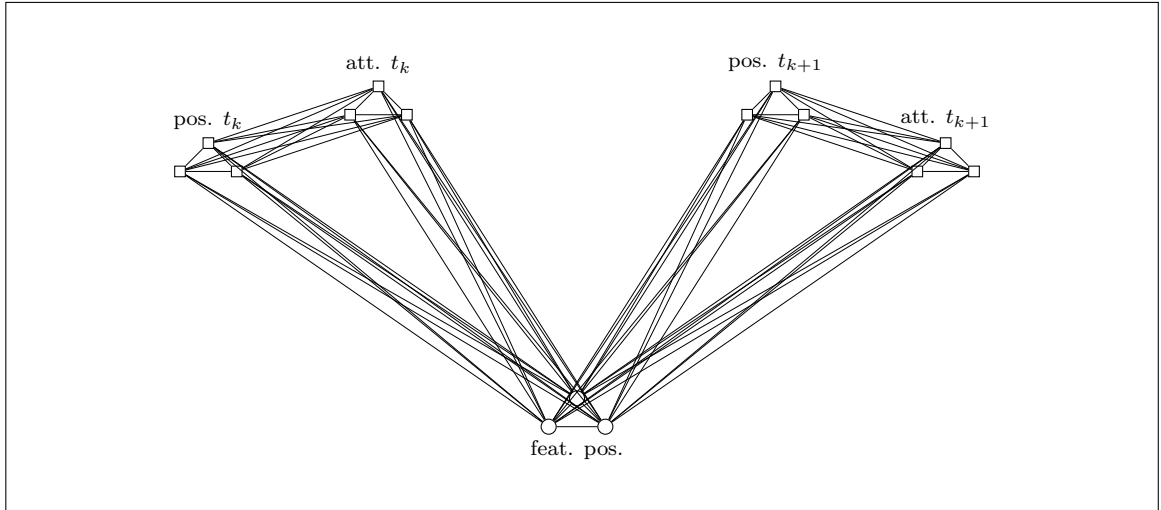
(d) The SLAM filtering form. The estimated variables are: the present vehicle pose state and the static feature map states. This is obtained from 2.1b by eliminating the vehicle trajectory states.



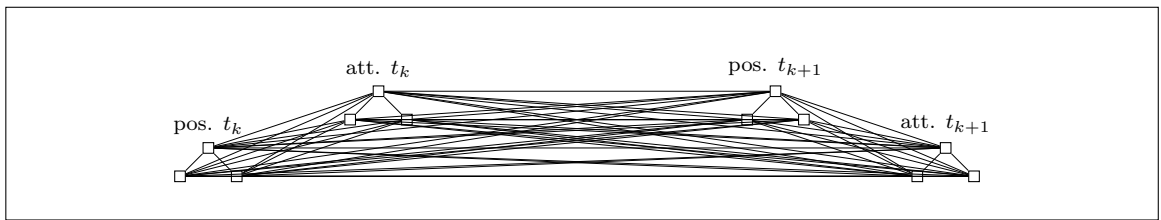
**Figure 2.1:** SLAM frameworks. Each framework considers certain sets of variables jointly. The methods all relate to marginalised subsets of the augmented system form. In marginalising these links, it is important to consider the internal dimensions hidden in each vertex, shown expanded in figure 2.2



(a) Augmented System Form (vehicle states, features and observations).  $N_{\text{links}} = 36$



(b) Smoothing and Mapping form (vehicle states and features).  $N_{\text{links}} = 69$



(c) Viewpoint form (vehicle states only).  $N_{\text{links}} = 66$

**Figure 2.2:** SLAM frameworks (detail). In this case, showing the internal dimensions of the vehicle position(3) and attitude(3), the feature position(3) and the vision observations(2) shows the significant fill-in when marginalising. The augmented system form contains the most variables (19) but the least number of links (36).



### 2.1.1 Smoothing and Mapping (SAM)

The smoothing and mapping (SAM) approach is illustrated in figure 2.1b. The estimated variables are the vehicle pose trajectory states and the static feature map states.

The SAM approach forms large scale sparse networks consisting of the vehicle trajectory and feature map states, followed by operation of sparse system solvers on that network [3, 20, 42].

The SAM approach is an important prior method to the approach proposed in this thesis. In this thesis, the proposed augmented system form augments the observation Lagrange multiplier variables onto the smoothing and mapping state variables. Consequently, from the point of view of this thesis, the smoothing and mapping approach is derived from the augmented system form by eliminating the observation Lagrange multiplier variables. Compared to the augmented system form approach proposed in this thesis, the SAM approach is an example of a *fixed elimination policy* approach; The observation variables are eliminated unconditionally.

The smoothing and mapping approach proposes the widest range of state variables among the methods reviewed in this section (consisting of the vehicle trajectory and the map states). Descriptions of the smoothing and mapping approach refer to this as the “full SLAM problem” [20, 69].

This thesis also recommends adopting such a fully representative formulation of the problem.<sup>1</sup>

The method of “bundle adjustment” from the field of photogrammetry [72] is closely related to the smoothing and mapping method; The estimated variables are the sequence of vehicle (or camera) positions together with the set of feature positions. However, in bundle adjustment there is no dynamic model linking the successive camera positions. The camera positions are only linked through the observations to

---

<sup>1</sup>Another class of significant state variables are bias variables associated with the observations, for example, calibration and alignment states or measurement bias states. These should also be augmented into the system where applicable.

the features. Triggs [72] provides an extensive summary and survey article on the methods of bundle adjustment. Methods in bundle adjustment frequently apply a fixed marginalisation or factorisation strategy of eliminating either the map points or camera points first [13, 47, 72]. This is particularly applicable in bundle adjustment since, in that context, the feature-feature and camera-camera blocks are sparse and block-diagonal.

Recent work in localisation and mapping estimation literature has a focus on high fidelity formulation frameworks coupled to efficient solution methods. For example, [3] forms a large scale graph network of vehicle states and features, linked by vision frames. Such a network represents a general smoothing-and-mapping formulation of the problem. This network is then selectively *marginalised* down to a tractable size for realtime operation. The method presented in [3] performs significant reductions in the network size through marginalisation aided by nonlinear parameterisation methods. The choice of parameterisation is important as it affects the accuracy of approximations introduced by linearisation and made permanent in the posterior after marginalisation [72]. The choice of parameterisation is an important topic but is orthogonal to the methods proposed in this thesis.

Dellaert [20] discusses the importance of the size of the representation of the problem formulation. The alternatives considered in [20] are the posterior information form ( $\mathbf{Y}^+$ ), the system measurement Jacobian matrix ( $\mathbf{H}$ ) and the system posterior covariance matrix ( $\mathbf{P}^+$ ). The information matrix and measurement Jacobian are shown to be naturally sparse in the SAM formulation, whereas the covariance matrix is naturally dense.

The sparsity of the SAM form arises due to the limited number of variables which are involved in any one *factor*. The properties of the possible types of factors are known when the system is designed. In particular the factors arising in mapping and localisation have guaranteed maximum degree. The SAM information and Jacobian systems are then sparse for nontrivial trajectories and maps. The sparsity of the SAM form is closely related to the discussion in section 3.4.1 regarding the augmented system form and factor graphs in smoothing and mapping.

The improved sparsity that arises from maintaining the additional vehicle trajectory states is discussed in [20]. By maintaining both the vehicle trajectory states and the feature map states, the *elimination or factorisation order* of the variables can mix between the trajectory and map states. Algorithms such as [18] which determine the factorisation ordering can operate on the feature and trajectory states jointly. The resulting orderings are, in general, better than choosing either the features first or the trajectory states first [20].

In summary, instead of using a fixed policy for the factorisation ordering or solving approaches, this thesis recommends building the formulation of the system followed by analysis of the factorisation ordering and operation of the solver on that formulation.

This motivated the development in this thesis of the augmented system form, in which the factorisation ordering can mix between the *observation and state* variables.

### 2.1.2 Viewpoint based SLAM

In viewpoint based SLAM, the estimation variables are the vehicle pose trajectory states only.

Descriptions of this approach in the literature [24, 25, 49] operate in contexts where the sensor data (images or scans) are able to be processed into relative pose relationships between pairs of vehicle poses. These pose displacements then chain and loop together to form the overall connected structure which defines the estimation problem. The viewpoint based approach does not directly estimate feature states. However, “features” are less well defined because some view based implementations process the sensor data into pose relationships without features (for example, scan or image matching). This means that viewpoint based approaches are effectively equivalent to the *elimination* of the features [24]. Thus the viewpoint based approach to SLAM takes the fixed elimination policy of eliminating the features. By contrast, the SAM and augmented system form approaches formulate the system with the features included.

The viewpoint based SLAM approach is claimed to have a naturally sparse information

matrix representation [24]. This is due to the pattern of using sensor images to generate vehicle pose relations.

The elimination of features onto the vehicle trajectory will cause fill-in among the remaining vehicle trajectory states. The systems in [24, 49] operate close to the sea floor. Features are seen at relatively close range for relatively short durations during traversal (and seen again occasionally in loop closure). This short duration gives the system structure a small feature degree such that it is beneficial to marginalise out the features and adopt a trajectory or pose oriented framework. The extent of fill-in is kept small because of the limited range of view to the features.

Repeatedly eliminating features will become inconsistent if the same feature or feature pair is used. However, the description in [24] states that it is able to avoid re-use of the data to overcome this. This is helped by the environment which has a short duration of visibility of features, and the specific features used for matching in images changes frequently.

However, the application considered for this thesis is in airborne mapping and localisation. In the airborne context it is possible and desirable to observe and track particular features for extended durations. The view based approach to SLAM makes long observations of a feature difficult because marginalising such a feature would cause fill-in among many vehicle states. If the single feature is repeatedly added and marginalised, the system will become inconsistent in a manner which will accumulate during the long observation of the feature.

The sparsity of the viewpoint based SLAM approach is not fundamentally guaranteed but is a consequence of the *typical* scenarios encountered. (For an *atypical* example, a single feature seen for all time would cause dense fill-in over all the vehicle trajectory states).

In these cases involving repeated or long duration use of a single feature, or in the general case (where such properties may not be known in advance or may vary from feature to feature) this thesis recommends the use of the smoothing and mapping (SAM) or augmented system form in order to deal with the sparsity and consistency

issues.

The graphSLAM system described in [69, 71] also has aspects in common with the view based approach. In particular it focuses on elimination of the map features first as a fixed factorisation policy. This can result in significant fill-in onto the vehicle trajectory states in the case of extended observation of a feature. However, the system described in [71] has much in common with the SAM approach of [20] and could feasibly eliminate the variables in any order.

The viewpoint based approach is therefore seen as a subclass of the smoothing and mapping approach which is appropriate in cases where the features are defined implicitly in sensor matching and/or the features have a short duration of visibility such that their structural pattern encourages their early elimination.

### 2.1.3 SLAM Filtering

The final primary approach considered in this section is the *filtering* approach to SLAM. In the filtering approaches to SLAM, the estimated variables are typically the single present vehicle pose state and the collection of static feature states (the map). This form is also known as “feature based SLAM”.

The primary difficulties with filtering based approaches to SLAM are linearisation errors and system sparsity problems. A filtering approach fundamentally aims to represent the entire problem history into the latest posterior probabilistic representation. The difficulty lies in the inability of reasonable functional forms (especially the Gaussian distribution) to properly represent the nonlinear distributions over the variables in the final posterior. Linearisation choices adopted earlier in the filtering cycle cannot be adjusted at later stages.

Even in a completely linear scenario, a second difficulty is that in both the covariance and the information form, the posterior Gaussian distribution becomes fully dense. This arises due to the elimination of the past vehicle states from the estimation variables. All seen features therefore become fully linked and correlated. This causes

infeasible scalability as the number of map features grows. This is discussed further in [24]. This filtering approach is the earliest method and a wide variety of derived methods exist [7, 23, 70].

## 2.2 Graphical Models Literature

A graphical model is a representation of a joint, high dimensional probabilistic model. For a high dimensional set of variables  $\mathbf{x}$ , a graphical model encodes the joint probability (or probability density)  $P(\mathbf{x})$ . In graphical models, *vertices* represent variables and *edges* represent conditional dependencies between variables. Graphical models have been referred to as “a family of techniques which exploit a duality between graph structures and probability models.” [66]. Broader introductions to graphical models are given in [40, 41, 44, 51, 56, 66].

There are three main types of graphical model which will be of interest to the methods developed in this thesis:

### Factor graphs

Factor graphs [20, 44] are *bipartite* graphs consisting of two types of vertex: state variables and observation variables. In this thesis, the augmented system form developed in chapter 3 is closely related to the factor graph model.

### Markov Random Fields

Markov Random Fields (MRFs) or Markov networks are *undirected* graphical models with vertices consisting of state variables [41]. Gaussian MRFs are equivalent to the *information form* (sparse, symmetric linear system). MRFs are related to factor graphs; Factor graphs are reduced into Markov Random Fields via marginalisation. This is developed further in chapter 3.

### Bayes Nets

Bayes nets are *acyclic directed* graphical models [51]. Bayes nets are equivalent to sparse *triangular* linear systems. Such systems are important in this thesis in the context of direct *factorisation* methods for the solution process. Both factor

graphs and MRFs can be factorised into acyclic directed graphical models for solving. This is developed further in chapter 5.

This thesis contributes to methods for estimation in localisation and mapping by applying techniques from the more general field of graphical models.

The general factor graph model and formulation approach [44] is applied back into Gaussian models in estimation and used to derive a form which has been missing from the estimation literature: The augmented system form (chapter 3). The augmented system form developed in chapter 3 adds a key ingredient from (factor graph) graphical models into the models used in estimation: a full formulation describing the existence and links between both observations and states. Further relationships between graphical models and the augmented system form are developed in chapter 3.

This thesis also derives from approaches used in graphical models, developing software for a fundamentally graphical representation for sparse linear systems (chapter 4) and associated graphical solving algorithm (chapter 5).

## 2.3 Assumptions and Context

Section 2.1 discussed the choice of variables which represent the problem. The estimation methods described in this thesis are founded upon a series of fundamental assumptions. This section notes a hierarchy of these assumptions in order to give context to the methods chosen for this thesis.

### Nonlinear Bayesian estimation problem

The observations and states of the estimation problem formulation describe a nonlinear, Bayesian, joint high-dimensional probability model. This thesis adopts the Bayesian probabilistic methods for describing and manipulating uncertainties. This assumption implies the existence of state variables, which define the problem output, and observations in the form of probabilistic models, which define the input to the estimation problem. This thesis assumes a discrete (or finite) set of

continuous-valued variables, thus excluding the treatment of smooth continuum functions. This Bayesian approach is a widely used fundamental approach to estimation problems [8, 50].

Note that at this point the problem is formulated as a joint and high-dimensional probability model. No assumption is made *here* regarding the ability to apply recursive Bayesian methods or assume Markovian dynamics of the variables and models. These assumptions are deferred until well into the solving techniques and instead, the full joint model is formulated.

### Continuous & Differentiable

The probability model is assumed to exist as a continuous & differentiable function over continuous valued states, thus taking the form of a probability density function (PDF). This assumption also applies to the observation model functions. The observation models are also assumed to exist as continuous & differentiable functions over continuous valued observation variables. This assumption excludes the treatment of sampled PDF approaches or discrete state or observation variables.

### Approximately Convex

This thesis assumes a convex log-likelihood model for the observation models. In the observation space, the log-likelihood model for that observation is assumed to be a convex function. A convex function has no local extrema, only global extrema. This implies using a reasonable observation residual,  $h(\mathbf{x}) - \mathbf{z}$  and obtaining the property that when the observation residual is zero ( $h(\mathbf{x}) - \mathbf{z} = \mathbf{0}$ ) the optimum extrema of the convex log-likelihood is obtained. The log-PDF model for the observations is important since the summation of these, projected as likelihoods into the state space, becomes the log-PDF model for the solution. Convexity and the representation of the observations in residual form are complementary. Therefore, this thesis represents observations in a residual form, whereby the observation function returns a vector-valued *residual*. This corresponds to the intuitive idea of giving preference to observation projections with smaller residual.

In addition to the assumed convex log-likelihood model for the observations (in



the observation space), the overall convexity assumption implies and requires an assumption that the transformations between the states and observations approximately preserve convexity.

In accordance with this assumption on the observation models, the PDF for the states is approximated as a single modal and approximately log-convex function. This thesis therefore does not consider multi-modal Bayesian functional representations. This assumption of approximate convexity applies only to the method for the selection of the estimate. It does not assume that the PDF can be *permanently replaced* by a convex model, only that a convex model is a reasonable *local* assumption for the operation of the estimation algorithm. Convex methods are described further in [12].

### MAP Estimate & Optimisation Methods

For the “solution” state estimate for the PDF, the maximum-a-posteriori  $\mathbf{x}$  estimate is used. Seeking the MAP estimate is consistent with the assumption of log-convexity of the PDF, since a convex function will have either a single optimum value or a convex region of equally optimum values. Seeking the MAP estimate allows us to characterise the posterior solution as a single state estimate,  $\hat{\mathbf{x}}$ , a point in  $\mathbb{R}^n$ , together with a region of uncertainty, rather than outputting a full representation of the PDF.

An alternative to the MAP estimate is the *mean estimate* on the PDF. The use of the MAP estimate is motivated by computational considerations; The MAP estimate can be obtained, adjusted and verified by (repeated) examination of an infinitesimal region of the PDF, whereas the mean estimate is a quantity obtained by integration over the infinite PDF. The nonlinear form of the PDF representation is easily suitable for evaluation and differentiation. The optimisation techniques used in this thesis rely on utilising the point-evaluated value, gradient and curvature of the function. For example, requiring that the gradients balance out to zero at the solution. By contrast, the mean requires a balance in the integrated “probability mass” all around the solution. But the nonlinear models are not, in general, suitable for analytical integration necessary to find the mean. Requiring a mean estimate therefore usually requires utilising

a global functional or sampled PDF approximate representation.

Utilising the MAP estimate does not entail discarding the underlying Bayesian probabilistic approach to the estimation problem. The system maintains the estimation problem formulation as the original representation of the Bayesian PDF. The solution state estimate is not intended to replace the problem formulation and PDF. Instead the solution state estimate is simply an output interface for one particular representative point in state space.

In previous conventional approaches to estimation, the full PDF is required in order to re-compute the estimate when further factors are fused in. However, the system in this thesis can update the formulation representation to reflect the fusion of additional factors and then proceed to find an updated estimate solution. In other words, the output of the posterior solution is different from the underlying representation of the PDF necessary to *compute* the posterior solution.

The MAP estimate is used as the basis for the iterated-extended Kalman filter (IEKF) [8]. The IEKF provides an intermediate example between Gaussian filtering based estimation and nonlinear optimisation based estimation.

## 2.4 Solving Overview

The above assumptions about the nature of the estimation problem break the estimation problem down to one of convex optimisation. The approach to solving this convex optimisation problem is broken down further into a series of subproblems as follows. The essence of these methods is summarised in figure 2.3.

### Convex Optimisation

Under the conditions of the approximately convex log-PDF assumption and the requirement for the MAP estimate, methods of convex optimisation are applied. For further background on convex optimisation, refer to [12].

### Newton's method

Given the above assumptions, it is appropriate to adopt Newton's method. The

essence of the method is that the log-PDF is approximated locally by a quadratic model. The quadratic model is then solved for the next solution point, where the gradient is projected to be zero. The process is then iterated to a solution within acceptable tolerance.

The essential form of Newton's method is:

$$\nabla^2 f(\mathbf{x}_0) \Delta \mathbf{x} = -\nabla f(\mathbf{x}_0) \quad (2.1)$$

$$\mathbf{A} \Delta \mathbf{x} = \mathbf{b} \quad (2.2)$$

- Where  $f(\mathbf{x})$  is the local quadratic approximation function.

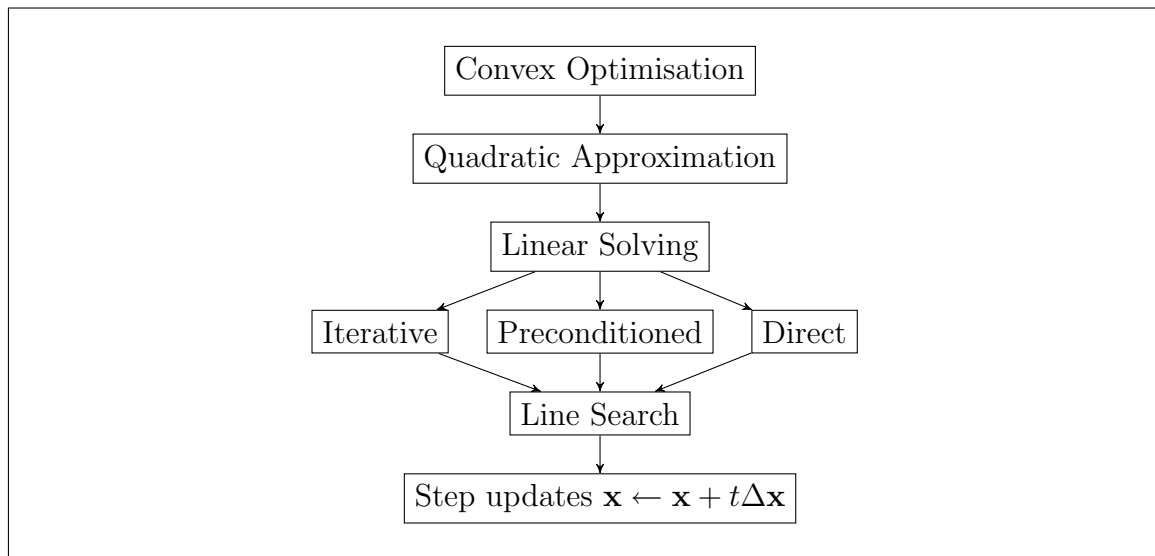
A variety of quadratic models are available. The second order Taylor expansion of the function about the current estimate requires evaluation of the Hessian of the objective function, which in turn requires evaluation of the Hessian of each scalar observation. By contrast, the Gauss-Newton approximation provides an alternative quadratic model involving only the Jacobian (first order) derivatives of the observations. This amounts to forming a linear approximation of the observations (which is then squared into a quadratic model), rather than forming a best-fit quadratic approximation to the nonlinear log-PDF function. For further details see [48].

### **Solution to sparse linear systems**

The solution of the quadratic model for Newton's method above requires the solution of linear equations. These are large, sparse linear systems. This thesis discusses the methods for solving these linear equations.

### **Line Search**

The optimisation of the nonlinear problem, following selection of a search direction from Newton's method, requires evaluation of the nonlinear problem along a one dimensional line. A step is finally chosen, updating the estimation variables via  $\mathbf{x} \leftarrow \mathbf{x} + t \Delta \mathbf{x}$ . In some cases  $t = 1$  is used, under the assumption that the quadratic model is well formed and that the solution  $\Delta \mathbf{x}$  is acceptable.



**Figure 2.3:** Major components of the optimisation algorithm. For the Nonlinear Optimisation on the System Graph the major components are the building of the quadratic model and the linear solving approach.

### 2.4.1 Step Based Approach

Newton's method in equation 2.2 above solves for a *step* in  $\mathbf{x}$ ,  $\Delta\mathbf{x}$ , rather than solving for an absolute solution. The step based approach introduces the *current* estimate, notated as  $\mathbf{x}_0$ , as a tool to aid the solution process.

The reasons for adopting this approach are as follows:

- As the solution tends towards its final, optimum value, both the right-hand-side vector ( $\nabla f(\mathbf{x})$ ) and solution  $\Delta\mathbf{x}$  tend toward zero. This allows a norm of  $\nabla f(\mathbf{x})$  or  $\Delta\mathbf{x}$  to serve as an indicator of the quality of the solution and a termination criterion. This also allows the use of sparse methods to exploit any actual or approximate zeros in  $\nabla f(\mathbf{x})$  or  $\Delta\mathbf{x}$ .
- For rank deficient  $\mathbf{A}$ , the system can be solved in a damped (also known as regularised) or minimum norm manner, which biases  $\Delta\mathbf{x}$  toward zero. In the

case of solving for  $\Delta \mathbf{x}$ , no permanent bias is introduced into the solution  $\mathbf{x}$ . Instead, the *steps* are moderately attenuated. However, in the case of solving absolutely for  $\mathbf{x}$ , a damped or minimum norm approach biases the solution towards  $\mathbf{x} = \mathbf{0}$  permanently.

### 2.4.2 Solving Linear Systems

This section describes the process for direct solving of linear systems in general terms, and introduces the  $\mathbf{LDL}^T$  factorisation. This solving process is significantly expanded in chapter 5.

The LDL factorisation helps solve linear systems ( $\mathbf{Ax} = \mathbf{b}$ ) by transforming  $\mathbf{A}$  into a product of triangular and diagonal systems, which are simpler to solve. The *diagonal pivoting*  $\mathbf{LDL}^T$  method ([14] and [37, page 168]) factorises  $\mathbf{A}$  as follows:

$$\mathbf{PAP}^T = \mathbf{LDL}^T \quad (2.3)$$

- $\mathbf{A}$  is the input square, symmetric matrix.
- $\mathbf{P}$  acting on  $\mathbf{A}$  via  $\mathbf{PAP}^T$  represents a symmetric permutation of  $\mathbf{A}$ .  $\mathbf{P}$  encodes the *factorisation ordering*.
- $\mathbf{L}$  is a unit lower triangular matrix.
- $\mathbf{D}$  is a block diagonal matrix.

The diagonal pivoted  $\mathbf{LDL}^T$  factorisation is based on the following block partitioning of  $\mathbf{A}$ :

$$\mathbf{PAP}^T = \begin{bmatrix} \mathbf{E} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{B} \end{bmatrix} \begin{matrix} s \\ n-s \end{matrix} \quad (2.4)$$

$s \quad n-s$

- $\mathbf{E}$  is the block which will be factorised out of  $\mathbf{A}$ .  $\mathbf{E}$  is built from  $s$  variables from  $\mathbf{A}$ , permuted into a block via  $\mathbf{P}$ .
- $\mathbf{C}$  is the symmetric matrix off diagonal block connecting  $\mathbf{E}$  to the rest of the system  $\mathbf{B}$ .

The block factorisation is then written as:

$$\mathbf{PAP}^T = \mathbf{LDL}^T \quad (2.5)$$

$$= \begin{bmatrix} \mathbf{I}_s & \mathbf{0} \\ \mathbf{CE}^{-1} & \mathbf{I}_{n-s} \end{bmatrix} \begin{bmatrix} \mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} - \mathbf{CE}^{-1}\mathbf{C}^T \end{bmatrix} \begin{bmatrix} \mathbf{I}_s & \mathbf{E}^{-1}\mathbf{C}^T \\ \mathbf{0} & \mathbf{I}_{n-s} \end{bmatrix} \quad (2.6)$$

The full factorisation continues by factorising  $\mathbf{A}_2 = \mathbf{B} - \mathbf{CE}^{-1}\mathbf{C}^T$  in the same manner, using Equation 2.6.

## LDL Solve

Using the factorisation  $\mathbf{LDL}^T = \mathbf{A}$ , the solution involves a sequence of solves as follows:

To solve	$\mathbf{Ax} = \mathbf{b}$	for $\mathbf{x}$ ,	
using factorisation	$\mathbf{LDL}^T \mathbf{x} = \mathbf{b}$		
1. solve:	$\mathbf{Lu} = \mathbf{b}$	for $\mathbf{u}$ , where $\mathbf{u} = \mathbf{DL}^T \mathbf{x}$	▣
2. solve:	$\mathbf{Dr} = \mathbf{u}$	for $\mathbf{r}$ , where $\mathbf{r} = \mathbf{L}^T \mathbf{x}$	▣
3. solve:	$\mathbf{L}^T \mathbf{x} = \mathbf{r}$	for $\mathbf{x}$	▣

Where ▣ indicates a triangular (directed-acyclic) solve and ▣ indicates a block diagonal solve.

Overall the process can be written as:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \quad (2.7)$$

$$\mathbf{x} = \mathbf{L}^{-T}\mathbf{D}^{-1}\mathbf{L}^{-1}\mathbf{b} \quad (2.8)$$

Each matrix inversion is a *notation* to indicate the required solve stages rather than explicit *inversion*.

## Factorisation versus Marginalisation

Marginalisation is equivalent to factorisation, but discards the factorised variables along with the  $\mathbf{D}$  and  $\mathbf{L}$  entries necessary to retrieve them again.

With marginalisation the aim is to reject a portion of the problem which is no longer interesting, while reflecting its influence back onto the rest of the system. Factorisation is more general since it includes the expression for the marginal, as well as the  $\mathbf{L}$  and  $\mathbf{D}$  entries which are used for the (optional) back solve to the “factorised-out” part. This allows the system to keep an active subset of states, while still having the option for returning to the inactive states.

This is the difference between marginalisation and factorisation. The process of marginalisation implicitly forms a triangular, block  $\mathbf{LDL}^T$  system, but then discards a portion of the  $\mathbf{L}$  and  $\mathbf{D}$  matrices to retain only the marginal and not the rest of the factorisation.

## Factorisation Ordering

The factorisation ordering describes the sequence in which variables are eliminated from the problem. In the solving stage, the (reversed) factorisation order describes the order of recovery of variables. Variables recovered early are used to recover later variables. Ideally this pattern of recovering variables, and subsequently using these to recover later variables follows a sparse pattern and is numerically stable.

## 2.5 Summary

Algorithm 1 outlines the main estimation loop in which the methods of this thesis are applied.

**Algorithm 1:** Main estimation loop

```

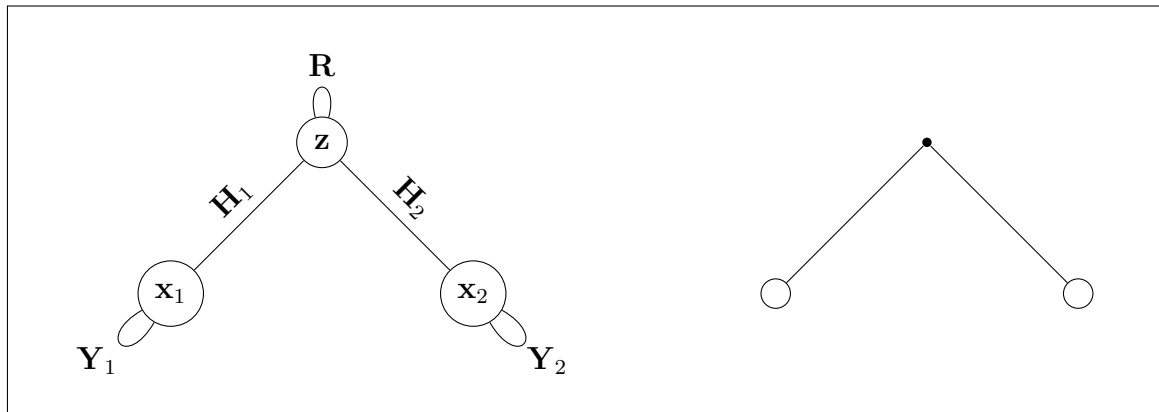
begin
  for Each timestep do
    Obtain sensor data observations
    for Each data association candidate do
      Add observations system according to data-association
      for Each iteration of nonlinear solution do
        Update the linearisation for changed entries
        Update the factorisation
        Linear solve  $\mathbf{A} \Delta \mathbf{x} = \mathbf{b}$  for the Newton direction
        for Each point of a line search: do
          Update parameterisations for changed entries
          Re-evaluate gradient terms for changed entries
        end
      end
      Optimise data associations over residual Mahalanobis distance
    end
  end

```

## 2.6 Graph Notation

Various examples in this thesis will show a set of states, some observations and their interconnections. In order to introduce the notation for these, consider a system with states  $\mathbf{x}_1$  and  $\mathbf{x}_2$ , and an observation  $\mathbf{z}$  with observation noise covariance  $\mathbf{R}$  and observation Jacobian  $\mathbf{H} = [\mathbf{H}_1 \quad \mathbf{H}_2]$  with respect to the two states. Suppose that the states have independent prior information  $\mathbf{Y}_1$  and  $\mathbf{Y}_2$  respectively. Figure 2.4 functions as a notation for indicating the structure of various examples but also contains some of the essential aspects of the augmented system form and graph embedded representation described in this thesis. Chapter 3 discusses the mathematical basis for this representation. Chapter 4 discusses the graph representation of systems in greater detail.





**Figure 2.4:** Graph notation for example systems, full details (left) and simplified schematic (right). This functions as a notation for indicating the structure of various examples in this thesis.

# Chapter 3

## Augmented Methods in Estimation

### 3.1 Introduction

This chapter proposes a generalisation of the information form by augmenting observations & constraints in addition to the states. The augmented system form is derived in section 3.2. The Lagrangian quadratic is introduced as the mathematical background of the approach, and the equivalence of the augmented system form to the information form is shown.

This form is referred to as the augmented system form, since it augments both observations and states. This thesis argues that the augmented system form is a more general starting point for estimation than the information form, allowing a wider range of alternative solution strategies.

The augmentation approach has benefits in the ability to retain and utilise the original nonlinear functions which define the estimation problem. In addition, the augmented system form has benefits in the flexibility of re-linearisation.

Depending on specific conditions of the system, the augmented system form can show improved sparsity and numerical performance than the information form. Since the augmented system form is easily reduced down to the information form, this thesis argues that the augmented system form should be formed first, followed by elimination

or factorisation of the observation and state variables in an appropriate, specifically considered order.

The augmented system form is complementary to the graph representation of the next chapter. Both the augmented system form and the graph structure favour using entries for all of the state and observation variables modelled by the system and their interconnections.

The augmented system form brings new insights into the nature of data fusion by writing out the roles of the observations, states and their interaction explicitly (see section 3.5). The augmented system form satisfies the requirement for a mathematical description of the estimation problem showing explicitly and separately the states and observations together with a cross-coupling interaction.

The contributions of this chapter are as follows:

- The augmented observation form is introduced in an estimation context and the relationships between estimation problems and the augmented system method are developed.
  - ▷ The augmented system method is developed in an estimation context with observations rather than from an equality constraint point of view. This results in a novel Lagrangian formulation, including a new term which generalises between observations and constraints.
  - ▷ The relationship between the augmented system method and the information form is shown. The information form is shown to be a marginalised reduction of the augmented system form. Whereas the augmented system form describes both the observation and state variables jointly, the information form describes only the state variables.
  - ▷ The augmented system form complements the trajectory state augmentation approach (section 3.3). Both of these approaches have similar goals and present similar challenges, and both are complementary to the graph based structure described later, in chapter 4. The trajectory state form is discussed because the augmented system form operates in the context of the

trajectory state form. The augmentation of observations makes sense only when the associated states are also augmented. The graph structure and augmented system method are both designed in context of the trajectory state approach.

- ▷ The observation-augmented form brings new insights into the nature of data fusion. The observation-augmented form shows both observations and states explicitly and shows their interaction via a shared Lagrange multiplier, rather than being expressed as an addition of information.
- ▷ Generalised extensions to the concept of innovations and innovation distances are presented under these additional augmented terms. The augmented system form has an inherent ability to evaluate network-wide residuals, not simply the innovations of new observations against current predictions as is conventional in filtering approaches. Given this, this thesis contributes a novel form of Mahalanobis distance, which is equivalent to the conventional innovation distance but offers additional generality, including the ability to operate with rank deficient information terms and multiple observation terms.
- The augmented system form is complementary to the graph structure for the representation of the problem structure and its linearised form.
  - ▷ Both the full-structural form and the graph structure use entries for all of the state and observation variables modelled by the system and provide facilities to represent their interconnections.
  - ▷ The full structural properties of the augmented system form are complementary to the explicit graph structure used to represent the system.
  - ▷ Together, the graph structure and the augmented observation approach form the necessary data structures to contain the full formulation of the estimation problem in the implementation.
- The benefits of using the augmented observation form for estimation problems are shown.

- ▷ The augmented system form allows a wider range of factorisation orderings than the information form. In the augmented system form, variables may be factorised in any order mixing observation and state variables, whereas the information form implicitly eliminates the observation variables first.
- ▷ Depending on specific conditions of the system, the augmented system form shows improved sparsity and numerical performance than the information form.
- ▷ Since the augmented system form is able to be reduced down to the information form, this thesis argues that the augmented observation form provides a more general starting point for estimation algorithms and that the augmented observation form should be formed first. The solving method may then generally consider the specific order of elimination of variables.
- ▷ In other words, *if* the information form is used, the “formation” of the information form  $(\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H})$ , given the set of observation Jacobians ( $\mathbf{H}$ ) and observation covariances ( $\mathbf{R}$ ) is equivalent to the *elimination* of the observation Lagrange multiplier variables out of the augmented system form. The use of the information form is therefore a “fixed-marginalisation policy” in the variables. This thesis argues that systems should not take a fixed elimination or factorisation policy and instead evaluate the ordering in the states and observations at runtime for the particular cases encountered.
- ▷ The factorisation or elimination of observations first, as in the information form, can adversely affect the numerical stability if the observations have small (or zero) noise covariance. In this case the augmented system form is able to factorise some states first, resulting in improved numerical stability.
- ▷ The augmented system form separately represents the observations and the states. This has the benefit that the Jacobians for each observation are separate from each other, enabling simple access for re-linearisation.
- ▷ The augmented system form is a jointly solvable system for the states and observations, rather than just a data structure. This means that the augmented system form describes the operations necessary to solve it from

both observations and states, whereas a simple data structure connecting observations to and from states would not imply a jointly solvable system and would instead require manual conversion to a solvable system (to the information form, for example).

- ▷ The augmented observation approach has benefits in the ability to retain and utilise the original nonlinear functions which define the estimation problem. The augmented system form has benefits in the flexibility of relinearisation, due to its property of avoiding marginalising observation terms into the states.

The augmented system form described in this chapter is designed to operate in conjunction with the graph embedded system representation described in chapter 4 and with the graph embedded solving algorithm described in chapter 5. The augmented system form provides a mathematical system representing both observations and states while the graph structure provides a fundamental tool for representing sparse relationships between variables generally. The graph structure helps by efficiently representing the observations and states, and the sparse observation-state link structure and its associated Jacobian entries.

## 3.2 Augmenting Observations and Constraints

This section derives the equations for the augmented observation and constraint system. The derivation starts with an estimation problem containing an *observation* term and later generalises this to include equality *constraints*. This section provides an initial description of *what* the augmented system is and what it involves. The justification for using the augmented system is given in section 3.7 and throughout this chapter.

For clarity, the explanation will initially consider a linear estimation problem. The case of a nonlinear estimation problem is described in section 3.2.7.

### 3.2.1 Information Formulation

1. Consider the linear estimation problem consisting of state  $\mathbf{x}$ , with:

- A prior information term with inverse-covariance  $\mathbf{Y}$  at a prior estimate of  $\mathbf{x}_p$ .
- A linear observation  $\mathbf{z} = \mathbf{h}(\mathbf{x}) + \mathbf{w} = \mathbf{H}\mathbf{x} + \mathbf{w}$  with unknown zero mean observation noise,  $\mathbf{w}$ , with covariance,  $\mathbf{R}$ . This will be expressed in terms of the residual,  $\mathbf{h}(\mathbf{x}) - \mathbf{z}$ .

$\mathbf{H}$  is the linear observation matrix, the derivative of  $\mathbf{h}(\mathbf{x})$  with respect to  $\mathbf{x}$ .  $\mathbf{H} = \frac{\partial}{\partial(\mathbf{x})}\mathbf{h}(\mathbf{x})$ .  $\mathbf{H} \in \mathbb{R}^{n_{\text{obs}} \times n_{\text{state}}}$ .

2. The objective function for this estimation problem is given by:

$$\mathbf{x}_e = \arg \min F(\mathbf{x}) \quad (3.1)$$

$$F(\mathbf{x}) = \underbrace{\frac{1}{2}(\mathbf{h}(\mathbf{x}) - \mathbf{z})^T \mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}) - \mathbf{z})}_{\text{Observation}} + \underbrace{\frac{1}{2}(\mathbf{x} - \mathbf{x}_p)^T \mathbf{Y}(\mathbf{x} - \mathbf{x}_p)}_{\text{Prior}} \quad (3.2)$$

- $\mathbf{x}_e$  is the solution for the state estimate.  $\hat{\mathbf{x}} \in \mathbb{R}^{n_{\text{state}}}$
- $\mathbf{x}$  is any point in the state, acting as a function argument.  $\mathbf{x} \in \mathbb{R}^{n_{\text{state}}}$
- $F(\mathbf{x})$  is the objective function.  $F : \mathbb{R}^{n_{\text{state}}} \rightarrow \mathbb{R}$ .
- $\mathbf{z}$  is the known, obtained observation vector.  $\mathbf{z} \in \mathbb{R}^{n_{\text{obs}}}$
- $\mathbf{R}$  is the observation covariance matrix.  $\mathbf{R} \in \mathbb{R}^{n_{\text{obs}} \times n_{\text{obs}}}$ .  $\mathbf{R}^{-1}$  is initially assumed to exist (as required for the above expressions), but this requirement is later relaxed as the observations are generalised into constraints.
- $\mathbf{Y}$  is the Hessian or Information matrix of the prior information term.  $\mathbf{Y} \in \mathbb{R}^{n_{\text{state}} \times n_{\text{state}}}$
- $\mathbf{x}_p$  is the centre estimate of the prior information term.  $\mathbf{x}_p \in \mathbb{R}^{n_{\text{state}}}$

3. Since the observation  $\mathbf{h}(\mathbf{x})$  is linear, and the prior term is quadratic,  $F(\mathbf{x})$  is

quadratic.  $F(\mathbf{x})$  can be written in incremental form as:

$$F(\mathbf{x}_0 + \Delta\mathbf{x}) = F(\mathbf{x}_0) + \nabla F(\mathbf{x}_0)^T \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \nabla^2 F(\mathbf{x}_0) \Delta\mathbf{x} \quad (3.3)$$

The Jacobian and Hessian of  $F(\mathbf{x})$  are:

$$\nabla F(\mathbf{x}) = \mathbf{H}^T \mathbf{R}^{-1} (\mathbf{h}(\mathbf{x}) - \mathbf{z}) + \mathbf{Y}(\mathbf{x} - \mathbf{x}_p) \quad (3.4)$$

$$\nabla^2 F(\mathbf{x}) = \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} + \mathbf{Y} \quad (3.5)$$

The incremental form for the gradient of  $F(\mathbf{x})$  is:

$$\nabla F(\mathbf{x}_0 + \Delta\mathbf{x}) = \nabla F(\mathbf{x}_0) + \nabla^2 F(\mathbf{x}_0) \Delta\mathbf{x} \quad (3.6)$$

The solution,  $\Delta\mathbf{x}$ , satisfies:

$$\nabla F(\mathbf{x}_0 + \Delta\mathbf{x}) = 0 \quad (3.7)$$

The result yields the information form expressions for the solution,  $\Delta\mathbf{x}$ , also known as the *Newton step* [12].

$$\nabla^2 F(\mathbf{x}_0) \Delta\mathbf{x} = -\nabla F(\mathbf{x}_0) \quad (3.8)$$

With equations 3.4 and 3.5 this becomes *the information form*:

$$(\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} + \mathbf{Y}) \Delta\mathbf{x} = -(\mathbf{H}^T \mathbf{R}^{-1} (\mathbf{h}(\mathbf{x}_0) - \mathbf{z}) + \mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p)) \quad (3.9)$$

Equation 3.9 shows how the observation  $\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}$  is added onto the prior information  $\mathbf{Y}$  (and similarly in the vector components on the right-hand-side). This indicates that the observation information is *merged* into the prior information (forming the posterior information).



The prior information,  $\mathbf{Y}$  is not treated *differently* from the observation information. Instead, the prior information term is included separately in order to show how the observation information is *merged into* the prior information in this information-form approach.

### 3.2.2 Lagrangian Formulation

In this section, the derivation moves to the *Lagrangian formulation*. This will be justified in the rest of this chapter by relating it back to the objective function and information form. The Lagrangian formulation leads to the augmented system form.

This Lagrangian formulation is a novel contribution of this thesis due to a novel term which generalises between observations and constraints.

Consider the following Lagrangian function,  $L(\mathbf{x}, \boldsymbol{\nu})$  (see figure 3.1 for a diagram):

$$L(\boldsymbol{\nu}, \mathbf{x}) = \underbrace{\frac{1}{2}(\mathbf{x} - \mathbf{x}_p)^T \mathbf{Y}(\mathbf{x} - \mathbf{x}_p)}_{\text{State-functional}} - \underbrace{\frac{1}{2}\boldsymbol{\nu}^T \mathbf{R}\boldsymbol{\nu}}_{\text{Constraint relaxation}} - \underbrace{\boldsymbol{\nu}^T(\mathbf{h}(\mathbf{x}) - \mathbf{z})}_{\text{Weighted constraint}} \quad (3.10)$$

- $\boldsymbol{\nu}$  and  $\mathbf{x}$  are independent variables.  $\boldsymbol{\nu} \in \mathbb{R}^{n_{\text{obs}}}$ .  $\mathbf{x} \in \mathbb{R}^{n_{\text{state}}}$ .

- $L(\boldsymbol{\nu}, \mathbf{x})$  is a scalar function from vector variables  $\boldsymbol{\nu}$  and  $\mathbf{x}$ .

$$L : \mathbb{R}^{n_{\text{obs}}} \times \mathbb{R}^{n_{\text{state}}} \rightarrow \mathbb{R}.$$

$L(\boldsymbol{\nu}, \mathbf{x})$  consists of three terms:

- A “state-functional” term. In general this state-functional term will contain any objective terms relating only to the state, not relating to the observations or constraints. However, in this *particular derivation example*, this term is the prior information quadratic objective term. It is necessary to place the prior information in this term because this enables a direct comparison with the information form approach.<sup>1</sup>

---

<sup>1</sup>If desired, one could model “prior information” in a uniform manner with observations by writing them as “identity observations” of the state.

- A “weighted constraint” term. This term multiplies each scalar observation or constraint term,  $h_i(\mathbf{x} - z_i)$ , by a Lagrange multiplier scalar  $\nu_i$ .
- A “constraint relaxation” term. The result of this term is that *observations* with nonzero  $\mathbf{R}$  are allowed to relax away from  $\mathbf{h}(\mathbf{x}) - \mathbf{z} = \mathbf{0}$ . The amount of relaxation is controlled by  $\mathbf{R}$ . This term is a novel contribution, which generalises between both observations and constraints.

The partial derivatives of  $L(\boldsymbol{\nu}, \mathbf{x})$  are:

$$\nabla_{\boldsymbol{\nu}} L = -\mathbf{R}\boldsymbol{\nu} - (\mathbf{h}(\mathbf{x}) - \mathbf{z}) \quad (3.11)$$

$$\nabla_{\mathbf{x}} L = \mathbf{Y}(\mathbf{x} - \mathbf{x}_p) - \mathbf{H}^T \boldsymbol{\nu} \quad (3.12)$$

A *necessary* condition for the solution  $\boldsymbol{\nu}$  and  $\mathbf{x}$  is that they are a *stationary point* of the Lagrangian in (3.10).

$$\nabla_{\boldsymbol{\nu}} L = \mathbf{0} \qquad \qquad \nabla_{\mathbf{x}} L = \mathbf{0} \quad (3.13)$$

The stationary point of the Lagrangian corresponds to a *saddle point* as opposed to an optimum point of the information quadratic or point of *maximum* probability density. The solution lies at the *saddle point* of the Lagrangian because the Lagrangian simultaneously represents both a *maximisation* and a *minimisation* relating to the observation Lagrange multipliers and the states. These need to be in opposite signs so that the marginalisation of observations *adds* information into the states instead of subtracting it.

The *stationary point* conditions in equation 3.13 generalise the stationary-point requirements for an *extrema*. In this case,  $\nabla_{\boldsymbol{\nu}} L = \mathbf{0}$  means that the solution must be an extrema with respect to variation in  $\boldsymbol{\nu}$  and  $\nabla_{\mathbf{x}} L = \mathbf{0}$  means that the solution must be an extrema with respect to variation in  $\mathbf{x}$ .

Since  $L$  is a quadratic, the solution to meet conditions 3.13 is given by the Newton

step:

$$\nabla^2 L(\boldsymbol{\nu}_0, \mathbf{x}_0) \Delta \begin{bmatrix} \boldsymbol{\nu} \\ \mathbf{x} \end{bmatrix} = -\nabla L(\boldsymbol{\nu}_0, \mathbf{x}_0) \quad (3.14)$$

$$\nabla^2 L(\boldsymbol{\nu}_0, \mathbf{x}_0) = - \begin{pmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{pmatrix} \quad (3.15)$$

$$\nabla L(\boldsymbol{\nu}_0, \mathbf{x}_0) = - \begin{pmatrix} \mathbf{R}\boldsymbol{\nu}_0 + (\mathbf{h}(\mathbf{x}_0) - \mathbf{z}) \\ \mathbf{H}^T \boldsymbol{\nu}_0 - \mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p) \end{pmatrix} \quad (3.16)$$

The result is the following augmented system form, written in incremental form:

$$\boxed{\begin{pmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{pmatrix} \begin{pmatrix} \Delta \boldsymbol{\nu} \\ \Delta \mathbf{x} \end{pmatrix} = - \begin{pmatrix} \mathbf{R}\boldsymbol{\nu}_0 + (\mathbf{h}(\mathbf{x}_0) - \mathbf{z}) \\ \mathbf{H}^T \boldsymbol{\nu}_0 - \mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p) \end{pmatrix}} \quad (3.17)$$

The augmented system form in equation 3.17 is the focus of this chapter.

- The left-hand-side of augmented system form,  $\begin{pmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{pmatrix}$  is a square, symmetric linear system representing the solution of the Lagrangian  $L(\boldsymbol{\nu}, \mathbf{x})$ . It has dimensions  $(n_{\text{obs}} + n_{\text{state}})^2$ .
- The right-hand-side is written in the incremental form where  $\boldsymbol{\nu}_0$  and  $\mathbf{x}_0$  are any initial values of the entire state before solving. When the system is at a *solution*,

the right-hand-side equals zero. At the solution  $\boldsymbol{\nu}$  and  $\mathbf{x}$ :

$$\begin{aligned}\mathbf{R}\boldsymbol{\nu} + (\mathbf{h}(\mathbf{x}) - \mathbf{z}) &= \mathbf{0} \\ \mathbf{H}^T\boldsymbol{\nu} - \mathbf{Y}(\mathbf{x} - \mathbf{x}_p) &= \mathbf{0}\end{aligned}$$

This indicates that, at the solution, the observation residual  $(\mathbf{h}(\mathbf{x}) - \mathbf{z})$  is balanced by the Lagrange multiplier through  $\mathbf{R}\boldsymbol{\nu}$ . For  $\mathbf{R} = \mathbf{0}$  it must have  $(\mathbf{h}(\mathbf{x}) - \mathbf{z}) = \mathbf{0}$ . The residual of the prior information  $\mathbf{Y}(\mathbf{x} - \mathbf{x}_p)$  is balanced by the Lagrange multiplier through  $\mathbf{H}^T\boldsymbol{\nu}$ .

The explanation below considers the relationship between the Lagrangian  $L(\boldsymbol{\nu}, \mathbf{x})$  and the quadratic  $F(\mathbf{x})$ . This relationship is obtained by requiring  $\nabla_{\boldsymbol{\nu}}L = \mathbf{0}$  to be satisfied. This requirement is one *part* of the complete requirement for the solution stated in equation 3.13. This yields a relationship from  $\mathbf{x}$  to  $\boldsymbol{\nu}$ , a function:  $\tilde{\boldsymbol{\nu}}(\mathbf{x})$ .

$$\tilde{\boldsymbol{\nu}}(\mathbf{x}) = -\mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}) - \mathbf{z}) \quad (3.18)$$

Under this functional relationship for  $\boldsymbol{\nu}$ ,  $L$  becomes a function of  $\mathbf{x}$  only. The resulting function  $L(\mathbf{x})$  is identically  $F(\mathbf{x})$ .

$$L(\boldsymbol{\nu}, \mathbf{x}) \text{ subject to } \{\boldsymbol{\nu} = \tilde{\boldsymbol{\nu}}(\mathbf{x})\} \implies L(\tilde{\boldsymbol{\nu}}(\mathbf{x}), \mathbf{x}) \quad (3.19)$$

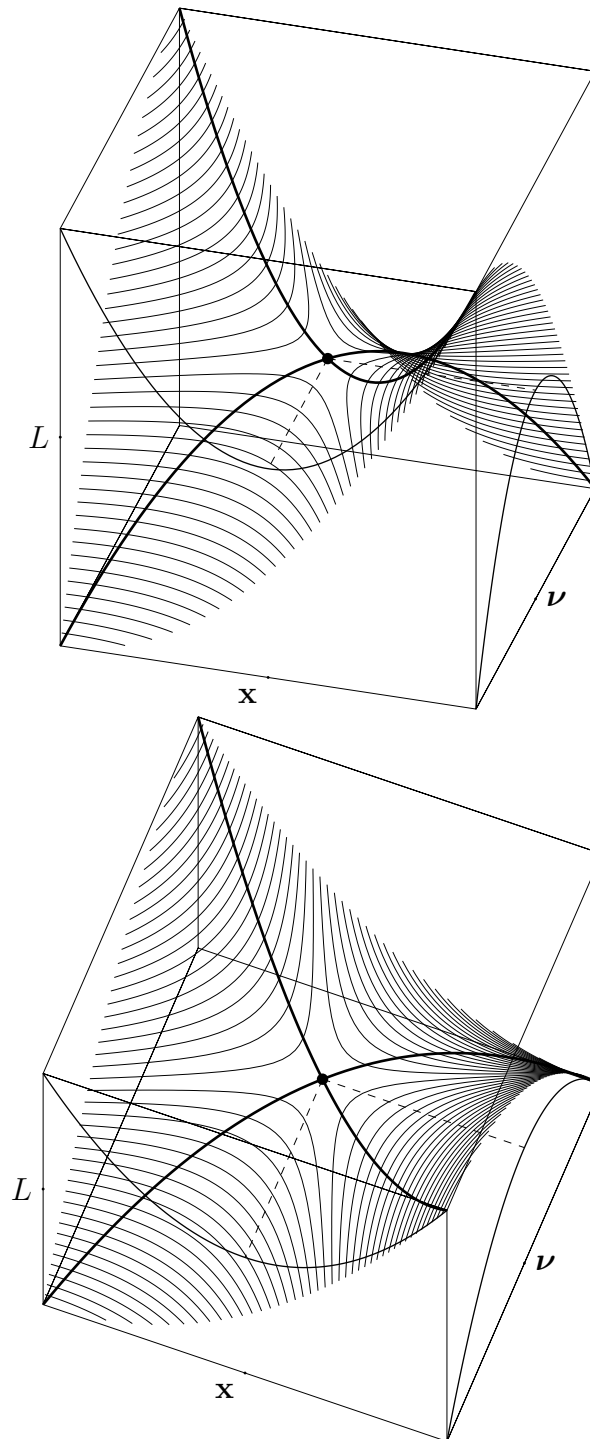
$$\implies F(\mathbf{x}) \quad (3.20)$$

In figure 3.1,  $L(\boldsymbol{\nu}, \mathbf{x})$  subject to  $\{\boldsymbol{\nu} = \tilde{\boldsymbol{\nu}}(\mathbf{x})\}$  is the concave-up dark line. It's projection into  $\mathbf{x}$  is identically  $F(\mathbf{x})$ , which is proven as follows:

$$\begin{aligned}
L(\boldsymbol{\nu}, \mathbf{x}) &= \frac{1}{2}(\mathbf{x} - \mathbf{x}_p)^T \mathbf{Y}(\mathbf{x} - \mathbf{x}_p) - \frac{1}{2}\boldsymbol{\nu}^T \mathbf{R}\boldsymbol{\nu} - \boldsymbol{\nu}^T(\mathbf{h}(\mathbf{x}) - \mathbf{z}) \\
L(\tilde{\boldsymbol{\nu}}(\mathbf{x}), \mathbf{x}) &= \frac{1}{2}(\mathbf{x} - \mathbf{x}_p)^T \mathbf{Y}(\mathbf{x} - \mathbf{x}_p) - \frac{1}{2}\tilde{\boldsymbol{\nu}}(\mathbf{x})^T \mathbf{R}\tilde{\boldsymbol{\nu}}(\mathbf{x}) - (\mathbf{h}(\mathbf{x}) - \mathbf{z})^T \tilde{\boldsymbol{\nu}}(\mathbf{x}) \\
&= \frac{1}{2}(\mathbf{x} - \mathbf{x}_p)^T \mathbf{Y}(\mathbf{x} - \mathbf{x}_p) - \frac{1}{2}\tilde{\boldsymbol{\nu}}(\mathbf{x})^T \mathbf{R}\tilde{\boldsymbol{\nu}}(\mathbf{x}) + (\mathbf{h}(\mathbf{x}) - \mathbf{z})^T \mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}) - \mathbf{z}) \\
&= \frac{1}{2}(\mathbf{x} - \mathbf{x}_p)^T \mathbf{Y}(\mathbf{x} - \mathbf{x}_p) + \frac{1}{2}\tilde{\boldsymbol{\nu}}(\mathbf{x})^T (\mathbf{h}(\mathbf{x}) - \mathbf{z}) + (\mathbf{h}(\mathbf{x}) - \mathbf{z})^T \mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}) - \mathbf{z}) \\
&= \frac{1}{2}(\mathbf{x} - \mathbf{x}_p)^T \mathbf{Y}(\mathbf{x} - \mathbf{x}_p) + \frac{1}{2}(\mathbf{h}(\mathbf{x}) - \mathbf{z})^T \mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}) - \mathbf{z}) \\
&= F(\mathbf{x})
\end{aligned}$$

The result of this section is that the Lagrangian of equation 3.10 generalises the quadratic objective function of equation 3.2. The Lagrangian explicitly separates the observation Lagrange multiplier variables from the state variables, and was shown to reduce back to the conventional objective function in the states only.

