The Development of Efficient Algorithms for Large-Scale Finite Element Limit Analysis

Nathan Carl Podlich

BEng(Hons); BBus; BCompSci

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in Civil Engineering

September 2017

I hereby certify that the work embodied in the thesis is my own work, conducted under normal supervision.

The thesis contains no material which has been accepted, or is being examined, for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. I give consent to the final version of my thesis being made available worldwide when deposited in the University's Digital Repository, subject to the provisions of the Copyright Act 1968 and any approved embargo.

(signed) Nathan Podlich ______________________________

Acknowledgements

The author is grateful for the financial assistance received from The Australian Research Council Centre of Excellence for Geotechnical Science and Engineering during his candidature.

The author would like to acknowledge his sincere gratitude for the patience and support of Professor Andrei Lyamin and Professor Scott Sloan. Their timely assistance and proof-reading are greatly appreciated. Thanks are also extended to Associate Professor Andrew Abbo for his guidance and enthusiasm.

Finally, thank you to my partner, Jessica, for her unwavering support and tolerance.

Abstract

Finite element limit analysis is a useful numerical method for stability assessment of a wide range of geotechnical and structural applications yielding lower and upper bound estimates on the ultimate loads which can be exerted on the structure. The most advanced formulations of numerical limit analysis are often cast as a conic optimisation problem, which is then solved very efficiently by specialised interior point methods. However, as the problems become larger, especially in three dimensions the computational demands in terms of both storage and time increase significantly.

This Thesis details the development of efficient methods for the solution of linear systems and presolve routines within an interior point framework for conic programs. Therefore, these methods all aim to reduce the computational time required to solve the finite element limit analysis problems. The solution of a linear system comprises the majority of the computational requirements and is thus the primary concern of this Thesis. A range of preconditioners for Krylov subspace iterative solvers are considered, as well as more conventional direct solvers and their parallelisation. The presolve routines seek to reduce the size of the optimisation problem to be solved and avoid likely numerical difficulties.

Preconditioners for Krylov subspace iterative solvers are the primary determinant for the success of an iterative solver-based approach. A range of preconditioners are developed for both positive-definite and symmetric indefinite linear systems in attempt to avoid the significant runtime and storage requirements associated with the direct solvers. The best performing methods are tested against state-of-the-art implementations using direct solvers on a set of test problems but are found to be uncompetitive in their runtime performance and their robustness. The focus is then switched to the parallelisation of a direct solver on modern hardware including massively parallel GPUs to reduce the computation time with significant gains achieved.

In addition to exploiting the full power of parallel processing, the Thesis develops and describes presolve routines which target effective treatment of fixed and free variables. The fixed variables cannot be immediately substituted out of the problem because they are associated with other variables through a conic constraint, but may still be exploited by careful manipulation of the linear system. The free variables can sometimes be

substituted out of the problem, however, avoiding the numerical difficulties they often present. This is achieved without increasing the size of the linear system to be solved, although it may require the ability to handle dense columns. Finally, an approach for solving a linear system with dense columns is detailed similar to that of exploiting the fixed variables.

Notation

Throughout this Thesis, matrices are denoted with upper-case bold symbols, while vectors are denoted by lower-case bold symbols. Scalar matrix or vector entries shall be denoted by lower-case symbols. Scalars are most often denoted by lower case characters from the Greek alphabet. $|\mathbf{x}|$ and $|\mathbf{A}|$ refer to the vector and matrix whose entries are the absolute value of the entries of $\mathbf{x}$ and $\mathbf{A}$, respectively. The inequality symbols $\leq$ and $\geq$ are element-wise inequalities, i.e. $\mathbf{A} \leq 0$ indicates that all entries of $\mathbf{A}$ are non-negative.

The equations defining the search direction within an interior point method are of the general form $\mathbf{Ax} = \mathbf{b}$. These symbols are used throughout to denote the components of a general system of linear equations. Any restrictions or limitations on the components of the general form are noted explicitly in the given context.

A bracketed superscript indicates the iteration index. However, in cases where inverse or transposition notation is required, the iteration index will be moved to the subscript and remain bracketed. Furthermore, a convention of using k for iteration indices will generally be adhered to, while i and j will normally represent row and column indices, respectively.

Table of contents

Acknowledgements v

Abstract vii

Notation ix

Table of contents xi

Chapter 1 Introduction 1

1.2 Finite element limit analysis 2

1.2.1 History of finite element limit analysis 2

1.2.2 Finite element limit analysis formulation 4

1.2.3 The FELA optimisation problem 14

1.3 Interior point methods for conic programming 15

1.3.1 Background 15

1.3.2 The search direction in interior point methods 22

1.3.3 Handling free variables 27

Chapter 2 Computing the search direction in IPMs 31

2.1 Direct solution schemes 32

2.1.1 Gaussian elimination 33

2.1.2 Orthogonal factorisation 40

2.1.3 Reordering 41

2.2 Inexact search directions in IPMs for conic optimisation 42

2.2.1 The relative performance of basic linear algebra operations 43

2.2.2 Iterative method termination 45

2.2.3 Iterative method termination within IPMs 47

2.3 Iterative solution schemes 49

2.3.1 Stationary methods 51

2.3.2 Ritz-Galerkin approach 53

2.3.3 Minimal norm residual approach 55

2.3.4 Petrov-Galerkin approach 59

2.3.5 Minimal norm error approach 62

2.3.6 Hybrid methods 62

2.4 Preconditioners for iterative linear solvers 64

2.4.1 Matrix splitting and incomplete factorisation preconditioners 66

2.4.2 Approximate inverses 68

2.4.3 Block structured preconditioners 70
2.4.4 Matrix permutation and ordering 78
Chapter 3 Performance of conventional approaches on some FELA problems 79
3.1 Test problems 79
3.1.1 Two-dimensional problems 79
3.1.2 Three-dimensional problems 82
3.1.3 Problem summary 87
3.2 Compared solvers 88
3.2.1 MOSEK 89
3.2.2 Gurobi 89
3.2.3 SDPT3 4.0 89
3.2.4 SeDuMi 1.31 90
3.2.5 Mix8 90
3.3 Comparison results 91
3.3.1 Smaller problems 92
3.3.2 Finer mesh problems 95
3.3.3 Comparison summary 99
3.4 Improving on the basic IPM implementation 103
3.4.1 Choice of direct method 104
3.4.2 Matrix reordering 107
3.4.3 Dealing with free variables 110
3.4.4 Presolving 114
3.4.5 Improvement summary 127
Chapter 4 Iterative solver approaches 131
4.1 Solving the normal equations 131
4.1.1 Test problems 132
4.1.2 Choices related to the iterative solver 133
4.1.3 Preconditioning the normal equations 135
4.2 Solving the augmented equations 165
4.3 Addressing the ill-conditioning in the search direction 170
4.4 Using PCG to compute the search direction in an IPM 171
Chapter 5 Parallelisation of the solution scheme 181
5.1 Overview of parallel computing 181
5.2 Parallelisation of the IPM 183

Chapter 6 Conclusions and future work.. 193
6.1 Future work.. 195
References.. 197

Chapter 1 Introduction

One of the most crucial aspects in the design of ground-based structures is the stability of the supporting material, the soil. The upper and lower bound theorems of limit analysis [1] provide a useful methodology to address the stability of the supporting body [2]. A lower bound on the true collapse load can be identified by finding a stress distribution which satisfies the equilibrium equations and stress boundary conditions, and does not violate the yield criterion at any point (a statically admissible stress field). An upper bound to the true collapse load can be determined by equating the external rate of work to the internal power dissipation through an assumed velocity field, and ensuring that the velocity boundary conditions, and the strain and velocity compatibility conditions are satisfied (a kinematically admissible velocity field). Using the lower and upper bound theorems with suitable stress and velocity fields, one can bracket the collapse load as accurately as is necessary for a given problem [3]. The availability of such a precise measure of the error sets limit analysis apart from many other forms of numerical analysis and makes it a very useful tool in predicting soil stability.

Formally, the lower bound can be stated as follows [3]:

If a distribution of stresses, σ_{ij}, can be found that satisfy equilibrium, balances the applied loads, T_i, on the stress boundary, A_t, and everywhere satisfies the yield condition $f(\sigma_{ij}) < 0$; then the body will not collapse.

The upper bound theorem states [3]:

If a compatible mechanism of plastic deformation, with strain rates $\dot{\varepsilon}_{ij}^p$ and strain rates $\dot{u}_{ij}^p$, is assumed satisfying $\dot{u}_i^p = 0$ on the displacement boundary A_u; then the applied loads, T_i, and the body forces, F_i, determined by equating the rate at which the external forces do work, $\int_{A_T} T_i \dot{u}_i^p dA + \int_V F_i \dot{u}_i^p dV$, to the rate of internal dissipation, $\int_V D(\dot{\varepsilon}_{ij}^p) dV = \int_V \sigma_{ij}^p \dot{\varepsilon}_{ij}^p dV$, will be either higher or equal to the actual limit load.

These theorems assume that the continuum will only be subject to small deformations and be composed of a perfectly plastic material obeying an associated flow rule. The associated flow rule requires the plastic strain rates, $\dot{\varepsilon}_{ij}^p$, to be normal to the surface of the material's yield function, denoted by $f(\sigma_{ij})$ [4].

As the geometry of the problem and the supporting soil being analysed becomes more complex, however, obtaining useful bounds on the true collapse load analytically becomes impossible or incredibly tedious. Fortunately, by discretising the problem using finite elements, realistic problems with stratified or anisotropic soils, complicated multi-structure geometries, and complicated loading may be analysed. This procedure is known as finite element limit analysis (FELA) and requires an optimisation problem to be solved to obtain each of the lower and upper bounds.

1.2 Finite element limit analysis

In the following, a brief history of FELA focussing on the work that has contributed to computing rigorous lower and upper bound solutions for problems in geomechanics is presented. The formulation of FELA problems into conic optimisation problems is then described before discussing the most common optimisation method being used to obtain solutions in the recent FELA literature.

1.2.1 History of finite element limit analysis

Lysmer [5] appears to be the first to apply a finite element discretisation to solve a limit analysis problem in soil mechanics. He obtained lower bounds on some plane problems by solving a linear program (LP) using the Simplex method [6]. In the formulation, Lysmer used three-noded linear finite elements, allowed for statically admissible stress discontinuities at the element interfaces, and used a linearised Mohr-Coulomb yield function. While it was stated that a minimum of a six-sided approximation was necessary, all Lysmer's results were obtained using an iterative method in which he solved the problem multiple times, and each time using just three linear inequalities to represent the yield criterion at each node; based on the stress state in the previously obtained solution, the three inequalities were modified to more closely resemble the Mohr-Coulomb criterion, although this process was not found to be stable [5]. The unknown nodal stresses for each element comprise a normal stress on the element faces (in two dimensions) either side of each node, and an additional normal stress perpendicular to one of the sides at the opposite node. The shear stresses are uniquely determined by an affine transformation of the normal stresses. While leading to fewer unknowns than a more conventional formulation (with three unknown stresses at each node in two dimensions), the constraint matrix may contain entries that vary widely in

magnitude because of their dependence on the element shape [5]. Anderheggen and Knöpfel [7] formulated linear programs to obtain both lower and upper bound solutions with a linearised Mohr-Coulomb material, but the equilibrium and compatibility conditions are only satisfied approximately and so the bounds obtained are not true bounds. Pastor [8] obtained lower bounds for the vertical cut and introduced prolongation or extension zones, which ensure the material does not violate the yield criterion beyond the finite element discretisation. Bottero *et al.* [9] solve some plane strain upper and lower bounds for a linearised Mohr-Coulomb through linear programming and mention that they have extended the kinematic formulation to use quadratic triangular elements, although no details are given. Sloan formulates both lower [10] and upper [11] bound problems as linear programs using a linearisation of the Mohr-Coulomb criterion, solving them efficiently by exploiting sparsity and using an active set method with a steepest edge search [12] that is better suited to LPs with more constraints than unknowns (which is generally the case when one linearises the Mohr-Coulomb yield condition). Sloan and Kleeman [13] improved the upper bound formulation, allowing for velocity discontinuities between each element and the direction of shearing to be found automatically.

In the early 1990s, a variety of solution schemes appeared in the literature capitalising on the advances being made in the optimisation field. Christiansen and Kortanek [14] solved Christiansen's [15] earlier mixed formulation (yielding neither a true lower nor upper bound) much more efficiently using an interior point method (IPM) for LP. Similarly, Zouain *et al.* [16] employed an IPM for nonlinear programming and solved a mixed formulation, representing the yield constraints as nonlinear inequalities and thus obviating the need for a large number of linear inequalities. The scheme does, however, require a smooth approximation to any non-differentiable points in the yield criterion (present in the Mohr-Coulomb criterion among others) [17]. This method was extended by Lyamin [18] and Lyamin and Sloan [19], [20] to obtain rigorous lower and upper bounds in a very general implementation, citing speedups of over $50\times$ compared with an LP formulation and allowing three-dimensional problems to be solved. Pastor *et al.* [21] also demonstrated the superiority of the IPM scheme in solving both lower and upper bound vertical cut problems. Krabbenhøft *et al.* [22] introduced a stress-based upper bound formulation that provides a significantly improved method for

incorporating discontinuities, and showed that the formulation was typically about $2\times$ faster than the conventional formulation given by Lyamin and Sloan [20].

The beginning of the third century saw finite element limit analysis (FELA) results published using conic inequalities to represent the yield criterion, generally for von Mises materials (see for example, [23], [24]). Building on these results, Makrodimopoulos and Martin [25], [26] presented second order cone program (SOCP) formulations for Drucker-Prager materials in two and three dimensions, and Mohr-Coulomb materials in two dimensions. They then solved the SOCP using one of the leading commercial solvers, `MOSEK` [27], that uses an efficient primal-dual IPM. The Mohr-Coulomb criterion for three-dimensional problems may also be cast as a semidefinite constraint, leading to a semidefinite program (SDP) [28]–[30]. These conic programs can exploit the large body of theoretical and practical results concerning IPMs obtained by the mathematical programming community during the last three decades. Details of these methods are considered in greater depth below.

The conic formulations are not the only FELA approaches being actively developed by researchers. An upper bound approach using an augmented Lagrangian optimisation scheme using `MUMPS` has the benefit that the matrix defining the search direction does not exhibit growth in the condition number as a solution is approached [31], and allows exploitation of parallel cluster-based systems to reduce the solve time, using domain decomposition to divide the work among processors [32].

1.2.2 Finite element limit analysis formulation

A brief overview of the finite element formulation of limit analysis problems is covered below, following [10], [11], [19], [20], [22], [25], [26]. For both the upper and lower bound formulations we consider a soil mass of volume V and surface area A, with prescribed tractions acting on the boundary A_t denoted as $\mathbf{t}$, $\mathbf{q}$ being the unknown tractions acting on A_q, and the known and unknown body forces acting on V denoted as $\mathbf{g}$ and $\mathbf{h}$, respectively. The soil material satisfies the yield function $f(\boldsymbol{\sigma}) \leq 0$, where $\boldsymbol{\sigma}$ represents the stresses in the soil. As the problems considered here are in both two and three dimensions, we denote the dimension of the problem as D, and consider the following subscripts to be equivalent when discussing problem formulations

$x \equiv 1, y \equiv 2, z \equiv 3$, corresponding to the standard rectangular Cartesian coordinate system. A superscript l indicates that the component corresponds to the l th node and a superscript e indicates that it corresponds to element e.

For computing both lower and upper bounds, the continuum is discretised using finite elements and thus the stresses in the lower bound and the velocities in the upper bound formulation at any point inside each element can be computed with

$$\boldsymbol{\sigma} = \sum_{l=1}^{D+1} N_l \boldsymbol{\sigma}^l \tag{1.1}$$

and

$$\dot{\mathbf{u}} = \sum_{l}^{D+1} N_l \dot{\mathbf{u}}^l , \tag{1.2}$$

respectively. Note that the description here considers linear elements, using the linear shape functions

$$N_l = \sum_{k=0}^{D} a_{lk} x_k , \tag{1.3}$$

where $\boldsymbol{\sigma}^l$ are the nodal stresses, $\dot{\mathbf{u}}^l$ are the nodal velocities, x_k are the nodal coordinates, and

$$a_{lk} = (-1)^{l+k+1} \frac{|\mathbf{C}_{lk}|}{|\mathbf{C}|},$$

$$\mathbf{C} = \begin{bmatrix} 1 & x_1^1 & \cdots & x_D^1 \\ 1 & x_1^2 & \cdots & x_D^2 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{D+1} & \cdots & x_D^{D+1} \end{bmatrix},$$

$|\mathbf{C}|$ is the determinant of $\mathbf{C}$, and $|\mathbf{C}_{lk}|$ is the determinant of matrix obtained by removing the l th row and the k th column from $\mathbf{C}$. Note that the index of the first column of $\mathbf{C}$ is 0 while the index of the first row is 1 when computing the determinants $|\mathbf{C}_{lk}|$.

This allows the standard finite element strain-displacement matrix $\mathbf{B}^e$ for linear-displacement constraint-strain elements to be represented as

$$\mathbf{B}^e = \begin{bmatrix} \mathbf{B}^1 & \cdots & \mathbf{B}^l & \cdots & \mathbf{B}^{D+1} \end{bmatrix},$$

where

$$\mathbf{B}^l = \begin{bmatrix} \delta_{111} a(l, \phi_{111}) & \cdots & \delta_{D11} a(l, \phi_{D11}) \\ \vdots & & \vdots \\ \delta_{1DD} a(l, \phi_{1DD}) & \cdots & \delta_{DDD} a(l, \phi_{DDD}) \\ \vdots & & \vdots \\ \delta_{11D} a(l, \phi_{11D}) & \cdots & \delta_{D1D} a(l, \phi_{D1D}) \end{bmatrix},$$

$$\delta_{ijk} = \begin{cases} 1 & \text{if } i = j \text{ or } i = k \\ 0 & \text{otherwise} \end{cases},$$

and

$$\phi_{ijk} = \begin{cases} k & \text{if } i = j \\ j & \text{if } i = k \\ 1 & \text{otherwise} \end{cases}.$$

This accounts for the symmetry in the stress tensor (and similarly the strain tensor)

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix}$$

by only considering the upper triangular portion of the stress tensor and using the ordering

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_{xx} & \sigma_{yy} & \sigma_{zz} & \sigma_{xy} & \sigma_{yz} & \sigma_{xz} \end{Bmatrix}^T \tag{1.4}$$

for the three-dimensional stress tensor, and

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_{xx} & \sigma_{yy} & \sigma_{xy} \end{Bmatrix} \tag{1.5}$$

in two dimensions. The strain vector is analogous to this for both two and three dimensions.

In the following, some common yield criteria which can be expressed as conic inequalities are covered before describing the details specific to each of the lower and upper bound formulations.

1.2.2.1 Common yield criteria as conic constraints

Common yield criteria used in stability analysis include the Mohr-Coulomb and Drucker-Prager yield conditions. The Mohr-Coulomb criterion contains within it the Tresca yield criterion through an appropriate choice of variables, and, similarly, the Drucker-Prager (or extended von Mises) criterion is a generalised form of the von Mises yield condition. Both of these criteria may be formulated as conic constraints, as is shown next.

1.2.2.1.1 The Mohr-Coulomb criterion

The Mohr-Coulomb yield criterion is one of the most common yield conditions in use today, and includes, as a simplification through setting the friction angle equal to 0, the Tresca condition. In three dimensions, the Mohr-Coulomb criterion is equivalent to restricting the nodal stresses to lie within a semidefinite cone [28]–[30]. The stress at a point is defined by

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix}, \text{ with } \sigma_{ij} = \sigma_{ji},$$

and the Mohr-Coulomb yield criterion in three dimensions is of the form

$$f(\boldsymbol{\sigma}) = (1+\sin\phi)\sigma_1 - (1-\sin\phi)\sigma_3 - 2c\cos\phi \le 0, \tag{1.6}$$

where σ_1 is the maximum principal stress, σ_3 is the minimum principal stress, and tensile stresses are assumed positive. The principal stresses $(\sigma_1, \sigma_2, \sigma_3)$ are the eigenvalues of the stress tensor. It has been shown [28]–[30], [33] that (1.6) is equivalent to enforcing the linear matrix inequalities

$$\begin{aligned} &\boldsymbol{\sigma} + \lambda \mathbf{I} \ge \mathbf{0} \\ &(k - a\lambda)\mathbf{I} - \boldsymbol{\sigma} \ge 0 \end{aligned},$$

for $a = \dfrac{1-\sin\phi}{1+\sin\phi}$, and $k = \dfrac{2c\cos\phi}{1+\sin\phi}$, both non-negative. Krabbenhøft *et al.* [29] prove the equivalence by noting that the eigenvalues of $\mathbf{A} + \lambda\mathbf{I}$ are the same as the eigenvalues of $\mathbf{A}$ plus λ [34], which provides the inequalities

$$\begin{aligned} &\sigma_3 + \lambda \geq 0 \\ &(k - a\lambda) - \sigma_1 \geq 0 \end{aligned}.$$

When combined, these two relations give $\sigma_1 - a\sigma_3 \leq k$.

For plane strain conditions, the Mohr-Coulomb condition can instead be formulated as a second-order cone constraint [25], [26], [28]. Considering the reduced Mohr-Coulomb criterion for plane strain conditions (see, e.g., [35])

$$\sqrt{(\sigma_{xx} - \sigma_{yy})^2 + 4\sigma_{xy}^2} + (\sigma_{xx} + \sigma_{yy})\sin\phi - 2c\cos\phi \leq 0,$$

where c is the cohesion and ϕ is the angle of internal friction, it can be rearranged to an equality constraint and second-order cone constraint [28] using the auxiliary variables, $\mathbf{z}$:

$$\begin{aligned} &z_1 \geq \sqrt{z_2^2 + z_3^2} \\ &\mathbf{z} = \mathbf{D\sigma} + \mathbf{d} \\ &\mathbf{D} = \begin{bmatrix} \sin\phi & \sin\phi & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 2 \end{bmatrix}. \\ &\mathbf{d} = \begin{Bmatrix} 2c\cos\phi \\ 0 \\ 0 \end{Bmatrix} \end{aligned}$$

The first equation is represented with the conic inequality $\mathbf{z} \geq_K 0$. Thus, in plane-strain, the Mohr-Coulomb yield condition may be represented as a second-order cone inequality suitable for use in a second order cone program (SOCP).

1.2.2.1.2 Drucker-Prager yield criterion

Drucker and Prager present their yield criterion as “a proper generalization of the Mohr-Coulomb hypothesis” [2], although (1.6) is considered to be the true generalisation of the Mohr-Coulomb criterion from two to three dimensions [3]. The Drucker-Prager

yield criterion is a right circular cone (or cylinder) [2]. Because of this, the Drucker-Prager and von Mises yield criteria can be cast as second-order cone constraints. Thus, the lower and upper bound theorems can be applied to materials governed by these conditions to formulate a second-order cone program (also known as a conic quadratic program) [25], [26]. The yield criterion is defined as

$$\alpha J_1 + \sqrt{J_2} \le k, \tag{1.7}$$

where α and k are non-negative material constants, $J_1 = \sigma_1 + \sigma_2 + \sigma_3 = \sigma_{11} + \sigma_{22} + \sigma_{33}$ and $J_2 = \frac{1}{2} s_{ij} s_{ij} = \frac{1}{6}\left[(\sigma_{11} - \sigma_{22})^2 + (\sigma_{22} - \sigma_{33})^2 + (\sigma_{33} - \sigma_{11})^2 + \sigma_{12}^2 + \sigma_{23}^2 + \sigma_{13}^2 \right]$. In the definition of J_2, $s_{ij} = \sigma_{ij} - \frac{J_1}{3}\delta_{ij}$ where δ_{ij} is the Kronecker delta, $\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$.

Under a linear transformation, this is equivalent to the second order cone

$$k - \alpha \frac{J_1}{3} \ge \frac{1}{\sqrt{2}} \|\mathbf{s}\|.$$

For equality with the Mohr-Coulomb criterion in plane strain analyses, $c = \frac{k}{\sqrt{1 - 12\alpha^2}}$ and $\sin\phi = \frac{3\alpha}{\sqrt{1 - 3\alpha^2}}$ [2]. In three dimensions, one can choose the Lode angle θ for which the Drucker-Prager condition should equal the Mohr-Coulomb. Writing the Mohr-Coulomb criterion in the form

$$\frac{J_1}{3}\sin\phi + \sqrt{J_2}\cos\theta - \frac{\sqrt{J_2}}{3}\sin\phi\sin\theta - \mathrm{c}\cos\phi = 0,$$

then

$$\alpha = \frac{\sin\phi}{\cos\theta - \frac{1}{\sqrt{3}}\sin\phi\sin\theta} \tag{1.8}$$

and

$$k = \frac{c\cos\phi}{\cos\theta - \frac{1}{\sqrt{3}}\sin\phi\sin\theta}. \tag{1.9}$$

1.2.2.2 Lower bound formulation

The lower bound formulation seeks a statically admissible stress field $\boldsymbol{\sigma}$ in equilibrium throughout the soil mass and everywhere satisfying the yield criterion while maximising the load

$$Q = \int_{A_q} \mathbf{q} \mathrm{d}A + \int_V \mathbf{h} \mathrm{d}V ,$$

where $\mathbf{q}$ are a set of unknown surface tractions acting on A_q and $\mathbf{h}$ are unknown body forces acting on the volume V. The domain is discretised using linear finite elements, in which each node l of element e is associated with the unknown vector $\boldsymbol{\sigma}^l$ (for two-dimensional problems $\boldsymbol{\sigma}^l = \left\{\sigma_{xx}^l, \sigma_{yy}^l, \sigma_{xy}^l\right\}^T$, while in three-dimensional problems $\boldsymbol{\sigma}^l = \left\{\sigma_{xx}^l, \sigma_{yy}^l, \sigma_{zz}^l, \sigma_{xy}^l, \sigma_{yz}^l, \sigma_{xz}^l\right\}$). The statically admissible stress field must satisfy equilibrium in the continuum, the discontinuities between each element face (side of a triangle in two dimensions or a triangular face of a tetrahedron in three dimensions), and along the domain boundaries. Continuum and boundary equilibrium is described by the equations

$$\sum_{j=1}^{D} \frac{\partial \sigma_{ij}}{\partial x_j} + g_i + h_i = t_i + \mathrm{q}_i \text{ for } i = 1, \dots, D , \tag{1.10}$$

with g_i being the fixed body forces and t_i the fixed surface tractions (h_i and q_i are components of $\mathbf{q}$ and $\mathbf{h}$ above). Obviously, elements adjacent to the domain boundary may have surface tractions associated with them, while the interior elements will not. The above also assumes the surface traction components are in the Cartesian coordinate system. Considering the symmetry of the stress tensor ($\sigma_{ij} = \sigma_{ji}$ for $i \neq j$), and using the linear shape functions describing the stress variation over the element, the continuum equilibrium constraint (1.10) can be formulated for each element as

$$\mathbf{B}^e \boldsymbol{\sigma}^e = \alpha \mathbf{p}^e + \mathbf{p}_0^e , \tag{1.11}$$

where $\mathbf{B}^e$ is as described above, $\boldsymbol{\sigma}^e = \left\{\boldsymbol{\sigma}^1 \quad \boldsymbol{\sigma}^2 \quad \boldsymbol{\sigma}^3\right\}^T$ with $\boldsymbol{\sigma}^l$ as in (1.4) and (1.5), $\mathbf{p}_0^e = \left(\mathbf{t}^e - \mathbf{g}^e\right)$ are the prescribed surface tractions and body forces, and $\mathbf{p}^e = \left(\mathbf{q}^e - \mathbf{h}^e\right)$ represents the surface tractions and body forces to be optimised with the scalar α.

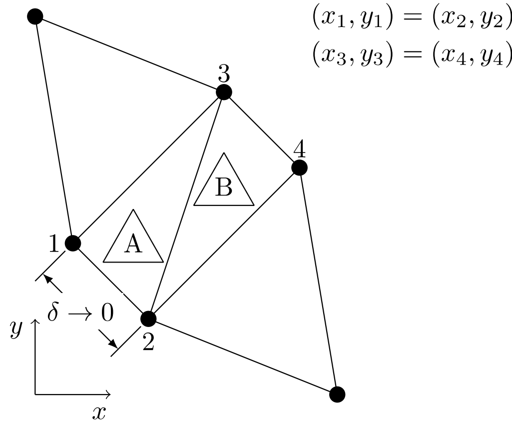


Figure 1. Stress discontinuity between two elements.

Statically admissible stress discontinuities are permitted at all inter-element interfaces. Across these discontinuities, the normal stress perpendicular to the interface and the shear stress must be continuous, but the tangential normal stress may jump between the elements. This constraint may be expressed by separating the mesh elements by discontinuity elements as shown in Figure 1. The equilibrium expression for the two triangular elements A and B forming a discontinuity patch can be expressed as

$$\begin{bmatrix} \mathbf{B}^A & \mathbf{0} \\ \mathbf{0} & \mathbf{B}^B \end{bmatrix} \begin{Bmatrix} \boldsymbol{\sigma}^A \\ \boldsymbol{\sigma}^B \end{Bmatrix} = \alpha \begin{Bmatrix} \mathbf{p}^A \\ \mathbf{p}^B \end{Bmatrix} + \begin{Bmatrix} \mathbf{p}_0^A \\ \mathbf{p}_0^B \end{Bmatrix}.$$

As the width of the discontinuity patch goes to zero, this ensures equality of the normal and shear stress across the discontinuity while allowing the tangential stress to jump. This simple approach permits statically admissible stress discontinuities by using the same constraint equalities on the patch elements as the regular elements of the mesh,

$$\mathbf{B}^T \boldsymbol{\sigma} = \alpha \mathbf{p} + \mathbf{p}_0 . \qquad (1.12)$$

All points throughout the continuum must have a state of stress which lies inside or on the yield surface of the material. As the stress varies linearly throughout the elements, the yield condition will be satisfied throughout if it is satisfied at the nodes. Thus, for each node in the mesh, an inequality constraint of the form

$$f\left(\boldsymbol{\sigma}^l\right) \leq_{\mathcal{K}} 0$$

will complete the requirements for a statically admissible stress field. This leads to the lower bound optimisation problem

$$\begin{aligned}&\text{maximise } \alpha \\ &\text{subject to } \mathbf{B}^T\boldsymbol{\sigma} = \alpha\mathbf{p} + \mathbf{p}_0 \\ &\qquad f(\boldsymbol{\sigma}^l) \leq_{\mathcal{K}} 0 \forall l = 1,2,...,n_n\end{aligned} \quad , \tag{1.13}$$

where n_n is the number of nodes in the mesh.

1.2.2.3 Upper bound formulation

The upper bound formulation described here follows Krabbenhøft *et al.* [22]. The weak form of the equilibrium equations (1.10) is

$$\int_V \dot{\mathbf{u}}^T \mathbf{L}^T \boldsymbol{\sigma} dV + \int_V \dot{\mathbf{u}}^T (\mathbf{g} + \mathbf{h}) dV - \int_{A_t + A_q} \dot{\mathbf{u}}^T (\mathbf{t} + \mathbf{q}) dA = \mathbf{0},$$

where $\mathbf{L}$ is the matrix of differential operators

$$\mathbf{L}^T = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

for three-dimensional problems, simplifying to

$$\mathbf{L}^T = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & \frac{\partial}{\partial y} \\ 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

in two dimensions. Note that this exploits the symmetry of the stress tensor. Using numerical integration with the shape functions (1.2) describing the velocities across each element, the following matrix expression for equilibrium is obtained for each element

$$\left(\mathbf{B}^e\right)^T \boldsymbol{\sigma}^e = \alpha \mathbf{p}^e + \mathbf{p}_0^e, \tag{1.14}$$

where $\mathbf{B}^e$ is described above, $\boldsymbol{\sigma}^e = \{\sigma_x^e \quad \sigma_y^e \quad \tau_{xy}^e\}$ for plane-strain problems and $\boldsymbol{\sigma}^e = \{\sigma_{xx}^e \quad \sigma_{yy}^e \quad \sigma_{zz}^e \quad \tau_{xy}^e \quad \tau_{yz}^e \quad \tau_{xz}^e\}$ for problems in three dimensions, $\mathbf{p}_0^e = \left(\mathbf{t}^e - \mathbf{g}^e\right)$

are the prescribed surface tractions and body forces, and $\mathbf{p}^e = \left(\mathbf{q}^e - \mathbf{h}^e\right)$ represents the surface tractions and body forces to be optimised with the scalar α. Note that the transpose of $\mathbf{B}^e$ is used and that the stress is constant across the element; this should not be confused with the linearly varying stress in the lower bound formulation. As with the lower bound formulation, only the elements on the boundary of the domain can be associated with surface tractions, and those tractions are assumed to be prescribed in the same coordinate system as the unknown stresses and body forces.

The weak form of equilibrium may also be applied to the element interfaces allowing for velocity discontinuities [22]. The normal stress components tangential to the element interface are allowed to differ on either side of each discontinuity. The normal stress component that is perpendicular to the interface and the shear stresses acting on the plane must, however, be continuous. The discontinuity is modelled as a thin “patch” of elements (two triangles in two dimensions and three tetrahedral elements in three), with the vertices of each patch element corresponding to those of the adjacent elements in a way identical to that in the lower bound method described previously.

By following the same approach as in the lower bound formulation to obtain the perpendicular normal stress, the equality constraints for a two-element mesh with a discontinuity at their interface are

$$\begin{bmatrix} \left(\mathbf{B}^A\right)^T & \mathbf{0} \\ \mathbf{0} & \left(\mathbf{B}^B\right)^T \end{bmatrix} \begin{Bmatrix} \boldsymbol{\sigma}^A \\ \boldsymbol{\sigma}^B \end{Bmatrix} + \mathbf{S}^T \begin{Bmatrix} \boldsymbol{\rho}^A \\ \boldsymbol{\rho}^B \end{Bmatrix} = \alpha \begin{Bmatrix} \mathbf{p}^A \\ \mathbf{p}^B \end{Bmatrix}.$$

Here, $\mathbf{S}^{A,B}$ again contains standard stress transformation matrices which convert the elemental stresses from the normal-tangent coordinate system for the discontinuity between elements A and B back to the Cartesian system. Note that the matrix $\mathbf{S}$ represented here is different to the one in the lower bound formulation, as the stress is constant in the upper bound formulation but varies linearly across the elements in the lower bound formulation. $\boldsymbol{\rho}^A = \dfrac{L}{2}\left\{\sigma_n^A \quad \tau^A\right\}^T$ (L is the length of the interface) contains the shear stress and perpendicular normal stress in the patch element adjacent to the first element for the discontinuity with the second element. The vector $\boldsymbol{\rho}^{B,A}$ is defined

similarly (note that it is the tangential stress component, σ_t, that is allowed to jump across the discontinuity and so does not appear in $\boldsymbol{\rho}^{e_1,e_2}$).