The above derivation shows the formation of the augmented system consisting of the state prior term in  $\mathbf{Y}$  and  $\mathbf{x}_p$  and the linear observation with  $\mathbf{R}$ ,  $\mathbf{H}$  and  $\mathbf{z}$ . Section 3.2.7 notes how the expressions are modified for nonlinear observations. Section 3.2.3 describes how this augmented form is suited to the generalisation of observations into constraints.



**Figure 3.1:** Two views of contours of the Lagrangian surface. The solution in  $\mathbf{x}$  and  $\nu$  (circled) is the stationary point on the Lagrangian. The two dark lines indicate solutions to the partial derivatives  $\nabla_{\nu} L = \mathbf{0}$  (concave up) and  $\nabla_{\mathbf{x}} L = \mathbf{0}$  (concave down). The quadratic in the  $(\mathbf{x}, L)$  space is the projection of the line  $\nabla_{\nu} L = \mathbf{0}$  into  $\mathbf{x}$ , which is the quadratic cost function  $F(\mathbf{x})$ .

### 3.2.3 Constraints

This section describes the generalisation of observations into constraints, in relation to the augmented form. Constraints are the subset of observations with zero  $\mathbf{R}$  (or more generally, singular  $\mathbf{R}$ ). At small but nonzero  $\mathbf{R}$ , the term may also be described as a “relaxed constraint” or a “tight observation”.

This thesis considers linear equality constraints. Inequality constraints require more general methods from convex optimisation [12]. Nonlinear equality constraints deviate away from convexity; The methods used for linear equality constraints can be applied with linearisation of the constraints under the assumption of approximate convexity.

Constraints appear whenever the system contains any observation or prediction model or mathematical requirement that is perfect or deterministic. Constraints are also an abstract generalisation of observations, taking the limit as the uncertainty tends to zero.

Using an observation with small but nonzero  $\mathbf{R}$  results in a small but nonzero deviation from the constraint, if pulled away by other terms. In other words, for nonzero  $\mathbf{R}$ , a finite value of the Lagrange multiplier,  $\boldsymbol{\nu}$  must arise from a finite deviation of the constraint, whereas a genuine constraint with zero  $\mathbf{R}$  can have any value of  $\boldsymbol{\nu}$  despite a zero deviation.

**Example 3.1.*****The elimination of constraint-deviation bias using equality constraints***

Consider a two dimensional state,  $x$  and  $y$  subject to a prior information term and an observation/constraint. The observation/constraint will be parametrised by  $R = \epsilon$  where finite  $R > 0$  corresponds to an observation and  $R = 0$  corresponds to a constraint.

For the observation/constraint:  $\mathbf{H} = \begin{bmatrix} 1 & -1 \end{bmatrix} \quad z = 2 \quad R = \epsilon$

The residual is:  $(\mathbf{H}\mathbf{x} - z) = (x - y - 2)$

For the prior information term:  $\mathbf{Y} = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \quad \mathbf{x}_p = \begin{bmatrix} 4 & 4 \end{bmatrix}^T$

The augmented system form is:

$$\begin{pmatrix} R & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{pmatrix} \begin{pmatrix} \nu \\ \mathbf{x} \end{pmatrix} = \begin{pmatrix} z \\ -\mathbf{Y}\mathbf{x}_p \end{pmatrix} \quad (3.21)$$

$$\begin{pmatrix} \epsilon & 1 & -1 \\ 1 & -3 & 0 \\ -1 & 0 & -3 \end{pmatrix} \begin{pmatrix} \nu \\ x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ -12 \\ -12 \end{pmatrix} \quad (3.22)$$

The solution is:  $\nu = \frac{6}{3\epsilon + 2} \quad x = \frac{12\epsilon + 6}{3\epsilon + 2} \quad y = \frac{12\epsilon + 10}{3\epsilon + 2} \quad (3.23)$

The residual is:  $\mathbf{H}\mathbf{x} - z = \frac{-6\epsilon}{3\epsilon + 2} \quad (3.24)$

For a pure constraint ( $\epsilon = 0$ ), the residual is zero ( $\mathbf{H}\mathbf{x} - z = 0$ ) indicating that the constraint is exactly satisfied. For small  $\epsilon$ , the residual is approximately  $-3\epsilon$ . Thus for tight-observations the residual is not exactly zero and some bias is introduced compared to using a true constraint.





Some reasons why constraints are important are described below:

### Deterministic Models

If an observation or prediction model has any deterministic component, then that component becomes a constraint.

An observation or prediction is modelled as a vector residual  $\mathbf{f}$  as a function of the state  $\mathbf{x}$ , obtained data  $\mathbf{z}$  and unknown noise  $\mathbf{w}$ :  $\mathbf{f}(\mathbf{x}, \mathbf{z}, \mathbf{w})$ . The covariance of the uncertainty of the residual is related through to the uncertainty of the noise injected into the model:

$$\mathbf{G} = \nabla_{\mathbf{w}} \mathbf{f} \quad (3.25)$$

$$E(\mathbf{w}\mathbf{w}^T) = \mathbf{Q} \quad (3.26)$$

$$\mathbf{R} = \mathbf{G}\mathbf{Q}\mathbf{G}^T \quad (3.27)$$

$\mathbf{R}$  becomes rank deficient (and therefore contains constraints) whenever  $\mathbf{Q}$  has fewer dimensions than  $\mathbf{R}$ . If the model  $\mathbf{Q}$  does not attribute noise to some components, these become constraints. Constraints are therefore introduced simply by the *absence* of modelled uncertainty.

Some examples of constraints include:

- Deterministic modelling of the relationship between successive positions and velocities.
- Zero lateral slip assumption in modelled dynamics of simple wheeled vehicles.

Constraints are important in these cases because the constraint terms will generally repeat over many timesteps and chain into each other. Genuine constraints are therefore important to prevent the accumulation of bias and reduced stiffness arising from using the alternative tight-observations.

### Agreement of separately modelled entities

Equality constraints can be used to enforce agreement between separate representations of a single entity, for example in local submaps [77] and in decentralisation estimation [62]. In decentralisation, entities are modelled at each estimation

node and are all required to agree. In data association, entities which may be initially distinct can be identified as being the same and forced into equality by linking them with equality constraints.

A single entity,  $\mathbf{x}$ , can be modelled multiple ( $n$ ) times (for example:  $\begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 \end{bmatrix}$ ), together with  $n - 1$  equality constraints indicating that these need to be identical:

$$\mathbf{H} = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \quad (3.28)$$

### Parametrisation constraints

Constraints are also required when groups of scalar states belong to a *constrained parametrisation*. For example:

- groups of 4 scalar states can belong to a quaternion parametrisation of an attitude, and as such are constrained to unit normalisation [45].
- pairs of 2 scalar states representing an angle are constrained to lie on the unit circle.

### Constraints in Localisation and Mapping Estimation

Constraints have been used in the localisation and mapping estimation literature, often for the purpose of binding together multiple instances of a single state. For example, [77] uses constraints to link features which are represented in both the global map and local submaps. This used the *covariance* form to implement the constraints, for which the covariance form is well suited.

Constraints applied in the *information form* are usually applied as tight (large information) observations. For example, [71] uses “infinite” information to represent the “anchoring constraint” necessary to constrain the initial pose, and [70] refers to a large information “soft correspondence constraint” used to bind two instances of a single state to represent a data association choice.

Constraints are also implied in the following practice: when two objects  $f_a$  and  $f_b$  are identified as being a single object, their links or information to measurements can be merged. This is equivalent to forming the equality constraint and eliminating the constraint and one of the two objects.

The problem with using large information to represent constraints is that it is numerically unstable and also does not enforce the constraints; a finite bias away from the constraint will exist if other terms affect the constrained states.

The augmented system form is able to fully represent constraints. Furthermore, the constraint Lagrange multiplier is obtained. If the constraint represents a data association choice, the Lagrange multiplier will show the amount of “force” necessary to enforce the constraint. This may be useful in future work to evaluate the consistency of supposed data association constraints via this Lagrange multiplier.

Constraints have also been applied in decentralised contexts for enforcing agreement (or more general convex relationships) among separately represented entities [62].

### 3.2.3.1 Constraints, Covariance, Information and Augmented Forms

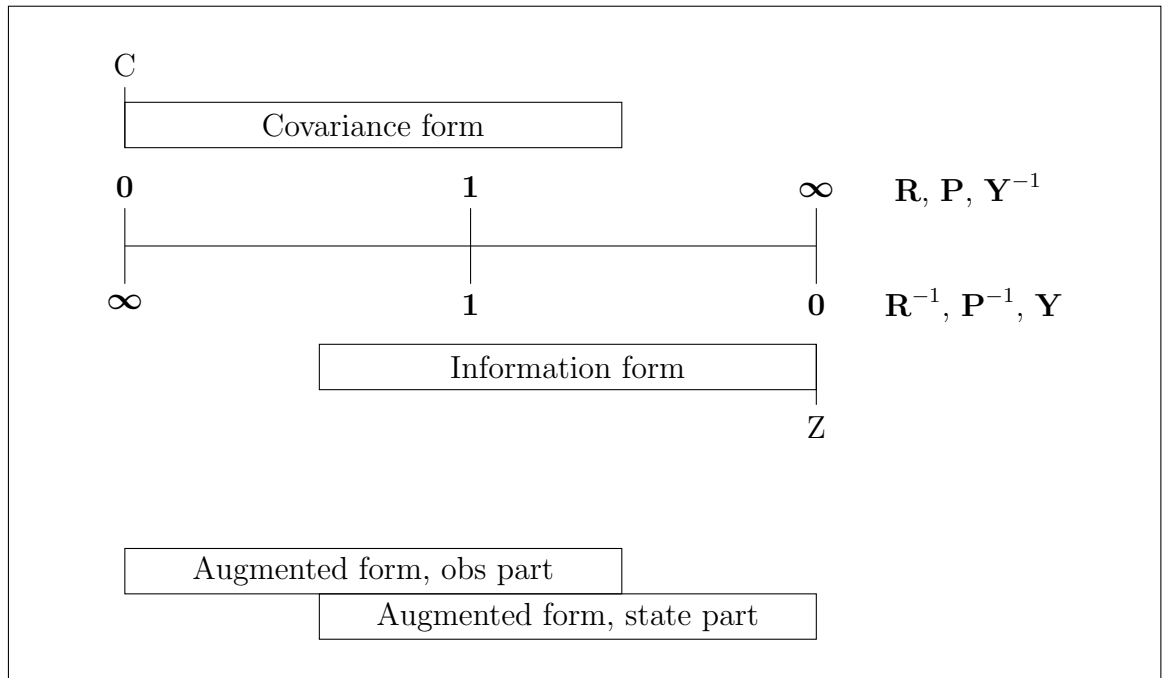
This section discusses constraints in the context of the covariance form, information form and the augmented system form. The point of this section is that constraints can be represented in the covariance form but not the information form. The technique which allows the augmented system form to include constraints is to use a dual representation containing both information and covariance terms.

Figure 3.2 shows the applicable ranges for the conventional covariance and information forms. The covariance form has the ability to represent constraints via a singular  $\mathbf{P}$ . This is not possible in the information form, where  $\mathbf{Y}$  does not exist for constrained terms (infinite information) or is numerically poorly conditioned, with large entries, for near-constraint terms. Similarly, zero information prior terms can be included in the information form, but in the covariance form these would require large covariance entries, resulting in poor numerical conditioning.

Constraints are easily represented in the augmented form, because the augmented form uses a *covariance* form for representing observations and constraints. Furthermore, the augmented form is able to represent zero-information terms. This section contrasts

this with the capabilities of the covariance form and information form.

The augmented form retains the observations in covariance form and the states in information form, and hence is able to include the case of both constraints and zero-information terms. The augmented form therefore achieves unification of observations and constraints. The zero covariance and zero information terms can coexist simultaneously. For example the system  $\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$  represents a scalar 1D state with zero prior information constrained in place by a (zero covariance) constraint.



**Figure 3.2:** The ranges 0 to  $\infty$  for covariance and information forms. The covariance axis spanning from 0 to  $\infty$  (top left to right) is the dual of the information axis spanning from  $\infty$  to 0 (bottom left to right). The covariance form covers from zero to finite covariance. Constraints are represented by ‘C’ at zero covariance (infinite information). The information form covers from zero to finite information. Zero information terms (eg priors) are represented by ‘Z’ at zero information. The augmented form, shown as the union of an ‘observation part’ and a ‘state part’ represents the observations in covariance form and the states in information form in a coupled manner.

### 3.2.3.2 Analytical Notes On Constraints

This section describes some analytical properties relating to the generalisation from observations to constraints.

Equation 3.18 is only valid in the *limit* as  $\mathbf{R}$  approaches zero and is undefined ( $\frac{0}{0}$ ) for constraints,  $\mathbf{R} = \mathbf{0}$ . For observations, equation 3.18 can be used to express  $\boldsymbol{\nu}$  in terms of  $\mathbf{x}$  and  $\mathbf{z}$ . However, for constraints both  $\mathbf{R}$  and  $\mathbf{H}\mathbf{x} - \mathbf{z}$  equal zero so  $\boldsymbol{\nu}$  cannot be determined from equation 3.18 but can instead be solved jointly using the augmented form. The Lagrange-multiplier  $\boldsymbol{\nu}$  therefore takes on a more significant role for constraints.

The above augmented observation system applies equally well to equality constraints as for observations. The Lagrangian, equation 3.10 expresses the observation in a form involving  $\mathbf{R}$  only (instead of  $\mathbf{R}^{-1}$ ). The augmented observation form inherently retains the  $\mathbf{R}$  expression throughout. In this way  $\mathbf{R}$  is not required to be invertible and reduces the constrained case when  $\mathbf{R} = \mathbf{0}$ .

The augmented system form also has the ability to handle *near*-constraints (also known as tight observations) in a manner which *smoothly* approaches the behaviour of constraints. No large transition in the approach required or in the values of variables occurs in shifting from an  $\mathbf{R} = \mathbf{0}$  absolute constraint to a near-constraint with  $\mathbf{R} = \boldsymbol{\epsilon}$  ( $\mathbf{R}$  infinitesimally small but positive definite).

### 3.2.4 Mixed Observations and Constraints

When  $\mathbf{R}$  is positive-definite, the term is described as an observation. When  $\mathbf{R}$  is zero, the term is described as a constraint. However,  $\mathbf{R}$  may also be positive-semi-definite, in which case the term is a linear combination of observations (non-zero eigenvalues of  $\mathbf{R}$ ) and constraints (zero valued eigenvalues of  $\mathbf{R}$ ). The observation/constraint term may also consist of an ill-conditioned  $\mathbf{R}$  where some eigenvalues approach zero.

In the case of prediction observations, the observation  $\mathbf{R}$  term is obtained from:

$$\mathbf{R} = \mathbf{G}\mathbf{Q}\mathbf{G}^T \quad (3.29)$$

$\mathbf{G}$  is a particular evaluation (linearisation) of the prediction residual's derivative with respect to any input noise terms. The term  $\mathbf{G}\mathbf{Q}\mathbf{G}^T$  depends on the specific conditions when evaluating the linearisation of the prediction model. In general,  $\mathbf{G}\mathbf{Q}\mathbf{G}^T$  will represent a mix of observations and possibly constraints.

For these cases of mixed observations and constraints, the augmented form is particularly useful. The augmented form is able to implicitly encode constraints without having to numerically identify them among other observations. A positive-semi-definite  $\mathbf{R}$ , containing a mix of observations and constraints, can be analysed by an eigenvalue decomposition to identify the constraint directions. The following replacement can be used to separate the components of the observation according to eigenvalues of  $\mathbf{R}$ . The  $\mathbf{R}$  is replaced by the diagonal  $\mathbf{D}$ , thus allowing the zero eigenvalues to be identified explicitly for separate treatment:

$$\text{Eigenvalue decomposition of } \mathbf{R} \quad \mathbf{R} = \mathbf{V}\mathbf{D}\mathbf{V}^T \quad (3.30)$$

$$\text{Replace } \mathbf{R} \quad \mathbf{R} \rightarrow \mathbf{D} \quad (3.31)$$

$$\text{Replace } \mathbf{H} \quad \mathbf{H} \rightarrow \mathbf{V}^T\mathbf{H} \quad (3.32)$$

However, this method is *not preferred*<sup>2</sup> since it is desirable to augment the observations even when  $\mathbf{R}$  has small eigenvalues, not only when  $\mathbf{R}$  has zero eigenvalues. It is also preferable to augment the observations & constraints because then the full structure is retained and uniform treatment is given to all terms. Furthermore, the selective elimination of certain terms ahead of others is a core operation in the direct solving method (see chapter 5) and it is preferable to only define and apply these operations once, consistently for all terms.

Observations can be converted to unit weight by dividing  $\mathbf{z}$  and  $\mathbf{H}$  by  $\sqrt{\mathbf{R}}$ , which is

---

<sup>2</sup>Eigenvalue decomposition of  $\mathbf{R}$  is *not preferred*, augmenting  $\mathbf{R}$  as-is is preferred.

sometimes used to express observations in a uniform manner without the  $\mathbf{R}$  parameters (for example: [20]). However, this cannot be performed for constraints and is numerically ill-advisable for tight observations.

### 3.2.5 Equivalence to the Information Form: Eliminating Observations

This section shows how the augmented system form relates back to the conventional information form for estimation problems. This shows that the augmented form is an extension to the information form and is reducible to the information form. This section considers the relation of the augmented-observation form to the information form by considering the elimination of the observations.

If the observation elements of the augmented form matrix are marginalised out, the resulting observation information term (Schur complement  $\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}$ ) is added onto the state elements ( $\mathbf{Y}$ ). The result is that the effect of the observations is expressed in the states only, in a form added onto the prior information.

The elimination of the observations requires the use of  $\mathbf{R}^{-1}$ . In the case of constraints (singular  $\mathbf{R}$ ) it is not possible to form  $\mathbf{R}^{-1}$ . For this reason, the information form cannot represent constraints and the more general augmented system form described in this chapter must be used.

Equation 3.17, marginalised into only the state variables,  $\mathbf{x}$ , results in the equation for the Newton step in  $\mathbf{x}$  for the original problem in Equation 3.2:

$$\begin{aligned} \begin{pmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{pmatrix} \begin{pmatrix} \Delta \boldsymbol{\nu} \\ \Delta \mathbf{x} \end{pmatrix} &= - \begin{pmatrix} \mathbf{R} \boldsymbol{\nu}_0 + (\mathbf{h}(\mathbf{x}_0) - \mathbf{z}) \\ \mathbf{H}^T \boldsymbol{\nu}_0 - \mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p) \end{pmatrix} \\ (-\mathbf{Y} - \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}) \Delta \mathbf{x} &= -(\mathbf{H}^T \boldsymbol{\nu}_0 - \mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p)) + \mathbf{H}^T \mathbf{R}^{-1} (\mathbf{R} \boldsymbol{\nu}_0 + (\mathbf{h}(\mathbf{x}_0) - \mathbf{z})) \\ (\mathbf{Y} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}) \Delta \mathbf{x} &= \mathbf{H}^T \boldsymbol{\nu}_0 - \mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p) - (\mathbf{H}^T \boldsymbol{\nu}_0 + \mathbf{H}^T \mathbf{R}^{-1} (\mathbf{h}(\mathbf{x}_0) - \mathbf{z})) \end{aligned}$$

$$\boxed{\left(\mathbf{Y} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}\right) \left(\Delta \mathbf{x}\right) = -\mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p) - \mathbf{H}^T \mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}_0) - \mathbf{z})} \quad (3.33)$$

or in non-incremental form <sup>3</sup>:

$$\left(\mathbf{Y} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}\right) \mathbf{x} = \mathbf{Y} \mathbf{x}_p - \mathbf{H}^T \mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}_0) - \mathbf{z} - \mathbf{H} \mathbf{x}_0) \quad (3.34)$$

The observation-update cycle of the information form is obtained by noting that the observation terms play the same role as the information prior terms, and hence can be accumulated into the information prior terms to represent the posterior information terms as shown in equation 3.35.

$$\mathbf{Y}^+ = \mathbf{Y} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} \quad (3.35a)$$

$$\mathbf{Y}^+ \mathbf{x}_p^+ = \mathbf{Y} \mathbf{x}_p + \mathbf{H}^T \mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}_0) - \mathbf{z} - \mathbf{H} \mathbf{x}_0) \quad (3.35b)$$

Equation 3.33 corresponds to the conventional information form, consisting of only the state variables. This is significant because it shows that the conventional information form is a marginalised reduction of variables from the joint state and observation (augmented) form down to only the state variables. In the information form, the observations appear directly as Hessian and gradient terms on the state  $\mathbf{x}$ .

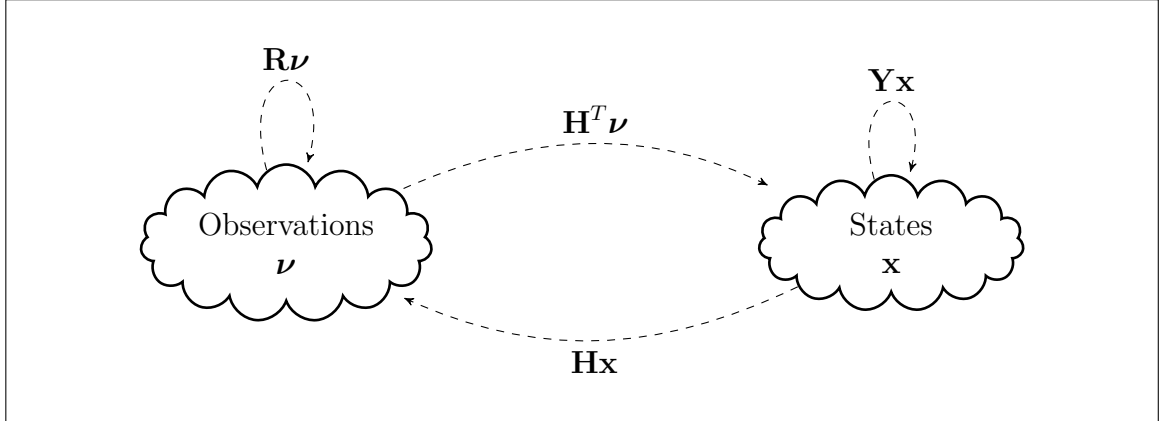
The difference between the augmented form and the information form is illustrated in figure 3.3.

The information form is additive, because the augmented-observation-form is augmentative. Extra observations augmented become extra terms *added* onto the information matrix when marginalised.

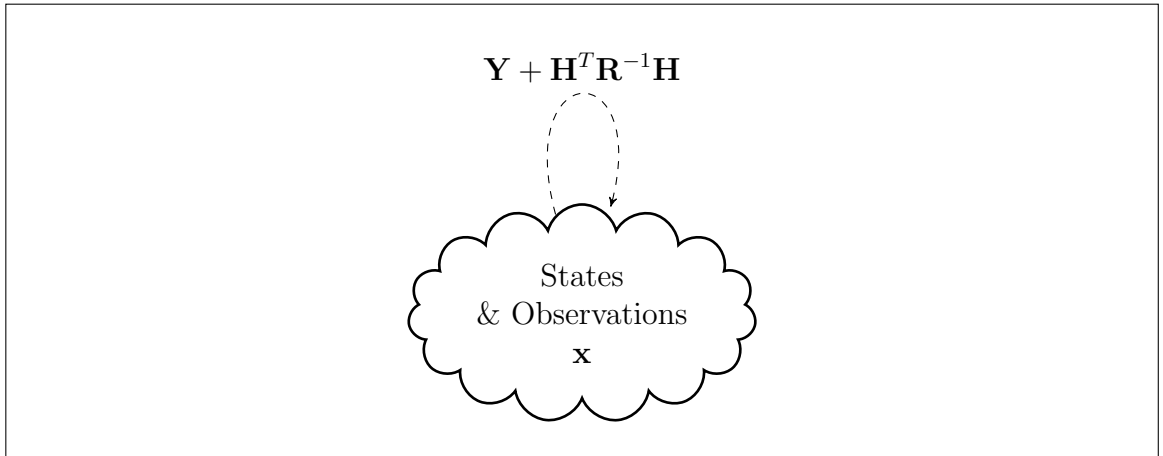
---

<sup>3</sup> The observation term in the right-hand-side of equation 3.34 comes from keeping  $\mathbf{h}(\mathbf{x}) - \mathbf{z}$  together as an irreducible expression for the residual of the observation, and adding in  $\mathbf{H} \mathbf{x}_0$  when converting from  $\Delta \mathbf{x}$  to  $\mathbf{x}$  on the left-hand-side.





(a) In the augmented system form, both the observation (constraint) and state variables are represented jointly. Gradient ( $\mathbf{H}^T \boldsymbol{\nu}$ ) and state projections ( $\mathbf{H}\mathbf{x}$ ) reflect between the observations and states.



(b) In the information form, the states are represented and observations are marginalised on top of the states as posterior terms. The observation terms,  $\mathbf{R}^{-1}$ , are projected back onto the states during marginalisation to form  $\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}$ , which is then permanently added onto the prior terms ( $\mathbf{Y}$ ).

**Figure 3.3:** Schematic illustration of the differences between the augmented system form and the information form regarding the representation of states and observations.

The elimination of the observations is equivalent to a direct solving approach.

Equation 3.33 shows the *marginalisation* of  $\mathbf{A}$ , eliminating the observations. The following shows the *factorisation* of  $\mathbf{A}$  for eliminating the observations out of the augmented system form.

$$\mathbf{A} = \begin{bmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{bmatrix} \quad (3.36)$$

$$\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T \quad (3.37)$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{H}^T\mathbf{R}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & -(\mathbf{Y} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}) \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{R}^{-1}\mathbf{H} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \quad (3.38)$$

The following are therefore mathematically equivalent:

- Forming the information form from the observations and prior terms,
- Eliminating (factorising or marginalising) the observations, and
- Direct solving the augmented system form (starting with eliminating the observations).

This section has shown that the augmented form is an extension to the information form and is reducible to the information form. This section has also shown that the formation of the information form is equivalent to direct solving. The consequences of this for the solution of estimation problems is further developed in section 3.7. Direct solving is further developed in chapter 5.

### 3.2.6 Literature - Augmented System Form

This section provides references to the literature on the augmented system form, used as prior work to the developments presented in this thesis.

The specific mathematical form of the augmented system described in this thesis derives from the systems described in references [10, 12, 37, 64]. None of these

specifically discuss its graph-theoretic connections or application to localisation and mapping.

- Bjork [10] uses the augmented system in a non-weighted, zero prior-information least squares context with the form shown in equation 3.39.

$$\mathbf{A} = \begin{pmatrix} \mathbf{I} & \mathbf{H} \\ \mathbf{H}^T & \mathbf{0} \end{pmatrix} \quad (3.39)$$

- An early example of the use of the augmented system form as an alternative to the “normal equations” (information form) is in Siegel [64]. Siegel essentially describes the advantages of the augmented system form as being a better starting point than the information form for the solution of least squares problems. Siegel describes the advantage of being able to write down the formulation of the problem without requiring additional calculations, followed by a full solving procedure. These properties and approaches are also exploited in this thesis.

Siegel also uses the augmented system form of equation 3.39 (i.e.: excluding the extensions for constraints and prior information).

- In linear algebra the augmented system form is known as an “equilibrium system” or “saddle point form” [37, pg 170] & [74]. Gansterer et al [29] survey a range of equilibrium system properties and problem domains which use such systems.
- The augmented system form for positive-definite  $\mathbf{R}$  and positive-definite  $\mathbf{Y}$  is known as *symmetric quasi-definite* (SQD) and special methods exist to take advantage of the SQD property [73], [32]. Positive-definite  $\mathbf{R}$  states that all observations are subject to nonzero observation uncertainty (i.e.: no constraints). Positive-definite  $\mathbf{Y}$  states that all states (all eigenvalues) have nonzero prior information.
- In electrical engineering and related fields, the augmented system form is known as the “sparse tableau” form [74]. In this context the augmented system applies

for simultaneously representing the dual variables voltage (at nodes) & current (in circuit loops).

- In the optimisation literature, the augmented system form is known as a KKT (Karush-Kuhn-Tucker) system where the augmented system is developed in an augmented Lagrangian context for equality constrained optimisation [12, 54].
- A simple code for assembling  $\mathbf{A}$  (with  $\mathbf{R} = \mathbf{I}$  and  $\mathbf{Y} = \mathbf{0}$ ) is available in Matlab as `spaument`.
- The method of solving the whole augmented system (as opposed to first eliminating one set of variables) is known as:
  - ▷ Primal-dual method [12, pg 532]
  - ▷ Augmented Lagrangian method [54]
  - ▷ The term “augmented system” is used in the numerical least squares literature [6].

This thesis generalises beyond equation 3.39 ([10]) to include general  $\mathbf{R}$  and  $\mathbf{Y}$  as shown in equation 3.41. The generalised  $\mathbf{R}$  (compared to  $\mathbf{R} = \mathbf{I}$ ) allows different weights on observations (diagonal  $\mathbf{R}$ ), groups of correlated observations (block diagonal  $\mathbf{R}$ ) and mixed observations and constraints ( $\mathbf{R}$  with some singular diagonal blocks). The generalised  $\mathbf{Y}$  allows varying structures of prior information to be inserted into the problem formulation.

$$\mathbf{A} = \begin{pmatrix} \mathbf{I} & \mathbf{H} \\ \mathbf{H}^T & \mathbf{0} \end{pmatrix} \quad (\text{Bjork [10]}) \quad (3.40)$$

$$\rightarrow \begin{pmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{pmatrix} \quad (\text{This thesis}) \quad (3.41)$$

This thesis makes a contribution beyond the literature described here: by generalising the form to include both observations and constraints with the associated Lagrangian

quadratic system theory; by considering the role of the augmented system form as a representation of the estimation problem in the wider system together with its graph based representation; by integrating it with trajectory state methods in localisation and mapping; and by considering both the sparsity and numerical properties in factorisation.

### 3.2.7 Nonlinear Observations

For a system with nonlinear observations, the function  $F(\mathbf{x})$  in equation 3.2 is not necessarily quadratic. Instead, a quadratic Taylor approximation,  $\hat{F}(\mathbf{x})$  is used.

The Jacobian derivative of  $\hat{F}(\mathbf{x})$  is still given by equation 3.4, with the replacement that  $\mathbf{H}$  is  $\nabla \mathbf{h}(\mathbf{x})$ , the Jacobian of  $\mathbf{h}(\mathbf{x})$  evaluated at  $\mathbf{x}$ .

The correct Hessian for the second order Taylor approximation is given by:

$$\nabla^2 \hat{F}(\mathbf{x}) = \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} + \sum_i t_i \nabla^2 h_i(\mathbf{x}) \quad (3.42)$$

$$\mathbf{t} = \mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}) - \mathbf{z}) \quad (3.43)$$

This contains an additional higher-order Hessian term beyond the Hessian given in equation 3.5.

However, the Hessian in equation 3.5 can be used as an approximation. Equation 3.5 avoids the requirement to compute second derivative Hessian matrices of the components of the observation function  $\mathbf{h}(\mathbf{x})$ . The use of equation 3.5 is called the Gauss-Newton method. For further details, refer to [12] and [48].

In this thesis, the Gauss-Newton Hessian of equation 3.5 will be used.

It is difficult to fit the full Taylor/Newton Hessian into the augmented system form of this thesis. This is because the additional higher-order Hessian term from the observations ( $\sum_i t_i \nabla^2 h_i(\mathbf{x})$ ) is obtained directly as a system in the state variables. By comparison, the term  $\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}$  is obtained first as  $\mathbf{H}$  and then reduced down into the states only.

The full extra Hessian term could be added as follows, however the extra Hessian terms of each observation are added onto each other:

$$\mathbf{A} = \begin{pmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} - \sum_i t_i \nabla^2 h_i(\mathbf{x}) \end{pmatrix} \quad (3.44)$$

Alternatively, another layer of augmentation can separate out the extra Hessian terms:

$$\mathbf{A} = \begin{pmatrix} \mathbf{R} & & \mathbf{H} \\ & \mathbf{T} & \mathbf{J} \\ \mathbf{H}^T & \mathbf{J}^T & -\mathbf{Y} \end{pmatrix} \quad (3.45)$$

Where  $\mathbf{T}$  is  $\sum_i t_i \nabla^2 h_i(\mathbf{x})$ , for  $i$  in the same block pattern as  $\mathbf{R}$ . This makes the extra Hessian terms spread out in a block diagonal fashion such that the Jacobian and Hessian of an individual observation can be modified independently of the other observations.  $\mathbf{J}$  is sparse and contains ones. However, these approaches are not pursued in this thesis.

The Gauss-Newton approximation is justified for *small residual* problems, since the deviation of the approximation is scaled by  $\mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}) - \mathbf{z})$  of  $\mathbf{x}$  at the solution. In general, one should evaluate the magnitude of the deviation in the particular environment of the application. If invalid, this approximation slows the nonlinear convergence rather than affecting the obtained solution estimate. The Gauss-Newton approximation is built into the information filtering and Kalman filtering approaches which precede this thesis.

### 3.2.8 Properties of the Augmented System Form

In general the augmented system matrix,  $\mathbf{A}$ , can have positive, zero and negative eigenvalues. This makes  $\mathbf{A}$  generally *semi-indefinite*. The solving of indefinite systems is considered in chapter 5. By contrast, the information matrix  $\mathbf{Y}$  and covariance matrix  $\mathbf{P}$  are either *positive definite* or possibly *positive-semi-definite* in degenerate

cases.

The augmented system form is indefinite since the observation and state terms are entered with opposite signs ( $\mathbf{R}$  versus  $-\mathbf{Y}$ ). These opposing signs are required so that the observation information  $\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}$  adds onto the prior information  $\mathbf{Y}$  when the observations are marginalised out. This difference in signs causes the overall  $\mathbf{A}$  augmented form matrix to be indefinite.

Systems for which the *information form* would have zero-valued eigenvalues also have zero-valued eigenvalues in the augmented system form. These indicate lack of observability in the estimates of the states. Furthermore, the augmented system form can also have zero eigenvalues as a result of conflicting or duplicated constraints.

### Example 3.2.

#### *The case of zero eigenvalues due to over-defined constraints*

For example, a system with two identical constraints is shown below. The resulting  $\mathbf{A}$  has a zero eigenvalue relating to the difference between the two Lagrange multipliers of the two (duplicated) constraints.

$$\mathbf{R} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad (3.46)$$

$$\mathbf{H} = \begin{pmatrix} 1 & 1 \end{pmatrix}^T \quad (3.47)$$

$$\mathbf{Y} = 0 \quad (3.48)$$

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \quad (3.49)$$

---

The solution of systems with zero eigenvalues can be approached by introducing regularisation as described in the next section.

### 3.2.9 Regularisation of the Augmented System Form

For a system,  $\mathbf{A} \Delta \mathbf{x} = \mathbf{b}$ , regularisation is the addition of a small multiple of the identity,  $\lambda \mathbf{I}$  onto the left-hand-side system, resulting in  $(\mathbf{A} + \lambda \mathbf{I}) \Delta \mathbf{x} = \mathbf{b}$ . Regularisation is discussed in [48, 54] in the context of damped (regularised) Newton optimisation methods.

In the positive definite case, regularisation puts a minimum bound on the smallest eigenvalue (since any direction with small or zero eigenvalue will have  $\lambda$  added on). As a result, a positive-*semi* definite system can be *regularised* into a positive definite system and hence solved using a positive-definite solution algorithm.

Regularisation of  $\mathbf{A} \Delta \mathbf{x} = \mathbf{b}$  into  $(\mathbf{A} + \lambda \mathbf{I}) \Delta \mathbf{x} = \mathbf{b}$  causes some attenuation of the resulting direction  $\Delta \mathbf{x}$  towards zero, and brings it closer to the steepest-descent direction [48, 54].

In this thesis, the solutions are obtained in the incremental or step-based form as discussed in section 2.4.1. In the step-based form, no permanent bias is introduced into the final solution since the regularisation does not affect the computation of the right-hand-side. The right-hand-side (relating to the gradient of an objective function) must still be zero at the solution. Instead only the steps towards the solution are moderately attenuated.

The remainder of this section contributes the method for the regularisation of the augmented system form. In the positive definite case above, regularisation changes  $\mathbf{A} \Delta \mathbf{x} = \mathbf{b}$  into  $(\mathbf{A} + \lambda \mathbf{I}) \Delta \mathbf{x} = \mathbf{b}$ . However, the augmented system form presented in this chapter is *indefinite*. This indefinite property arises from the dual effects of the observations and states acting in opposing signs. Therefore in order to regularise the augmented system form, it is necessary to use the appropriate opposing signs in the regularisation:

The regularised augmented system form is therefore:

$$\begin{pmatrix} \mathbf{R} + \lambda \mathbf{I} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} - \lambda \mathbf{I} \end{pmatrix} \begin{pmatrix} \Delta \boldsymbol{\nu} \\ \Delta \mathbf{x} \end{pmatrix} = - \begin{pmatrix} \mathbf{R} \boldsymbol{\nu}_0 + (\mathbf{h}(\mathbf{x}_0) - \mathbf{z}) \\ \mathbf{H}^T \boldsymbol{\nu}_0 - \mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p) \end{pmatrix} \quad (3.50)$$



Equation 3.50 has the same right-hand-side as the non-regularised augmented system form in equation 3.17 this means that the regularisation does not permanently weaken constraints or move the solution.

**Example 3.3.** 

---

***The regularisation of over-defined constraints***

Referring to the example 3.2, the system  $\mathbf{A}$  in equation 3.49 can be regularised into:

$$\mathbf{A}_{reg} = \begin{pmatrix} +\lambda & 0 & 1 \\ 0 & +\lambda & 1 \\ 1 & 1 & -\lambda \end{pmatrix} \quad (3.51)$$

$\mathbf{A}_{reg}$  is now symmetric quasi-definite (SQD) with positive-definite  $\mathbf{R} + \lambda\mathbf{I}$  and negative-definite  $-\mathbf{Y} - \lambda\mathbf{I}$ .

---

- The regularisation of the augmented system form acts as a *prior term*, which tends the solution and Lagrange multipliers towards *staying at their current values*.
- In this indefinite case arising from the augmented system form, regularisation puts a minimum bound on the smallest *absolute value* of the eigenvalues. Both the positive and negative eigenvalues are pushed away from zero.

### Regularisation and Constraints

- If a system with  $\mathbf{R} = 0$  constraints, the regularisation adds onto  $\mathbf{R}$  in the right-hand-side system. This effectively *relaxes* the constraint. Fortunately, when using the step-based approach (see section 2.4.1) this only relaxes the constraint for a particular *step* and does not introduce a permanent relaxation of the constraint. The regularisation also does not affect the right-hand-side. (By comparison, re-writing the constraint as a tight observation affects the right-hand-side term and therefore *does* have a permanent relaxation effect on the constraint).

- This regularisation of constraints is useful when the constraints are in conflict with each other, or *over constrained* because the regularisation resolves the zero eigenvalue associated with the over-constraint, allowing the system to be solved.

4

### 3.3 Augmenting Trajectory States

Trajectory state augmentation is an approach to formulating and solving estimation problems consisting of dynamic states. The trajectory state method formulates the problem as a large sparse network consisting of the sequence of dynamic states linked together by observation or dynamic model links. Having formulated this sparse trajectory state model, the system then solves this model using sparse solvers.

By contrast, the alternative *filtering* approach binds the dynamic state aspects into the solving method such that the two form a *prediction step* for moving the dynamic state forward at each time step. Both the trajectory state and filtering approaches involve the use of *process* or *dynamic* models to describe the relationship between the dynamic state at successive time steps. Trajectory state models were introduced in section 2.1 due to their importance in the localisation and mapping literature.

The trajectory state approach is complementary to the augmented system form of this chapter. Both have similar goals and challenges. Both the trajectory state approach and observation augmentation approach aim to present a full structured, accessible representation of the problem (the formulation) by augmenting additional variables, followed by suitable solving methods operating on that structure. Both the trajectory state approach and observation augmentation approach have the challenge of dealing with the increased dimensionality of the system. However, in both cases they improve the detail and sparsity of the representation of the problem structure and allow a wider range of possible *elimination orderings* in the extra augmented variables, to aid the solving process.

---

<sup>4</sup>Systems with conflicting constraints are fundamentally inconsistent; Regularisation re-interprets such conflicting constraints, relaxing them into small  $\mathbf{R}$  observations

The trajectory state approach complements the observation augmented approach in another important manner. The observation augmentation approach operates by maintaining the existence of distinct observation variables separately from state variables, thereby aiding the formulation of nonlinear observations and allowing original nonlinear observation terms to be retained. In a dynamic context, the trajectory state augmentation enables the observation augmentation by keeping the actual past states which the observations need to refer to. The augmented system form of this chapter requires the existence of trajectory states, such that the observations have the appropriate states to link to.

( By contrast, in both the *filtering* approach (marginalisation of past states) or the *information form* approach (marginalisation of observations), the marginalisation of either the observations or the states makes it difficult (or impossible) to perform key operations such as altering the observation structure, re-linearising observations and obtaining observation residuals ).

This thesis describes the trajectory state approach because it is a key foundation of the estimation formulation and solving approach presented in this thesis. The contributions of this thesis, the augmented system method, the graph structure representation and the graph solving algorithm are all designed in context of the trajectory state approach and these contributions offer extensions to the methods presently available.

### 3.3.1 Formation of the Trajectory States

The trajectory state method will be explained in the following analytical introduction. The trajectory state method will be explained by referring to a single pair of states relating to successive time steps of a dynamic state. The trajectory state method forms expressions jointly over the two timestep states.

1. Consider the joint state vector of a state  $\mathbf{x}$  at times  $k$  and  $k + 1$ . The state  $\mathbf{x}$  at time  $k$  is assumed to have prior information matrix  $\mathbf{Y}$  and prior estimate  $\mathbf{x}_p$ . The state at time  $k + 1$  has no prior information, since it is the state in the

future and the only information is obtained via the process model linking time  $k + 1$  to time  $k$ .

2. Consider a discrete-time, linear, dynamic model between the two successive  $\mathbf{x}$  states:

$$\text{Linear dynamic model} \quad \mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{B}\mathbf{u} + \mathbf{G}\mathbf{v} \quad (3.52a)$$

$$\text{Rearranging terms} \quad \mathbf{B}\mathbf{u} = \mathbf{I}\mathbf{x}_{k+1} - \mathbf{F}\mathbf{x}_k - \mathbf{G}\mathbf{v} \quad (3.52b)$$

$$\mathbf{B}\mathbf{u} = \begin{bmatrix} \mathbf{I} & -\mathbf{F} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{k+1} & \mathbf{x}_k \end{bmatrix}^T + \mathbf{G}\mathbf{v} \quad (3.52c)$$

- $\mathbf{F}$  is a linear model which maps the successive states deterministically.
  - $\mathbf{G}$  is a linear model which maps in the noise  $\mathbf{v}$ .
  - $\mathbf{v}$  is zero mean noise with covariance  $E[\mathbf{v}\mathbf{v}^T] = \mathbf{Q}$ .
  - $\mathbf{B}$  is a linear model mapping in the control  $\mathbf{u}$ .
  - Equation (3.52a) expresses the dynamic model in the standard form, expressing the later state as a function of the previous state, inputs and noises.
  - Equation (3.52c) expresses the dynamic model as an observation operator on the joint state of the two successive states.
3. Identifying components of equation 3.52c with observation notation (3.53), results in the following:

$$\mathbf{Z} = \mathbf{H}\mathbf{X} + \mathbf{w} \quad \mathbf{R} = E[\mathbf{w}\mathbf{w}^T] \quad (3.53)$$

$$\mathbf{Z} \rightarrow \mathbf{B}\mathbf{u} \quad \mathbf{H} \rightarrow \begin{bmatrix} \mathbf{I} & -\mathbf{F} \end{bmatrix} \quad (3.54a)$$

$$\mathbf{X} \rightarrow \begin{bmatrix} \mathbf{x}_{k+1} & \mathbf{x}_k \end{bmatrix}^T \quad (3.54b)$$

$$\mathbf{w} \rightarrow \mathbf{G}\mathbf{v} \quad \mathbf{R} \rightarrow \mathbf{G}\mathbf{Q}\mathbf{G}^T \quad (3.54c)$$

4. Applying the dynamic model as an observation update, using the replacements from equation (3.54) and the augmented observation form from equation (3.17) results in the following system:

$$\begin{pmatrix} \mathbf{G}\mathbf{Q}\mathbf{G}^T & \mathbf{I} & -\mathbf{F} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \\ -\mathbf{F}^T & \mathbf{0} & -\mathbf{Y} \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{\nu} \\ \Delta\mathbf{x}_{k+1} \\ \Delta\mathbf{x}_k \end{pmatrix} = - \begin{pmatrix} (\mathbf{x}_{k+1} - \mathbf{F}\mathbf{x}_k - \mathbf{B}\mathbf{u}) + \mathbf{G}\mathbf{Q}\mathbf{G}^T\boldsymbol{\nu}_0 \\ \boldsymbol{\nu}_0 \\ -\mathbf{F}^T\boldsymbol{\nu}_0 - \mathbf{Y}(\mathbf{x}_k - \mathbf{x}_p) \end{pmatrix} \quad (3.55)$$

- $\mathbf{x}_k^0$  and  $\mathbf{x}_{k+1}^0$  on the right-hand-side are initial values. The final values are adjusted by the results of  $\Delta\mathbf{x}_k$  and  $\Delta\mathbf{x}_{k+1}$ . Similarly,  $\boldsymbol{\nu}_0$  is an initial value to be adjusted by  $\Delta\boldsymbol{\nu}$ . The reasons for using this incremental approach were discussed in section 2.4.1.
  - For this simple two-step case, the solution gives the obvious values for both  $\mathbf{x}_k$  and  $\mathbf{x}_{k+1}$ , equal to the conventional prediction, and  $\boldsymbol{\nu} = \mathbf{0}$ .
  - The system in equation (3.55) above, shows the pair of states and their linking dynamic model step as a joint system.
5. In general, this process of linking successive pairs of dynamic states is continued indefinitely. The overall effect of the dynamic models is to form a continuous *chained* sequence of states linked together by the dynamic model instances. The estimates are linked together and hence *smoothed* as a result.

### 3.3.2 Discussion

The resulting system, shown in equation 3.55 shows the pair of states formed together jointly, along with an observation variable relating to their dynamic model.

The trajectory state approach unifies observations and predictions, since predictions are written as an observation linking successive dynamic states. Therefore predictions are considered as a subset of observations generally. This thesis will not explicitly refer

to “observations and predictions” unless specifically required to distinguish predictions from other observations.

The trajectory state approach allows the original nonlinear observations to be retained. Retaining the past vehicle states enables the system to maintain the original nonlinear observations, since these link to the past vehicle states. This, in effect, forms a representation for the estimation problem. Rather than attempting to amortise all the past observations into a present state marginal (and correspondingly attempting to find reasonable functional forms which capture the nonlinear shape of the observations), the trajectory state approach allows the system to simply use the original nonlinear observation models as the representation of the log-PDF of the estimation problem. Keeping the actual nonlinear observation terms means that final estimates can be checked by re-projecting the estimates into the observations again and checking the residuals against the actual obtained observations. These aspects of the trajectory state approach lead to the development of the augmented system form described in this chapter.

The trajectory state approach frames the prediction models as a general mesh among the states. The classical method of filtering, with its single-directional predict and observe cycle exploits the *chain-like* structure of the predict-observe cycle. This chain-like nature of the algorithms is appropriate for systems such as inertial-GPS, which genuinely have a chain-like structure of their models. However, for systems in localisation and mapping in which the overall structure of the models is not a chain, but a general mesh, the more general trajectory state approach is more appropriate. Such non-chain topologies of the system are obtained, for example, when static features are linked by observations to the vehicle states at various different times. The overall structure of the system can transition from chain-like into *looped* topologies as a result of new observations. This is known as *loop closure*.

The trajectory state approach is more appropriate for general mesh structured systems because it aims only to describe the structure of the system, not to introduce a particular solving algorithm. (By comparison, the filtering approach *is* a solving algorithm.) For general mesh systems the solving problem is considerably more

difficult than for chain-like systems. Therefore it is worthwhile describing the full structure first and considering the operation of the solution algorithm separately.

In general the dynamic model observation noise ( $\mathbf{G}\mathbf{Q}\mathbf{G}^T$ ) may be singular, indicating a constraint component to the model. In general the exact *direction* of the constraint will vary depending on the particular linearisation. The augmented system form is able to operate with the constraint without decomposing  $\mathbf{G}\mathbf{Q}\mathbf{G}^T$  into constraint and observation components.

Note that mathematically, the  $\mathbf{G}\mathbf{Q}\mathbf{G}^T$  observation uncertainty comes from the *marginalisation* of hidden states representing the instantaneous value of noise or control inputs into the dynamic model. In theory these could be augmented too (which would extend the augmented system form to cover states, observations, noises and controls). This might be useful for highly nonlinear dynamic systems. Also, the structure of noises and controls usually link with small degree into specific observations. This means that there is no sparsity reason to augment noises and controls. Any noise or control parameter which links into a larger range of observations, or is known to be of interest should be treated as a parameter state and accordingly augmented and estimated.

### 3.3.3 Equivalence

This section will show the equivalence between a single timestep recursive formulation and the multi-timestep trajectory state formulation.

The recursive formulation is obtained by marginalising out the first timestep, resulting in expressions only in the second timestep.

1. This step considers the reduction obtained by eliminating the  $\boldsymbol{\nu}$  variable and the resulting joint system in the pair of states. Eliminating the variable  $\boldsymbol{\nu}$  from

equation 3.55 results in the following system:

$$\tilde{\mathbf{A}} \begin{pmatrix} \Delta \mathbf{x}_{k+1} \\ \Delta \mathbf{x}_k \end{pmatrix} = \tilde{\mathbf{b}} \quad (3.56)$$

Where:

$$\tilde{\mathbf{A}} = \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} + \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{Y} \end{pmatrix} \quad (3.57)$$

$$= \begin{pmatrix} \mathbf{F}^T \mathbf{R}^{-1} \mathbf{F} & -\mathbf{F}^T \mathbf{R}^{-1} \\ -\mathbf{R}^{-1} \mathbf{F} & \mathbf{R}^{-1} + \mathbf{Y} \end{pmatrix} \quad (3.58)$$

$$\tilde{\mathbf{b}} = - \begin{pmatrix} \mathbf{0} \\ \mathbf{Y}(\mathbf{x}_k - \mathbf{x}_p) \end{pmatrix} - \mathbf{H}^T \mathbf{R}^{-1}(\mathbf{x}_{k+1} - \mathbf{F}\mathbf{x}_k - \mathbf{B}\mathbf{u}) \quad (3.59)$$

$$= - \begin{pmatrix} \mathbf{R}^{-1}(\mathbf{x}_{k+1} - \mathbf{F}\mathbf{x}_k - \mathbf{B}\mathbf{u}) \\ \mathbf{Y}(\mathbf{x}_k - \mathbf{x}_p) - \mathbf{F}^T \mathbf{R}^{-1}(\mathbf{x}_{k+1} - \mathbf{F}\mathbf{x}_k - \mathbf{B}\mathbf{u}) \end{pmatrix} \quad (3.60)$$

2. Eliminating the variable  $\boldsymbol{\nu}$  requires an invertible  $\mathbf{G}\mathbf{Q}\mathbf{G}^T$ . This is not assumed in this section in general, only for the purposes of demonstrating this intermediate step in the explanation. In fact, having a singular  $\mathbf{G}\mathbf{Q}\mathbf{G}^T$  is one of the primary applications of constraints: The ability to express deterministic components of dynamic models.

Eliminating the variable  $\boldsymbol{\nu}$  adds an structure  $\mathbf{I}_p = \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}$  on top of the pair of states. This  $\mathbf{I}_p$  expresses the marginalised dynamic model in information form.

$$\mathbf{I}_p = \begin{pmatrix} \mathbf{F}^T \mathbf{R}^{-1} \mathbf{F} & -\mathbf{F}^T \mathbf{R}^{-1} \\ -\mathbf{R}^{-1} \mathbf{F} & \mathbf{R}^{-1} \end{pmatrix} \quad (3.61)$$

All  $\mathbf{I}_p$  terms for dynamic models have the property:

$$\det(\mathbf{I}_p) = 0 \quad (3.62)$$



The reason for this is that a “prediction” model has the property that applying (adding) the prediction model  $\mathbf{I}_p$  to predict forward into an unknown (zero information) future state, followed by marginalisation back to the present only does not alter the present state marginal. In other words there is no gain or loss of information to the present obtained by predicting forward in time into a new unknown future state.  $\det(\mathbf{I}_p) = 0$  is a necessary condition for  $\mathbf{I}_p$  to represent a dynamic model.

3. This step considers the reduction obtained by eliminating both  $\boldsymbol{\nu}$  and  $\mathbf{x}_k$  from equation 3.55, resulting in a posterior system in  $\mathbf{x}_{k+1}$ .

The marginal for the state  $\mathbf{x}_{k+1}$  of equation 3.55 is given by:

$$\tilde{\mathbf{A}} = \mathbf{0} - \begin{pmatrix} \mathbf{I} \\ \mathbf{0} \end{pmatrix}^T \begin{bmatrix} \mathbf{G}\mathbf{Q}\mathbf{G}^T & -\mathbf{F} \\ -\mathbf{F}^T & -\mathbf{Y} \end{bmatrix}^{-1} \begin{pmatrix} \mathbf{I} \\ \mathbf{0} \end{pmatrix} \quad (3.63)$$

This is expanded using a block-matrix inverse formula [59]:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{F}_{11}^{-1} & -\mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{F}_{22}^{-1} \\ -\mathbf{F}_{22}^{-1}\mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{F}_{22}^{-1} \end{bmatrix} \quad (3.64)$$

$$\mathbf{F}_{11} = \mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}_{21} \quad (3.65)$$

$$\mathbf{F}_{22} = \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \quad (3.66)$$

Resulting in the following for  $\tilde{\mathbf{A}}$  and similarly  $\tilde{\mathbf{b}}$ :

$$\tilde{\mathbf{A}} = -(\mathbf{G}\mathbf{Q}\mathbf{G}^T + \mathbf{F}\mathbf{Y}^{-1}\mathbf{F}^T)^{-1} \quad (3.67)$$

$$= -(\mathbf{G}\mathbf{Q}\mathbf{G}^T + \mathbf{F}\mathbf{P}\mathbf{F}^T)^{-1} \quad (3.68)$$

$$\tilde{\mathbf{b}} = -(\mathbf{G}\mathbf{Q}\mathbf{G}^T + \mathbf{F}\mathbf{P}\mathbf{F}^T)^{-1}(\mathbf{F}\mathbf{x}_k + \mathbf{B}\mathbf{u} - \mathbf{x}_{k+1}) \quad (3.69)$$

In the inverse-covariance, incremental form:

$$(\mathbf{G}\mathbf{Q}\mathbf{G}^T + \mathbf{F}\mathbf{P}\mathbf{F}^T)^{-1} \Delta\mathbf{x}_{k+1} = (\mathbf{G}\mathbf{Q}\mathbf{G}^T + \mathbf{F}\mathbf{P}\mathbf{F}^T)^{-1}(\mathbf{F}\mathbf{x}_k + \mathbf{B}\mathbf{u} - \mathbf{x}_{k+1}^0) \quad (3.70)$$

Where  $\mathbf{x}_{k+1} = \mathbf{x}_{k+1}^0 + \Delta\mathbf{x}_{k+1}$ .

Equation 3.70 simplifies to:

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{B}\mathbf{u} \quad (3.71)$$

$$\mathbf{P}_{k+1} = \mathbf{F}\mathbf{P}\mathbf{F}^T + \mathbf{G}\mathbf{Q}\mathbf{G}^T \quad (3.72)$$

These are the conventional recursive formulae for the prediction stage of a Kalman (covariance form) filter. The requirement for invertible  $\mathbf{G}\mathbf{Q}\mathbf{G}^T$  does not apply in this covariance form expression,

The form of equation 3.72 operates *directionally*; It aims to replace the prior  $\mathbf{P}$  with the posterior  $\mathbf{P}_{k+1}$ . By contrast, the trajectory state form of equation 3.55 aims to provide observations and constraints on the joint pair of states at the sequential timesteps, binding them to each other in a symmetrical fashion. In other words, the aim of the trajectory state form is not to *replace* one prior by a posterior but instead to link them together using the dynamic model information.

### 3.4 Relation To Graphical Models

The augmented system form is closely related to methods in the graphical models literature. The augmented system form is a *formulation approach*, as are graphical models. Both are intended to capture sparsity and decoupling properties of the formulation and permit a variety of *solving* methods subject to the initial formulation. Section 3.4.1 discusses the relation of the augmented system form to factor graphs, section 3.4.2 discusses the form of the augmented system form *before* the system is *conditioned* on the obtained observations. See also section 5.6.1 in relation to junction-tree algorithms.

### 3.4.1 Relation to Factor Graphs

The augmented system form relates closely to the *factor graph* theory described in [20, 44]. The factor graph represents the set of state variables of the estimation problem formulation, together with the set of observation nodes. Each observation node links to various states and defines a *potential* function over those states. The total probability density function is proportional to the product of those potentials. Hence the observation potentials are referred to as factors, and the overall graph as the factor graph. The factor graph representation applies equally well for any Bayesian representation of states (for example, continuous, discrete states) and factors. The factor graph is bipartite, meaning that states never link directly to other states and factors never link directly to other factors (this follows naturally from the definition of the factors as the models which affect any related states). For Gaussian systems, the edges in a factor graph provide a natural representation for the sparse measurement Jacobian,  $\mathbf{H}$ .

In the discussion in [20] the measurement Jacobian,  $\mathbf{H}$ , is described as equivalent to the *factor graph*, and the information matrix is described as equivalent to the undirected Markov random field. The Markov random field is obtained by elimination of the factor nodes. The *Markov random field* representation consists of undirected links between state variables. The state variables, which were previously linked to a factor in the factor graph, form a *clique* in the Markov graph as a result of eliminating the factor. The Markov graph is undirected and operates over the state variables. The Gaussian equivalent of the Markov graph is the information form, which is correspondingly also symmetric and operates over the state variables. The information matrix represents an undirected representation of the total interactions within and between the state variables, resulting in a fundamentally *square* matrix. The observation Jacobian matrix, by contrast, is a fundamentally *rectangular* matrix associated with the fundamentally *bipartite* links between observations and states.

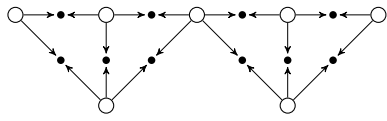
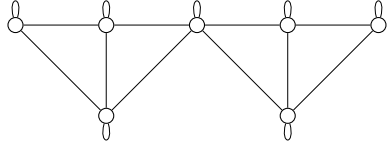
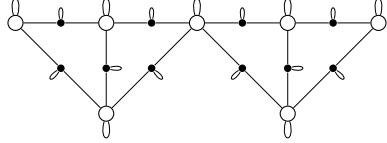
In order to define the Markov graph as a result of eliminating (marginalising) the observation factors or nodes, it is necessary to define such an augmented system containing the state and observation nodes. Informally, it is possible to state that

this elimination is obtained by transforming  $\mathbf{H}$  and  $\mathbf{R}$  alone into  $\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}$ . However such a transformation (bipartite rectangular  $\rightarrow$  symmetric) is not as well defined as marginalisation in symmetric systems. Therefore it is necessary to consider some symmetric system containing the states, some variable in the size of the observations, linked together by  $\mathbf{H}$  in the off-diagonals between them.

Such a system is exactly the augmented system form presented in this chapter. Therefore this thesis proposes the following categorisation of the equivalences, in figure 3.4. The factor graph [20] can be considered equivalent to the measurement Jacobian,  $\mathbf{H}$  if the factor graph is taken to be *directed bipartite*. The Markov network (or information form) is equivalent to a symmetric set of edges between the states. Then the augmented system form is equivalent to the graph of figure 3.4 (c), containing the observation Lagrange multipliers and states.

For Gaussian random variables, the factor graph effectively encodes a representation of the quadratic in equation 3.2 with each block diagonal term in  $\mathbf{R}$  and  $\mathbf{Y}$  being a separate factor.

By comparison, the augmented system form uses the same state nodes and the same number of observation nodes as the factor graph, but the observation nodes for the augmented system form contain the observation Lagrange multiplier variables. Then the augmented system form, as a graphical model, encodes the Lagrangian of equation 3.10.

Graph concept:	Category:	Linear system:
(a) 	directed bipartite	Measurement Jacobian $\mathbf{H}$ (observations $\times$ states)
(b) 	symmetric	Information form $\mathbf{Y} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}$ (states $\times$ states)
(c) 	symmetric	Augmented System $\mathbf{A} = \begin{pmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{pmatrix}$ ( [obs,states] $\times$ [obs,states] )

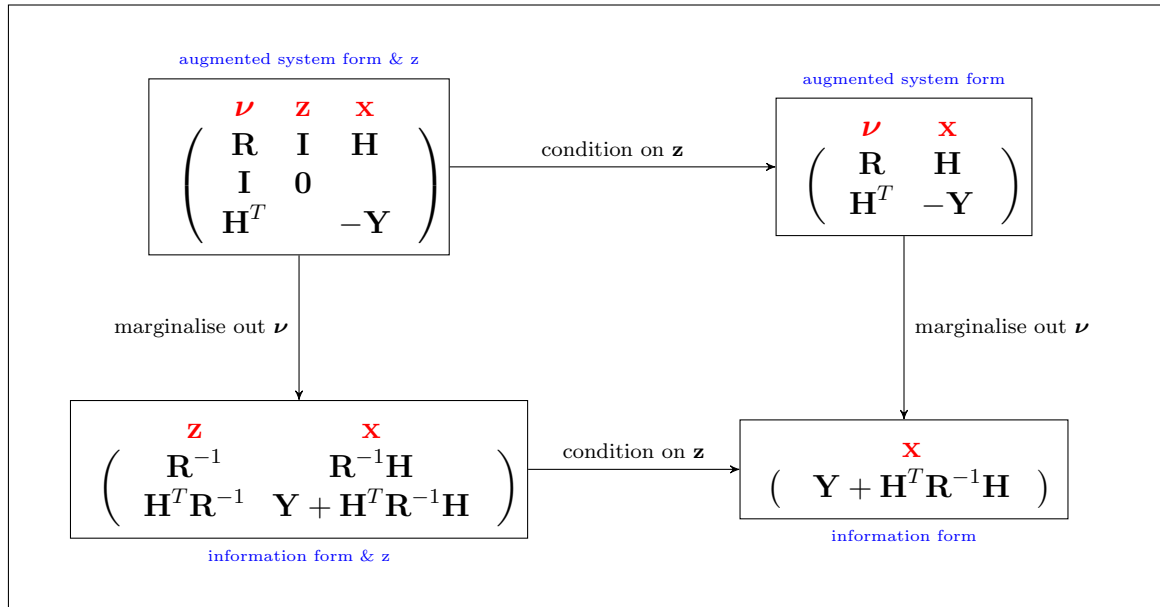
**Figure 3.4:** A set of equivalences between graph concepts and linear systems in estimation. The *directed* bipartite graph containing the observations and states is effectively equivalent to the measurement Jacobian,  $\mathbf{H}$ . The symmetric Markov random field graph is equivalent to the information form. A symmetric graph containing the observations and states is equivalent to the augmented system form.

In conclusion, the augmented system form provides the same graph structure and capabilities as the factor graph. The augmented system form provides a mathematically consistent model which justifies the extension of the dimension of the variables to allow the representation of the observation nodes. Such extended, observation-sized variables turn out to be the observation Lagrange multipliers.

### 3.4.2 What are the systems *before* conditioning on the observations?

The information form and augmented system form described above both assume that observations occur with a known observation  $\mathbf{z}$ .

If the system is modelled with an unknown  $\mathbf{z}$ , then these effectively become further (unknown) states. Such a system can be represented in the forms: “augmented system form &  $\mathbf{z}$ ” and “information form &  $\mathbf{z}$ ” shown below. When these are *conditioned* on the obtained, known  $\mathbf{z}$ , the resulting *conditioned* systems are expressed in either the information form or augmented system form shown below. These systems with  $\mathbf{z}$  included would be useful for reasoning about expected values of  $\mathbf{z}$  before they are obtained. However, in most regards, these systems including  $\mathbf{z}$  have similar properties as their counterparts without  $\mathbf{z}$ .



**Figure 3.5:** The augmented system form and information form, together with their counterparts including  $\mathbf{z}$  (before conditioning on  $\mathbf{z}$ ). The variables involved in each system are shown in red above the systems.

Figure 3.5 shows the augmented system form and the information form together with their counterparts containing  $\mathbf{z}$  *before* conditioning on  $\mathbf{z}$ . Four variations are shown containing both, either and neither of  $\mathbf{nu}$  and  $\mathbf{z}$  in addition to the state  $\mathbf{x}$ . The most general system (“augmented system form &  $\mathbf{z}$ ”) contains the state  $\mathbf{x}$ , observations  $\mathbf{z}$  and observation Lagrange multipliers  $\nu$  (top left). Conditioning on  $\mathbf{z}$  leads to the augmented system form ( $\mathbf{x}$  &  $\nu$ ). Marginalising out  $\nu$  leads to the “information form &  $\mathbf{z}$ ”.

### 3.5 Insights for Data Fusion

The augmented system form brings an alternative insight into the nature of data fusion.

The augmented system form shows explicitly how the state estimates are perturbed by gradient effects which flow through from the observations, via the  $\boldsymbol{\nu}$  Lagrange multiplier and via the state-to-observation conversion operator  $\mathbf{H}$ .

This forms a Lagrange-multiplier vector interpretation of data fusion, in which individual terms (observations or prior estimates) operate individually but are forced to interact via the sharing of Lagrange-multiplier vectors.

To explain this more specifically, consider an example consisting of two observations and a prior term for a single one dimensional state.

This explanation will expand equation 3.17 to show the two observations. The complete observation noise covariance  $\mathbf{R}$  will consist of the noise covariance matrices for each of the two observations acting independently and the complete observation Jacobian  $\mathbf{H}$  will consist of the entries from each observation:

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_a & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_b \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} \mathbf{H}_a \\ \mathbf{H}_b \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} \mathbf{z}_a \\ \mathbf{z}_b \end{bmatrix} \quad (3.73)$$

From equation (3.17):

$$\begin{pmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{pmatrix} \begin{pmatrix} \Delta \boldsymbol{\nu} \\ \Delta \mathbf{x} \end{pmatrix} = - \begin{pmatrix} (\mathbf{H}\mathbf{x}_0 - \mathbf{z}) + \mathbf{R}\boldsymbol{\nu}_0 \\ \mathbf{H}^T \boldsymbol{\nu}_0 - \mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p) \end{pmatrix} \quad (3.74)$$

$$\begin{pmatrix} \mathbf{R}_a & \mathbf{0} & \mathbf{H}_a \\ \mathbf{0} & \mathbf{R}_b & \mathbf{H}_b \\ \mathbf{H}_a^T & \mathbf{H}_b^T & -\mathbf{Y} \end{pmatrix} \begin{pmatrix} \Delta \boldsymbol{\nu}_a \\ \Delta \boldsymbol{\nu}_b \\ \Delta \mathbf{x} \end{pmatrix} = - \begin{pmatrix} (\mathbf{H}_a \mathbf{x}_0 - \mathbf{z}_a) + \mathbf{R}_a \boldsymbol{\nu}_{a0} \\ (\mathbf{H}_b \mathbf{x}_0 - \mathbf{z}_b) + \mathbf{R}_b \boldsymbol{\nu}_{b0} \\ -\mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p) + \mathbf{H}_a^T \boldsymbol{\nu}_{a0} + \mathbf{H}_b^T \boldsymbol{\nu}_{b0} \end{pmatrix} \quad (3.75)$$

The right hand side of equation (3.75) consists of terms for each observation and for the state.

- For the observation terms, there is a *residual* expression (for example  $(\mathbf{H}_a \mathbf{x}_0 - \mathbf{z}_a)$ ), which expresses the residual for the observation on its own plus an expression which brings in the effect of the rest of the system via the Lagrange multiplier  $\boldsymbol{\nu}$  (for example:  $\mathbf{R}_a \boldsymbol{\nu}$ ). For each observation, the “rest of the system” is only the state. Each observation links only to the common state, rather than to each other directly.

Each observation on its own would bring the estimate to the *maximum-likelihood* value, for example an  $\mathbf{x}$  such that  $(\mathbf{H}_a \mathbf{x}_0 - \mathbf{z}_a) = \mathbf{0}$ . However, the posterior estimate arrives at a value *different* from the maximum-likelihood of the observation because *other* terms (the prior and other observation) pull the estimate there via the Lagrange multiplier  $\boldsymbol{\nu}$ .

- For the state term, there is a *residual* expression  $\mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p)$  which expresses the residual for the prior information on its own plus the expression  $\mathbf{H}_a^T \boldsymbol{\nu}_{a0} + \mathbf{H}_b^T \boldsymbol{\nu}_{b0}$  which brings in the effect of the rest of the system via the Lagrange multipliers  $\boldsymbol{\nu}$ . For the state, the “rest of the system” is the two observations.

The prior term on its own would leave the estimate at  $\mathbf{x}_p$ . However, the posterior estimate arrives at a different value because the combined effect of the sum of the observation Lagrange multipliers pulls the estimate there.

The observation augmented form shows an alternative insight into the nature of data fusion by showing separately the observation and state terms and showing their interaction via Lagrange multipliers. This is in contrast to conventional approaches in estimation. In conventional approaches the interaction is always expressed within the states, either in an information or covariance matrix in the states, or any other form of probabilistic model. These have the property that the observations are summarised onto the states. In conventional approaches, observations and prior information interact with each other in terms of the states, resulting in a *posterior* model.

The observation augmented form therefore shows an alternative approach, by maintaining the observations and states separately and showing their interaction via Lagrange multipliers.



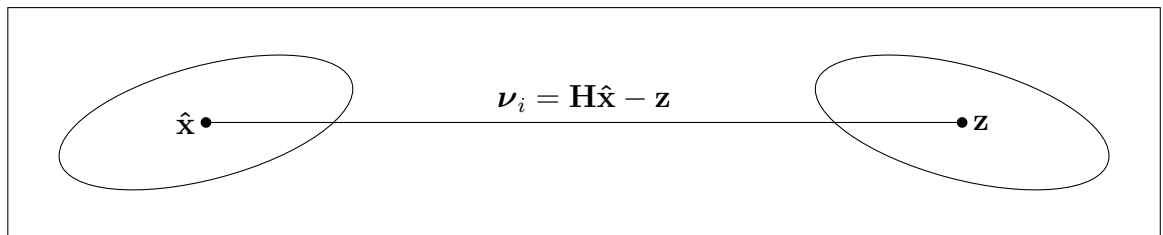
In this manner, the interface between an observation and its related states is not a high-fidelity functional representation of the observation but instead simply a *vector*. This Lagrange multiplier vector,  $\boldsymbol{\nu}$ , is defined in the observation space but translates into the state space via  $\mathbf{H}^T \boldsymbol{\nu}$ . The Lagrange multiplier vector is not *static* but instead responds *dynamically* to proposed changes in the state estimate. (The augmented system form orchestrates this simultaneous manipulation of the states and Lagrange multipliers).

This insight could have a number of future applications:

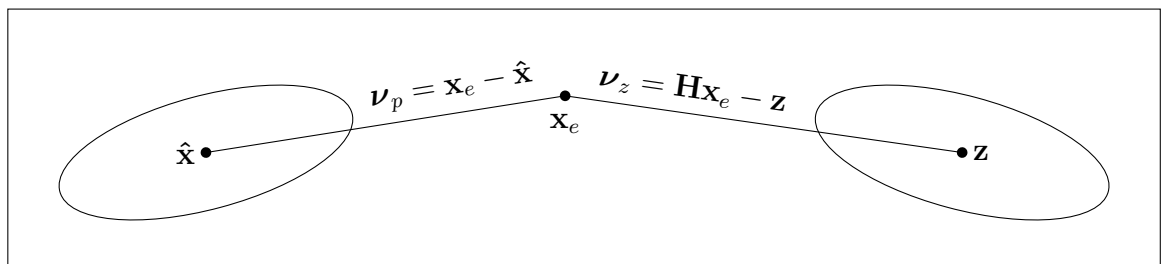
- For nonlinear observations (including their communication in a decentralised fashion) the interface between observations and states need only be the Lagrange multiplier.
- For decentralised systems across multiple platforms, the interface might therefore consist of the Lagrange multiplier vector and state estimate rather than consisting of entire marginal posterior distributions.

### 3.6 Residuals & Innovations

This section proposes a novel form of Mahalanobis distance, the *residual* distance, which is equivalent to the innovation distance, but is more general. The main purpose of the residual distance is to provide a distance measure for measuring consistency under more complex situations. The proposed residual distance suits the other estimation structures proposed in this thesis, particularly for evaluating cases with multiple prediction and observation models, and for evaluating consistency throughout a complex network of states and observations. The residual distance also achieves the goal of generalising the innovation distance for cases with *rank-deficient* prior and/or observation terms. As an introductory illustration, figure 3.6 shows the conventional *innovation* approach, in which a distance measure is obtained from the single innovation ( $\boldsymbol{\nu}_i = \mathbf{H}\hat{\mathbf{x}} - \mathbf{z}$ ), as a single pairwise expression. By comparison the *residual* approach forms residuals for each term in the network, relative to a common state estimate ( $\boldsymbol{\nu}_z = \mathbf{H}\mathbf{x}_e - \mathbf{z}$  and  $\boldsymbol{\nu}_p = \mathbf{x}_e - \hat{\mathbf{x}}$ ). The residual distance is obtained from a combination of these residuals.



(a) The *innovation* approach. A single innovation,  $\boldsymbol{\nu}_i$ , is produced from  $\hat{\mathbf{x}}$  &  $\mathbf{z}$ .



(b) The *residual* approach. A residual is produced for each term, from  $\hat{\mathbf{x}}$  &  $\mathbf{x}_e$  and from  $\mathbf{z}$  &  $\mathbf{x}_e$ .  $\mathbf{x}_e$  is the posterior state estimate.

**Figure 3.6:** Illustration of the innovation and residual terms

This section will develop expressions based on a simple example estimation problem, consisting of the fusion of a single prior information term with a single observation.

The prior information term is described by estimate  $\hat{\mathbf{x}}$  with covariance  $\mathbf{P}$ . The prior term can be related to the underlying state,  $\mathbf{x}$  and error (or noise) in the prior term  $\mathbf{w}_p$  as follows:

$$\hat{\mathbf{x}} = \mathbf{I}\mathbf{x} + \mathbf{w}_p \quad (3.76)$$

$$E[\mathbf{w}_p] = \mathbf{0} \quad E[\mathbf{w}_p \mathbf{w}_p^T] = \mathbf{P} \quad (3.77)$$

The observation term is described by observation result  $\mathbf{z}$ . The observation relates to the underlying state and observation noise  $\mathbf{w}_z$  as follows:

$$\mathbf{z} = \mathbf{H}\mathbf{x} + \mathbf{w}_z \quad (3.78)$$

$$E[\mathbf{w}_z] = \mathbf{0} \quad E[\mathbf{w}_z \mathbf{w}_z^T] = \mathbf{R} \quad (3.79)$$

### 3.6.1 Innovations

The innovation distance is a standard tool for measuring the consistency between a prior (or predicted) information term and an observation term, taking into account the obtained values and their given uncertainties. The *innovation* is defined as the difference between a predicted observation from the latest estimate (in this case simply the prior  $\hat{\mathbf{x}}$ ) and the obtained observation. Thus the innovation is a distance measure in the *observation* space. The variance of the innovation,  $\mathbf{S}$ , is obtained as a result of the linear combination of random variables  $\mathbf{w}_z$  and  $\mathbf{w}_p$ .

$$\boldsymbol{\nu}_i \triangleq \mathbf{H}\hat{\mathbf{x}} - \mathbf{z} \quad (3.80)$$

$$= \mathbf{H}\mathbf{w}_p - \mathbf{w}_z \quad (3.81)$$

$$E[\boldsymbol{\nu}_i] = \mathbf{0} \quad E[\boldsymbol{\nu}_i \boldsymbol{\nu}_i^T] = \mathbf{S} \quad (3.82)$$

$$\mathbf{S} = \mathbf{H}\mathbf{P}\mathbf{H}^T + \mathbf{R} \quad (3.83)$$

The innovation Mahalanobis distance, also known as the normalised innovation squared, is given by:

$$M_i = \frac{1}{2} \boldsymbol{\nu}_i^T \mathbf{S}^{-1} \boldsymbol{\nu}_i \quad (3.84)$$

$$= \frac{1}{2} (\mathbf{h}(\hat{\mathbf{x}}) - \mathbf{z})^T (\mathbf{H} \mathbf{P} \mathbf{H}^T + \mathbf{R})^{-1} (\mathbf{h}(\hat{\mathbf{x}}) - \mathbf{z}) \quad (3.85)$$

The innovation Mahalanobis distance is also used as the basis for the “information gate” or “information Mahalanobis distance” in [52]. The information Mahalanobis distance shares many of the properties of the innovation Mahalanobis distance. In particular, the innovation is obtained pairwise for two terms, and  $\mathbf{Y}$  is required to be invertible. The information Mahalanobis distance is equivalent to the innovation Mahalanobis distance but offered in a slightly modified form suitable for use with some information filtering expressions. Therefore, the information Mahalanobis distance will not be discussed any further.

The next section will discuss the proposed alternative, the residual Mahalanobis distance.

### 3.6.2 Residuals

The motivation behind the residual Mahalanobis distance approach is illustrated in figure 3.6. The residual Mahalanobis distance was motivated by the following question: How does the innovation Mahalanobis distance (of equation 3.85) relate to the following quadratic form, evaluated at the solution  $\mathbf{x} = \mathbf{x}_e$ :

$$F(\mathbf{x}) = \frac{1}{2} (\mathbf{x} - \hat{\mathbf{x}})^T \mathbf{P}^{-1} (\mathbf{x} - \hat{\mathbf{x}}) + \frac{1}{2} (\mathbf{h}(\mathbf{x}) - \mathbf{z})^T \mathbf{R}^{-1} (\mathbf{h}(\mathbf{x}) - \mathbf{z}) \quad (3.86)$$

The answer to this is that they are exactly equivalent. This section (and appendix A) proves this equivalence. The relationships among these quadratics are also summarised in appendix A.3.

Equation 3.86 is a standard expression relating to the log-PDF of the estimation problem of this section. It also indicates an alternative distance measure, the residual Mahalanobis distance. The residual Mahalanobis distance is summarised as follows:

- Each term is compared against a common state estimate  $\mathbf{x}_e$ , typically the optimal posterior (MAP) estimate. The estimate,  $\mathbf{x}_e$ , is formed from the fusion of the prior and the observation.  $\mathbf{x}_e$  is distinct from the prior estimate  $\hat{\mathbf{x}}$ .  $\mathbf{x}_e$  is correlated to both  $\hat{\mathbf{x}}$  and to  $\mathbf{z}$  since these are used to calculate  $\mathbf{x}_e$ .
- Now, instead of defining a single *innovation*, define two *residuals* representing the difference of each term (observation and prior) away from their predicted value given the posterior state  $\mathbf{x}_e$ .
- The residual of the observation is defined as the difference between the observation obtained and the observation predicted from the posterior state  $\mathbf{x}_e$ :

$$\boldsymbol{\nu}_z(\mathbf{x}) \triangleq \mathbf{h}(\mathbf{x}) - \mathbf{z} \quad (3.87)$$

$$M_z(\mathbf{x}) \triangleq \frac{1}{2}(\mathbf{h}(\mathbf{x}) - \mathbf{z})^T \mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}) - \mathbf{z}) \quad (3.88)$$

The residual of the prior is defined as the difference between the prior  $\hat{\mathbf{x}}$  and the corresponding value predicted from the posterior state  $\mathbf{x}_e$ :

$$\boldsymbol{\nu}_p(\mathbf{x}) \triangleq \mathbf{x} - \hat{\mathbf{x}} \quad (3.89)$$

$$M_p(\mathbf{x}) \triangleq \frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^T \mathbf{Y}(\mathbf{x} - \hat{\mathbf{x}}) \quad (3.90)$$

- Note that the innovation  $\boldsymbol{\nu}_i$  and residuals  $\boldsymbol{\nu}_z$  are not to be confused with the observation Lagrange multiplier  $\boldsymbol{\nu}$ . They are, however, closely related. See the appendix section A.2.

- The residual Mahalanobis distance is given by:

$$M_r(\mathbf{x}) = M_z(\mathbf{x}) + M_p(\mathbf{x}) \quad (3.91)$$

$$M_r(\mathbf{x}) = \frac{1}{2} \boldsymbol{\nu}_z^T \mathbf{R}^{-1} \boldsymbol{\nu}_z + \frac{1}{2} \boldsymbol{\nu}_p^T \mathbf{P}^{-1} \boldsymbol{\nu}_p \quad (3.92)$$

$$= F(\mathbf{x}) \quad (3.93)$$

- The claim here is that the residual Mahalanobis distance (Equation 3.92) evaluated at the solution,  $\mathbf{x} = \mathbf{x}_e$  and the innovation Mahalanobis distance (Equation 3.85) are equal:

$$M_i = M_r(\mathbf{x}_e) \quad (3.94)$$

This is proven in appendix A.5.

### 3.6.3 Discussion

The residual distance and the innovation distance are mathematically equivalent in the two term, full rank case presented above. However, under more general circumstances there are differing properties and benefits. These are outlined in table 3.1.

**Table 3.1:** Residual vs. Innovation distance measures.

Residual Distance	Innovation Distance
Describes a distance measure between multiple estimation terms symmetrically. Each term is referenced against a common state estimate. The terms are represented separately, without reference to each other. Additional distance terms are added into the residual distance (equation 3.92) for each observation and prior. The resulting expression is associated with the graph structure of the system.	Describes a distance measure between only two estimation terms in a pair. The two terms are referenced against each other.
Involves the terms $\mathbf{R}^{-1}$ & $\mathbf{P}^{-1}$ which allows zero-information cases (non-invertible prior information $\mathbf{Y}$ ), but excludes constraint cases (non-invertible $\mathbf{R}$ or $\mathbf{P}$ ). Distance measures under constraints are inherently difficult since any infinitesimal deviation from the constraint has infinite cost.	Involves the term $(\mathbf{H}\mathbf{P}\mathbf{H}^T + \mathbf{R})^{-1}$ , which excludes zero-information cases (non-invertible prior information $\mathbf{Y}$ ).
A measure of the consistency of a particular solution, $\mathbf{x}_e$ , rather than the intrinsic consistency of the terms involved. The measure of the intrinsic consistency of the terms is obtained by simply evaluating at a MAP solution.	Refers directly to the terms involved without reference to a proposed solution point.

**Table 3.1:** Residual vs. Innovation distance measures. (continued)

Residual Distance	Innovation Distance
Suitable for operation with multiple observation and prediction terms. As a result, the residuals approach is well suited to the trajectory state and observation augmented approaches.	Suited to operation with a pair of terms, conventionally a prediction and an observation. This approach is therefore well suited to a straightforward predict-observe filtering context.

### 3.6.4 Multiple Observation Terms

The residual Mahalanobis distance expression is more suitable for evaluating the consistency of scenarios with multiple terms than the innovation Mahalanobis distance expression.

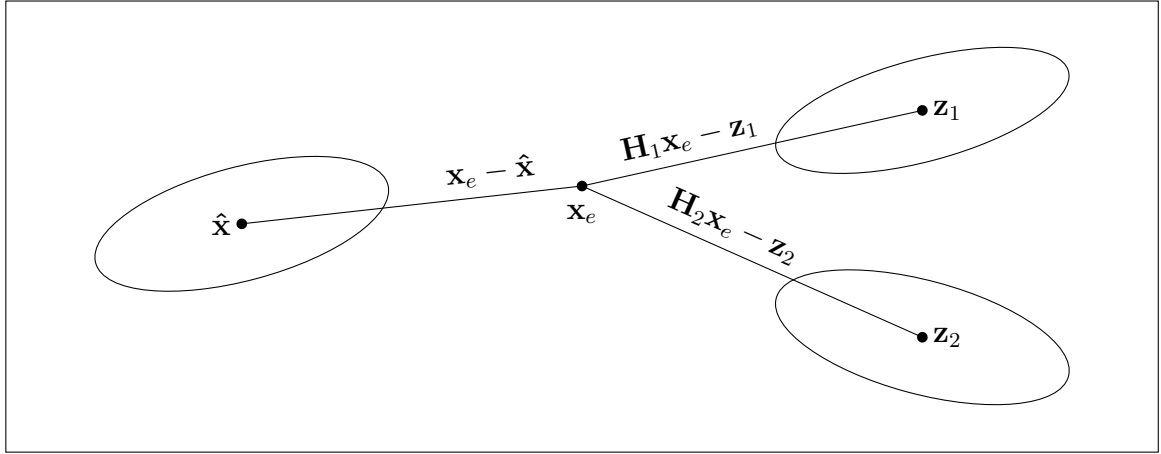
In figure 3.7, a prior information term is observed twice simultaneously. In general the observation functions  $\mathbf{H}_1$  and  $\mathbf{H}_2$  will correspond to different types of observation with different observation dimensions. The sum in equation 3.92 is easily able to extend to the multiple observation case, for example:

$$M_r = \frac{1}{2} \boldsymbol{\nu}_{\mathbf{z}_1}^T \mathbf{R}_1^{-1} \boldsymbol{\nu}_{\mathbf{z}_1} + \frac{1}{2} \boldsymbol{\nu}_{\mathbf{z}_2}^T \mathbf{R}_2^{-1} \boldsymbol{\nu}_{\mathbf{z}_2} + \frac{1}{2} \boldsymbol{\nu}_p^T \mathbf{P}^{-1} \boldsymbol{\nu}_p \quad (3.95)$$

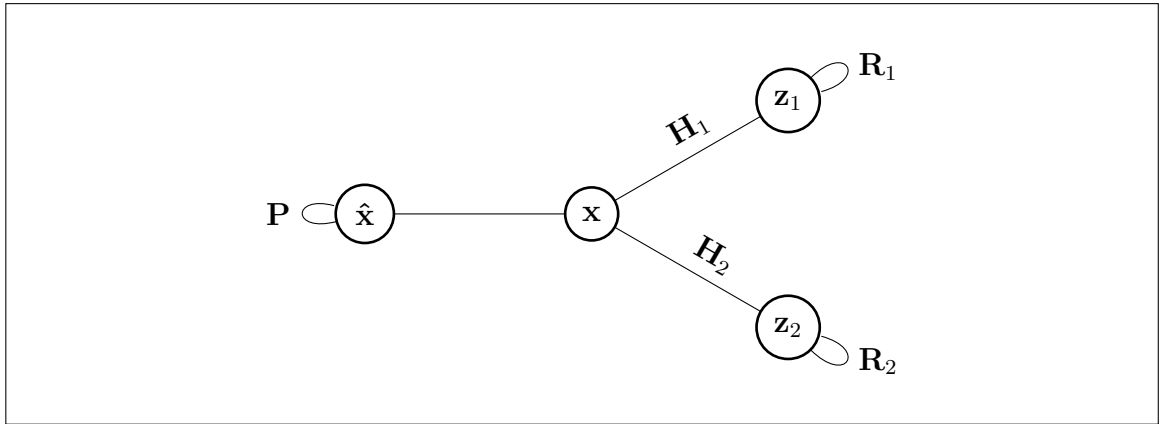
Similarly, in figure 3.8, a state linked to both past and future states by a dynamic model has no clear “prediction” direction. In this case the residual Mahalanobis distance operates symmetrically across all the involved terms in a clear manner.

By comparison, it is not clear how to apply the *innovation* distance to these multiple observation and prediction scenarios, since the innovation approach relies on converting the prior information term into the observation space (see section 3.6.1).





(a) Spatial Illustration showing the projection of the observations into the state space.



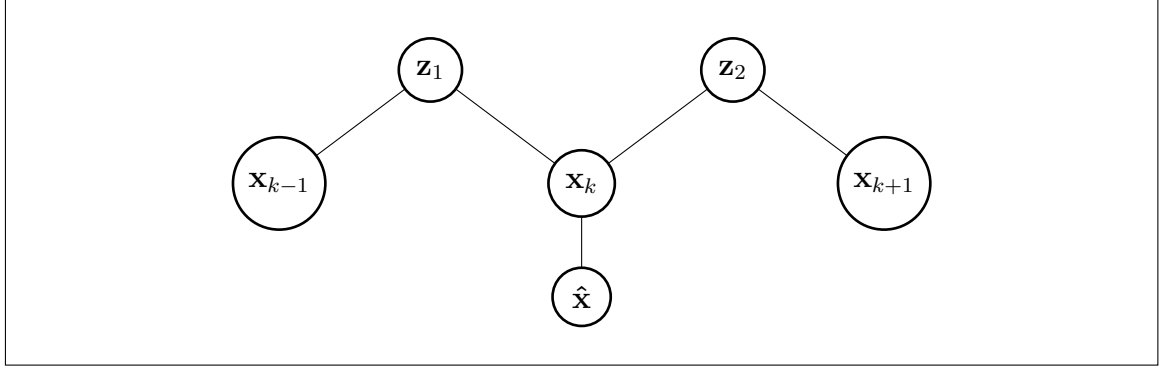
(b) Graphical model (see section 2.6) showing the structure of the state and observation terms.

**Figure 3.7:** Illustration of the *residual* approach for multiple observations. Three residuals are produced for the three terms. The Mahalanobis distance for the total configuration is then obtained from the sum in equation 3.92 extended to three terms (for the single prior and two observations).

The scenario of “multiple observations” applies to the trajectory state formulation. In the trajectory state approach, a particular instance of a dynamic state will naturally be linked by the dynamic model to the past and future state instances. In that case, if one or more observations link to this state, then the state will have a set of two prediction and one or more observation terms. The residual Mahalanobis distance is then still applicable to these multiple observation terms (including the dynamic model

links). This is depicted in figure 3.8.

The observations and predictions are therefore treated symmetrically. Each observation or prior model checks for residuals against the state estimate,  $\mathbf{x}_e$ , and the total Mahalanobis distance is obtained from a sum over the individual terms' distances.



**Figure 3.8:** Graphical model (see section 2.6) for a multiple-residual case arising from a trajectory smoothing structure. Prediction models link the state  $\mathbf{x}_k$  to the past and future, together with a plain prior term on  $\mathbf{x}_k$ . This structure would require three terms for the residual Mahalanobis distance (one for each prediction observation plus one for the prior). By contrast it is not clear how the conventional innovation distance would apply to this multiple-term scenario.

Therefore the residual Mahalanobis distance expression is more suitable for evaluating the consistency of scenarios with multiple terms, including the trajectory state case, than the innovation Mahalanobis distance expression.

### 3.6.5 Chi-Squared Degrees of Freedom

Under conditions in which the noises in equations 3.76 and 3.78 are Gaussian, the resulting distances,  $M_r$  &  $M_i$  are  $\chi^2$  (chi-squared) random variables.

The shape of the  $\chi^2$  distribution is affected by the number of *degrees of freedom* (DoF) in the interacting squared Gaussian random variables. The number of degrees of

freedom depends on the dimension of the underlying variables, but also on the rank and directions of the terms involved.

The residual Mahalanobis distance expression is suitable for scenarios with general rank  $\mathbf{Y}$  and general dimensions of the observation. By comparison, the innovation Mahalanobis distance is suitable only for full rank prior information. Under general rank conditions, the *degrees of freedom* of the  $\chi^2$  distribution are slightly more complicated than in the full rank case.

The number of degrees of freedom of the Mahalanobis distance depends on the number of dimensions that are shared in common between the various terms, because it is these in which disagreement (and hence residuals) can arise. Terms which contribute information in an entirely orthogonal direction to all the others cannot give rise to any residual, since there are no other terms to disagree in that direction, and hence this dimension does not contribute to the number of degrees of freedom of the  $\chi^2$  distribution of the Mahalanobis distance.

The dimension of the  $\chi^2$  distribution is given by:

$$n_{\text{DoF}} = \sum_i \text{rank}(\mathbf{Y}_i) - \text{rank}\left(\sum_i \mathbf{Y}_i\right) \quad (3.96)$$

Where all observation and prior terms are written in the information form,  $\mathbf{Y}_i$ .

If there are only two terms, each with full rank ( $\text{rank}(\mathbf{Y}_i) = n_{\text{state}}$ ) then equation 3.96 reduces down to  $n_{\text{DoF}} = n_{\text{state}}$ .

In the innovation distance, there are only two terms and the prior covariance term is usually assumed to be full rank  $n_{\text{state}}$ . Therefore the degrees of freedom of the innovation Mahalanobis distance  $\chi^2$  distribution is the dimension of the observation.

It is possible in the residual Mahalanobis distance to obtain  $n_{\text{DoF}} = 0$ . This indicates that each of the information terms is operating in orthogonal directions to the others. This indicates that the solution is acceptable but has no built in redundancy for error

checking.  $M_r$  will be obtained as zero at the solution. For example:

$$\mathbf{Y}_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \mathbf{Y}_2 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \quad (3.97)$$

or:

$$\mathbf{Y}_1 = \begin{pmatrix} +1 & -1 \\ -1 & +1 \end{pmatrix} \quad \mathbf{Y}_2 = \begin{pmatrix} +1 & +1 \\ +1 & +1 \end{pmatrix} \quad (3.98)$$

### 3.6.6 Lagrange Multipliers for Measurement of Consistency

The observation Lagrange multipliers are useful for the analysis of the consistency of the system.

Consider an individual observation (or constraint) term in the context of a broader system. Measures of consistency for this term can be derived from  $\boldsymbol{\nu}_z$  and  $M_z(\mathbf{x})$  as well as  $\boldsymbol{\nu}$ :

$$\boldsymbol{\nu}_z = \mathbf{h}(\mathbf{x}) - \mathbf{z} \quad (3.99)$$

$$M_z(\mathbf{x}) = \frac{1}{2}(\mathbf{h}(\mathbf{x}) - \mathbf{z})^T \mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}) - \mathbf{z}) \quad (3.100)$$

$$\boldsymbol{\nu} = -\mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}) - \mathbf{z}) \quad (3.101)$$

- $M_z(\mathbf{x})$  is the observation Mahalanobis distance, a positive scalar.
- $\boldsymbol{\nu}_z$  is the observation residual, a vector in observation space.
- $\boldsymbol{\nu}$  is the observation Lagrange multiplier, which is a vector in *inverse* observation units.

For an observation which is perfectly consistent with the rest of the system (the solution  $\mathbf{x}$  will be identical with or without this observation term), then  $M_z(\mathbf{x})$ ,  $\boldsymbol{\nu}_z$  and  $\boldsymbol{\nu}$  are all zero.

The difference between these is significant in the presence of *constraints*.

For constraints the residual  $\nu_z$  and Mahalanobis distance  $M_z(\mathbf{x})$  will both be zero at the solution and hence these give no indication regarding any potential conflict between the constraint and other linked terms. By comparison the Lagrange multiplier  $\nu$  for constraints will obtain varying values according to the direction and extent of any conflict with other terms.

In data association, entities which may be initially distinct can be identified as being the same and forced into equality by linking them with equality constraints. The value of the equality Lagrange multiplier indicates the “force” required to bring the states into agreement and thus indicates the level of inherent agreement.

It becomes important later in this chapter whether the system needs the Lagrange multiplier variables. If they are *not* needed in the application, then their existence in the system must be justified by their ability to speed up the solution for the state estimates. This can apply in some extreme cases (see example 3.4) but for localisation and mapping it generally does not. However, if the Lagrange multipliers are needed (as is argued in this thesis), then it does become worthwhile augmenting them jointly with the states and solving for everything jointly. Future work will apply this mechanism of Lagrange multipliers to the full analysis of consistency and implementation of data association.

**Force and Energy Interpretations** A linear weighted estimation problem is mathematically equivalent to a system of interconnected linear springs. The observation Lagrange multipliers are mathematically equivalent to *forces* in such a system.

The augmented system form is equivalent to a combined position and force formulation of the linear spring system based on stiffness. The information form is equivalent to a position-only formulation of the system based on energy.

The solution can be equivalently described as a position with either minimum energy or zero net force. At minimum energy, the derivative of energy with respect to position is zero. The derivative of energy with respect to position is the net force.

The force formulation explicitly includes evaluation of the forces throughout the system, whereas the energy formulation amortises the various forces together into energy terms.

A similar force based mechanical analogy is drawn in [35, 36] which in turn cites matrix based methods in structural analysis for the analysis of stiffness modes and internal stresses in the estimation network. This could be an interesting avenue for further investigation along the lines of the data association and consistency analysis mentioned above.

### 3.6.7 Conclusion

This section proposed an alternative form of Mahalanobis distance, the *residual* distance, which is equivalent to the innovation distance, but is a more general expression. The main purpose of the residual distance is to provide a distance measure for measuring consistency under more complex situations. The proposed residual distance suits the other estimation structures proposed in this thesis, particularly for evaluating cases with multiple prediction and observation models, and for evaluating consistency throughout a complex network of states and observations. The residual distance also achieves the goal of generalising the innovation distance for cases with *rank-deficient* prior and/or observation terms.

These advantages will be of benefit in future work for implementing the data association and online verification algorithms under the trajectory state and augmented system approaches.

## 3.7 Benefits for Estimation

This section will discuss the benefits to the estimation process in using the augmented system form compared to using the information form.

The augmented system form has the following benefits which will be discussed in this section:

- Improved sparsity for problems with large observation degree and small state degree.
- Improved numerical stability for constraints and tight-observations.

These are achieved by having the option of factorising some states ahead of their observations.

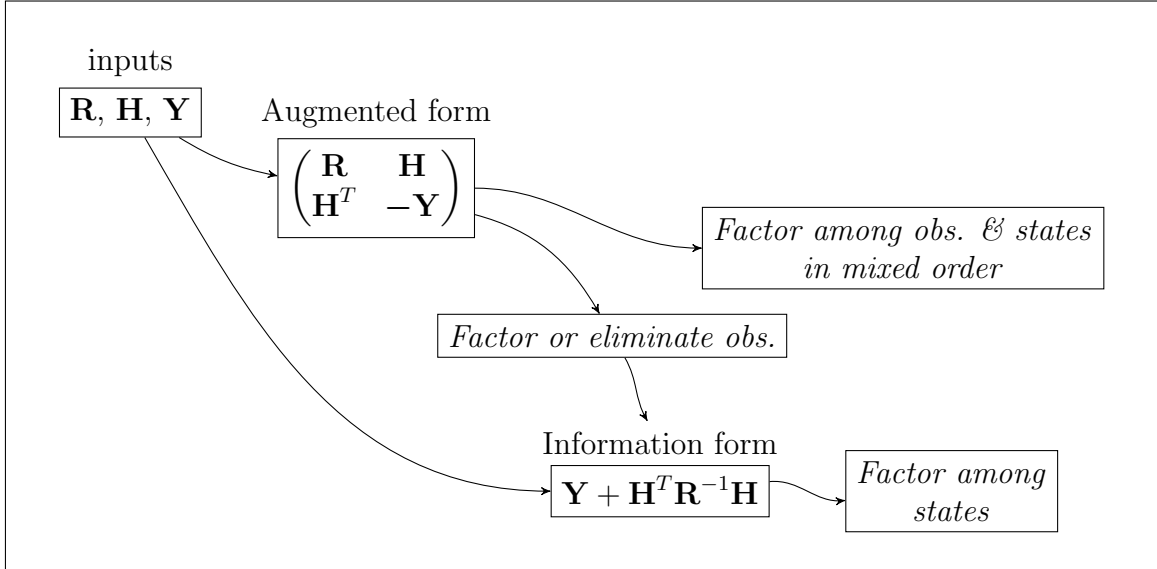
The main alternative solving approaches to be discussed in this section are shown in figure 3.9. Given the input  $\mathbf{R}$ ,  $\mathbf{H}$  and  $\mathbf{Y}$  systems, the alternatives considered here consist of forming either the augmented system form or the information form first. The augmented system form may then be solved by factorising the observations first or factorising among the observations and states in a mixed order. The approach of solving the augmented system form by factorising the observations first is equivalent to using the information form and therefore will not be discussed separately. The two basic alternatives discussed here therefore consist of:

- Factorising the observations first and states second. This is equivalent to the information form.
- Factorising in a mixed order among the observations and states. This is only possible if the augmented system form is used. (The approach of factorising the states first is another possibility, but that is a subset of the mixed approach).

### 3.7.1 Factorisation Ordering for Sparsity

This section discusses the benefits of the augmented form in relation to factorisation orderings for sparsity by comparison to the information form.

Choosing a factorisation ordering for minimum fill in is an NP-complete problem [30]. For small examples it is possible to explicitly evaluate all orderings. For larger examples the algorithm `colamd` [18] is used. This thesis does not propose new factorisation ordering algorithms. Instead, this thesis claims that the augmented system form gives



**Figure 3.9:** Alternative system forms and solving approaches. Given the system’s  $\mathbf{R}$ ,  $\mathbf{H}$  and  $\mathbf{Y}$  formulation, the paths considered in the discussion consist of forming either the augmented system form or the information form first. The augmented system form may then be solved either by factorising the observations first, which effectively constructs the information form, or by factorising among the observations and states in a mixed order. However, if the *information form* is formed first, it can then only be solved by factorising among the states.

a wider range of factorisation ordering possibilities than the information form, since it is able to choose a factorisation ordering from both the observations and constraints.

The factorisation orderings for particular examples are compared by evaluating the number of nonzeros in the  $\mathbf{L}$  factor of the LDL factorisation of the augmented system form or information form. The number of nonzeros of  $\mathbf{L}$  is an appropriate measure because:

- It is a mathematical property of the system and the factorisation order, rather than a measure specific to the computational environment or implementation (unlike, for example, the time taken for factorisation).
- Nonzeros in  $\mathbf{L}$  each correspond to numerical operations required in the solution.

This section discusses some cases in which it is better for sparsity reasons to eliminate some *states* first, ahead of their observations. These cases motivate the use of the



augmented system form, since performing the factorisation in this order is not possible in the information form.

This section will present a number of examples in order to show the sparsity properties of the augmented system form versus the information form.

### **Large observation degree example**

Example 3.4 below shows a case of large observation degree and small state degree. In this case the augmented system form is more compact than the information form, and the factorisation of states first is more compact than the alternative observations first.

### **Large state degree example**

On the other hand, example 3.5 shows one particular case of superior performance of the *information form*. This demonstrates that the relative performance of the augmented system form and the information form depends on the graph structure properties between the observations and states.

### **A Dynamic vehicle and map example**

Example 3.6 introduces an example containing a sequence of vehicle states and some features. These are linked by models of typical sizes for localisation and mapping problems. The resulting sparsity and factorisation patterns consist of a mix of observations and states and the augmented system form becomes advantageous when the system actually needs the values for both the states and Lagrange multipliers.

**Example 3.4.*****Sparsity factorisation ordering in a large observation degree case***

*In this example each observation links to many states (large observation degree) but each state is only linked to a few observations (small state degree). For this example the observation is defined as measuring the sum of the states as shown in figure 3.12.*

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & \cdots & 1 & 1 \end{pmatrix} \quad R = 1 \quad \mathbf{Y} = \mathbf{I} \quad (3.102)$$

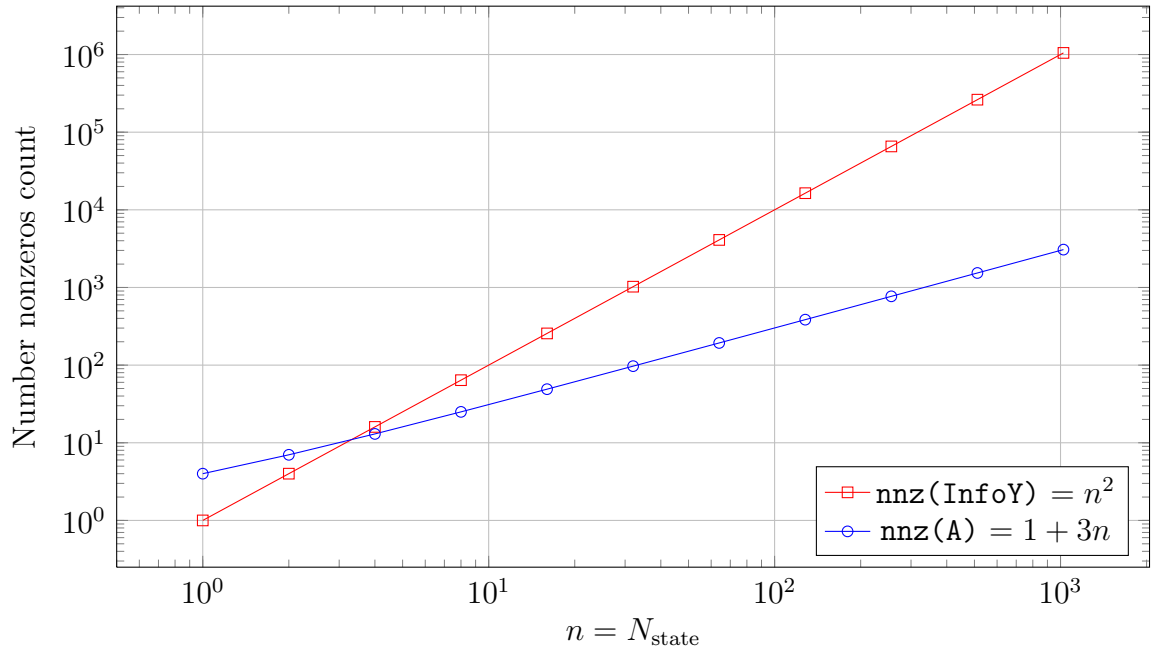
*The number of nonzeros in the augmented system matrix ( $\mathbf{A}$ ) will be compared against that of the information matrix ( $\mathbf{Y}^+ = \mathbf{Y} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}$ ). This example will also compare the number of nonzeros in the triangular factors ( $\mathbf{L}$ ) under the permutations: (observation,states), (states,observation) and (states only). The case of considering only the states (as in the information form) is a subset of the (observation,states) ordering obtained by discarding the eliminated observations.*

**Results**

*Figure 3.10 and table 3.2 show the number of nonzeros in the augmented form and the information form. For  $N_{state} \geq 4$  the augmented form has fewer nonzeros than the corresponding information form as it scales linearly. Figure 3.11 and table 3.3 show the number of nonzeros in  $\mathbf{L}$ . The augmented system form allows the use of the ordering (states, observations) which is sparser than the alternatives for  $N_{state} \geq 4$ .*

**Discussion**

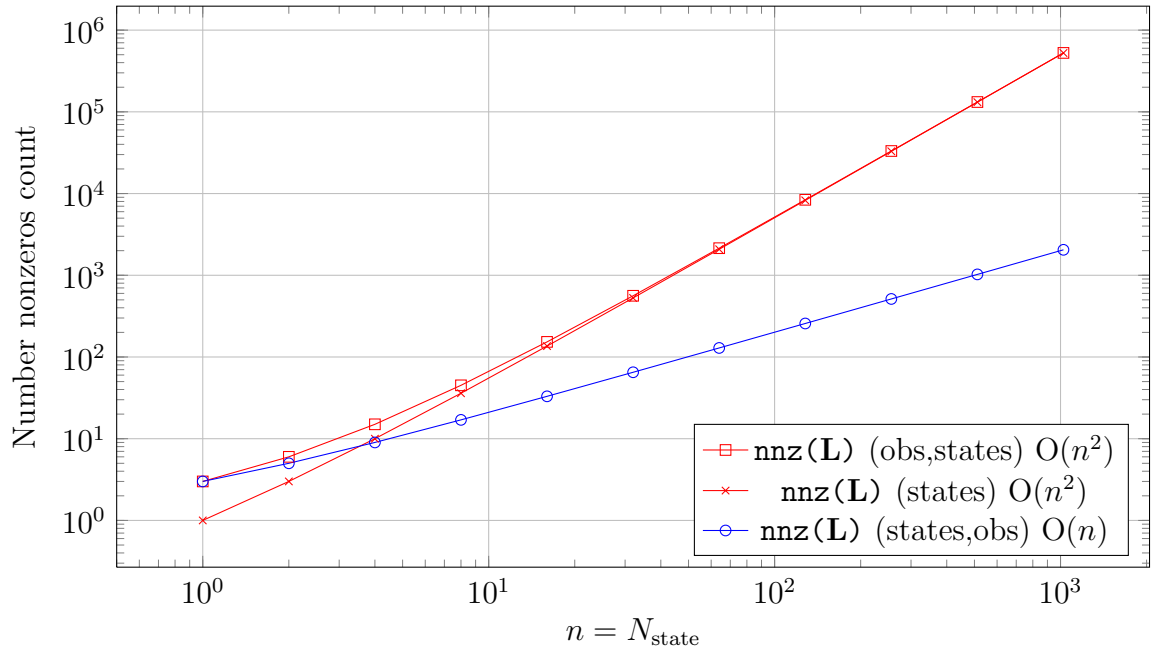
*Figures 3.12 and 3.13 illustrate  $\mathbf{A}$ ,  $\mathbf{Y}^+$  and  $\mathbf{L}$  for the case  $N_{state} = 6$ . Figure 3.12a shows the augmented system form. The number of nonzero entries is  $1 + 3n = 19$ . By comparison the information form, shown in figure 3.12b, has the full  $n^2 = 36$  entries, due to the dense single observation of the sum of all states in this example. Therefore the augmented form is a more compact representation in this case. Similarly, figure 3.13a shows the  $\mathbf{L}$  factor of the system when factorised in the (states,observation) ordering. This preserves the sparsity found in the augmented form. By comparison, figure 3.13b shows the  $\mathbf{L}$  factor of the system under the (observation,states) ordering, which results in dense fill in similarly to the information form.*



**Figure 3.10:** Number of nonzeros in the augmented form and the information form for various  $N_{\text{state}}$ , in the case of a large observation degree & small state degree.