Again, the additional variables $\boldsymbol{\rho}^{e_1,e_2}$ may be substituted out of the problem, leaving the general form of the equality constraints as

$$\mathbf{B}^T\boldsymbol{\sigma} = \alpha\mathbf{p} + \mathbf{p}_0 . \tag{1.15}$$

To complete the problem formulation, the yield condition must not be violated anywhere. Since the element stresses are constant, this leads to inequality constraints of the form

$$f\left(\boldsymbol{\sigma}^e\right) \leq_{\mathcal{K}} 0 \tag{1.16}$$

for each element, including discontinuity elements.

Combining the equilibrium constraints with the yield inequalities leads to the upper bound optimisation problem

$$\begin{aligned} &\text{maximise } \alpha \\ &\text{subject to } \mathbf{B}^T\boldsymbol{\sigma} = \alpha\mathbf{p} + \mathbf{p}_0 \\ &\qquad f(\boldsymbol{\sigma}^e) \leq_{\mathcal{K}} 0 \forall e = 1,2,...,n_e \end{aligned} , \tag{1.17}$$

where n_e is the total number of elements. Equation (1.17) represents the dual of the conventional upper bound formulation that minimises the power dissipation subject to flow rule and compatibility constraints to ensure a kinematically admissible failure mechanism. This dual, or stress-based, upper bound formulation provides a more convenient problem that can be solved approximately twice as fast as the conventional formulation while still providing a rigorous upper bound on the true collapse load [22].

1.2.3 The FELA optimisation problem

As seen in the preceding sections, finite element limit analysis leads to the formulation of an optimisation problem of the form [28]

$$\begin{aligned} &\text{maximise } \alpha \\ &\text{subject to } \mathbf{A}\boldsymbol{\sigma} = \alpha\mathbf{p} + \mathbf{p}_0 \\ &\qquad f_i(\boldsymbol{\sigma}) \leq_{\mathcal{K}} 0 \forall i = 1,2,...,n_f \end{aligned} , \tag{1.18}$$

where $\mathbf{A}$ is the matrix of equality constraints, $\boldsymbol{\sigma}$ are the stresses, $\mathbf{p}$ and $\mathbf{p}_0$ are force vectors, α is the load multiplier, f represents the yield functions, $\leq_{\mathcal{K}}$ is some type of conic inequality, and n_f is the number of points that the yield criterion must be satisfied at. This can be cast into the canonical form for conic programs as

$$\begin{aligned}&\text{minimise } \mathbf{c}^T\mathbf{x}\\&\text{subject to } \mathbf{A}\mathbf{x} = \mathbf{b}\\&\qquad\qquad \mathbf{x}_i \geq_{\mathcal{K}_i} 0 \forall i = 1,2,...,n_k\end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^{n_k}$, $\mathbf{c}$ is the objective function, n_k is the number of cones, and $\geq_{\mathcal{K}_i}$ represents a general partial ordering over the cones $\mathcal{K}_i$. Common (and useful) cones include the nonnegative orthant $\mathbb{R}_n^+$ (corresponding to the common partial ordering over the real numbers, i.e. $\geq$), the Lorentz, quadratic, or second order cone (given a vector $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x} \geq_{\mathcal{K}_i} 0 \equiv x_n \geq \sqrt{\sum_{i=1}^{n-1} x_i^2}$), and the cone of semidefinite matrices (given a symmetric matrix $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n\times n}$, $\mathbf{A} \geq_{\mathcal{K}_i} 0 \equiv \mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$). These are known as linear programs (LPs), second order cone programs (SOCPs), and semidefinite programs (SDPs). Formulations involving variables with constraints from more than one of these cones are generally referred to as mixed cone linear programs or semidefinite, quadratic and linear programs (or SQLP) [36].

Although the solution to these optimisation problems has been obtained using various methods including the Simplex method [5], active set methods [12], augmented Lagrangian methods [31], and various nonlinear programming schemes, FELA solutions are most commonly obtained today using an interior point method (IPM). A brief overview of these approaches is provided below.

1.3 Interior point methods for conic programming

1.3.1 Background

Since the middle of the last century, linear programming has been a powerful framework for solving optimisation problems in the standard form

$$\begin{aligned} &\text{minimise } \mathbf{c}^T\mathbf{x} \\ &\text{subject to } \mathbf{A}\mathbf{x} = \mathbf{b}\,. \\ &\qquad\qquad \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{1.19}$$

or solving the dual problem

$$\begin{aligned} &\text{maximise } \mathbf{b}^T\mathbf{y} \\ &\text{subject to } \mathbf{A}^T\mathbf{y} \leq \mathbf{c} \end{aligned}$$

which is usually modified by the addition of the slack variables, $\mathbf{s}$, to convert the inequality constraints to equality constraints. This modified form of the dual problem then becomes

$$\begin{aligned} &\text{maximise } \mathbf{b}^T\mathbf{y} \\ &\text{subject to } \mathbf{A}^T\mathbf{y} + \mathbf{s} = \mathbf{c}\,. \\ &\qquad\qquad \mathbf{s} \geq \mathbf{0} \end{aligned} \tag{1.20}$$

Solution of these problems relied heavily on the introduction of the Simplex method by Dantzig [6] (although von Neumann is believed to have encountered it in his study of zero-sum two-person games [37]), and most advances in mathematical programming are still initially developed in linear programming. The Simplex algorithm was found, in practice, to solve most linear problems efficiently by moving from vertex to vertex on the boundary of the feasible region based on some heuristic or rule, demonstrating the combinatorial nature of even continuous problems. Development of the digital computer enabled larger and more complex problems to be solved that were often intractable only a few years earlier. With a growing demand for efficient algorithms to be used on these digital computers, there was a significant increase in research on algorithm complexity during the 1960s and 1970s. The Simplex algorithm was proven to have a worst-case iteration complexity which is exponential in the size of the problem, and that examples exist that force the algorithm to visit a large majority, if not all, of the feasible boundary's vertices [38]. Note that this did not reflect practitioner's experience on many real-world problems, and the Simplex method has been shown, in a probabilistic sense (or the expected performance), to be strongly polynomial [39]. The exponential complexity was, nevertheless, a disturbing feature of the method. This led researchers to seek out some provably polynomial method (or otherwise prove it does not exist).

In 1979, Khachiyan [44] showed that, by progressively reducing the size of an ellipsoid containing the optimal solution until the desired accuracy was achieved, a linear program could be solved polynomially in $O(n^2)$ iterations, where n is the number of unknowns. Despite the much improved worst-case complexity, the Simplex method was still far superior in practice [43]. Even so, the confirmation that a polynomially-bounded algorithm existed brought renewed attention to the field, and in 1984, Karmarkar [45] published the landmark paper describing what is now categorised as a primal projective potential reduction interior point method. This was followed by the recognition that the origins of Karmarkar's algorithm could be seen in the logarithmic barrier method for nonlinear optimisation [46] (indeed, it has since been proven that the basic logarithmic barrier method for linear programming has polynomial complexity [47]). Shortly thereafter, Renegar [48] published a primal method tracing the analytic centres of the successively smaller subsets of the feasible set using Newton's method. This procedure was a precursor to the development of the central path for linear programming, although the concept of the central path first appeared in the context of nonlinear complementarity problems in 1980 [49], and is now almost universally used in IPM implementations.

For linear programming, interior point methods have almost wholly supplanted the simplex and active-set linear programming algorithms, based not only on their better theoretical complexity, but also on their practical performance [40] (although, the warm-start ability of the Simplex method means it is still in use in cases where additional problems with slightly different constraints need to be solved). Through the use of their self-concordant theory, Nesterov and Nemirovskii [41] extended the polynomial complexity results to include any case where the a self-concordant barrier could be identified. The main class of problems are known as conic programs, and include linear programs (LP), second-order cone programs (SOCP) (which subsumes quadratic programming, or QP), and semidefinite programs (SDP). An SOCP can be cast as an SDP (and an LP cast as either an SOCP or SDP), but an SDP has more expressive power than an SOCP, meaning that some SDPs cannot be cast as an LP or SOCP. Furthermore, the iteration bounds on the three conic programs increase from LP to SOCP to SDP, providing incentive to work with the formulation that is most efficient to solve. The primal conic program is (as described above)

$$\begin{aligned}&\text{minimise } \mathbf{c}^T\mathbf{x}\\&\text{subject to } \mathbf{A}\mathbf{x}=\mathbf{b}\\&\qquad\qquad \mathbf{x}_i \geq_{\mathcal{K}_i} 0 \forall i=1,2,...,n_k\end{aligned} \tag{1.21}$$

while the dual conic program is

$$\begin{aligned}&\text{maximise } \mathbf{b}^T\mathbf{y}\\&\text{subject to } \mathbf{A}^T\mathbf{y}+\mathbf{s}=\mathbf{c}\\&\qquad\qquad \mathbf{s}_i \geq_{\mathcal{K}_i^*} 0 \forall i=1,2,...,n_k\end{aligned}, \tag{1.22}$$

where the dual Lorentz cones are $\mathcal{K}^* = \left\{\mathbf{s} \middle| \mathbf{s}^T\mathbf{x} \geq 0, \forall \mathbf{x} \in \mathcal{K}\right\}$.

Self-concordance is essentially a pair of differential inequalities concerning the first, second, and third directional derivatives of a three-times continuously differentiable convex barrier [42]. Alternatively, Peng *et al.* [43] have shown that by using a self-regular barrier (requiring a two-times continuously differentiable function, with two specific inequality conditions), the convergence complexity of the long-step path following scheme, which is known to be superior to the short-step method in practice, can get arbitrarily close to the theoretical short-step iteration bound. It is these conditions on the smoothness of the barrier that ensures the Newton method can identify points very close to the optimal point of each sub-problem (that is, points lying on the central path), usually in only one or very few iterations.

Interior point methods are usually either a potential reduction method or a path-following method, and can act on the primal or dual problem. In both theory and practice, work is almost entirely focussed on primal-dual methods [50]–[53], which utilise information from both the primal and the dual problem as an optimal solution is approached. The potential reduction methods use some measure to evaluate the quality of points in the feasible set along the search direction, while preventing the unknowns from prematurely reaching the boundary of the feasible set. Thus, these methods do not explicitly follow the central path. The path-following methods approximately trace out what is known as the central path by staying within some neighbourhood of it. The path-following approaches can be further split between short-step and long-step methods, with long-step methods being the superior approach in practice, despite the fact that short-step methods have long had better theoretical iteration bounds, although

long-step methods with self-regular barriers have been proven to have arbitrarily close iteration complexity to their short-step counterparts [43]. These algorithms include the infeasible path-following methods, which allow some infeasibility in the constraints, but ensure that feasibility is approached, usually as fast as or faster than the optimal solution convergence. Another widely-appreciated development for path-following methods was that of Mehrotra's predictor-corrector method, as well as a number of other small but effective implementation details [54]. Mehrotra's method uses the same factorisation first as an affine-scaling step, which effectively considers steps parallel to the central path, and then a combined centering and quadratic corrector step, which combines a direction towards the central path as well as towards the solution, adaptively selecting a suitable weighting between the centering direction and a direction approaching the solution. Mehrotra's predictor-corrector approach has been extended to include information from higher-order terms of the Taylor expansion that the direction is based on. The most notable of these is Gondzio's [55] multiple centrality correctors for LP (and later extended to SQLP [56]), which seek to increase the step length able to be taken in the found search direction rather than trying to follow the central path more closely.

Most of the common state-of-the-art implementations today embed the problem in a homogeneous self-dual (HSD) model (see, for example, [27], [36], [57]–[59]). This reformulated problem is a linear complementarity problem (LCP), where a standard LCP seeks an $(\mathbf{x},\mathbf{y}) \in \mathbb{R}^{2n}$ such that

$$\mathbf{y} = \mathbf{M}\mathbf{x} + \mathbf{q}, \mathbf{x} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0}, x_i y_i = 0 \forall i = 1,2,...,n, \quad (1.23)$$

$\mathbf{M} \in \mathbb{R}^{n \times n}$, $\mathbf{q} \in \mathbb{R}^n$, and $\mathbf{M}$ is usually restricted to be a P_0 matrix [60]. The class of P_0 matrices includes skew-symmetric, positive semidefinite, and positive definite matrices, among others. These problems have been extensively studied by a Japanese group led by Kojima [60], and may be solved efficiently by IPMs (Kojima *et al.* describe a unified interior point framework using both potential reduction and path-following concepts [60]). The original embedding of an LP into an LCP by Ye *et al.* [61] was later simplified into what is known as the simplified HSD formulation [62]. These were extended from LP to handle conic programs, leading to the standard form for SOCP to find a strictly complementary point satisfying

$$
\begin{aligned}
&\mathrm{mat}(\mathbf{x})\,\mathrm{mat}(\mathbf{s})\,\mathbf{e}^0 = \mathbf{0} \\
&\tau\kappa = 0 \\
&\begin{bmatrix} \mathbf{0} & -\mathbf{c}^T & \mathbf{b}^T \\ \mathbf{c} & \mathbf{0} & -\mathbf{A}^T \\ -\mathbf{b} & \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \tau \\ \mathbf{x} \\ \mathbf{y} \end{Bmatrix} = \begin{Bmatrix} \kappa \\ \mathbf{s} \\ \mathbf{0} \end{Bmatrix}, \\
&\mathbf{x}_i \geq_{\mathcal{K}_i} \mathbf{0}, \mathbf{s}_i \geq_{\mathcal{K}_i^*} \mathbf{0}, \tau \geq 0, \kappa \geq 0
\end{aligned}
\tag{1.24}
$$

where

$$
\mathbf{x}_i = \begin{Bmatrix} x_i^0 \\ \overline{\mathbf{x}}_i^T \end{Bmatrix},
$$

$$
\mathrm{mat}(\mathbf{x}_i) = \begin{bmatrix} x_i^0 & \overline{\mathbf{x}}_i^T \\ \overline{\mathbf{x}}_i & x_i^0 \mathbf{I} \end{bmatrix},
$$

$$
\mathrm{mat}(\mathbf{x}) = \mathrm{diag}\left(\mathrm{mat}(\mathbf{x}_i), \ldots, \mathrm{mat}(\mathbf{x}_{n_k})\right) = \begin{bmatrix} \mathrm{mat}(\mathbf{x}_i) & & \\ & \ddots & \\ & & \mathrm{mat}(\mathbf{x}_{n_k}) \end{bmatrix}, \quad \mathbf{e}^0 = \begin{Bmatrix} \mathbf{e}^1 \\ \vdots \\ \mathbf{e}^{n_k} \end{Bmatrix},
$$

$$
\mathbf{e}^i = \begin{Bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{Bmatrix} \in \mathbb{R}^{n_i},
$$

and n_i is size of the i th cone

The coefficient matrix in (1.24) is clearly skew-symmetric and the slacks $\mathbf{s}$ and κ convert the homogeneous system with inequalities to this system of equalities. The solution to the reformulated problem provides a solution to the original or it indicates that the original problem is infeasible. The HSD and simplified HSD forms ensure that the problem being solved will always have a solution, even if the original problem does not. Note that the simplified HSD formulation does not have any strictly feasible points, but the HSD formulation does [62].

Generally, the path-following IPMs considered in this Thesis proceed in a similar fashion to the simplified steps shown below. This primal-dual framework incorporates the simplified HSD formulation with Mehrotra's predictor-corrector search direction.

$\left(\mathbf{x},\tau,\mathbf{y},\mathbf{s},\kappa\right)=\left(\mathbf{x}^{0},\tau^{0},\mathbf{y}^{0},\mathbf{s}^{0},\kappa^{0}\right)$

Do until converged to tolerance ε

Determine affine-scaling search direction, $\left(\mathbf{d}_x^{aff},\mathrm{d}_\tau^{aff},\mathbf{d}_y^{aff},\mathbf{d}_s^{aff},\mathrm{d}_\kappa^{aff}\right)$

Determine step size, α^{aff}

Determine corrector search direction, $\left(\mathbf{d}_x^{cor},\mathrm{d}_\tau^{cor},\mathbf{d}_y^{cor},\mathbf{d}_s^{cor},\mathrm{d}_\kappa^{cor}\right)$

Determine α for combined search direction

Update $\left(\mathbf{x},\tau,\mathbf{y},\mathbf{s},\kappa\right)=\left(\mathbf{x},\tau,\mathbf{y},\mathbf{s},\kappa\right)+\alpha\left(\mathbf{d}_x,\mathrm{d}_\tau,\mathbf{d}_y,\mathbf{d}_s,\mathrm{d}_\kappa\right)$

End

Here, $\mathbf{x}$ are the primal unknowns, $\mathbf{s}$ the dual unknowns, $\varepsilon\geq 0$ is the convergence tolerance of the IPM scheme, and α is the step size variable damping factor controlling the length of the step taken in the Newton direction to improve convergence when far from the solution. The scheme outlined here assumes a primal-dual starting point that lies within the neighbourhood of the central path (that is, $\Phi\left(\mathbf{x}^{0},\mathbf{s}^{0},\mu^{0}\right)\leq\tau$). The search direction, $\left(\mathbf{d}_x,d_\tau,\mathbf{d}_y,\mathbf{d}_s,d_\kappa\right)$, is computed by solving a system of linear equations of the form

$$\begin{bmatrix} \mathbf{A} & -\mathbf{b} & & & \\ & -\mathbf{c} & \mathbf{A}^T & \mathbf{I} & \\ -\mathbf{c}^T & & \mathbf{b}^T & & -1 \\ \tilde{\mathbf{S}} & & & \tilde{\mathbf{X}} & \\ & \kappa & & & \tau \end{bmatrix} \begin{Bmatrix} \mathbf{d}_x \\ d_\tau \\ \mathbf{d}_y \\ \mathbf{d}_s \\ d_\kappa \end{Bmatrix} = \begin{Bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ r_3 \\ \mathbf{r}_4 \\ r_5 \end{Bmatrix}, \tag{1.25}$$

where $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{S}}$ are matrices based on the previous iterates $\mathbf{x}$ and $\mathbf{s}$ (described below), and the right-hand-side depends on the residuals in the problem constraints (the "infeasibility"), some parameters, and whether the predictor or a corrector direction is being computed. For feasible path-following and potential reductions methods, the starting point must be feasible (that is, the points must satisfy $\mathbf{Ax}=\mathbf{b}$ and $\mathbf{A}^{\mathrm{T}}\mathbf{y}+\mathbf{s}=\mathbf{c}$).

In development of an approach to solve FELA optimisation problems, it is crucial to be as efficient as possible at solving both SOCPs and SDPs because of the common material idealisations utilised in analyses. Thus, an ideal optimisation scheme must be effective for both formulations or else each formulation must be addressed individually.

1.3.2 The search direction in interior point methods

The Newton method for unconstrained optimisation approximates the objective function f by a second order Taylor expansion

$$f(\mathbf{y}) = f(\mathbf{x}) + (\mathbf{y}-\mathbf{x})^T f^{'}(\mathbf{x}) + \frac{1}{2}(\mathbf{y}-\mathbf{x})^T f^{''}(\mathbf{x})(\mathbf{y}-\mathbf{x}).$$

Using the solution to this approximation, a solution estimate to the original problem is obtained. Close to the solution, Newton's method exhibits quadratic convergence behaviour. Unfortunately, away from the solution the behaviour can be somewhat erratic and so it is usually damped through the use of a damping parameter, α, determined by a line search to minimise the objective along the computed search direction. This ensures the desired convergence behaviour as we approach the solution while avoiding the undesirable behaviour early in the search process.

The Karush-Kuhn-Tucker (KKT) conditions specify the necessary and sufficient conditions for an optimal point in the optimisation problem. The Newton search direction required in the interior point algorithm is the solution obtained from the perturbed Newton system obtained from the Karush-Kuhn-Tucker equations. The Newton system for an SOCP is obtained by applying Newton's method to the mildly nonlinear KKT system

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ \mathbf{A}^T\mathbf{y} + \mathbf{s} &= \mathbf{c} \\ \mathrm{mat}(\mathbf{x})\mathrm{mat}(\mathbf{s}) &= \mathbf{0} \end{aligned} \tag{1.26}$$

or to (1.24) for simplified HSD formulation. Applying Newton's method to (1.26) and perturbing the system in the third equation by $\mu\mathbf{e}^0$ gives

$$\begin{bmatrix} \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}^T & \mathbf{I} \\ \mathrm{mat}(\mathbf{x}) & \mathbf{0} & \mathrm{mat}(\mathbf{s}) \end{bmatrix} \begin{Bmatrix} \mathbf{d}_x \\ \mathbf{d}_y \\ \mathbf{d}_s \end{Bmatrix} = \begin{Bmatrix} -(\mathbf{Ax}-\mathbf{b}) \\ -(\mathbf{A}^T\mathbf{y}+\mathbf{s}-\mathbf{c}) \\ \mu\mathbf{e}^0 - \mathrm{mat}(\mathbf{x})\mathbf{s} \end{Bmatrix}. \tag{1.27}$$

This is the linearised Newton direction for an SOCP. This system has a unique solution if and only if $\mathbf{A}\mathrm{mat}(\mathbf{s})^{-1}\mathrm{mat}(\mathbf{x})\mathbf{A}^T$ is non-singular [43]. Even for strictly feasible primal-dual pairs, this is not necessarily true, and, therefore, the Newton search direction is not well-defined for an SOCP or SDP in general [43]. This is addressed by

scaling, where the pair $(\mathbf{x},\mathbf{s})$ is scaled to $(\mathbf{Fx},\mathbf{F}^{-1}\mathbf{s})$ where $\mathbf{F}$ is a scaling matrix. This scaling is also performed in IPM implementations for LP. There are numerous different scaling schemes that have been published (for a discussion on scaling for semidefinite programming, refer to [63]). Common scaling schemes in the literature lead to the AHO direction [64], the primal or dual HKM direction [65], and the NT direction [66], [67]. The AHO direction assumes an identity scaling (resulting in the AHO search direction being the solution of (1.27), i.e. $\mathbf{F}=\mathbf{I}$). The primal HKM direction is obtained by setting $\mathbf{Fx}=\mathbf{e}$ (where $\mathbf{e}$ is the vector of 1's), while the dual HKM direction sets $\mathbf{F}^{-1}\mathbf{s}=\mathbf{e}$. By setting $\mathbf{Fx}=\mathbf{F}^{-1}\mathbf{s}$, the NT direction is obtained. Interestingly, by considering which scaling leads to the optimal approximate solution in the sense of minimising the duality gap plus the barrier function applied to the primal and dual solution approximations, it is found that the NT scaling is optimal. Furthermore, Tsuchiya [68], [69] has shown that the NT direction has a lower theoretical iteration bound than other scaling approaches in long-step path-following methods. It is also the scaling used in popular SOCP packages (see, for example, [57], [70]).

Defining the scaling matrix $\mathbf{F}$ such that it satisfies $\mathbf{Fx}=\mathbf{F}^{-1}\mathbf{s}=\mathbf{v}$, the scalar $\mu=\dfrac{\mathbf{x}^T\mathbf{s}+\tau\kappa}{n_k+1}$, and parameterising the central path with γ, applying the NT scaling leads to the scaled and perturbed Newton search direction for the simplified HSD formulation allowing for infeasible iterates is given by

$$\begin{bmatrix} \mathbf{A} & -\mathbf{b} & & & \\ & -\mathbf{c} & \mathbf{A}^T & \mathbf{I} & \\ -\mathbf{c}^T & & \mathbf{b}^T & & -1 \\ \operatorname{mat}\left(\mathbf{F}^{-1}\mathbf{s}\right)\mathbf{F} & & & \operatorname{mat}\left(\mathbf{Fx}\right)\mathbf{F}^{-1} & \\ & \kappa & & & \tau \end{bmatrix} \begin{Bmatrix} \mathbf{d}_x \\ d_\tau \\ \mathbf{d}_y \\ \mathbf{d}_s \\ d_\kappa \end{Bmatrix} = \begin{Bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ r_3 \\ \mathbf{r}_4 \\ r_5 \end{Bmatrix}, \qquad (1.28)$$

where

$$\begin{Bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ r_3 \\ \mathbf{r}_4 \\ r_5 \end{Bmatrix} = \begin{Bmatrix} (\gamma-1)(\mathbf{Ax}-\mathbf{b}\tau) \\ (\gamma-1)\left(\mathbf{A}^T\mathbf{y}+\mathbf{s}-\mathbf{c}\tau\right) \\ (\gamma-1)\left(\mathbf{b}^T\mathbf{y}-\mathbf{c}^T\mathbf{x}-\kappa\right) \\ \gamma\mu\mathbf{e}-\operatorname{mat}(\mathbf{Fx})\operatorname{mat}\left(\mathbf{F}^{-1}\mathbf{s}\right)\mathbf{e} \\ \gamma\mu-\tau\kappa \end{Bmatrix}. \tag{1.29}$$

Note that the infeasibility of the solution approximations may be zero for algorithms that maintain feasibility for all iterations, but this will not be considered further as this generally requires identifying an initial point which is both primal and dual feasible. Infeasible iterates are generally allowed in IPM implementations which simplifies issues surrounding initial points and rounding errors (which can introduce infeasibility, even for well-conditioned coefficient matrices), and the IPM can progress towards optimality whilst simultaneously reducing the infeasibility in the approximation. Mehrotra's predictor-corrector method provides a worthwhile reduction in the number of iterations needed by the IPM to reach a solution by first computing a pure Newton search direction (also referred to as a predictor or affine-scaling direction), achieved by solving (1.28) with $\gamma = 0$, and then taking a corrector step. The corrector step addresses the fact that the Newton direction ignores the quadratic term $\operatorname{mat}(\mathbf{d}_x)\operatorname{mat}(\mathbf{d}_s)$ in linearising the third equation in (1.26). By approximating this term after computing the predictor direction by $\operatorname{mat}\left(\mathbf{d}_x^p\right)\operatorname{mat}\left(\mathbf{d}_s^p\right)$ (the superscript p indicating that they are components of the predictor direction defined as the solution to (1.28) with $\gamma = 0$), the corrector direction is defined as the solution to the system with the same coefficient matrix as in (1.28) but the right hand side

$$\begin{Bmatrix} (\gamma-1)(\mathbf{Ax}-\mathbf{b}\tau) \\ (\gamma-1)\left(\mathbf{A}^T\mathbf{y}+\mathbf{s}-\mathbf{c}\tau\right) \\ (\gamma-1)\left(\mathbf{b}^T\mathbf{y}-\mathbf{c}^T\mathbf{x}-\kappa\right) \\ \gamma\mu\mathbf{e}-\operatorname{mat}(\mathbf{Fx})\operatorname{mat}\left(\mathbf{F}^{-1}\mathbf{s}\right)\mathbf{e}-\operatorname{mat}\left(\mathbf{Fd}_x^p\right)\operatorname{mat}\left(\mathbf{F}^{-1}\mathbf{d}_s^p\right)\mathbf{e} \\ \gamma\mu-\tau\kappa-d_\tau^p d_\kappa^p \end{Bmatrix}. \tag{1.30}$$

The system of equations defined by (1.28) can be reduced to a 3×3 system through elimination of

$$d_\kappa = \frac{1}{\tau}\left(r_5 - \kappa d_\tau\right) \tag{1.31}$$

and

$$\mathbf{d}_s = \mathbf{F}\,\mathrm{mat}\left(\mathbf{Fx}\right)^{-1}\left(\mathbf{r}_4 - \mathrm{mat}\left(\mathbf{F}^{-1}\mathbf{s}\right)\mathbf{F}\mathbf{d}_x\right), \tag{1.32}$$

giving

$$\begin{bmatrix} \mathbf{0} & \mathbf{A}^T & -\mathbf{c} \\ \mathbf{A} & \mathbf{0} & -\mathbf{b} \\ -\mathbf{c}^T & \mathbf{b}^T & \dfrac{\kappa}{\tau} \end{bmatrix} \begin{Bmatrix} \mathbf{d}_x \\ \mathbf{d}_y \\ d_\tau \end{Bmatrix} = \begin{Bmatrix} \mathbf{r}_2 - \mathbf{F}\,\mathrm{mat}\left(\mathbf{Fx}\right)^{-1}\mathbf{r}_4 \\ \mathbf{r}_1 \\ r_3 + \dfrac{r_5}{\tau} \end{Bmatrix}. \tag{1.33}$$

Further elimination of $\mathbf{d}_x$ and $\mathbf{d}_y$ from the last equation gives the expression for d_τ

$$\left(\frac{\kappa}{\tau} + \begin{Bmatrix} -\mathbf{c} \\ \mathbf{b} \end{Bmatrix}^T \begin{bmatrix} -\mathbf{F}^2 & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix}^{-1} \begin{Bmatrix} \mathbf{c} \\ \mathbf{b} \end{Bmatrix}\right) d_\tau = r_3 + \frac{r_5}{\tau} - \begin{Bmatrix} -\mathbf{c} \\ \mathbf{b} \end{Bmatrix}^T \begin{bmatrix} -\mathbf{F}^2 & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix}^{-1} \begin{Bmatrix} \mathbf{r}_2 - \mathbf{F}\,\mathrm{mat}\left(\mathbf{Fx}\right)^{-1}\mathbf{r}_4 \\ \mathbf{r}_1 \end{Bmatrix}. \tag{1.34}$$

Computing the vectors $\begin{Bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \end{Bmatrix}$ and $\begin{Bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{Bmatrix}$ such that they satisfy

$$\begin{bmatrix} -\mathbf{F}^2 & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \end{Bmatrix} = \begin{Bmatrix} \mathbf{c} \\ \mathbf{b} \end{Bmatrix} \tag{1.35}$$

and

$$\begin{bmatrix} -\mathbf{F}^2 & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{Bmatrix} = \begin{Bmatrix} \mathbf{r}_2 - \mathbf{F}\,\mathrm{mat}\left(\mathbf{Fx}\right)^{-1}\mathbf{r}_4 \\ \mathbf{r}_1 \end{Bmatrix}, \tag{1.36}$$

respectively, allows d_τ to be computed as

$$d_\tau = \frac{r_3 + \dfrac{r_5}{\tau} + \mathbf{c}^T\mathbf{h}_1 - \mathbf{b}^T\mathbf{h}_2}{\dfrac{\kappa}{\tau} - \mathbf{c}^T\mathbf{g}_1 + \mathbf{b}^T\mathbf{g}_2}. \tag{1.37}$$

These components can be combined to give

$$\begin{Bmatrix} \mathbf{d}_x \\ \mathbf{d}_y \end{Bmatrix} = \begin{Bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{Bmatrix} + d_\tau \begin{Bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \end{Bmatrix}. \tag{1.38}$$

Finally, d_κ and $\mathbf{d}_s$ can be computed from (1.31) and (1.32), respectively. As has been shown, the bulk of the computational effort in obtaining the search direction is in solving systems with the 2×2 block matrix in (1.35) and (1.36). Using a direct method allows the additional solve to be computed at a significantly lower cost than that of factorising the matrix, and this same factorisation may also be used for the corrector direction components. This system is known as the augmented or KKT system. Systems with a block structure like this are also known as saddle point systems and they arise in many applications [71]. Note that it is symmetric but indefinite and, in practice, this KKT matrix is usually reduced further, allowing (1.35) to be solved in two steps as

$$\mathbf{AF}^{-2}\mathbf{A}^T\mathbf{g}_2 = \mathbf{b} + \mathbf{AF}^{-2}\mathbf{c} \tag{1.39}$$

and then

$$\mathbf{g}_1 = -\mathbf{F}^{-2}\left(\mathbf{c} - \mathbf{A}^T\mathbf{g}_2\right). \tag{1.40}$$

Note that equation (1.36) can clearly be reduced and solved in the same way. The system $\mathbf{AF}^{-2}\mathbf{A}^T$ is symmetric positive definite (SPD), and is usually solved using Cholesky decomposition. In practice (that is, in finite precision on a modern computer), (1.39) is sometimes symmetric positive semidefinite (or even symmetric indefinite) due to ill-conditioning in $\mathbf{F}^2$. When the matrix becomes semidefinite or indefinite, the system gets more difficult to factorise, as the diagonal entries are no longer guaranteed to be stable pivots [72]. While most implementations, especially for SOCP, use a direct solver to compute the search direction, in the early iterations of the IPM for LP and SDP, this system can generally be solved using the preconditioned conjugate gradient method (often coupled with an incomplete Cholesky decomposition as a preconditioner), but, because of severe increase in the condition number of the coefficient matrix, it becomes much more difficult to solve with the iterative methods as the IPM approaches a solution. Note that the corresponding systems defining the search directions for SDP are defined in similar way, making allowances for the unknowns now being matrices instead of vectors and adjusting the algebra used to define the complementarity condition.

If the time to solve each system is not effectively negligible compared with the factorisation, then it may be advantageous to solve for the $\mathbf{g}$ and $\mathbf{h}$ variables simultaneously as

$$\mathbf{AF}^{-2}\mathbf{A}^T\left[\mathbf{g}_2 \quad \mathbf{h}_2\right] = \left[\mathbf{b} + \mathbf{AF}^{-2}\mathbf{c} \quad \mathbf{r}_1 - \mathbf{AF}^{-2}\mathbf{r}_2 + \mathbf{AF}^{-1}\,\mathrm{mat}\left(\mathbf{Fx}\right)^{-1}\mathbf{r}_4\right]. \quad (1.41)$$

By solving for the two variables at once allows more arithmetic operations per load for the components in the coefficient matrix factorisation (or the coefficient matrix and any preconditioner in the situation of a preconditioned iterative solver). It also provides more scope for exploiting the vector-capable hardware in most modern desktop machines.