**Table 3.2:** Number of nonzeros in the augmented form and the information form, in the case of a large observation degree & small state degree.

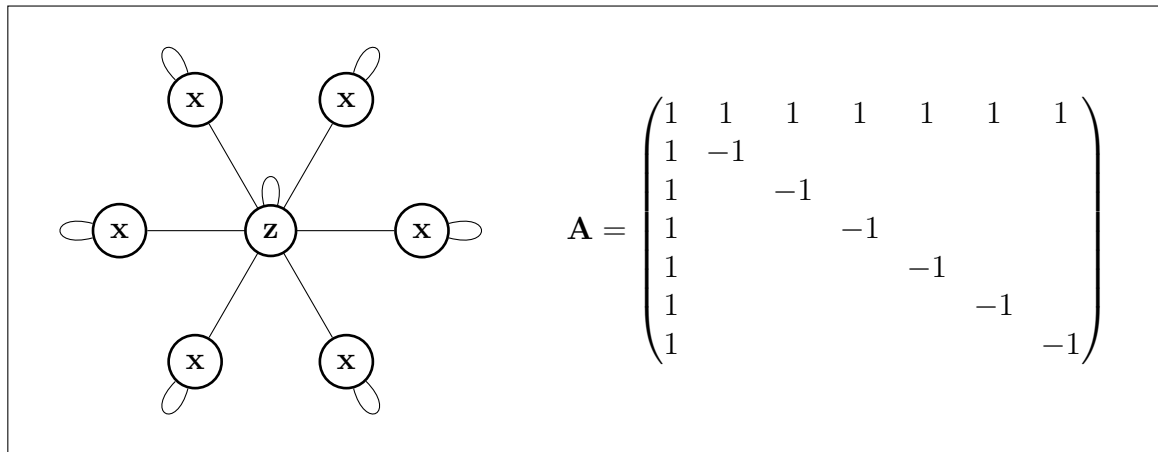
$N_{\text{state}}$	$\text{nnz}(\mathbf{A})$	$\text{nnz}(\mathbf{Y}^+)$
1	4	1
2	7	4
4	13	16
8	25	64
16	49	256
32	97	1,024
64	193	4,096
128	385	16,384
256	769	65,536
512	1,537	262,144
1,024	3,073	1,048,576



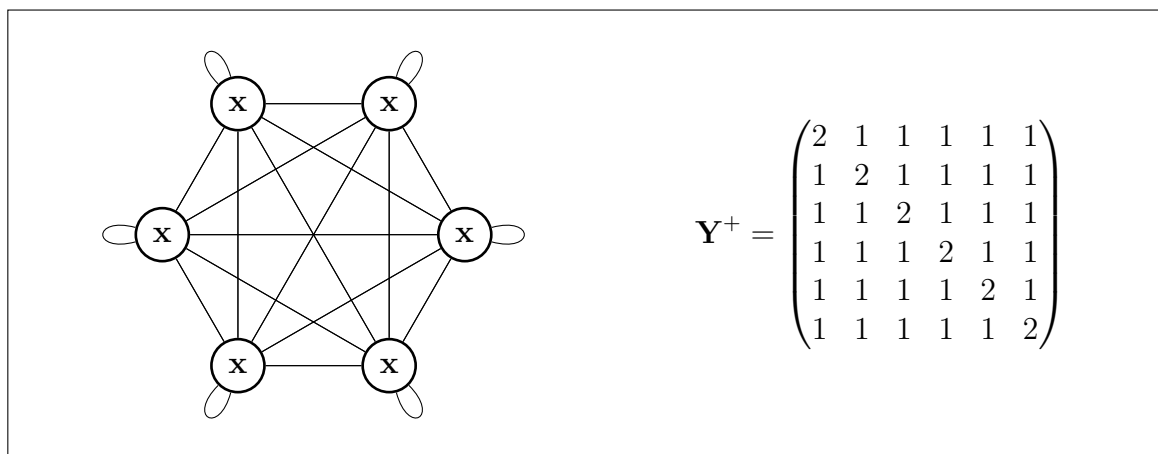
**Figure 3.11:** Number of nonzeros in the  $\mathbf{L}$  factor for various ordering approaches, in the case of a large observation degree & small state degree..

**Table 3.3:** Number of nonzeros in the  $\mathbf{L}$  factor for various ordering approaches, in the case of a large observation degree & small state degree.

$N_{\text{state}}$	(states,obs)	(obs,states)	(states)
1	3	3	1
2	5	6	3
4	9	15	10
8	17	45	36
16	33	153	136
32	65	561	528
64	129	2,145	2,080
128	257	8,385	8,256
256	513	33,153	32,896
512	1,025	131,841	131,328
1,024	2,049	525,825	524,800

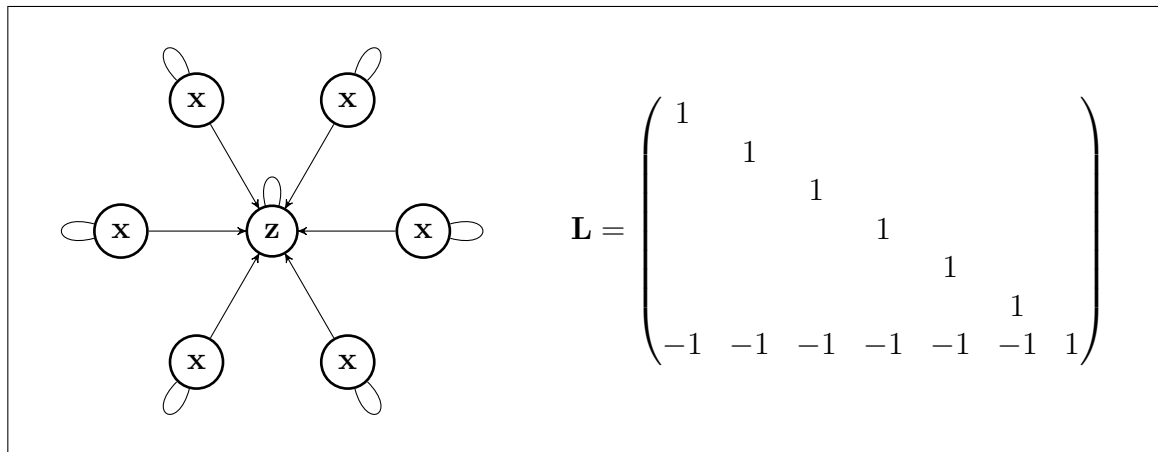


(a) The augmented system form,  $\mathbf{A}$ .

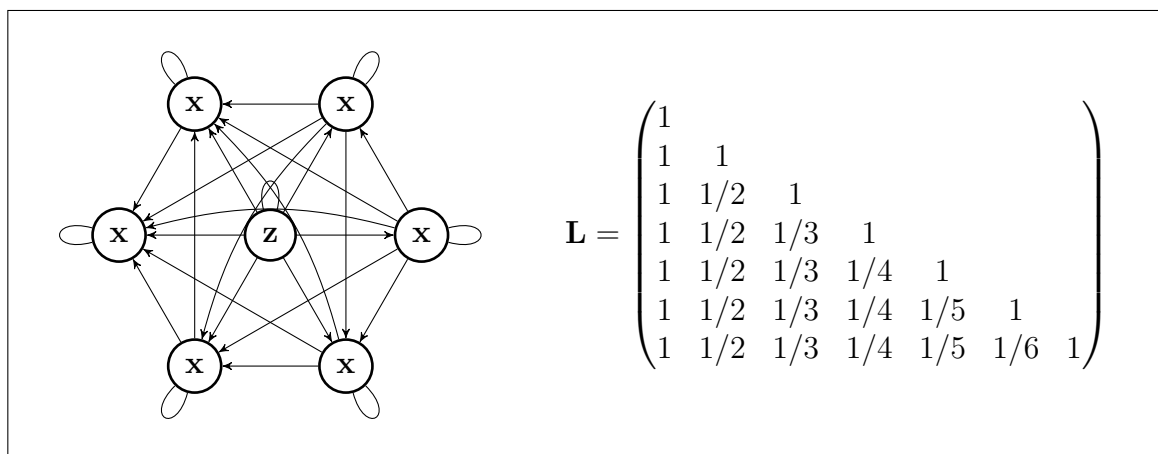


(b) The information form  $\mathbf{Y}^+$

**Figure 3.12:** A large observation degree, small state degree example showing the systems  $\mathbf{A}$  and  $\mathbf{Y}^+$



(a)  $\mathbf{L}$  under the (states,obs) ordering.



(b)  $\mathbf{L}$  under the (obs,states) ordering.

**Figure 3.13:** A large observation degree, small state degree example showing the  $\mathbf{L}$  for the alternative orderings.

*This example showed an extreme case of large observation degree and small state degree: A single dense observation across all states. This example demonstrated a case in which the augmented system form is more compact than the information form, and the factorisation of states first is more compact than the alternative observations first.*

---

**Example 3.5.** 

---

***Sparsity factorisation orderings in a large state degree case***

*In this example each state links to many observations (large state degree) but each observation is only linked to a one state (small observation degree). This is the opposite scenario than that presented in example 3.4.*

*The large state degree, small observation degree scenario is the motivation behind the information form. The idea is that the representation of the posterior system in the states only is more compact than the representation in the joint (observation, state) system.*

*This example is included to show how the relative advantages of the augmented system form compared to the information form depend on the structural properties of the system. In this example, the large state degree & small observation degree leads to a compact information form.*

*This example defines a single scalar state and defines many ( $N_{obs}$ ) scalar observations simply observing the state value.*

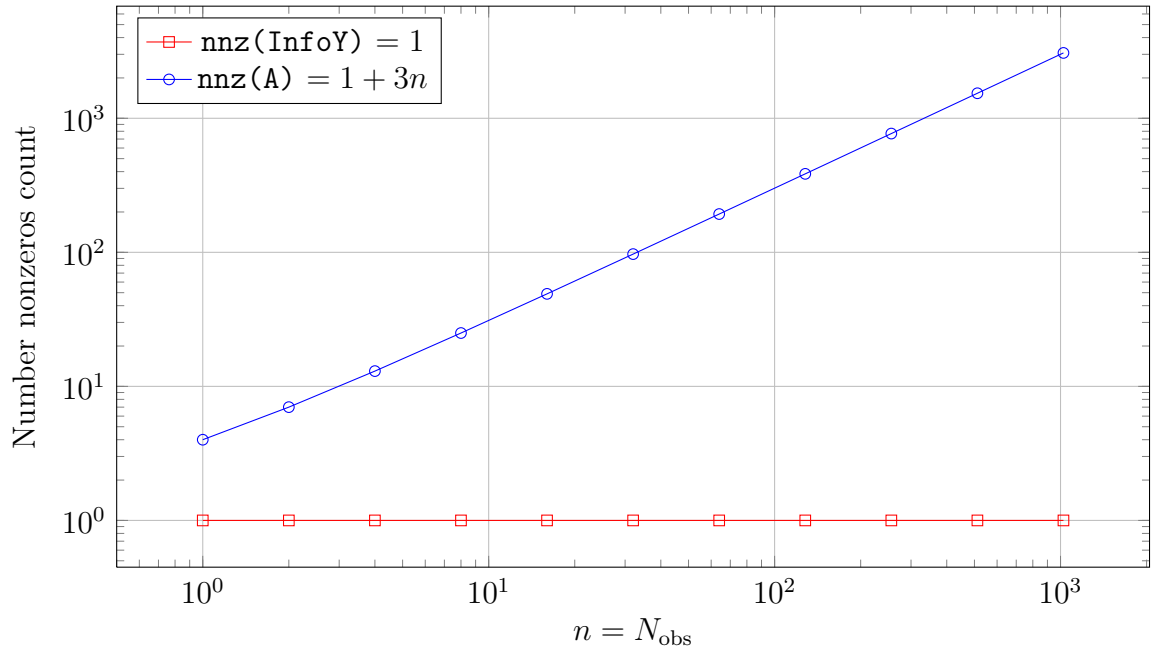
$$\mathbf{H} = \begin{pmatrix} 1 & 1 & \cdots & 1 & 1 \end{pmatrix}^T \quad \mathbf{R} = \mathbf{I} \quad Y = 1 \quad (3.103)$$

**Results**

*Figure 3.14 and table 3.4 show the number of nonzeros of the augmented system form and the information form. For this example the information form has a constant size equal to  $N_{state} = 1$  regardless of the number of observations. The augmented system form has a number of nonzeros equal to  $3N_{obs} + 1$ .*

*Figure 3.15 and table 3.3 show the number of nonzeros in the  $\mathbf{L}$  factor of  $\mathbf{A}$  for various ordering approaches. Factorising the states first results in an  $O(n^2)$  fill-in in the observations whereas factorising the observations first maintains an  $O(n)$  sized factor. The  $\mathbf{L}$  factor for the (scalar) information form is simply a scalar in this single-state example.*

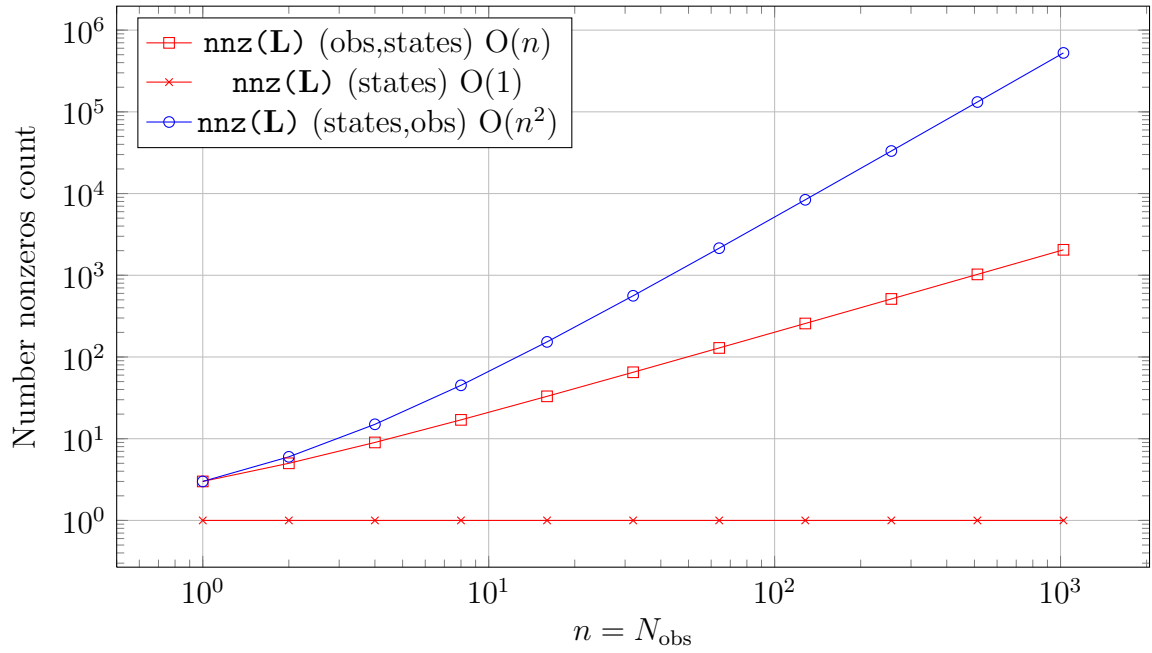




**Figure 3.14:** Number of nonzeros in the augmented form and the information form for various  $N_{\text{state}}$ , in the case of a large state degree & small observation degree.

**Table 3.4:** Number of nonzeros in the augmented form and the information form, in the case of a large state degree & small observation degree.

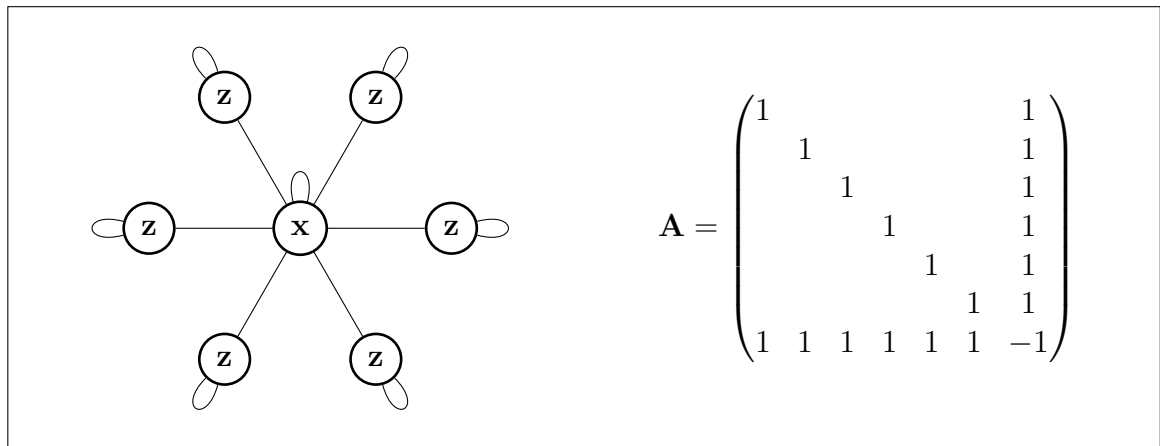
$N_{\text{obs}}$	$\text{nnz}(\mathbf{A})$	$\text{nnz}(\mathbf{Y}^+)$
1	4	1
2	7	1
4	13	1
8	25	1
16	49	1
32	97	1
64	193	1
128	385	1
256	769	1
512	1,537	1
1,024	3,073	1



**Figure 3.15:** Number of nonzeros in the  $\mathbf{L}$  factor for various ordering approaches, in the case of a large state degree & small observation degree.

**Table 3.5:** Number of nonzeros in the  $\mathbf{L}$  factor for various ordering approaches, in the case of a large state degree & small observation degree.

$N_{\text{obs}}$	(states,obs)	(obs,states)	(states)
1	3	3	1
2	6	5	1
4	15	9	1
8	45	17	1
16	153	33	1
32	561	65	1
64	2,145	129	1
128	8,385	257	1
256	33,153	513	1
512	131,841	1,025	1
1,024	525,825	2,049	1

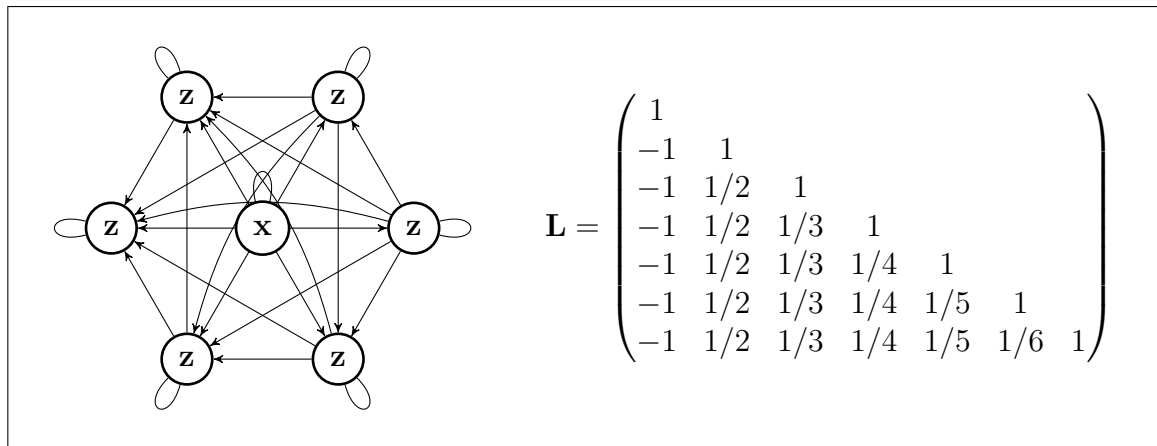


(a) The augmented system form,  $\mathbf{A}$ .

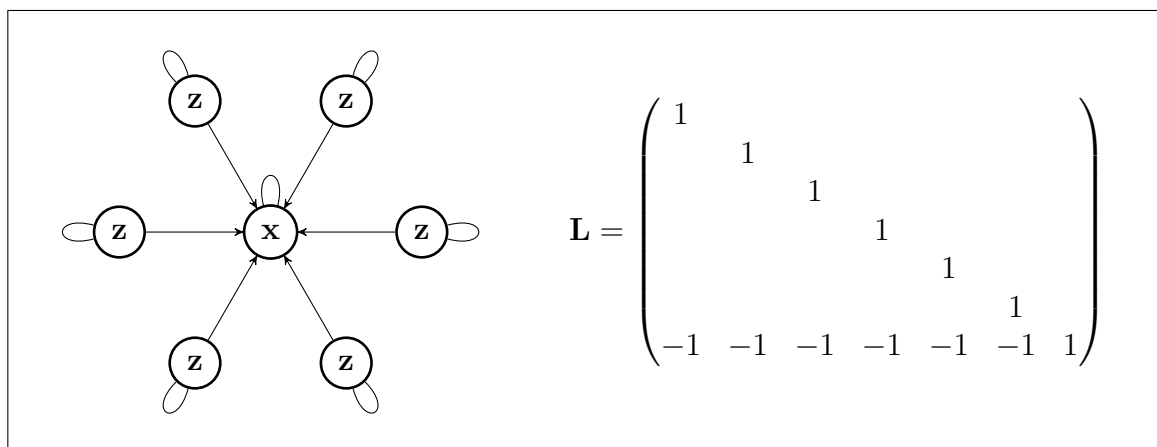


(b) The information form  $\mathbf{Y}^+$

**Figure 3.16:** A large state degree, small observation degree example showing the systems  $\mathbf{A}$  and  $\mathbf{Y}^+$



(a)  $\mathbf{L}$  under the (states,obs) ordering.



(b)  $\mathbf{L}$  under the (obs,states) ordering.

**Figure 3.17:** A large state degree, small observation degree example showing the  $\mathbf{L}$  for the alternative orderings.

**Discussion**

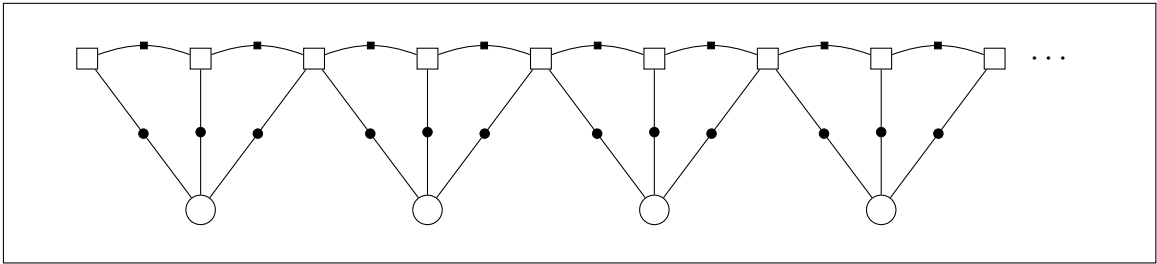
*The elimination of the observations into the information form results in a compact (scalar) state posterior in this example (figure 3.16). The fill-in resulting from the factorisation of states is illustrated in figure 3.17a. Therefore the information form is a more compact representation in this particular example. Similarly, the factorisation ordering (observations,state) is also more efficient than the alternative (state,observation) in this particular example. The use of the information form is motivated by the assumption of large state degree and small observation degree. Under this assumption the information form is a more compact and efficient representation and solving ordering for the estimation problem. However, this assumption does not hold true for all observations in all systems and is not guaranteed in general. This example showed the opposite case than example 3.4. In this example the large number of observations linking to a small state results in superior performance of the information form. This demonstrates that the relative performance of the augmented system form and the information form depends on the graph structure properties between the observations and states.*

---

**Example 3.6.*****Sparsity factorisation in a localisation and mapping example***

Example 3.4 showed how the augmented system form allows the factorisation of states ahead of observations, which is beneficial for scenarios with large observation degree and small state degree. That example used the extreme case of a single observation linking to all states.

This example considers the same properties of the augmented system form regarding the factorisation orderings for sparsity. However, this example considers an observation and state pattern typical of estimation problems in localisation and mapping. This example will consider a system with a sequence of vehicle states linked by a dynamic model and a sequence of feature states linked to the vehicle states by an external observation model (vision observations). This structure is illustrated in figure 3.18. The chain of states, observations and features consists of 101 vehicle states and 50 features. The dimensions of the various models and states are important because



**Figure 3.18:** Structure of states and observations for this example. The vehicle states (nodes  $\square$ ) are linked in a chain by dynamic model observations (nodes  $\blacksquare$ ). The vehicle states link to feature states (nodes  $\bigcirc$ ) via vision observations (nodes  $\bullet$ ). Each feature is linked to three vehicle states as shown. Each state and observation node shown in the figure represents a cluster of dimension  $N_{\text{state}}$  or  $N_{\text{obs}}$ . Each observation-state link shown in the figure represents a cluster of  $N_{\text{state}} \times N_{\text{obs}}$  links.

these affect the degree properties of the graph structure of the system, which affects the sparsity effects of various factorisation approaches. This example assumes that given an observation of size  $N_{\text{obs}}$  linking to a state  $N_{\text{state}}$  that the full  $N_{\text{obs}} \times N_{\text{state}}$  scalar-scalar links are used and are nonzero. The dimensions of the various models are given on the next page. The dimensions are summarised in tables 3.6 and 3.7.

**Vehicle states**

*The vehicle state consists of the position, velocity and attitude. The vehicle position and velocity are assumed to be 3D. The vehicle attitude parametrisation is assumed to be a 4D quaternion. The vehicle state is therefore of dimension 10.*

**Feature states**

*The feature state is assumed to be 3D.*

**External observations**

*External observations between a vehicle and a feature may consist of range only (1), bearing only (1-2), range & bearing (3) or Cartesian (3). Therefore the external observation is of dimension in the range (1-3). This example will use dimension 2, representing a vision observation in 3D. The linked states are the feature and vehicle pose (excluding velocity) of total dimension 10.*

**Dynamics observations**

*There are various approaches to the formulation of the dynamic models. This example assumes that the various controls, measurements and models of the dynamics result in a predicted vehicle pose, given the previous pose. The dimension of the model then becomes dimensions of the residual comparing the poses. Therefore the dimension of the dynamics observations is the dimension of the vehicle pose residual, which is assumed to be the same as the vehicle pose dimension, i.e.: 10. The linked states are the two vehicle poses, of total dimension 20.*

*In summary, the dimensions of the models are given in tables 3.6 and 3.7:*

**Table 3.6:** Summary of dimensions of observations and their linked states

Observation type	observation size	linked state size
external observation	2	10
dynamics observation	10	20

**Table 3.7:** Summary of dimensions of states and their linked observations

State type	state size	linked observation size
feature state	3	6
vehicle position	3	22-24
vehicle velocity	3	20
vehicle attitude	4	22-24

### ***Analysis***

*This section will consider a variety of factorisation orderings of the augmented system form, regarding the ordering of observations and states. Existing ordering algorithms will be used to choose the orderings for each set of variables. The ordering algorithms listed in table 3.8 were considered. Some of these algorithms are sensitive to the initial ordering passed into the algorithm. Therefore, the algorithms were examined over a sequence of 10 random initial orderings. There was also a notable improvement obtained by performing the ordering **colperm** after the random shuffle and before the main algorithm. **colperm** sorts the variables by their vertex degree. The algorithm chosen for analysing the remainder of this example was the sequence (**colperm,colamd**) because it provides a good sparse ordering with little or no sensitivity to the initial ordering. For brevity this sequence will be referred to as **colpermamd**.*



**Table 3.8:** Augmented system  $\mathbf{L}$  factor sparsity for various ordering algorithms. This table lists the number of nonzeros of  $\mathbf{L}$  for the factorisation of  $\mathbf{A}$  under different ordering algorithms. The Matlab name of the algorithm is listed. The resulting ranges for  $\text{nnz}(\mathbf{L})$  correspond to ranges obtained in 10 trials of a random shuffling ordering applied before the algorithms (where indicated).

Sys:	Order:		$\text{nnz}(\mathbf{L} \text{ of sys in order } )$	range
A	shuffle,	colamd	41271 - 41585	314
A	shuffle, colperm,	colamd	41055 - 41055	0
A		colamd	41393	-
A		colperm, colamd	41055	-
A	shuffle,	amd	43193 - 44229	1036
A	shuffle, colperm,	amd	43198 - 44041	843
A		amd	42509	-
A		colperm, amd	42221	-
A	shuffle,	symrcm	41113 - 41113	0
A	shuffle, colperm,	symrcm	41113 - 41113	0
A		symrcm	41113	-
A		colperm, symrcm	41113	-
A	shuffle,	symamd	55710 - 56134	424
A	shuffle, colperm,	symamd	55710 - 55922	212
A		symamd	55922	-
A		colperm, symamd	55922	-
A	shuffle, colperm		802099 - 834965	32866
A		colperm	834898	-

colamd and symamd are described in [18]

amd is described in [4]

symrcm is the symmetric reverse Cuthill-McKee ordering (Matlab)

colperm orders variables by their un-factorised degree (Matlab)

**Sparsity of the initial systems**

Table 3.9 lists the number of nonzeros in the augmented system form and the information form. The number of nonzeros of the augmented system form is greater than that of the information form:  $\text{nnz}(\mathbf{A}) > \text{nnz}(\mathbf{Y}^+)$ . However, the augmented system form represents more of the system than the information form. On this basis, suppose that alongside the information form that a typical implementation will also store the  $\mathbf{H}$  and  $\mathbf{R}$ . Therefore, considering the information form plus the  $\mathbf{H}$  and  $\mathbf{R}$  systems <sup>5</sup>:

$$\text{nnz}(\text{tril}(\mathbf{Y}^+)) + \text{nnz}(\mathbf{H}) + \text{nnz}(\text{tril}(\mathbf{R})) > \text{nnz}(\text{tril}(\mathbf{A})) \quad (3.104)$$

$$47955 > 31472 \quad (3.105)$$

Thus the augmented system form is more compact under this condition where the nonzeros of  $\mathbf{H}$  and  $\mathbf{R}$  are counted onto the information form figures. The augmented system form achieves this compactness by having no fill-in resulting from the elimination of observations.

**Table 3.9:** Sparsity of the un-factorised augmented and information form systems.

Expression	result
$\text{nnz}(\mathbf{A})$	60484
$\text{nnz}(\mathbf{Y}^+)$	36850
$\text{nnz}(\mathbf{H})$	23000
$\text{nnz}(\mathbf{Y}^+) + \text{nnz}(\mathbf{H}) + \text{nnz}(\mathbf{R})$	70450
$\text{nnz}(\text{tril}(\mathbf{Y}^+)) + \text{nnz}(\mathbf{H}) + \text{nnz}(\text{tril}(\mathbf{R}))$	47955
$\text{nnz}(\text{tril}(\mathbf{A}))$	31472
$\text{nnz}(\text{tril}(\mathbf{Y}^+))$	19005

<sup>5</sup>It is only necessary to count a single half “tril” of any symmetric systems.

***Sparsity of the factorised systems***

Table 3.10 lists the sparsity of the factorisation of  $\mathbf{A}$  obtained under different orderings. The most preferable algorithm from table 3.8, **colpermamd**, is applied to the systems  $\mathbf{A}$  and  $\mathbf{Y}^+$  to obtain orderings over (observations,states) and (states) respectively.

**Table 3.10:** Triangular Factor Sparsity

system	in order:	$\text{nnz}(\mathbf{L} \text{ of system in order } )$
$\mathbf{A}$	<b>colpermamd</b> ( $\mathbf{A}$ )	41055
$\mathbf{A}$	(obs, <b>colpermamd</b> ( $\mathbf{Y}^+$ ) )	48504
$\mathbf{A}$	( <b>colpermamd</b> ( $\mathbf{Y}^+$ ), obs )	680248
$\mathbf{Y}^+$	<b>colpermamd</b> ( $\mathbf{Y}^+$ )	19554
$\mathbf{Y}^+$ & $\mathbf{H}$ & $\mathbf{R}$		48504

The best ordering over the joint observations and states gives an  $\mathbf{L}$  factor with 41055 nonzeros. By comparison, enforcing an ordering which has the observations factorised first and using the best ordering on the remaining states results in 48504 nonzeros. This shows that the augmented system form is able to achieve sparser  $\mathbf{L}$  factors than when enforcing an “observations first” factorisation policy.

The  $\mathbf{L}$  factor of the information form has fewer nonzeros than those of the augmented system form. However, if the system needs to compute the observation Lagrange multipliers for data verification or data association purposes, then the information form factorised system is effectively subject to further nonzeros  $\text{nnz}(\mathbf{H}) + \text{nnz}(\text{tril}(\mathbf{R})) = 28950$ . This brings the information form to a total nonzero count of 48504, the same as the augmented system form under the (obs, **colpermamd**( $\mathbf{Y}^+$ ) ) ordering.

$$\begin{aligned} & \text{nnz}(\mathbf{L} \text{ of } \mathbf{A} \text{ in order } \mathbf{colpermamd}(\mathbf{A})) \\ & < \text{nnz}(\mathbf{L} \text{ of } \mathbf{A} \text{ in order } (\text{obs}, \mathbf{colpermamd}(\mathbf{Y}^+))) \end{aligned} \quad (3.106)$$

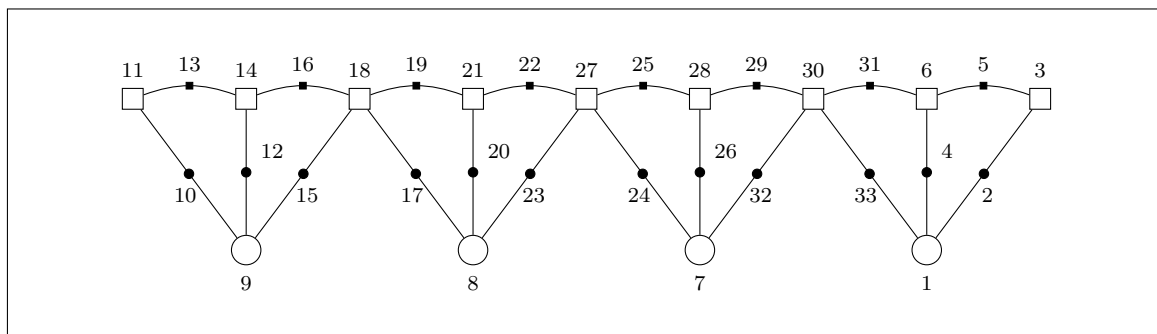
Therefore, the augmented system form is again competitive under the assumption that

the system will need to maintain the representation of the observations and calculate the observation Lagrange multipliers. This will be the case for data verification and data association algorithms.

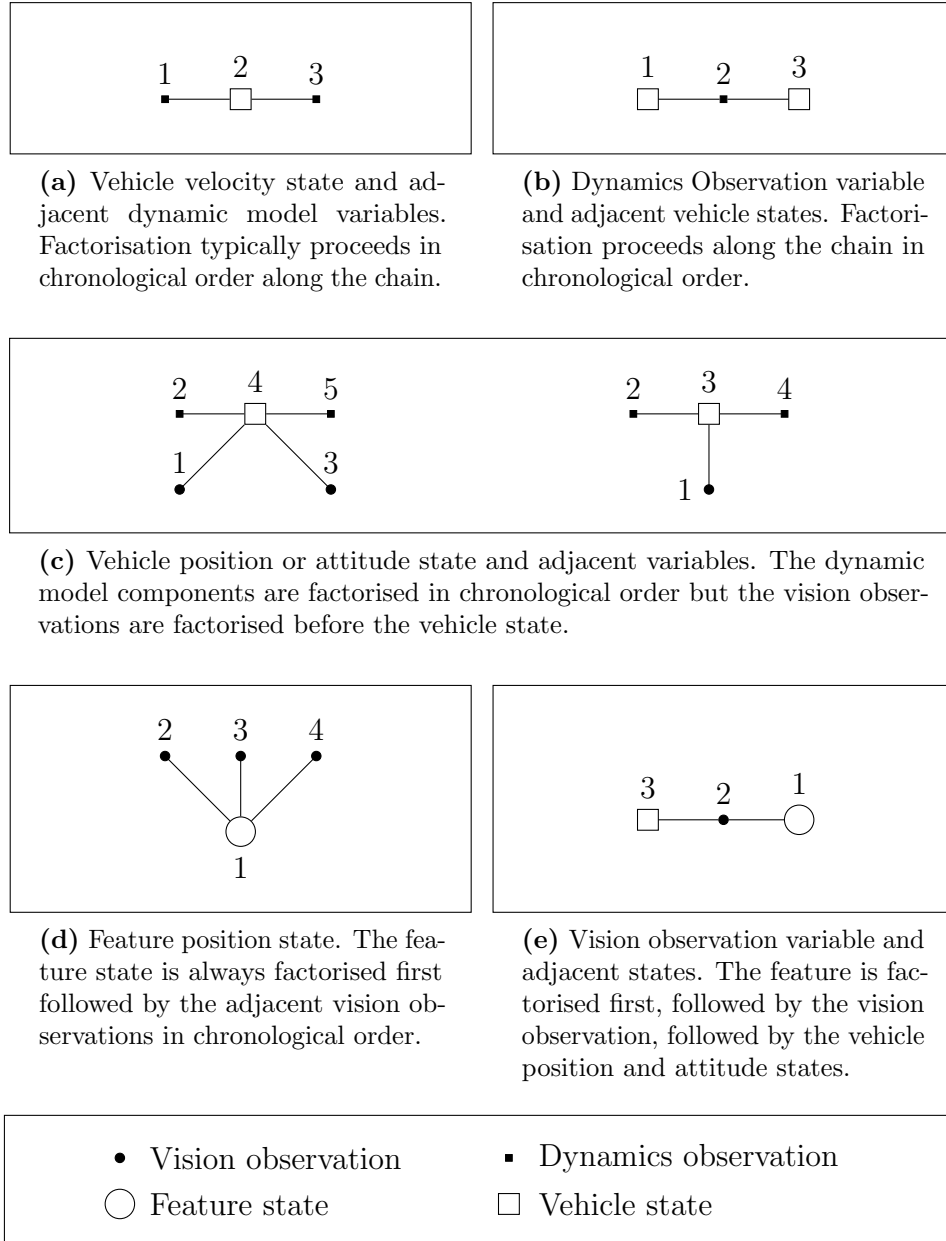
### Factorisation Order Patterns

This section describes the factorisation ordering pattern of  $\text{colpermamd}(\mathbf{A})$ . A key benefit of the augmented system form is the ability to use factorisation orderings beyond the observations-first approach of the information form. The complete factorisation order of  $\text{colpermamd}(\mathbf{A})$  for a much smaller 9 vehicle state, 4 feature state example is shown in figure 3.19. The important point is that the chosen ordering skips freely between observations and states.

Figure 3.20 decomposes the factorisation ordering by showing the relative factorisation ordering of variables immediately adjacent to a central variable. This is repeated for each type of variable in the system.



**Figure 3.19:** The specific factorisation ordering for the small (9 vehicle state) example. Notice that the factorisation ordering changes between the observations and states frequently. Vehicle states: (nodes  $\square$ )      Feature states: (nodes  $\bigcirc$ )  
Dynamic model observations: (nodes  $\blacksquare$ )      Vision observations: (nodes  $\bullet$ )



**Figure 3.20:** Typical fragments of the factorisation ordering generated by `colpermamd(A)`. Due to the regular structure of this example, these patterns occur very frequently. These patterns were taken from the full 101 vehicle state example and are not directly comparable with figure 3.19. These patterns indicate that the factorisation ordering very frequently skips between observations and states, which is a key capability of the augmented system form.

- *Figure 3.20 shows that factorisation occurs in an ordering which traverses along the chain-like segments. For example, the (vehicle state) - (dynamics observation) sequence of 3.20a and 3.20b.*
- *All cases except 3.20d factorise in an order which mixes states and observations. Only 3.20d factorises the feature state first before the observations. In this example, the feature is factorised first because it was linked with fairly small degree (3). Generally small-degree features should be factorised early and large-degree features factorised late.*
- *Neither the observations nor the states are factorised first.*

*Note that the scenarios found in this example are very regular due to the regular layout of this example. In general the system should analyse the graph structure properties and decide the factorisation ordering among the observations and states at runtime. This thesis does not recommend adopting the patterns shown in this example as fixed policies.*

*The augmented system form is necessary in order to be able to adopt these generalised factorisation orderings over the observations and states. The result is the improvement in the sparsity of the factorised system as shown in table 3.10 under the assumption that systems do need to compute the observation Lagrange multipliers. The augmented system form allows the system to use the calculated Lagrange multipliers as intermediate variables to help calculate the states faster in some cases, and vice versa in other cases, depending on the structure properties of the system.*

---

## Conclusion

This section discussed the benefits of the augmented form in relation to factorisation orderings for sparsity. This section argued that the augmented system form has benefits for the sparsity. The same performance of the information form can be obtained by eliminating the observation Lagrange multiplier variables from the augmented system. In cases of more complex observation degree, *particular states* can be factored ahead

of their observations to obtain a sparser factorisation. In general the system should analyse the graph structure properties and decide the factorisation ordering among the observations and states at runtime.

### 3.7.2 Factorisation Ordering for Numerical Stability

Numerical stability is important in systems consisting of a wide range of uncertainties. In particular, systems with constraints, observations and uninformative prior information.

The augmented system form allows the factorisation or elimination to occur over both observations and constraints. This flexibility allows numerically stable treatment of constraints and tight observations, as well as poor or uninformative prior information.

Numerical stability is affected by the elimination of variables. This is because the elimination of a variable  $i$  from a linear system  $\mathbf{A}$  forms expressions in the reduced or factorised system proportional to  $A_{ii}^{-1}$ . Small  $A_{ii}$  therefore propagate large entries into the subsequent reduced or factorised systems. Directly eliminating a zero  $A_{ii}$  corresponding to a constraint is not possible due to the resulting  $0^{-1}$ . Refer to [37, pg 239] for additional discussion.

As a linear system, the augmented form has a different numerical conditioning than the information (normal) equations. If 'high information', 'low covariance' or constraint (zero covariance) observation terms are included in a system which is otherwise well conditioned, then the augmented system form will have a better numerical conditioning than the information form. This is shown in example 3.7:

The numerical stability of the factorisation of the augmented system is affected by the factorisation ordering, allowing a choice of ordering to improve the numerical stability, as shown in example 3.8.

**Example 3.7.** 

---

***Numerical stability of the augmented system versus information form near constraints***

*Consider a system with a near-constraint observation with  $R = \epsilon$ . As  $R$  tends towards zero, the numerical conditioning of the augmented form remains near 1 (well conditioned) but the numerical conditioning of the information form tends to infinity (poorly conditioned).*

$$\mathbf{Y}^- = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad R = \epsilon \rightarrow 0 \quad \mathbf{H} = \begin{pmatrix} 1 & -1 \end{pmatrix} \quad (3.107)$$

*The associated augmented system form is:*

$$\mathbf{A} = \begin{pmatrix} R & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y}^- \end{pmatrix} = \begin{pmatrix} \epsilon & +1 & -1 \\ +1 & -1 & 0 \\ -1 & 0 & -1 \end{pmatrix} \quad (3.108)$$

*The condition number of  $\mathbf{A}$  (the ratio of the largest over smallest singular values of  $\mathbf{A}$ ) is 2.*

*The associated information form system is:*

$$\mathbf{Y}^+ = \mathbf{Y}^- + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \frac{1}{\epsilon} \begin{pmatrix} +1 & -1 \\ -1 & +1 \end{pmatrix} \quad (3.109)$$

*The condition number of  $\mathbf{Y}^+$  is approximately  $\frac{2}{\epsilon}$ .*

*Thus when high information, low covariance or constraint (zero covariance) observation terms are included in a system which is otherwise well conditioned, then the augmented system form will have a better numerical conditioning than the information form.*

---



**Example 3.8.** 

---

***Numerical stability of differing factorisation orderings near constraints***

*This example shows how the choice of factorisation order affects the numerical stability of the factorisation. In the augmented system form, it is possible to factorise the states and observations in any order and therefore possible to reorder the factorisation to improve numerical stability. This example is an extension to that given in [37, pg 161].*

*Consider the augmented system from the previous example:*

$$\mathbf{A} = \begin{pmatrix} R & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y}^- \end{pmatrix} \quad (3.110)$$

$$= \begin{pmatrix} \epsilon & 1 & -1 \\ 1 & -1 & 0 \\ -1 & 0 & -1 \end{pmatrix} \quad (3.111)$$

*The condition number of  $\mathbf{A}$  is 2. This is an invariant of the initial system,  $\mathbf{A}$ . The numerical stability of the factorisation depends on the factorisation ordering. For this, two options will be presented:*

- 1. Factorising the observation first.*
- 2. Factorising the states first.*

*Refer to section 2.4.2 for an introduction to the LDL factorisation.*

**Factorising the Observations First**

Factorising  $\mathbf{A}$  via the ordering (observation, states) results in the factors:

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{R} & 1 & 0 \\ -\frac{1}{R} & 0 & 1 \end{pmatrix} \quad (3.112)$$

$$\mathbf{D} = \begin{pmatrix} R & 0 & 0 \\ 0 & -1 - \frac{1}{R} & \frac{1}{R} \\ 0 & \frac{1}{R} & -1 - \frac{1}{R} \end{pmatrix} \quad (3.113)$$

The condition number of  $\mathbf{D}$  is  $\frac{2}{R^2}$ . Thus having factorised the observation/constraint first, the remaining system in  $\mathbf{D}$  is poorly conditioned. Both the  $\mathbf{L}$  and  $\mathbf{D}$  factors contain large entries, which tend to infinity as  $R$  tends to zero.

**Factorising the States First**

Factorising  $\mathbf{A}$  via the ordering (states, observation) results in the factors:

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \quad (3.114)$$

$$\mathbf{D} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 2 + R \end{pmatrix} \quad (3.115)$$

The condition number of  $\mathbf{D}$  is  $2 + R$ . Thus the remaining un-factorised system in  $\mathbf{D}$  is as well conditioned as the original system. The entries in  $\mathbf{L}$  are also well contained within the range  $[-1, 1]$ . These values will remain stable as  $R$  tends to 0.

---

**Conclusion**

This section 3.7.2 discussed some simple cases illustrating that the augmented system form has benefits for the numerical stability, allowing the use of constraints and tight-observations. The benefit of the augmented system form is in the ability to choose to

factorise states & observations in any ordering. The ordering has an increasing effect on the numerical stability as  $\mathbf{R} \rightarrow \mathbf{0}$  for constraint or near constraint observations.

Numerical stability in the factorisation process is also discussed in section 5.7.2.1 in algorithmic terms under the topic of graph theoretic direct solving. Chapter 5 discusses cases which include both constraints and uninformative priors. In such cases, neither the state nor the observation can be eliminated first, instead they must be eliminated simultaneously (see example 5.1).

### 3.7.3 Handling Nonlinear Observations

The augmented system form aids the representation and treatment of nonlinear observations. The observations are easily re linearised since their Jacobians exist *separately* in  $\mathbf{A}$  and are easily able to be individually replaced. This occurs because the augmented system form avoids marginalising observations into the states.

In the augmented system form,  $\mathbf{A} = \begin{pmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{pmatrix}$ , the observation Jacobians,  $\mathbf{H}$ , exist separately from each other and separately from  $\mathbf{R}$  and  $\mathbf{Y}$ . The observation Jacobians also exist in  $\mathbf{A}$  without further calculations. By existing as the off-diagonal links between the observations and the states, the observation Jacobian entries define the link structure of the estimation problem. Therefore, in the augmented system form the observation Jacobians are immediately available for replacing with new values of  $\mathbf{H}$  when relinearisation is performed.

By comparison, in the information form the observations are merged into the states via the addition of  $\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}$  onto the prior information. Thus in the information form, the observations are mixed in together with each other, and with  $\mathbf{R}$  and  $\mathbf{Y}$ . Any given observation is then represented by a *clique* in the information matrix. However, without further data structures the information form offers no method to maintain or track these cliques and link back to the observations they represent. In the information form, performing relinearisation of a single observation involves subtracting  $\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}$ , reforming the new  $\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}$  and adding it back into  $\mathbf{Y}$ . This involves the same

operations as in the augmented system relinearisation, plus significant additional operations to re-form the altered information form.

### 3.7.4 Conclusion

This section discussed the benefits to the estimation process in using the augmented system form compared to using the information form. The augmented system form improves the sparsity for problems with large observation degree and small state degree and improves numerical stability for constraints and tight-observations. These are achieved by having the option of factorising states and observations in a mixed ordering.

## 3.8 Future Research

This chapter showed the benefits of the augmented system form in the ability to choose factorisation orderings which mix between observations and states in order to offer improved sparsity and numerical stability. For the factorisation ordering for sparsity, the comparisons were drawn using the algorithm `colpermamd` (`colperm` followed by `colamd` [18]). An algorithm for choosing a factorisation ordering for numerical stability is given in chapter 5.

However, the factorisation ordering is still an important problem for future research. It remains a problem to incorporate both sparsity and numerical stability concerns in the factorisation ordering. In addition, another concern in the factorisation ordering is the ordering for *online modification*. These are discussed further for future research in section 6.2.

## 3.9 Chapter Conclusion

This chapter presented the augmented system form, a generalisation of the information form consisting of augmenting observations & constraints in addition to the states.

The augmented form provides a mathematical system showing explicitly and distinctly the states and observations & constraints together with Lagrange multipliers for their interaction. The augmented system form was shown to be more general than the information form, and this thesis proposed that it therefore provides a more general starting point for the formulation and solving process. The information form is able to be recovered by eliminating the observations first, in a manner which is required for the formation of the information form anyway. By forming the augmented system first, the process of forming the information form is formalised and is performed using direct solving structures and methods. Furthermore, new alternative solving approaches can be realised by factorising variables in a more flexible order than the fixed observations-first approach. This is strictly required for constraints and also improves numerical stability under small  $\mathbf{R}$  and improves the fill-in sparsity under high observation-degree circumstances.

For complex, large scale problems with various mixes of structural properties, the best available orderings for sparsity perform the factorisation with an ordering that mixes between observations and states.

It is not recommended to adopt any fixed policy of marginalisation of variables. Instead, this thesis proposes using the augmented system form as the initial formulation of the estimation problem, capturing the structure of the states and observations in their full form. The system can subsequently be subject to analysis at runtime, given the structural and numerical properties and understanding which variables are *required* in the solution (as opposed to variables which are only intermediate variables required to compute others) to determine which variables can or should be factorised and in which order. The abilities of the augmented system form to support factorising among observations and states complements the abilities of the trajectory state form (in the smoothing and mapping and viewpoint SLAM frameworks), which support choosing good factorisation orderings among the vehicle and feature states.

A novel Lagrangian was introduced and shown to generalise both the quadratic objective function over the states and a quadratic relating to the innovation Mahalanobis distance. The related residual Mahalanobis distance was introduced and shown to

offer extensions for more complex multi-term cases than the conventional innovation Mahalanobis distance.

This chapter derived connections between the proposed augmented system form and a mix of analytical expressions related to estimation problems: The augmented system Lagrangian, the objective function quadratic and the problem Mahalanobis distance are all closely related. Given these, this chapter contributed a novel form of Mahalanobis distance which is equivalent to the conventional innovation distance but offers additional generality, including the ability to operate with rank deficient information terms.

The next chapter describes a graph representation for the formulation of the estimation problem. The augmented system form and the graph representation are complementary to each other, since both discuss ways of representing the sparse, structured system of variables involved. The augmented form provides the mathematical system, whereas the the graph structure of the next chapter is a data structure for describing inter-relations of variables generally.

# Chapter 4

## Graph Theoretic Representation

### 4.1 Introduction

This chapter contributes a novel graph based representation for the sparse structure of variables, their graph links and their associated sparse linear systems.

The previous chapter proposed the use of the augmented system form for estimation problems in localisation and mapping. The augmented system forms a large sparse network of state and observation Lagrange multiplier variables for the estimation problem formulation.

Given this graph-theoretic nature of the formulation approach, the motivation for this chapter was to develop an entirely graph based representation for the system of variables and their links. While this seems intuitive, the typical approach is to use a sparse *matrix* representation. Unlike sparse *matrix* representations, the representation proposed in this chapter is a true *graph*; it offers benefits such as constant time insertion and removal of variables, and constant time access to adjacent variables. The proposed graph based representation is illustrated in figures 4.1 and 4.2.

Beyond the proposal to use a graph based representation, this thesis contributes a novel graph representation which is suited to sparse symmetric and directed systems,

and the associated linear sparse direct solver which operates in this representation (see the next chapter).

When discussing systems of variables involved in the nonlinear localisation and mapping problem, recall from chapter 2 how such nonlinear systems descend into *linear* problems. The graph representation described in this chapter is capable of representing such a nonlinear system. Since the system augments the observations and their necessary past and present states, the overall nonlinear system is represented by each observation's type and its particular nonlinear observation function. The system does not attempt to amortise the *functional* nonlinear representation into any *manipulable* nonlinear representation. As described in chapter 2, this thesis uses only a *local quadratic* approximation. In turn, this local quadratic is represented by the equation for the zero-gradient solution, which has the form of a linear system. Therefore, the representation described in this thesis focuses on storing the variables, their nonlinear functions, the *linearised* functions and finally the overall *sparse linear systems* involved in their solution algorithms.

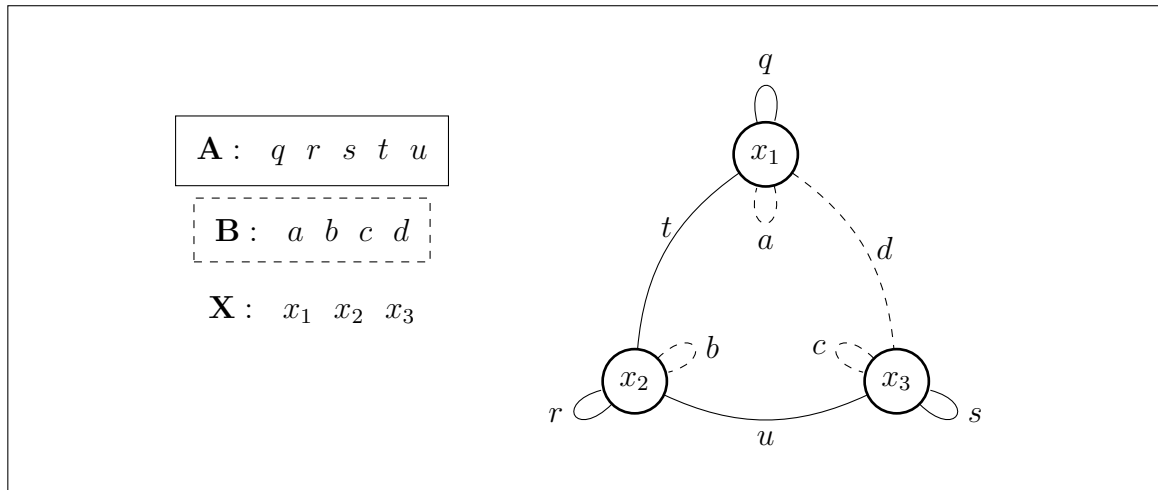
This graph structure approach allows the development of graph embedded solving methods of the next chapter. The graph structure allows the structure of the problem to be exploited by the solution methods, since the full graph structure is available. The conditional independence and sparsity properties exploited by some estimation algorithms are graph-theoretic properties which are available to the system at runtime when formulated as a graph. Such conditional independence properties include, for example, the Markov property of dynamic systems (a dynamic state is conditionally independent of its whole past history, given the previous state). By encoding these sparsity and conditional independence structures in the representation, solution algorithms can exploit them where applicable.

The graph based representation of linear systems is described in section 4.3. This graph based representation of linear systems includes the ability to store and manipulate multiple vectors and sparse matrices and is used in both the theory and runtime operation of the methods presented in this thesis. The graph operates as both the data structure and framework for the solving algorithms described in the next chapter.



$$\mathbf{A} = \begin{bmatrix} q & t \\ t & r & u \\ & u & s \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} a & d \\ & b \\ d & c \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

(a) Example symmetric linear systems  $\mathbf{A}$  and  $\mathbf{B}$  over variables  $\mathbf{X}$  in *matrix* form.

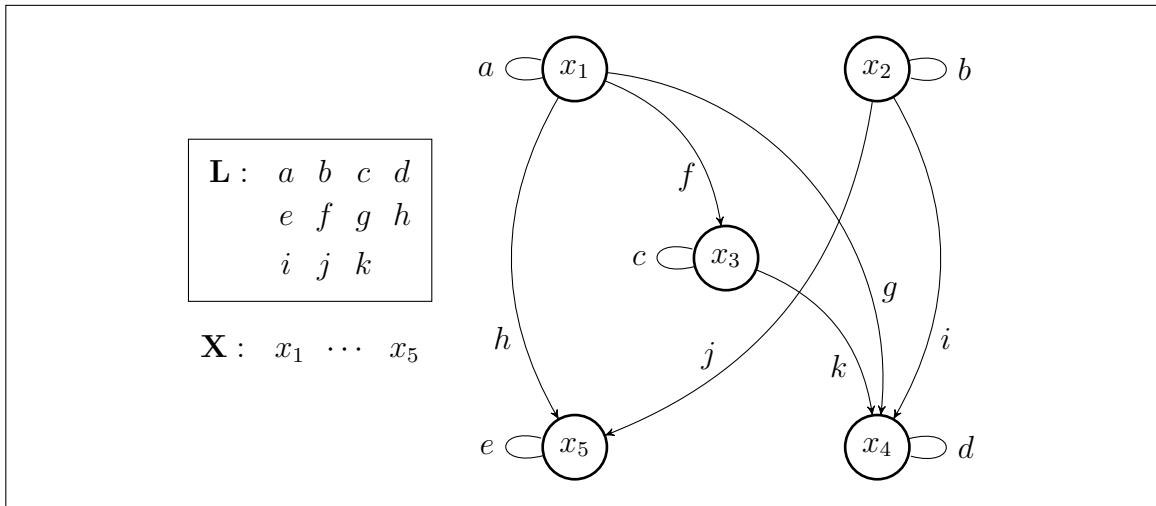


(b) Equivalent symmetric linear systems  $\mathbf{A}$  and  $\mathbf{B}$  from (a) in *graph* form. The variables of  $\mathbf{X}$  are represented by graph vertices in no particular order. Linear systems  $\mathbf{A}$  and  $\mathbf{B}$  are represented by graph edges and loops. Linear system  $\mathbf{A}$  is shown in solid edges (—). Linear system  $\mathbf{B}$  is shown in dashed edges (---). An important aspect of the representation is that multiple matrices are represented on a single graph by distinct sets of edge objects (edge-sets).

**Figure 4.1:** Graph representation of symmetric linear systems. Example symmetric linear systems  $\mathbf{A}$  and  $\mathbf{B}$  are shown in both matrix and graph forms.

$$\mathbf{L} = \begin{bmatrix} a & & & & \\ & b & & & \\ f & & c & & \\ g & i & k & d & \\ h & j & & & e \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

(a) An example triangular square linear system,  $\mathbf{L}$  over  $\mathbf{x}_1$  to  $\mathbf{x}_5$ , in *matrix* form.



(b) The example triangular square linear system,  $\mathbf{L}$  over  $\mathbf{x}_1$  to  $\mathbf{x}_5$ , in *graph* form.  $\mathbf{L}$  is the set of loops and directed edges. The directed edges are *acyclic*.

**Figure 4.2:** An example triangular square linear system  $\mathbf{L}$  shown in both matrix and graph forms.

This chapter bridges the gap between general purpose graph structures and sparse matrix structures. The graph representation, described in section 4.4, combines novel elements beyond existing matrix and graph representations. The proposed graph representation distinguishes between loops, symmetric and directed edges and has the ability to contain multiple *edge-sets* representing multiple matrix systems. These innovations are motivated by the need to represent both symmetric and triangular linear systems for the representation and factorisation of systems arising in estimation problems.

To give clarity to the concepts and contribution of the graph based representation, the implementation is described in section 4.5. The graph representations are compared to alternative methods in section 4.6. These numerical tests show the improved efficiency of the graph based representations for insertions and traversals, highlighting the differences of this representation against standard methods. Future directions for research in the graph and linear system representation are described in section 4.7.

## 4.2 Literature

Graph based methods have had an ongoing presence in the localisation and mapping literature. However, this thesis proposes a significantly expanded role for graph representations of the estimation variables.

This thesis proposes the use of an *explicit* graph based data structure implementation. Other references in the field propose *graph based methods* to explain the approach of augmenting trajectory states and explain the elimination of variables in that context. For example, the graphSLAM system [69, 71] describes an approach in which the vehicle poses and feature locations exist as vertices in a graph. However, it appears that they utilise a matrix implementation, despite the use of graph-based terminology. For example, their description of the incorporation of a measurement refers to the process of splitting a  $5 \times 5$  information matrix block into blocks for the pose and feature entries. Such issues indicate a matrix based implementation and such issues do not exist in the graph representation proposed in this thesis. In another case, their

description of the elimination of features follows the process and terminology expected of a matrix based implementation: maintaining the indices of variables, accessing submatrices, and removing rows and columns after elimination of variables.

By contrast, the methods proposed in this chapter and the next constitute an entirely graph based representation and associated solving algorithm for such sparse linear systems. The resulting graph based representation avoids the above inconvenient complications of matrix based representations.

In the smoothing and mapping approach (SAM) [20], the *factor graph* is shown as an appropriate representation for the states and variables. However, the data structure behind [20] appears to be the compressed-sparse-column matrix representation required in order to utilise the library algorithms `colamd` [18] and `LDL` [16].

Other references have stated their use of an explicit graph structure, but have not elaborated significant details. In [28], the authors describe a graph based representation for the vehicle trajectory and map states in SLAM. As is similar to the approach described in this thesis, [28] states that “[The graph representation] will be easier to work with than matrices and long state vectors” and “the edges represent the non-zero components of the information matrix”. However, [28] does not focus on the graph representation of the variables as a true alternative to sparse matrix representations and does not provide further details. This thesis proposes the graph representation as a true alternative to a sparse matrix representation and extends novel graph theoretic structures based on the requirements for use in linear algebra.

The remainder of this section considers the literature beyond the field of estimation and considers sparse linear systems generally. The topic of connections between graphs and linear systems has a vast literature and it is beyond the scope of this thesis to present a full review. Graph-theoretic methods are the dominant methods applied for the *analysis* of sparse matrices and direct solving algorithms [31]. Early references apply graph theory to the analysis of Gaussian elimination [55] and matrix inversion [39]. However, this usage of graphs in the *analysis* does not appear to extend into the actual implementation and runtime operation of linear system manipulations as is

proposed in this thesis <sup>1</sup>.

On the other hand, in the field of graph theory, matrices are applied for the analysis of graph-theoretic problems [11]. It is common to see graph theory practitioners describing the storage and manipulation of graphs in a *matrix* format (especially for spectral analysis)[11]. This thesis adopts the *opposite* approach: instead of analysing graphs using matrices, this thesis operates on linear systems using graphs.

Graph algorithms involved in sparse matrix algorithms frequently use the very same compressed-sparse-column (CSC) matrix representation (for example: [17, 18, 43]) to ensure in-memory compatibility. However, these dense integer index structures lack the same complexity properties required of a graph representation, especially constant time insertions.

The graph embedded linear system described in this chapter assigns the matrix elements to the graph edges between the variables. Given the intuitive basis for this and the long history of graph theory and linear algebra, it is surprising that few papers or available software systems use a graph based linear system. A rare exception is in [68], which comments that the matrix entry  $m_{ij}$  is stored on a graph edge  $i \rightarrow j$ , as in the scheme proposed in this thesis. Tarjan comments that “we consider the system of equations defined graph-theoretically in this way”.

However, at present, commonly available linear system software does not use graph based data structures, but instead use the “compressed sparse column” (CSC) format (or the row oriented transposed equivalent, CSR), for example: [1, 2, 4, 16, 18, 22, 33, 60]. The Bayes-net toolbox [51] represents graphical models in Matlab using integer indexing of vertices, with the adjacency matrix in CSC sparse matrix format, as opposed to the object-access and pointer direct graph approach as proposed in this thesis. The proposed graph representation is compared to the CSC in section 4.6.

The distinction in data structures between matrices and graphs is important because of the strong relationship between data structures and algorithms. When so much

---

<sup>1</sup> Graphs are not typically used at runtime because it is preferable to use the CSC format for fixed-size, pre-analysed problems.

of the matrix analysis is described in a graph terminology it makes sense to have a graph representation in software.

In conclusion, while many authors have adopted graph-theoretic analysis methods or used graph-theoretic methods on matrix or matrix-like representations, the motivation for this chapter was to contribute an entirely graph embedded representation and solving system for sparse linear systems generally and for estimation in localisation and mapping.

### 4.3 Graph Representation of Linear Systems

This section shows how linear systems can be represented in a graph based form. This section describes specific constructs such as vector and matrix entries, whole vectors and matrices, and finally sets of vectors and matrices. Each is generally intuitive but there are subtle differences compared to conventional methods, which affects algorithms later on. These points are contributed by this chapter as important new capabilities that a graph based representation of linear systems fundamentally requires. In turn, these lead to the graph structure extensions described in section 4.4.

Figures 4.1 to 4.3 are initial illustrations of the application of a graph structure to linear systems. Figure 4.1 shows two example linear systems in both matrix and graph representations. In figure 4.1a the systems are shown in matrix form, in which entries are associated with row and column indices. By comparison, in figure 4.1b entries exist as graph vertices with no particular ordering and are linked by explicit graph edges. Two families of edges (edge-sets) represent the two linear systems. This representation allows multiple systems to refer to the same underlying set of variables in a separate but tightly linked manner.

Figure 4.2 shows the case of a *triangular* linear system in both matrix and graph representations. The graph representation forms a *directed acyclic graph*.

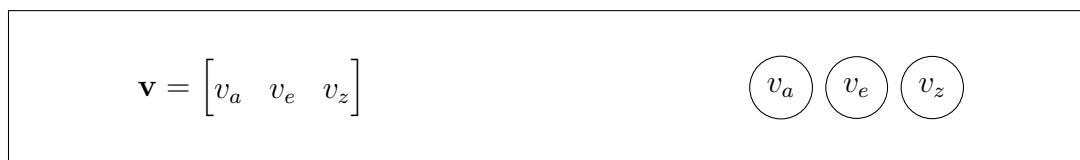
Figure 4.3 shows a linear system,  $\mathbf{A}$ , together with three alternative graph based representations. The matrix representations for the three alternatives are identical,

resulting in an ambiguity. The graph representation alternatives are: (self-referencing symmetric), (self-referencing unsymmetric) and bipartite.

The above figures (4.1 to 4.3) generally illustrate the graph representations of linear algebraic constructs. The following subsections (4.3.1 to 4.3.4) explain these representations in greater detail.

### 4.3.1 Dense Vectors

- The graph-theoretic equivalent of a dense vector,  $\mathbf{v}$ , is to store vector entries within each vertex, (see Figure 4.4). To refer to the vector,  $\mathbf{v}$ , requires referring to the offset of the vector entries within each vertex. Storing the vector entries in each vertex results in dense storage of the vector.



**Figure 4.4:** Matrix (left) and graph (right) equivalents for a dense vector.

- For a set of vectors,  $\mathbf{v}_a$  through to  $\mathbf{v}_c$  each of length  $n$ , the graph-theoretic equivalent is to associate scalars  $a$  through to  $c$  with each of  $n$  vertices. In the graph-theoretic arrangement, the association of vector entries to the underlying objects is explicit. In the conventional matrix-vector scheme, the association of vector entries to integer indices is explicit and the association of integer indices to underlying objects is only *indirectly implied* by common integer indexing. The vector-oriented scheme (of conventional matrix and vector approaches) is more flexible for adding and removing whole vectors but less flexible for adding & removing individual *objects*. The vertex oriented scheme (of the representation proposed here<sup>2</sup>) is very flexible for adding new objects but inflexible for adding new vectors. (See figure 4.5).

The vertex-oriented scheme is appropriate for the applications motivating this

---

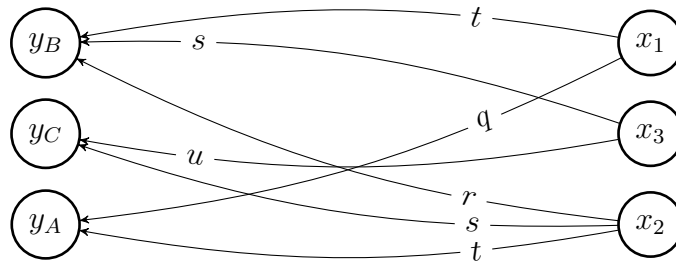
<sup>2</sup>also known as an object-oriented approach

$$\mathbf{A} = \begin{bmatrix} q & t & \\ t & r & u \\ & u & s \end{bmatrix}$$

(a) Example linear systems  $\mathbf{A}$  in *matrix* form. The system  $\mathbf{A}$  does not link explicitly to any vector or objects. The symmetry and squareness of  $\mathbf{A}$  are not guaranteed and  $\mathbf{A}$  could be a nonsymmetric rectangular system.



(b) Equivalent linear system  $\mathbf{A}$  from (a) in *graph* form, indicating that the system is self-referencing, making  $\mathbf{A}$  fundamentally *square*. (left)  $\mathbf{A}$  is not necessarily symmetric. The edges are reprinted twice, indicating that the symmetry is not fundamental. (right)  $\mathbf{A}$  is interpreted as an inherently symmetric operator from the objects of the vertices back onto the same set of objects. Having a single undirected edge for symmetrical pairs saves space but also indicates the strong intent of the symmetric relationship.

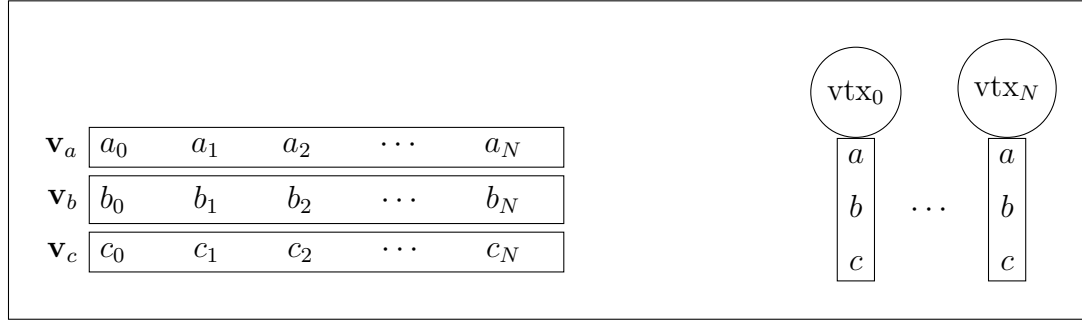


(c) Equivalent linear system  $\mathbf{A}$  from (a) in *graph* form, indicating that the system is an operator which refers one set of objects onto another distinct set of objects, thus making  $\mathbf{A}$  fundamentally *rectangular*. Symmetry has no significance because the labelling of objects is arbitrary. The graph representation shows a bipartite graph linking the two sets of objects.

**Figure 4.3:** Squareness and symmetry ambiguity of matrices resolved in the graph form. Example linear system  $\mathbf{A}$  is shown in matrix form together with a range of graph forms which are different interpretations of the system  $\mathbf{A}$  relating to squareness and symmetry.



thesis. The ability to add new objects is critical in the ability to extend states and observations in online localisation and mapping. The inflexibility in adding new entire vectors is not significant, since the set of vectors used is usually known when designing algorithms, or known at compile time.

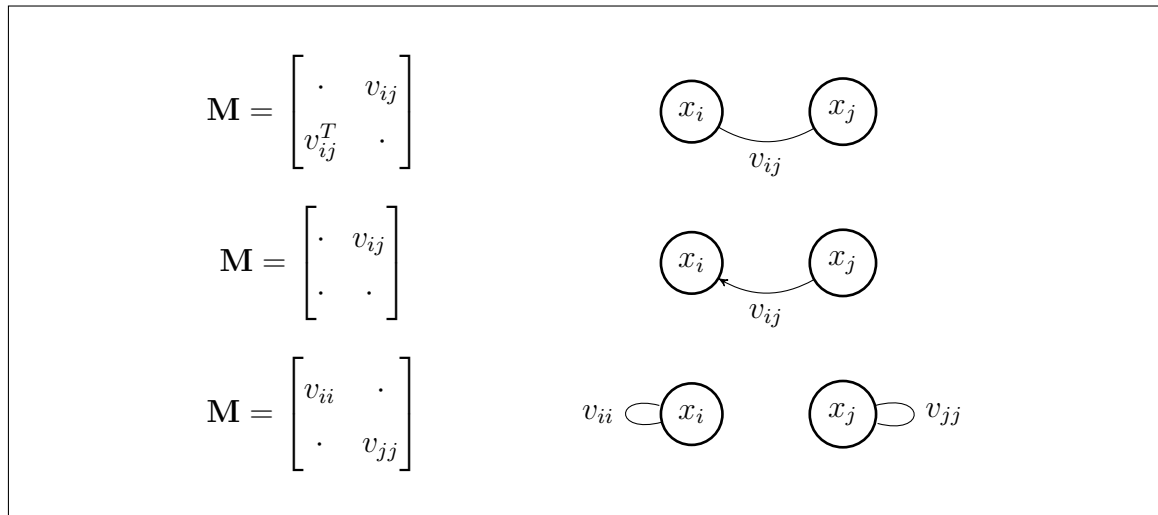


**Figure 4.5:** (left): a vector-oriented scheme. The data are grouped by belonging to particular vectors and the common relation to underlying objects is implicit in the use of common integer indices. (right): An object and vertex oriented scheme. The data are explicitly associated with particular objects, each containing the data for several vectors.

### 4.3.2 Matrix Entries

Figure 4.6 illustrates the embedding of symmetric, unsymmetric and diagonal scalars of a linear system in a graph representation.

- The graph-theoretic equivalent of a matrix entry at  $(i, j)$  relating variables  $i$  and  $j$  is a number associated with a particular graph edge connecting the vertices representing variables  $i$  and  $j$ .
- The graph-theoretic equivalent of a pair of symmetric matrix entries is an undirected graph edge.
- The graph-theoretic equivalent of a single non-symmetric matrix entry at  $(i, j)$  is a directed graph edge from vertex  $j$  to vertex  $i$ .
- The graph-theoretic equivalent of a diagonal matrix entry of a symmetric matrix is a graph loop.



**Figure 4.6:** Matrix and graph equivalents for scalar matrix entries, for symmetric (undirected), unsymmetric (directed) and diagonal entries.

The above points concern individual scalar matrix entries. The following points concern whole and multiple matrices.

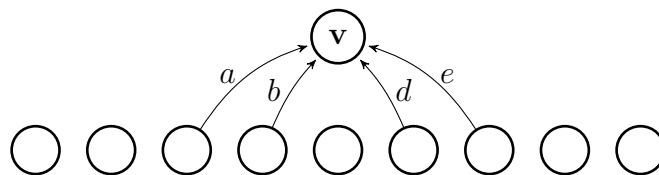
- The graph-theoretic equivalent of an entire matrix is a *set of edges*. For example, in figure 4.1 all edges  $q$  to  $u$  represent system  $\mathbf{A}$ .
- For multiple distinct matrices over a single set of variables, the graph-theoretic equivalent is multiple distinct *sets of edges* over the single set of vertices. Figure 4.1 illustrates this, showing two distinct matrices over a single set of variables in matrix and graph based forms.
- The existence of multiple distinct edge-sets allows the representation of *layers* of edges and matrix entries, since each edge-set retains a separate identity.
- This identification of the need for multiple *edge-sets* is a contribution of this thesis. It helps enable the graph representation as an alternative to a matrix and vector approach for sparse linear systems and variables.

### 4.3.3 Sparse Vectors

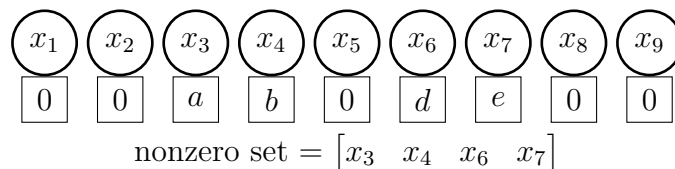
A representation of sparse vectors can be obtained from the structure used to represent matrix rows or columns (see Figure 4.7b). In this manner, only the nonzero entries are stored. This representation obtains the same properties as the matrix representation, for example, algorithms can access the set of nonzero entries by accessing the appropriate in or out edges of the common vertex. This representation is used when algorithms operate on matrix rows or columns as vectors.

$$\mathbf{v} = [0 \ 0 \ a \ b \ 0 \ d \ e \ 0 \ 0]$$

(a) An example sparse vector for use in the figures below



(b) Sparse vector representation using graph edges. This representation is exactly the representation of a matrix row or column and can be used when matrix rows or columns are interpreted as vectors. The arrow directions shown imply that the vector is equivalent to a matrix row. A vector equivalent to a matrix column would have the reverse directions to those shown here.

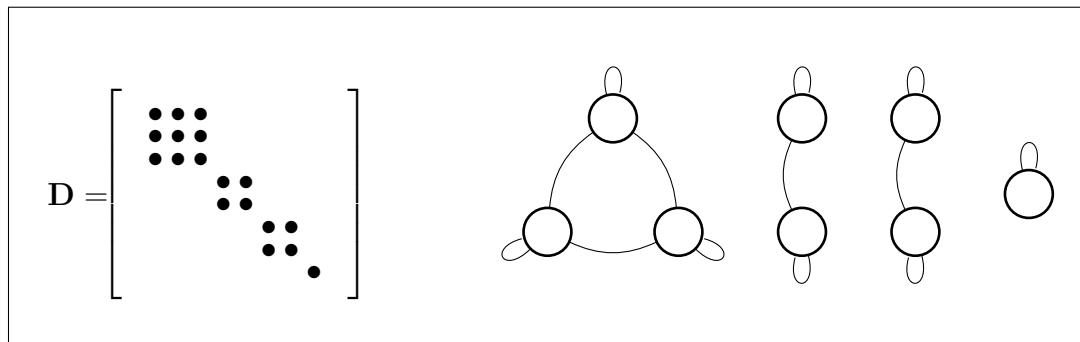


(c) Pseudo-sparse vector representation using (dense) vector storage and a set indicating vertex pointers to the nonzero entries.

**Figure 4.7:** Sparse vector representation using graph edges, analogous to a matrix row or column.

### 4.3.4 Matrix Categories

- For a graph edge-set to be equivalent to a triangular matrix, the graph edge-set must consist of only directed edges and be acyclic. This is illustrated in Figure 4.2. This is also illustrated ahead in figure 5.4. This is shown by considering the variables in *topological order* as defined by the directed acyclic graph edges. In the topological order, each variable may have links *in from* any variable *earlier* in the topological ordering, and each variable may have links *out to* any other variable *later* in the topological ordering. In a matrix representation for the linear system, for each variable  $i$ , the *input* coefficients from the other variables lie on the same *row* as  $i$ , the *output* coefficients to the other variables lie on the same *column* as  $i$ . The result is that triangular linear systems are equivalent to directed acyclic graphs (providing the linear system is stored in topological order).
- For a graph edge-set to be equivalent to a diagonal matrix, the graph edge-set must consist only of loops, since each variable only links to itself.
- For a graph edge-set to be equivalent to a block diagonal matrix, the vertices must form separated subgraphs in the same pattern as the matrix diagonal blocks, as shown in figure 4.8.



**Figure 4.8:** Matrix and graph equivalents for a block diagonal matrix. In the matrix approach (left), each block is represented by a contiguous block of consecutively indexed entries in the matrix, lacking off-diagonal entries into any of the other blocks. In the graph approach (right), each block is represented by a separated subgraph, lacking edges into any of the other subgraph blocks.

### 4.3.5 Discussion

#### Self-Referring versus Bipartite

The proposed graph based representation has a *tight binding* of linear system entries to *objects*. Vector entries are inherently bound to particular objects in memory, not simply integer indexes. Similarly, the graph-edge “matrix” entries link to pairs of objects rather than pairs of indices. This *object* based accessing semantics is different from conventional matrix and vector *integer indexing* semantics. This difference has a subtle but important effect on how systems are represented.

Figure 4.3 shows an example linear system,  $\mathbf{A}$ . Suppose  $\mathbf{A}$  is used as an operator:  $\mathbf{B} = \mathbf{A}\mathbf{X}$ . In a matrix and vector oriented scheme,  $\mathbf{B}$  and  $\mathbf{X}$  are separately stored and may or may not have any inherent common relation to each other:

- For example:  $\mathbf{B}$  and  $\mathbf{X}$  might have different sizes,  $\mathbf{A}$  might be rectangular and entries  $\mathbf{B}_{(i)}$  need not be a property of the same object  $i$  as entries  $X_i$ . This interpretation can still hold *even if* the sizes are (coincidentally) the same. This interpretation is referred to as the *bipartite* interpretation. It assumes that

$\mathbf{X}$  and  $\mathbf{Y}$  are distinct objects. An example of this interpretation is a set of observations,  $\mathbf{z}$ , and states,  $\mathbf{x}$ , linked by an observation Jacobian matrix,  $\mathbf{H}$ .  $\mathbf{H}$  is inherently rectangular since observations are distinct from states, even if their size is the same. Observation  $i$  need not correspond physically with state  $i$ .

- Alternatively:  $\mathbf{B}$  and  $\mathbf{X}$  might be exactly the same size and correspond to different properties of some underlying vector of objects.  $\mathbf{B}_i$  and  $\mathbf{X}_i$  would refer to properties of object  $i$ . In this interpretation the sizes of the vectors *must inherently* be the same. This is referred to as the *self-referring* interpretation.

For example, given a set of states  $\mathbf{x}$ , the information gradients  $\mathbf{y}$  are obtained as  $\mathbf{y} = \mathbf{Y}\mathbf{x}$ . Entries  $y_i$  belong to the same object  $i$  as entries  $x_i$ .  $\mathbf{Y}$  is fundamentally square because of this.

In the graph based representation, however, there is no ambiguity about the interpretation, due to the binding of entries to objects. Figure 4.3b shows the self-referencing interpretation of  $\mathbf{A}$ , which itself may be interpreted further as being inherently symmetric (right) or simply numerically symmetric (left). Figure 4.3c shows the bipartite interpretation of  $\mathbf{A}$ , which has a significantly different structure than 4.3b.

This section showed how linear systems can be represented in a graph based form. This section described specific constructs such as vector and matrix entries, whole vectors and matrices and finally sets of vectors and matrices. These are mathematical properties noted by this chapter but independent of the specific structure proposed by this thesis. The following structure describes the features of the graph representation proposed in this chapter. Section 4.5 describes more specifically how this representation is formed.

## 4.4 Graph Representation

This section describes the novel graph representation developed for this thesis. The graph representation presented in this thesis distinguishes between edges and loops,

maintains multiple edge-sets and allows both symmetric and directed edges. These innovations are specifically motivated by the requirements for representing and operating with sparse linear systems, particularly symmetric linear systems and their factorisations. These graph representation extensions are part of the contribution of the graph based linear system representation.

#### 4.4.1 Edges and Loops

The graph representation presented in this thesis distinguishes between edges and loops. Loops are edges in which both ends of the edge refer to the same vertex. Loops are the graph-theoretic equivalent of a square, symmetric matrix's diagonal entries. Diagonal entries are important in the context of symmetric systems since they identify a matrix's reference to a *self* matrix entry for each variable.

In matrix algorithms, loops play a different role than general edges and it is possible for algorithms to know in advance when a given element will be a loop or a general edge. Matrix algorithms frequently need to access the diagonals of a particular matrix, for a particular variable. It is important for a vertex to have constant time access to its own loops, irrespective of other edges. It is therefore important to separately store the loops for each vertex from the other edges of each vertex. It is also convenient to store the overall list of loops of the graph separately from the overall list of other edges of the graph. These claims will be described more specifically in relation to the LDL factorisation and solve, in chapter 5.

Loops are important given the absence of row and column indexing in the graph based representation. Usually matrix algorithms can identify diagonal elements in the obvious way, checking row and column indices.

#### 4.4.2 Symmetric and Directed Edges

The need for representing matrix systems in a graph representation motivates a requirement for both symmetric (undirected) and directed edges.

Figure 4.1b illustrates a symmetric linear system represented with symmetric edges (and loops). Figure 4.2b illustrates a directed acyclic system represented with directed edges. Figure 4.3b illustrates the distinction between using a single symmetric edge versus using a pair of directed edges.

In figure 4.3b (left) the system  $\mathbf{A}$  is *not necessarily* symmetric - only the numerical values  $t$  and  $u$  happen to coincide for both directions of the directed edges. The repetition of the edge values indicates that the symmetry is not fundamental to the system. If one of the repeated directed edges were removed, the resulting structure could not be distinguished from a *directed* system.

In figure 4.3b (right) the system  $\mathbf{A}$  is *inherently* symmetric. The single, undirected edges clearly indicate the intent for the system to represent a symmetric system.

Symmetric edges encode the mathematical concept of symmetry, which is significantly important for the numerical properties of the linear system. Symmetric edges also reduce storage size by allowing entries to be stored only once, as is the case with symmetric matrix representations, without introducing ambiguity with similar *triangular* systems.

Directed edges arise out of factorisations representing their output as directed acyclic graphs. Directed edges imply a directional properties such as input-output, upstream and downstream on the vertices. Directed and acyclic edges imply a topological ordering on the vertices. These properties are not applicable for symmetric edges.

It is important to be able to represent both directed and symmetric edges, since both can occur simultaneously. In particular during factorisation it is important to represent both the symmetric nature of the un-factorised part and the directed (and acyclic) nature of the factorised part.

Existing graph representations can not simultaneously represent both symmetric and directed edges. This thesis contributes an extended graph representation which does allow the graph to contain both symmetric and directed edges.



### 4.4.3 Multiple Edge Sets

The graph-theoretic equivalent of a matrix is a *set of edges* (see Section 4.3). The graph system proposed in this thesis allows for the representation of multiple edge-sets thus allowing the representation of multiple linear systems over the same set of variables. The multiple edge-sets are used like computer data *registers* in order to hold various sparse linear systems at various stages of the algorithms.

This can be interpreted as allowing multiple distinct *layers* of edges and loops in the graph. Each layer can be viewed as a graph in itself. This may also be interpreted as a series of graphs, together with a tight coupling of the vertices between the various layers.

In a matrix and vector scheme, vectors are stored as a mapping from a single integer index to variables. Matrices are stored as a mapping from pairs of integer indices to matrix values. The common use of integer indices provides the linking between matrix and vector entries. Multiple matrices are created independently and have no common link other than identical integer indices.

In the graph representation proposed in this thesis, the matrix edges are tightly bound to the vector variables by edge connectivity. Without any other disambiguation, a double edge between variables would represent the summation of their edge matrix entries. Therefore the representation of multiple matrices in the graph representation is achieved by separately storing multiple sets of edges.

The graph contains an array of containers for the graph's list of edges and loops. Each vertex also contains an array of containers for it's own edges and loops. All the graph operations on the vertex set (such as iterating over all edges, accessing adjacent vertices and vertex in/out degree) are therefore required to refer to a particular *edge-set* number.

Figure 4.1 shows a simple example where multiple matrices refer to a single set of variables. In graph form, as shown in Figure 4.1b, the representation of multiple matrices is achieved via multiple edge-sets.

The separate storage of multiple edge-sets allows algorithms to access the edges of a particular edge-set independently of the number of edges in other edge-sets.

#### 4.4.4 Discussion and Conclusion

In the design presented in the thesis, it is important that the vertices are *code objects* which are only represented *once*. Therefore the various capabilities of the graph must be represented on the one set of vertices. This has several consequences:

- Vertices are involved in multiple types of linear systems (eg: the original system  $\mathbf{A}$ , its factorisation  $\mathbf{L}$  and  $\mathbf{D}$ ). It is therefore important to be able to represent both directed and symmetric systems.
- Vertices are involved in multiple separate linear systems. Therefore it is important to have multiple edge-sets per graph rather than using multiple graphs.

The intent of this is to provide a greater level of detail regarding the intent of matrix entries. In the graph we know whether an entry is intended to be directed, symmetric (undirected) or a loop. By contrast, in the matrix approaches we only have “entry  $A_{ij}$  at  $(i, j)$ ” which leaves such an entry ambiguous about whether it is a directed, symmetric or loop entry.

Arguably, one could use a pair of directed edges to represent a symmetric/undirected relation; This is unnecessarily redundant. One could also use a single directed edge to represent a loop but loops are special in such a way that it is helpful to know exactly when the code is operating on loops and when on general edges.

An example showing a mixed use of symmetric and directed edges is shown in [51]. Mixed directed and symmetric edges also occur midway through LDL factorisation; Symmetric edges are gradually moved over to directed edges while referring to the same vertex objects.

This section has presented the novel contributions of this thesis relating to the underlying graph structure. All of these innovations are essentially graph-theoretic,

relating generally to vertices and edges and could be applied in any graph application, but these innovations are motivated by matrix based concepts, especially those for symmetric linear systems.

A practical realisation of this representation is presented in the next section.

## 4.5 Graph Representation Implementation

This section presents the practical implementation of the graph representation from the above sections 4.3 to 4.4. The purpose of this section is to show explicitly the structures used in the discussion above, in order to clarify their properties, and to present some of the design decisions and approaches undertaken in preparing this structure.

The graph representation described in section 4.4 is designed for use in representing multiple sparse linear systems. This requires extensions to the underlying graph representation beyond conventional representations such as in [63, 65].

This graph representation was initially based on the *interface concepts* in [65]. Some compatibility with the interface concepts of [65] could be obtained by casting a single edge-set “view” as meeting the interface of [65]. However, even within a single edge-set the representation here has extensions beyond the generic interface in [65]. In particular: the distinction between loops and edges; the simultaneous presence of directed and undirected edges;

The graph representation is defined by the representations of each of the graph, vertex and edge types. The graph data structure fundamentally has a “multi-indexing” role. The graph, vertices and edges all mutually refer to each other through containment and through pointing. The graph as a whole needs to refer to its vertices and edges, each vertex refers to & from its own adjacent edges, each edge refers to & from the source and target vertices.

### 4.5.1 Edges

Edge objects represent the graph connectivity, store auxiliary properties of the edges and manage their own storage. Edge objects represent the graph connectivity by storing pointers to their source and target vertices. Edge objects store auxiliary properties of the edges as member data of the edge objects. The only property necessary in the current implementation is a scalar `double val`, the graph-theoretic equivalent of a matrix entry.

The storage of the edges is simplified by the use of an *intrusive* container design. Edge objects have the necessary “hooks” to implement all of the containers in which the edges are involved (in the two vertices and the graph). In this manner, when an existing edge object is added to the list of out-edges of a “source” vertex, the edge’s `srcList_prev` and `srcList_next` are manipulated to link into the other out-edges of the source vertex.

Intrusive list hooks are defined for storing the edge in a list at each of the source and target vertices and in a list at the graph overall. In the implementation, two hooks are used to represent a doubly linked list. The implementation uses the Boost intrusive container hooks and algorithms library [1].

The primary purpose of the edge is to store the actual `val` numerical matrix entry. However, there is some additional storage overhead for storing the edge graph connectivity and list nodes for the edge containers. The data structure overhead per edge is 8 pointers, including its storage in both vertices’ edge lists and the graph’s edge list. The use of intrusive list hooks means that this overhead is minimal for its present capabilities. Some reductions could be made by moving to singly-linked lists or dropping the graph’s edge lists, but at a tradeoff to capabilities. The use of intrusive list hooks means that `edge` contains all the memory that the program requires to link the edge into the vertex and graph edge lists, thereby minimising dynamic memory allocations. Existing statically allocated edges can be added into a graph *without invoking dynamic memory*.

The overhead of structure pointers dominates the storage, compared to the actual

numerical edge values. Other matrix storage schemes, such as the compressed sparse column (CSC) form, are designed to minimise the integer and pointer overhead. However, the goals and capabilities of these schemes are different from the graph oriented scheme presented here.

edge		
double	val;	
vertex	*source,	*target;
edge	*srcList_prev,	*srcList_next;
edge	*trgList_prev,	*trgList_next;
edge	*gphList_prev,	*gphList_next;

**Figure 4.9:** The edge data structure

### 4.5.2 Loops

In the implementation presented here, loops are represented separately from general edges. Loops can be represented with smaller storage than general edges, since loops do not need to refer twice to the same vertex. The data structure overhead per loop is 5 pointers. Otherwise, the loops are implemented in a similar manner as for the general edges.

loop		
double	val;	
vertex	*srctrg;	
edge	*vtxList_prev,	*vtxList_next;
edge	*gphList_prev,	vgphList_next;

**Figure 4.10:** The loop data structure.

### 4.5.3 Multiple Edge-Sets

As discussed in section 4.4, the graph and vertices are required to be able to represent multiple edge-sets. That is, instead of a vertex having a single list of out-edges, it has multiple lists of out-edges (one for each edge-set). The multiple edge-sets are implemented as fixed size, integer indexed arrays. At present it appears sufficient to have a small ( $< 10$ ) fixed length array to store the multiple edge-sets. This requires choosing a maximum number of allowable edge-sets. This implementation uses 7. When designing algorithms, applications will be able to identify the number of linear systems which are required simultaneously and choose an appropriate number of edge-sets. This design provides constant time random-access to the contained lists and also static storage for the array.

edgeSets<TList>	
TList	0;
...	
TList	N

### 4.5.4 Vertices

Vertices provide multiple edge-set lists (each via `edgeSets`) for the loops, and both directed and undirected in and out edges. This design permits algorithms to access the loops immediately and independent of the number of other edges and vice-versa.

Each of the types `LoopList`, `SourceEdgeList` and `TargetEdgeList` are doubly-linked list types, matched to the intrusive list hooks (`trgListHook` or `srcListHook`) of the edge and loop types. The root of the container stores two pointers: to the first list entry and to the last list entry. In practice the implementation uses the Boost intrusive container library [1].

#### SourceEdgeList

Each source `vertex` uses the hooks `srcList_prev` and `srcList_next` in the edges to represent the list of *out* edges.

**TargetEdgeList**

Similarly, each target **vertex** uses the hooks `trgList_prev` and `trgList_next` in the edges to represent the list of *in* edges.

**LoopList**

As for the above, each **vertex** uses the hooks `vtxList_prev` and `vtxList_next` in the loops to represent the list of loops on the vertex.

In the graph representation proposed in this thesis, each individual edge can be directed or symmetric. In this implementation this is achieved by linking the edges in either the directed list or the undirected list. Again, the loops are also separate; The loops are basically a third case in addition to directed and undirected edges.

- The *directed* edges are maintained in lists `outEdges` and `inEdges`.
- The *undirected* edges are maintained in both lists `outEdges_undir` and `inEdges_undir`.

This is because the intrusive container design relies on the source and target vertices using the appropriate different list hooks to link the edges into the edge lists. Therefore the undirected edges must be split across nominal “out” and “in” storage lists, even though the direction is arbitrary. In practice, the implementation developed for this thesis provides a mechanism to pair the “iterators” of these two containers so that two containers appear unified.

Vertices provide list hooks `gphList_prev` and `gphList_next` for linking the vertices into the graph’s vertex list.

vertex	
<code>edgeSets&lt;LoopList&gt;</code>	<code>loopLists;</code>
<code>edgeSets&lt;SourceEdgeList&gt;</code>	<code>outEdges, outEdges_undir;</code>
<code>edgeSets&lt;TargetEdgeList&gt;</code>	<code>inEdges, inEdges_undir;</code>
<code>vertex</code>	<code>*gphList_prev, *gphList_next;</code>
<code>double</code>	<code>x, dx, xnew, b, bnew, ...;</code>

Figure 4.13 shows the manner in which the vertices *contain* and link to the edges. Figure 4.11 shows how the graph contains and links to the vertices.

Vertices bring in additional intrusive list hooks from the user (not shown) to allow further application specific indexing and sorting operations on the vertices. Finally, vertices store the *vector* entries required for the application, for example `x,dx`, `xnew`, `b,bnew`. These are presently added specifically into the implementation. In future work, compile-time metaprogramming can be used to add such customised entries into the static vertex, according to the application.

This inclusion of the vector entries in each vertex object makes adding new vectors possible only at compile-time (impossible at runtime). However, this design is focused on fast insertions of new *objects* rather than new *vectors*.

### 4.5.5 Graph

The graph object itself provides access to the graph's edges, loops (via multiple edge-sets) and vertices. The edges are discriminated into symmetric and directed storage, since the edges themselves are not marked as being either symmetric or directed. Each of the types `LoopList`, `EdgeList` and `VertexList` are doubly-linked list types, matched to the intrusive list hooks (`graphListHook`) of the edge, loop and vertex types:

#### EdgeList

The `graph` uses the hooks `gphList_prev` and `gphList_next` in the edges to represent the complete list of edges of the graph. These hooks are part of the “intrusive” container design.

#### LoopList

Similarly, the `graph` uses the hooks `gphList_prev` and `gphList_next` in the loops to represent the complete list of loops of the graph.

#### VertexList

As for the above, the `graph` uses the hooks `gphList_prev` and `gphList_next` in the vertices to represent the complete list of vertices of the graph.



graph	
edgeSets<EdgeList>	symEdgeLists;
edgeSets<EdgeList>	dirEdgeLists;
edgeSets<LoopList>	loopLists;
VertexList	vertexList;

### 4.5.6 Ordering Properties

Given the above implementation, consider the properties of the graph in relation to *ordering* of the variables and edges:

- The vertex objects exist in memory in some order depending on when and how they are created. This is *not necessarily* a dense, contiguous memory layout. The memory ordering can affect *cache* performance properties.
- The vertices link together to form a linked list (the graph's list of all the vertices) and the loops on the vertices also link together to form a linked list (the graph's list of all the loops). These links each define an ordering, but not necessarily the same as each other or the memory-ordering. Furthermore, as a linked list the ordering can be changed very easily without moving the actual vertex objects in memory.
- The flexibility of the linking and the existence of various different linked lists means that there is no built-in or preferred ordering to the variables represented by the vertices.
- The *identity* of the vertex, loop and edge objects is their pointer address in memory rather than their integer index in a dense array. Therefore their identity and ordering is decoupled from their storage.
- These properties make the creation and removal of vertices and links possible in constant time.

### 4.5.7 Examples

The following figures 4.11-4.13 show specifically the data structures used in representing the graph. Figure 4.11 shows the containment of the vertices from the perspective of the graph and figure 4.12 shows the containment of the edges of the graph. These figures show how the vertex, edge and loop objects provide the intrusive hooks necessary to link themselves within the various containers of the graph. Figure 4.12 also shows the multiple edge-sets facility of the graph. Figure 4.13 shows how the edges are linked from the perspective of their source vertex.

## 4.6 Comparisons

This section presents comparisons of the proposed graph representation against the compressed-sparse-column (CSC) sparse matrix format.

The compressed-sparse-column format is a widely used standard sparse matrix format. Most commonly available linear system software implementations use the CSC (or the row oriented transposed equivalent, CSR), for example: [1, 2, 4, 16, 18, 22, 33, 60] Full details of the representation and algorithms based on the CSC matrix format are given in [2, 9, 17, 67]. The CSC matrix format is illustrated in figure 4.14.

### 4.6.1 Qualitative Comparison

The following table lists qualitative comparisons of the properties of the proposed graph representation and the CSC sparse matrix representation.

Graph Approach	Matrix Approach
Considers the linear system as a mapping between specific <i>objects</i> in the system. The vector values are therefore tightly bound as attributes of the vertices.	Considers the linear system as a mapping between integer indices. The vectors are kept separately from the matrix and are only interlinked by their common use of unique integer indexes.

Graph Approach	Matrix Approach
<p>The graph representation has access semantics of:</p> <p>“What other objects (<math>\mathbf{b}_i</math>) does this object (<math>\mathbf{x}_i</math>) link to, and with what coefficient?”</p> <p>for a given object <math>\mathbf{x}_i</math>, returning (pointers to) the linked objects <math>\mathbf{b}_i</math>.</p>	<p>Matrices have access semantics of:</p> <p>“Coefficient of value at <math>(i, j)</math>?”</p> <p>with input integer indexes <math>i</math> and <math>j</math>. Sparse matrix formats derived historically from dense matrix formats retain these “integer pair” indexing semantics.</p>
<p>There is no required integer indexing of the state blocks. This avoids having to re-index states when another is added or deleted.</p>	<p>Unique integer indexing of the state blocks is required.</p>
<p>Algorithms still have to choose an ordering for many operations. The data structure has various flexible orderings but there is no preferred ordering.</p>	<p>The ordering is encoded into the storage data structure such that algorithms which require changes to the ordering must permute the matrix and vector elements into and back from that ordering.</p>
<p>The placement and identity of each vertex is independent of other vertices. The state vector can grow and/or shrink without affecting the un-changed states. This allows for state augmentation and any-sequence deletions in <math>O(1)</math> (constant) time.</p>	<p>The matrix representation uses contiguous, dense storage. Contiguous, dense storage allows only <i>amortised</i> <math>O(1)</math> (constant) time augmentation by doubling the storage when required. Random order deletions are <math>O(n)</math> time required to shift all the subsequent entries.</p>
<p>Has insertion and erasure operations analogous to a <b>list</b> data structure.</p>	<p>Has insertion and erasure operations analogous to an <b>array</b> or <b>vector</b> data structure.</p>

Graph Approach	Matrix Approach
Does not store the data in contiguous arrays. Pre-computing and allocating the matrix structure is less important.	Stores all the data and indexing in contiguous arrays. Pre-computing and allocating the matrix structure is very important.
Has symmetric and efficient operation in both the direct ( $\mathbf{A}$ ) and transposed ( $\mathbf{A}^T$ ) directions. Repeatedly operating with the transpose is straightforward.	Requires a choice of either column (CSC) or row (CSR) alignment (the “major” alignment). In the CSC alignment, the column iteration is fast and columns are stored densely and contiguously; Rows, on the other hand, are scattered throughout the storage. Algorithms therefore then have to be careful of the difference between row and column aspects and operations or algorithms involving both the direct ( $\mathbf{A}$ ) and transposed ( $\mathbf{A}^T$ ) matrix are avoided. In the CSC alignment, accessing across the rows repeatedly is best done by reordering the entire storage into CSR alignment before accessing the rows [17, pg 9].
Simultaneously represents the system matrix and its graph representation.	The graph representation has to be recovered from the matrix representation at an early stage in planning the solving or factorising process.

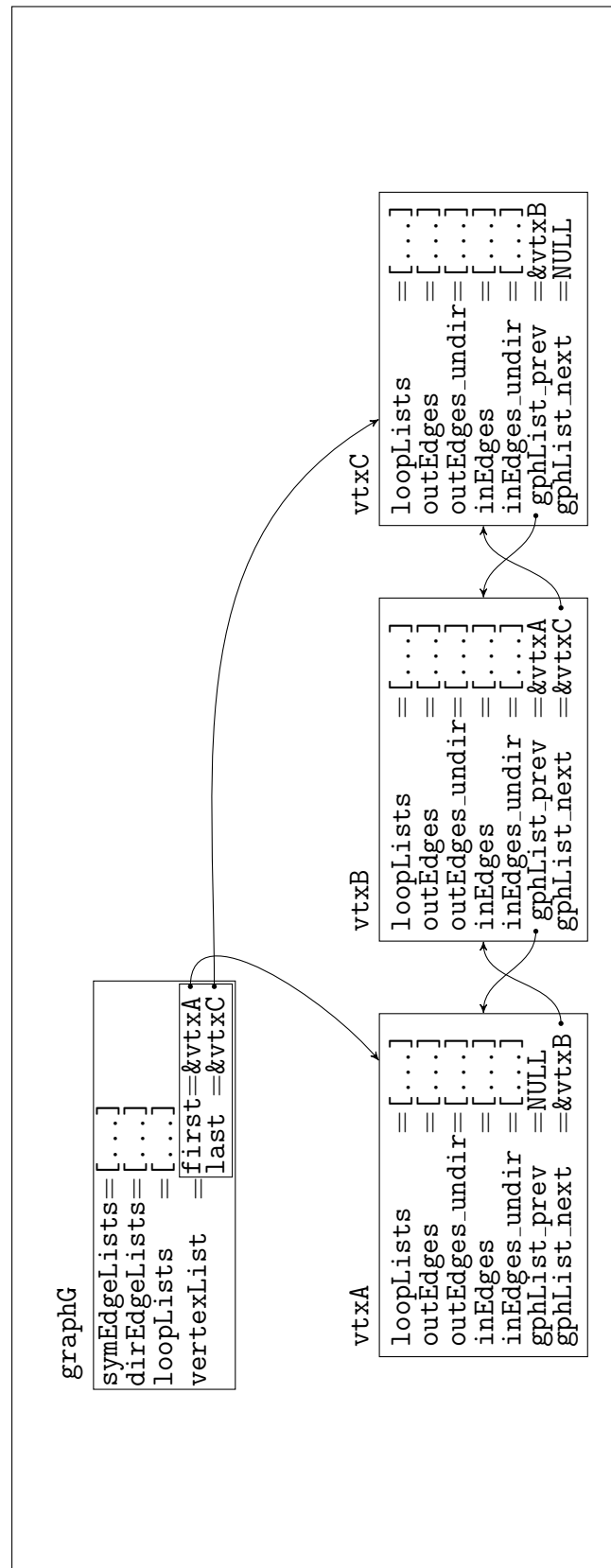
Graph Approach	Matrix Approach
An individual vertex's degree can be evaluated by counting the number of edges (of the directed or symmetric edges or loops, in a particular edge-set). This takes time proportional to the number of edges/loops to be counted but is independent of the number of vertices or edges in the overall system.	For the CSC matrix representation, an individual <i>column</i> degree (count of nonzero entries in a column) is easy to compute in constant time: The degree in column $j$ is $p[j+1]-p[j]-1$ . However, an individual <i>row</i> degree requires a check on all nonzeros in the matrix; Each entry in the appropriate row increments the degree result.
Each sparse matrix nonzero entry has an overhead beyond the numerical value itself of 8 pointers.	Each sparse matrix nonzero entry has an overhead (beyond the numerical value itself) of 1 integer plus 1 integer per column.
Designed for dynamic problems with frequent alterations, insertions and online application.	Designed for large, fixed size problems subject to batch operations.
Vector and edge objects can be allocated randomly in memory (including in dense layouts)	The CSC data structure has dense memory layout optimised for dense, sequential access cache memory systems.

### 4.6.2 Insertion Test

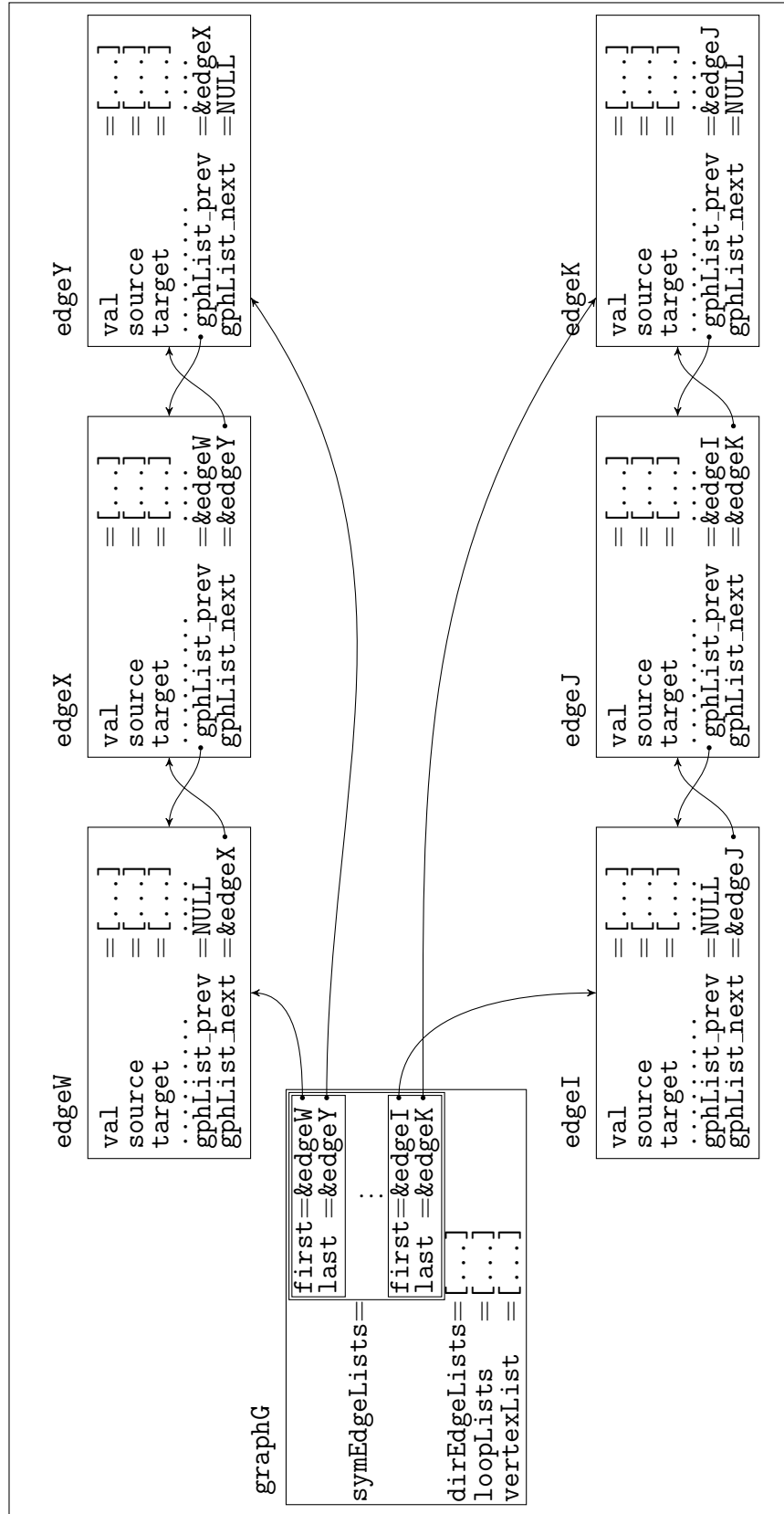
The aim of this test is to compare the performance of the CSC matrix and the proposed graph in the scenario of repeated insertion of new nonzero elements.

This test highlights the difference in the storage approaches of the CSC matrix and the proposed graph representation.