1.3.3 Handling free variables

In addition to the conically constrained variables discussed so far, there are those scalar variables which are not constrained, i.e. $x_f \in \mathbb{R}$. These are known as free variables and are denoted by a subscript f. The handling of free variables in solving the optimisation problem can play a significant part in the stability and runtime performance of the approach used [73]. The standard form of the problem considered in this section is then

$$\begin{aligned} &\text{minimise } \mathbf{c}^T\mathbf{x} + \mathbf{f}^T\mathbf{x}_f \\ &\text{subject to } \mathbf{Ax} + \mathbf{Ex}_f = \mathbf{b} \\ &\qquad\qquad \mathbf{x}_i \geq_{\mathcal{K}_i} 0 \forall i = 1,2,\ldots,n_k \end{aligned}$$

and the dual

$$\begin{aligned} &\text{maximise } \mathbf{b}^T\mathbf{y} \\ &\text{subject to } \mathbf{A}^T\mathbf{y} + \mathbf{s} = \mathbf{c} \\ &\qquad\qquad \mathbf{E}^T\mathbf{y} = \mathbf{f} \\ &\qquad\qquad \mathbf{s}_i \geq_{\mathcal{K}_i^*} 0 \forall i = 1,2,\ldots,n_k \end{aligned}.$$

There are a few different approaches to dealing with free variables in IPMs:

1. Solving the resulting indefinite Newton search direction equation directly. The coefficient matrix in the augmented system in this case becomes

$\begin{bmatrix} -\mathbf{F}^2 & \mathbf{0} & \mathbf{A}^T \\ \mathbf{0} & \mathbf{0} & \mathbf{E}^T \\ \mathbf{A} & \mathbf{E} & \mathbf{0} \end{bmatrix}$, thus the (1,1) block (shown here as 2×2) is singular and so any scheme which relies on the non-singularity of this block cannot be used. This excludes reducing the augmented system to symmetric positive definite form, instead arriving at the indefinite coefficient system $\begin{bmatrix} \mathbf{AF}^{-2}\mathbf{A}^{\mathrm{T}} & \mathbf{E}^{\mathrm{T}} \\ \mathbf{E} & \mathbf{0} \end{bmatrix}$.

2. Eliminating the free variables from the problem by finding a suitable basis in $\mathbf{E}$ and converting the problem [74]. If we join $\mathbf{A}$ and $\mathbf{E}$ as $[\mathbf{A} \quad \mathbf{E}]$, and permute the constraint matrix and the vectors $\mathbf{b}$ and $\mathbf{y}$ such that we have the partitions $\begin{bmatrix} \mathbf{A}_B & \mathbf{E}_B \\ \mathbf{A}_N & \mathbf{E}_N \end{bmatrix}$, $\begin{Bmatrix} \mathbf{b}_B \\ \mathbf{b}_N \end{Bmatrix}$, and $\begin{Bmatrix} \mathbf{y}_B \\ \mathbf{y}_N \end{Bmatrix}$, where $\mathbf{E}_B$ is a suitable (full rank) basis of $\mathbf{E}$ and all other partitions are permuted accordingly. The free variables can then be eliminated from the problem and the constraint matrix becomes $\bar{\mathbf{A}} = \mathbf{A}_N - \mathbf{E}_N \mathbf{E}_B^{-1} \mathbf{A}_B$, the primal and dual objective functions become $\mathbf{f}^T \mathbf{E}_B^{-1} \mathbf{b}_B + \left(\mathbf{c}^T - \mathbf{f}^T \mathbf{E}_B^{-1} \mathbf{A}_B\right)\mathbf{x}$ and $\mathbf{b}_B^T \mathbf{E}_B^{-1} \mathbf{f} + \left(\mathbf{b}_N^T - \mathbf{b}_B^T \mathbf{E}_B^{-T} \mathbf{E}_N^T\right)\mathbf{y}_N$, respectively, and the primal and dual systems of equality constraints become $\bar{\mathbf{A}}\mathbf{x} = \mathbf{b}_N - \mathbf{E}_N \mathbf{E}_B^{-1} \mathbf{b}_B$ and $\bar{\mathbf{A}}^T \mathbf{y}_N + \mathbf{s} - \mathbf{c} + \mathbf{A}_B^T \mathbf{E}_B^{-T} \mathbf{f} = \mathbf{0}$. The sparsity of the constraint matrix, and more importantly the Schur complement matrix $\bar{\mathbf{A}}\mathbf{F}^{-2}\bar{\mathbf{A}}^T = \mathbf{A}_N \mathbf{F}^{-2} \mathbf{A}_N^T + \mathbf{E}_N \mathbf{E}_B^{-1} \mathbf{A}_B \mathbf{F}^{-2} \mathbf{A}_B^T \mathbf{E}_B^{-T} \mathbf{E}_N^T$, may be affected severely. Some solver packages will consider eliminating the free variables in a presolve phase and eliminate any of them (not necessarily all of them) if the benefits of the reduced problem size outweigh the cost of a more dense constraint matrix (see, for example, [75]).

3. By using a "slack" variable to convert each free variable into two linear variables. The original free variable is then the difference between two linear variables and so $x_f = x_p - x_n$, where x_p and x_n are known as the positive and negative parts, respectively, of x_f. This approach often leads to numerical issues related to variable growth. This is a direct consequence of the

unboundedness of the new variables, that is x_p and x_n can be arbitrarily large for any x_f, even if x_f is tiny or zero. This is known to cause numerical difficulties as it leaves the solution set unbounded (that is, both x_p and x_n can be arbitrarily large but still represent the same value for x_f) [76].

4. Perturbing, or regularising, the augmented equations so that the components of the $(1,1)$ block corresponding to the free variables have a small but non-zero value, leading to the augmented system $\begin{bmatrix} -\mathbf{F}^2 & \mathbf{0} & \mathbf{A}^T \\ \mathbf{0} & -\delta\mathbf{I} & \mathbf{E}^T \\ \mathbf{A} & \mathbf{E} & \mathbf{0} \end{bmatrix}$. This allows the system to be reduced into the SPD form $\mathbf{AF}^{-2}\mathbf{A}^T + \delta^{-1}\mathbf{EE}^T$ and solved by a Cholesky solver. This has the obvious downside that the direction is no longer a true Newton direction but instead an inexact Newton direction (even when solved exactly), and, depending on the value of δ, this can often impact on numerical performance. The perturbation value has conventionally been fixed, although adaptive approaches have been developed that consider global convergence theory with positive results on SDPs and SQLPs [77]. Unfortunately, the bound on $\delta^{(k)}$ guaranteeing global convergence may necessitate refactorisation and recomputing the search direction if the $\delta^{(k)}$ is too large (possibly more than once) [77].
5. By embedding the free variables in a second-order cone (Andersen, 2002, cited in [77]). This approach imposes a second-order cone constraint on the free variable x_f such that $x_c \geq \|x_f\|$, where x_c is an upper bound on the magnitude of x_f. Note that this can be extended to grouping some or all of the free variables instead of using a separate cone for each variable, although increasing the size of the cones may affect the sparsity of the Schur complement system.

Of these schemes, the last three were considered in preliminary simulations with the quadratic cone embedding with one free variable per cone providing the best performance in terms and numerical stability, but not necessarily runtime. Compared to the perturbation method and the addition of slack variables, embedding the free

variables in a second-order cone generally leads to an increase in the number of non-zeros in the $(1,1)$ block of the augmented system and also in the Schur complement. The diagonal "block" associated with a free variable in the $(1,1)$ block of the augmented systems are based on the NT scaling matrices for the second-order cone embedding, which also requires additional work to compute. In contrast, the perturbation method and addition of slack variables leads to a diagonal "block" (the perturbation method has a 1×1 diagonal matrix and the addition of slacks leads to a 2×2 diagonal matrix).

Chapter 2 Computing the search direction in IPMs

Traditionally, the systems defining the search direction have been solved using direct solution methods, specifically, Cholesky decomposition on the normal equations, as this approach is generally much more robust than iterative solution schemes [78]. Unfortunately, when using state-of-the-art direct solution methods, such as `MA57` [79], it is not uncommon to observe the ratio of storage between the factorisation of $\mathbf{A}$ and $\mathbf{A}$ itself to exceed several orders of magnitude. For three-dimensional and large-scale two-dimensional problems that have refined or large meshes, the time complexity to construct such large factors and the space complexity involved in storing them is very expensive. Furthermore, because of the increasingly ill-conditioned nature of the system defining the search direction as the optimisation scheme nears a solution, direct methods become susceptible to round-off error, and may not be able to compute sufficiently accurate search directions.

A potential solution to storage and runtime requirements is sought using iterative solution methods. These methods may also provide improved solution approximations when direct methods fail to compute sufficiently accurate search directions. In the available conic optimisation package `SeDuMi` [58], if the solution obtained using the Cholesky decomposition does not satisfy some accuracy requirement, the factorisation is then used as a preconditioner for the PCG method [80] which attempts to improve the determined direction. Similarly, in `SDPT3` [70], [81] the Cholesky factorisation is always used as a preconditioner in the symmetric QMR method [82]. Iterative refinement schemes are also used to ensure that the norm of the residual is small enough to provide a useful direction [83]. Iterative solution methods attempt to solve a given linear system without having to fully factor the coefficient matrix, and generally only requires a few additional vectors to be stored with the most expensive operation performed being matrix-vector multiplications. This means that, for a given system of dimension n, the time-complexity is reduced from $O\left(n^3\right)$ for direct methods to $O\left(n^2\right)$ for iterative methods. Similarly, the additional storage requirements are reduced from $O\left(n^2\right)$ for direct solution methods to $O(n)$ for iterative schemes. Note that, in practice, sparsity is

exploited for both direct and iterative schemes, making the asymptotic complexities significantly less than that described, although the basic difference in the complexity of the underlying methods remains.

The convergence of iterative solution schemes depends in a non-trivial way on the distribution of the eigenvalues and the condition number of the coefficient matrix. The augmented system and the larger system with coefficient matrix (1.28) defining the search direction have both positive and negative eigenvalues, which will generally mean slower convergence will be exhibited by a Krylov subspace solver. All forms of the search direction generally have increasingly large condition numbers as the IPM approaches a solution. Thus the use of preconditioners to improve the spectrum and conditioning of the coefficient matrices is necessary. Unfortunately, the common algebraic preconditioners (e.g. incomplete factorisations and approximate inverses) often do not significantly improve convergence due to the presence of indefiniteness, large condition numbers, a lack of diagonal dominance, and the absence of decay in the coefficient matrices in question [78]. Where possible, exploitation of the block-structure and details of the block components is necessary. This leads to the development of highly specialised preconditioners, which are often not suitable for solving systems outside their intended use. It is because of these difficulties, and the robustness of the direct solvers with their high performance on modern computers, that has seen the widespread use of direct solvers, and in particular sparse Cholesky factorisations along with the exploitation of supernodes, in state-of-the-art IPM implementations. A brief overview of direct methods is given below, providing a solid grounding for the development of effective preconditioners for the iterative methods discussed afterwards.

2.1 Direct solution schemes

The most common direct solution schemes for sparse matrices rely heavily on the relative ease of solving a triangular system. Popular direct methods such as $\mathbf{LU}$ and $\mathbf{LDU}$ factorisations are equivalent variants of Gaussian elimination for reduction to a product of lower and upper triangular matrices, $\mathbf{L}$ and $\mathbf{U}$, respectively (and a diagonal matrix, $\mathbf{D}$, in the latter case), which are amenable to providing a solution to systems of the general form $\mathbf{Ax} = \mathbf{LDUx} = \mathbf{b}$ by first solving $\mathbf{Lc} = \mathbf{b}$, then $\mathbf{Dd} = \mathbf{c}$, and finally $\mathbf{Ux} = \mathbf{d}$. Different computational orders for computing the same (in exact arithmetic)

factors provide equivalent approaches to Gaussian elimination while allowing better use of different hardware architectures for efficiency and performance [84]. These methods are generally referred to as up-looking, left-looking, and right-looking, and lead to the three main approaches used in available packages, up-looking for very sparse matrices, supernodal factorisation, and frontal methods, respectively [85]. The up-looking methods compute one row of the factors at each step, the left-looking methods compute one column each step, and the right-looking methods update the active submatrix (the bottom right submatrix of entries of the rows and columns for which a pivot has not yet been chosen) with a newly chosen column of $\mathbf{L}$ and row of $\mathbf{U}$ at each step. For symmetric systems, simplifications to the general Gaussian elimination schemes exploiting the symmetric nature of the coefficient matrix as well as the factors generally result in worthwhile benefits in terms of both the amount of work required to compute the factors and the amount of memory necessary to store them. Further benefits are also available for symmetric positive definite systems (those systems satisfying $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$ for all $\mathbf{x}$ of suitable dimension). Orthogonal methods, the most common being $\mathbf{QR}$ factorisation, with orthogonal $\mathbf{Q}$ and upper triangular $\mathbf{R}$, are alternative solution methods, but are generally less efficient in terms of runtime and required storage [86]. Another factorisation scheme is that of the SPIKE algorithm, computing $\mathbf{A} = \mathbf{DS}$ for diagonal $\mathbf{D}$ and a block-tridiagonal matrix $\mathbf{S}$ with identity matrix blocks along its diagonal [87]. The method is suitable for narrow banded linear systems, and is designed with parallelisation in mind. Because of the need for a narrow bandwidth, the scheme will not be considered in this Thesis. A brief overview describing and comparing the different approaches for symmetric matrices is given. Unsymmetric methods are not considered given that (1.28)-(1.39) are symmetric, although it should be noted that the package SDPT3 does use $\mathbf{LU}$ factorisation when sufficient accuracy is not obtained using the symmetric solution schemes.

2.1.1 Gaussian elimination

Gaussian elimination reduces the coefficient matrix to upper triangular form by making the entries below the diagonal equal to zeros, column-by-column. For symmetric systems, the required storage may be halved by computing only one of the two factors (as each is the transpose of the other). In order to preserve the symmetry when

reordering, the permutations must be symmetric and so entries on the diagonal remain on the diagonal after reordering. Because of the possibility of having zero entries on the diagonal in symmetric indefinite systems, the Bunch-Kaufman factorisation preserves symmetry by using 2×2 block pivots (where the block pivot satisfies some numerical threshold), leading to the factorisation $\mathbf{PLDL}^T\mathbf{P}^T$, where $\mathbf{P}$ is a permutation matrix [88]. The algorithm requires (when not exploiting sparsity) $n^3/3$ floating point operations (FLOPs), $O(n^2)$ comparisons and $(n^2+n)/2$ storage for a full-storage (symmetric) matrix [34].

Fortunately, significant savings may be gained by exploiting sparsity and operating only on the non-zero entries in the original coefficient system and the resulting factors. This makes the computer code significantly more complex, but results in huge improvements in runtime, especially for cases in which there are many fewer non-zeros than n^2. Although a precise statement on the number of FLOPs and amount of storage required is not possible for sparse methods in general, the number of non-zeros in the coefficient matrix provide a rough indication of the work and storage requirements [84]. Unfortunately, a basic implementation of a sparse direct scheme will often not achieve a significant fraction of the peak machine speed and nor that of its dense factorisation counterparts. This issue is addressed by identifying portions of the scheme that allow dense submatrices to be exploited with dense matrix kernels that are implemented on most systems with standard interfaces in the collections known as BLAS and LAPACK. Because of their importance in many programs, vendor-supplied libraries such as Intel's MKL (Math Kernel Library) have often had significant effort spent on them to ensure that they operate very efficiently. In addition to the vendor-supplied libraries, open source libraries such as ATLAS [89] and OpenBLAS [90] use both low-level tuning and autotuning to achieve high performance on most platforms. Autotuning enables a high percentage of peak speed to be achieved on most systems by performing numerous tests upon installation to determine which approaches achieves the highest performance in each of a wide range of cases, and based on the characteristics of the data supplied at runtime, the most efficient approach is chosen to perform the computations. The two most common direct methods for sparse schemes that utilise these dense matrix kernels are the supernodal and multifrontal methods.

2.1.1.1 Direct method overview

In order to exploit dense matrix kernels, it is necessary to know in advance the non-zero structure of the factor. Define the graph, $\mathcal{G}_{L+L^T}$, which represents the sparsity pattern of the matrix $\mathbf{L}+\mathbf{L}^T$, where $\mathbf{A}=\mathbf{L}\mathbf{L}^T \in \mathbb{R}^{n\times n}$, with vertices $1 \le i \le n$ and an edge between vertices i and j if and only if $l_{ij} \neq 0$ (ignoring numerical cancellation). The graph is then known to have edge (i,j) if a path from i to j exists in the graph of $\mathbf{A}$, $\mathcal{G}_A$, through vertices less than $\min(i,j)$ [91]. The simplest case including fill-in can be seen for the matrix with sparsity pattern ($\times$ denoting a non-zero and diagonals are numbered for ease of identification)

$$\begin{bmatrix} 1 & \times & \times \\ \times & 2 & \\ \times & & 3 \end{bmatrix} \tag{2.1}$$

with the graph of Figure 2.

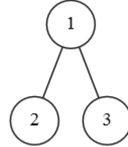


Figure 2. The undirected graph, $\mathcal{G}_A$, of the matrix with sparsity pattern (2.1).

Here, there is a path from node 2 to 3 (and vice versa) via $1<\min(2,3)$, and thus l_{32} will be non-zero. The graph $\mathcal{G}_{L+L^T}$ is shown in Figure 3.

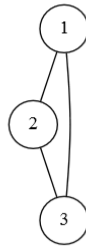


Figure 3. The undirected graph, $\mathcal{G}_{L+L^T}$, of the matrix with sparsity pattern (2.1).

The symbolic analysis used in most solvers, however, uses the elimination tree, $\mathcal{T}$ [92]. The elimination tree for $\mathbf{L}$ is easily described whereby each node j's parent is the first non-zero in column j of $\mathbf{L}$, and also results from pruning the directed graph $\mathcal{G}_L$ of $\mathbf{L}$ (that is, removing some subset of edges from $\mathcal{G}_L$) such that the reach from any node is

not affected. A directed graph of $\mathbf{L}$ has an edge from i to j if and only if $l_{ij} \neq 0$ (and paths go with the direction of the edges). Given the block form

$$\mathbf{L}\mathbf{L}^T = \begin{bmatrix} \mathbf{L}_{11} & \\ \mathbf{l}_{21} & l_{22} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{11}^T & \mathbf{l}_{21}^T \\ & l_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{a}_{21}^T \\ \mathbf{a}_{21} & a_{22} \end{bmatrix} = \mathbf{A},$$

where $\mathbf{L}_{11}$ is assumed known and the bottom row of $\mathbf{L}$, $\mathbf{l}_{21}$ and l_{22}, is to be computed (note that $\mathbf{l}_{21}$ and $\mathbf{a}_{21}$ are row vectors). This leads to $\mathbf{L}_{11}\mathbf{l}_{21}^T = \mathbf{a}_{21}^T$ and $l_{22} = \sqrt{a_{22} - \mathbf{l}_{21}\mathbf{l}_{21}^T}$. The sparsity pattern of the bottom row of $\mathbf{L}$ is thus defined by the triangular solve with the last column of $\mathbf{A}$. This is known to be defined by the reach of the set of vertex in the last column of $\mathbf{A}$ on the directed graph of $\mathbf{L}_{11}$, $\mathcal{G}_{L_{11}}$. The reach of a given vertex is defined as all vertices reachable via a path in $\mathcal{G}_{L_{11}}$ [93]. If, for some $i < j < k$, there are non-zero l_{ji} and l_{ki}, then l_{kj} will be non-zero and there is a path from k to i via j and thus removing the edge (k,i) from $\mathcal{G}_L$ does not affect the reachable nodes from k, nor from any nodes that have a path to k [94].

To compute the elimination tree, one builds it progressively starting with the first row subtree, $\mathcal{T}_1$, and proceeds row-by-row, computing each row subtree, $\mathcal{T}_i$ [85]. Two values are kept for each vertex, the *parent* and the *ancestor*. The ancestor array is simply a work vector that eliminates the need to walk through tree parent-by-parent, and instead skip to the greatest known ancestor of the vertex. Initially, all parents and ancestors are set to null. The first row subtree is always the trivial subtree containing just itself. For each subsequent row i of $\mathbf{A}$, each non-zero entry a_{ij} will also be a non-zero (structurally) in $\mathbf{L}$, so the ancestor of j is set to i (as no values greater than i could currently be an ancestor of j), and if j has no parent then it is set to i. Because of fill-in, it is then necessary to go to j's previously greatest ancestor (before it became i) to update it's ancestor to i and it's parent to i if it is null. This step is continued until a null greatest ancestor is encountered, at which point the next entry in row i is considered.

For example, given the matrix with sparsity pattern

$$\begin{bmatrix} 1 & & \times & \times & \\ & 2 & & & \times \\ \times & & 3 & & \times \\ \times & & & 4 & \\ & \times & \times & & 5 \end{bmatrix},$$

the row subtrees are shown in Figure 4.

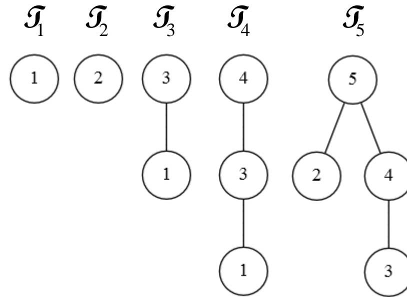


Figure 4. Example row subtrees.

Combining the row subtrees gives the full elimination tree in Figure 5.

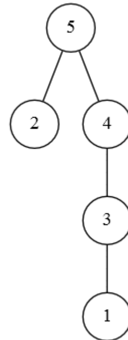


Figure 5. Example elimination tree.

The sparsity pattern of row i of $\mathbf{L}$ is then given by the reach from each non-zero of row i of $\mathbf{A}$ in $\mathcal{T}_i$. The elimination tree is usually post-ordered, which enables the number of non-zeros in each column to be computed efficiently, and usually results in an ordering which often performs slightly better in spite of the fact that there is no change to the amount of fill-in [85]. The post-ordered elimination tree is computed by a depth-first search on the elimination tree, giving the post-ordered elimination tree in Figure 6. As can be seen, this just has 1 and 2 swapped. The row indices of each column are then usually computed by working through an up-looking Cholesky factorisation without doing any numerical computation.

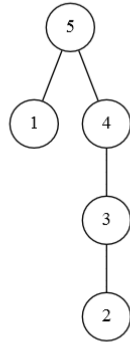


Figure 6. Example post-ordered elimination tree.

For the left-looking supernodal approaches, the elimination tree is also used to determine "fundamental" supernodes, defined as groups of adjacent columns that have identical sparsity patterns [95]. If a node only has a single child and that child has an identical sparsity pattern, then they can be combined into a supernode. If the column counts are known, then it is not necessary to check the sparsity patterns but simply compare the column counts (which will obviously only differ by 1 if they have an identical sparsity pattern). Knowing the sparsity pattern of the factor enables one to also consider how similar adjacent column sparsity patterns are and combine them into a supernode if there is little difference between them, thus trading some additional storage for a slightly larger supernode [95]. In the example shown, 3, 4, and 5 form a supernode. The numerical supernodal factorisation routine then uses four level 3 BLAS routines; the symmetric update (DSYRK), the Cholesky factorisation (the LAPACK routine DPOTRF), a sparse matrix-matrix product but using the dense kernel (DGEMM), and a triangular solve (DTRSM).

The frontal method, upon which the multifrontal method is based, exploits the non-zero pattern, or sparsity of the coefficient matrix and is a right-looking method [96]. The frontal method was developed for use with finite element codes that eliminate a variable once all its interactions are assembled into the coefficient matrix [84]. The method can, however, be used for problems with general matrices. Essentially, the algorithm operates by adding rows to a frontal matrix, starting with the first. The frontal matrix is a dense square submatrix in which the indices are those of the rows that have been added but not yet eliminated. Once a column has become fully summed (a column is fully summed when no more entries will be assembled into the column), a pivot from the fully summed column may be chosen and the associated row and column removed

from the frontal matrix. Dense matrix operations may be used within the frontal matrix. The frontal method can also be extended to include multiple fronts that may run in parallel and is known as the multifrontal method [97], [98]. In the multifrontal method, the processing jobs are divided among processors based on the elimination tree [98]. The elimination tree allows identification of suitable substructures of the coefficient matrix that have pivots within them that have no influence on other substructures. This allows the variables corresponding to those pivots to be eliminated in the assembled substructure. There can be numerous substructures which are dealt with in a similar fashion before assembling two of the substructures together and eliminating any pivots which have no influence outside the newly joined substructure. This process continues until the newly joined substructure is the whole of the coefficient matrix (now in factored form).

Supernodal and multifrontal methods form the basis for most of the more sophisticated and efficient direct solution software packages currently available. Multifrontal method implementations for symmetric indefinite systems include the Harwell Subroutine Library's (HSL) `MA57` [79], and the MPI-based parallel `MUMPS` [99]. `PARDISO` [100] implements the multifrontal method, as well as left- and right-looking supernodal approaches, although there is little difference between the methods in performance [101]. `CHOLMOD` [102] includes a left-looking supernodal Cholesky solver that is used in the popular software package `MATLAB` [103], and includes an implementation utilising GPUs for the dense matrix kernels (although this is not available through `MATLAB`). Left-looking supernodal techniques are recommended by some authors for solving the normal equations in interior point method implementations for LPs and SQLPs [40], [57], [104].

Gould and Scott [105] undertook a numerical comparison between the popular direct solution packages including `MA57`, `MUMPS`, `CHOLMOD` and `PARDISO` among others. The tests were performed on a single processor, meaning no parallelisation benefits were included. In their indefinite test cases, `PARDISO` was found to be superior in terms of factorisation runtime, with `MA57` in second, although there was little between the two when considering the analyse, factorise, and solve phases because of the increased likelihood of requiring iterative refinement with `PARDISO`. Memory usage mirrors these

results, with PARDISO using the least memory and MA57 the second-least. They note that the Oblio [106] user documentation reports that for large three-dimensional problems, left-looking and right-looking factorisations can outperform the multifrontal factorisation [101]. For symmetric positive definite systems, they report that CHOLMOD provides the best balance between the analysis phase, the factorisation, and the solve phase, and recommend PARDISO and MA57 for situations in which multiple solves are required [105].

2.1.2 Orthogonal factorisation

The orthogonal $\mathbf{QR}$ decomposition is the reduction of a matrix to the product of an orthonormal matrix, $\mathbf{Q}$, and an upper triangular matrix, $\mathbf{R}$. The computation of $\mathbf{Q}$ and $\mathbf{R}$ is commonly achieved through modified Gram-Schmidt orthogonalisation or Householder reflections, although Givens rotations can also be used [34].

The classical Gram-Schmidt orthogonalisation, which fills the i th column of $\mathbf{R}$ and $\mathbf{Q}$ at each iteration, is numerically unstable in finite precision arithmetic. By modifying the calculations so that the algorithm fills the i th row of $\mathbf{R}$ instead, it is equivalent to the classical algorithm producing the same factors in exact arithmetic, but results in smaller errors with finite precision arithmetic. This is known as the modified Gram-Schmidt procedure and is the most common orthogonalisation method in use [34]. Unfortunately, in some cases the rounding errors from the modified Gram-Schmidt scheme can still affect the orthogonality of the resulting system. By checking whether the norm of orthogonalised vector is significantly smaller than the pre-orthogonalised vector norm, the effect of cancellation can be identified, and rectified by performing reorthogonalisation [107].

Note that Householder reflections are more numerically stable but cannot be stopped before completion (truncated), and so are less attractive for the construction of orthogonal bases in iterative solution methods [86]. The ability to truncate the Gram-Schmidt orthogonalisation suggests an iterative procedure, which, if taken to completion, is known as the Full Orthogonalisation Method (FOM) [86].

Multifrontal implementations of $\mathbf{QR}$ factorisations have been developed, see, e.g. [108], [109], although Gaussian elimination or a variant of it is commonly used over

QR factorisation. This is because Gaussian elimination schemes generally require approximately half the number of floating point operations compared with **QR** factorisations [86].

2.1.3 Reordering

The differences in the number of non-zeros (and, consequently, the amount of work to compute them) between the factorisation of a sparse coefficient matrix in permuted form and unpermuted form can be very significant. Unfortunately though, the optimal ordering is NP-complete [110]. This has led to a large body of work on heuristic methods for computing orderings that minimise the number of non-zeros in the factored matrix.

One of the earliest methods is that of Markowitz [111], who proposed choosing an entry a_{ij} as a pivot in a general matrix if it minimises the Markowitz count, $(r_i - 1)(c_j - 1)$, where r_i is the number of entries in row i, and c_j the number of non-zeros in column j, and it satisfied some numerical criterion to ensure stability of the factorisation. This method was used to improve the efficiency of factorising the basis matrix in the Simplex method for linear programming, and is computed as the factorisation progresses. Markowitz's scheme simplifies when considering only symmetric permutations for symmetric positive definite matrices in which the diagonal entries are known to be stable pivots [72]; symmetry means that r_i and c_i are the same, so minimising the Markowitz count is equivalent to finding the minimum row count in the active submatrix (known as the degree), and positive definiteness avoids the need for numerical stability thresholds. This simplification is known as the minimum degree ordering [112], and the related but very efficient approximate minimum degree (AMD) method which is in wide use today [113], [114]. Markowitz also noted the possibility of minimising the number of fill-in entries at each step but did not pursue it, favouring the simpler approach described above. This scheme is now referred to as minimum fill (or minimum local fill) scheme, and approaches that seek to find a pivot leading to approximate minimum local fill can be found in use in ordering schemes for interior point methods [115].

The nested dissection orderings are based on a different approach and can be very effective for some large-scale matrices. Some nested dissection orderings are based on graph partitionings, with one of the most popular packages for computing nested dissection orderings being `METIS` [116]. The nested dissection approach usually follows a nested bisection recursively, where each bisection seeks a permutation of the matrix into

$$\begin{bmatrix} \mathbf{A}_{11} & & \mathbf{A}_{13} \\ & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{bmatrix}.$$

Here, the separator is the set of indices contained in $\mathbf{A}_{33}$ and the two sets separated are the indices contained in $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$, respectively. Each of these two index sets is then treated in the same way, recursively, down to some specified depth. The partition generally seeks two sets of the same size with a minimum separator set. When the sets become small enough, a different ordering scheme is often used such as a minimum degree or minimum local fill method (`METIS` uses AMD).

Another method that was proposed early on and still finds some use today in reordering for incomplete factorisations (described below) is that of the Reverse Cuthill-McKee (the Cuthill-McKee ordering [117] was found to lead to generally better performance when it was reversed [118]). This method simply takes the reverse of the permutation obtained by ordering the nodes in the order that they are visited in a breadth-first search. While other ordering approaches have been proposed in the literature, those methods described above constitute the most applicable approaches for the needs considered in this Thesis.

2.2 Inexact search directions in IPMs for conic optimisation

Instead of computing (practically) exact search directions using direct methods, it may be advantageous to utilise iterative methods and reduce the accuracy at which the search direction is determined. This will significantly improve storage requirements, and may also improve runtime performance. The use of iterative methods would thus allow problems which are prohibitively large to be solved.

Computing the search direction in IPMs using iterative solution methods has been studied by numerous authors. Indeed, it was even suggested by Karmarkar [45] in his landmark paper that iterative methods could be used to determine the search direction. In the literature, there are interior point algorithms using iterative solution methods to compute the search direction with proven polynomial convergence for both linear programming and semidefinite programming. An overview of these schemes, as relevant, is outlined below, but first, a summary of iterative method termination for general systems of linear equations is presented, which touches on some important issues when using iterative methods, such as finite termination. But first, the performance of various linear algebra operations that comprise the major operations in both direct and iterative methods is discussed.

2.2.1 The relative performance of basic linear algebra operations

When comparing the use of direct and iterative solvers for a practical implementation, it is crucial to consider the relative performance of the basic linear algebra operations (referred to as basic linear algebra subprograms, or BLAS). The BLAS operations are grouped into three "levels" based on the complexity of the operation. So multiplying a vector of size n by a scalar requires $O(n)$ multiplications and is thus a level one operation. Multiplying a dense $n \times n$ matrix by a vector of size n requires $O(n^2)$ floating point multiplications and additions and so is level two BLAS. Finally, multiplying two $n \times n$ dense matrices together requires $O(n^3)$ floating point operations and is a level three BLAS operation. Because of the way modern computers are designed and built, the peak achievable speed for each level of these operations varies considerably. The higher level operations generally have a much greater arithmetic operation to memory transfer ratio. Consider that, for scaling a vector, each element of the vector is loaded once to be used in a single multiplication before being stored again. Including the load of the scaling value, there are a total of $2n+1$ memory transfers (either loads or stores) and just n multiplications, giving a floating point operation (flop) to memory transfer ratio of $\frac{n}{2n+1}$. A dense matrix-vector multiplication requires at least loading each entry in the matrix and vector once, and storing each element of the resulting vector once, with n^2 multiplications and $n^2 - n$ additions, has a theoretical

upper bound on the ratio of $\frac{2n^2 - n}{n^2 + 2n} = \frac{2n - 1}{n + 2}$. Obviously, the ratio for $n > 1$ is larger for the matrix-vector product than the vector scale. Similarly, the upper bound on the ratio for a dense $n \times n$ matrix-matrix multiplication is n times the number of operations in matrix-vector product, and the vector load and store is replaced by a matrix load and store, giving $\frac{2n^3 - n^2}{3n^2} = \frac{2n - 1}{3}$. This is greater than both the level one and the level operations for $n > 1$, and generally provides the processor with enough computational work to hide the latency of the memory transfer functions, resulting in an overall higher number of arithmetic operations per second. Importantly, the triangular factorisation of a matrix and solving a system of equations with a triangular matrix are both level 3 BLAS operations, and turn out to be governed by matrix-matrix multiplication speed. When suitably arranged, the factorisation of a dense matrix can achieve near the same speed as that of matrix-matrix multiplication.

Unfortunately, an intelligent exploitation of sparsity generally requires some extra level of indexing to avoid working with any zero elements explicitly, which leads to additional memory operations. This is why the high-performance direct solvers attempt to arrange the entries of the matrix in such a way that certain portions of the matrix may be treated as dense. In this way, factorisation of sparse matrices may still achieve speeds near the dense matrix-matrix multiplication speed [102]. This is in contrast to the iterative solvers, which, in unpreconditioned form, are comprised of sparse matrix-dense vector multiplications, dot products, vector additions, and vector scaling – all level one and two BLAS. Generally speaking, solution of a dense linear system can achieve around 70% of a machine's peak speed, while sparse matrix-vector multiply will often only achieve around 10% of peak speed [119]. This means that an iterative solver will need to perform less than $7\times$ fewer arithmetic operations than that involved in the factorisation and substitution phases of a direct solver in order to solve a system faster than the direct method (assuming that the sparse matrix-vector multiply dominates the work performed by the iterative solver). This difference is amplified considerably if more than one linear system is to be solved, as is the case for the Mehrotra-style predictor corrector methods. Additionally, it is the relatively high

number of transfers in the lower level BLAS operations that form the bottleneck in highly parallel implementations of iterative methods [107].

Because of this fundamental difference between the different levels of operation, just as in the high-performance direct solvers using dense matrix kernels to achieve high speeds, it is imperative that any opportunities to utilise higher-level operations be exploited. For example, if many iterations are expected when using an iterative solver to obtain the search direction within a simplified HSD approach or similar, both (1.35) and (1.36) could be solved simultaneously instead of sequentially (note that there are three independent right-hand sides in the case of the three-term HSD method). While performing exactly the same number of arithmetic operations (assuming the same number of iterations is required for each right-hand side), solving the systems simultaneously will load the entries of the coefficient matrix from memory half the number of times compared to solving them one after the other.

2.2.2 Iterative method termination

The nature of iterative methods for large linear systems requires some form of method termination, which will often be long before full working precision is achieved. Barrett *et al*. [120] define a good stopping criterion as one which can:

1. Identify when the error, $\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}^*$, is satisfactorily small.
2. Identify stagnation or near-stagnation.
3. Limit the maximum number of iterations spent by the method.

The first requirement above is the most difficult to identify, as the error is not readily available without the solution *a priori*. However, tests can be performed which bound the error using relations between norms of $\mathbf{A}$, $\mathbf{x}^{(k)}$, $\mathbf{b}$ and $\mathbf{r}^{(k)}$. Additionally, a stop tolerance, s, is required which should be less than one but greater than the machine precision, which for double precision IEEE Standard Floating Point Arithmetic is $2^{-53} \approx 10^{-16}$.

Arioli *et al*. [121] use backward error analysis to develop a family of termination tests. Setting

$$\omega = \max \frac{\left|\mathbf{r}^{(k)}\right|_i}{\left(\mathbf{E} \cdot \left|\mathbf{x}^{(k)}\right| + \mathbf{f}\right)_i}$$

for some matrix $\mathbf{E}$ and vector $\mathbf{f}$, the iterations may be terminated when $\omega \le s$, indicating the solution to a nearby system has been found. Specifically,

$$(\mathbf{A} + \delta\mathbf{A})\mathbf{x}^{(k)} = \mathbf{b} + \delta\mathbf{b},$$

for $|\delta\mathbf{A}| \le \omega\mathbf{E}$ (component-wise) and $|\delta\mathbf{b}| \le \omega\mathbf{f}$. Skeel [122] use this scheme to guarantee the numerical stability of Gaussian elimination through iterative refinement with $\mathbf{E} = |\mathbf{A}|$ and $\mathbf{f} = |\mathbf{b}|$. However, Arioli *et al.* [121] point out that with its reliance on the non-zero entries of $\mathbf{A}$, this may not be suitable for iterative solvers as the convergence of iterative schemes relies more heavily on its eigensystem. They suggest two alternative choices for $\mathbf{E}$ and $\mathbf{f}$. Taking $\mathbf{E} = \mathbf{0}$ and $\mathbf{f} = \|\mathbf{b}\|_\infty \mathbf{e}$, with $\mathbf{e}$ the column vector of all 1's,

$$\omega = \frac{\|\mathbf{r}^{(k)}\|_\infty}{\|\mathbf{b}\|_\infty},$$

which may be generalised to other mutually consistent norm pairs. This test ensures the residual has been reduced by a factor of the stop tolerance. It is important to note that the use of $\|\mathbf{b}\|$ is not equivalent to the use of $\mathbf{r}^{(0)}$ for $\mathbf{x}^{(0)} = \mathbf{0}$. With an initial estimate $\mathbf{x}^{(0)} \ne \mathbf{0}$, $\|\mathbf{r}^{(0)}\|$ may be very large leading to premature termination. However, using $\|\mathbf{b}\|$ may lead to the opposite problem in that it may be very difficult to satisfy the test for ill-conditioned $\mathbf{A}$ with $\mathbf{x}$ close to the null-space of $\mathbf{A}$, as $\|\mathbf{A}\|_\infty \|\mathbf{x}\|_1 \gg \|\mathbf{b}\|_\infty$ [120]. This means for good approximations $\mathbf{x}^{(k)}$, $\mathbf{r}^{(k)}$ may still be quite large.

By setting $\mathbf{E} = \|\mathbf{A}\|_\infty \mathbf{e}\mathbf{e}^T$ (with $\mathbf{f} = \|\mathbf{b}\|_\infty \mathbf{e}$ still) and again generalising to all mutually consistent norm pairs,

$$\omega = \frac{\|\mathbf{r}^{(k)}\|}{\|\mathbf{A}\| \cdot \|\mathbf{x}^{(k)}\| + \|\mathbf{b}\|}.$$

Barrett *et al.* [120] state that even an order of magnitude estimate of $\|\mathbf{A}\|$ is sufficient for the above. Note that this form is generally less strict than the previous test, but both cases result in a final forward error bound of [120]

$$\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \le \left\|\mathbf{A}^{-1}\right\| \cdot \left\|\mathbf{r}^{(k)}\right\|.$$

Alternatively, Greenbaum [123] suggests estimating the spectral condition number for symmetric $\mathbf{A}$, $\kappa(\mathbf{A})$, using the Ritz values (the eigenvalues of $\mathbf{T}$), where the 2-norm leads to $\kappa(\mathbf{A}) = \lambda_{\max} / \lambda_{\min}$. This allows terminating the iteration process when

$$\kappa(\mathbf{T}^{(k)}) \cdot \frac{\left\|\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}\right\|}{\left\|\mathbf{b}\right\|} \le s,$$

where the right hand side is an upper bound to the relative error norm $\left\|\mathbf{e}^{(k)}\right\| / \left\|\mathbf{x}^{(k)}\right\|$.

2.2.3 Iterative method termination within IPMs

Polynomial complexity proofs have been obtained when solving the search directions inexactly in interior point methods for linear programming [124]–[128]. Mizuno and Jarre's [124] approach was quite theoretical and may not be suitable for implementation directly. Korzak's [126] method requires iterates to remain feasible once infeasibility is removed, requiring significant effort in determining the search directions and making it rather impractical if this situation ever arises. Al-Jeiroudi [129] uses a progressive tolerance to terminate the PCG method within the Higher Order Primal Dual Method (HOPDM) interior point solver package. Initially, the tolerance is set to 10^{-2}. When the relative duality gap is less than 10^{-3}, the tolerance is reduced to 10^{-3}, and when the relative duality gap falls below 10^{-4}, the tolerance is reduced again to 10^{-4}. This tolerance is used to compare the relative residual norm. The polynomial complexity bounds on these path-following IPMs generally require the Newton direction to be solved for some forcing term times the duality gap. Monteiro and O'Neal [127] prove convergence using a general class of iterative solver on the normal equations with a tolerance proportional to $\sqrt{\mu}/\sqrt{n}$, which is likely to be a difficult target when solving large-scale problems.

Wang and O'Leary [130] use the progressive tolerance scheme

$$s = \begin{cases} 5.0 \times 10^{-3} \text{ for early iterations} \\ \min(g \times 10^{3}, 10^{-3}) \text{ for middle iterations} \\ \min(g \times 10^{4}, 10^{-4}) \text{ for end iterations} \end{cases},$$

where g is the relative duality gap from the previous IPM iteration. Interestingly, their approach switches to a direct method when the iterative method is having trouble converging. They do not prove convergence using this method, but do provide successful numerical results. Similarly, Bergamaschi *et al.* [131] use $s=10^{-2}$ and $s=10^{-4}$ with a maximum number of iterations between 50 and 100 in determining the search direction for quadratic programs from the augmented equations.

There have also been polynomial convergence results published for solving SDPs with IPMs incorporating inexact search directions. In the first of such works, Kojima *et al.* [132] look to generalise the convergence proof for SDP across a class of search directions, noting that all search directions are equivalent when the iterate lies on the central path, and relax the centrality condition to a centrality inequality. Zhou and Toh [133] provide an iteration complexity for their inexact infeasible IPM the same as the best known exact complexity. They require the solution to the normal equation form for the search direction to be solved so as $\left\|\mathbf{A}^T\left(\mathbf{A}\mathbf{A}^T\right)^{-1}\mathbf{r}^{(k)}\right\| \leq \gamma_1\rho\theta_k\sigma_k$, where the θ_k terms are used to drive the iterates towards feasibility and are less than or equal to the corresponding complementarity gap, the σ_k values are defined as a sequence of scalars with a finite sum, $\gamma_1 \in (0,1)$ and $\rho \geq \frac{1}{n}\left(\mathrm{Tr}\left(\mathbf{X}^*\right)+\mathrm{Tr}\left(\mathbf{S}^*\right)\right)$ are constants, $\mathbf{A}$ defines the equality constraints, and $\mathbf{r}$ is the residual vector. Similarly, solving the augmented equations requires that the same inequality as the normal equation is satisfied, as well as the residual for the second block equation being less than the right-hand-side in the normal equation inequality. Solving some large-scale dense SDPs with an iterative solver to compute the search direction, Toh [134] simply requires that the residual vector in solving a reduced form of the augmented equations be less than 0.05 times the norm of the residual of the right-hand-side of the full block 3×3 system of equations defining the search direction.

There have been no studies solving SOCPs using IPMs with inexact search directions. However, due to the similarity between LP, SOCP, and SDP, and given the general use of convergence tolerances for the residual norm being related to the duality gap, it

appears that IPMs solving SOCPs with inexact search direction should aim to compute the search direction within some factor of the duality gap also.

It should be noted that when solving the Schur complement form of the search direction, the primal infeasibility is affected by the accuracy of the solution [76]. This can lead to an increase in the primal infeasibility (or stagnation) as the IPM approaches a solution, even when direct solvers are used, because of the severe ill-conditioning present in the Schur complement system [76]. Cai and Toh show that if $\boldsymbol{\xi}$ is the residual vector after solving the Schur system for $\mathbf{d}_y$, then the primal infeasibility after taking the step with step length α is $\mathbf{r}_p^+ = (1-\alpha)\mathbf{r}_p + \alpha\boldsymbol{\xi}$ for $\mathbf{r}_p = \mathbf{b} - \mathbf{A}\mathbf{x}$, where the superscript $+$ indicates the value after the step is taken, and the problem is not embedded in a HSD form. While the impact on the primal infeasibility for a HSD embedded problem will not be the same, the effect is similar. It is thus reasonable to require the tolerance in the search direction to be smaller than the primal infeasibility. Similarly, the dual infeasibility should not increase at any iteration because of poor accuracy in the search direction. This suggests seeking a search direction with a residual norm at least as small as the minimum of the primal and dual infeasibilities.

2.3 Iterative solution schemes

Iterative solution schemes can be split into two basic categories, stationary and non-stationary. Both iterative method classes seek out $\mathbf{x}^{(k)}$ with a progressively better approximation to the true answer. Without *a priori* knowledge of the true answer, or a sufficiently close approximation to it, the residual is often used to determine whether the approximate solution is accurate enough. For each k less than the maximum number of iterations allowed, $\left\|\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}\right\| < \left\|\mathbf{b} - \mathbf{A}\mathbf{x}^{(k-1)}\right\|$ if the method is to converge.

Stationary methods are characterised by construction of $\mathbf{x}^{(k)}$ of the general form $\mathbf{x}^{(k)} = \mathbf{B}\mathbf{x}^{(k-1)} + \mathbf{c}$, with neither $\mathbf{B}$ nor $\mathbf{c}$ depending upon the iterate k [120]. Common stationary schemes include the Jacobi, Gauss-Seidel, Successive Over-Relaxation (SOR), and Uzawa methods. The Jacobi, Gauss-Seidel and SOR methods are relatively simple but often require significantly more iterations (and hence computational effort) than their non-stationary counterparts. In general, they are also less robust than Krylov

subspace iteration methods. The Uzawa method was developed to solve saddle point systems similar in form to the augmented systems and can be quite efficient.

The non-stationary Krylov methods (we do not consider non-stationary methods that do not seek a solution in the Krylov subspace) search out an approximation $\mathbf{x}^{(k)}$ in the Krylov subspace, $\mathcal{K}$, defined by $\mathcal{K}^{(k)} = \text{span}\{\mathbf{r}^{(0)}, \mathbf{A}\mathbf{r}^{(0)}, \mathbf{A}^2\mathbf{r}^{(0)}, ..., \mathbf{A}^{k-1}\mathbf{r}^{(0)}\}$ (not to be confused with the cone constraints of the optimisation problem). In general, the initial solution estimate, $\mathbf{x}^{(0)}$, will be set to zeros, giving the initial residual, $\mathbf{r}^{(0)}$, equal to the right-hand side vector, $\mathbf{b}$. The Krylov subspace methods can be categorised into four general approaches; the Ritz-Galerkin approach, the minimum norm residual approach, the Petrov-Galerkin approach and the minimum norm error approach [107]. All four, however, are based on the construction of an orthogonal basis, which is completed using the Arnoldi algorithm or a simplification of it.

In 1951, Arnoldi proposed an iterative method to solve the eigenproblem $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$. The method happens to reduce a given matrix, $\mathbf{A}$, to Hessenberg form while computing an orthogonal basis for it. That is, the algorithm factors $\mathbf{A}$ as $\mathbf{Q}^T\mathbf{A}\mathbf{Q} = \mathbf{H}$, or $\mathbf{A}\mathbf{Q} = \mathbf{Q}\mathbf{H}$, with $\mathbf{Q}$ orthogonal and $\mathbf{H}$ upper Hessenberg. The Arnoldi method is often implemented with modified Gram-Schmidt orthogonalisation or the reorthogonalised version [135]. However, Householder orthogonalisation may be a reasonable alternative when developing software where robustness is critical [136]. Note that when $\mathbf{A}$ is symmetric, $\mathbf{H}$ reduces to a tridiagonal matrix, requiring new basis vectors to be orthogonalised with the two preceding vectors only. In the following, the partially constructed matrix (leading to $\mathbf{Q}$) after k steps of the orthogonalisation procedure shall be denoted $\mathbf{V}_{(k)}$, thus giving $\mathbf{V}_{(k)}^T\mathbf{A}\mathbf{V}_{(k)} = \mathbf{H}_{(k,k)}$, a $k \times k$ upper Hessenberg matrix.

The Lanczos algorithm [137] is a simplification of the Arnoldi algorithm for symmetric matrices. When $\mathbf{A}$ is symmetric, the Hessenberg $\mathbf{H}$ is also symmetric and thus tridiagonal. This leads to three term recurrences involving the sub-diagonal, diagonal, and super-diagonal terms of $\mathbf{H}$. This negates the need to store the basis vectors as the method proceeds. Exploitation of this short three-term recurrence is the basis for the Lanczos method and the iterative solution schemes Conjugate Gradients (CG) and Minimum Residual (MINRES).

2.3.1 Stationary methods

One of the simpler stationary schemes is the Jacobi method. Given a square linear system, the Jacobi iteration proceeds by refining the current solution estimate through

$$x_i^{(k+1)} = \frac{b_i - \sum_{j \neq k} a_{ij} x_j^{(k-1)}}{a_{ii}}.$$

This method requires that the coefficient matrix be diagonally dominant for the process to converge, where a matrix is diagonally dominant if $|a_{ii}| \geq \sum_{i \neq j} |a_{ij}|$ for all rows i [120]. Note that each iteration creates a solution estimate, $\overline{\mathbf{x}}$, which overwrites $\mathbf{x}$ at the end of each iteration. An improvement on the Jacobi scheme simply updates the solution estimate in place, thus reducing storage by an n-vector. This is known as the Gauss-Seidel method, and leads to improved convergence behaviour in many cases over the Jacobi method [120]. Similar to the Jacobi method, it requires strict diagonal dominance or symmetric positive definite matrices for convergence.

The SOR method attempts to improve on the Gauss-Seidel method through the use of a weighting coefficient to accelerate convergence. The solution estimate is updated as $x_i^{(k)} = \omega z_i^{(k)} + (1-\omega) x_i^{(k-1)}$, with $\mathbf{z}^{(k)}$ being the kth Gauss-Seidel iterate and ω the relaxation factor. However, the estimation of an optimal value for ω is difficult, requiring *a priori* knowledge of the spectral radius [34].

The first iterative methods to be developed and applied specifically to saddle point problems are the Arrow-Hurwicz and Uzawa methods [138]. These methods are still being actively developed today [78], and can even be found in large scale FELA implementations (see, for e.g., [31], [32]). Essentially, the Uzawa algorithm attempts to solve systems of the form

$$\begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & -\mathbf{C} \end{bmatrix} \begin{Bmatrix} \mathbf{x} \\ \mathbf{y} \end{Bmatrix} = \begin{Bmatrix} \mathbf{p} \\ \mathbf{q} \end{Bmatrix}$$

through the updates [139]:

$$\begin{aligned} &\text{Solve } \mathbf{A}_{11}\mathbf{x}^{(k+1)} = \mathbf{p} - \mathbf{B}^T\mathbf{y}^{(k)} \\ &\text{Update } \mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \omega(\mathbf{B}\mathbf{x}^{(k+1)} - \mathbf{C}\mathbf{y}^{(k)} - \mathbf{q}) \end{aligned},$$

Elman and Golub [139] show that the optimal choice for ω is

$$\omega = \frac{2}{\lambda_{\min} + \lambda_{\max}},$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are the smallest and largest eigenvalues of the Schur complement, $-\mathbf{C} - \mathbf{B}\mathbf{A}^{-1}\mathbf{B}^T$ (where it is possible that $\mathbf{C} = \mathbf{0}$), respectively. To improve the rate of convergence, the Uzawa method can also be preconditioned, giving the algorithm based on the two steps

$$\begin{aligned} &\text{Solve } \mathbf{A}\mathbf{x}^{(k+1)} = \mathbf{p} - \mathbf{B}^T\mathbf{q} \\ &\text{Update } \mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \omega\mathbf{Q}_S^{-1}\left(\mathbf{B}\mathbf{x}^{(k+1)} - \mathbf{C}\mathbf{y}^{(k)} - \mathbf{q}\right) \end{aligned},$$

where $\mathbf{Q}_S$ is an approximation of the Schur complement. Elman and Golub [139] also found that when using an iterative method to solve for $\mathbf{x}^{(k+1)}$, the method converges with a rate close to that of the exact Uzawa method. This kind of method falls under the category known as the inexact Uzawa method, and includes schemes where an approximation for $\mathbf{A}$ is used in place of the original block. Note that this scheme is often applied to a saddle-point system of the same form as the augmented system after regularisation, and so $\mathbf{A}$ is no longer trivial to invert or requires good approximate solves with the Schur complement. The inexact Uzawa scheme is defined by [140]

$$\begin{aligned} \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \mathbf{Q}_A^{-1}\left(\mathbf{p} - \mathbf{A}\mathbf{x}^{(k)} - \mathbf{B}^T\mathbf{y}^{(k)}\right) \\ \mathbf{y}^{(k+1)} &= \mathbf{y}^{(k)} + \omega\mathbf{Q}_S^{-1}(\mathbf{B}\mathbf{x}^{(k+1)} - \mathbf{C}\mathbf{y}^{(k)} - \mathbf{q}) \end{aligned}.$$

A common variant of the Uzawa method is the augmented Lagrangian Uzawa method. This approach instead solves the system

$$\begin{bmatrix} \mathbf{A} + \omega\mathbf{B}^T\mathbf{B} & \mathbf{B} \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{x} \\ \mathbf{y} \end{Bmatrix} = \begin{Bmatrix} \mathbf{p} + \omega\mathbf{B}^T\mathbf{q} \\ \mathbf{q} \end{Bmatrix}$$

using the Uzawa method. It similarly requires the scalar ω to be set, but now the parameter affects both the convergence and the difficulty of obtaining the solution in the first step. Increasing ω speeds up convergence of the Uzawa method while at the same time making the system in the first step more difficult to solve. It is widely known that the method converges for $0 \le \omega \le \frac{2}{\lambda_{\max}}$ and the optimal choice is $\omega = \frac{2}{\lambda_{\min} + \lambda_{\max}}$,

where $\lambda_{\min}$ and $\lambda_{\max}$ are the minimum and maximum eigenvalues of the Schur complement of the augmented Lagrangian system, $\mathbf{B}\left(\mathbf{A}^2+\omega\mathbf{B}^T\mathbf{B}\right)^{-1}\mathbf{B}^T$ [71].

The Arrow-Hurwicz method is often used in place of the Uzawa method for cases where solving with $\mathbf{A}$ is too expensive [71]. Slow convergence is often experienced using the method, however, and so some form of preconditioned variant is usually employed which is very similar to the inexact Uzawa method [71].

2.3.2 Ritz-Galerkin approach

The Ritz-Galerkin approach identifies $\mathbf{r}^{(k)}=\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)}$ orthogonal to $\mathcal{K}^{(k)}$. The Galerkin condition with the Krylov subspace basis vectors is equivalent to $\mathbf{V}_{(k)}^T(\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)})=\mathbf{0}$, where $\mathbf{V}_{(k)}$ holds the first k Krylov subspace basis vectors. Given the initial solution estimate $\mathbf{x}^{(0)}=\mathbf{0}$, giving $\mathbf{r}^{(0)}=\left\|\mathbf{r}^{(0)}\right\|_2\mathbf{v}_1$, leads to the simplification $\mathbf{V}_{(k)}^T\mathbf{b}=\left\|\mathbf{r}^{(0)}\right\|_2\mathbf{e}_1$, where $\mathbf{e}_1$ is the first canonical vector in $\mathbb{R}^k$. The Galerkin condition then becomes $\mathbf{V}_{(k)}^T\mathbf{A}\mathbf{V}_{(k)}\mathbf{y}=\left\|\mathbf{r}_0\right\|_2\mathbf{e}_1$, for $\mathbf{x}^{(k)}=\mathbf{V}^{(k)}\mathbf{y}$. The Ritz-Galerkin approach leads to the full orthogonalisation method (FOM) [141] (the Arnoldi method for $k=n$), and Conjugate Gradients method [80].

2.3.2.1 Conjugate Gradients

The CG method is one of the most popular iterative solution methods. It is generally considered the solver of choice for symmetric positive-definite systems (see, e.g.[120], [142]). However, it is based on the Lanczos method, and thus requires that $\mathbf{A}$ be symmetric. Furthermore, $\mathbf{A}$ must be positive definite to guarantee the existence of the implicit $\mathbf{LU}$ factorisation (or to satisfy the positive-definiteness of the $\mathbf{A}$-inner product), and is not applicable to the indefinite systems arising in the optimisation process described here. This prevents the CG method from being able to solve the augmented system, although Dollar *et al.* [143] show that the preconditioned CG method with certain preconditioners can be used to solve a projection of this system. This is discussed further in Block structured preconditioners below.

Wang and O'Leary [130] used the CG method to solve (1.39). However, the CG method was not used towards the end of the optimisation when the required accuracy is considerably higher. They use an adaptive scheme that changes from the preconditioned

CG solver to a direct method when the optimisation process closes in on the solution [130].

The CG method updates the solution vector at each iteration by

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha_k \mathbf{p}^{(k)},$$

for some scalar α and the search direction vector $\mathbf{p}^{(k)}$. The residuals may also be updated at each iteration with

$$\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} - \alpha_k \mathbf{q}^{(k)},$$

where $\mathbf{q}^{(k)} = \mathbf{A}\mathbf{p}^{(k)}$. α_k is chosen as

$$\alpha_k = \frac{\left\|\mathbf{r}^{(k-1)}\right\|_2^2}{\left\langle \mathbf{p}^{(k)}, \mathbf{A}\mathbf{p}^{(k)} \right\rangle}$$

at each iteration to minimise $\mathbf{r}_{(k)}^T \mathbf{A}^{-1} \mathbf{r}_{(k)}$. Importantly, when $\mathbf{A}$ is not positive definite, $\left\langle \mathbf{p}^{(k)}, \mathbf{A}\mathbf{p}^{(k)} \right\rangle$ no longer defines an inner product (as $\mathbf{p}^T \mathbf{A}\mathbf{p}$ is not guaranteed to be positive).

The search direction vector can be updated with

$$\mathbf{p}^{(k)} = \mathbf{r}^{(k)} + \beta_{k-1} \mathbf{p}^{(k-1)},$$

where, using

$$\beta_k = \frac{\left\langle \mathbf{r}^{(k)}, \mathbf{r}^{(k)} \right\rangle}{\left\langle \mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)} \right\rangle}$$

ensures that the residuals are orthogonal (and the search direction vectors are $\mathbf{A}$-orthogonal). The coefficients computed above correspond to the entries in an $\mathbf{LU}$ factorisation of $\mathbf{T}$, the tridiagonal Hessenberg. Note that the cost of applying the preconditioner, $\mathbf{M}$, is the cost of solving a linear system with it.

The convergence of the CG method is described by the well-known equation [34]

$$\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|_{\mathbf{A}} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \left\|\mathbf{x}^{(0)} - \mathbf{x}^*\right\|_{\mathbf{A}},$$

where κ is the spectral condition number, equal to $\lambda_{\max} / \lambda_{\min}$ for λ the eigenvalues of $\mathbf{A}$, and $\mathbf{x}^*$ is the exact solution.

Approaches that deal with multiple and sequential right hand sides have been developed. These methods exploit the identified eigenvectors corresponding to small eigenvalues, which are typically the reason for the slow convergence of the CG method [144].

2.3.3 Minimal norm residual approach

The minimal norm residual approach locates $\mathbf{x}^{(k)}$ at each iteration such that $\left\|\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)}\right\|_2$ is minimal. This approach is the basis of the popular methods MINRES for symmetric indefinite systems, and Generalised Minimal Residual (GMRES) for general systems.

Again, an orthogonal basis is constructed for the Krylov subspace,

$$\mathbf{A}\mathbf{V}^{(k)} = \mathbf{V}^{(k+1)}\mathbf{H}^{(k+1,k)}.$$

Noting that the solution estimates are calculated as $\mathbf{x}^{(k)} = \mathbf{V}^{(k)}\mathbf{y}$ for some appropriate $\mathbf{y}$, minimisation of the residual norm, $\left\|\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)}\right\|_2$, leads to

$$\left\|\mathbf{b}-\mathbf{A}\mathbf{V}^{(k)}\mathbf{y}\right\|_2 = \left\|\rho\mathbf{V}^{(k+1)}\mathbf{e}_1 - \mathbf{V}^{(k+1)}\mathbf{H}^{(k+1,k)}\mathbf{y}\right\|_2,$$

where $\mathbf{b} = \rho\mathbf{v}^{(1)}$ and $\rho = \left\|\mathbf{r}^{(0)}\right\|_2$. Noting that the common factor $\mathbf{V}^{(k+1)}$ is an orthonormal transformation, the residual norm can be simplified to

$$\left\|\rho\mathbf{e}_1 - \mathbf{H}^{(k+1)}\mathbf{y}\right\|_2.$$

This will be minimised by solving the least squares problem

$$\mathbf{H}^{(k+1,k)}\mathbf{y} = \rho\mathbf{e}_1 \qquad (2.2)$$

for $\mathbf{y}$. This is usually completed using a $\mathbf{QR}$ decomposition of $\mathbf{H}^{(k+1,k)}$.

2.3.3.1 Minimal Residual

The MINRES algorithm minimises $\left\|\mathbf{r}^{(k)}\right\|_2$ for $\mathbf{x}^{(k)}$ in $\mathbf{x}^{(0)} + \mathcal{K}^k$, and exploits the symmetry of $\mathbf{A}$ and the tridiagonal Hessenberg reduction with the Lanczos process in a similar fashion to CG [145].

The MINRES algorithm starts from a variant of the Lanczos relation [146]

$$\beta_{k+1}\mathbf{v}^{(k+1)} = \mathbf{A}\mathbf{v}^{(k+1)} - \alpha_k \mathbf{v}^{(k)} - \beta_k \mathbf{v}^{(k-1)},$$

with $\alpha_k = \mathbf{v}_{(k)}^T \mathbf{A}\mathbf{v}_{(k)}$ and β_{k+1} chosen such that $\left\| v^{(k+1)} \right\|_2 = 1$, which, after k iterations, leads to

$$\mathbf{A}\mathbf{V}^{(k)} = \mathbf{V}^{(k)}\mathbf{T}^{(k)} + \beta_{k+1}\mathbf{v}^{(k+1)}\mathbf{e}_k^T,$$

where

$$\mathbf{T}^{(k)} = \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & & \\ & & \ddots & \\ & & \beta_k & \alpha_k \end{bmatrix}.$$

Then $\mathbf{T}^{(k)}$ may be either factored into $\mathbf{LQ}$ form with Givens rotations or $\mathbf{QR}$ form. Because of the tridiagonality of $\mathbf{T}$, $\mathbf{L}$ and $\mathbf{R}$ have only three non-zero diagonals which provides the basis for exploitation of a symmetric $\mathbf{A}$. Taking the $\mathbf{QR}$ decomposition approach with $\mathbf{T} = \mathbf{F}^T\mathbf{R}$, where $\mathbf{F}$ is the product of the Givens rotations, and defining $\mathbf{W}_{(k)} \equiv \mathbf{V}_{(k)}\mathbf{R}_{(k)}^{-1}$, then $\mathbf{w}^{(0)}$ is a multiple of $\mathbf{v}^{(1)}$. The remaining columns of $\mathbf{W}^{(k)}$ can be computed through $\mathbf{W}^{(k)}\mathbf{R}^{(k)} = \mathbf{V}^{(k)}$, or

$$\mathbf{w}^{(k)} = \frac{(\mathbf{v}^{(k)} - \rho_2\mathbf{w}^{(k-1)} - \rho_3\mathbf{w}^{(k-2)})}{\rho_1}, \tag{2.3}$$

where $\rho_1 = r_{kk}$, $\rho_2 = r_{k-1,1}$, and $\rho_3 = r_{k-2,k}$ are the three entries in the kth column of $\mathbf{R}^{(k)}$.

With the $\mathbf{QR}$ decomposition of $\mathbf{H}$ completed, $\mathbf{y}$ can be obtained by solving (2.2) and updating $\mathbf{x}$. This can be further simplified utilising the entries calculated for $\mathbf{Q}$, leading to the update equation

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha_k \mathbf{w}^{(k)},$$

where α_k is the kth entry of $\beta\mathbf{F}\mathbf{e}_1$, but can be calculated without explicitly forming the product of Givens rotations.

MINRES minimises the 2-norm of the residual over the space

$$\mathbf{r}^{(0)} + \mathrm{span}\{\mathbf{A}\mathbf{r}^{(0)}, \mathbf{A}^2\mathbf{r}^{(0)}, ..., \mathbf{A}^k\mathbf{r}^{(0)}\}.$$

From this, the residual after k iterations is of the form

$$\mathbf{r}^{(k)} = P_k(\mathbf{A})\mathbf{r}^{(0)},$$

where P_k is the kth degree polynomial with value 1 at the origin, minimising the residual 2-norm. That is,

$$\left\|\mathbf{r}^{(k)}\right\|_2 = \min_{p_k}\left\|p_k(\mathbf{A})\mathbf{r}^{(0)}\right\|_2,$$

where p_k is the set of all polynomials of degree k or less for which $p_k(\mathbf{0}) = 1$. This leads to the residual norm bounds

$$\frac{\left\|\mathbf{r}^{(k)}\right\|_2}{\left\|\mathbf{r}^{(0)}\right\|_2} \le \min_{p_k} \max_{i=1,...,n}\left|p_k(\lambda_i)\right|,$$

where λ_i are the eigenvalues of $\mathbf{A}$ [123]. Given the simple polynomial $\left(1 - x/c\right)^k$, it is clear that matrices with a tight clustering of eigenvalues around a single value are likely to exhibit good convergence. Furthermore, indefinite matrices with eigenvalues on either side of the origin are unlikely to exhibit good convergence because of the difficulties in approximating zero at a number of points while maintaining the value 1 at the origin. Even in the simple case when all the eigenvalues are contained within the two specific intervals $[a,b] \cup [c,d]$, with the kth degree polynomial

$$p_k(x) = \frac{T_l(q(x))}{T_l(q(0))},\ q(x) = 1 + \frac{2(x-b)(x-c)}{ad-bc},$$

l equal to the integer part of $k/2$, and T_l the lth Chebyshev polynomial, the residual 2-norm bound becomes

$$\frac{\left\|\mathbf{r}^{(k)}\right\|_2}{\left\|\mathbf{r}^{(0)}\right\|_2} \le 2\left(\frac{\sqrt{|ad|} - \sqrt{|bc|}}{\sqrt{|ad|} + \sqrt{|bc|}}\right)^l.$$

With the intervals placed symmetrically about the origin, the bound is the same as that obtained for positive definite $\mathbf{A}$ with $\kappa = (d/c)^2$. However, the bounds are better for intervals not symmetric about the origin [123].

In their initial presentation, Paige and Saunders [145] warn that the condition number of the triangular factor of the Hessenberg approaches that of $\mathbf{A}$, which can lead to errors in $\mathbf{x}^{(k)}$ for ill-conditioned $\mathbf{A}$. Furthermore, Sleijpen *et al.* [147] show that MINRES suffers from the propagation of rounding errors proportional to the square of the condition number. MINRES also requires SPD preconditioners to ensure the preconditioned system remains symmetric, which significantly limits the available choices for preconditioning for indefinite $\mathbf{A}$.

2.3.3.2 Generalised Minimal Residual

The GMRES method was developed as a robust algorithm for solving linear systems in which the coefficient matrix is not positive real and symmetric [141]. Because of the nonsymmetric $\mathbf{A}$, the GMRES method must use the Arnoldi method, and uses Givens rotations to construct the $\mathbf{QR}$ factorisation and solve for $\mathbf{y}$ in a similar fashion to MINRES. The updated solution estimate is then calculated as $\mathbf{x}^{(k)} = \mathbf{V}^{(k)}\mathbf{y}$, which unlike MINRES, requires all previously computed Krylov subspace basis vectors, $\mathbf{v}^{(k)}$.

The increased computational burden and storage costs involved with the use of all basis vectors, $\mathbf{v}^{(k)}$, suggest using a restarted version of GMRES, denoted GMRES(m) [141]. In this scheme, the GMRES algorithm is restarted every m iterations. However, van der Vorst [107] points out that the choice for m is difficult, as the speed of convergence may exhibit significant difference for nearby values of m. Embree [148] provides an example where convergence occurs in three iterations for $m = 1$, while for $m = 2$, GMRES stagnates. Saad [136] also states that GMRES(m) can suffer from stagnation for semidefinite and indefinite coefficient matrices, while noting the prohibitive cost of attempting full convergence in n steps.

The original GMRES scheme used Givens rotations to decompose the upper Hessenberg, $\mathbf{H}$, into $\mathbf{QR}$ form, and thus the entries denoted h_{ij} are actually entries of the triangular matrix, $\mathbf{R}$.