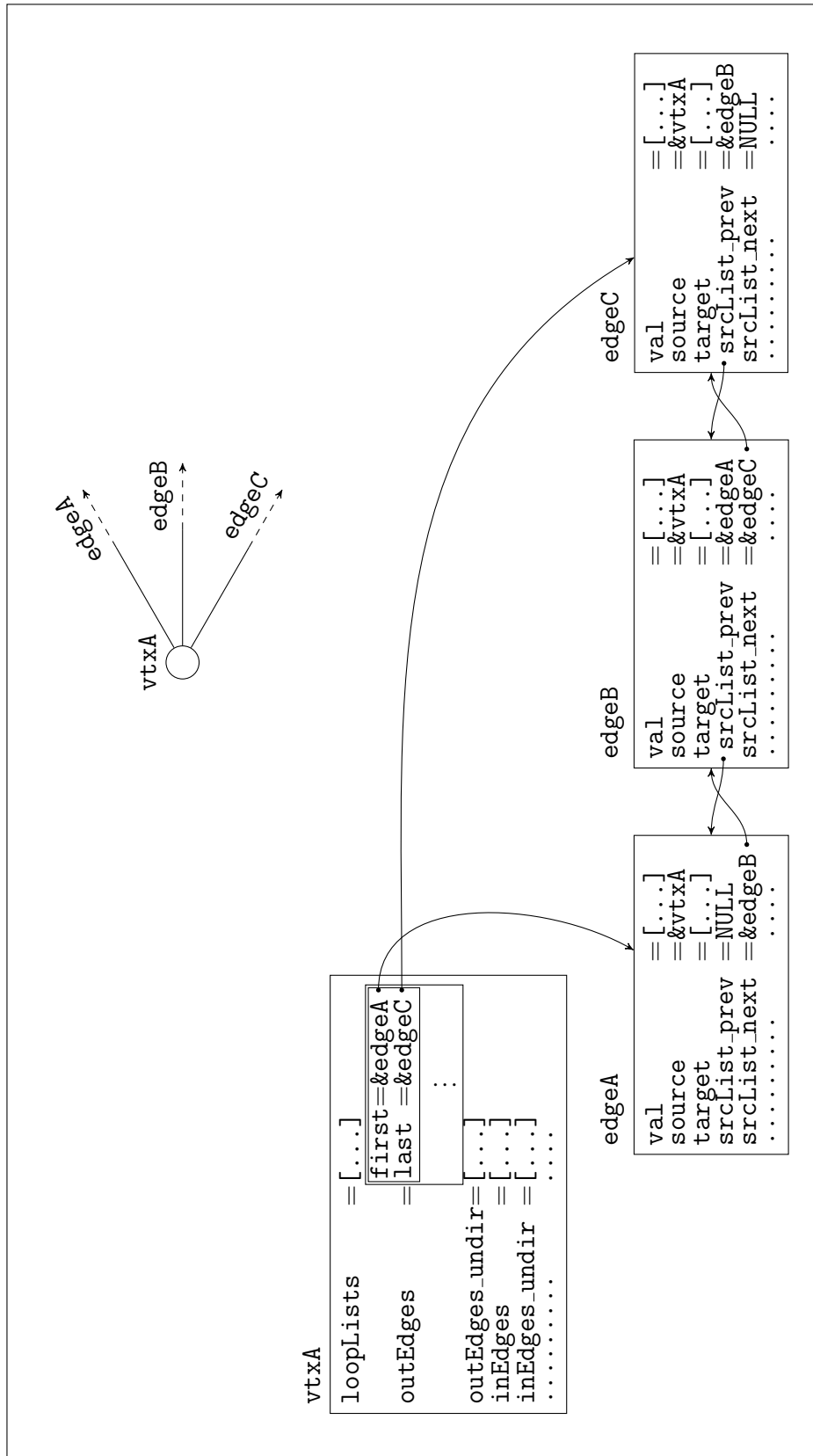
Adding a new nonzero entry in the CSC matrix representation requires shifting some



**Figure 4.11:** The graph containment of the vertex objects. The `graph` has pointers to the first and last vertex, while the vertices themselves have the list pointers to build the vertex list without any additional storage.

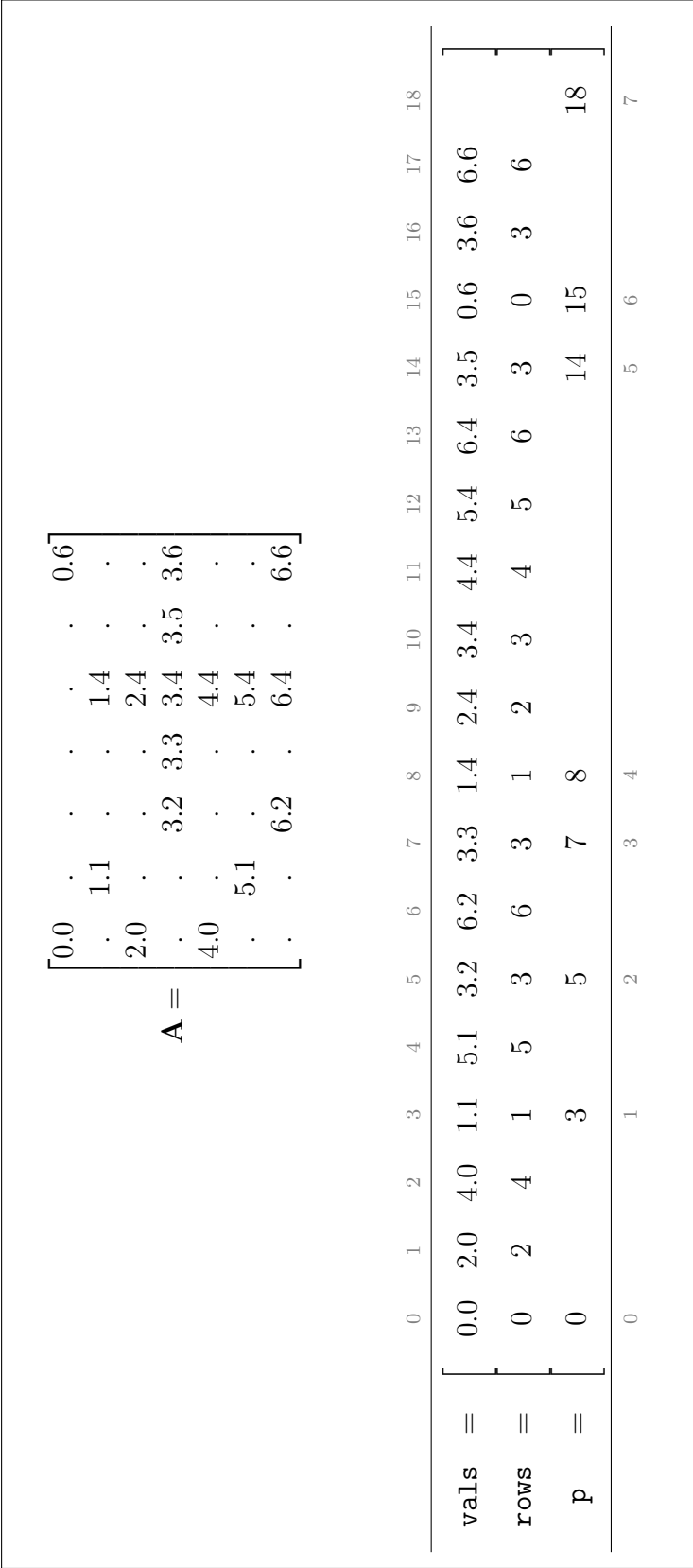


**Figure 4.12:** The graph containment of the edge objects. As for the vertices in figure 4.11, the edges have their own list pointers. The graph object has several lists, one for each edge-set (this figure shows two edge-sets:  $(W,X,Y)$  and  $(I,J,K)$ ). There are also separate lists for directed and symmetric edges and loops.



**Figure 4.13:** The vertex containment of the edge objects. Each vertex has (for each edge-set) the pointers to the start and end of the edge list, for the out and in edges, for the directed and undirected edges. The loops are managed similarly. This diagram shows a single vertex with 3 out-edges, (shown as a graph in the top right). Again, the edge objects have the storage for the containment of themselves in the various edge lists of vertex objects.





**Figure 4.14:** The CSC matrix format. The matrix is defined by the nonzero values (**vals**), the row numbers for each nonzero (**rows**) and **p**, which has the offsets for each column start. A given column **j** from **A** is stored densely and contiguously in **vals** and rows from **vals**[**p**[**j**]] to **vals**[**p**[**j**]+1]-1]. The storage has **nnz(A)** floating-point nonzero values, plus **nnz(A)** row integers, plus **Ncols+1** integers in **p**.

of the other nonzero entries. The shifted entries are those *below* the inserted entry in the same column plus the entries in columns to the *right* of the inserted entry. This will consist of shifting the entire matrix storage if the new entry is inserted at the beginning. In the CSC format, insertion is therefore  $O(n)$  where  $n$  is the number of nonzeros in the matrix.

By contrast, the graph representation does not use dense contiguous storage for the entries. New vertices (nonzeros) can be created anywhere in memory and linked into the graph in constant time. The insertion of new vertices does not cause existing vertices or edges to be copied or moved.

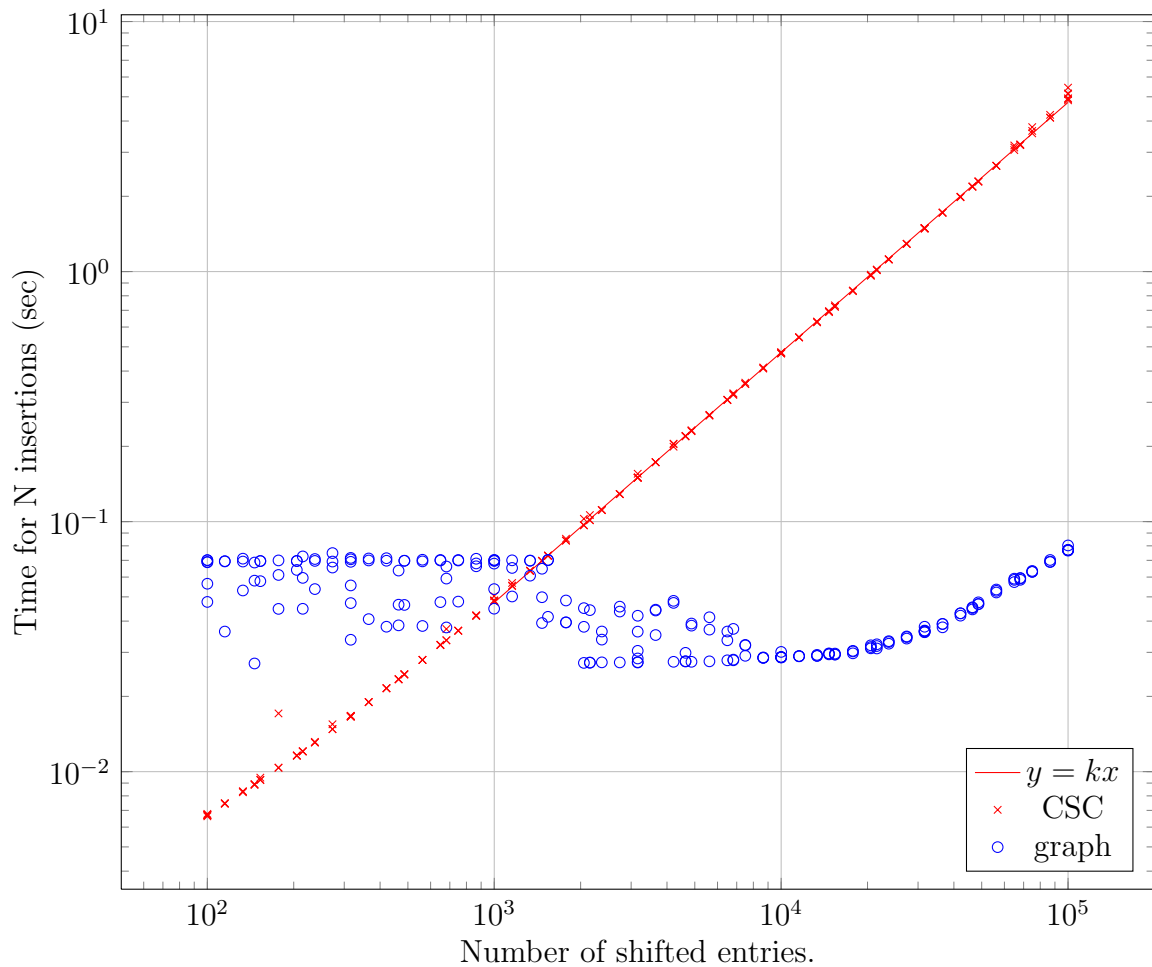
#### 4.6.2.1 Setup

The method of this test was as follows. The test times how long it takes to fill the *first* column with  $M = 1000$  nonzero entries one-by-one. The rest of the  $M \times M$  matrix and the  $N$  other nonzero entries are present before the timing starts. Each of the timed  $M$  first-column insertions has to shift the  $N$  other nonzeros each time. The first  $M$  of these “other” entries are all in a column and so on, filling up columns to reach the necessary  $N$  other entries. The matrix implementation uses the CSC matrix, `compressed_matrix` from the Boost `ublas` library [1].

Similarly, the graph insertions are measured by timing the addition of  $M$  edges one-by-one as out-edges from a given vertex to each of the other vertices (equivalent to the “first column” of the matrix). The  $M$  vertices and the  $N$  “other” edges are created before the timing begins. The rest of the graph has the same  $N$  other edges in the same pattern as for the matrix.

#### 4.6.2.2 Results

The results are shown in figure 4.15.



**Figure 4.15:** Insertion time versus the number of shifted entries for the CSC matrix format and the graph format. In the CSC format the insertion requires a copying of all entries to the left or below the inserted entry in the matrix, and the insertion is therefore  $O(n)$  where  $n$  is the number of shifted entries. The graph approach has constant time insertion.

- The CSC matrix shows a definite  $O(n)$  scaling, highlighted by the straight line  $y = kx$ . This shows that the matrix insertions have to shift the  $N$  “other” matrix entries on each insertion.
- The structure and overheads of the graph format are higher. As a result, the CSC format is faster when there are few other entries in the matrix to shift forward.

- The graph insertions are constant time up until  $N = M = 1000$ , with occasionally *faster* results. The insertions are then achieved in a *faster* but still constant time. This is because beyond  $N = M$ , the  $M$  entries used to time the insertions are linking to vertices which have already been recently accessed in the graph. This is likely a benefit obtained by memory caching effects. This effect diminishes as  $N$  increases to  $10^5$ .

#### 4.6.2.3 Conclusion

This test showed the improved performance of the graph based sparse system representation over the CSC matrix representation for insertions.

The graph insertions are less dependent on the underlying storage than in the matrix representation. This property makes the graph representation more beneficial for online modifications and structure changes. This ability to perform fast structure changes is exploited in chapter 5 for performing the direct solving algorithm.

#### 4.6.3 Access Test

The aim of this test is to show the different properties of the proposed graph representation and the CSC matrix representation relating to the access of adjacent variables.

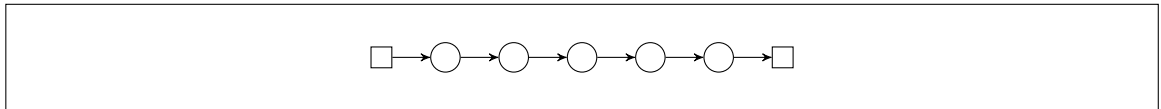
##### 4.6.3.1 Setup

A prototypical scenario to enable benchmark testing of this adjacency access time is the traversal of a linear chain. The traversal of a linear chain is not in itself interesting but is used here as a proxy for adjacency access in general.

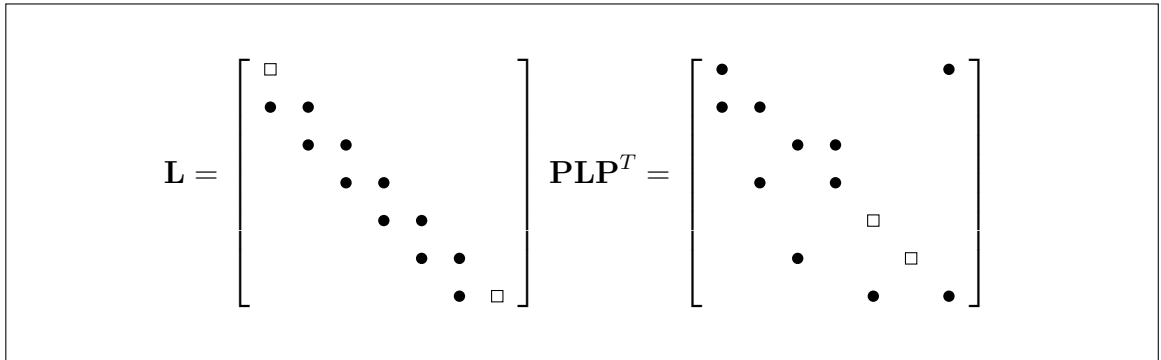
In the setup of this test, the order of linking in the chain is randomised relative to the order of storage in memory of the underlying vertices in the graph. The order of

entry of the variables in the matrix is permuted in exactly the same order. Over the successive tests plotted, the ordering is re-drawn randomly.

This randomisation simulates an unstructured or unpredictably structured problem. This randomisation changes the matrix from  $\mathbf{L}$  in figure 4.16b to  $\mathbf{PLP}^T$ . The permuted system  $\mathbf{PLP}^T$  is used to emphasise that the matrix form is a complicated representation when presented in a general permutation (even of simple chain systems as shown here). The banded structure of  $\mathbf{L}$  is only apparent when  $\mathbf{L}$  is ordered in *topological order*.



(a) A linear chain for the traversal test in graph form.



(b) A lower triangular matrix  $\mathbf{L}$  equivalent to the linear chain in (a). The square entries are the start and end points of the chain. The starting (ending) entry of the chain has no other entries in its row (column). A permuted system,  $\mathbf{PLP}^T$ , is shown to emphasise that the matrix form is a complicated representation when presented in a general permutation (even of simple chain systems as shown here). The banded structure of  $\mathbf{L}$  is only apparent when  $\mathbf{L}$  is ordered in topological order.

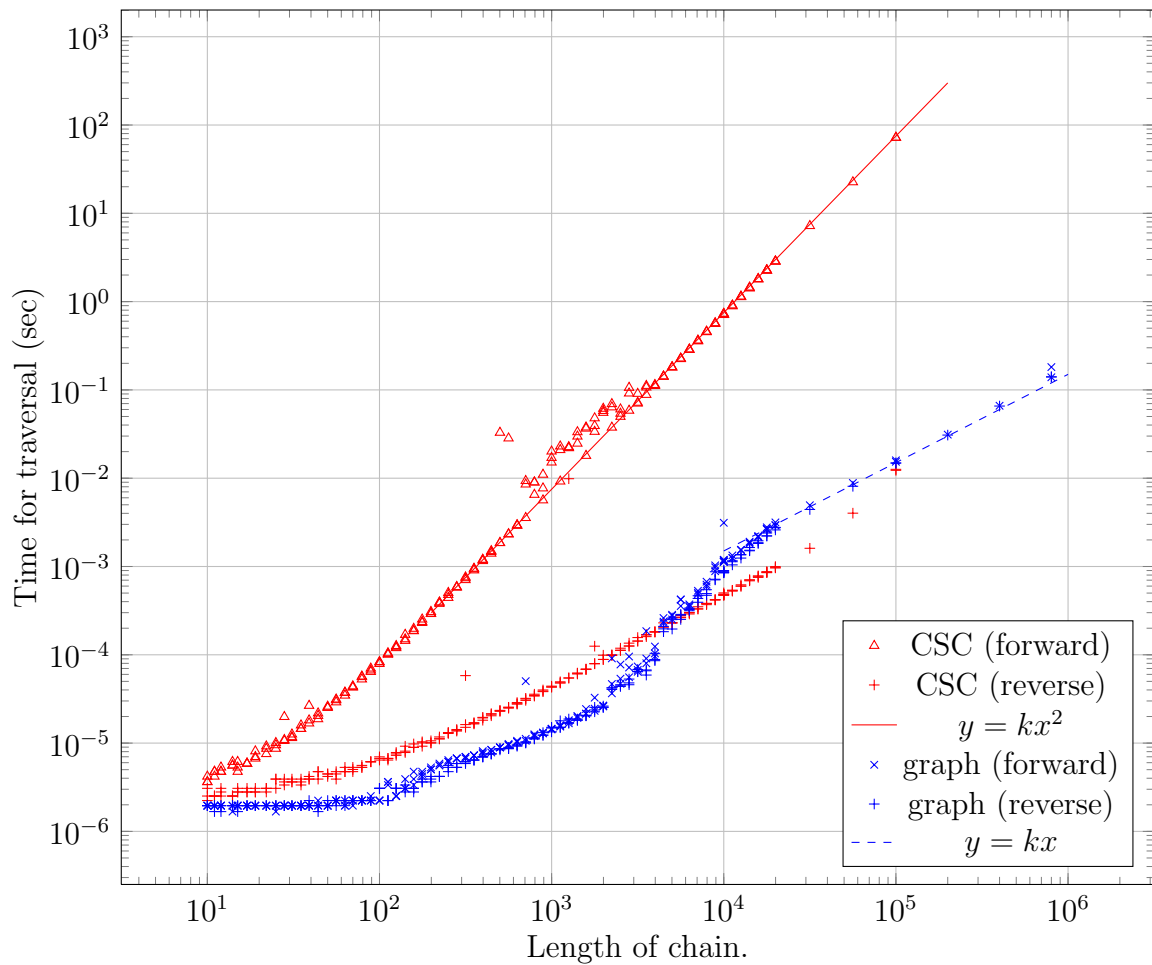
**Figure 4.16:** Graph and Matrix representations of a linear chain for the traversal test.

#### 4.6.3.2 Results

Figure 4.17 shows the time for traversal of the chain graph or matrix versus the length of the chain. For the matrix, the time depends on the *direction*. In one direction

the search for adjacent variables occurs within the *columns* and the single adjacent variable can be found in constant time and the resulting overall traversal occurs in linear time with the length of the chain. However, for the matrix, traversing in the other direction involves a search for the adjacent variable in the *rows*. This search in the rows itself takes linear time and hence the overall chain traversal takes  $O(n^2)$  time.

For the graph, the access of adjacent variables occurs in constant time regardless of the direction. Hence the overall traversal takes linear time, for both directions.



;;

**Figure 4.17:** Traversal time versus chain length. The CSC format has a clear fast direction (+) and a slow direction ( $\triangle$ ). By comparison the graph approach is identically fast in both directions. The speed of the graph and CSC in the fast direction are approximately the same.

#### 4.6.3.3 Conclusion

The adjacency access operations on the graph occur in constant time regardless of the direction. This is because the underlying representation stores pointers which can be used in both directions for constant time access.

By comparison, the matrix has a preferred direction (column accesses). The properties

of operating in the matrix is more dependent on the underlying storage representation; The speed is affected by the direction of access in relation to the storage order.

In the matrix's preferred direction, its speed was not significantly different to the speed of the graph representation. In the matrix's non-preferred direction, its speed was orders of magnitude slower and ultimately scales worse at  $O(n^2)$  compared to the graph representation's  $O(n)$ .

The ability of the graph to make constant time adjacency accesses in either direction is generally very useful for decoupling algorithms from the peculiarities of the underlying storage. This ability is exploited in the graph based direct solving algorithm of chapter 5.

## 4.7 Future Research

This section describes some possibilities for future research and development in the graph representation.

### Generic programming

The graph representation described in this chapter makes a number of extensions beyond the generic graph concepts specified in [65]. The approach taken in [65] is to decompose various graph representations into what *capabilities* they represent via C++ generic programming with templates. The various representations then support interfaces relating to those capabilities.

The extensions proposed in this thesis could be mapped into generic programming concepts in order to capture the capabilities behind each of the extensions presented here. This would be a superset of the existing generic interfaces in [65].

### Block based representation

Block based sparse matrix representations focus on systems in which variables occur in dense clusters and where the sparsity patterns typically follow these clustering patterns. In this way, the clusters of variables can be stored together.



Block representations exist for sparse matrices (for example: Block Sparse Row/Column [60]).

In the development of this chapter a block approach was considered. However, the simpler scalar based representation was deemed to be required before a block approach could be pursued. Each graph vertex would contain a dense group of variables. Suppose a two vertices  $a$  and  $b$  contain, respectively  $n$  and  $m$  variables. The edge between  $a$  and  $b$  would therefore carry a dense matrix of size  $n \times m$  (or  $m \times n$ ).

Another complication is in the manipulation of blocks of different sizes. Ideally a statically allocated vertex should contain enough memory for the data of the block rather than requiring dynamic allocation of the block separately. It is easy to achieve this for blocks of a fixed size and to use templates to generalise these into a “family of fixed sizes”.

This representation would require the appropriate block solving algorithms. For iterative algorithms, the matrix-vector multiply in this block based graph representation is very convenient. Each adjacency access accesses the adjacent block of variables and multiplies through by the edge matrix. For direct solving algorithms, the situation would be somewhat more complicated. Block factorisation could be pursued such that each operation in the factorisation operates on the blocks. Dense algorithms would then be applied for each of the scalar operations (see chapter 5).

### **Shared Memory Parallel Representation**

The graph representation described in this chapter is well suited to a (shared memory) parallel implementation, since the structure is embodied in linked lists rather than dense memory. With the ongoing use of multi-core computers, shared memory parallelism is increasingly important [58]. Libraries with parallel containers and operations [58] could replace the linking structures used in the implementation described here. A shared-memory implementation would allow multiple processors to operate simultaneously within the graph structure.

### **Decentralised Representation**

The graph representation proposed in this chapter could be extended to a

decentralised (distributed memory parallel) implementation. Some vertices would be held locally and some held on remote platforms. Access to “adjacent” variables on remote platforms would invoke the necessary communications to obtain those variables.

## 4.8 Conclusion

This chapter described the graph based representation of linear systems, which is a fundamental contribution of this thesis. The graph based representation of linear systems together with the underlying graph representation contribute in a bottom-up manner, filling a fundamental role in the formulation of an estimation system.

The advantages of this graph embedded linear system representation will be used in the development of the direct solving algorithms. In summary, the benefits of the graph structure are as follows:

### **Easy insertions**

Additional edges in the graph can be created or removed in constant time in random order.

### **Flexible Ordering of Variables**

The access of variables is based entirely on “object access” and adjacency rather than integer indexing. The various data structure linked lists define orderings but these are very flexible to changes. The ordering is not fundamentally related to the identity of the objects or their underlying storage. Symmetric systems distinguish only between the loop entry (linear system entry for a variable with respect to itself) and edge entries (linear system entries for a variable with respect to other variables). There is no concept of “above or below” the diagonal in a row or column in symmetric systems.

### **Fast constant time adjacency accesses**

From a particular variable there is fast and constant time access to a list of the adjacent variables of a linear system, in both directions, for both directed and

symmetric systems.

### **Graph properties**

As a graph based representation properties such as the vertex degree are easily evaluated.

The proposed graph representation also introduces novel features as a result of its application to linear systems:

### **Multiple Edgesets**

The proposed graph representation adds an argument to each operation which refers to a particular *edge-set*. This effectively couples multiple graphs together in an efficient and logical manner. This extension was motivated by the need to represent multiple linear systems over a single set of vertices.

### **Edges and Loops**

The proposed graph representation distinguishes between edges and loops. This was motivated by the focus in this thesis on symmetric systems. In symmetric systems the loop entries (diagonal entries) in the linear system are distinct from the edge (off-diagonal) entries.

### **Symmetric and Directed Edges**

The proposed graph representation is able to operate with a mix of directed and symmetric edges in each graph. This is motivated by cases in linear algebra where a matrix may have both symmetric and triangular portions simultaneously.

The following chapter builds on the graph representation contributed here by proposing direct solution methods to operate within the graph structure.

# Chapter 5

## Graph-Theoretic Solution Methods

### 5.1 Introduction

This chapter presents a graph based direct solving algorithm. This algorithm solves linear symmetric indefinite systems which arise in the augmented system form for estimation problems. This chapter will propose methods for the solving algorithms to exploit the graph embedded representation of the linear systems. The proposed graph embedded representation opens up the development of a new variety of graph based estimation and linear algebraic tools which may be useful in future for faster, online solving algorithms.

The previous chapter discussed the graph representation of systems of variables generally. In this thesis, the variables consists of observation and state variables of estimation problems, coupled together by the augmented system form of chapter 3.

Section 5.2 introduces the LDL factorisation algorithm, which is the basis for this chapter. Section 5.3 shows the proposed graph based LDL factorisation algorithm. The use of the LDL factorisation for the solution of linear systems is discussed in section 5.4. The reconstruction of the original system given the LDL factorisation is shown in section 5.5 and the construction of the full Section 5.7 describes areas for future work in graph based solution algorithms.

## 5.2 Symmetric LDL Factorisation Introduction

This section introduces the LDL factorisation algorithm for the direct solution of symmetric linear systems.

Direct solution methods modify the problem by successively eliminating some variables, expressing an equivalent problem without those variables, together with back references which allow the eliminated variables to be computed once the solution to the reduced problem is later obtained. Once the problem is reduced to a trivially solvable problem in one or two dimensions, the solutions can be propagated back to compute the eliminated variables in sequence. Direct methods have a progressive pattern which is strongly affected by the sparse graph structure of the problem.

The LDL factorisation algorithm is a direct solving method and is a specialisation of *Gaussian elimination* for symmetric indefinite and positive definite systems [37]. The need for the solution of indefinite systems arises from estimation problems with augmented observations and constraints, as used in this thesis. The solution of positive definite systems is included within the methods described in this chapter, since positive definite systems are a subset of indefinite systems and their solution method is a subset of the solution method for indefinite systems. The LDL factorisation algorithm of this chapter could be modified into the LU factorisation [37] to apply to unsymmetric linear systems, if required for other applications.

The direct solving algorithms are strongly tied to the linear system representation on which they operate. Changes to the linear system representation therefore have a significant effect on the overall direct solving algorithm. In this thesis, the proposed graph representation of linear systems introduces some significant changes, in particular:

### Fast constant time insertions

Additional edges in the graph can be created or removed in constant time in random order.

This affects the solving algorithm because direct solving algorithms fundamentally require performing structure changes to the systems. This altered

property therefore affects how structure changes are planned and performed in the algorithm.

### **Flexible Ordering of Variables**

The access of variables is based entirely on “object access” and adjacency rather than integer indexing. The various data structure linked lists define orderings but these are very flexible to changes. The ordering is not fundamentally related to the identity of the objects or their underlying storage. Symmetric systems distinguish only between the loop entry and edge entries. There is no concept of “above or below” the diagonal in a row or column for symmetric systems.

This affects the solving algorithm because it decouples the relationship between the storage and the factorisation ordering, allowing more flexible factorisation approaches.

### **Fast constant time adjacency accesses**

From a particular variable there is fast and constant time access to a list of the adjacent variables of a linear system, in both directions, for both directed and symmetric systems.

The adjacency accesses are a core operation in the direct solving algorithm. This property also decouples factorisation from the underlying storage.

The changes introduced by the proposed graph embedded linear system representation affect changes in the solving algorithm in its graph embedded form, as follows:

### **Flexible factorisation ordering**

The ability of the graph embedded linear system representation to perform constant time sparsity pattern changes means that the requirement of pre-planning of the factorisation sparsity pattern is relaxed in the proposed graph based factorisation algorithm. This means that the process of computing the factorisation ordering is not dictated by storage considerations.

Typically, direct solving algorithms pre-plan and pre-build the sparsity pattern of the factorisation in order to avoid performing modifications, which are expensive in the CSC sparse matrix representation.

For well conditioned positive definite systems, this pre-planned factorisation order is sufficient, since the sparsity pattern can be pre-computed quickly and exactly from sparsity pattern considerations alone. The algorithm does not need to depart from the pre-planned ordering.

However, for indefinite systems and for poorly conditioned positive definite systems, as are considered in this thesis, the factorisation ordering can also be altered for *numerical stability* reasons. The need for these cannot be pre-determined faster than performing the actual factorisation itself. The proposed graph based factorisation algorithm can incorporate numerical factorisation-order changes easily, since the ordering is not pre-computed and the sparsity pattern is not pre-built.

Changing factorisation ordering for numerical reasons is also known as “pivoting”. In the proposed graph based factorisation algorithm, changing the factorisation ordering can be performed without needing to explicitly move the data around to permute the system into the new ordering, since the graph based linear system representation does not dictate the factorisation ordering. In the graph based algorithms presented in this thesis, the flexibility of the ordering of the variables means that no permutation or data copying is required regardless of which variable is factorised next.

In conventional matrix terminology, “pivoting” is associated with explicit *changes* to an existing ordering (implied by the integer indexing of the matrix). Since pivoting involves changes to the matrix ordering, explicit permutation operations are performed to move the ordering of the variables in memory. The explicit permutation is required because the algorithms use progression through the storage to mark the progress of the algorithm [37]. By contrast, in the graph based scheme, the algorithm is able to mark the progress of factorisation by clearing edges from the source.

The graph based representation has the advantage of immediate runtime access to the graph representation of the matrix structure, in order to be able to evaluate the next choice of factorisation variable.

Choosing a factorisation ordering is still an important and difficult topic. The

proposed graph embedded approach makes the choice easier by decoupling it from storage, pre-planning and data re-arranging considerations. However, the more fundamental concerns of how to order the factorisation on the basis of sparsity structure, numerical concerns and online modifications still remain as important and difficult topics for further research.

### Algorithm form

The LDL factorisation algorithm described in this chapter uses the “outer-product form” of the LDL factorisation. The outer-product form differs from the LDL algorithm of Davis [16], which uses the “sparse triangular solve form”. Both result in exactly the same factorisation  $\mathbf{L}$  and  $\mathbf{D}$  given the same factorisation ordering.

The outer-product form has much in common with methods from the estimation literature. The Schur complement formed during factorisation is the same as the Schur complement of the marginals formed in estimation when marginalising variables. In particular, if the observation variables are factorised first, the Schur complement formed is the *information form* (See chapter 3). The outer-product form explicitly modifies the un-factorised part of the system during factorisation, forming the Schur complement (the *marginal* of the system in the un-factorised variables). The factorised part is not accessed after factorisation. Instead, the modifications resulting from factorisation are reflected forward onto the un-factorised part of the system. This makes many structure changes to the linear system: Both in the construction of the factorised system and in the modifications to the un-factorised system.

The sparse triangular solve form, by contrast, accesses back to the factorised part rather than forward modifying the un-factorised part. This also fits in well with the approach of pre-planning the factorisation algorithm and pre-building the sparsity pattern.

The outer-product form makes many temporary structure changes in the original system during factorisation. The outer-product form is unfeasible for the conventional CSC matrix representation because each of these structure changes in the CSC matrix take  $O(n)$  time in the number of nonzeros. However, in the



graph based representation proposed in this thesis, the outer-product form is feasible because each structure change can be performed in constant time.

### 5.2.1 LDL Factorisation, Mathematical Form

The LDL factorisation was introduced in section 2.4.2. This section further describes the  $\mathbf{LDL}^T$  factorisation in general mathematical terms, for positive definite and indefinite symmetric systems.

The LDL factorisation is suitable for indefinite systems (but not semi-indefinite systems). For semi-indefinite systems, a *regularised* solution can be obtained by regularisation of the system. (See section 3.2.9).

The *diagonal pivoting*  $\mathbf{LDL}^T$  method ([14] and [37, page 168]) factorises  $\mathbf{A}$  as follows:

$$\mathbf{PAP}^T = \mathbf{LDL}^T \quad (5.1)$$

- $\mathbf{A}$  is the input square, symmetric matrix.
- $\mathbf{P}$  acting on  $\mathbf{A}$  via  $\mathbf{PAP}^T$  represents a symmetric permutation of  $\mathbf{A}$ .
- $\mathbf{L}$  is a unit lower triangular matrix.
- $\mathbf{D}$  is a block diagonal matrix.

The diagonal pivoted  $\mathbf{LDL}^T$  factorisation is based on the following block partitioning of  $\mathbf{A}$ :

$$\mathbf{PAP}^T = \begin{bmatrix} \mathbf{E} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{B} \end{bmatrix} \begin{matrix} s \\ n-s \end{matrix} \quad (5.2)$$

$s \quad n-s$

- $\mathbf{E}$  is the  $s \times s$  block which will be factorised out of  $\mathbf{A}$ . This block,  $\mathbf{E}$ , consists of the chosen  $s$  variables from  $\mathbf{A}$ , permuted into a block via  $\mathbf{P}$ .
- $\mathbf{C}$  is the  $(n-s) \times s$  off-diagonal block from  $\mathbf{A}$  which connects  $\mathbf{E}$  to the rest of the system.
- $\mathbf{B}$  is the  $(n-s) \times (n-s)$  block of those variables from  $\mathbf{A}$  which were not included in  $\mathbf{E}$ .

The block factorisation is then written as:

$$\mathbf{PAP}^T = \mathbf{LDL}^T \quad (5.3)$$

$$= \begin{bmatrix} \mathbf{I}_s & \mathbf{0}_{(s,n-s)} \\ \mathbf{CE}^{-1} & \mathbf{I}_{n-s} \end{bmatrix} \begin{bmatrix} \mathbf{E} & \mathbf{0}_{(s,n-s)} \\ \mathbf{0}_{(n-s,s)} & \mathbf{B} - \mathbf{CE}^{-1}\mathbf{C}^T \end{bmatrix} \begin{bmatrix} \mathbf{I}_s & \mathbf{E}^{-1}\mathbf{C}^T \\ \mathbf{0}_{(n-s,s)} & \mathbf{I}_{n-s} \end{bmatrix} \quad (5.4)$$

The full factorisation continues by factorising  $\mathbf{A}_2 = \mathbf{B} - \mathbf{CE}^{-1}\mathbf{C}^T$  in the same manner, using Equation 5.4, until  $\mathbf{A}_2$  is itself easily solvable (usually when  $\mathbf{A}_2$  is of size  $n \leq 2$ ).

The  $\mathbf{LDL}^T$  factorisation allows operation with indefinite symmetric matrices, since the  $\mathbf{D}$  factor can represent positive or negative entries. This cannot be achieved in the Cholesky factorisation, which requires a positive definite input matrix.

The process of choosing an  $\mathbf{E}$  other than the scalar  $A_{11}$  is referred to as *pivoting* and  $\mathbf{E}$  is the *pivot*.

### Positive Definite Case

A positive definite  $\mathbf{A}$  is an important special case. This section explains the positive definite case to contrast more general cases which are considered later. In the positive definite case:

- $\mathbf{E}$  is always invertible regardless of the choice of  $\mathbf{E}$ . Therefore the choice of  $\mathbf{E}$  is able to be made primarily on the basis of computational performance, with numerical stability a secondary or trivial consideration.
- $\mathbf{E}$  can be a scalar and  $\mathbf{D}$  can be scalar diagonal.

### Indefinite Case

In the case of indefinite  $\mathbf{A}$ :

- To guarantee that  $\mathbf{E}$  is invertible at each step, whenever  $\mathbf{A} \neq \mathbf{0}$ , it is sufficient to allow both scalar and  $2 \times 2$  block factorisation steps [37, page 168].

A  $2 \times 2$  block factorisation step corresponds to simultaneously eliminating 2 variables at once. In an indefinite matrix, the diagonals may be all zero despite the whole matrix being nonzero overall, since the off-diagonals may be nonzero. If all the diagonals of  $\mathbf{A}$  are zero, then any scalar  $\mathbf{E}$  will be zero. Therefore it is

not sufficient to only allow scalar  $\mathbf{E}$ . It is not sufficient to allow only  $2 \times 2$   $\mathbf{E}$  since  $\mathbf{A}$  may be of rank 1, which will not have an invertible  $2 \times 2$  submatrix for which to choose  $\mathbf{E}$ .

Using either scalar and  $2 \times 2$   $\mathbf{E}$  guarantees  $\mathbf{E}$  is invertible at each step, whenever  $\mathbf{A} \neq \mathbf{0}$ : If the diagonals of  $\mathbf{A}$  are not all zero, the algorithm could choose a nonzero scalar  $\mathbf{E}$  from the diagonal of  $\mathbf{A}$ . Otherwise, the algorithm can choose  $\mathbf{E} = \begin{bmatrix} 0 & a \\ a & 0 \end{bmatrix}$  from any nonzero offdiagonal,  $a$ .

- The case of indefinite  $\mathbf{A}$  arises for the augmented system form discussed in chapter 3. The example 5.1 below shows a numerical example of when a  $2 \times 2$   $\mathbf{E}$  is required.

**Example 5.1.** 

---

***The requirement for 2 by 2 block factorisation steps.***

When the diagonals of  $\mathbf{A}$  are all zero, as occurs in this example, the algorithm can choose a  $2 \times 2$   $\mathbf{E}$  in order to guarantee an invertible  $\mathbf{E}$  for Equation 5.4. This example shows a numerical example of when a  $2 \times 2$   $\mathbf{E}$  is required.

Consider a  $n$  dimensional set of states with zero prior information  $\mathbf{Y}$  and a full rank (consistent) set of constraints.

$$\text{To indicate perfect constraints: } \mathbf{R} = \mathbf{0} \quad (5.5)$$

$$\text{To indicate zero prior information: } \mathbf{Y} = \mathbf{0} \quad (5.6)$$

$$\text{then } \mathbf{A} = \begin{bmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{bmatrix} \quad (5.7)$$

$$= \begin{bmatrix} \mathbf{0} & \mathbf{H} \\ \mathbf{H}^T & \mathbf{0} \end{bmatrix} \quad (5.8)$$

In  $\mathbf{A}$ , the diagonals will all be zero, therefore  $\mathbf{E}$  cannot be a scalar. The algorithm can instead choose an invertible  $\mathbf{E}$  as:

$$\mathbf{E} = \begin{bmatrix} 0 & h \\ h & 0 \end{bmatrix}$$

with  $h$  from any nonzero part of  $\mathbf{H}$ .

Consider a numerical example:

$$\mathbf{A} = \begin{bmatrix} 0 & & 2 & -2 & 1 \\ & 0 & 8 & 3 & -8 \\ & & 0 & 3 & 5 & 9 \\ 2 & 8 & 3 & 0 & & \\ -2 & 3 & 5 & & 0 & \\ 1 & -8 & 9 & & & 0 \end{bmatrix}$$

The algorithm may choose an invertible  $2 \times 2$   $\mathbf{E}$  from indices 1 and 4, for example. This results in:

$$\mathbf{E} = \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} \quad \mathbf{C}^T = \begin{bmatrix} 0 & 0 & -2 & +1 \\ +8 & +3 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 3 & -8 \\ & 0 & 5 & 9 \\ 3 & 5 & 0 \\ -8 & 9 & & 0 \end{bmatrix}$$

From Equation 5.4:

$$\mathbf{L} = \begin{bmatrix} \mathbf{I}_s & \mathbf{0} \\ \mathbf{CE}^{-1} & \mathbf{I}_{n-s} \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} \mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} - \mathbf{CE}^{-1}\mathbf{C}^T \end{bmatrix}$$

$$\mathbf{L} = \begin{bmatrix} +1 & & & & & \\ & 0 & +1 & & & \\ +4 & 0 & +1 & & & \\ +1.5 & 0 & 0 & +1 & & \\ 0 & -1 & 0 & 0 & +1 & \\ 0 & +0.5 & 0 & 0 & 0 & +1 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 0 & +2 & & & & \\ +2 & 0 & & & & \\ & & 0 & & +11 & -12 \\ & & & 0 & +8 & +7.5 \\ & & +11 & +8 & 0 & \\ & -12 & +7.5 & & & 0 \end{bmatrix}$$

with  $\mathbf{LDL}^T = \mathbf{PAP}^T$



### 5.2.2 LDL Factorisation, Dense Matrix Form

This section presents a conventional LDL factorisation for dense matrices for reference purposes.

Algorithm 2 performs the LDL factorisation on a dense matrix using the outer-product approach. Algorithm 2 simply uses the factorisation order built into the matrix:  $(1, \dots, N)$  without pivoting. This algorithm relies on the index  $k$  to to *implicitly* maintain track of the progress of the factorisation - which variables have already been factorised and those yet to be factorised. Therefore changing the sequencing of the factorisation (pivoting) requires explicit *permutations* of the data in memory [37]. The factorisation ordering in this algorithm is simply  $[1 : N]$ , following literally the partitioning in equation 5.2.

**Algorithm 2:**  $\text{LDL}^T$  Factorisation, dense matrix outer-product form

**Input:** A symmetric matrix,  $\mathbf{A}$ , size  $N \times N$   
**Result:** Lower triangular matrix  $\mathbf{L}$   
**Result:** diagonal matrix  $\mathbf{D}$   
**Result:**  $\text{LDL}^T = \mathbf{A}^T$   
**begin**  
     $\mathbf{L} = \mathbf{I}_N$   
     $\mathbf{D} = \mathbf{A}$   
    **for**  $k = 1 : N$  **do**  
        pivots =  $k$   
        notpivots =  $(k+1):N$   
         $\mathbf{E} = \mathbf{D}(\text{pivots}, \text{pivots})$   
         $\mathbf{C} = \mathbf{D}(\text{notpivots}, \text{pivots})$   
         $\mathbf{B} = \mathbf{D}(\text{notpivots}, \text{notpivots})$   
         $\mathbf{L}(\text{notpivots}, k) = \mathbf{C}\mathbf{E}^{-1}$   
         $\mathbf{D}(\text{notpivots}, \text{notpivots}) -= \mathbf{C}\mathbf{E}^{-1}\mathbf{C}^T$   
         $\mathbf{D}(\text{pivots}, \text{notpivots}) = 0$   
         $\mathbf{D}(\text{notpivots}, \text{pivots}) = 0$   
    **end**

Another variant is algorithm 3: Algorithm 3 also uses a dense representation and outer-product form. This algorithm explicitly maintains track of the factorised and non-factorised variables and is able to choose the pivots arbitrarily without having to *copy the data* in the pivoting process. This form of the algorithm is able to factorise

the scalar and  $2 \times 2$  blocks. Algorithm 3 was written for this thesis as a “half-way” approach between the versions found in references such as [37, 38] and the graph based approach of this chapter, in order to help explain the graph based factorisation approach described in this chapter.

For further details on dense matrix LDL factorisation and variants see [37, 38]. For dense matrices, the LDL factorisation is available in LAPACK as DSYTRF [5] using the Bunch-Kaufman diagonal pivoting method based on [15] and in MATLAB as `ldl`.

**Algorithm 3:**  $\text{LDL}^T$  Factorisation, Matrix Form (with Matlab notation)

**Input:** A symmetric matrix,  $\mathbf{A}$ , size  $n \times n$   
**Result:** Permuted triangular matrix  $\mathbf{L}$   
**Result:** Permuted block diagonal matrix  $\mathbf{D}$   
**Result:** Factorisation ordering vector  $\mathbf{P}$   
**Result:**  $\text{LDL}^T = \mathbf{A}^T$

```

begin
    Set the permutation ordering vector, P empty
    Define a vector done = zeros(1,n)
    Define a vector todo = find(not done)
    L = eye(n)
    D = A
    while todo is not empty do
        Choose pivots (length 1 or 2) from todo
        Define notpivots as the rest of todo excluding pivots
        Append pivots to P
        Gather up the block partitioning terms C, B, E
        B = D(notpivots,notpivots)
        C = D(notpivots,pivots)
        E = D(pivots,pivots)
        if min(abs(eig(E))) < tol then
            break // Error, E too small
        L(notpivots,pivots) = CE-1
        D(notpivots,notpivots) -= CE-1CT
        D(notpivots,pivots) = 0
        D(pivots,notpivots) = 0
        done(pivots) = true
        todo = find(not done)
    end

```

## 5.3 Symmetric LDL Factorisation - Graph Form

This section describes the graph based LDL factorisation algorithm. This factorisation algorithm is based entirely in the graph representation of the sparse linear systems. This is one of the core contributions of this thesis.

The mathematical form of the algorithm is the same as those given in section 5.2. The contribution of this section is the form of the algorithm based entirely in the graph representation for the linear systems.

When a vertex (the pivot vertex) is chosen for factorisation, the algorithm finds the set of off-diagonals  $\mathbf{C}$  as simply the set of adjacent edges to the pivot vertex. The LDL factorisation algorithm consists of copying this set of symmetric adjacent edges  $\mathbf{C}$  in to  $\mathbf{L}$  as directed edges  $\mathbf{CE}^{-1}$  and subtracting onto the remaining un-factorised system as the outer product  $\mathbf{CE}^{-1}\mathbf{C}^T$ . These operations are also generalised for the case of 2 simultaneous pivot vertices (for the indefinite factorisation). In the case of 2 pivot vertices  $\mathbf{E}$  is a  $2 \times 2$  system and  $\mathbf{C}$  consists of the set of the 2 vertex's adjacent edges. The process of obtaining the adjacent edges and their manipulation to perform the factorisation is entirely based in the graph representation.

Section 5.3.1 describes the major factorisation functions. Section 5.3.2 describes the underlying graph-based linear algebra procedures used in the factorisations.

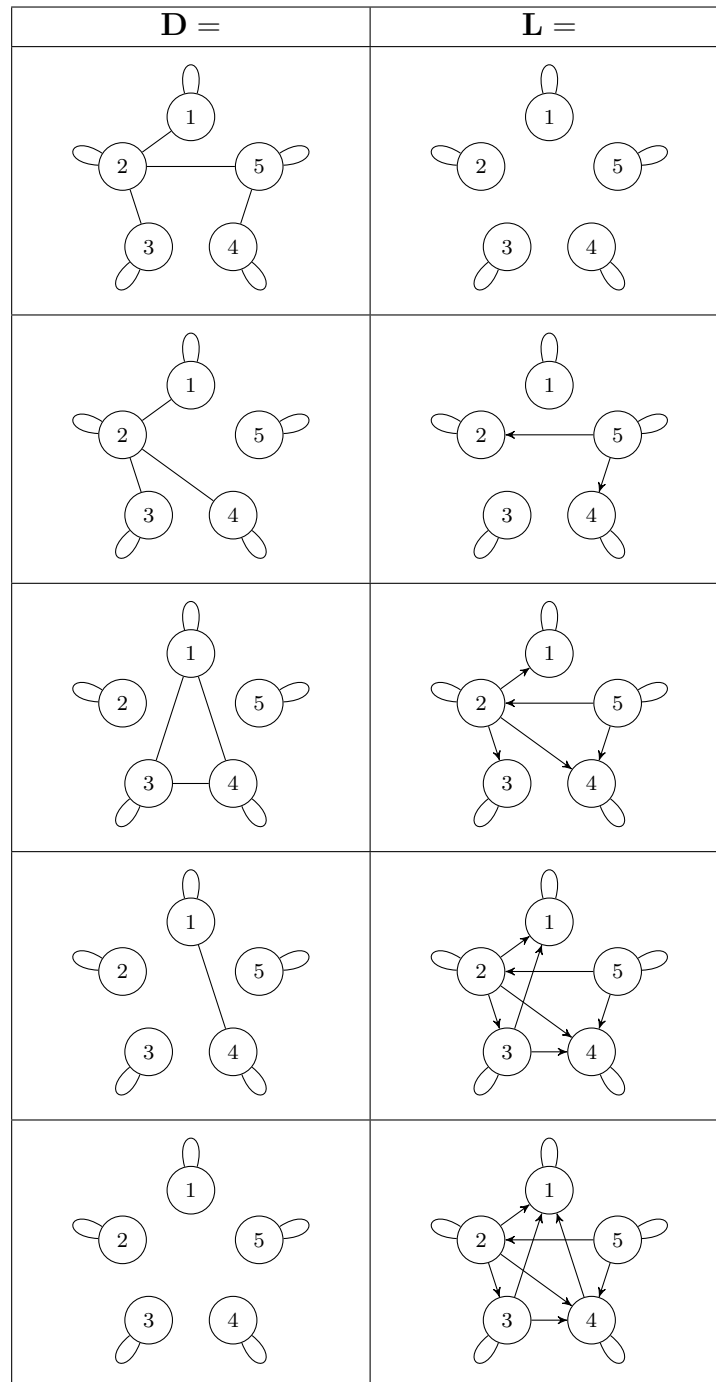
### 5.3.1 Graph Based Factorisation Steps

The following sections define the indefinite (5.3.1.1) and positive-definite (5.3.1.2) LDL factorisation. These, in turn, use the subfunctions for scalar (5.3.1.3) and  $2 \times 2$  factorisation (5.3.1.4).

These factorisation functions use subroutines which will be defined in section 5.3.2.

The graph based factorisation is illustrated in an example figure 5.1, which shows the graph structure of  $\mathbf{D}$  and  $\mathbf{L}$  as they change through a small example factorisation sequence. During factorisation, both  $\mathbf{L}$  and  $\mathbf{D}$  exist simultaneously as different edge-sets in exactly the same set of vertex objects.





**Figure 5.1:** Graph based LDL factorisation example. The original system is contained within  $\mathbf{D}$  at the start with  $\mathbf{L} = \mathbf{I}$ , the identity system (each vertex isolated with a single unity loop). Each vertex is then factorised, in this case the ordering is  $[5, 2, 3, 4, 1]$ . Each factorisation of a vertex isolates that vertex from the rest of the system in  $\mathbf{D}$  and adds the outer-product marginal onto the remaining part of  $\mathbf{D}$ . Directed acyclic edges are added into  $\mathbf{L}$  corresponding to the sparsity structure and factorisation ordering.

### 5.3.1.1 Indefinite LDL Factorisation

Algorithm 4, `LDL_factorise_indefinite`, factorises a symmetric indefinite linear system. The inputs are:

- Reference to the graph  $\mathcal{G}$
- Keys referring to the edge-sets  $\mathcal{A}, \mathcal{L}$  and  $\mathcal{D}$ . Edgeset  $\mathcal{A}$  is equivalent to a symmetric indefinite linear system,  $\mathbf{A}$ . Edgeset  $\mathcal{L}$  is equivalent to the directed acyclic (triangular) factor,  $\mathbf{L}$ . Edgeset  $\mathcal{D}$  is equivalent to the symmetric block diagonal factor,  $\mathbf{D}$ .
- References to sets of vertices `roots` and `leaves` indicating the starting and finishing vertices of the factorisation.
- A function `pivotSelect` called to choose the factorisation vertices at each round.

The results are:

- The system in  $\mathcal{A}$  is destroyed. The created systems  $\mathbf{L}$  and  $\mathbf{D}$  satisfy  $\mathbf{LDL}^T = \mathbf{A}$ .

<b>Algorithm 4:</b> Sparse graph-theoretic indefinite $\mathbf{LDL}^T$ factorisation
--

<p><b>Name:</b> <code>LDL_factorise_indefinite</code></p> <p><b>while</b> <code>symEdges(<math>\mathcal{G}, \mathcal{A}</math>)</code> is not empty and <code>loops(<math>\mathcal{G}, \mathcal{A}</math>)</code> is not empty <b>do</b></p> <p>    <code>pivotSet = pivotSelect(<math>\mathcal{G}, \mathcal{A}</math>)</code></p> <p>    <b>if</b> <code>pivotSet.num() == 1</code> <b>then</b></p> <p>        <code>LDL_factorise_1x1(<math>\mathcal{G}, \mathcal{A}, \mathcal{L}, \mathcal{D}, \text{roots}, \text{leaves}, \text{pivotSet}</math>)</code></p> <p>    <b>else</b></p> <p>        <code>LDL_factorise_2x2(<math>\mathcal{G}, \mathcal{A}, \mathcal{L}, \mathcal{D}, \text{roots}, \text{leaves}, \text{pivotSet}</math>)</code></p>
---

### 5.3.1.2 Positive-Definite LDL Factorisation

Algorithm 5, `LDL_factorise_posdefinite`, factorises a symmetric positive definite linear system. The inputs and results are as for the indefinite case above:

### 5.3.1.3 Single-step LDL Factorisation - Scalar

Algorithm 6, `LDL_factorise_1x1`, performs a single step of the LDL factorisation for a single scalar pivot vertex. The inputs to this function are the graph  $\mathcal{G}$ , edge-sets

**Algorithm 5:** Sparse graph-theoretic positive definite  $\mathbf{LDL}^T$  factorisation

```

Name: LDL_factorise_posdefinite
while loops( $\mathcal{G}, \mathcal{A}$ ) is not empty do
  pivotSet = pivotSelect( $\mathcal{G}, \mathcal{A}$ )
  (pivotSet must return only a scalar pivot)
  LDL_factorise_1x1( $\mathcal{G}, \mathcal{A}, \mathcal{L}, \mathcal{D}$ , roots, leaves, pivotSet)

```

$\mathcal{A}, \mathcal{L}$  and  $\mathcal{D}$ , vertex sets **roots** and **leaves** and a single chosen pivot vertex **pvtx**.