There are a number of variants based on GMRES, including flexible GMRES [149], simpler GMRES [150], loose GMRES [151], GMRES with Householder transformations [152], and the hybrid GMRES*. Flexible GMRES allows a different preconditioner to be used at each iteration. Simpler GMRES avoids construction of the upper Hessenberg factorisation involved in the GMRES algorithm, but the cost of doing so negates the benefit [107]. Loose GMRES (LGMRES) arose from Baker *et al.*'s [151] observation that the restarted residuals in GMRES(m) were alternating direction in a cyclic fashion. They propose an algorithm to identify such a situation and prevent it from occurring, leading in some cases to significant savings in iterations, although the improvement was not as significant when using preconditioners. GMRES* utilises an inner iteration using another iterative solution scheme [153], and is known as GMRESR when GMRES is used for both inner and outer iterations. There are also various methods that give the same solution approximations at each iteration in exact arithmetic as GMRES; these methods include Vinsome's ORTHOMIN (cited in [107]), Orthodir [154] and Axelsson's method [155]. These methods require more work per iteration and are generally less robust than GMRES, although ORTHOMIN can be used effectively in a truncated fashion (in which case it is no longer equivalent to GMRES) [107].

In general, there is no neat convergence relation for GMRES as there is with CG. Specifically, Greenbaum *et al.* [156] have shown that any non-increasing convergence curve is possible for GMRES, and that eigenvalues and the condition number are not necessarily indicative of the expected convergence with GMRES. Saad [157] provides some methods for relatively good upper bounds on the residual norm for the early stages, but they are not as sharp once GMRES (or MINRES) begins superlinear convergence.

2.3.4 Petrov-Galerkin approach

In seeking short recurrences similar to MINRES and CG but for nonsymmetric $\mathbf{A}$, one can construct the biorthogonal bases, $\mathbf{V}$, for $\mathcal{K}^k(\mathbf{A};\mathbf{v}^{(1)})$, and $\mathbf{W}$, for $\mathcal{K}^k(\mathbf{A}^T;\mathbf{w}^{(1)})$, where the biorthogonality condition requires $\left\langle \mathbf{v}_i, \mathbf{w}_j \right\rangle = \delta_{ij}$. The biorthogonal sets of vectors lead to methods such as Quasi-Minimal Residual (QMR) [158](Freund and Nachtigal 1991) and Bi-Conjugate Gradients (BiCG).

The Bi-Lanczos method [159] starts from the Lanczos relations $\mathbf{A}\mathbf{V}^{(k)} = \mathbf{V}^{(k+1)}\mathbf{H}^{(k+1,k)}$. Seeking a three-term recursion, multiplying both sides by $\mathbf{V}_{(k)}^T$ will not result in a tridiagonal $\mathbf{H}^{(k)}$. Thus, given $\mathbf{W}_{(k)}^T\mathbf{V}_{(k)} = \mathbf{D}_{(k)}$ for some $\mathbf{W}_{(k)}$, where $\mathbf{D}_{(k)}$ is the matrix with entries $d_{ij} \neq 0$ for $i = j$ and zero otherwise. Multiplying the Lanczos relation with $\mathbf{W}^T$ from the left now gives

$$\mathbf{W}_{(k)}^T\mathbf{A}\mathbf{V}_{(k)} = \mathbf{D}_{(k)}\mathbf{H}_{(k)}.$$

For $\mathbf{H}^{(k)}$ to be tridiagonal and thus provide the three-term recurrence, $\mathbf{V}_{(k)}^T\mathbf{A}^T\mathbf{W}_{(k)}$ must also be tridiagonal. This form suggests generating the $\mathbf{w}_i$ with $\mathbf{A}^T$ in much the same way as the $\mathbf{v}_i$ are generated with $\mathbf{A}$ in the Lanczos process [107]. This leads to $\mathbf{W}_{(k)}$ being the set of vectors, $\mathbf{w}_i$, biorthogonal to $\mathbf{V}^{(k)}$, i.e. $\langle \mathbf{v}_i, \mathbf{w}_j \rangle = \delta_{ij}$.

In 1976, Fletcher (cited in [107]) set $\mathbf{W}_{(k)}^T(\mathbf{b} - \mathbf{A}\mathbf{x}_{(k)}) = \mathbf{0}$, leading to $\mathbf{T}^{(k,k)} = \|\mathbf{r}^{(0)}\|_2 \mathbf{e}_1$ and $\mathbf{x}^{(k)} = \mathbf{V}^{(k)}\mathbf{y}$ for the BiCG method. However, BiCG suffers from highly irregular convergence (van der Vorst 2003), which can affect the attainable accuracy [86], and as such is not considered further here. The QMR method is discussed below.

In the above, two breakdowns can occur; $\mathbf{v}_i$ or $\mathbf{w}_i$ can be set to $\mathbf{0}$, or $\mathbf{w}_{(k)}^T\mathbf{v}_{(k)} = 0$ for non-zero $\mathbf{v}_i$ and $\mathbf{w}_i$. The latter is known as a serious breakdown [86]. This serious breakdown can be avoided by making successive Krylov subspace basis vectors block-wise biorthogonal as shown by Freund and Nachtigal in their look-ahead variant of QMR [158]. Alternatively, the method can simply be restarted when a small diagonal element is identified. However, this method forgets the Krylov subspace basis that has been constructed and thus loses the potential for superlinear convergence [107].

In BiCG and the iterative methods based on BiCG, the residual vector is updated via a relation similar to

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \mathbf{A}\mathbf{w}^{(k)},$$

with the solution estimate being updated in a similar fashion as

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}\mathbf{w}^{(k)}.$$

The multiplication in the residual update can and does lead to differences between $\mathbf{r}^{(k)}$ and $\mathbf{b}-\mathbf{Ax}^{(k)}$, which presents a difficulty in testing for convergence with the updated residual vector. Because of the importance of $\mathbf{r}^{(k)}$ in defining $\mathbf{T}$ throughout the iterative procedure, replacing the residual vector with $\mathbf{b}-\mathbf{Ax}^{(k)}$ can increase the chances for stagnation by ignoring the previous rounding errors [107].

2.3.4.1 Quasi-Minimal Residual

The QMR method [158] is a variant of the BiCG method exhibiting smoother convergence and avoids one of the breakdown conditions in BiCG [107].

As with the minimal residual approach, the norm of the residual can be rearranged to

$$\left\|\mathbf{b}-\mathbf{Ax}^{(k)}\right\|_2 = \left\|\mathbf{b}-\mathbf{AV}^{(k)}\mathbf{y}\right\|_2 = \left\|\rho\mathbf{V}^{(k+1)}\mathbf{e}_1 - \mathbf{V}^{(k+1)}\mathbf{H}^{(k+1,k)}\mathbf{y}\right\|_2,$$

which, because $\mathbf{V}^{(k+1)}$ is no longer orthonormal, gives

$$\left\|\mathbf{r}^{(k)}\right\|_2 \le \left\|\mathbf{V}^{(k+1)}\right\|_2 \left\|\rho\mathbf{e}_1 - \mathbf{H}^{(k+1,k)}\mathbf{y}\right\|_2. \tag{2.4}$$

The second norm on the right-hand side of (2.4) is the norm of the quasi-residual. To update $\mathbf{x}^{(k)}$, a $\mathbf{y}$ is sought to minimise this quasi-residual, resulting in the QMR method.

Although the QMR method does not minimise the true residual, Nachtigal [160] has shown that the residual obtained with the QMR method, $\mathbf{r}_Q^{(k)}$, can be related to the GMRES residual, $\mathbf{r}_G^{(k)}$, by

$$\left\|\mathbf{r}_Q^{(k)}\right\|_2 \le \left\|\mathbf{V}^{(k+1)}\right\|_2 \left\|\rho\mathbf{e}_1 - \mathbf{H}^{(k+1,k)}\mathbf{y}\right\|_2,$$

which appears promising, although the condition number of $\mathbf{V}^{(k+1)}$ cannot be bounded *a priori* [123].

In addition to the original QMR method [158], there exists a variant of QMR based on coupled two-term recurrences, avoiding the generally less robust three-term recursions [161], a transpose-free QMR method (TFQMR) [162], and the Symmetric QMR algorithm [82]. The Symmetric QMR method avoids the need for SPD preconditioners, which is well-suited to the needs of solving symmetric indefinite systems. Various authors have found the Symmetric QMR method to be the solver of choice for

symmetric indefinite problems when the preconditioned CG method cannot be used, see, e.g. [163], [164].

2.3.5 Minimal norm error approach

The minimal norm error approach seeks to minimise the norm of the error, $\left\|\mathbf{e}^{(k)}\right\|_2 = \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|_2$. For $\mathbf{A} = \mathbf{A}^T$, this approach leads to the Symmetric LQ (SYMMLQ) method [145], while for the general case it leads to the Generalised Minimal Error (GMERR) method [165]. A brief overview of SYMMLQ is provided below.

2.3.5.1 SYMMLQ

Paige and Saunders [145] present the SYMMLQ method with MINRES as an alternative to CG for indefinite systems, using $\mathbf{T}^{(k)} = \overline{\mathbf{L}}^{(k)}\mathbf{Q}^{(k)}$ for lower triangular $\overline{\mathbf{L}}^{(k)}$ and orthonormal $\mathbf{Q}^{(k)}$:

$$\mathbf{T}_{(k)}\mathbf{Q}_{(k)}^T = \overline{\mathbf{L}}_{(k)} = \begin{bmatrix} \gamma_1 & & & & \\ \delta_2 & \gamma_2 & & & \\ \varepsilon_3 & \delta_3 & \gamma_3 & & \\ & & \ddots & & \\ & & \varepsilon_k & \delta_k & \overline{\gamma}_k \end{bmatrix}.$$

This provides short recurrences in a stable factorisation which exists for indefinite $\mathbf{A}$. To compute approximate solutions to a linear system, SYMMLQ forces the residual to be orthogonal to the Krylov subspace in a similar fashion to CG. Similar to MINRES, SYMMLQ requires positive-definite preconditioners but may converge much more slowly than MINRES for ill-conditioned systems [107].

2.3.6 Hybrid methods

Hybrid methods combine various components of the approaches discussed above. The Stabilised Bi-Conjugate Gradient (Bi-CGSTAB) [107] is one of the more common of the hybrid methods and is discussed below.

2.3.6.1 Stabilised Bi-Conjugate Gradients

Similar to QMR, the stabilised bi-conjugate gradients (Bi-CGSTAB) method introduced by van der Vorst [166] seeks short recurrences for general matrices while exploiting the desirable convergence properties of the Conjugate Gradient Squared method (CGS)

[167](Sonneveld 1989), but stabilising its erratic behaviour. The Bi-CGSTAB method is essentially the combined effect of Bi-CG and GMRES(1).

CGS is another hybrid method, and relies on the residual vector being a function of the polynomial with the Bi-CG residual vector written as $\mathbf{r}^{(k)} = P_k(\mathbf{A})\mathbf{r}^{(0)}$ and the shadow residual $\tilde{\mathbf{r}}^{(k)} = P_k(\mathbf{A}^T)\tilde{\mathbf{r}}^{(0)}$, where the shadow residual results from the $\mathbf{w}_i$ components of the biorthogonal set. Because of the biorthogonality between the $\mathbf{v}_i$ and the $\mathbf{w}_i$,

$$\left\langle \mathbf{r}^{(j)}, \tilde{\mathbf{r}}^{(i)} \right\rangle = \left\langle P_j(\mathbf{A})\mathbf{r}^{(0)}, P_i(\mathbf{A}^T)\tilde{\mathbf{r}}^{(0)} \right\rangle = \left\langle P_i(\mathbf{A})P_j(\mathbf{A})\mathbf{r}^{(0)}, \tilde{\mathbf{r}}^{(0)} \right\rangle = 0$$

for all $i < j$ [107]. The shadow residuals can thus be constructed as

$$\tilde{\mathbf{r}}^{(k)} = P_j^2(\mathbf{A})\mathbf{r}^{(0)},$$

which avoids the computation with $\mathbf{A}^T$ [167]. Because of the squared term in calculating the residual, CGS can converge more rapidly than Bi-CG. The method performs most effectively when $\mathbf{A}$ contains a uniform distribution of eigenvalues over some interval not containing the origin, but can, in practical cases, exhibit significantly more erratic behaviour than Bi-CG [107]. Fokkema *et al*. [168] note that the good approximation in the direction of eigenvectors associated with the extreme eigenvalues when using CGS is well-suited for solving the linear systems arising from Newton's scheme for nonlinear equations.

However, because of the serious effects of irregular convergence (residual norm varying in magnitude in subsequent iterations), a more smoothly converging variant is desirable. The Bi-CGSTAB method smooths the convergence by modifying $P_j(\mathbf{A})P_i(\mathbf{A})\mathbf{r}^{(0)}$ to

$$\mathbf{r}^{(k)} = Q_k(\mathbf{A})P_k(\mathbf{A})\mathbf{r}^{(0)},$$

where Q_k is a polynomial of the form $Q_i(\mathbf{z}) = (1-\omega_1\mathbf{z})(1-\omega_2\mathbf{z})...(1-\omega_i\mathbf{z})$ and ω_i are suitable constants. In Bi-CGSTAB, the ω_k are chosen to minimise the residual $\mathbf{r}^{(k)}$ [166]. This leads to

$$\mathbf{r}^{(k)} = (\mathbf{I} - \omega_k\mathbf{A})(\mathbf{r}^{(k-1)} - \alpha_{k-1}\mathbf{A}\mathbf{p}^{(k-1)})$$

and

$$\begin{aligned}\mathbf{p}^{(k)} &= T_{k-1}(\mathbf{A})\mathbf{r}^{(0)} \\ &= \mathbf{r}^{(k)} + \beta_k(\mathbf{I} - \omega_k\mathbf{A})\mathbf{p}^{(k-1)}\end{aligned},$$

for scalar α_k and β_k, the diagonal and super-diagonal entries of $\mathbf{T}$, respectively.

Bi-CGSTAB is a finite method, terminating in at most n steps in exact arithmetic [166]. However, the method can suffer the same breakdown conditions as CGS and BiCG where some ρ_k or $\tilde{\mathbf{r}}^T\mathbf{v}^{(k)}$ is zero or essentially so. Implementations of the method should check for these occurrences and either restart with a different $\tilde{\mathbf{r}}$ or restart with a different solution method [107].

Variations on BiCGSTAB have been developed. Sleijpen *et al.* [169] present various methods based on BiCGSTAB(l), where l performs the same function as m in GMRES(m). Zhang [170] presented the generalised product Bi-CG (GPBi-CG) method. However, GPBi-CG is based on a three-term recursion which is considered less stable numerically then two-term recursions [107].

2.4 Preconditioners for iterative linear solvers

Preconditioners have become the focus of much recent computational science research, and constructing methods for transforming a problem that appears intractable into another whose solution can be approximated is likely to remain a prominent research topic for the foreseeable future [86]. With regards to solving linear systems by the use of iterative methods, Chen [171] states that devoting effort to the construction of improved preconditioners is likely to yield better results compared with searching for a more effective solution method. Although preconditioners have been effective for many problems, the preconditioning of ill-conditioned matrices and symmetric indefinite matrices is still largely an open problem (see, e.g. [71], [78], [107]).

Preconditioners generally attempt to improve the condition number of the coefficient matrix, while clustering the eigenvalues around one (or at least away from zero). Preconditioning can be of the form $\mathbf{MAx} = \mathbf{Mb}$ (left preconditioning) or $\mathbf{AMM}^{-1}\mathbf{x} = \mathbf{b}$ (right preconditioning), or mixed preconditioning, $\mathbf{M}_L\mathbf{A}\mathbf{M}_R\mathbf{M}_R^{-1}\mathbf{x} = \mathbf{M}_L\mathbf{b}$. The preconditioner $\mathbf{M}$ will either seek to replicate $\mathbf{A}$ or its inverse, $\mathbf{A}^{-1}$. Note that if $\mathbf{M} \approx \mathbf{A}$, then the inverse $\mathbf{M}^{-1}$ must be used in place of $\mathbf{M}$ in the preceding forms. In

the case of $\mathbf{M} \approx \mathbf{A}$, $\mathbf{M}$ must be easily invertible (or in a form allowing easy solution as in the case of factorisation, e.g. $\mathbf{M} = \mathbf{LU} \approx \mathbf{A}$ leads to solving $\mathbf{Ax} = \mathbf{b}$ as $\mathbf{Lc} = \mathbf{b}$, $\mathbf{Ux} = \mathbf{c}$). For $\mathbf{M} \approx \mathbf{A}^{-1}$, application of the preconditioner yields $\mathbf{MA} \approx \mathbf{I}$ (for right preconditioning). The common preconditioners can be categorised into the following:

- Matrix splitting and incomplete factorisation preconditioners.
- Approximate inverses.
- Block structured preconditioners.
- Domain decomposition and multilevel methods.

There are also more exotic approaches such as support graph or Vaidya-type preconditioners [172], [173], and wavelet-based methods [171]. Both of these methods have achieved success in preconditioning discretised PDE problems.

In solving problems with indefinite coefficient matrices, the standard algebraic preconditioners of the incomplete factorisations and approximate inverses are often found to be less effective than those methods taking into account the block structure of the symmetric indefinite and general systems defining the Newton search direction. Part of the reason for this is the highly irregular behaviour as the preconditioner becomes progressively closer to the “exact” preconditioner. This may be at least partially explained by the fact that as the negative eigenvalues of the preconditioned system become progressively closer to 1, some of them end up closer to zero than with a more sparse approximation resulting in an increase in iterations to convergence [107]. The standard algebraic preconditioners do provide ideas for approximating blocks in the block structured and constraint preconditioners. Domain decomposition and multilevel methods are used primarily for the solution of partial differential equations. Domain decomposition splits the domain into subdomains, with each being solved independently and the solutions being combined in periodic global solves. Multilevel methods seek to use coarse grid approximations to smooth specific components of the error and interpolating the solution estimate back to the finer grid. Neither of these methods are directly applicable to the problem at hand, although the algebraic multilevel method has allowed some of the ideas to be utilised when no explicit grids are present, although the method was developed for M-matrices and its performance generally degrades as the

coefficient matrix becomes further from an M-matrix. An M-matrix is a symmetric positive definite matrix with positive diagonal entries and non-positive off-diagonal entries.. It should be noted that the Vaidya-type and support graph preconditioners generally outperform incomplete decompositions for Stieltjes and symmetric diagonally dominant matrices [173], where a Stieltjes matrix is a symmetric positive-definite matrix with non-positive off-diagonal entries. Because of their limited applicability, they are not considered further.

Interestingly, Bocanegra *et al*. [174] suggest using different preconditioners for different stages of the optimisation process. By using different strategies over the course of the optimisation process, this allows the utilisation of cheap and efficient preconditioners in the early stages where the search direction may not need to be as accurately determined as the later stages, for which more computationally-heavy preconditioners may be used to achieve greater solution accuracy.

2.4.1 Matrix splitting and incomplete factorisation preconditioners

Matrix splitting preconditioners are based on taking some representational part of $\mathbf{A}$ such as a diagonal band and exploiting it. Such preconditioners can be very cheap to construct, require little to no storage, and can be easy to use. Common methods include the Jacobi, triangular, banded and banded arrow preconditioners. Similarly, incomplete factorisations attempt to approximate the main characteristics of $\mathbf{A}$, but in a more efficient form for solving $\mathbf{Ax} = \mathbf{b}$. While more complex and expensive than the simpler splittings, the incomplete $\mathbf{LU}$ (ILU) factorisation is one of the most popular of the standard algebraic preconditioners, and has proved to be a fairly robust preconditioning method. A similar preconditioning method is the incomplete $\mathbf{QR}$ factorisation.

2.4.1.1 Jacobi preconditioner

The point Jacobi preconditioner consists of only the diagonal entries of $\mathbf{A}$. It is cheap to construct, store and implement, assuming the diagonal elements of $\mathbf{A}$ are readily accessible. Unfortunately, more sophisticated preconditioners are likely to yield a more significant improvement in terms of iterations [120], although for large systems on parallel hardware, its simplicity and lack of required communication may give this preconditioner an advantage in terms of solve time. The block Jacobi preconditioner generalises the point Jacobi idea to include diagonal blocks of $\mathbf{A}$.

2.4.1.2 Incomplete decompositions

Incomplete $\mathbf{LU}$ decompositions [175] were initially used with symmetric non-singular matrices with non-positive off-diagonal entries known as M-matrices where their existence has been proven [107], but have become a fairly robust form of preconditioning for large classes of problems [78]. The family of methods operate much the same as a complete factorisation and hence $\mathbf{M} \approx \mathbf{A}$, but will not store all non-zero entries of the factors. All incomplete decompositions follow some dropping scheme that dictates which entries are kept and which are discarded. If the set of entries in the incomplete factor is denoted $\boldsymbol{S}$, then the incomplete decomposition proceeds as follows.

$$
\begin{array}{l}
\text{for } j = 1, m \\
\quad \text{for } k = 1, j-1 \\
\quad\quad a_{j+1:m,j} = a_{j+1:m,j} - a_{k,j} a_{j+1:m,k} \\
\quad \text{end for} \\
\quad a_{ii} = \sqrt{a_{ii}} \\
\quad \text{for } i = j+1, m \\
\quad\quad \text{if } \left(a_{ij} \in \boldsymbol{S} \right) \\
\quad\quad\quad a_{ij} = \dfrac{a_{ij}}{a_{ii}} \\
\quad\quad\quad a_{ii} = a_{ii} - a_{ij}^2 \\
\quad\quad \text{end if} \\
\quad \text{end for} \\
\text{end for}
\end{array}
$$

Early approaches used include dropping any entry that doesn't occur in the sparsity pattern of $\mathbf{A}$ (known as $\text{ILU}(0)$), or a generalisation of this approach that discards non-zeros in the factors based on the level-of-fill (known as $\text{ILU}(l)$, where l is the level-of-fill parameter). These approaches reportedly work well for some problems [171], but may not exist when working with matrices that are not diagonally dominant, M-matrices, H-matrices, or Stieltjes matrices. An H-matrix is defined as having a comparison matrix that is an M-matrix, where the comparison matrix is formed by taking the absolute value of the diagonal entries and the negation of the absolute value of the off-diagonal entries. The modified ILU (MILU) adds the discarded entries to the diagonal value of $\mathbf{U}$, which, for some problems, meant that some property (e.g. energy)

was conserved [171]. A variant of MILU for grid-based problems adds a constant times the square of the mesh size to the entries that is a significant improvement for certain problems [107].

To improve the robustness of the approach, the factors are computed while dynamically choosing a sparsity pattern based on row or column count restrictions and fill thresholding, where values below some threshold, τ , are discarded. These two rules are combined in the popular incomplete $\mathbf{LU}$ threshold with level-of-fill control ($\mathrm{ILUT}(\tau, p)$) preconditioner, with inputs p , the maximum number of entries per row in the factors, and τ , the value threshold for fill-in [176]. In order to guarantee the existence of the decomposition, it is still required that the coefficient matrix be an M-matrix, Stieltjes matrix, H-matrix, or a generalised diagonally dominant matrix [171], although Chow and Saad [177] note that with the use of pivoting, reordering, diagonal perturbations and scaling, ILU preconditioning can still be applied relatively successfully to indefinite problems. A more recent development in ILU preconditioning provides control over the growth of the inverse of the factors [178].

An alternative to ILU which works for general nonsingular matrices is the incomplete $\mathbf{QR}$ and incomplete $\mathbf{LQ}$ factorisations (ILQ). Bai *et al.* [179] present an incomplete $\mathbf{QR}$ factorisation using incomplete Givens orthogonalisation (IGO), the construction of which is more easily parallelised than ILU-type preconditioners. Similar to the incomplete $\mathbf{LU}$ decompositions, they present variations, generalised with the GTIGO(τ , p) which includes threshold and level-of-fill controls. Saad [180] presents the ILQ method based on modified Gram-Schmidt orthogonalisation, showing that the application of CG to the normal equations (CGNE) preconditioned with ILQ is a more robust method for an indefinite problem than other common methods (the CGNE approach sets $\mathbf{x} = \mathbf{A}^T\mathbf{y}$ and solves $\mathbf{A}\mathbf{A}^T\mathbf{y} = \mathbf{b}$).

2.4.2 Approximate inverses

Approximate inverse preconditioners seek to replicate the inverse of $\mathbf{A}$, giving $\mathbf{MA} \approx \mathbf{I}$. The two common methods for calculating approximate inverse preconditioners are the Sparse Approximate Inverse (SPAI) method, and the Approximate Inverse (AINV) or Factorised Sparse Approximate Inverse (FSAI) methods.

2.4.2.1 SPAI

SPAI preconditioners are constructed by minimising $\|\mathbf{AM}-\mathbf{I}\|$ [171]. Using the Frobenius norm naturally decouples the problem into n least squares problems to solve for the column vector $\mathbf{m}_j$

$$\min_{m_j} \|\mathbf{Am}_j - \mathbf{e}_j\|_F^2, \tag{2.5}$$

with $\mathbf{e}_j$ the jth unit vector. Solving for $j = 1,2,....,n$ leads to $\mathbf{M} = [\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_n]$. Gould and Scott [181] tested two different approaches based on this method for nonsymmetric matrices, and found that they may not perform as efficiently as other popular standard preconditioning techniques when executing serially. SPAI preconditioners do provide alternatives for problems which fail using other schemes. Note that (2.5) represents one of n independent problems and, as such, is trivially parallelisable. Grote and Huckle [182] developed one of the more effective Frobenius norm minimisation methods with their SPAI preconditioner.

2.4.2.2 AINV and FSAI

The other popular approximate inverse methods are published under the name AINV for the general case [183] and FSAI for a symmetric $\mathbf{A}$ [184]. Both methods construct the preconditioner

$$\min_{\mathbf{W},\mathbf{Z}} \|\mathbf{W}^T\mathbf{AZ} - \mathbf{I}\|_F^2,$$

where $\mathbf{W}$ and $\mathbf{Z}$ are upper triangular matrices satisfying level-of-fill and threshold controls. Note that for the symmetric case $\mathbf{W} = \mathbf{Z}$. Similar to ILU, these preconditioners are also subject to breakdown, and sometimes the system is solved for the coefficient matrix $\mathbf{A} + \alpha\mathbf{A}$ for some scalar α instead [171]. Benzi and Tůma [183] found that AINV was about as effective as ILU(0) for their range of nonsymmetric test problems, while the implicit factorisations were generally more robust in large ill-conditioned systems with entries slowly decaying away from the diagonal [185]. The SPAI approach is inherently more parallelisable than the AINV and FSAI techniques, but is computationally more expensive [185].

2.4.3 Block structured preconditioners

Block structured preconditioners are block-diagonal or block-triangular matrices which exploit the structure of the KKT system. In this section on block structured preconditioning, the 2×2 block system considered is

$$\begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{x} \\ \mathbf{y} \end{Bmatrix} = \begin{Bmatrix} \mathbf{p} \\ \mathbf{q} \end{Bmatrix}.$$

These preconditioners generally require the approximation of $\mathbf{A}$ and the Schur complement, $\mathbf{S} = -\mathbf{B}\mathbf{A}^{-1}\mathbf{B}^T$ (with a zero $(2,2)$ block as in the KKT system). They are generally categorised as either block diagonal or block triangular, both of which are discussed below.

2.4.3.1 Block diagonal preconditioners

The basic block diagonal preconditioner for the augmented system is

$$\mathbf{M} = \begin{bmatrix} \hat{\mathbf{A}} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{S}} \end{bmatrix},$$

where $\hat{\mathbf{A}}$ and $\hat{\mathbf{S}}$ are suitable non-singular approximations of $\mathbf{A}$ and $\mathbf{S}$, respectively. The explicitly preconditioned matrix is then

$$\mathbf{M}^{-1}\mathbf{A} = \begin{bmatrix} \mathbf{I} & \hat{\mathbf{A}}^{-1}\mathbf{B}^T \\ \hat{\mathbf{S}}^{-1}\mathbf{B} & \mathbf{0} \end{bmatrix}.$$

Kuznetsov [186] and Murphy *et al.* [187] show that this preconditioned system has three eigenvalues (assuming $\mathbf{A}$ is non-singular); 1 and $\frac{1}{2}\left(1 \pm \sqrt{5}\right)$. This means that GMRES would take only three iterations to solve the KKT equations. Unfortunately, constructing the preconditioned system would be as expensive as directly computing its inverse [71]. Hence, an approximation to the Schur complement must be made. Note that in computing the search direction in the IPM, no approximation of the inverse of the (1,1) block needs to be made as it can be computed for a second-order cone of dimension n_i in $O(n_i)$ operations when using NT scaling [57], similarly, the scaling matrix for the linear cone is diagonal and so inversion is trivial.

Phoon *et al.* [163] generalise the preconditioner to

$$\mathbf{M} = \begin{bmatrix} \hat{\mathbf{A}} & \mathbf{0} \\ \mathbf{0} & \alpha\hat{\mathbf{S}} \end{bmatrix} \quad (2.6)$$

where α is some non-zero scalar (possibly negative), and $\hat{\mathbf{H}}$ and $\hat{\mathbf{S}}$ are approximations of the $(1,1)$ block and the Schur complement, respectively. They found that the choice of α can have a significant impact on the number of iterations when using a cheap approximation to the $(1,1)$ block, contrary to its theoretical behaviour. The optimal choice for α is -4, which reduces the number of eigenvalues (without approximating the $(1,1)$ block) to two; $\frac{1}{2}$ and 1. Toh [134] also uses this preconditioner (with $\alpha = -20$) on some large scale (but dense) SDPs using an IPM with the symmetric QMR iterative solver, achieving comparable results to some first-order methods thought to be superior for large scale SDP problems. Note that their implementation sets $\hat{\mathbf{A}} = \mathrm{diag}(\mathbf{A})$, and $\hat{s}_{ij}$, the entries of $\hat{\mathbf{S}}$, are calculated as

$$\hat{s}_{ij} = \sum_{j=1}^{n} \frac{b_{ji}^2}{a_{jj}}.$$

This generalised Jacobi preconditioner resulted in smaller eigenvalue clusters, where the preconditioned system had eigenvalues in the right half-plane only. Such a preconditioner appears attractive due to the ease of construction and solution with it.

2.4.3.2 Block triangular preconditioner

Upper and lower block triangular preconditioners of the form

$$\mathbf{M}_U = \begin{bmatrix} \hat{\mathbf{A}} & \mathbf{B}^T \\ \mathbf{0} & \hat{\mathbf{S}} \end{bmatrix} \text{ and } \mathbf{M}_L = \begin{bmatrix} \hat{\mathbf{A}} & \mathbf{0} \\ \mathbf{B} & \hat{\mathbf{S}} \end{bmatrix}$$

can be transformed to inverse factors to solve systems of the form $\mathbf{Mv} = \mathbf{w}$ as (using the upper triangular $\mathbf{M}_U$)

$$\mathbf{v} = \begin{bmatrix} \hat{\mathbf{A}}^{-1} & \mathbf{0} \\ \mathbf{0} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{B}^T \\ \mathbf{0} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & -\hat{\mathbf{S}}^{-1} \end{bmatrix} \mathbf{w},$$

which is only more expensive than the full block-diagonal preconditioner by a multiplication with $\mathbf{B}^T$ [78]. Setting $\hat{\mathbf{A}} = \mathrm{diag}(\mathbf{A})$ and $\hat{\mathbf{S}} = -\mathbf{B}\hat{\mathbf{A}}^{-1}\mathbf{B}^T$ is a common and effective preconditioner for a diagonally dominant $\hat{\mathbf{A}}$ [78].

2.4.3.3 Constraint preconditioners

Constraint or indefinite preconditioners exploit a block structure similar to the augmented system being solved. This provides more scope for domain-specific knowledge to be incorporated into the preconditioner, and has proven effective for solving saddle point systems [71]. In using preconditioners to solve indefinite block-structured systems, Bergamaschi [188] recommends dealing with the augmented system (rather than the normal equations) in order to provide more freedom in the construction of the preconditioner, although there are still occasions when the normal equations are solved with preconditioners derived from the augmented system. However, using the exact constraint preconditioner may still be computationally expensive, which leads to the idea of inexact constraint preconditioning. Consequently, a number of authors have focussed on inexact constraint preconditioners of the form

$$\mathbf{M} = \begin{bmatrix} \hat{\mathbf{A}} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0} \end{bmatrix},$$

where $\hat{\mathbf{A}}$ is again some approximation to $\mathbf{A}$, for mixed finite element schemes, and linear and nonlinear optimisation (see, e.g. [129], [188]–[192]). Rozložník and Simoncini [193] and Haws and Meyer [190] approximate the (1,1) block with the identity matrix, although in solving the KKT systems, approximating $\mathbf{A}$ with the identity matrix is often not sufficiently effective [131], [164], [194]–[196]. Instead, $\mathbf{A}$ may be approximated using AINV or an incomplete Cholesky factorisation. AINV provides the inverse factors of the approximation, $\mathbf{L}_A^{-T}\mathbf{L}_A^{-1} \approx \mathbf{A}^{-1}$, allowing relatively cheap solves, while the incomplete Cholesky provides an approximation to the factors $\mathbf{L}_A\mathbf{L}_A^T \approx \mathbf{A}$. Using AINV, the Schur complement is then approximated by taking the incomplete Cholesky decomposition $\mathbf{S} = \mathbf{B}\mathbf{Z}\mathbf{Z}^T\mathbf{B}^T \approx \mathbf{L}_S\mathbf{L}_S^T$ (the matrix $\mathbf{Z}$ here arises in the AINV computation described above). The inverse of the preconditioner can then be formed as

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{L}_A^{-T} & -\mathbf{L}_A^{-T}\mathbf{L}_A^{-1}\mathbf{B}^T\mathbf{L}_S^{-T} \\ \mathbf{0} & \mathbf{L}_S^{-T} \end{bmatrix} \begin{bmatrix} \mathbf{L}_A^{-1} & \mathbf{0} \\ \mathbf{L}_S^{-1}\mathbf{B}\mathbf{L}_A^{-T}\mathbf{L}_A^{-1} & -\mathbf{L}_S^{-1} \end{bmatrix},$$

an upper and lower triangular matrix. This method was found to be more robust than ILU and diagonal scaling (using the diagonal of the preconditioner of (2.6) by Phoon *et al.* [163]) [164], [196].

Bergamaschi *et al.* [131] extended the inexact constraint preconditioning scheme and mitigated the effects of memory-intensive inverses. They noted that anything more simple than a diagonal matrix to approximate $\mathbf{A}$ will be ineffective, and looked for ways to approximate the off-diagonal blocks $\mathbf{B}$ and its transpose, as $\hat{\mathbf{B}}$ and $\hat{\mathbf{B}}^T$, respectively, where $\hat{\mathbf{B}}$ has full row rank. $\hat{\mathbf{B}}$ is defined through the splitting $\hat{\mathbf{B}} = \mathbf{B} - \mathbf{E}$, where the entries of $\mathbf{E}$ are defined as

$$e_{ij} = \begin{cases} b_{ij} & \text{if } |b_{ij}| < \tau \|\mathbf{b}_j\| \text{ and } |i-j| < n_b \\ 0 & \text{otherwise} \end{cases},$$

where τ is a drop tolerance, n_b is a band size, and $\mathbf{b}_j$ is the jth column of $\mathbf{B}$.