The result is that the vertex is factorised out of **A** and into **L** and **D**, via:

$$\mathbf{L} += \mathbf{E}^{-1}\mathbf{C}^T \qquad \mathbf{D} += \mathbf{E} \qquad \mathbf{A} -= \mathbf{C}\mathbf{E}^{-1}\mathbf{C}^T$$

**Algorithm 6:** Single step of  $\mathbf{LDL}^T$  factorisation for a scalar pivot

```

Name: LDL_factorise_1x1
vertex pvtx = pivotSet.first
The pivot E comes from the sum of loop values:
  double E = getLoopVal(pvtx,  $\mathcal{A}$ )
if abs(E) < min tolerance then
  // Error, E too small
Add identity loops into L:
  add_loop( pvtx,  $\mathcal{G}, \mathcal{L}, 1.0$  )
Copy E into D:
  add_loop( pvtx,  $\mathcal{G}, \mathcal{D}, E$  )
Identify pvtx as a leaf and/or root:
  if undirEdges(pvtx,  $\mathcal{A}$ ) is empty then
    insert pvtx in leaves set
  if inEdges(pvtx,  $\mathcal{L}$ ) is empty then
    insert pvtx in roots set
Add  $\mathbf{E}^{-1}\mathbf{C}^T$  to L:
  adj_edges_copy_1x1( $\mathcal{G}, \text{pvtx}, \mathcal{A}, \mathcal{L}, 1.0/E$ , directed, undirEdges)
Add  $-\mathbf{C}\mathbf{E}^{-1}\mathbf{C}^T$  to A:
  outerproduct_1x1( $\mathcal{G}, \text{pvtx}, \text{NULL}, \mathcal{A}, \mathcal{A}, -1.0/E$ , undirEdges)
Clear factorised loops and edges from  $\mathcal{A}$ :
  clear_vertex( pvtx,  $\mathcal{A}$  )

```

### 5.3.1.4 Single-step LDL Factorisation - 2 by 2

Algorithm 7, `LDL_factorise_2x2`, performs a single step of the LDL factorisation for a  $2 \times 2$  pair of pivot vertices. The inputs to this function are the graph  $\mathcal{G}$ , edge-sets  $\mathcal{A}, \mathcal{L}$  and  $\mathcal{D}$ , vertex sets `roots` and `leaves` and a `pivotSet` containing the pair of vertices `pvtxA` and `pvtxB`.

The result is that the two vertices are factorised out of  $\mathbf{A}$  and into  $\mathbf{L}$  and  $\mathbf{D}$ , via:

$$\mathbf{L} += E^{-1}\mathbf{C}^T \quad \mathbf{D} += E \quad \mathbf{A} -= \mathbf{C}E^{-1}\mathbf{C}^T$$

This algorithm operates on the  $2 \times 2$  system  $\mathbf{E}$  which is symmetric indefinite. It uses analytical expressions for the determinant, inverse and eigenvalues of  $\mathbf{E}$  as follows:

$$\mathbf{E} = \begin{pmatrix} E_{00} & E_{01} \\ E_{01} & E_{11} \end{pmatrix} \quad (5.9)$$

$$\det \mathbf{E} = E_{00}E_{11} - E_{01}^2 \quad (5.10)$$

$$\mathbf{E}^{-1} = \frac{1}{\det(\mathbf{E})} \begin{pmatrix} E_{11} & -E_{01} \\ -E_{01} & E_{00} \end{pmatrix} \quad (5.11)$$

$$\text{eig } \mathbf{E} = \frac{E_{00} + E_{11} \pm \sqrt{E_{00}^2 + E_{11}^2 - 2E_{00}E_{11} + 4E_{01}^2}}{2} \quad (5.12)$$

$\mathbf{E}$  comes from `pivotSet_to_2x2Matrix` from the two vertices. This function finds and sums the loops of `pvtxA` and `pvtxB` and their cross-edges to form  $E_{00}$ ,  $E_{11}$  and  $E_{01}$  respectively.

**Algorithm 7:** Single step of  $\mathbf{LDL}^T$  factorisation for a  $2 \times 2$  pivot**Name:** `LDL_factorise_2x2``pivotSet` defines values for this  $2 \times 2$  pivot:

```

    vertex [pvtxA,pvtxB] = [pivotSet.first,pivotSet.second]
    double E00,E01,E11; pivotSet.to_2x2Matrix( pivotSet, E00, E01,
    E11)

```

**if** `min(abs(eig(E))) < min_abs_eig_E_tol` **then**

```

    break // Error, E too small

```

Form  $\mathbf{E}^{-1}$  analytically:

```

    double Einv00,Einv01,Einv11 = ...

```

Add identity loops into  $\mathbf{L}$ :

```

    add_loop( pvtxA, G, L, 1.0 ); add_loop( pvtxB, G, L, 1.0 )

```

Add  $\mathbf{E}$  into  $\mathbf{D}$ :

```

    add_loop( pvtxA, G, D, E00 )
    add_edge( pvtxA, pvtxB, G, D, E01, symmetric )
    add_loop( pvtxB, G, D, E11 )

```

Identify `pvtxA` and `pvtxB` as a leaf and/or root:(if the only edges are within this  $2 \times 2$  block then they form a  $2 \times 2$  leaf)**if** `undirEdges(pvtxA,A)` and `undirEdges(pvtxB,A)` are empty other than each other **then**

```

    insert pvtxA and pvtxB in leaves set

```

**if** `inEdges(pvtxA,A)` and `inEdges(pvtxB,A)` are empty other than each other **then**

```

    insert pvtxA and pvtxB in roots set

```

Add  $\mathbf{E}^{-1}\mathbf{C}^T$  to  $\mathbf{L}$ :

```

    adj_edges_copy_2x2
    (G,pvtxA,pvtxB,A,L,Einv00,Einv01,Einv11, directed, undirEdges)

```

Add  $-\mathbf{CE}^{-1}\mathbf{C}^T$  to  $\mathbf{A}$ :

```

    outerproduct_2x2
    (G,pvtxA,pvtxB,A,A,-Einv00,-Einv01,-Einv11, undirEdges)

```

Clear factorised loops and edges from  $\mathcal{A}$ :

```

    clear_vertex( pvtxA, A )
    clear_vertex( pvtxB, A )

```

### 5.3.2 Graph Based Linear Algebra Procedures

This section describes various general purpose graph based linear algebraic procedures which are used in the factorisation and solving algorithms. These functions are an important part of the contribution of this thesis because they show how the graph theoretic representation concept translates into practical operations in the graph for the solution algorithm.

#### Adjacent edges copy - scalar

This function computes  $\mathbf{A} += m\mathbf{C}^T$ , where  $\mathbf{C}$  is the set of adjacent edges to the given vertex, `pvtx`. See section 5.3.2.1.

#### Adjacent edges copy - $2 \times 2$

This function computes  $\mathbf{A} += \mathbf{M}\mathbf{C}^T$ , where  $\mathbf{C}$  is the pair of two sets of adjacent edges to the block of two vertices, `pvtxA` and `pvtxB`. See section 5.3.2.2.

#### The outer product - scalar

This function computes  $\mathbf{C}m\mathbf{C}^T$ , where a set of edges is equivalent to a sparse vector  $\mathbf{C}$ . See section 5.3.2.3.

#### The outer product - $2 \times 2$

This function computes  $\mathbf{C}\mathbf{M}\mathbf{C}^T$  where  $\mathbf{C}$  is the set of adjacent off-diagonals to a pair of two vertices equivalent to two sparse vectors. See section 5.3.2.4.

#### The outer product - off-diagonal

The “off-diagonal” outer product is so called because it is used in the above  $2 \times 2$  outer product to compute expressions  $\mathbf{C}_1\mathbf{M}_{12}\mathbf{C}_2^T$  and  $\mathbf{C}_2\mathbf{M}_{12}\mathbf{C}_1^T$ , which involve the off-diagonal parts of the  $2 \times 2$   $\mathbf{M}$ . This algorithm computes the general rectangular outer product,  $\mathbf{C}m\mathbf{D}^T$ , where  $\mathbf{C}$  and  $\mathbf{D}$  are the set of adjacent off-diagonals to each of a pair of two vertices. See section 5.3.2.5.

Each of these functions involves accessing a set of one or more adjacent edge ranges, which may be either the undirected edges or directed edges. Therefore these functions have an input argument `whichEdges` which is a templated function pointer used to obtain different types of edges for example `whichEdges` can be:

- `undirEdges`: specified when, for example, factorising a symmetric system.
- `inEdges` or `outEdges`: when re-constructing a symmetric system back from directed  $\mathbf{L}$  or  $\mathbf{L}^T$ .

### 5.3.2.1 Adjacent Edges Copy - Scalar

Algorithm 8 computes  $\mathbf{A} += m\mathbf{C}^T$ , where  $\mathbf{C}$  is the set of adjacent edges to the given vertex, `pvtx`.

The inputs are:

- Reference to a graph,  $\mathcal{G}$ .
- A vertex pointer `vtx`, which has the edges,  $\mathbf{C}$ .
- Keys for the source edge-set of  $\mathbf{C}$  (*src*) and destination edge-set of  $m\mathbf{C}^T$  (*dest*).
- The scalar  $m$ .
- A flag `dirsym` indicating if the resulting edges should be directed or symmetric.
- Function pointer `whichEdges` for which container to use to obtain  $\mathbf{C}$ .

The result is:

- $\mathbf{A} += m\mathbf{C}^T$ .

<b>Algorithm 8:</b> Adjacent Edges Copy, Scalar
---

<b>Name:</b> <code>adj_edges_copy_1x1</code>
--

<code>adjEdges = whichEdges( pvtx, src )</code>
---

<b>for</b> edge <code>Cedge</code> in <code>adjEdges</code> <b>do</b>
---

<table border="1"> <tr> <td> <code>adjVertex = other(Cedge, pvtx)</code> </td> </tr> </table>	<code>adjVertex = other(Cedge, pvtx)</code>
<code>adjVertex = other(Cedge, pvtx)</code>	

<table border="1"> <tr> <td> <code>add_edge(pvtx, adjVertex, <math>\mathcal{G}</math>, <i>dest</i>, <math>m * \text{Cedge} \rightarrow \text{val}()</math>, <code>directed</code>)</code> </td> </tr> </table>	<code>add_edge(pvtx, adjVertex, <math>\mathcal{G}</math>, <i>dest</i>, <math>m * \text{Cedge} \rightarrow \text{val}()</math>, <code>directed</code>)</code>
<code>add_edge(pvtx, adjVertex, <math>\mathcal{G}</math>, <i>dest</i>, <math>m * \text{Cedge} \rightarrow \text{val}()</math>, <code>directed</code>)</code>	

### 5.3.2.2 Adjacent Edges Copy - 2 by 2

Algorithm 9 computes  $\mathbf{A} += \mathbf{MC}^T$ , where  $\mathbf{C}$  is the pair of two sets of adjacent edges to the block of two vertices, `pvtxA` and `pvtxB`.

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_0 & \mathbf{C}_1 \end{bmatrix} \quad \mathbf{M} = \begin{bmatrix} M_{00} & M_{01} \\ M_{01} & M_{11} \end{bmatrix} \quad (5.13)$$

$$\mathbf{MC}^T = \begin{bmatrix} M_{00}\mathbf{C}_0^T + M_{01}\mathbf{C}_1^T \\ M_{01}\mathbf{C}_0^T + M_{11}\mathbf{C}_1^T \end{bmatrix} \quad (5.14)$$

The inputs are:

- Reference to a graph,  $\mathcal{G}$ .
- Vertex pointers `vtxA` and `vtxB`, which have the edges,  $\mathbf{C}_A$  and  $\mathbf{C}_B$ .
- Keys for the source edge-set of  $\mathbf{C}$  (*src*) and destination edge-set of  $\mathbf{CMC}^T$  (*dest*).
- The  $2 \times \mathbf{M}$  as  $M_{00}$ ,  $M_{01}$ ,  $M_{11}$ .
- A flag `dirsymb` indicating if the resulting edges should be directed or symmetric.
- Function pointer `whichEdges` for which container to use to obtain  $\mathbf{C}_A$  and  $\mathbf{C}_B$ .

The result is:

- $\mathbf{A} += \mathbf{MC}^T$ .

#### Algorithm 9: Adjacent Edges Copy $2 \times 2$

```

Name: adj_edges_copy_2x2
adjEdgesRangeA = whichEdges( pvtxA, src )
for edge CedgeA in adjEdgesRangeA do
    adjVertex = other(CedgeA, pvtxA)
    if adjVertex == pvtxB then continue
    add_edge(pvtxA, adjVertex,  $\mathcal{G}$ , dest, dirsymb, CedgeA->val()* $M_{00}$  )
    add_edge(pvtxB, adjVertex,  $\mathcal{G}$ , dest, dirsymb, CedgeA->val()* $M_{01}$  )
adjEdgesRangeB = whichEdges( pvtxB, src )
for edge CedgeB in adjEdgesRangeB do
    adjVertex = other(CedgeB, pvtxB)
    if adjVertex == pvtxA then continue
    add_edge(pvtxB, adjVertex,  $\mathcal{G}$ , dest, dirsymb, CedgeB->val()* $M_{11}$  )
    add_edge(pvtxA, adjVertex,  $\mathcal{G}$ , dest, dirsymb, CedgeB->val()* $M_{01}$  )

```



### 5.3.2.3 Symmetric Outer Product - Scalar

The scalar outer product, algorithm 10, forms the equivalent of a sparse matrix,  $\mathbf{C}m\mathbf{C}^T$  where  $\mathbf{C}$  is a set of edges, equivalent to a sparse vector. This is illustrated in figure 5.2.

The inputs are as follows.

- Reference to a graph,  $\mathcal{G}$ .
- A vertex pointer `vtxFrom`, which has the edges,  $\mathbf{C}$ .
- A vertex pointer `vtxExclude` for which to *exclude* edges to (or `NULL` for none)
- Keys for the source edge-set of  $\mathbf{C}$  (*src*) and destination edge-set of  $\mathbf{C}m\mathbf{C}^T$  (*dest*).
- The scalar  $m$ .
- Function pointer `whichEdges` for which container to use to obtain  $\mathbf{C}$ .

The result is:

- Performs  $\mathbf{A} += \mathbf{C}m\mathbf{C}^T$  where  $\mathbf{A}$  is the specified edge-set of *dest*. The added edges of the result are always symmetric, since this is a symmetric outer product. Only the loops (diagonals) and a single “half-triangle” of the edges need be computed and added, due to the symmetry.

### 5.3.2.4 Symmetric Outer Product - 2 by 2

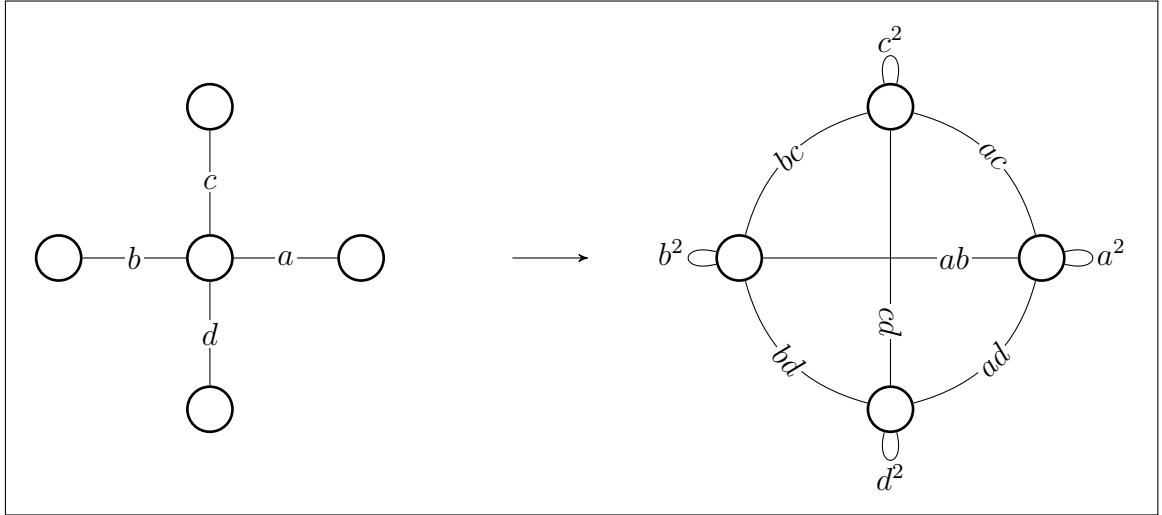
The  $2 \times 2$  outer product, algorithm 11, refers to computing  $\mathbf{C}\mathbf{M}\mathbf{C}^T$  where  $\mathbf{C}$  is the set of adjacent off-diagonals to a pair of two vertices equivalent to two sparse vectors.

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_0 & \mathbf{C}_1 \end{bmatrix} \quad (5.15)$$

$$\mathbf{M} = \begin{bmatrix} M_{00} & M_{01} \\ M_{01} & M_{11} \end{bmatrix} \quad (5.16)$$

$$\mathbf{C}\mathbf{M}\mathbf{C}^T = \mathbf{C}_0M_{00}\mathbf{C}_0^T + \mathbf{C}_1M_{11}\mathbf{C}_1^T + \mathbf{C}_0M_{01}\mathbf{C}_1^T + \mathbf{C}_1M_{01}\mathbf{C}_0^T \quad (5.17)$$

Equation 5.17 shows how this is broken down into scalar outer products of the form  $\mathbf{C}m\mathbf{C}^T$  (see above, section 5.3.2.3). and off-diagonal outer products of the form



**Figure 5.2:** Scalar outer product, graph form. The outer product takes a set of  $n$  edges from a given vertex,  $v$ , and forms the outer product consisting of  $(n^2 - n)/2$  edges and  $n$  loops. Where the set of edges is equivalent to a sparse vector  $\mathbf{C}$ , the outer product is the equivalent of a sparse matrix,  $\mathbf{CmC}^T$ .

$\mathbf{CmD}^T$  (see below, section 5.3.2.5).

The inputs are as follows:

- Reference to a graph,  $\mathcal{G}$ .
- Vertex pointers `vtxA` and `vtxB`, which have the edges,  $\mathbf{C}_A$  and  $\mathbf{C}_B$ .
- Keys for the source edge-set of  $\mathbf{C}$  (*src*) and destination edge-set of  $\mathbf{CmC}^T$  (*dest*).
- The  $2 \times \mathbf{M}$  as  $M_{00}$ ,  $M_{01}$ ,  $M_{11}$ .
- Function pointer `whichEdges` for which container to use to obtain  $\mathbf{C}_A$  and  $\mathbf{C}_B$ .

The result is as follows:

- $\mathbf{A} += \mathbf{CmC}^T$

### 5.3.2.5 Off-diagonal Outer Product

This “off-diagonal” outer product, algorithm 12, used in the above  $2 \times 2$  outer product to compute expressions  $\mathbf{C}_1 M_{12} \mathbf{C}_2^T$  and  $\mathbf{C}_2 M_{12} \mathbf{C}_1^T$ . See figure 5.3.

This algorithm forms the more general rectangular outer product  $\mathbf{CmD}^T$ .  $\mathbf{C}$  is the

**Algorithm 10:** Scalar Outer product, Graph Form

```

Name: outerprod_1x1
adjEdges = whichEdges( vtxFrom, key_src )
for each edge-iterator itorAdj in adjEdges do
    edge Cedge = *itorAdj
    adjVertex = other(Cedge, vtxFrom)
    if adjVertex == vtxExclude then continue

    ( Form the diagonal part of  $\mathbf{C}m\mathbf{C}^T$  (loops) )
    double val = Cedge->val() * m * Cedge->val()
    loop found = adjVertex->findLoop( (dest) )
    if none found then
        | add_loop ( adjVertex,  $\mathcal{G}$ , dest, val )
    else
        | found->val() += val

    (Form the single half of the off-diagonal part of  $\mathbf{C}m\mathbf{C}^T$  (edges) )
    for each edge-iterator itor2 from next(itorAdj) to end of adjEdges do
        edge Cedge2 = *itor2
        adjVertex2 = other(Cedge2, vtxFrom)
        if adjVertex2 == vtxExclude then continue
        double val = Cedge->val() * m * Cedge2->val()
        edge found = adjVertex->findUndirEdgeTo(adjVertex2, dest )
        if none found then
            | add_edge(adjVertex, adjVertex2,  $\mathcal{G}$ , dest, symmetric, val )
        else
            | found->val() += val

```

set of adjacent edges to one vertex  $\mathbf{vtxA}$  and  $\mathbf{D}$  is the set of adjacent edges to another vertex  $\mathbf{vtxB}$  (excluding those to each other).

The inputs are as follows:

- Reference to a graph,  $\mathcal{G}$ .
- Vertex pointers  $\mathbf{vtxA}$  and  $\mathbf{vtxB}$ , which have the edges,  $\mathbf{C}$  and  $\mathbf{D}$ .
- Keys for the source edge-set of  $\mathbf{C}$  (*src*) and destination edge-set of  $\mathbf{C}m\mathbf{C}^T$  (*dest*).
- The scalar  $m$ .
- Function pointer `whichEdges` for which container to use to obtain  $\mathbf{C}$  and  $\mathbf{D}$ .

The result is as follows:

**Algorithm 11:**  $2 \times 2$  Symmetric Outer Product (Graph Form).**Name:** `outerprod_2x2` $M_{00}$  outer product  $\mathbf{C}_0 M_{00} \mathbf{C}_0^T$ :└ `outerprod_1x1(grph, vtxA, vtxB, src, dest, M00)` $M_{11}$  outer product  $\mathbf{C}_1 M_{11} \mathbf{C}_1^T$ :└ `outerprod_1x1(grph, vtxB, vtxA, src, dest, M11)` $M_{01}$  outer products  $\mathbf{C}_0 M_{01} \mathbf{C}_1^T + \mathbf{C}_1 M_{01} \mathbf{C}_0^T$ :└ `outerprod_offdiag(grph, pvtxA, pvtxB, src, dest, M01 )`└ `outerprod_offdiag(grph, pvtxB, pvtxA, src, dest, M01 )`

- $\mathbf{A} += \mathbf{C} \mathbf{m} \mathbf{D}^T$

## 5.4 Linear Systems Solve Using the LDL Factorisation

The LDL factorisation helps solve linear systems ( $\mathbf{Ax} = \mathbf{b}$ ) by transforming  $\mathbf{A}$  into a product of triangular and diagonal systems, which are simpler to solve. Using the factorisation  $\mathbf{LDL}^T = \mathbf{A}$ , the solution involves a sequence of solves as follows:

To solve	$\mathbf{Ax} = \mathbf{b}$	for $\mathbf{x}$ ,	
using factorisation	$\mathbf{LDL}^T \mathbf{x} = \mathbf{b}$		
1. solve:	$\mathbf{Lu} = \mathbf{b}$	for $\mathbf{u}$ , where $\mathbf{u} = \mathbf{DL}^T \mathbf{x}$	▣
2. solve:	$\mathbf{Dr} = \mathbf{u}$	for $\mathbf{r}$ , where $\mathbf{r} = \mathbf{L}^T \mathbf{x}$	▣
3. solve:	$\mathbf{L}^T \mathbf{x} = \mathbf{r}$	for $\mathbf{x}$	▣

Where ▣ indicates a triangular (directed-acyclic) solve and ▣ indicates a block diagonal solve.

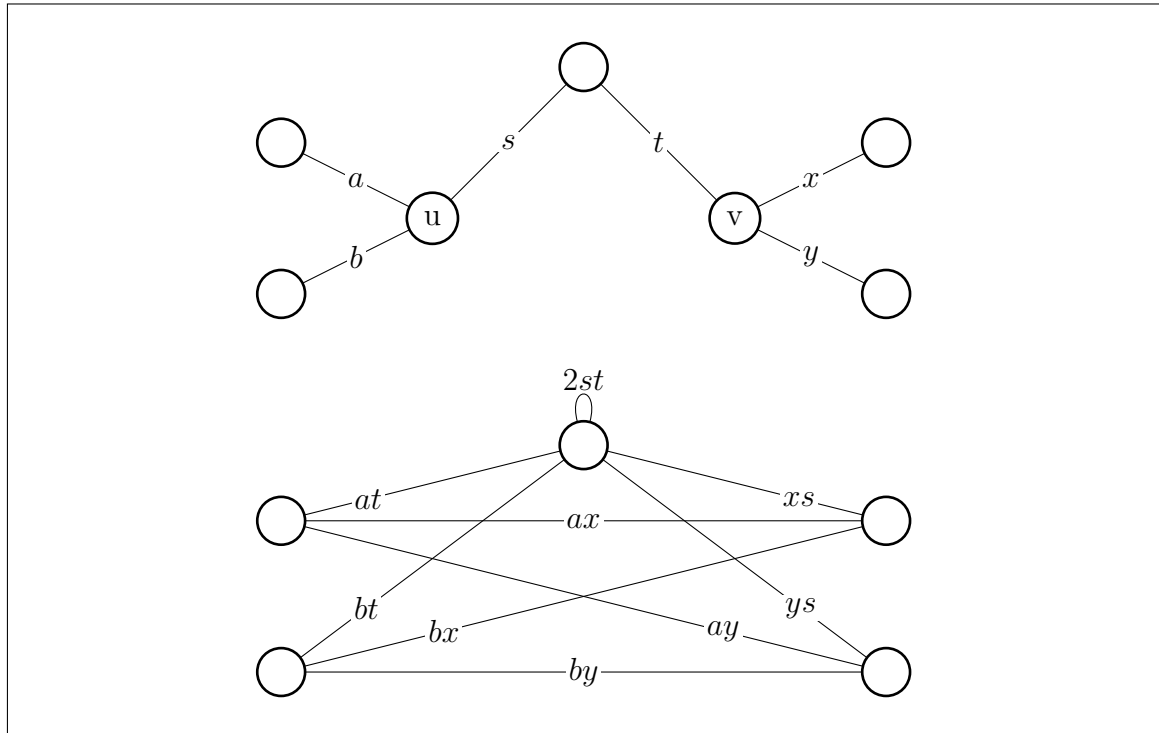
The block diagonal solve is explained in Section 5.4.1 and the triangular solve is explained in Section 5.4.2.

**Algorithm 12:** Off-Diagonal Outer product, Graph Form

```

Name: outerprod_offdiag
adjEdgesRangeA = whichEdges( pvtxA, src )
adjEdgesRangeB = whichEdges( pvtxB, src )
for edge-iterator itorAdj1 in adjEdgesRangeA do
    edge Cedge1 = *itorAdj1
    adjVertex1 = other(Cedge1, pvtxA)
    if adjVertex1 == pvtxB then continue
    for edge-iterator itorAdj2 in adjEdgesRangeB do
        edge Cedge2 = *itorAdj2
        adjVertex2 = other(Cedge2, pvtxB)
        if adjVertex2 == pvtxA then continue
        if adjVertex1 > adjVertex2 then (Due to symmetry of pairs,
            compute only for one half “triangle”)
            continue
        (Perform  $\mathbf{A} += \mathbf{C}m\mathbf{D}^T$ ;)
        double val = Cedge1->val()* M *Cedge2->val();
        if adjVertex1 == adjVertex2 then
            ( loop part )
            loop found = adjVertex1->findLoop( dest )
            if none found then
                | add_loop ( adjVertex1,  $\mathcal{G}$ , dest, val )
            else
                | found->val() += val
        else
            ( edges part )
            edge found = adjVertex1->findUndirEdgeTo(adjVertex2, dest)
            if none found then
                | add_edge(adjVertex1,adjVertex2, $\mathcal{G}$ ,dest,symmetric,val)
            else
                | found->val() += val

```



**Figure 5.3:** The off-diagonal outer product, graph form. This figure shows the result of  $\mathbf{CmD}^T + \mathbf{DmC}^T$  (two applications of off-diagonal outer product).  $\mathbf{C}$  and  $\mathbf{D}$  are two sparse vectors. This is part of the algorithm for computing the  $2 \times 2$  outer product as part of the  $2 \times 2$  pivot part, of the indefinite LDL factorisation algorithm.

Overall the process can be written as:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \quad (5.18)$$

$$\mathbf{x} = \mathbf{L}^{-T}\mathbf{D}^{-1}\mathbf{L}^{-1}\mathbf{b} \quad (5.19)$$

Where each matrix inversion is a notation to indicate the required solve stages rather than explicit inversion.

### 5.4.1 Graph Based Block Diagonal Solve

The block-diagonal solve solves a sub-problem:

$$\begin{array}{lll} \text{Solve:} & \mathbf{D}\mathbf{r} = \mathbf{u} & \text{for } \mathbf{r} \\ \text{Solution:} & \mathbf{r} = \mathbf{D}^{-1}\mathbf{u} & \end{array}$$

$\mathbf{D}$  is block diagonal, with either scalar or  $2 \times 2$  diagonal blocks. That is, the graph edge-set  $\mathbf{D}$  consists of multiple isolated components in groups of size 1 or 2 vertices. The  $2 \times 2$  diagonal blocks can occur because the  $\mathbf{LDL}^T$  factorisation operates with symmetric indefinite matrices, as explained in section 5.2.1.

Since  $\mathbf{D}$  is block diagonal, the overall  $\mathbf{D}$  solve consists simply of repeated solves in  $1 \times 1$  or  $2 \times 2$  variables.

- The scalar  $\mathbf{D}$  solve consists of a scalar division at each scalar vertex in  $\mathbf{D}$ .
- The  $2 \times 2$   $\mathbf{D}$  solve consists of analytical symmetric  $2 \times 2$  matrix inversion and product at each  $2 \times 2$  cluster of  $\mathbf{D}$ .

### 5.4.2 Graph Based Triangular Solve

The triangular-solve is used to solve systems in both  $\mathbf{L}$  and  $\mathbf{L}^T$ , depending on the stage of the  $\mathbf{LDL}^T$  solve, where  $\mathbf{L}$  is a directed-acyclic edge-set, equivalent to a square

triangular matrix. The difference between solving with  $\mathbf{L}$  and solving with  $\mathbf{L}^T$  is a matter of interpretation of the direction of the directed edges.

**Example 5.2.** 

---

**$\mathbf{L}$  versus  $\mathbf{L}^T$ , forward versus backward directions for directed-acyclic (triangular) solving**

Solving  $\mathbf{L}\mathbf{x} = \mathbf{b}$  with  $\mathbf{L}$  from Figure 4.2 would involve starting from  $x_1$  or  $x_2$  and solving “forwards” following along the direction of the edges indicated, resulting with  $x_4$  or  $x_5$  solved last.

Solving  $\mathbf{L}^T\mathbf{x} = \mathbf{b}$  (the transposed problem) with the same  $\mathbf{L}$  (Figure 4.2) would involve starting from  $x_4$  or  $x_5$  and solving “backwards” following against the direction of the edges indicated, resulting with  $x_1$  or  $x_2$  solved last.

Since these two possibilities exist, note that neither is a preferred direction over the other, and it is a matter of convention as to which is regarded as the “forward” and which is regarded as the “backward” direction and which is the “direct” and which is the “transposed” system.

---

Since  $\mathbf{L}$  is *directed* each vertex has distinct *input* and *output* edges. Since  $\mathbf{L}$  is *acyclic*, the solution for each vertex is uniquely and analytically determined from the solutions from other vertices through the *input* edges and is not at all affected by the solution results for vertices through the *output* edges.

The input and output edges are shown by example in Figure 5.4. Each variable,  $x_i$ , is solved as follows:

$$\text{Solve:} \quad L_{ii}x_i + \sum_{j \in \pi(x_i)} L_{ij}x_j = b \quad (5.20)$$

$$\text{Solution:} \quad x_i = L_{ii}^{-1}(b - \sum_{j \in \pi(x_i)} L_{ij}x_j) \quad (5.21)$$

The solution in Equation 5.21 requires that each of the input  $\mathbf{x}_i$  are solved already. Therefore the solutions must be obtained in a topological order, ensuring that the



required input values are always computed before being required for subsequent calculations.

Algorithm 13 solves  $\mathbf{L}\mathbf{x} = \mathbf{b}$  for  $\mathbf{x}$  for a single vertex, assuming that its input edges (if any) are solved. By swapping the definition for *backward* versus *forward* edges, the algorithm is able to swap solving in  $\mathbf{L}^T$  for solving in  $\mathbf{L}$ .

**Algorithm 13:** Graph based single vertex triangular solve [37]

**Input:** Graph  $\mathcal{G}$ , edge-set key  $\mathcal{L}$   
**Input:** Functions defining *backward* vs. *forward* edges. ( Swapping these solves in  $\mathbf{L}^T$  vs.  $\mathbf{L}$  )  
**Result:** Solve  $\mathbf{L}\mathbf{x} = \mathbf{b}$  for a particular vertex,  $\text{vtx}$   
 $\text{vtx.x} = \text{vtx.b}$   
**for** each **edge** in the *backward* edges of  $\mathcal{L}$  in  $\mathcal{G}$  **do**  
    other input vertex,  $\text{vtx\_input} = \text{other}(\text{edge}, \text{vtx})$   
     $\text{vtx.x} -= \text{edge} \rightarrow \text{val}() * \text{vtx\_input.x}$   
 $\text{vtx.x} /= \text{getLoopVal}(\text{vtx}, \mathcal{G}, \mathcal{L})$   
Mark  $\text{vtx}$  solved

### 5.4.3 Graph Based Solve Implementation

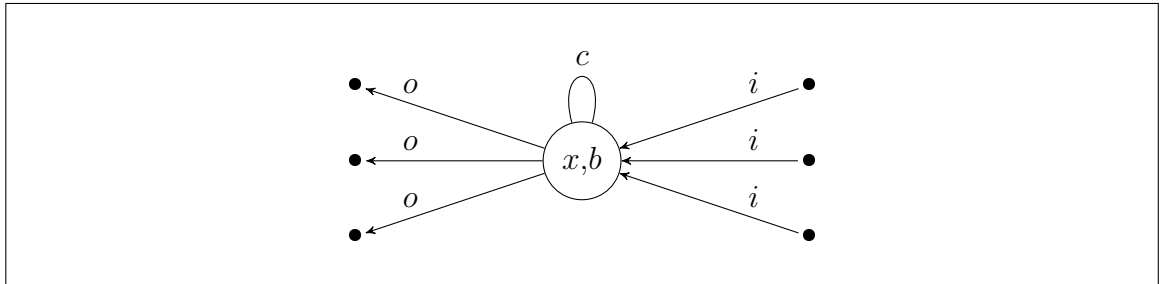
#### 5.4.3.1 Block Diagonal Solve

Algorithm 14 performs the multiple block diagonal  $\mathbf{D}$  solve using  $\mathbf{L}$  as a guide for the sequence and limiting to the region specified by **currents** and **finals**. The inputs are:

- The graph reference,  $\mathcal{G}$
- The edge-set keys  $\mathcal{L}$  and  $\mathcal{D}$
- Member data pointers **b** and **x** to  $\mathbf{b}$  and  $\mathbf{x}$  to identify each entry  $b$  and  $x$  within each vertex.
- An integer tag, **solvetag**, which is different for each new solving round.
- Function pointer **forwardEdges** specifies the direction in which to move along  $\mathbf{L}$ .
- The set of vertices **currents**, which are the *start* of the solving region.

$$\begin{array}{c}
 \mathbf{L} \quad \mathbf{x} = \mathbf{b} \\
 \left[ \begin{array}{ccccccc}
 \bullet & & & & & & \\
 \bullet & \bullet & & & & & \\
 \bullet & \bullet & \bullet & & & & \\
 i & i & i & c & & & \\
 \bullet & \bullet & \bullet & o & \bullet & & \\
 \bullet & \bullet & \bullet & o & \bullet & \bullet & \\
 \bullet & \bullet & \bullet & o & \bullet & \bullet & \bullet
 \end{array} \right]
 \left[ \begin{array}{c}
 \bullet \\
 \bullet \\
 \bullet \\
 x_i \\
 \bullet \\
 \bullet \\
 \bullet
 \end{array} \right]
 =
 \left[ \begin{array}{c}
 \bullet \\
 \bullet \\
 \bullet \\
 b \\
 \bullet \\
 \bullet \\
 \bullet
 \end{array} \right]
 \end{array}$$

(a) Matrix form. The input dependencies in  $\mathbf{L}$  exist within the same row as  $x_i$  and the output dependencies exist in the same column as  $x_i$ .



(b) Graph form. The in edges ( $i$ ) and out edges ( $o$ ) model the input and output dependencies in the directed-acyclic (triangular) linear system  $\mathbf{L}$ .

**Figure 5.4:** In vs. out edges for triangular (acyclic) systems. The single current variable,  $x_i$ , is computed to solve:  $L_{ii}x_i + \sum_{j \in \pi(x_i)} L_{ij}x_j = b$ . This requires the input of preceding variables,  $x_j$ , multiplied through input edges,  $L_{ij}$ , (marked  $i$  in the figure).  $j$  ranges through all preceding variables, the parent variables of  $x_i$  that is,  $j \in \pi(x_i)$ . The current variable,  $x_i$ , is used to compute other subsequent variables via the *output* edges (marked  $o$ )

- The set of vertices **finals**, which are the *end* of the solving region.

Algorithm 15 is the subroutine for the solution of just a single  $1 \times 1$  or  $2 \times 2$  block of

**D**. The inputs are:

- The graph reference,  $\mathcal{G}$
- The pair or single vertex **vtxs**
- The edge-set key  $\mathcal{D}$
- Member data pointers **b** and **x** to **b** and **x** to identify each entry  $b$  and  $x$  within each vertex.
- The integer tag, **solvetag**

Algorithm 15 uses the analytical  $2 \times 2$  inverse:

$$\det \mathbf{D} = -D_{01}^2 + D_{00}D_{11} \quad (5.22)$$

$$\mathbf{D}^{-1} = \begin{pmatrix} D_{00}^{-1} & D_{01}^{-1} \\ D_{01}^{-1} & D_{11}^{-1} \end{pmatrix} = \frac{1}{\det \mathbf{D}} \begin{pmatrix} D_{11} & -D_{01} \\ -D_{01} & D_{00} \end{pmatrix} \quad (5.23)$$

$$\mathbf{x} = \mathbf{D}^{-1} \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} D_{00}^{-1}b_0 + D_{01}^{-1}b_1 \\ D_{01}^{-1}b_0 + D_{11}^{-1}b_1 \end{pmatrix} \quad (5.24)$$

**Algorithm 14:** Graph based multiple **D** solve

```

while currents is not empty do
  vtx = *currents.begin()
  if vtx->solvetag != solvetag then
    (find_D_block obtains either the  $1 \times 1$  or  $2 \times 2$  starting from vtx )
    vertex pair vtxs = find_D_block(  $\mathcal{G}$ , vtx,  $\mathcal{D}$  )
    D_solve_single(  $\mathcal{G}$ , vtxs,  $\mathcal{D}$ , x, b, solvetag)

    find vtx in finals
    if not found then
      (vtx is not on the finals boundary, continue the solving)
      for each edge edesc in forwardEdges( vtx,  $\mathcal{L}$ ) do
        nextVtx = other(edesc,vtx)
        if nextVtx->solvetag != solvetag then
          (queue-in the next vertex)
          currents.insert ( nextVtx )

    if vtxs.num() == 1 then
      | currents.erase( vtx )
    else
      (erase both vertices in vtxs from current)
      currents.erase( vtxs.first )
      currents.erase( vtxs.second )
  else
    (already solved)
    currents.erase(vtx)

```

**Algorithm 15:** Graph based single **D** solve

```

if vtxs.num()==1 then
  (Solving  $Dx = b$  for scalar  $D$   $x = b/D$ )
  double D = getLoopVal(vtxs.first, $\mathcal{G}$ , $\mathcal{D}$ )
  if abs(D) < abs_D_tol then
    | (no change)
  else
    | vtx->*x = vtx->*b / D
  (mark as solved)
  vtx->solvetag = solvetag
else
  (Solving  $\mathbf{D}\mathbf{x} = \mathbf{b}$  for  $2 \times 2\mathbf{D}$ )
  double D00,D01,D11
  chosen_to_2x2Matrix( vtxs, D00, D01, D11)

  ( get  $\mathbf{D}^{-1}$  explicit  $2 \times 2$  inverse)
  double DetD = -D01*D01+D00*D11
  if abs(DetD) > abs_D_tol then
    | (no change)
  else
    double Dinv00 = +D11/DetD
    double Dinv01 = -D01/DetD
    double Dinv11 = +D00/DetD

    double b0 = vtx0->*b
    double b1 = vtx1->*b

    vtx0->*x = Dinv00 * b0 + Dinv01 * b1
    | vtx1->*x = Dinv01 * b0 + Dinv11 * b1
  (mark as solved)
  vtx0->solvetag = solvetag
  | vtx1->solvetag = solvetag

```

### 5.4.3.2 Triangular Solve

Algorithm 16 solves  $\mathbf{L}\mathbf{x} = \mathbf{b}$  or  $\mathbf{L}^T\mathbf{x} = \mathbf{b}$ .

The inputs:

- Graph reference,  $\mathcal{G}$
- Edge-set key,  $\mathcal{L}$
- Member data pointers  $\mathbf{b}$  and  $\mathbf{x}$  to  $\mathbf{b}$  and  $\mathbf{x}$  to identify each entry  $b$  and  $x$  within each vertex.
- An integer tag, `solvetag`, which is different for each new solving round.
- Function pointers `forwardEdges` and `backwardEdges`. Specifying `forwardEdges = outEdges` and `backwardEdges = inEdges` corresponds to solving  $\mathbf{L}\mathbf{x} = \mathbf{b}$ . Swapping these swaps the solve for  $\mathbf{L}$  into  $\mathbf{L}^T$ .
- The set of vertices `currents`, which are the *start* of the solving region.
- The set of vertices `finals`, which are the *end* of the solving region.

The result is:

- Entries  $x$  in each vertex are filled in with the solution to  $\mathbf{L}\mathbf{x} = \mathbf{b}$  or  $\mathbf{L}^T\mathbf{x} = \mathbf{b}$ .

Algorithm 16 uses sets of vertices to specify the starting and ending vertices to be solved. The starting vertices, `currents` is given the `root` vertices from the LDL factorisation.

To compute *all* vertices, the argument `finals` can be given the `leaves` vertices from the LDL factorisation (or left empty). *Leaves* and *roots* are the vertices at the extremities of the directed-acyclic graph.

In this manner, the algorithm can retain control of *which* variables it back-computes. This controls the *extent* of the triangular solve. Although computation must begin at the appropriate root of the triangular system, it does not have to continue through to all variables but can stop at any vertex and leave the downstream vertices unsolved.

The `current` and `final` sets form a *boundary* defining a selection of vertices. The selection of vertices then consists of all vertices reachable “upstream” on the directed

acyclic graph from the “downstream” boundary. This is a compact way to represent a selection of vertices, which is consistent with the fork and branch structure of the acyclic graph, and the corresponding logic of which vertices must be solved first in order to solve subsequent vertices.

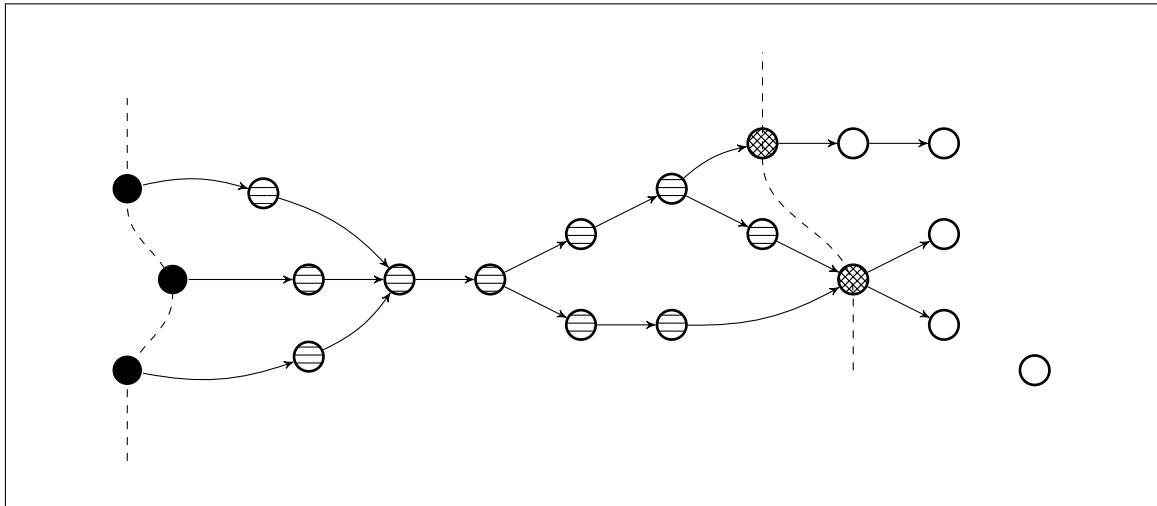
### Roots

The roots are those vertices with no in edges. In Figure 4.2  $x_1$  and  $x_2$  are roots.

### Leaves

The leaves are those vertices with no out edges. In Figure 4.2  $x_4$  and  $x_5$  are leaves.

A diagram of acyclic graph, showing the leaf and root boundaries is given in figure 5.5.



**Figure 5.5:** Acyclic graph root and leaf boundaries.

Vertices  $\bullet$  indicate root vertices, marking the start of the region to be solved.

Vertices  $\ominus$  indicate vertices which are solved, being in the selected region.

Vertices  $\otimes$  indicate leaf vertices, marking the end of the region to be solved.

Vertices  $\circ$  indicate vertices not solved, being beyond the selected region.

### 5.4.3.3 Solve Sequence

The solve sequence from section 5.4 is performed as follows:

1. Solve  $\mathbf{L}\mathbf{u} = \mathbf{b}$ :

```
Tri_solve( $\mathcal{G}$ ,  $\mathcal{L}$ ,  $\mathbf{b}$ ,  $\mathbf{u}$ , 1, outEdges, inEdges, roots, leaves);
```

2. Solve  $\mathbf{D}\mathbf{r} = \mathbf{u}$ :

```
D_solve_multi( $\mathcal{G}$ ,  $\mathcal{L}$ ,  $\mathcal{D}$ ,  $\mathbf{r}$ ,  $\mathbf{u}$ , 2, outEdges, roots, leaves );
```

3. Solve  $\mathbf{L}^T \mathbf{x} = \mathbf{r}$ :

```
Tri_solve( $\mathcal{G}$ ,  $\mathcal{L}$ ,  $\mathbf{r}$ ,  $\mathbf{x}$ , 3, inEdges, outEdges, leaves, roots);
```

The specification of  $\mathbf{L}$  versus  $\mathbf{L}^T$  involves simply swapping the arguments for forward and backward edges and for the starting and ending vertex sets, compared to the roots and leaves obtained from the original LDL factorisation.

## 5.5 Reconstruction From LDL Factorisation

This section describes how to re-compute  $\mathbf{A}$  given the factorisation  $\mathbf{L}$  and  $\mathbf{D}$ . The reconstruction of  $\mathbf{A}$  is obtained by computing  $\mathbf{A} = \mathbf{LDL}^T$ .

This section describes how this computation is expanded and applied to the graph structure.

Consider a permutation and partitioning of  $\mathbf{L}$  and  $\mathbf{D}$  and the corresponding expansion of  $\mathbf{LDL}^T$ :

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_{11} & \\ & \mathbf{D}_{22} \end{bmatrix} \quad \mathbf{L} = \begin{bmatrix} \mathbf{I}_{11} & \\ \mathbf{L}_{12} & \mathbf{L}_{22} \end{bmatrix} \quad (5.25)$$

$$\mathbf{LDL}^T = \begin{bmatrix} \mathbf{D}_{11} & \mathbf{D}_{11}\mathbf{L}_{12}^T \\ \mathbf{L}_{12}\mathbf{D}_{11} & \mathbf{L}_{12}\mathbf{D}_{11}\mathbf{L}_{12}^T + \mathbf{L}_{22}\mathbf{D}_{22}\mathbf{L}_{22}^T \end{bmatrix} \quad (5.26)$$

Equation 5.26 indicates that the expansion is obtained as follows:

- Start with empty  $\mathbf{A}$



**Algorithm 16:** Graph based triangular solve

```

while currents is not empty do
  vtx = *currents.begin()
  vtx->*x = vtx->*b
  for each edge edgePrev in backwardEdges(vtx,  $\mathcal{L}$ ) do
    vtxPrev = other(edgePrev, vtx)
    vtx->*x -= edgePrev->val() * vtxPrev->*x
  vtx->*x /= getLoopVal(vtx,  $\mathcal{G}$ ,  $\mathcal{L}$ )
  vtx->solvetag = solvetag

  find vtx in finals
  if found then
    (queue-in the solving of the solvable dependencies)
    for each edge edgeNext in forwardEdges(vtx,  $\mathcal{L}$ ) do
      (vtxNext is an output dependent of vtx)
      vtxNext = other(edgeNext, vtx)
      bool all_inputs_of_next_are_solved = true
      for each edge eDepNextVtx in backwardEdges(vtxNext,  $\mathcal{L}$ ) do
        (depVtxNext is an input dependency of vtxNext)
        depVtxNext = other(eDepNextVtx, vtxNext)
        if depVtxNext->solvetag != solvetag then
          all_inputs_of_next_are_solved = false
          break
      if all_inputs_of_next_are_solved then
        (queue-in vtxNext for solution soon )
        currents.insert( vtxNext )
      else
        (ignore vtxNext, it will get considered again by one of its other
         input dependencies)
    else
      (dependencies are beyond the region to solve)
      (done with vtx)
      currents.erase(vtx)

```

- For each block of  $\mathbf{D}$ :
  1. Collect the current  $\mathbf{D}$  diagonal block,  $\mathbf{D}_{11}$ , into  $\mathbf{A}$ . In the graph this corresponds to accessing the immediate loops and cross edge for the scalar or  $2 \times 2$  block of  $\mathbf{D}$ .
  2. Collect the off-diagonal structure  $\mathbf{L}_{12}$ . In the graph, the off-diagonal  $\mathbf{L}_{12}$  corresponds to the out-edges of the vertices in the current  $\mathbf{D}$  block.
  3. Compute  $\mathbf{L}_{12}\mathbf{D}_{11}$  into  $\mathbf{A}$ . This also implies  $\mathbf{D}_{11}\mathbf{L}_{12}^T$  in  $\mathbf{A}$ , but does not need to be stored since it is simply the symmetrical transpose.
  4. Compute the outer-product  $\mathbf{L}_{12}\mathbf{D}_{11}\mathbf{L}_{12}^T$  into  $\mathbf{A}$ .

## 5.6 Discussion

### 5.6.1 Relation to the Junction Tree Algorithm

Junction-tree [56, 57] is a parallel direct solving approach. Junction-tree can be considered as both an *ordering algorithm* and a *solving algorithm*:

#### Ordering

The junction-tree algorithm applies a *tree decomposition* to the graph. This creates a tree of *clusters* of variables. Each node in the tree is a cluster, each edge in the tree represents the intersection of the two clusters. The *width* of the junction tree is the dimension of the largest cluster minus one. The *treewidth* of the original problem is the minimum width among all possible junction trees for the problem.

#### Solving

The factorisation ordering referred to above is defined implicitly by the tree structure. The solving process is able to begin in parallel at each *leaf* of the tree structure.

The junction-tree solving algorithm is a generalisation of the outer-product marginalisation operation. It utilises the tree structure and re-uses computations

in order to arrive at the marginals for each variable-set. In effect, the marginal for a given variable-set is computed via successive marginalisation, starting from the outer leaf variable-sets inwards. When this process is required for all the variable-sets, most calculations appear in common. The junction tree algorithm organises itself such that these common calculations are not repeated but are re-used. The process of marginalising each variable-set onto its subsequent variable-sets (across the junction-set) is formulated in a message-passing style. Refer to [41, 56, 57] for further details. The junction-tree ordering can also be used in the “factorise & backsolve” manner described in this chapter [57].

The junction-tree solving algorithm requires solving multiple sub-problems, each in the dimension of the local cluster size. The solution process takes  $O(n \cdot w^3)$  time for  $n$  clusters, the largest of size  $w$ . Thus the key to the efficiency of the junction tree algorithm is in finding a sufficiently thin (small width) junction tree for the problem.

The junction tree algorithm focusses on problems which can be reduced into a sufficiently thin tree structure, where the clusters within the tree then consist of small, dense subproblems. The junction tree algorithm focusses on partial marginal message passing on the tree.

The approach described in this chapter is applicable to general structures, not necessarily decomposed into a tree. The focus of this algorithm is in successive scalar factorisation in the graph structure.

Both approaches are highly complementary. For problems which can be decomposed into a tree, such a tree will offer efficient factorisation ordering for any exact solving algorithm, including the algorithm of this chapter. Furthermore, the graph based linear algebraic operations described in this chapter could be used to implement a junction-tree based solving algorithm.

## 5.7 Future Research

### 5.7.1 Factorisation Approach

The factorisation algorithm presented in this chapter used the outer-product LDL factorisation algorithm.

An improved implementation of the graph based factorisation algorithms will also incorporate a sparse triangular solve based algorithm, operating in the graph structure. This will make the graph based approach more comparable with the conventional matrix approaches and may lead to a more competitive implementation.

The sparse triangular solve algorithm is used in typical sparse matrix implementations[16]. The outer-product algorithm shares more in common with methods in the estimation literature.

The graph based representation and factorisation approach of this thesis achieves a number of fundamental changes, such as decoupling of the representation and factorisation algorithm from storage and indexing considerations. The graph representation is significantly different from existing sparse matrix representations. These considerations make the task of adapting existing factorisation algorithms fairly difficult. The factorisation algorithm described in this chapter achieves the goal of presenting a full, sparse, graph-based factorisation and solving algorithm.

However, it remains for future research to adapt into the graph form some of the more advanced state-of-the-art algorithms in sparse matrix solving. These include multifrontal, supernodal, parallel and out-of-core approaches [17].

Since the graph approach is flexible to modifications and re-ordering it is expected, in future research, to be well suited to online, adaptive factorisation algorithms suited for realtime applications in localisation and mapping.

### 5.7.2 Factorisation Ordering Choice

This section discusses the problem of choosing a factorisation ordering in the context of the graph based LDL factorisation algorithm. The choice of the order in which variables are factorised is important because it affects the computation complexity and numerical stability of the factorisation and solution process [17, 18, 37, 55].

Conventionally, ordering is performed either for improvement of sparsity [17] or for improvement of numerical stability [37]. In the special case of positive definite linear systems, the aspect of ordering for numerical stability can be ignored while still guaranteeing completion of the factorisation. Therefore implementations usually focus on factorisation ordering for sparsity. Choosing a factorisation order to optimise sparsity is an NP-complete problem [30]. Adding the issue of considering factorisation ordering for numerical stability makes the problem more complex.

This thesis does not propose new factorisation ordering algorithms. Instead, this section discusses how existing methods for the ordering for sparsity and ordering for numerical stability fit in with the graph embedded approach.

The factorisation ordering was discussed in section 3.7 in relation to the augmented system form. Section 3.7 argued that the augmented system form is beneficial because it allows the factorisation ordering to mix between observations and states. The mixing of observations and states in the factorisation ordering allows improved sparsity and numerical stability of the factorisation under certain conditions, compared to the fixed approach of factorising (or eliminating) the observations first (equivalent to the information form).

The factorisation ordering was also discussed in section 5.2 in relation to how the graph based representation helps to allow more flexible factorisation ordering.

The algorithms 4 and 5 contain lines `pivotSet = pivotSelect( $\mathcal{G}, \mathcal{A}$ )`. The approaches discussed in this section provide the function `pivotSelect`.

The following sections describe factorisation ordering approaches for

- Numerical stability

- Sparsity
- Online modification problems

The problem for future research is to incorporate these together to give practical algorithms applicable to online localisation and mapping, which requires a complex mix of requirements on the algorithm: Fast and accurate performance in a realtime context.

### 5.7.2.1 Factorisation Ordering for Numerical Stability

This section discusses factorisation ordering methods for numerical stability.