Dollar *et al.* [197] organised a number of these efforts into a framework, and showed that using some of them, the projected form of the augmented system may be solved with the preconditioned CG (PCG) method. Essentially, by implementing a non-standard inner product in the PCG algorithm, one can solve systems with an indefinite coefficient matrix. For this to succeed, the preconditioner must be nonsingular, and an additional preconditioning matrix, $\mathbf{P}$, is SPD, such that $\mathbf{PM}^{-1}\mathbf{A}$ is symmetric and positive definite. This means that $\mathbf{M}^{-1}\mathbf{A}$ is positive definite in the inner product $\langle \bullet, \bullet \rangle_P$. Thus, the saddle point system can be solved using CG to solve the equivalent, but SPD, system

$$\mathbf{PM}^{-1}\mathbf{Ax} = \mathbf{PM}^{-1}\mathbf{b}.$$

The preconditioner $\mathbf{PM}^{-1}$ is defined as

$$\sigma\mathbf{I} + \begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{D} & \mathbf{F}^T \\ \mathbf{F} & \mathbf{E} \end{bmatrix},$$

where σ is a scalar, and matrices $\mathbf{D}$, $\mathbf{F}$, and $\mathbf{E}$ match the dimensions of the corresponding matrices $\mathbf{A}$, $\mathbf{B}$ and the $\mathbf{0}$ block, respectively. This preconditioner provides the framework for solving the system through a number of common forms (see Reference [197] and the references therein for details). Note that an efficient implementation would not require $\mathbf{PM}^{-1}\mathbf{A}$ and $\mathbf{PM}^{-1}\mathbf{b}$ to be explicitly formed. Instead, an alternative implementation of the CG method is suggested, which, for suitable choices $\mathbf{M}$ and $\mathbf{P}$, will outperform the traditional algorithm [197].

Rozložník and Simoncini [193] use the simpler preconditioner

$$\mathbf{M} = \begin{bmatrix} \mathbf{I} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix}.$$

By using the convenient form of the Cholesky-like factorisation of the preconditioner $\mathbf{M}^{-1} = \mathbf{L}^{-1}\mathbf{D}^{-1}\mathbf{L}^{-T}$, it is easy to solve the systems $\mathbf{Mv} = \mathbf{w}$ using only matrix-vector products. Note that the inverse has only blocks of $\mathbf{I}$, $\mathbf{B}$, $\mathbf{B}^T$, and $(\mathbf{BB}^T)^{-1}$. Preconditioning the augmented system with it allows the CG method to be utilised, although scaling $\mathbf{A}$ is necessary in many cases to avoid non-convergence.

2.4.3.4 Analytic inverse

The analytic inverse of the KKT matrix is

$$\begin{bmatrix} \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}^T\mathbf{S}^{-1}\mathbf{B}\mathbf{A}^{-1} & \mathbf{A}^{-1}\mathbf{B}^T\mathbf{S}^{-1} \\ \mathbf{S}^{-1}\mathbf{B}\mathbf{A}^{-1} & -\mathbf{S}^{-1} \end{bmatrix}. \tag{2.7}$$

Note that using this form to solve with a given right-hand side is equivalent to using the block $\mathbf{LDL}^T$ and the block $\mathbf{LU}$ factorisation. The steps in solving with the approximated block factorisation $\mathbf{LU} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{B}\hat{\mathbf{A}}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{A}} & \mathbf{B}^T \\ \mathbf{0} & -\hat{\mathbf{S}} \end{bmatrix}$ leads to an efficient way to solve a system using the analytic inverse:

$$\begin{aligned} &1.\ \text{Solve } \hat{\mathbf{A}}\mathbf{z} = \mathbf{p} \\ &2.\ \text{Solve } \hat{\mathbf{S}}\mathbf{y} = \mathbf{Bz} - \mathbf{q} \\ &3.\ \text{Solve } \hat{\mathbf{A}}\mathbf{x} = \left(\mathbf{p} - \mathbf{B}^T\mathbf{y}\right) \end{aligned}.$$

Note that this is nothing more than the procedure one goes through when solving the Schur complement equation instead of the augmented equations. Compared with the block-diagonal Schur preconditioner, this approach requires two more matrix-vector products and two vector updates.

2.4.3.5 Augmented preconditioner

Golub and Greif [198] focus on an approach to make the augmented equations easier to solve; the augmented Lagrangian approach. This system is the same as that used in the augmented Lagrangian Uzawa method, where the second equation is multiplied by $\mathbf{BW}$ and added to the first equation, where $\mathbf{W}$ is an $n \times n$ matrix (and n is the number of columns in $\mathbf{B}$). This gives

$$\begin{bmatrix} \hat{\mathbf{A}} + \mathbf{B}^T\mathbf{W}\mathbf{B} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{x} \\ \mathbf{y} \end{Bmatrix} = \begin{Bmatrix} \mathbf{p} + \mathbf{BWq} \\ \mathbf{q} \end{Bmatrix} \tag{2.8}$$

Depending on the choice of $\mathbf{W}$, (2.8) may be easier to solve, even when $\mathbf{A}$ is singular or severely ill-conditioned. Furthermore, $\mathbf{A} + \mathbf{B}^T\mathbf{W}\mathbf{B}$ may even be positive-definite or possess a small condition number (Golub and Greif 2003). Choosing $\mathbf{W} = \gamma\mathbf{I}$, where γ is a scalar constant, requires a choice for γ. Based on experimental evidence, Golub and Greif [198] found performance was dependent upon the choice of γ, and suggested $\gamma = \|\mathbf{A}\| / \|\mathbf{B}\|^2$ based on empirical evidence as it may result in a significant difference in the spectrum and condition number compared with the original $(1,1)$ block.

The augmented block preconditioner of Rees and Grief [199], originally intended for solving LPs and QPs with IPMs, is of the form

$$\begin{bmatrix} \mathbf{A} + \mathbf{B}^T\mathbf{W}^{-1}\mathbf{B} & k\mathbf{B}^T \\ \mathbf{0} & \mathbf{W} \end{bmatrix},$$

for positive-definite $\mathbf{W}$ and scalar k. This preconditioner has the benefit of becoming increasingly effective as the optimization method approaches an optimal point. In order to maintain a norm of the augmenting term that is comparable to the $(1,1)$ block, Rees and Greif [199] suggest that for quadratic programs the augmenting term is multiplied by the norm of the $(2,1)$ block squared over the norm of the $(1,1)$ block. This preconditioner unfortunately does not, in general, perform very well in the earlier steps

of the interior point method. Instead, a more effective preconditioner, such as a constraint preconditioner, could be used until its effectiveness becomes beneficial (i.e. the spectrum of the preconditioned matrix approaches two distinct values, for which an optimal solution method such as MINRES will converge in just two iterations).The extended preconditioner of Zeng and Li [200] is

$$\begin{bmatrix} \mathbf{A} + \eta \mathbf{B}^T \mathbf{W}^{-1} \mathbf{B} & (1-\eta\varepsilon)\mathbf{B}^T \\ \mathbf{0} & \varepsilon \mathbf{W} \end{bmatrix},$$

for scalars ε and η. They recommend setting $\mathbf{W} = \mathbf{I}$ and $\varepsilon = -\eta^{-1}$ to minimise the number of distinct eigenvalues.

The challenge with these preconditioner forms is solving systems with the $n \times n$ $(1,1)$ block, which is often significantly larger in dimension and has more non-zeros than the Schur complement.

2.4.3.6 Reduced augmented equations

This approach is specific to the augmented equations defining the search direction in the IPM for conic optimisation, (1.35), and reverts to using the coefficient matrix of the augmented equations with the generic unknown and right-hand side vectors, giving

$$\begin{bmatrix} -\mathbf{F}^2 & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{x} \\ \mathbf{y} \end{Bmatrix} = \begin{Bmatrix} \mathbf{p} \\ \mathbf{q} \end{Bmatrix},$$

except where noted otherwise.

In the IPM, as μ approaches zero, the eigenvalues of the (1,1) matrix in the augmented equations ($\mathbf{F}^2$, where $\mathbf{F}$ is the Nesterov-Todd scaling matrix for SOCP and SDP) split into three major groups. These groups are $O(\mu)$, $O(1)$ and $O(1/\mu)$. This means the spectral condition number is $O(1/\mu^2)$, and explains the significant increase in inner iterations required by iterative solvers to compute the search direction as the duality gap is reduced.

To combat the growth in the condition number of the NT scaling matrix, Cai and Toh [76] and Toh [134] form an eigendecomposition of the (1,1) block and address the ill-conditioning directly for SOCP and SDP. The approach is also applicable to LPs [201].

As the interior point method process approaches an optimal solution, the linear systems determining the search direction becoming increasingly ill-conditioned. This is of concern for direct solution methods as the equations get closer and closer to a singular system, which affects the solution accuracy and thus the optimality of the overall solution. For iterative solution methods, the ill-conditioning adversely impacts the convergence behaviour. Toh [134], Cai and Toh [76], and Chai and Toh [201] attempt to solve the first issue through a reduced augmented equation approach that takes into consideration the spectrum of the augmented equations in IPMs for SDP, SOCP, and LP, respectively, thus allowing the search direction to be calculated more accurately. This approach is similar to that of Freund *et al.*'s [125] for LP, in which the primal and dual unknowns are reordered at each iteration, and then some are eliminated leading to a reduced but more stable form of the augmented equations in order to ensure that none of the values $O(\mu)$ in the diagonal $(1,1)$ block are inverted.

If at the *k*th iteration of the IPM, the residual vector resulting from the solution of the Schur complement equation, (1.39), is $\boldsymbol{\xi}$ and the primal step is computed exactly, then the primal infeasibility after the step is taken for some $\alpha \in [0,1]$ is $\mathbf{r}_p^{(k+1)} = (1-\alpha)\mathbf{r}_p^{(k)} + \alpha\xi$. This shows that the accuracy attained in solving the Schur complement equation affects the primal feasibility. This leads directly to a deterioration of the primal infeasibility as the optimal solution is approached in some problems, even when direct solvers are used.

Using the eigenvalue decomposition of the NT scaling matrix, Cai and Toh [76] suggest forming what they call the reduced augmented equations (RAE). By partitioning the eigenvalues of $\mathbf{F}^2$ into a group of smaller eigenvalues, $\mathbf{D}_1$, and a group of larger eigenvalues, $\mathbf{D}_2$, where the partition is stored in the permutation matrix $\mathbf{P}$ such that $\mathrm{diag}(\mathbf{D}_1, \mathbf{D}_2) = \mathbf{P\Lambda P}^T$ and partitioning the columns of the eigenvector matrix as $\mathbf{VP}^T = [\mathbf{V}_1 \quad \mathbf{V}_2]$ to match, one arrives at the partitioned augmented equations

$$\begin{bmatrix} -\mathbf{D}_1 & \mathbf{0} & \mathbf{V}_1^T\mathbf{A} \\ \mathbf{0} & -\mathbf{D}_2 & \mathbf{V}_2^T\mathbf{A} \\ \mathbf{A}\mathbf{V}_1 & \mathbf{A}\mathbf{V}_2 & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{V}_1^T\mathbf{x} \\ \mathbf{V}_2^T\mathbf{x} \\ \mathbf{y} \end{Bmatrix} = \begin{Bmatrix} \mathbf{V}_1^T\mathbf{p} \\ \mathbf{V}_2^T\mathbf{p} \\ \mathbf{q} \end{Bmatrix},$$

where $\mathbf{p}$ and $\mathbf{q}$ are the appropriate right-hand side being solved for. It is the inversion of the small entries (eigenvalues) in $\mathbf{D}_1$ that lead to the ill-conditioning in the Schur complement, and so Cai and Toh [76] suggest using a positive definite diagonal matrix $\mathbf{E}_1$, setting $\mathbf{S}_1 = \mathbf{D}_1 + \mathbf{E}_1$, eliminating $\mathbf{V}_2^T \mathbf{d}_x$, then adding the first block row multiplied by $\mathbf{AV}_1\mathbf{S}_1^{-1/2}$ to the third block row, and scaling the first block row by $\mathbf{S}_1^{-1/2}$, giving the reduced augmented matrix

$$\begin{bmatrix} -\mathbf{D}_1\mathbf{E}_1^{-1} & \left(\mathbf{AV}_1\mathbf{S}_1^{-1/2}\right)^T \\ \mathbf{AV}_1\mathbf{S}_1^{-1/2} & \mathbf{AV}\mathrm{diag}\left(\mathbf{S}_1^{-1}, \mathbf{D}_2^{-1}\right)\mathbf{V}^T\mathbf{A}^T \end{bmatrix}.$$

They then show that, under a suitable partition of the eigenvalues, the reduced augmented matrix has a condition number bounded independently of the normalised complementarity gap, μ [76]. Similar to the augmented Lagrangian-style approaches, the challenge of using this better conditioned system lies in solving a Schur-complement-like system that has more non-zeros than the standard Schur complement, along with both of the diagonal blocks being non-zero.

2.4.4 Matrix permutation and ordering

The use of ordering algorithms, traditionally utilised in direct solution schemes, have achieved mixed results in the literature. Improvement in the convergence of ILU-preconditioned Krylov projection methods can be achieved through the permutation of the coefficient matrix, effectively renumbering the nodes [78]. Interestingly, the more sophisticated orderings such as minimum degree fill (MD) and nested dissection (ND) often have not performed as well as bandwidth-reducing orderings such as the reverse Cuthill-McKee (RCM) [78] and that of Sloan [202], although as the accuracy of incomplete factorisation approaches that of the direct factorisation, the more sophisticated ordering will begin to provide benefit. A theoretical reason for the improved performance of the RCM ordering for incomplete factorisations was provided by Bridson and Tang [203], who showed that the inverse of the incomplete factorisation under a RCM ordering is fully dense, while this is not necessarily true for the other orderings considered. In contrast, MD and ND orderings have been found to improve the quality of the approximate inverse preconditioners, and thus the rate of convergence of the approximate inverse preconditioned Krylov projection methods [204].

Chapter 3 Performance of conventional approaches on some FELA problems

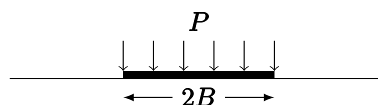
3.1 Test problems

Test problems in both two and three dimensions to compare various approaches for solving the linear systems that arise when computing the search direction at each step of the interior point method are introduced here. The purpose of the problem test set is to provide a realistic indication of how the method should perform when it is applied to solve a broad spectrum of FELA problems in geotechnics. In two dimensions, a strip-footing on frictional material and a tunnel heading in purely cohesive material are considered. A square footing on purely cohesive material, a square excavation in cohesive-frictional material, and a tunnel heading in purely cohesive material are considered in three dimensions. A brief description of each of the test problems is included below, with a diagram of the coarsest mesh used for each problem and a summary detailing the pertinent characteristics of each of the associated optimisation problems. Details of the hand calculated lower and upper bounds for the strip footing are included as a comparison to the more general capability of FELA.

3.1.1 Two-dimensional problems

3.1.1.1 Strip footing

We consider a long strip footing of width $2B$ that rests on a semi-infinite domain of frictional weightless material. For the simulations here, the Mohr-Coulomb soil is assumed to have a cohesion of 1 kPa and a friction angle of 20°, with bounds on the solution which are within 0.5%. The problem is shown in Figure 7 and the mesh in Figure 8. Note that the problem symmetry has been exploited.



Soil parameters (c, ϕ, γ)

Figure 7. Strip footing.

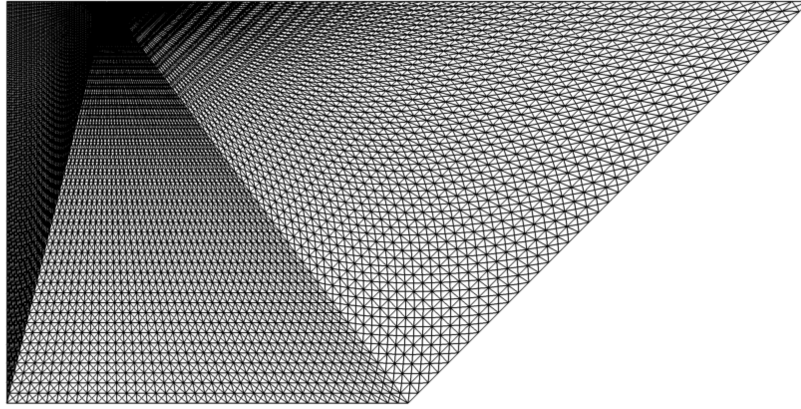


Figure 8. Two-dimensional footing mesh.

The bearing capacity of such a footing is often determined by hand calculation with

$$Q_D = 2BcN_C + 2BqN_q + 2B^2\gamma N_\gamma ,$$

where c is the soil cohesion, q is the surcharge per unit area on the soil surface, γ is the unit weight of the soil, and N_c, N_q and N_γ are bearing capacity factors [35]. With $\gamma = q = 0$ and $B = c = 1$, this problem results in calculation of the factor N_c. A comparison of the bounds computed for N_c as ϕ varies is shown in Figure 9, along with closed solutions due to Prandtl [228] and Terzaghi [35]. As can be seen, the FELA bounds are very tight and in good agreement with the exact values.

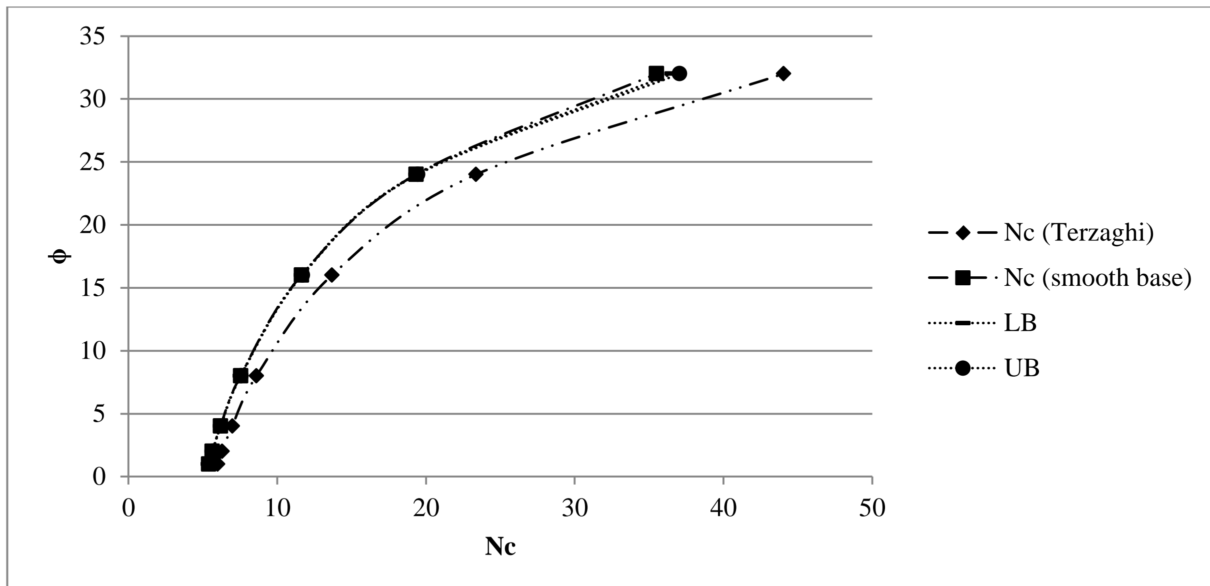


Figure 9. The bearing capacity factor N_c as ϕ varies.

The mesh used here is the coarsest of the three meshes considered. The Terzaghi [35] value is given by

$$N_c = \cot\phi\left(\frac{a_\theta^2}{2\cos^2\left(\dfrac{\pi}{4}+\dfrac{\phi}{2}\right)}-1\right) \text{ and } a_\theta = e^{\left(\frac{3}{4}\pi-\frac{\phi}{2}\right)\tan\phi}.$$

For the perfectly frictionless base [228]

$$N_c = \cot\phi\left(\mathrm{e}^{\pi\tan\phi}\tan^2\left(\frac{\pi}{4}+\frac{\phi}{2}\right)-1\right).$$

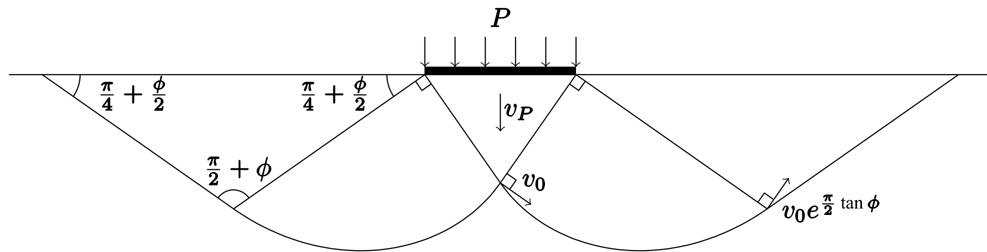


Figure 10. Prandtl failure mechanism.

This value can be derived as an upper bound from the Hill and Prandtl failure mechanisms [3]. The Prandtl failure mechanism is shown in Figure 10. The same value may also be derived as a lower bound by considering an infinite number of symmetrically inclined column stress states superposed (with a corresponding horizontal stress to satisfy equilibrium) [3]. A three column version of a statically admissible stress state is shown in Figure 11.

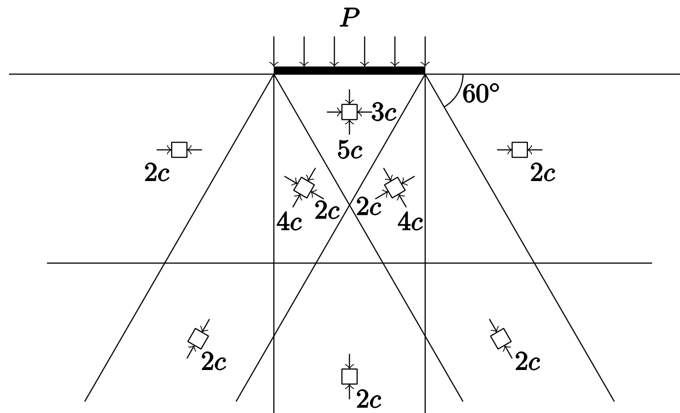


Figure 11. Three column lower bound stress state for strip footing.

3.1.1.2 Circular tunnel in cohesive material

The stability of a tunnel heading has received attention from a number of authors, with solutions being obtained by computing lower bounds by hand [229], upper bounds via rigid blocks [229]–[232], and both lower and upper bounds using FELA with varying solution schemes and degrees of problem complexity [4], [230], [232], [233]. Figure 12 shows a cross-section of a circular tunnel, of infinite length, embedded in a semi-infinite domain of Mohr-Coulomb material. The unit weight of the soil γ is assumed to be 0.5 kN/m^3, the cohesion is $1.0\,\text{kPa}$, the friction angle is 0°, and the bounds computed to within 1.5% of each other. Note that the actual values of c and γ are not critical, since the bound solutions are presented in dimensionless form. The mesh used for the problem is shown in Figure 13.

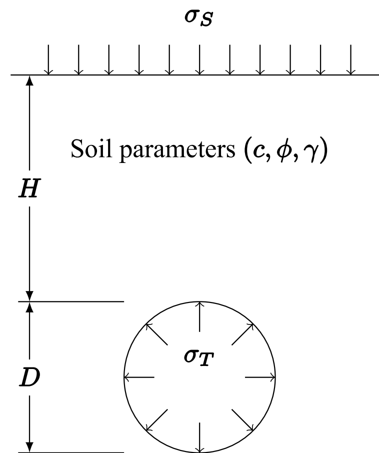


Figure 12. Plane strain tunnel problem.

3.1.2 Three-dimensional problems

The three-dimensional set of problems considered here is designed to provide a realistic test of the performance of the limit analysis solvers for large-scale problems. The connectivity of three-dimensional problems is significantly greater than that for two-dimensional problems, and has been known to cause issues for many interior point solvers. For the square footing and square excavation, both constant strain and linear strain elements can be used in the formulation of the upper bound problem. The latter formulations (post-fixed **UB2*) do not require inter-element discontinuities to achieve good quality bounds, and result in a smaller but more dense constraint matrix [234]. The problems all use the Drucker-Prager yield criterion, with cohesion, friction angle, the

Lode angle (which is used to change the points at which the Drucker-Prager yield surface matches the Mohr-Coulomb yield surface), and the unit weight of the soil.

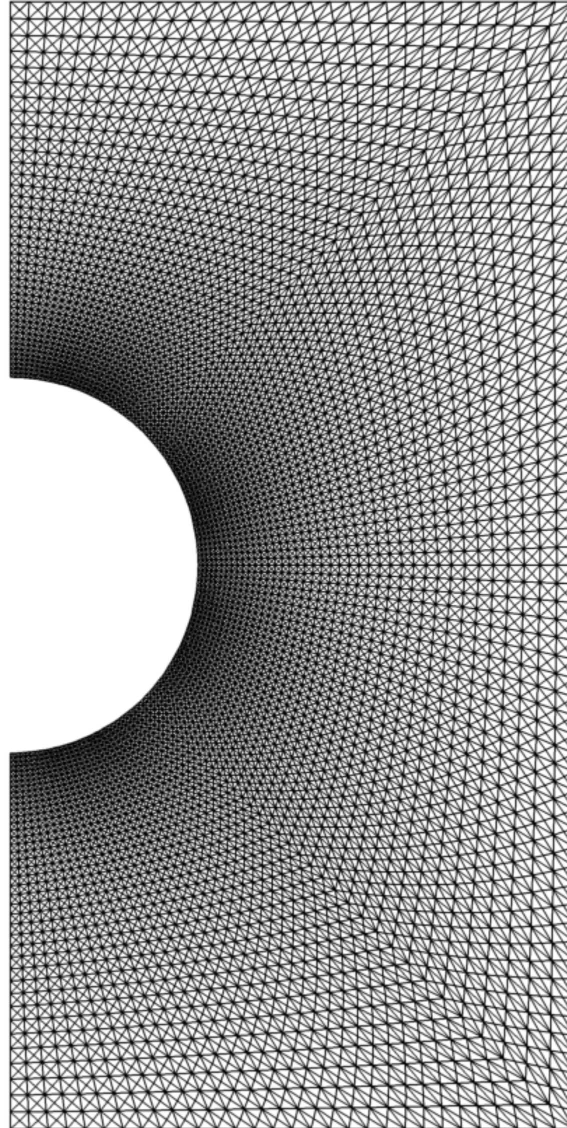


Figure 13. Two-dimensional tunnel mesh.

3.1.2.1 Square excavation in cohesive-frictional material

The square excavation problem optimises the unit weight of the material to obtaining the stability number $\gamma H / c$. While just one eighth of the problem could be used to model the problem by exploiting symmetry, one quarter of the problem has been modelled here to increase the three dimensional connectivity of the problem (as shown in Figure 14). The mesh for this case is shown in Figure 15. There has been little attention given to this problem in the literature, although there has been much work that

has focused on axisymmetric excavation problems [235], [236]. The parameters used in the simulations were $c = 1\text{kPa}$, $\phi = 1^{\circ}$, $\theta = 25^{\circ}$, and $\gamma = 1$.

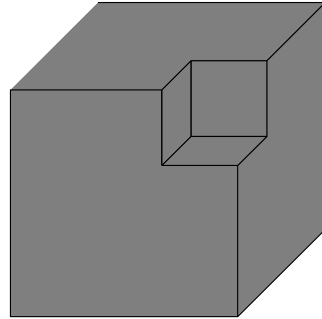


Figure 14. Square excavation.

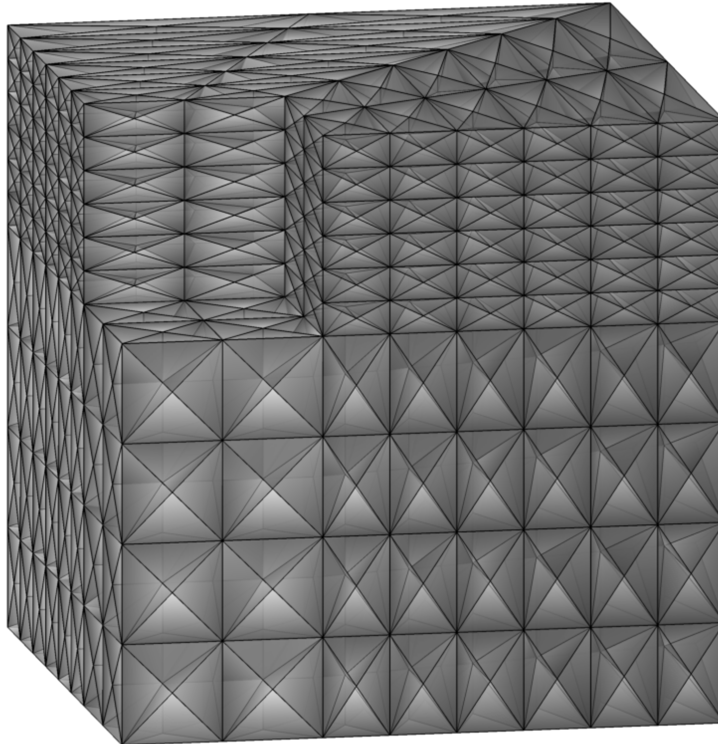


Figure 15. Three-dimensional square excavation mesh.

3.1.2.2 Square footing on weightless cohesive material

The square footing may be considered a special case of the rectangular footing with its breadth B equal to its length L. The problem is similar to that of the strip footing in two dimensions, so that the exact solution for the strip footing is also a lower bound for the square footing [3]. The test problem used has a cohesion of 1kPa, $\phi = 0^{\circ}$, $\theta = 25^{\circ}$, and is weightless. The problem is shown in Figure 16 and the mesh in Figure 17.

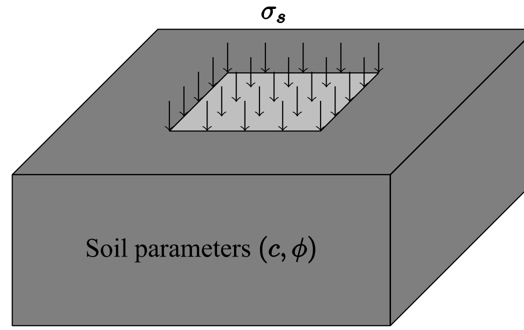


Figure 16. Square footing.

Bounds on the limit loads of rectangular and circular footings have been formulated by Shield and Drucker [237] with an upper bound for a rectangular footing of

$$q_u = c\left(5.24 + 0.47\frac{B}{L}\right)$$

for a weightless cohesive material. For a square footing $B = L$ and an upper bound is simply $q_u = 5.71c$. Since the exact bearing capacity for a strip footing is $q_u = (2+\pi)c \approx 5.14c$, this serves as a rigorous lower bound for the square footing. This can be compared with the best lower and upper bounds computed by FELA here of, respectively, $q_u = 5.63c$ and $q_u = 5.95c$ (a gap of over 5%). Relative to two-dimensional problems, it is much more difficult to achieve tight bounds on the solution in three dimensions due to the rapid growth in computational burden as the mesh is refined.

3.1.2.3 Tunnel heading in cohesive-frictional material

The three-dimensional tunnel heading problem has not received as much attention in the numerical analysis literature as the two-dimensional case (see [238] and references therein). This is due to the added complexity introduced by its three-dimensional nature and the commensurate computational demands that are required to analyse its behaviour. Moreover, the plane strain case considered above provides a conservative estimate of the collapse load [4]. A longitudinal section through the problem considered is shown in Figure 18 with the mesh in Figure 19. The soil parameters used in the simulations were $c = 1\text{kPa}$, $\phi = 10^\circ$, $\theta = 25^\circ$, and $\gamma = 0.5$. The tunnel heading has a cover of $H = 4$, a diameter $D = 2$ and a length of 6. This problem is demonstrably harder to solve, and the gap between the bounds was greater than 30%.

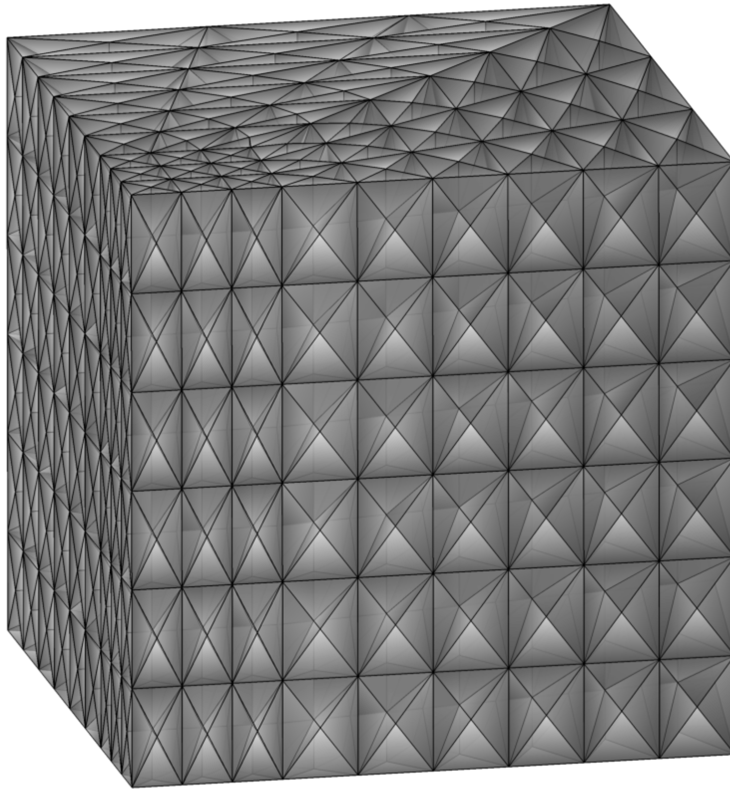


Figure 17. Three-dimensional square footing mesh.

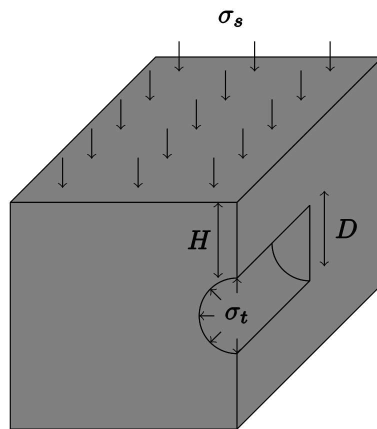


Figure 18. Section through the three-dimensional tunnel heading.

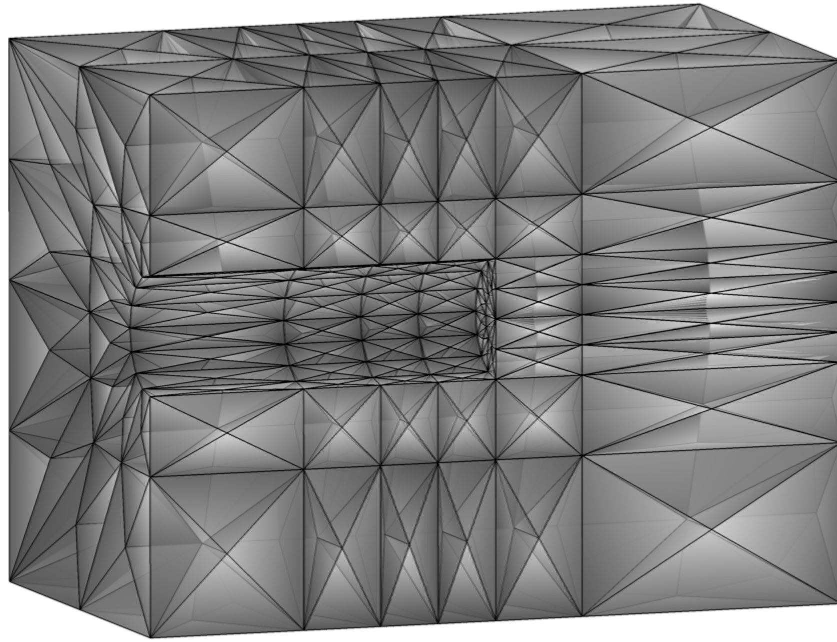


Figure 19. Three-dimensional tunnel heading mesh.

3.1.3 Problem summary

Table 1 and Table 2 present the two- and three-dimensional test problems used to evaluate optimisation solver performance, respectively. The tables provide the following information:

Problem – the name of the problem;

Velocity – the number of velocity nodes in the FELA mesh;

Stress – the number of stress nodes in the FELA mesh;

n_F – the number of free variables in the optimisation problem formulation;

n_{SOC} – the number of variables involved in a second-order conic constraint in the optimisation problem formulation;

n_k – the number of second-order conic inequalities in the optimisation problem formulation;

m – the number of rows in the constraint matrix in the optimisation problem formulation (and hence the number of constraints in the problem);

n – the number of columns in the constraint matrix in the optimisation problem formulation (and hence the number of unknowns in the problem); and

nnz(A) – the number of non-zeros in the constraint matrix in the optimisation problem formulation.

Table 1. Two-dimensional problem summary.

Problem	Elements	Velocity	Stress	n_F	n_{SOC}	n_k	m	n	nnz(A)
2DfootingLBS	232,330	232,330	174,510	280	523,530	174,510	465,360	523,810	2,960,442
2DfootingLBM	524,895	524,895	394,065	420	1,182,195	394,065	1,050,840	1,182,615	6,694,380
2DfootingLBL	935,060	935,060	701,820	560	2,105,460	701,820	1,871,520	2,106,020	11,922,120
2DfootingUBS	232,330	174,510	232,330	980	696,990	232,330	349,020	697,970	2,960,442
2DfootingUBM	524,895	394,065	524,895	1,470	1,574,685	524,895	788,130	1,576,155	6,694,380
2DfootingUBL	935,060	701,820	935,060	1,960	2,805,180	935,060	1,403,640	2,807,140	11,922,120
2DtunnelLBS	76,480	76,480	57,600	58,080	172,800	57,600	211,799	230,880	1,092,936
2DtunnelLBM	172,320	172,320	129,600	130,320	388,800	129,600	476,099	519,120	2,460,206
2DtunnelLBL	306,560	306,560	230,400	231,360	691,200	230,400	845,999	922,560	4,374,676
2DtunnelUBS	76,480	57,600	76,480	77,480	229,440	76,480	192,159	306,920	1,110,556
2DtunnelUBM	172,320	129,600	172,320	173,820	516,960	172,320	432,239	690,780	2,501,036
2DtunnelUBL	306,560	230,400	306,560	308,560	919,680	306,560	768,319	1,228,240	4,448,316

Table 2. Three-dimensional problem summary.

Problem	Elements	Velocity	Stress	n_F	n_{SOC}	n_k	m	n	nnz(A)
3DsqrexcLBS	41,472	41,472	24,576	43,008	147,456	24,576	164,355	190,464	964,142
3DsqrexcLBM	141,696	141,696	82,944	145,152	497,664	82,944	554,048	642,816	3,271,166
3DsqrexcLBL	265,632	265,632	155,136	271,488	930,816	155,136	1,034,703	1,202,304	6,123,526
3DsqrexcUBS	41,472	24,576	41,472	119,280	248,832	41,472	188,927	368,112	1,170,302
3DsqrexcUBM	141,696	82,944	141,696	399,708	850,176	141,696	639,359	1,249,884	3,982,174
3DsqrexcUBL	337,920	196,608	337,920	944,064	2,027,520	337,920	1,517,567	2,971,584	9,477,630
3DsqrexcUB2S	6,144	9,769	24,576	56,726	147,456	24,576	83,189	204,182	2,826,051
3DsqrexcUB2M	20,736	31,741	82,944	184,470	497,664	82,944	273,389	682,134	9,551,771
3DsqrexcUB2L	57,600	87,113	230,400	503,750	1,382,400	230,400	753,077	1,886,150	26,573,448
3DsqrfootLBS	43,956	43,956	25,920	26,244	155,520	25,920	153,648	181,764	932,326
3DsqrfootLBM	105,024	105,024	61,440	62,016	368,640	61,440	363,904	430,656	2,215,008
3DsqrfootLBL	357,264	357,264	207,360	208,656	1,244,160	207,360	1,227,168	1,452,816	7,493,136
3DsqrfootUBS	43,956	25,920	43,956	48,492	263,736	43,956	121,932	312,228	998,460
3DsqrfootUBM	105,024	61,440	105,024	113,088	630,144	105,024	289,728	743,232	2,377,408
3DsqrfootUBL	357,264	207,360	357,264	375,408	2,143,584	357,264	980,208	2,518,992	8,059,824
3DsqrfootUB2S	6,480	10,153	25,920	29,094	155,520	25,920	56,549	184,614	2,862,860
3DsqrfootUB2M	15,360	23,473	61,440	67,014	368,640	61,440	132,149	435,654	6,792,354
3DsqrfootUB2L	51,840	77,257	207,360	219,750	1,244,160	207,360	439,757	1,463,910	22,945,686
3DtunheadLBS	67,788	67,788	40,032	42,300	240,192	40,032	246,167	282,492	1,829,646
3DtunheadLBM	163,472	163,472	95,744	99,920	574,464	95,744	587,983	674,384	4,399,198
3DtunheadLBL	561,240	561,240	326,016	335,736	1,956,096	326,016	1,998,455	2,291,832	15,051,585
3DtunheadUBS	67,788	40,032	67,788	77,400	406,728	67,788	190,151	484,128	1,916,456
3DtunheadUBM	163,472	95,744	163,472	180,848	980,832	163,472	454,879	1,161,680	4,611,356
3DtunheadUBL	561,240	326,016	561,240	600,984	3,367,440	561,240	1,549,007	3,968,424	15,794,780

3.2 Compared solvers

The following software packages are all able to solve semidefinite-quadratic-linear programs (SQLP), interface to `MATLAB`, and are widely available. In all cases, except where noted otherwise, the default settings were used for the comparison. The relative

complementarity gap, and the primal and dual infeasibility tolerances, were by default set to 10^{-8}. All the packages reported here implement infeasible primal-dual IPMs, although there are significant variations in the methods beyond such a categorisation.

3.2.1 MOSEK

`MOSEK` [57] was designed to solve large-scale problems via a predictor-corrector interior point method, and incorporates sophisticated pre-solve and parallel processing. It is also based on the simplified homogeneous and self-dual (HSD) model with NT scaling. In computing the search direction, `MOSEK` uses a left-looking supernodal Cholesky factorisation and uses a graph-partition ordering for the results reported here. The latest version can also solve SDPs as well as LPs, SOCPs, QPs, and general convex NLPs. It also has mixed integer methods and simplex-based procedures, but these are of no use for FELA. `MOSEK` is able to run in parallel, although it is not recommended for problems that can be solved in under 60 seconds. As part of the presolve phase, `MOSEK` includes a linear dependency checker to ensure the full row rank of the constraint matrix. For the large problems considered in this Thesis, this will dominate both time and storage, and so is not performed. Another part of the presolve phase, the eliminator, is used to check whether it is possible and likely to be beneficial to remove any linear or free variables before solving the problem with the IPM. Free variables are embedded in a second-order cone.

3.2.2 Gurobi

`Gurobi` is another commercial optimisation package, and provides similar functionality to `MOSEK`. For solving the FELA problems here, `Gurobi` has an IPM designed to solve LPs and SOCPs. Few details of the solver implementation are readily available, other than it being a primal-dual barrier-based method with a sophisticated presolve. The default is the standard barrier method, with the option to use a HSD embedding to solve the problem being recommended only if the problem appears to be infeasible or if numerical difficulties are encountered. By default, `Gurobi` automatically chooses between a nested dissection and AMD ordering.

3.2.3 SDPT3 4.0

`SDPT3` (version 4.0) [70], [81] now contains both a three-term HSD (not the simplified HSD) method (termed `SDPT3HSD` herein) and a primal-dual infeasible-interior point

predictor-corrector algorithm (termed SDPT3SQL herein). By default, it uses NT scaling for second-order cone variables. There is no central path neighbourhood enforcement used in the SQLP solver of SDPT3. The Schur complement is factorised using the standard MATLAB [103] Cholesky decomposition, the left-looking supernodal CHOLMOD [102], with an approximate minimum degree (AMD) ordering. It should be noted that the ordering is not handled explicitly in SDPT3, relying on MATLAB to do so each time the system is factorised. The decomposition is then used to precondition a symmetric quasi-minimal residual iterative solver implementation [82]. Unfortunately, the supernodal factors constructed by CHOLMOD are destroyed when the factors are returned in MATLAB, so the supernodes are not able to be exploited in the Krylov solver preconditioning operation. Free variables are addressed with the addition of slack variables and then treating the free variables as the difference between the two linear variables.

3.2.4 SeDuMi 1.31

SeDuMi [58] implements a primal-dual interior point algorithm using a simplified HSD embedding technique. The search direction at each iteration is obtained through a supernodal-based $\mathbf{LDL}^T$ factorisation with a multiple minimum degree ordering. If the decomposition does not yield an answer with sufficient accuracy, the decomposition is used as a preconditioner for the conjugate gradient (CG) method [80]. Free variables are embedded in a second-order cone.

3.2.5 Mix8

Mix8 is a standalone FELA package developed at the University of Newcastle. The IPM is based on a simplified HSD scheme and uses NT scaling with Mehrotra's predictor corrector method for the search direction based on the extension to SOCP described by Andersen *et al.* [57]. HSL MA57 [79] is used to build a $\mathbf{LDL}^T$ decomposition of the Schur complement equation with the multifrontal method using the HSL MC50 implementation of AMD for the ordering. Different step lengths in the primal and dual variables are not allowed. Free variables are embedded in a second-order cone of dimension 2 for numerical stability.

3.3 Comparison results

The solvers described in the previous section were used to solve the set of test problems presented in Section 3.1. The simulations were all run using a single thread on an Apple MacBook Pro with an Intel Core i7-3740QM 2.70GHz CPU with 16.0GB RAM running Windows 7 (64 bit) and MATLAB R2012b. `Mix8` was compiled using the Intel Fortran Composer XE 2013, with the Intel MKL BLAS routines used where suitable. The results are discussed below.

For each analysis, a range of values were recorded in order to compare the performance of the various solver packages. For each problem, the following values were recorded:

nit – the number of IPM iterations;

t_T – the total time taken to solve the problem in seconds;

t_P – the time spent in the presolve phase in seconds ('-' denotes that the solver does not use a presolve phase);

t_O – the time taken for the ordering method in seconds (as noted above, `SDPT3` does not explicitly use an ordering, but relies on `MATLAB`'s internal `chol` function to perform the ordering before factorising the matrix, and, for `Mix8`, the analysis phase for `MA57` which incorporates the `MC50` ordering is displayed);

pobj – the final primal objective value reported by the solver;

pfeas – the final primal infeasibility reported by the solver;

dfeas – the final dual infeasibility reported by the solver;

gap – the final normalised complementarity gap reported by the solver;

m – the number of rows in the constraint matrix $\mathbf{A}$ as reported by the solver after any presolving is completed (and hence the number of constraints in the problem);

n – the number of columns in the constraint matrix $\mathbf{A}$ as reported by the solver after any presolving is completed (and hence the number of variables in the problem);

nnz(A) – the number of non-zeros in the constraint matrix $\mathbf{A}$ as reported by the solver after any presolving or free variable conversion is completed; and

nnz(L) – the number of non-zero values in the factorisation determining the search direction at each iteration of the IPM. The number of non-zeros in `SDPT3`'s factorisations vary from iteration to iteration, probably due to the use of random numbers within the AMD ordering performed before each factorisation. Thus, *nnz(L)* reported here for the `SDPT3` solvers is the maximum number of non-zeros present in the factors over the duration of the IPM, although the differences are usually negligible.

3.3.1 Smaller problems

The coarsest mesh problems (those problems in Table 1 and Table 2 with an 'S' at the end of the problem name) were analysed using all of the solvers described in the previous section. These "smaller" problems are not small by modern standards with roughly 100,000 to 500,000 variables, but do not prove to be a serious computational burden for the better solvers with solution times generally less than 90 seconds. A column chart of the total runtime for the problems in the small test set is shown in Figure 20 and a summary of the results for the coarse mesh problems is provided in Table 3. See above for a description of the values displayed.

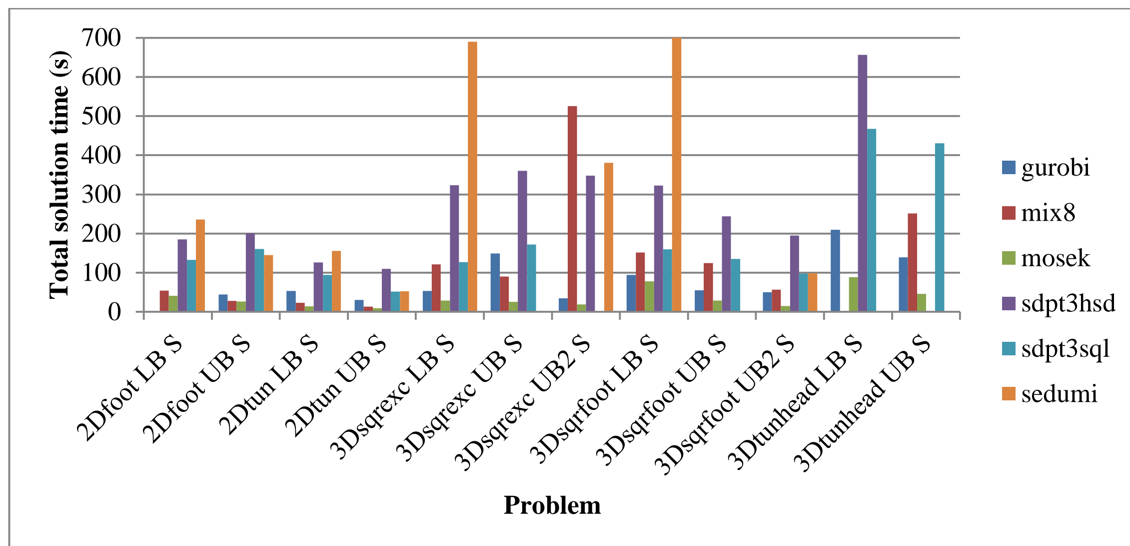


Figure 20. Comparison of the total solution time of the small problem set between all available solvers.Note that two of the `SeDuMi` values have been cut off (the values are available in Table 3) and simulations that did not successfully converge have been removed from the chart.

Considering Figure 20, it is obvious that there is a large difference between the runtime performance of the solvers considered and that the four two-dimensional problems take, in general, less time to solve than the eight three-dimensional problems. As can be seen

in the figure and Table 3, SeDuMi exhibits rather unpredictable behaviour compared to its open-source rivals SDPT3. SeDuMi was over twice as fast as the HSD variant on the quadratic formulation of the upper bound for the square footing and the upper bound of the two-dimensional tunnel problem, but taking twice as long on, for example, the quadratic formulation of the square excavation upper bound. More concerning is SeDuMi's difficulty in achieving primal feasibility, and thus convergence, in all of the lower bound problems except the square footing. SeDuMi also fails to converge on the upper bounds for the square footing and the tunnel heading and was the only solver that had severe numerical difficulties on more than one of the problems. In comparison to the better solvers, SeDuMi close to an order of magnitude slower than MOSEK for the majority of the coarse mesh problems. Note that while SeDuMi embeds all of the free variables into a single second order cone by default, splitting the free variables (that is, using a slack variable to convert the free variable into the difference of two linear variables) does not provide any net performance gain. Because of the poor performance exhibited on these smaller problems, SeDuMi was not used to analyse the larger problems in the test set.

SDPT3SQL was, on average, over 50% faster than its HSD counterpart, SDPT3HSD and was slower only on the upper bound for the tunnel heading, which is the only problem where the HSD formulation completed in fewer iterations. While this is partially due to the increased work required to solve the HSD formulation at each IPM iteration as compared with the standard approach apparent both theoretically and in the time per iteration, such a discrepancy between the methods is an unexpected observation. Interestingly, SDPT3SQL was the only solver to achieve full convergence on *3DtunheadLBS*, with Mix8 failing totally, SeDuMi stopping prematurely, and the two commercial solvers along with SDPT3HSD failing to achieve primal feasibility, although SDPT3HSD did make more progress than MOSEK and Gurobi. In general, though, the MATLAB-based solvers were slower than the standalone and commercial solvers.

Of these standalone solvers, it is clear that MOSEK exhibits a comparatively higher level of performance in terms of total runtime, beating the other commercial solver, Gurobi, by a factor of two or more in most of the problems. It was also the most consistent, solving each of the problems to the required accuracy in less than 26 iterations. It

should be noted, however, that MOSEK converged to a slightly different solution compared with all of the other solvers for problems *2DtunnelLBS* and *2DtunnelUBS*. This discrepancy could be a result of rounding errors as a result of scaling the coefficient matrix, but most likely it is caused by the significant elimination of constraints and variables in the presolve phase, and the removal of constraints identified as redundant. For both of these problems (as well as *2DfootingLBS* and some of the three-dimensional problems), MOSEK eliminates more constraints than there are free variables, indicating that the presolver has removed constraints considered redundant (and that the constraint matrix is thus not considered to be full rank). MOSEK's documentation suggests reducing the tolerances used to identify redundant constraints if such a problem is expected, and that if this improves the situation then the problem is poorly formulated [75]. It is also noteworthy that MOSEK eliminates some of the conically-constrained variables in *2DfootingLBS* and *3DtunheadLBS*. Regardless of the cause, such behaviour is unexpected and strongly suggests that one should check the validity of their solutions, even they are reported by well-regarded software as "optimal" and "feasible".

Unsurprisingly, the commercial solvers, MOSEK and Gurobi, were generally better in terms of runtime and robustness, although Gurobi did exhibit some unexpectedly high IPM iteration counts for some of the problems, with the worst being *2DfootingLBS* with 126 iterations (where it was also the slowest solver tested). It also struggled to achieve sufficient primal feasibility on the two-dimensional lower bound problems and *3DtunheadLBS*. The better performance across the board is likely due to three main factors, the presolve phase removing many or all of the free variables from the problem formulation, the nested dissection ordering, and the purpose-built Cholesky factorisation routines used to obtain the search direction.

Mix8 performs well on the two-dimensional problems, but is slower when solving those in three dimensions, and even failing on the lower bound tunnel heading (*2DtunheadLBS*), with MA57 reporting that the Schur complement matrix is singular in the first IPM iteration. There are obviously worthwhile gains possible for the *UB2 problems where Mix8 exhibits slow convergence. It should be noted that the MA57 documentation [79] does recommend using the graph partition-based nested dissection

ordering in METIS for large-scale matrices, but, for this subset of the problems, there is a negligible to small improvement over the MC50 AMD ordering.

Another important difference between the two commercial solvers and the others is the eliminations performed during the presolve phase. In some of the problems, all of the free variables are eliminated from the problem before the problem is solved. In some cases, this leads to a significant reduction in the size of the problem (reducing both constraints and unknowns), as well as reducing the number of non-zero entries in the constraint matrix. The reduction in the number of constraints is most pronounced in the linear strain element-based upper bound formulations (*UB2 problems). This may be the basis of the difficulty encountered by the Mix8 solver on these problems.

3.3.2 Finer mesh problems

The slightly larger problems (identified by the 'M' at the end of the problem name) were simulated using all of the solvers except SeDuMi, which was deemed to be too slow on the small-scale problems to warrant its use on larger problems. The results of these analyses are shown in Figure 21 and Table 4. For a description of the table values, see Section 3.3 above.

The difference between the two- and three-dimensional problems is more pronounced in Figure 21 than the corresponding chart for the small problem set yet is similar to that of the smaller problems, generally sitting within an order of magnitude within one another. Again, the cases in which some solvers were unable to compute acceptable solutions included MOSEK on the two-dimensional tunnel problems, where it reports an optimal solution that is over 20% and 13% different to the respective lower and upper bound solutions reported by all the remaining solvers. Mix8 failed on the first iteration for *3DtunheadLBM*, reporting a singular coefficient matrix. SDPT3SQL had numerical difficulties on a few of the problems, having to stop because the Schur complement was either singular or indefinite to machine precision. Even so, SDPT3SQL was only unable to solve one of those problems, *3DsqrexcUB2M*, to sufficient accuracy.

Table 3. Available solver performance on the small size problems.

Problem	Solver	nit	t_T	t_P	t_O	pobj	pfeas	dfeas	gap	m	n	nnz(A)	nnz(L)
	sedumi	8	235.7	-	85.7	-14.629	1E-5	7E-7	6E-12	465,360	523,812	2.96E+6	2.14E+7
	sdpt3sql	35	133.1	-	-	-14.834	9E-10	3E-8	4E-5	465,360	524,090	2.96E+6	2.29E+7
2DfootingLBS	sdpt3hsd	40	185.5	-	-	-14.834	5E-8	2E-6	7E-5	465,360	524,090	3.14E+6	2.48E+7
	gurobi	126	332.6	2.0	2.1	-14.834	3E-4	6E-9	8E-10	464,800	523,250	2.96E+6	1.98E+7
	mosek	25	41.2	0.3	8.2	-14.831	6E-9	6E-9	6E-9	464,940	523,391	2.79E+6	1.82E+7
	mix8	29	54.0	-	0.2	-14.832	2E-8	2E-8	2E-8	465,360	524,090	2.79E+6	2.75E+7
	sedumi	22	145.3	-	56.9	-14.916	1E-9	3E-9	1E-14	349,020	697,972	2.96E+6	1.97E+7
	sdpt3sql	51	160.8	-	-	-14.917	2E-10	3E-11	8E-9	349,020	698,950	2.96E+6	1.93E+7
2DfootingUBS	sdpt3hsd	56	200.3	-	-	-14.917	2E-8	1E-11	2E-9	349,020	698,950	3.20E+6	1.84E+7
	gurobi	32	44.7	2.3	1.5	-14.917	8E-11	9E-14	2E-11	348,320	697,270	2.96E+6	1.53E+7
	mosek	20	26.4	0.5	5.1	-14.914	5E-10	7E-9	3E-9	348,040	696,990	1.91E+6	1.41E+7
	mix8	22	28.2	-	0.1	-14.916	8E-9	7E-9	7E-9	349,020	698,950	1.92E+6	1.88E+7
	sedumi	25	155.7	-	15.8	-0.787	8E-5	4E-7	8E-11	211,799	230,882	1.09E+6	6.38E+6
	sdpt3sql	68	94.3	-	-	-0.791	3E-10	2E-9	7E-7	211,799	288,960	1.44E+6	6.45E+6
2DtunnelLBS	sdpt3hsd	76	125.9	-	-	-0.791	5E-7	3E-8	9E-6	211,799	288,960	1.73E+6	7.13E+6
	gurobi	59	53.9	1.0	1.0	-0.791	3E-5	3E-10	1E-10	210,819	229,900	1.09E+6	6.23E+6
	mosek	19	14.5	0.9	3.3	-0.767	5E-8	3E-8	3E-8	106,462	182,924	1.61E+6	6.32E+6
	mix8	43	23.2	-	0.1	-0.790	6E-9	8E-9	8E-9	211,799	288,960	1.32E+6	7.84E+6
	sedumi	31	52.4	-	14.0	-0.824	1E-7	2E-9	2E-13	192,159	306,922	1.11E+6	5.85E+6
	sdpt3sql	48	52.0	-	-	-0.824	1E-12	2E-11	8E-9	192,159	384,400	1.46E+6	4.49E+6
2DtunnelUBS	sdpt3hsd	79	109.5	-	-	-0.824	6E-9	3E-11	7E-9	192,159	384,400	1.84E+6	5.47E+6
	gurobi	45	30.9	1.4	0.9	-0.824	3E-11	8E-14	4E-12	191,069	305,830	1.10E+6	5.46E+6
	mosek	17	9.5	1.0	1.2	-0.796	4E-8	3E-8	3E-8	51,916	243,158	7.69E+5	3.89E+6
	mix8	26	13.6	-	0.1	-0.824	5E-9	6E-9	6E-9	192,159	384,400	1.06E+6	6.25E+6
	sedumi	18	689.8	-	13.5	-121.988	4E-5	5E-7	1E-10	164,355	190,466	9.64E+5	3.43E+7
	sdpt3sql	26	127.5	-	-	-121.989	3E-11	7E-8	3E-7	164,355	233,472	1.22E+6	3.29E+7
3DsqrexcLBS	sdpt3hsd	57	323.3	-	-	-121.989	7E-7	9E-13	5E-13	164,355	233,472	1.42E+6	3.44E+7
	gurobi	20	53.2	2.0	3.9	-121.988	6E-8	4E-10	2E-10	145,924	172,033	9.27E+5	2.17E+7
	mosek	17	29.3	0.2	2.6	-121.987	1E-8	3E-8	3E-8	121,348	147,458	1.41E+6	2.08E+7
	mix8	19	121.1	-	0.1	-121.988	7E-10	7E-9	7E-9	164,355	233,472	1.74E+6	3.60E+7
	sedumi	6	152.3	-	13.9	-31.145	1E+1	3E-2	2E-5	188,927	368,114	1.17E+6	2.77E+7
	sdpt3sql	43	171.8	-	-	-155.150	5E-10	3E-9	9E-9	188,927	487,392	1.59E+6	3.44E+7
3DsqrexcUBS	sdpt3hsd	69	360.2	-	-	-155.150	2E-6	4E-13	6E-13	188,927	487,392	2.10E+6	3.31E+7
	gurobi	52	149.3	42.2	2.9	-155.150	9E-8	4E-10	9E-10	112,752	291,937	9.99E+5	1.75E+7
	mosek	19	25.5	0.3	1.3	-155.147	3E-8	4E-8	4E-8	69,648	248,834	7.89E+5	1.50E+7
	mix8	23	89.9	-	0.1	-155.149	1E-9	9E-9	9E-9	188,927	487,392	1.32E+6	3.04E+7
	sedumi	30	380.8	-	4.8	-138.248	4E-7	2E-9	6E-13	83,189	204,184	2.83E+6	1.83E+7
	sdpt3sql	63	188.8	-	-	-137.365	6E-12	2E-3	1E-2	83,189	260,908	3.50E+6	1.76E+7
3DsqrexcUB2S	sdpt3hsd	93	347.7	-	-	-138.248	1E-7	8E-11	4E-9	83,189	260,908	3.76E+6	1.85E+7
	gurobi	30	34.3	1.6	1.2	-138.248	9E-11	9E-12	3E-10	52,010	173,003	2.62E+6	8.10E+6
	mosek	26	19.1	0.6	1.3	-138.246	2E-7	2E-8	2E-8	31,227	152,221	1.15E+6	6.92E+6
	mix8	49	525.3	-	0.3	-138.242	2E-9	9E-9	9E-9	83,189	260,908	1.99E+6	3.83E+7
	sedumi	21	6539.8	-	8.7	-5.492	1E-6	4E-9	2E-12	153,648	181,766	9.32E+5	3.43E+7
	sdpt3sql	33	159.5	-	-	-5.492	5E-12	1E-10	7E-9	153,648	208,008	1.13E+6	3.30E+7
3DsqrfootLBS	sdpt3hsd	59	322.3	-	-	-5.492	7E-9	4E-11	1E-9	153,648	208,008	1.26E+6	3.33E+7
	gurobi	40	94.5	1.1	5.3	-5.492	2E-10	6E-11	4E-11	153,252	181,440	9.31E+5	2.59E+7
	mosek	21	78.2	0.8	4.6	-5.492	9E-9	2E-8	2E-8	109,786	163,895	2.12E+6	3.28E+7
	mix8	23	152.0	-	0.1	-5.492	5E-10	5E-9	5E-9	153,648	208,008	1.79E+6	3.55E+7
	sedumi	14	418.4	-	6.1	-6.484	2E-1	2E-6	8E-10	121,932	312,230	9.98E+5	3.14E+7
	sdpt3sql	34	135.1	-	-	-6.234	2E-13	1E-10	1E-8	121,932	360,720	1.21E+6	3.25E+7
3DsqrfootUBS	sdpt3hsd	51	244.5	-	-	-6.234	9E-9	2E-10	3E-9	121,932	360,720	1.45E+6	3.01E+7
	gurobi	23	55.1	1.0	5.0	-6.234	9E-10	5E-10	2E-11	119,340	309,636	9.75E+5	2.21E+7
	mosek	18	29.1	0.7	2.3	-6.234	9E-9	2E-8	2E-8	36,430	270,023	1.03E+6	1.53E+7
	mix8	25	124.5	-	0.1	-6.234	3E-10	3E-9	3E-9	121,932	360,720	1.19E+6	3.08E+7
	sedumi	21	98.6	-	3.1	-6.170	5E-7	2E-9	9E-13	56,549	184,616	2.86E+6	1.07E+7
	sdpt3sql	44	97.7	-	-	-6.170	3E-13	1E-10	6E-9	56,549	213,708	3.48E+6	1.16E+7
3DsqrfootUB2S	sdpt3hsd	76	195.2	-	-	-6.170	6E-9	8E-11	6E-9	56,549	213,708	3.62E+6	1.17E+7
	gurobi	40	50.5	1.6	1.3	-6.170	1E-9	1E-11	3E-11	54,605	182,645	2.73E+6	1.14E+7
	mosek	19	15.0	0.5	1.1	-6.169	1E-8	3E-8	3E-8	27,882	181,843	1.06E+6	7.92E+6
	mix8	32	56.9	-	0.1	-6.170	3E-9	4E-9	4E-9	56,549	213,708	2.00E+6	1.16E+7
	sedumi	10	546.6	-	21.5	-18.848	4E-2	5E-4	6E-8	246,167	282,494	1.83E+6	5.13E+7
	sdpt3sql	60	467.6	-	-	-22.395	2E-8	2E-9	5E-8	246,167	324,792	2.26E+6	5.37E+7
3DtunheadLBS	sdpt3hsd	76	656.5	-	-	-22.395	1E-6	6E-11	3E-10	246,167	324,792	2.46E+6	5.34E+7
	gurobi	47	210.0	1.9	8.0	-22.395	9E-5	9E-7	2E-10	243,708	280,225	1.82E+6	3.87E+7
	mosek	24	88.3	0.5	4.9	-22.394	2E-5	7E-8	6E-7	203,063	239,581	2.39E+6	3.75E+7
	mix8	0	11.1	-	0.2	0.000	3E-1	1E+0	1E+0	246,167	324,792	2.91E+6	5.55E+7
	sedumi	24	800.6	-	14.2	-33.423	3E-2	1E-7	2E-11	190,151	484,130	1.92E+6	4.44E+7
	sdpt3sql	73	430.4	-	-	-33.432	8E-9	4E-10	1E-8	190,151	561,528	2.39E+6	4.91E+7
3DtunheadUBS	sdpt3hsd	43	317.9	-	-	-33.432	7E-6	6E-6	1E-4	190,151	561,528	2.76E+6	4.96E+7
	gurobi	29	139.2	1.6	5.9	-33.432	6E-6	2E-9	7E-11	184,860	478,837	1.87E+6	3.04E+7
	mosek	20	46.1	0.4	2.0	-33.430	4E-8	4E-8	4E-8	112,752	406,730	1.21E+6	2.62E+7
	mix8	44	251.1	-	0.1	-33.432	7E-9	5E-11	5E-11	190,151	561,528	1.85E+6	4.18E+7

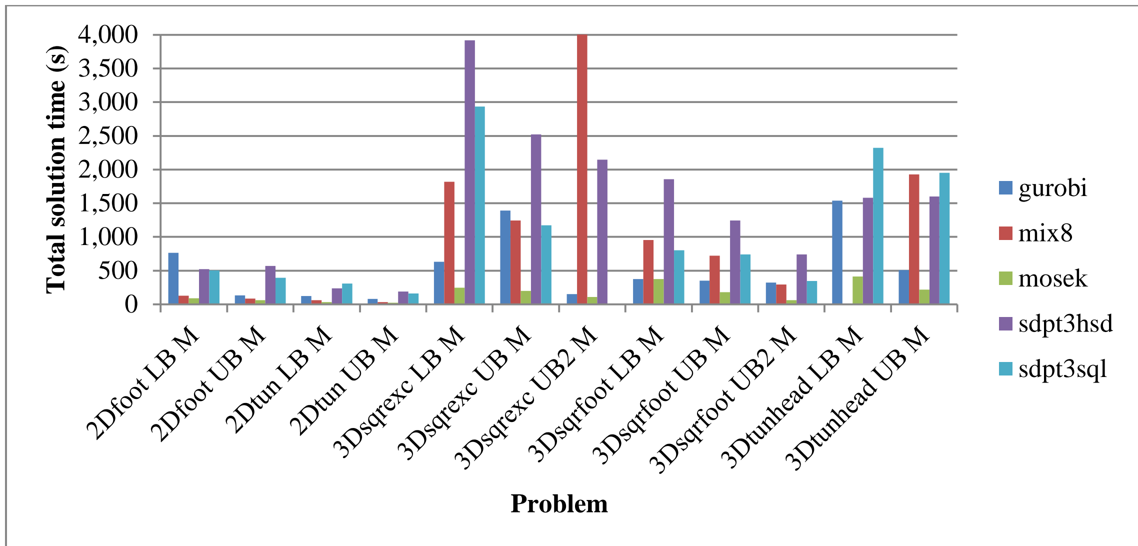


Figure 21. Comparison of the total solution time of the medium problem set between some of the available solvers. Note that one of the Mix8 *values has been cut off (the value is available in Table 4) and simulations that did not successfully converge have been removed from the chart.*

The two simplified HSD formulations, MOSEK and Mix8, generally had the lowest iteration counts of approximately 15 to 25 iterations. Notable exceptions were the *2DtunnelLBM* and *3DsqrexcUB2M*, where the Mix8 solver was slower to converge and even stopped due to a maximum number of iterations for the latter problem. Unfortunately, the *UB2 problems still proved difficult for all of the non-commercial solvers yet, in addition to the improved upper bounds computed, were solved significantly faster than their counterpart *UB problems by the commercial solvers. For example, on the square footing problem, MOSEK solved the quadratic formulation three times faster than *3DsqrfootUB* with a bound improvement of 1%. Similar to the small problems, Gurobi spent a large number of iterations on some of the problems and had a blow-out in the presolve time on *3DsqrexcUBM*. In general, though, it reliably attained a suitably accurate and feasible solution.