Numerical stability considerations of the factorisation ordering were also discussed in section 3.7.2 in relation to the ability of the augmented system form to handle constraints and tight-observations.

As discussed in section 3.7.2 and [37] discuss, the basic consideration in the choice of factorisation ordering for numerical stability is to avoid or defer factorisation of *small*  $\mathbf{E}$  variables. The resulting factors involve  $\mathbf{E}^{-1}$  which causes *large* entries in  $\mathbf{L}$  or can halt the progress of the factorisation, if  $\mathbf{E}$  is zero or singular. The factorisation for numerical stability is critical when the system contains constraints or states with zero prior information, since these have zeros on the diagonals (loops) of the augmented system form.

For indefinite systems, the `pivotSelect` function is able to choose either a  $2 \times 2$  pivot from a pair of vertices or a single scalar pivot. This process is based on the Bunch-Parlett algorithm [14] and [37, pg 168]).

The Bunch-Parlett algorithm aims to choose the largest  $\mathbf{E}$  in Equations 5.4 and 5.2 such that  $\mathbf{E}^{-1}$  exists and is small. The aim is to maintain the entries in  $\mathbf{L}$  to be less than or near unit magnitude and the entries in  $\mathbf{B} - \mathbf{C}\mathbf{E}^{-1}\mathbf{C}^T$  to be sufficiently bounded. This algorithm also balances between selecting single and  $2 \times 2$  pivots.

The Bunch-Parlett algorithm requires a search for the following:

$$\mu_{\text{all}} = \max_{i,j} |a_{ij}| \quad (5.27)$$

$$\mu_{\text{diag}} = \max_i |a_{ii}| \quad (5.28)$$

**Algorithm 17:** Bunch-Parlett pivot selection strategy ([14] and [37, pg 168])

**Name:** pivots\_BunchParlett

**Input:** A Graph  $\mathcal{G}$

**Input:** An edge-set  $\mathbf{A}$  for analysing numerical properties of the edges. The chosen pivots are vertices linked to edges in  $\mathbf{A}$

**Result:** Choice of 1 (pvtx) or 2 (pvtxA and pvtxB) vertices for factorisation.

**begin**

Find  $\mu_{\text{all}}$  as  $\max_{i,j} |a_{ij}|$  where  $a_{ij}$  is the edge value of any edge or loop in edge-set  $\mathbf{A}$

Find  $\mu_{\text{diag}}$  as  $\max_i |a_{ii}|$  where  $a_{ii}$  is the loop value of any loop in edge-set  $\mathbf{A}$

$\alpha = \frac{1+\sqrt{17}}{8} \approx 0.6404$

**if**  $\mu_{\text{diag}} \geq \alpha \mu_{\text{all}}$  **then**

Choose a single vertex pvtx

pvtx is the source of any loop with value  $\mu_{\text{diag}}$

**else**

Choose two vertices pvtxA and pvtxB

pvtxA is the source of any edge with value  $\mu_{\text{all}}$

pvtxB is the target of any edge with value  $\mu_{\text{all}}$

**end**

- When a large enough loop entry exists, the algorithm will choose it as a scalar pivot.
- When the diagonal (loop) entries are all zero, the algorithm will choose a  $2 \times 2$  pivot which has the largest off-diagonal.
- When the system is of rank 1 (for example all entries equalling 1), the algorithm chooses a scalar pivot. (No invertible  $2 \times 2$   $\mathbf{E}$  exists).
- The algorithm specifies a search and selection of the maximum  $\mu$  values over all edges and loops. In future work it will be necessary to limit such a search to

find only *sufficiently* large  $\mu$  values or maximum values over a limited search, in order to maintain fast performance.

While this algorithm is sufficient for the choice of factorisation for numerical stability, it remains a problem for future research to incorporate this with the factorisation for sparsity and online modification.

### 5.7.2.2 Factorisation Ordering for Sparsity

Choosing a factorisation order for minimum fill-in to optimise sparsity is an NP-complete problem [30]. This thesis does not propose new factorisation ordering algorithms.

The key principle in the selection of the factorisation ordering for sparsity is to avoid or defer the factorisation of vertices with large *degree*.

Section 3.7.1 discussed the factorisation ordering for sparsity in terms of the flexibility offered by the augmented system form in particular.

The typical approach of [4, 16, 67] to the factorisation of positive definite systems is to:

1. Pre-plan the factorisation order for minimum fill-in
2. Perform a *symbolic* factorisation necessary to pre-plan the matrix nonzero structure and pre-allocate the memory required during the factorisation
3. Perform the numerical factorisation.

However, for the indefinite or positive-semi-definite systems considered here, the factorisation for numerical stability makes unpredictable changes to the factorisation order which are difficult to pre-plan. Fortunately, the graph based representation removes the need to pre-allocate the matrix nonzero structure and memory.

Therefore this section presents a basic minimum degree ordering approach. This is based on the property that in the graph representation, the algorithm can evaluate



**Algorithm 18:** Explicit minimum-degree pivot selection

**Input:** A Graph  $\mathcal{G}$   
**Input:** An edge-set  $src$  for analysing degree properties of the edges. The chosen pivots are vertices linked to edges in  $src$   
**Result:** Choice of a vertex ( $pvtx$ ) for factorisation.

```

minDegree = MAX_INT
minDegreeVertex = NULL
int n = 0
for each loop in loops( $\mathcal{G}, src$ ) do
    vertex = sourcetarget(loop)
    degree = unDegree(vertex,  $src$ )
    if degree < minDegree then
        minDegree = degree
        minDegreeVertex = vertex
    if minDegree <= 2 then
        break
    ++n
    if n >= Nmax then
        break

```

the degree counts of the vertices during factorisation and hence choose the next vertex for factorisation.

This algorithm selects the minimum degree vertex from the next  $N_{max}$  vertices in the order existing in the graph's list of loops.

However, this approach of finding the next immediate vertex without an overall plan is not as efficient as a pre-planned ordering.

Future work shall consider how to mix the ordering for sparsity with the ordering for numerical stability and how to mix a pre-planned order with mid-factorisation changes due to numerical stability considerations.

### 5.7.2.3 Factorisation Ordering for Online Problems

Mahon [49] discusses online methods in relation to factorisation order. Mahon suggests ordering the current vehicle pose states *last* in the factorisation ordering. During vehicle prediction operations, this allows factorisation modification algorithms to

modify only the small number of states relating to the current and next vehicle pose states. Ordering the current vehicle pose states last also allows these to be recovered *first* in the back-solving phase.

Future research directions in online modification problems are discussed in section 6.2.1.

## 5.8 Chapter Conclusion

This chapter presented a new implementation of the LDL direct factorisation algorithm operating entirely in the graph embedded representation. In the graph structure, the factorisation operations correspond directly to operations accessing adjacent vertices and modifying graph edges.

The fast insertion and adjacency capabilities of the graph based linear system representation allows the direct solving algorithm greater flexibility and capabilities, particularly regarding the factorisation ordering. This allows the factorisation method to determine or alter the ordering during factorisation, instead of pre-planning it. This makes the algorithms simpler and enables the factorisation of indefinite systems, as arises in solving the augmented system form.

The operation of the solving algorithm in the graph representation demonstrates the applicability of the graph based representation for the linear systems. The lack of matrix and vector semantics (value of elements at integer indices) does not present an obstacle to the solving algorithm, since every operation of the solving algorithm translates elegantly into operations on the graph, such as iterating through lists of loops, vertices and adjacent vertices, and adding and removing edges.

The operation of factorisation based on graph theoretic terms, such as adjacency and topological ordering, gives further insights into the algorithms beyond that obtained when operating only with matrix indexing, especially given the complexity of conventional sparse matrix formats. Future algorithm development focused on the

capabilities of the graph representation may be able to exploit these further for faster results.

# Chapter 6

## Conclusion and Future Research

This thesis proposed the use of the augmented system form, in conjunction with a novel graph representation for the estimation problem, together with a graph based linear direct solving algorithm. This chapter summarises the contributions of this thesis and outlines areas for future research.

### 6.1 Summary of Contributions

#### 1. Augmented Methods in Estimation

This thesis contributed extensions to the augmented system form - a generalisation of the information form, consisting of augmenting observations & constraints in addition to the states - and proposed its use as a general formulation of the estimation problem. The augmented form provides a mathematical system showing explicitly and distinctly the states and observations together with Lagrange multipliers for their interaction. This thesis showed the augmented system form is more general than the information form, and this thesis proposed that it therefore provides a more general starting point for the formulation and solving process. The information form is able to be recovered by eliminating the observations first. Alternative solving approaches can be realised by factorising variables in a general order beyond the fixed observations-first approach. This

is strictly required for constraints, and also improves numerical stability under small  $\mathbf{R}$  and improves the fill-in sparsity by offering the ability to factorise between the observation Lagrange multipliers and states in a mixed order.

## 2. A novel graph-theoretic structure for sparse estimation problems

This thesis presented a novel graph embedded representation for estimation problems and their associated sparse linear systems. The representation focuses on objects and the links between them rather than having a matrix and vector representation (typified by access to entries at integer indices). It includes estimation problem states, nonlinear observation terms and their linearisation, and is suitable for sparse linear systems generally. The representation also contributes a mapping of the matrix and vector concepts into graph vertices and edges, including novel graph representations for these linear algebraic concepts. These consist of graph loops, mixed symmetric and directed edges, and multiple graph edgesets. The resulting graph representation has the benefit of constant time insertion and removal of edges and vertices, constant time access to adjacent variables, and a decoupling of the identity of variables with their storage method. These benefits are applied to the implementation of the graph based solving algorithm.

## 3. Estimation algorithms in the graph structure

This thesis presented a new implementation of the LDL direct factorisation algorithm operating entirely in the graph embedded representation. In the graph structure, the factorisation operations correspond directly to operations accessing adjacent vertices and modifying graph edges. The capabilities of the graph based linear systems representation allows the solving algorithm greater flexibility and capabilities, particularly regarding the factorisation ordering. This allows the factorisation method to determine or alter the ordering during factorisation, instead of pre-planning it. This, in turn, makes the algorithms simpler and enables the factorisation of indefinite systems, which arise in solving the augmented system form.

The operation of the solving algorithm in the graph representation demonstrates

the applicability and novel capabilities of the graph based representation.

## 6.2 Future Research

The contributions of this thesis open up a range of new tools which may be applied to new and outstanding research problems. This section outlines broad areas for such future research, in addition to the specific areas for future research identified in each chapter.

### 6.2.1 Online Methods

Online methods consist of techniques to improve the computational efficiency of the solving methods suitable for repeated solving of the growing estimation problem during operation of the system, i.e.: online execution as opposed to offline or batch execution.

Online methods exploit the fact that observations arrive in succession and that the structure of old parts of the estimation problem are identical in successive timesteps. Parts of the state estimates may be similar in successive timesteps, depending on the structure of the changes resulting from new observations.

Particular benefit can be obtained from new observations which *extend* the estimation problem rather than substantially *changing* existing parts of the problem. Typically this occurs during exploration of new spatial regions.

Mahon [49] discusses online methods in relation to factorisation order. Mahon suggests ordering the current vehicle pose states *last* in the factorisation ordering. During vehicle prediction operations, this allows factorisation modification algorithms to modify only the small number of states relating to the current and next vehicle pose states. Ordering the current vehicle pose states last also allows these to be recovered *first* in the back-solving phase.

The graph structure proposed in this thesis allows full incremental, constant time modifications to the graph structure (as opposed to matrix embedded implementations).

The implementation of direct solving methods in the graph structure opens up the possibility of developing more advanced factorisation algorithms especially online (re)-factorisation methods. Future research will adapt online modification algorithms from the sparse matrix literature [19, 34] into the graph based factorisation approach of this chapter.

The implementation of the direct solving methods in the graph structure also has the benefit that elimination orders can be chosen arbitrarily without requiring data copying (“pivoting”), since the elimination ordering is not implied by the implementation (storage) ordering.

In addition, the augmented system form proposed in this thesis is expected to help in the online solving case. The augmented system form has easier access to the observation Jacobians than in the information form. This allows easy re-linearisation in  $\mathbf{A}$ , which represents a full formulation of the estimation problem.

Following additions and re-linearisations, the augmented system form is also expected to help online problems by allowing the choice of eliminating or factorising states and observations in any order.

For online localisation and mapping, future research will incorporate regions of variables in the graph which are subject to the following *phases*:

### **Nonlinear variation**

Variables immediately adjacent to new states or observations will vary significantly due to nonlinear iterations towards a solution and the arrival of additional new observations. It may not be worthwhile factorising such variables and *iterative* methods may be more favourable.

### **Linear variation**

Variables further away from new observations and states (in terms of graph links) will be subject to lesser variations, within the range of *linear variation*.

### **Static**

Variables even further away will be subject to few variations and become effectively static. These may still be retained in case of future loop-closures. These

variables will have been factorised earlier on and should not need to be touched during normal solving operations. The limited *solving regions* of section 5.4.3 would be used to avoid touching these static variables when solving for variations in the above linear-variation region.

### Discarded

Old variables with stable values, especially those which cannot be re-linked by loop-closures, can be safely marginalised out and discarded.

When exploring, these phases would be defined by graph propagation back along the chain starting from the new observations and states. Changes in structure in loop-closure events would cause a wide-ranging re-evaluation of these regions. When loop closing, the nonlinear region would expand out to encompass the large scale nonlinear variations encountered in loop closure. Variables previously static would enter nonlinear adjustment again.

Therefore, the graph representation and solving framework is expected to play an important role in advanced implementations of online localisation and mapping in future work.

## 6.2.2 Iterative Methods

Iterative methods aim to solve systems via a sequence of approximations which converge to the solution [61]. Iterative methods are important for large scale systems which are beyond the reach of direct solving approaches [61].

Iterative methods, especially *conjugate gradients*, have been applied in the estimation and SLAM literature (for example [71, 75] ). Iterative methods have also been noted in the context of the online solution methods [75]. When the problem structure consists of a growing chain-like structure, the solution estimates change only slowly. In these cases, the iterative methods can be used with a bounded number of iterations per timestep [21, 75]



For the iterative solving of indefinite linear systems, the biconjugate gradient (BCG) method can be used [27]. The biconjugate gradient method solves indefinite problems with a monotonic reduction in the norm of the residual  $\mathbf{Ax} - \mathbf{b}$ , whereas the plain conjugate gradient method applied to indefinite problems results in alternating reduction and increases in the norm of the residual.

The direct methods developed in this thesis may be applied, in future work, to *preconditioning* iterative methods. Combining elements of direct solving with iterative solving is a vital part of modern iterative solution methods [61] and has also been applied in the SLAM context (for example, [76]).

The graph representation proposed in this thesis is well suited to iterative methods for various reasons: It is well suited and fast for performing the matrix-vector product, even when only one half of symmetric systems are stored, even for both direct and transposed matrix-vector products. The graph representation allows the matrix-vector product to be evaluated for particular vertices individually. As such, the graph representation proposed here can be used in future to aid the performance of iterative methods for online estimation problems in localisation and mapping.

### 6.2.3 Data Association Methods

The main estimation loop outlined in algorithm 1 refers to the fact that the algorithms of this thesis reside in an inner loop. The selection of data association is effectively an outer-loop process that would use the methods proposed in this thesis. It has been the intention of this thesis to support the representation and evaluation of particular data association choices, even though this thesis has not focused in detail on data association. In particular, the observation Lagrange multipliers will be important, in future work, for identifying particular observation terms and *paths through the graph structure* which are subject to large Lagrange multipliers. These identify “stress pathways”, on which a data association algorithm may be able to focus.

The residual Mahalanobis distance of chapter 3 is intended to aid data association evaluations in the trajectory state and augmented system form contexts.

The graph representation is able to encode data association choices through the use of *agreement enforcing* equality constraints between separate instances of a state. Such links would be, of course, fully reversible and would also provide a Lagrange multiplier for evaluating the agreement.

In future research, these tools may aid the representation, evaluation and optimisation of data association choices.

### 6.2.4 Decentralisation

In future research, the methods of this thesis may be applied to the problem of decentralised estimation. In a decentralised estimation network, a state may be estimated simultaneously on multiple platforms. The platforms are required to achieve estimates equivalent to a centralised estimator.

This problem can be cast as a problem in which the platforms estimate independently but are subject to equality constraints which impose a requirement for agreement [62].

Such a system with multiple instances of the states bound together with equality constraints can be shown to be equivalent to the *marginal-passing* channel filtering approach used in [52, 53], by *eliminating* the agreement constraints and *all but one* of the multiple state instances.

Such agreement constraints can also be represented using the augmented system form proposed in this thesis. Under these conditions, the decentralisation network may communicate the shared states and the Lagrange multipliers of the agreement constraints on the shared states. These are *vectors* in the size of the shared states. This approach may have different or improved communication properties than the communication of information *marginal* matrices in the shared states.

The Lagrange multipliers of these agreement constraints would also indicate the extent and direction of any inconsistency between the platforms, which may help incorporate data association and verification algorithms.

In *cyclic* decentralised agreement networks, the set of equality constraints violates the full rank condition on constraints. The agreement network then becomes over-constrained. However, by *regularising* the augmented system form of the network, the constraints are infinitesimally relaxed and the system becomes solvable. Such an over-constrained (but regularised) system is more general than the *acyclic* structure. In future research this may lead to better algorithms for the solution of cyclic decentralisation networks.

### 6.3 Conclusion

This thesis contributed the augmented system form which augments the observation Lagrange multipliers into a joint system with the state variables. This forms a mathematical formulation of the estimation problem that contains all the variables of interest and the sparse links between them. This system is then embedded in a graph based representation. Together, the augmented system form and the graph embedded representation form an approach in which the estimation problem is formulated in as much detail as possible, with sparse links between the variables encoding the conditional independence structure of the problem. This formulation is then able to be considered by the graph based solving algorithm, which operates in the graph to eliminate or factorise variables.

This set of the augmented system form, graph representation and solving algorithm are fundamental tools which will lead to a new variety of online, flexible, formulation and solving approaches for estimation in localisation and mapping.

# Appendix A

## Augmented System Details

227

### A.1 Eliminating States

In section 3.2.2, we applied the condition  $\nabla_{\boldsymbol{\nu}} L = \mathbf{0}$  from equation 3.13 to the Lagrangian,  $L(\boldsymbol{\nu}, \mathbf{x})$ , of equation 3.10 to obtain the *information form* quadratic  $F(\mathbf{x})$  of equation 3.2. This was equivalent to eliminating the observations from the augmented form, obtaining the resulting information form.

To complement this, we can eliminate the states from the augmented form, by applying the condition  $\nabla_{\mathbf{x}} L = \mathbf{0}$ . In equation 3.13, we can require that  $\nabla_{\mathbf{x}} L = \mathbf{0}$  and obtain the resulting system in  $\boldsymbol{\nu}$ . This yields a relationship from  $\boldsymbol{\nu}$  to  $\mathbf{x}$ , a function:

$\tilde{\mathbf{x}}(\boldsymbol{\nu})$ . The relationship:

$$\mathbf{Y}(\tilde{\mathbf{x}}(\boldsymbol{\nu}) - \mathbf{x}_p) - \mathbf{H}^T \boldsymbol{\nu} = \mathbf{0} \quad (\text{A.1})$$

$$\tilde{\mathbf{x}}(\boldsymbol{\nu}) = \mathbf{Y}^{-1}(\mathbf{H}^T \boldsymbol{\nu} + \mathbf{Y} \mathbf{x}_p) \quad (\text{A.2})$$

$$= \mathbf{P}(\mathbf{H}^T \boldsymbol{\nu} + \mathbf{Y} \mathbf{x}_p) \quad (\text{A.3})$$

Starting from equation 3.10 and applying equation A.3:

$$L(\boldsymbol{\nu}, \mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}_p)^T \mathbf{Y}(\mathbf{x} - \mathbf{x}_p) - \frac{1}{2} \boldsymbol{\nu}^T \mathbf{R} \boldsymbol{\nu} - \boldsymbol{\nu}^T (\mathbf{h}(\mathbf{x}) - \mathbf{z}) \quad (\text{A.4})$$

$$L(\boldsymbol{\nu}, \tilde{\mathbf{x}}(\boldsymbol{\nu})) = \frac{1}{2}(\tilde{\mathbf{x}}(\boldsymbol{\nu}) - \mathbf{x}_p)^T \mathbf{Y}(\tilde{\mathbf{x}}(\boldsymbol{\nu}) - \mathbf{x}_p) - \frac{1}{2} \boldsymbol{\nu}^T \mathbf{R} \boldsymbol{\nu} - \boldsymbol{\nu}^T (\mathbf{h}(\tilde{\mathbf{x}}(\boldsymbol{\nu})) - \mathbf{z}) \quad (\text{A.5})$$

$$L(\boldsymbol{\nu}, \tilde{\mathbf{x}}(\boldsymbol{\nu})) = -\frac{1}{2} \boldsymbol{\nu}^T (\mathbf{H} \mathbf{P} \mathbf{H}^T + \mathbf{R}) \boldsymbol{\nu} - \boldsymbol{\nu}^T (\mathbf{H} \mathbf{x}_p - \mathbf{z}) \quad (\text{A.6})$$

Equation A.6 is a reduced Lagrangian quadratic for the  $\boldsymbol{\nu}$  variables only. The relationships among these quadratics are summarised in appendix A.3.

The solution is obtained by setting  $\nabla_{\boldsymbol{\nu}} L(\boldsymbol{\nu}, \tilde{\mathbf{x}}(\boldsymbol{\nu})) = \mathbf{0}$ , which results in:

$$(\mathbf{R} + \mathbf{H} \mathbf{P} \mathbf{H}^T) (\boldsymbol{\nu}) = -(\mathbf{H} \mathbf{x}_p - \mathbf{z})$$

(A.7)

Similarly, in terms of increments to the existing value  $\boldsymbol{\nu}_0$ :

$$\left(\mathbf{R} + \mathbf{H}\mathbf{P}\mathbf{H}^T\right) \left(\Delta\boldsymbol{\nu}\right) = -(\mathbf{H}\mathbf{x}_p - \mathbf{z}) - \mathbf{H}\mathbf{P}\mathbf{H}^T\boldsymbol{\nu}_0 - \mathbf{R}\boldsymbol{\nu}_0 \quad (\text{A.8})$$

Equation A.7 is the augmented system form of equation 3.17 marginalised into only the  $\boldsymbol{\nu}$  variables.

- The proces of marginalising away the states requires an invertible  $\mathbf{Y}$ . The covariance  $\mathbf{P}$  is equal to  $\mathbf{Y}^{-1}$ .
- The posterior inverse-covariance for the values of  $\boldsymbol{\nu}$  is  $\mathbf{R} + \mathbf{H}\mathbf{P}\mathbf{H}^T$ .
- Just as the information form shows the combined effect of the prior and the observation on the posterior state estimate, this form shows the effect on the posterior observation Lagrange multipliers.

In conclusion, taking the augmented system form and eliminating the states results in a quadratic ( equation A.6 ) in the  $\boldsymbol{\nu}$  variables and associated linear system for the solution ( equation A.7). This is interesting as a complement of the information form: representing the observation Lagrange multiplier marginal rather than the state marginal. The resulting expressions relate closely to the expressions of the innovation and innovation distance and therefore explains their origin in another alternative way.

## A.2 Terms Relating to Residuals and Innovations

The innovation  $\boldsymbol{\nu}_i = \mathbf{h}(\mathbf{x}_p) - \mathbf{z}$

The observation residual  $\boldsymbol{\nu}_z = \mathbf{h}(\mathbf{x}_e) - \mathbf{z}$

The prior residual  $\boldsymbol{\nu}_p = \mathbf{x}_e - \mathbf{x}_p$

The observation Lagrange multiplier  $\boldsymbol{\nu} = -\mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}_e) - \mathbf{z})$  if  $\mathbf{R}^{-1}$  exists

$$\boldsymbol{\nu} = -\mathbf{R}^{-1}\boldsymbol{\nu}_z$$

### A.3 Quadratic Forms and Mahalanobis Distances

The augmented system Lagrangian	$L(\boldsymbol{\nu}, \mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}_p)^T \mathbf{Y}(\mathbf{x} - \mathbf{x}_p) - \frac{1}{2}\boldsymbol{\nu}^T \mathbf{R}\boldsymbol{\nu} - \boldsymbol{\nu}^T(\mathbf{h}(\mathbf{x}) - \mathbf{z})$
The state objective function	$F(\mathbf{x}) = L(\tilde{\boldsymbol{\nu}}(\mathbf{x}), \mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}_p)^T \mathbf{Y}(\mathbf{x} - \mathbf{x}_p) + \frac{1}{2}(\mathbf{h}(\mathbf{x}) - \mathbf{z})^T \mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}) - \mathbf{z})$
The $\boldsymbol{\nu}$ quadratic	$L(\boldsymbol{\nu}, \tilde{\mathbf{x}}(\boldsymbol{\nu})) = -\frac{1}{2}\boldsymbol{\nu}^T(\mathbf{H}\mathbf{P}\mathbf{H}^T + \mathbf{R})\boldsymbol{\nu} - \boldsymbol{\nu}^T(\mathbf{h}(\mathbf{x}_p) - \mathbf{z})$
The innovation Mahalanobis distance	$M_i = \frac{1}{2}(\mathbf{h}(\mathbf{x}_p) - \mathbf{z})^T(\mathbf{R} + \mathbf{H}\mathbf{P}\mathbf{H}^T)^{-1}(\mathbf{h}(\mathbf{x}_p) - \mathbf{z})$
The residual Mahalanobis distance	$M_r(\mathbf{x}) = \frac{1}{2}(\mathbf{h}(\mathbf{x}) - \mathbf{z})^T \mathbf{R}^{-1}(\mathbf{h}(\mathbf{x}) - \mathbf{z}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_p)^T \mathbf{P}^{-1}(\mathbf{x} - \mathbf{x}_p)$

The above functions (and constant  $M_i$ ) all evaluate identically at the solution  $\mathbf{x}$  and  $\boldsymbol{\nu}$ .

$$L(\boldsymbol{\nu}, \mathbf{x}) = F(\mathbf{x}) = L(\tilde{\boldsymbol{\nu}}(\mathbf{x}), \mathbf{x}) = L(\boldsymbol{\nu}, \tilde{\mathbf{x}}(\boldsymbol{\nu})) = M_i = M_r(\mathbf{x}) \quad \text{at the solution } \mathbf{x} \text{ and } \boldsymbol{\nu}$$



## A.4 Linear Systems

Related linear systems, in incremental form:

$$\begin{aligned}
 \text{The augmented system form} \quad & \begin{pmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{pmatrix} \begin{pmatrix} \Delta \boldsymbol{\nu} \\ \Delta \mathbf{x} \end{pmatrix} = - \begin{pmatrix} \mathbf{R} \boldsymbol{\nu}_0 + (\mathbf{h}(\mathbf{x}_0) - \mathbf{z}) \\ \mathbf{H}^T \boldsymbol{\nu}_0 - \mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p) \end{pmatrix} \\
 \text{The information form} \quad & (\mathbf{Y} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}) (\Delta \mathbf{x}) = -\mathbf{Y}(\mathbf{x}_0 - \mathbf{x}_p) - \mathbf{H}^T \mathbf{R}^{-1} (\mathbf{h}(\mathbf{x}_0) - \mathbf{z}) \\
 \text{The } \boldsymbol{\nu} \text{ form} \quad & (\mathbf{R} + \mathbf{H} \mathbf{P} \mathbf{H}^T) (\Delta \boldsymbol{\nu}) = -(\mathbf{R} + \mathbf{H} \mathbf{P} \mathbf{H}^T) \boldsymbol{\nu}_0 - (\mathbf{h}(\mathbf{x}_p) - \mathbf{z})
 \end{aligned}$$

Related linear systems, non-incremental form:

$$\begin{aligned}
 \text{The augmented system form} \quad & \begin{pmatrix} \mathbf{R} & \mathbf{H} \\ \mathbf{H}^T & -\mathbf{Y} \end{pmatrix} \begin{pmatrix} \boldsymbol{\nu} \\ \mathbf{x} \end{pmatrix} = - \begin{pmatrix} (\mathbf{h}(\mathbf{x}_0) - \mathbf{z}) - \mathbf{H} \mathbf{x}_0 \\ \mathbf{Y} \mathbf{x}_p \end{pmatrix} \\
 \text{The information form} \quad & (\mathbf{Y} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H}) \mathbf{x} = \mathbf{Y} \mathbf{x}_p - \mathbf{H}^T \mathbf{R}^{-1} ((\mathbf{h}(\mathbf{x}_0) - \mathbf{z}) - \mathbf{H} \mathbf{x}_0) \\
 \text{The } \boldsymbol{\nu} \text{ form} \quad & (\mathbf{R} + \mathbf{H} \mathbf{P} \mathbf{H}^T) \boldsymbol{\nu} = -(\mathbf{h}(\mathbf{x}_p) - \mathbf{z})
 \end{aligned}$$

## A.5 Proof of Equivalence of Residual and Innovation Distances

This section proves equation 3.94, that the proposed residual Mahalanobis distance equals the conventional innovation Mahalanobis distance.

1. The “innovation” approach terms are:

$$\begin{aligned}
 &\text{Innovation from (3.80):} & \boldsymbol{\nu}_i &= \mathbf{H}\hat{\mathbf{x}} - \mathbf{z} \\
 &\text{Inserting (3.76) \& (3.78)} & &= \mathbf{H}(\mathbf{x} + \mathbf{w}_p) - (\mathbf{H}\mathbf{x} + \mathbf{w}_z) \\
 & & \boldsymbol{\nu}_i &= \mathbf{H}\mathbf{w}_p - \mathbf{w}_z \\
 &\text{Innovation Distance, From (3.85):} & M_i &= \boldsymbol{\nu}_i^T \mathbf{S}^{-1} \boldsymbol{\nu}_i
 \end{aligned} \tag{A.9}$$

2. The “residuals” approach terms depend on  $\mathbf{x}_e$ . The observation and prior are fused to obtain  $\mathbf{x}_e$ . The expressions are written using Information matrix  $\mathbf{Y}$  and vector  $\mathbf{y}$ .

$$\text{Prior information matrix \& vector} \quad \mathbf{Y}_p = \mathbf{P}^{-1} \quad \mathbf{y}_p = \mathbf{Y}\hat{\mathbf{x}} = \mathbf{P}^{-1}\hat{\mathbf{x}} \tag{A.10}$$

$$\text{Observation information matrix \& vector} \quad \mathbf{Y}_z = \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} \quad \mathbf{y}_z = \mathbf{H}^T \mathbf{R}^{-1} \mathbf{z} \tag{A.11}$$

$$\text{Posterior information matrix} \quad \mathbf{Y}_{\text{post}} = \mathbf{Y}_p + \mathbf{Y}_z \quad (\text{A.12})$$

$$\text{Posterior estimate} \quad \mathbf{x}_e = (\mathbf{Y}_{\text{post}})^{-1} (\mathbf{y}_p + \mathbf{y}_z) \quad (\text{A.13})$$

$$\mathbf{x}_e = (\mathbf{P}^{-1} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H})^{-1} (\mathbf{P}^{-1} \hat{\mathbf{x}} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{z}) \quad (\text{A.14})$$

3. Expanding terms for the observation residual,  $\boldsymbol{\nu}_z$ :

$$\text{Observation residual from (3.87)} \quad \boldsymbol{\nu}_z = \mathbf{H} \mathbf{x}_e - \mathbf{z}$$

$$\text{Inserting equations (3.78) and (A.14)} \quad = \mathbf{H} (\mathbf{Y}_{\text{post}})^{-1} (\mathbf{P}^{-1} \mathbf{w}_p + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{w}_z) - \mathbf{w}_z$$

$$\text{Expanding} \quad = \mathbf{H} \mathbf{Y}_{\text{post}}^{-1} \mathbf{Y}_p \mathbf{w}_p + \mathbf{H} \mathbf{Y}_{\text{post}}^{-1} \mathbf{H}^T \mathbf{R}^{-1} \mathbf{w}_z - \mathbf{w}_z \quad (\text{A.15})$$

$$\text{note:} \quad \mathbf{Y}_{\text{post}}^{-1} \mathbf{Y}_p = \mathbf{I} - \mathbf{Y}_{\text{post}}^{-1} \mathbf{Y}_z \quad (\text{A.16})$$

$$\text{Using (A.16)} \quad \boldsymbol{\nu}_z = \mathbf{H}(\mathbf{I} - (\mathbf{Y}_{\text{post}})^{-1} \mathbf{Y}_z) \mathbf{w}_p + \mathbf{H} \mathbf{Y}_{\text{post}}^{-1} \mathbf{H}^T \mathbf{R}^{-1} \mathbf{w}_z - \mathbf{w}_z$$

$$\text{Collecting} \quad = \mathbf{H} \mathbf{w}_p - \mathbf{H} \mathbf{Y}_{\text{post}}^{-1} \mathbf{Y}_z \mathbf{w}_p + \mathbf{H} \mathbf{Y}_{\text{post}}^{-1} \mathbf{H}^T \mathbf{R}^{-1} \mathbf{w}_z - \mathbf{w}_z$$

$$\text{Factoring} \quad = (\mathbf{I} - \mathbf{H} \mathbf{Y}_{\text{post}}^{-1} \mathbf{H}^T \mathbf{R}^{-1})(\mathbf{H} \mathbf{w}_p - \mathbf{w}_z) \quad (\text{A.17})$$

$$\text{Substituting (A.9)} \quad \boldsymbol{\nu}_z = (\mathbf{I} - \mathbf{H} \mathbf{Y}_{\text{post}}^{-1} \mathbf{H}^T \mathbf{R}^{-1})(\boldsymbol{\nu}_i) \quad (\text{A.18})$$

4. Expanding the observation residual distance,  $M_z$ :

$$\text{From (3.92)} \quad M_z = \boldsymbol{\nu}_z^T \mathbf{R}^{-1} \boldsymbol{\nu}_z \quad (\text{A.19})$$

$$\text{From (A.18)} \quad M_z = ((\mathbf{I} - \mathbf{H}\mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1})(\boldsymbol{\nu}_i))^T \mathbf{R}^{-1} ((\mathbf{I} - \mathbf{H}\mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1})(\boldsymbol{\nu}_i)) \quad (\text{A.20})$$

$$\text{Expanding the Quadratic:} \quad M_z = \boldsymbol{\nu}_i^T (\mathbf{R}^{-1} - 2\mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}(\mathbf{H}^T\mathbf{R}^{-1}) + \mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}\mathbf{Y}_z\mathbf{Y}_{\text{post}}^{-1}(\mathbf{H}^T\mathbf{R}^{-1}))\boldsymbol{\nu}_i \quad (\text{A.21})$$

$$\text{Which is of the form:} \quad M_z = \boldsymbol{\nu}_i^T \mathbf{A} \boldsymbol{\nu}_i \quad (\text{A.22})$$

5. Expanding terms for the prior residual,  $\boldsymbol{\nu}_p$ :

$$\text{Prior residual from equation (3.89)} \quad \boldsymbol{\nu}_p = \mathbf{x}_e - \hat{\mathbf{x}}$$

$$\text{Inserting equations (3.76) and (A.14)} \quad = -\mathbf{w}_p + \mathbf{Y}_{\text{post}}^{-1} (\mathbf{P}^{-1}\mathbf{w}_p + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{w}_z) \quad (\text{A.23})$$

$$= -(\mathbf{I}_p - \mathbf{Y}_{\text{post}}^{-1}\mathbf{Y}_p)\mathbf{w}_p + \mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1}\mathbf{w}_z \quad (\text{A.24})$$

$$= -(\mathbf{Y}_{\text{post}}^{-1}\mathbf{Y}_z)\mathbf{w}_p + \mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1}\mathbf{w}_z \quad (\text{A.25})$$

$$= -\mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}\mathbf{w}_p + \mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1}\mathbf{w}_z \quad (\text{A.26})$$

$$= -\mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1}(\mathbf{H}\mathbf{w}_p - \mathbf{w}_z) \quad (\text{A.27})$$

$$\text{Substituting (A.9)} \quad \boldsymbol{\nu}_p = -\mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1}(\boldsymbol{\nu}_i) \quad (\text{A.28})$$

6. Expanding the prior residual distance,  $M_p$ :

$$\text{From (3.92)} \quad M_p = \boldsymbol{\nu}_p^T \mathbf{P}^{-1} \boldsymbol{\nu}_p \quad (\text{A.29})$$

$$\text{From (A.28)} \quad M_p = (\mathbf{Y}_{\text{post}}^{-1} \mathbf{H}^T \mathbf{R}^{-1}(\boldsymbol{\nu}_i))^T \mathbf{P}^{-1} (\mathbf{Y}_{\text{post}}^{-1} \mathbf{H}^T \mathbf{R}^{-1}(\boldsymbol{\nu}_i)) \quad (\text{A.30})$$

$$\text{Expanding the quadratic:} \quad M_p = \boldsymbol{\nu}_i^T (\mathbf{R}^{-1} \mathbf{H} \mathbf{Y}_{\text{post}}^{-1} \mathbf{P}^{-1} \mathbf{Y}_{\text{post}}^{-1} \mathbf{H}^T \mathbf{R}^{-1}) \boldsymbol{\nu}_i \quad (\text{A.31})$$

$$\text{Which is of the form:} \quad M_p = \boldsymbol{\nu}_i^T \mathbf{B} \boldsymbol{\nu}_i \quad (\text{A.32})$$

7. The main expression, distance  $M_r$  is:

$$M_r = M_z + M_p \quad (\text{A.33})$$

$$M_r = \boldsymbol{\nu}_i^T \mathbf{A} \boldsymbol{\nu}_i + \boldsymbol{\nu}_i^T \mathbf{B} \boldsymbol{\nu}_i \quad (\text{A.34})$$

$$M_r = \boldsymbol{\nu}_i^T (\mathbf{A} + \mathbf{B}) \boldsymbol{\nu}_i \quad (\text{A.35})$$

8. To expand the expression  $\mathbf{S}^{-1}$ , a matrix inversion lemma is used from [59].

$$\text{Matrix inversion lemma} \quad (\mathbf{A} + \mathbf{X} \mathbf{B} \mathbf{X}^T)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1} \mathbf{X} (\mathbf{B}^{-1} + \mathbf{X}^T \mathbf{A}^{-1} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{A}^{-1} \quad (\text{A.36})$$

$$\text{From (3.83)} \quad \mathbf{S}^{-1} = (\mathbf{R} + \mathbf{H} \mathbf{P} \mathbf{H}^T)^{-1} = \quad (\text{A.37})$$

$$\text{From (A.36)} \quad \mathbf{S}^{-1} = \mathbf{R}^{-1} - \mathbf{R}^{-1} \mathbf{H} (\mathbf{P}^{-1} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{R}^{-1} \quad (\text{A.38})$$

$$\mathbf{S}^{-1} = \mathbf{R}^{-1} - \mathbf{R}^{-1} \mathbf{H} \mathbf{Y}_{\text{post}}^{-1} \mathbf{H}^T \mathbf{R}^{-1} \quad (\text{A.39})$$

9. The claim of the proof, from (3.94) is:

$$M_i = M_r \quad (\text{A.40})$$

$$\boldsymbol{\nu}_i^T \mathbf{S}^{-1} \boldsymbol{\nu}_i = \boldsymbol{\nu}_i^T (\mathbf{A} + \mathbf{B}) \boldsymbol{\nu}_i \quad (\text{A.41})$$

$$\mathbf{S}^{-1} = (\mathbf{A} + \mathbf{B}) \quad (\text{A.42})$$

$$\mathbf{0} = (\mathbf{A} + \mathbf{B}) - \mathbf{S}^{-1} \quad (\text{A.43})$$

Using (A.22) & (A.32):

$$\begin{aligned} \mathbf{0} &= (\mathbf{R}^{-1} - 2\mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}(\mathbf{H}^T\mathbf{R}^{-1}) + \mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}\mathbf{Y}_z\mathbf{Y}_{\text{post}}^{-1}(\mathbf{H}^T\mathbf{R}^{-1})) \\ &\quad + (\mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}\mathbf{P}^{-1}\mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1}) - \mathbf{S}^{-1} \end{aligned} \quad (\text{A.44})$$

Using (A.39):

$$\begin{aligned} \mathbf{0} &= (\mathbf{R}^{-1} - 2\mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}(\mathbf{H}^T\mathbf{R}^{-1}) + \mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}\mathbf{Y}_z\mathbf{Y}_{\text{post}}^{-1}(\mathbf{H}^T\mathbf{R}^{-1})) + (\mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}\mathbf{P}^{-1}\mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1}) \\ &\quad - (\mathbf{R}^{-1} - \mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1}) \end{aligned} \quad (\text{A.45})$$

$$\mathbf{0} = -\mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}(\mathbf{H}^T\mathbf{R}^{-1}) + \mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}\mathbf{Y}_z\mathbf{Y}_{\text{post}}^{-1}(\mathbf{H}^T\mathbf{R}^{-1}) + \mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}\mathbf{P}^{-1}\mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1} \quad (\text{A.46})$$

$$\mathbf{0} = \mathbf{R}^{-1}\mathbf{H}\mathbf{Y}_{\text{post}}^{-1}(-(\mathbf{H}^T\mathbf{R}^{-1}) + \mathbf{Y}_z\mathbf{Y}_{\text{post}}^{-1}(\mathbf{H}^T\mathbf{R}^{-1}) + \mathbf{P}^{-1}\mathbf{Y}_{\text{post}}^{-1}\mathbf{H}^T\mathbf{R}^{-1}) \quad (\text{A.47})$$

$$\mathbf{0} = -\mathbf{H}^T\mathbf{R}^{-1} + (\mathbf{Y}_z\mathbf{Y}_{\text{post}}^{-1} + \mathbf{Y}_p\mathbf{Y}_{\text{post}}^{-1})(\mathbf{H}^T\mathbf{R}^{-1}) \quad (\text{A.48})$$

Now,  $(\mathbf{Y}_z \mathbf{Y}_{\text{post}}^{-1} + \mathbf{Y}_p \mathbf{Y}_{\text{post}}^{-1}) = \mathbf{I}$ , from (A.12)

$$\mathbf{0} = (- (\mathbf{H}^T \mathbf{R}^{-1}) + (\mathbf{H}^T \mathbf{R}^{-1})) \quad (\text{A.49})$$

$$\text{True} \quad (\text{A.50})$$

10. This completes the proof that equation 3.94 holds.

# Bibliography

- [1] “Boost C++ libraries.” [Online]. Available: <http://www.boost.org/>
- [2] “Intel math kernel library reference manual,” Intel, Document Number: 630813-029US, August 2008. [Online]. Available: <http://www.intel.com/software/products/mkl/docs/WebHelp/whnjs.htm>
- [3] M. Agrawal and K. Konolige, “FrameSLAM: From bundle adjustment to real-time visual mapping,” *IEEE Transactions on Robotics*, vol. 24, no. 5, October 2008.
- [4] P. R. Amestoy, T. A. Davis, and I. S. Duff, “Algorithm 837: AMD, an approximate minimum degree ordering algorithm,” *ACM Transactions on Mathematical Software*, vol. 30, no. 3, pp. 381–388, 2004. [Online]. Available: <http://dx.doi.org/10.1145/1024074.1024081>
- [5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users’ Guide*, 3rd ed. Philadelphia: SIAM, 1999.
- [6] M. Arioli, I. Duff, and P. de Rijk, “On the augmented system approach to sparse least-squares problems,” *Numerische Mathematik*, vol. 55, no. 6, pp. 667–684, 1989. [Online]. Available: <http://www.springerlink.com/content/q225u701377003h2/>
- [7] T. Bailey and H. Durrant-Whyte, “Simultaneous localization and mapping (SLAM): Part ii,” *IEEE Robotics and Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006. [Online]. Available: <http://dx.doi.org/10.1109/MRA.2006.1678144>
- [8] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation*. Wiley, 2001.
- [9] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. Philadelphia, PA: SIAM, 1994.
- [10] A. Björck, *Numerical methods for Least Squares Problems*. SIAM Philadelphia, 1996.



- [11] B. Bollobás, *Modern Graph Theory*. Springer, 1998.
- [12] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
- [13] D. Brown, “The bundle adjustment - progress and prospects,” *International Archives of Photogrammetry*, vol. 21, no. 3, 1976.
- [14] J. R. Bunch and B. N. Parlett, “Direct methods for solving symmetric indefinite systems of linear equations,” *SIAM Journal on Numerical Analysis*, vol. 8, no. 4, pp. 639–655, 1971. [Online]. Available: <http://www.jstor.org/stable/2949596>
- [15] J. R. Bunch and L. Kaufman, “Some stable methods for calculating inertia and solving symmetric linear systems,” *Mathematics of Computation*, vol. 31, no. 137, pp. 163–179, 1977. [Online]. Available: <http://www.jstor.org/stable/2005787>
- [16] T. A. Davis, “Algorithm 849: A concise sparse Cholesky factorization package,” *ACM Trans. Math. Softw.*, vol. 31, no. 4, pp. 587–591, 2005.
- [17] —, *Direct Methods for Sparse Linear Systems*, ser. Fundamentals of Algorithms. SIAM, 2006.
- [18] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, “A column approximate minimum degree ordering algorithm,” *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 353–376, 2004.
- [19] T. A. Davis and W. W. Hager, “Row modifications of a sparse cholesky factorization,” *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 3, pp. 621–639, 2005. [Online]. Available: <http://dx.doi.org/10.1137/S089547980343641X>
- [20] F. Dellaert and M. Kaess, “Square root SAM: Simultaneous localization and mapping via square root information smoothing,” *International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006.
- [21] T. Duckett, S. Marsland, and J. Shapiro, “Learning globally consistent maps by relaxation,” in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 4, 2000, pp. 3841–3846 vol.4.
- [22] I. S. Duff, M. A. Heroux, and R. Pozo, “An overview of the sparse basic linear algebra subprograms: The new standard from the blas technical forum,” *ACM Trans. Math. Softw.*, vol. 28, no. 2, pp. 239–267, 2002.
- [23] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: Part i,” *IEEE Robotics and Automation Magazine*, vol. 13, no. 2, pp. 99–108, 2006.

- [24] R. Eustice, H. Singh, and J. Leonard, “Exactly sparse delayed-state filters,” in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005.
- [25] R. Eustice, “Large-area visually augmented navigation for autonomous underwater vehicles,” Ph.D. dissertation, Massachusetts Institute of Technology / Woods Hole Oceanographic Institution, June 2005.
- [26] R. Eustice, O. Pizarro, and H. Singh, “Visually augmented navigation in an unstructured environment using a delayed state history,” *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2004, no. 1, pp. 25–32, 2004.
- [27] R. Fletcher, *Conjugate gradient methods for indefinite systems*, ser. Lecture Notes in Mathematics, 1976, vol. 506, pp. 73–89.
- [28] J. Folkesson and H. Christensen, “Graphical SLAM - a self-correcting map,” vol. 2004, no. 1, New Orleans, LA, United states, 2004, pp. 383–390.
- [29] W. N. Gansterer, J. Schneid, and C. W. Ueberhuber, “A survey of equilibrium systems,” 2002. [Online]. Available: [citeseer.ist.psu.edu/gansterer02survey.html](http://citeseer.ist.psu.edu/gansterer02survey.html)
- [30] M. R. Garey and D. S. Johnson, *Computers and intractability : a guide to the theory of NP-completeness*. San Francisco: W. H. Freeman, 1979.
- [31] A. George, J. Gilbert, and J. Liu, *Graph theory and sparse matrix computation*. Springer-Verlag New York, 1993.
- [32] A. George, K. Ikramov, and A. B. Kucherov, “Some properties of symmetric quasi-definite matrices,” *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1318–1323, 2000. [Online]. Available: <http://link.aip.org/link/?SML/21/1318/1>
- [33] J. R. Gilbert, C. Moler, and R. Schreiber, “Sparse matrices in matlab: Design and implementation,” *SIAM Journal on Matrix Analysis and Applications*, vol. 13, no. 1, pp. 333–356, 1992. [Online]. Available: <http://link.aip.org/link/?SML/13/333/1>
- [34] P. Gill, G. Golub, W. Murray, and M. Saunders, “Methods for modifying matrix factorizations,” *Mathematics of Computation*, vol. 28, no. 126, pp. 505–535, 1974.
- [35] M. Golfarelli, D. Maio, and S. Rizzi, “Correction of dead-reckoning errors in map building for mobile robots,” *IEEE Transactions on Robotics and Automation*, vol. 17, no. 1, pp. 37–47, 2001, dead reckoning errors;Map building;Odometry;. [Online]. Available: <http://dx.doi.org/10.1109/70.917081>

- [36] —, “Elastic correction of dead-reckoning errors in map building,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1998, pp. 905–911.
- [37] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. The John Hopkins University Press, 1996.
- [38] M. S. Grewal and A. P. Andrews, *Kalman Filtering - Theory and Practice*, 1993.
- [39] F. Harary, “A graph theoretic approach to matrix inversion by partitioning,” *Numerische Mathematik*, vol. 4, no. 1, pp. 128–135, December 1962.
- [40] M. Jordan, “Graphical models,” *Statistical Science*, vol. 19, no. 1, pp. 140–155, 2004.
- [41] M. I. Jordan, *An Introduction to Probabilistic Graphical Models*. University of California, Berkeley. unpublished, 2002.
- [42] M. Kaess, A. Ranganathan, and F. Dellaert, “iSAM: Incremental smoothing and mapping,” *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378, 2008. [Online]. Available: <http://dx.doi.org/10.1109/TRO.2008.2006706>
- [43] G. Karypis and V. Kumar, *Metis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0*, 1995. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.376>
- [44] F. Kschischang, B. Frey, and H.-A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.
- [45] J. B. Kuipers, *Quaternions and Rotation Sequences*. Princeton, 2002.
- [46] J. J. Leonard and R. J. Rikoski, “Incorporation of delayed decision making into stochastic mapping,” *International Symposium On Experimental Robotics*, 2000.
- [47] M. Lourakis and A. Argyros, “The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm,” Institute of Computer Science - FORTH, Heraklion, Crete, Greece, Tech. Rep. 340, Aug. 2004, available from <http://www.ics.forth.gr/~lourakis/sba>.
- [48] K. Madsen, H. B. Nielsen, and O. Tingleff, “Methods for non-linear least squares problems,” Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, p. 56, 1999. [Online]. Available: <http://www2.imm.dtu.dk/pubdb/p.php?660>
- [49] I. Mahon, “Vision-based navigation for autonomous underwater vehicles,” Ph.D. dissertation, March 2007.

- [50] P. S. Maybeck, *Stochastic models, estimation, and control*, ser. Mathematics in Science and Engineering, 1979, vol. 1.
- [51] K. Murphy, “The Bayes net toolbox for Matlab,” *Comput. Sci. Stat.*, vol. 33, pp. 1–20, 2001.
- [52] E. Nettleton, “Decentralised architectures for tracking and navigation with multiple flight vehicles,” Ph.D. dissertation, Australian Centre for Field Robotics, Department of Aerospace, Mechanical and Mechatronic Engineering, The University of Sydney, 2003.
- [53] E. Nettleton, H. Durrant-Whyte, and S. Sukkarieh, “A robust architecture for decentralised data fusion,” *International Conference on Advanced Robotics*, 2003.
- [54] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer-Verlag, 1999.
- [55] S. Parter, “The use of linear graphs in Gauss elimination,” *SIAM Review*, vol. 3, no. 2, pp. 119–130, 1961. [Online]. Available: <http://www.jstor.org/stable/2027387>
- [56] M. A. Paskin, “Thin junction tree filters for simultaneous localization and mapping,” in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, G. Gottlob and T. Walsh, Eds. San Francisco, CA: Morgan Kaufmann Publishers, 2003, pp. 1157–1164.
- [57] M. A. Paskin and G. D. Lawrence, “Junction tree algorithms for solving sparse linear systems,” University of California, Berkeley., Tech. Rep. UCB/CSD-03-1271, 2003. [Online]. Available: <http://ai.stanford.edu/~paskin/pubs/csd-03-1271.pdf>
- [58] J. Reinders, *Intel Threading Building Blocks*. O’Reilly, 2007.
- [59] S. Roweis, “Matrix identities.” [Online]. Available: <http://www.cs.toronto.edu/~roweis/notes/matrixid.pdf>
- [60] Y. Saad, “Sparskit: a basic tool kit for sparse matrix computations,” University of Minnesota, Tech. Rep., 1994. [Online]. Available: <http://www-users.cs.umn.edu/~saad/software/SPARSKIT/paper.ps>
- [61] ———, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia: SIAM, 2003.
- [62] S. Samar, S. Boyd, and D. Gorinevsky, “Distributed estimation via dual decomposition,” in *Proceedings European Control Conference (ECC)*, Kos, Greece, July 2007, pp. 1511–1516.
- [63] R. Sedgewick, *Algorithms in C++*. Pearson Education, 2002.

- [64] I. Siegel, “Deferment of Computation in the Method of Least Squares,” *Mathematics of Computation*, vol. 19, no. 90, pp. 329–331, 1965.
- [65] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *The boost graph library: user guide and reference manual*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [66] P. Smyth, “Belief networks, hidden markov models, and markov random fields: a unifying view,” *Pattern Recognition Letters*, vol. 18, no. 11-13, pp. 1261–1268, 1997. [Online]. Available: [http://dx.doi.org/10.1016/S0167-8655\(97\)01050-7](http://dx.doi.org/10.1016/S0167-8655(97)01050-7)
- [67] G. Stewart, “Building an old-fashioned sparse solver,” University of Maryland, Tech. Rep., 2003. [Online]. Available: <http://hdl.handle.net/1903/1312>
- [68] R. E. Tarjan, “Graph theory and Gaussian elimination.” Stanford, CA, USA, Tech. Rep., 1975.
- [69] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. MIT press, Cambridge, Massachusetts, USA, 2005. [Online]. Available: <http://www.probabilistic-robotics.org/>
- [70] S. Thrun and J. Leonard, “Simultaneous localization and mapping,” in *Springer Handbook of Robotics*. Springer, 2008, pp. 871–889.
- [71] S. Thrun and M. Montemerlo, “The graph SLAM algorithm with applications to large-scale mapping of urban structures,” *International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.
- [72] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, “Bundle adjustment - a modern synthesis,” in *Vision Algorithms: Theory and Practice*, ser. LNCS, W. Triggs, A. Zisserman, and R. Szeliski, Eds. Springer Verlag, 2000, pp. 298–375. [Online]. Available: [citeseer.ist.psu.edu/triggs00bundle.html](http://citeseer.ist.psu.edu/triggs00bundle.html)
- [73] R. J. Vanderbei, “Symmetric quasi-definite matrices,” Rutgers University, Tech. Rep., 1993. [Online]. Available: <ftp://dimacs.rutgers.edu/pub/dimacs/TechnicalReports/TechReports/1993/93-72.ps>
- [74] S. A. Vavasis, “Stable numerical algorithms for equilibrium systems,” *SIAM Journal on Matrix Analysis and Applications*, vol. 15, no. 4, pp. 1108–1131, 1994. [Online]. Available: [citeseer.ist.psu.edu/vavasis92stable.html](http://citeseer.ist.psu.edu/vavasis92stable.html)
- [75] M. R. Walter, R. M. Eustice, and J. J. Leonard, “Exactly Sparse Extended Information Filters for Feature-based SLAM,” *The International Journal of Robotics Research*, vol. 26, no. 4, pp. 335–359, 2007. [Online]. Available: <http://ijr.sagepub.com/cgi/content/abstract/26/4/335>

- [76] Z. Wang, S. Huang, and G. Dissanayake, “D-SLAM: A Decoupled Solution to Simultaneous Localization and Mapping,” *The International Journal of Robotics Research*, vol. 26, no. 2, pp. 187–204, 2007. [Online]. Available: <http://ijr.sagepub.com/cgi/content/abstract/26/2/187>
- [77] S. B. Williams, “Efficient solutions to autonomous mapping and navigation problems,” Ph.D. dissertation, The University Of Sydney, Sep. 2001.