MOSEK was the fastest solver on every problem except *3DsqrfootLBM*, where Gurobi was less than a quarter of a percent faster. Furthermore, MOSEK was the fastest solver per iteration except on *3DsqrfootLBM* (taking the presolve and ordering time into account makes negligible difference) and also the lowest iteration count on every problem but *3DsqrexcUB2M*. The per iteration time difference may be correlates with the dimension of the system to be factorised and the amount of fill-in in the factor with MOSEK

eliminating more equations and variables from the original problem formulation in every problem but *2DfootingLBM*, where `Gurobi` has a slight lead over `MOSEK`. `MOSEK` also has fewer non-zeros in the factorisation in every problem except *3DsqrfootLBM* where it is slower than `Gurobi` on a per iteration basis but as `Gurobi` takes 31 iterations to convergence versus `MOSEK`'s 21, `MOSEK` takes less than a second longer to solve the problem than `Gurobi`'s 374.4s. This shows that performance improvements are possible in the details of the IPM algorithm as well as in the linear solver.

The largest problems (with a 'L' at the end of the problem name) were only solved using the two commercial solvers, `MOSEK` and `Gurobi`, with the remaining solvers requiring too much time and/or memory to solve these problems. Both of these solvers presolve the problem and use a nested dissection ordering with a supernodal Cholesky solver, a clear sign of the suitability of the combination. Note that all bar one of these problems has over a million unknowns in the original optimisation problem formulation (the exception being the *2DtunnelLBL*, still with over 900,000 variables), with the largest having almost four million variables. The number of constraints in these problems ranges from 750,000 to 2,000,000, indicating that these are truly large-scale problems. The results of these analyses are shown in Table 5 (the description of the table values is provided in Section 3.3 above).

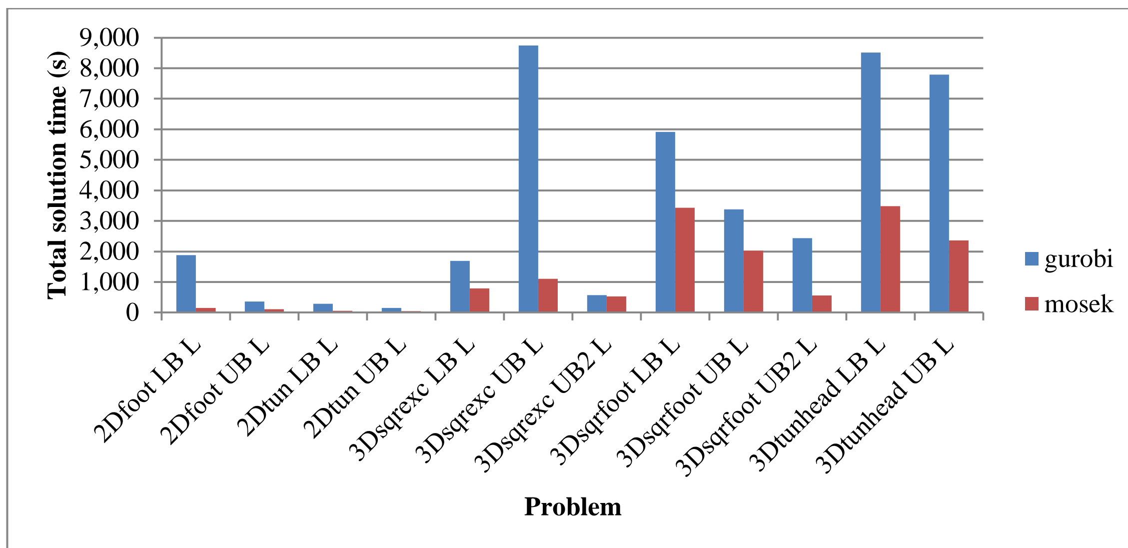


Figure 22. Comparison of the total solution time of the large problem set between the two commercial solvers. Note that simulations that did not successfully converge have been removed from the chart

For the two-dimensional tunnel problems *2DtunnelLBL* and *2DtunnelUBL*, MOSEK's results are even further from the optimal value than on the coarser meshes, with both solutions more than 40% off the Gurobi results. What is most striking about these results is the extreme growth in the time required to solve some of the problems over their coarser mesh counterparts. The solution times ranged from just under 10 minutes to over two hours for the three-dimensional problems, while MOSEK solved the two-dimensional problems in approximately three minutes or less. The algorithmic details of the IPM implementation evidently have an impact, as seen in the better performance of MOSEK over Gurobi on these larger problems in the number of IPM iterations spent converging to a solution with Gurobi taking two to 9 times more iterations than MOSEK except on *3DsqrexcUB2L*, where Gurobi converges in 24 iterations against MOSEK's 20. Between these two solvers, it is the capability of the IPM algorithm to converge within a small number of iterations that drives the total performance of the solver, but the ability to efficiently obtain the search direction as the problems become larger that is the primary factor in whether or not a given problem is able to be solved. MOSEK is again faster than Gurobi on a per iteration basis on every problem except *3DsqrfootLBL*.

The relative performance of Gurobi and MOSEK on the large problems in the test set supports the notion that any performance enhancement effort should target the approach used to obtain the search direction at each iteration of the IPM in order to ensure progress can be made efficiently towards a solution. In addition, the algorithm used by the MOSEK solver generally dominates that used in Gurobi with a consistent and low iteration count on all the problems in the test set, which drives the total performance of the solver.

3.3.3 Comparison summary

Over the set of small, medium, and large problems, a few important patterns emerge concerning the effort necessary to solve the problems as they grow in size. The average number of IPM iterations remains constant as the problems grow in size, with an average of 35 to 40 iterations per problem for the results provided. A minimum of 15 to a maximum of 150 iterations are needed to obtain solutions of sufficient accuracy. The time and storage required to solve the three-dimensional problems, however, shows a different behaviour as they grow in size, despite the fact that the number of entries in

the constraint matrix is constant at approximately 5 to 15 per row (with roughly the same number of non-zeros per column).

While the use of the nested dissection ordering in `Gurobi` and `MOSEK` appears superior to the AMD orderings used in the other solvers, this difference plays a secondary role to the problem dimensionality. The ratio of the number of non-zeros in the factorisation to the number of non-zeros in the constraint matrix, as a function of the number of unknowns in the problem, is much higher for the three-dimensional problems than for the two-dimensional problems. This is due to the greater connectivity in the three-dimensional mesh, resulting in a higher level of fill-in.

Interestingly, in the linear strain element-based upper bound method the storage growth observed is not as severe as the formulations with inter-element discontinuities, since the factorisations in the UB2 problems have a fill-in ratio that is roughly comparable with that in two dimensions. A summary of the ratio of the non-zeros in the factorisation to the non-zeros in the constraint matrix as reported by `MOSEK` is shown in Table 6. Even though the different formulation can lead to a significant improvement in terms of tightness of the limit load computed and a reduction in runtime, there is still a distinct difference in the runtime behaviour between the three-dimensional and two-dimensional problems. This is seen clearly in Figure 23, which shows the time per iteration as a function of the number of unknowns in the problem as solved (after presolving and variable conversion has taken place) for `MOSEK` on all of the problems in the test set.

The runtime for the two-dimensional problems, at this scale, appears almost linear, while the three-dimensional problems display a distinctly nonlinear growth. This behaviour is evident across all the solvers considered, regardless of whether time per iteration or total run time is considered (because of the relatively constant number of IPM iterations as the problem size grows), or whether the number of constraints or the number of unknowns is used as a measure of the size of the problem. It is hoped that, through a suitable use of parallelisation and some combination of efficient iterative and direct methods, the growth in solution time for the three-dimensional problems can be contained to behave more like the two-dimensional problem solution time.

Table 4. Available solver performance on the medium size problems.

Problem	Solver	nit	t_T	t_P	t_O	pobj	pfeas	dfeas	gap	m	n	nnz(A)	nnz(L)
2DfootingLBM	sdpt3sql	48	504.2	-	-	-14.834	2E-9	2E-9	2E-6	1,050,840	1,183,035	6.69E+6	6.67E+7
	sdpt3hsd	46	520.1	-	-	-14.834	1E-7	7E-8	1E-5	1,050,840	1,183,035	7.09E+6	6.50E+7
	gurobi	115	766.4	4.6	5.3	-14.834	4E-5	5E-7	1E-9	1,050,000	1,181,775	6.69E+6	5.00E+7
	mosek	20	90.0	0.7	20.5	-14.825	8E-9	8E-9	8E-9	1,050,210	1,181,986	6.32E+6	4.58E+7
	mix8	24	127.3	-	0.4	-14.832	1E-8	2E-8	2E-8	1,050,840	1,183,035	6.30E+6	6.86E+7
2DfootingUBM	sdpt3sql	55	393.4	-	-	-14.891	3E-10	2E-11	9E-9	788,130	1,577,625	6.70E+6	4.45E+7
	sdpt3hsd	63	570.0	-	-	-14.891	3E-8	1E-11	2E-9	788,130	1,577,625	7.23E+6	4.84E+7
	gurobi	37	134.8	4.9	3.6	-14.891	2E-11	9E-14	8E-12	787,080	1,575,105	6.69E+6	3.76E+7
	mosek	20	62.3	1.1	12.4	-14.886	5E-10	5E-9	2E-9	786,660	1,574,685	4.32E+6	3.41E+7
	mix8	23	86.2	-	0.3	-14.885	9E-9	8E-9	8E-9	788,130	1,577,625	4.33E+6	5.13E+7
2DtunnelLBM	sdpt3sql	87	310.0	-	-	-0.798	3E-10	1E-10	7E-8	476,099	649,440	3.24E+6	1.77E+7
	sdpt3hsd	60	237.7	-	-	-0.798	6E-6	6E-7	2E-4	476,099	649,440	3.89E+6	1.82E+7
	gurobi	56	123.8	2.0	2.3	-0.798	5E-5	2E-11	4E-10	474,629	517,650	2.46E+6	1.56E+7
	mosek	16	33.3	2.0	7.3	-0.649	4E-8	4E-8	4E-8	267,653	439,945	3.18E+6	1.55E+7
	mix8	44	60.5	-	0.2	-0.798	5E-9	7E-9	7E-9	476,099	649,440	2.98E+6	1.90E+7
2DtunnelUBM	sdpt3sql	60	160.0	-	-	-0.816	4E-12	1E-11	7E-9	432,239	864,600	3.28E+6	1.16E+7
	sdpt3hsd	56	191.5	-	-	-0.816	9E-8	5E-6	6E-4	432,239	864,600	4.15E+6	1.38E+7
	gurobi	47	80.3	2.9	2.1	-0.816	4E-10	6E-13	2E-12	430,604	689,145	2.49E+6	1.39E+7
	mosek	15	22.5	2.6	3.4	-0.718	2E-8	2E-8	2E-8	144,197	575,059	1.73E+6	1.02E+7
	mix8	25	33.4	-	0.2	-0.816	6E-9	7E-9	7E-9	432,239	864,600	2.38E+6	1.58E+7
3DsqrexcLBM	sdpt3sql	30	2935.3	-	-	-125.524	7E-11	8E-8	5E-7	554,048	787,968	4.14E+6	2.39E+8
	sdpt3hsd	63	3917.7	-	-	-125.524	4E-6	6E-13	7E-13	554,048	787,968	4.80E+6	2.29E+8
	gurobi	35	633.0	24.2	19.6	-125.524	4E-7	2E-9	2E-10	491,841	580,609	3.15E+6	1.15E+8
	mosek	16	245.8	0.5	11.1	-125.520	1E-8	3E-8	3E-8	408,897	497,666	4.77E+6	1.14E+8
	mix8	21	1817.6	-	0.6	-125.524	3E-9	7E-9	7E-9	554,048	787,968	5.91E+6	2.29E+8
3DsqrexcUBM	sdpt3sql	44	1174.6	-	-	-148.410	2E-9	2E-8	1E-7	639,359	1,649,592	5.39E+6	2.26E+8
	sdpt3hsd	65	2522.9	-	-	-148.410	1E-5	5E-11	1E-10	639,359	1,649,592	7.13E+6	2.32E+8
	gurobi	37	1391.6	803.1	14.5	-148.410	8E-7	5E-9	2E-10	385,020	995,545	3.43E+6	9.79E+7
	mosek	18	197.8	0.9	4.8	-148.403	2E-8	3E-8	3E-8	239,652	850,178	2.74E+6	8.50E+7
	mix8	23	1242.8	-	0.5	-148.410	3E-9	5E-9	5E-9	639,359	1,649,592	4.49E+6	1.89E+8
3DsqrexcUB2M	sdpt3sql	100	2088.3	-	-	-134.865	9E-11	1E-3	2E-2	273,389	866,604	1.18E+7	1.14E+8
	sdpt3hsd	100	2146.6	-	-	-135.588	2E-7	2E-10	2E-8	273,389	866,604	1.27E+7	1.06E+8
	gurobi	19	151.7	4.7	6.7	-135.588	3E-6	7E-12	3E-10	174,134	582,879	9.05E+6	4.63E+7
	mosek	23	110.4	2.2	5.4	-135.584	1E-7	2E-8	2E-8	104,910	513,656	4.01E+6	3.74E+7
	mix8	60	15941.1	-	1.4	-135.578	6E-9	1E-8	1E-8	273,389	866,604	6.63E+6	3.45E+8
3DsqrfootLBM	sdpt3sql	36	804.5	-	-	-5.557	8E-11	1E-10	1E-8	363,904	492,672	2.68E+6	1.30E+8
	sdpt3hsd	77	1854.7	-	-	-5.557	1E-8	5E-14	2E-12	363,904	492,672	2.99E+6	1.31E+8
	gurobi	31	374.4	2.7	15.1	-5.557	1E-9	2E-10	7E-12	363,200	430,080	2.21E+6	8.73E+7
	mosek	21	375.3	1.9	12.9	-5.557	1E-8	3E-8	3E-8	261,560	389,881	5.02E+6	1.08E+8
	mix8	26	956.2	-	0.3	-5.557	5E-10	5E-9	5E-9	363,904	492,672	4.25E+6	1.24E+8
3DsqrfootUBM	sdpt3sql	37	741.8	-	-	-6.112	2E-14	7E-11	8E-9	289,728	856,320	2.88E+6	1.23E+8
	sdpt3hsd	56	1246.4	-	-	-6.112	7E-9	9E-11	2E-9	289,728	856,320	3.43E+6	1.23E+8
	gurobi	33	351.9	2.1	10.9	-6.112	2E-10	5E-11	9E-12	285,120	738,624	2.34E+6	7.47E+7
	mosek	21	179.2	2.0	7.1	-6.112	8E-9	2E-8	2E-8	91,185	648,546	2.51E+6	5.67E+7
	mix8	23	721.3	-	0.2	-6.112	4E-10	4E-9	4E-9	289,728	856,320	2.83E+6	1.09E+8
3DsqrfootUB2M	sdpt3sql	47	348.2	-	-	-6.048	2E-10	7E-11	8E-9	132,149	502,668	8.24E+6	4.10E+7
	sdpt3hsd	89	738.9	-	-	-6.048	7E-9	6E-11	6E-9	132,149	502,668	8.57E+6	4.12E+7
	gurobi	67	323.8	3.4	5.1	-6.048	2E-9	4E-11	8E-12	128,794	432,258	6.56E+6	3.43E+7
	mosek	19	60.7	1.3	3.0	-6.048	1E-8	3E-8	3E-8	65,970	430,875	2.58E+6	2.72E+7
	mix8	33	293.4	-	0.2	-6.048	5E-9	6E-9	6E-9	132,149	502,668	4.70E+6	4.14E+7
3DtunheadLBM	sdpt3sql	67	2321.2	-	-	-22.736	9E-10	7E-9	2E-7	587,983	774,304	5.42E+6	2.00E+8
	sdpt3hsd	41	1582.4	-	-	-22.735	5E-6	1E-5	2E-4	587,983	774,304	5.92E+6	2.03E+8
	gurobi	87	1536.9	4.9	22.5	-22.736	1E-5	2E-6	2E-11	583,424	670,209	4.38E+6	1.30E+8
	mosek	28	414.5	1.3	12.8	-22.735	1E-7	2E-7	2E-6	486,495	573,281	5.77E+6	1.18E+8
	mix8	0	70.1	-	0.5	0.000	2E-1	1E+0	1E+0	587,983	774,304	7.00E+6	2.03E+8
3DtunheadUBM	sdpt3sql	73	1949.8	-	-	-32.297	9E-9	2E-9	7E-8	454,879	1,342,528	5.73E+6	1.69E+8
	sdpt3hsd	53	1599.3	-	-	-32.296	7E-6	9E-6	2E-4	454,879	1,342,528	6.62E+6	1.72E+8
	gurobi	30	508.7	3.6	16.3	-32.297	2E-6	2E-11	5E-11	445,280	1,152,081	4.53E+6	1.08E+8
	mosek	20	217.1	1.0	5.6	-32.293	3E-8	3E-8	3E-8	274,032	980,834	2.96E+6	8.98E+7
	mix8	48	1925.7	-	0.3	-32.297	1E-8	3E-11	3E-11	454,879	1,342,528	4.43E+6	1.59E+8

Table 5. Available solver performance on the large size problems.

Problem	Solver	nit	t_T	t_P	t_O	pobj	pfeas	dfeas	gap	m	n	nnz(A)	nnz(L)
2DfootingLBL	gurobi	144	1875.4	8.4	19.2	-14.834	5E-4	7E-8	2E-9	1,870,400	2,104,900	1.19E+7	9.67E+7
	mosek	16	148.3	1.3	38.9	-14.823	8E-9	5E-9	5E-9	1,870,680	2,105,181	1.12E+7	8.55E+7
2DfootingUBL	gurobi	47	357.5	8.1	6.8	-14.877	2E-11	6E-14	5E-12	1,402,240	2,805,740	1.19E+7	7.28E+7
	mosek	17	108.3	1.9	24.3	-14.868	4E-10	6E-9	3E-9	1,401,680	2,805,180	7.70E+6	6.53E+7
2DtunnelLBL	gurobi	70	284.6	3.5	4.3	-0.801	1E-4	4E-11	4E-10	844,039	920,600	4.37E+6	3.03E+7
	mosek	15	57.8	3.5	13.2	-0.480	5E-8	5E-8	5E-8	513,831	820,353	5.19E+6	2.81E+7
2DtunnelUBL	gurobi	45	145.5	4.9	3.9	-0.813	3E-10	1E-13	1E-12	766,139	1,226,060	4.44E+6	2.63E+7
	mosek	15	44.8	4.6	6.9	-0.481	4E-8	3E-8	3E-8	301,965	1,068,447	3.04E+6	1.99E+7
3DsqrexcLBL	gurobi	40	1689.1	93.9	35.9	-123.875	3E-6	7E-7	3E-10	918,352	1,085,953	5.89E+6	2.39E+8
	mosek	20	785.3	1.0	22.7	-123.869	1E-8	3E-8	3E-8	763,216	930,818	8.92E+6	2.37E+8
3DsqrexcUBL	gurobi	53	8741.4	5199.0	33.8	-144.453	4E-6	3E-11	6E-11	917,952	2,371,969	8.20E+6	3.29E+8
	mosek	17	1108.8	2.2	14.4	-144.429	3E-8	4E-8	4E-8	573,504	2,027,522	6.60E+6	2.93E+8
3DsqrexcUB2L	gurobi	20	567.2	10.4	34.0	-134.449	1E-7	7E-11	6E-11	411,026	1,380,371	2.17E+7	1.45E+8
	mosek	24	523.2	5.2	14.0	-134.439	2E-7	3E-8	3E-8	246,277	1,215,623	9.60E+6	1.28E+8
3DsqrfootLBL	gurobi	46	5916.4	6.8	48.8	-5.629	3E-10	2E-11	4E-12	1,225,584	1,451,520	7.49E+6	4.93E+8
	mosek	20	3432.3	7.6	51.4	-5.628	2E-8	3E-8	3E-8	914,109	1,347,406	1.58E+7	5.64E+8
3DsqrfootUBL	gurobi	33	3378.4	6.4	35.7	-5.991	5E-10	1E-10	1E-12	969,840	2,508,624	7.97E+6	4.18E+8
	mosek	23	2030.6	7.7	27.3	-5.991	1E-8	3E-8	3E-8	394,485	2,287,918	7.93E+6	3.36E+8
3DsqrfootUB2L	gurobi	65	2435.6	10.7	35.5	-5.949	1E-9	1E-11	4E-12	432,446	1,456,514	2.24E+7	1.86E+8
	mosek	18	556.1	4.8	12.6	-5.949	1E-8	3E-8	3E-8	221,686	1,453,115	8.92E+6	1.53E+8
3DtunheadLBL	gurobi	51	8512.7	22.6	93.3	-22.752	4E-5	3E-9	1E-10	1,987,776	2,282,113	1.50E+7	6.97E+8
	mosek	22	3482.4	4.4	56.6	-22.737	8E-6	1E-6	1E-5	1,658,879	1,953,217	1.98E+7	6.60E+8
3DtunheadUBL	gurobi	52	7787.1	11.5	58.7	-30.780	8E-6	2E-11	8E-12	1,526,976	3,946,393	1.56E+7	6.17E+8
	mosek	19	2363.7	3.3	25.0	-30.768	3E-8	3E-8	3E-8	948,024	3,367,442	1.03E+7	5.24E+8

Table 6. The ratio of the number of non-zeros in the factorisation used to determine the search direction to the number of non-zeros in the constraint matrix as reported by `MOSEK`.

		min	**avg**	**max**
2D	**Small**	4	6	7
	Medium	5	6	8
	Large	5	7	8
3D without UB2	**Small**	15	17	22
	Medium	20	25	31
	Large	27	39	51
3D	**Small**	6	14	22
	Medium	9	21	31
	Large	13	33	51
UB2 only	**Small**	6	7	7
	Medium	9	10	11
	Large	13	15	17

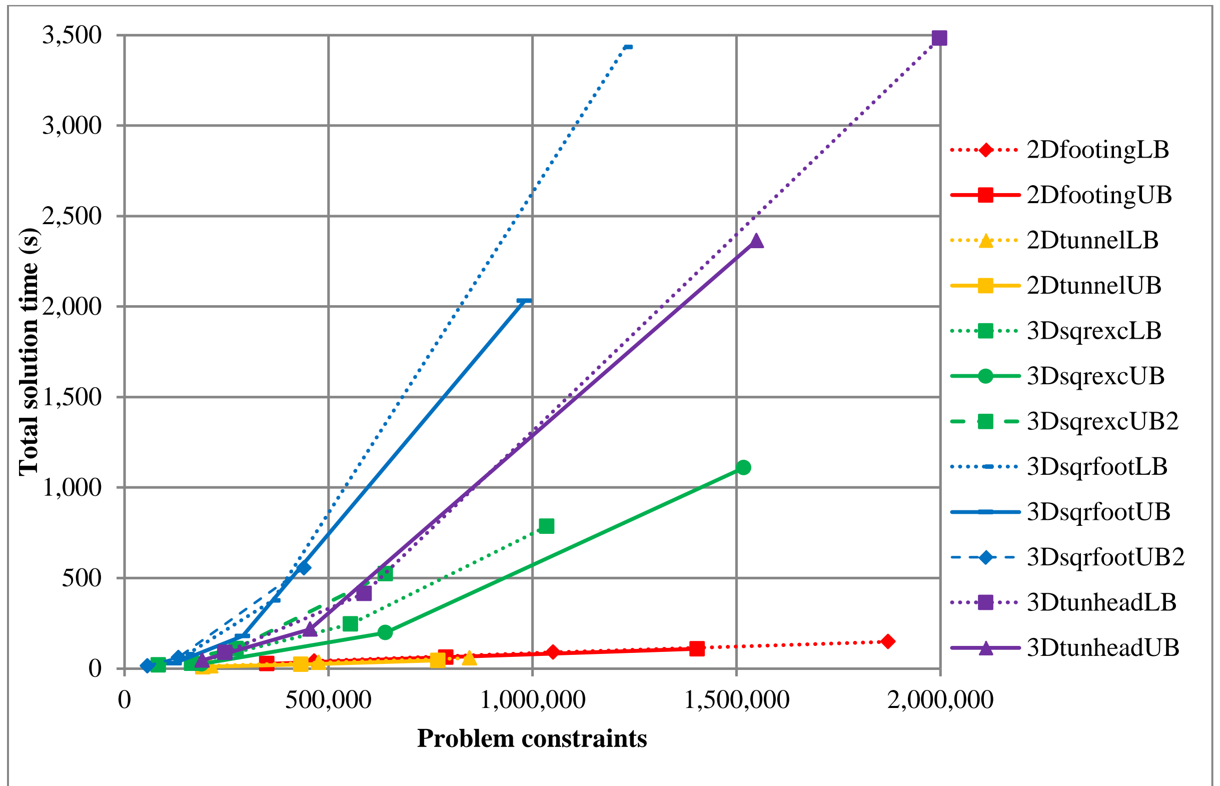


Figure 23. Total solution time vs number of problem constraints for `MOSEK`*. Note that the number of problem constraints is that of the original problem formulation before any presolving takes place.*

3.4 Improving on the basic IPM implementation

While `Mix8` performed well on the two-dimensional problems in the test set, its relative performance on the three-dimensional problems was significantly less satisfactory. In seeking to improve the performance of `Mix8`, major modifications that are likely to improve its performance include the method used to obtain the search direction at each iteration of the IPM, how the free variables in the problem are treated by the IPM, and exploiting any structure in the original problem formulation that may be beneficial such as fixed variables and dense columns. If a factorisation routine is used to compute the search direction, the choice of factorisation method and the sparsity-preserving permutation used can have an order of magnitude difference in performance in terms of runtime and a considerable difference in the amount of required memory. The common high-performance factorisation approaches applicable in this case are the multifrontal and left-looking supernodal methods, while the two common permutations are the approximate minimum degree and nested dissection (or graph partition) orderings.

Below, the performance of Mix8 is reported comparing the improvements possible with different reorderings, how free variables are represented, and when some types of problem structure are exploited.

3.4.1 Choice of direct method

Most described IPMs report that they use supernodal Cholesky factorisations to determine the search direction at each step of the IPM. Because of the increasingly ill-conditioned Schur complements, the use of a Cholesky solver generally requires some way of dealing with non-positive pivots before taking the square root [40]. For the tests here, a left-looking supernodal Cholesky routine based on CHOLMOD [102] with a modified BLAS dpotf2 subroutine that substitutes a large value (10^{32} is used) for any diagonal entry that is less than or equal to zero, although a more elegant approach is described by Stewart [205] for non-singular systems.

Table 7 shows the results on the small problem set and Table 8 the results for the medium problem set. The non-zeros in the factor reported for the supernodal Cholesky factorisation reflect the total floating point storage required to hold the factor; each supernode is held in a rectangle, so there are a number of unused entries in the diagonal block, as well as the zeros introduced through supernode amalgamation. This leads to a higher effective non-zero count on each problem even though the same ordering is used. In the small problem set, the supernodal solver reports between an additional two and 14 million non-zeros with an average of six million additional non-zeros over the multifrontal solver. This increases to an average of 15 million additional non-zeros in the medium problem set, although, in the case of *3DsqrexcUB2M*, the multifrontal solver reports 345 million non-zeros while the supernodal solver reports just 154 million. This discrepancy is caused by the dynamic reordering in the multifrontal solver that provides the ability to solve indefinite problems when such situations are encountered in the numerical phase. The runtime, however, is reduced on all the problems solved as shown in Figure 24 and Figure 25, with the exception of the lower bound on the tunnel heading where the multifrontal solver reports a singular system and the IPM stops before reaching convergence. While there is no major improvement for the two-dimensional problems, most of the three-dimensional problems benefit significantly from the use of the supernodal solver over the multifrontal solver. This is

especially so for the *3DsqrexcUB2* problems, with almost a $6\times$ speedup per iteration on the small problem, and a $13\times$ speedup on the medium problem. The diagonal perturbation of non-positive pivots has allowed some progress to be made towards solving the *3DtunheadLB* problems. The average time per iteration in the small problem set was reduced from 4.0s to 2.4s, and from 48.7s to 18.0s on the medium problem set showing an increasing improvement as the linear system becomes larger. While the iteration counts are, on average, almost identical at 27.9 and 27.8 iterations per problem in the small test set and 29.2 and 29.6 in the medium for the multifrontal and supernodal solvers, respectively, only two of the 12 problems in the small set and one of the 12 in the medium had the same iteration count. This is due to the small differences in the computed search direction between the two solvers.

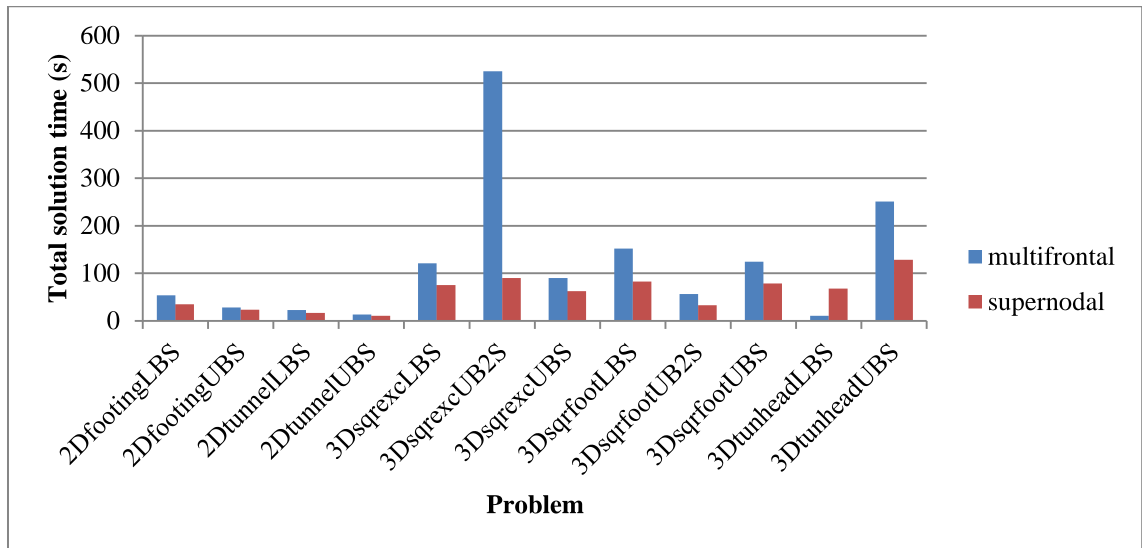


Figure 24. Comparison of the total solution time on the small problem set between the multifrontal and supernodal direct solvers.

Table 7. Comparison of multifrontal and supernodal factorisation on small problem set.

Problem	Solver	nit	t_T	pobj	pfeas	dfeas	gap	m	n	nnz(A)	nnz(L)	t_T/nit
2DfootingLBS	multifrontal	29	54.0	-14.832	2E-8	2E-8	2E-8	465,360	524,090	2.79E+6	2.75E+7	1.9
	supernodal	27	35.1	-14.833	1E-8	2E-8	2E-8	465,360	524,090	2.79E+6	3.42E+7	1.3
2DfootingUBS	multifrontal	22	28.2	-14.916	8E-9	7E-9	7E-9	349,020	698,950	1.92E+6	1.88E+7	1.3
	supernodal	24	23.6	-14.916	7E-9	6E-9	6E-9	349,020	698,950	1.92E+6	2.35E+7	1.0
2DtunnelLBS	multifrontal	43	23.2	-0.790	6E-9	8E-9	8E-9	211,799	288,960	1.32E+6	7.84E+6	0.5
	supernodal	40	17.0	-0.790	6E-9	8E-9	8E-9	211,799	288,960	1.32E+6	1.03E+7	0.4
2DtunnelUBS	multifrontal	26	13.6	-0.824	5E-9	6E-9	6E-9	192,159	384,400	1.06E+6	6.25E+6	0.5
	supernodal	24	10.8	-0.824	6E-9	7E-9	7E-9	192,159	384,400	1.06E+6	8.18E+6	0.4
3DsqrexcLBS	multifrontal	19	121.1	-121.988	7E-10	7E-9	7E-9	164,355	233,472	1.74E+6	3.60E+7	6.4
	supernodal	19	75.6	-121.988	1E-9	9E-9	9E-9	164,355	233,472	1.74E+6	4.83E+7	4.0
3DsqrexcUB2S	multifrontal	49	525.3	-138.242	2E-9	9E-9	9E-9	83,189	260,908	1.99E+6	3.83E+7	10.7
	supernodal	51	90.3	-138.244	2E-9	6E-9	6E-9	83,189	260,908	1.99E+6	2.38E+7	1.8
3DsqrexcUBS	multifrontal	23	89.9	-155.149	1E-9	9E-9	9E-9	188,927	487,392	1.32E+6	3.04E+7	3.9
	supernodal	23	62.9	-155.149	1E-9	1E-8	1E-8	188,927	487,392	1.32E+6	4.11E+7	2.7
3DsqrfootLBS	multifrontal	23	152.0	-5.492	5E-10	5E-9	5E-9	153,648	208,008	1.79E+6	3.55E+7	6.6
	supernodal	24	82.5	-5.492	5E-10	5E-9	5E-9	153,648	208,008	1.79E+6	4.40E+7	3.4
3DsqrfootUB2S	multifrontal	32	56.9	-6.170	3E-9	4E-9	4E-9	56,549	213,708	2.00E+6	1.16E+7	1.8
	supernodal	30	33.1	-6.170	3E-9	5E-9	5E-9	56,549	213,708	2.00E+6	1.48E+7	1.1
3DsqrfootUBS	multifrontal	25	124.5	-6.234	3E-10	3E-9	3E-9	121,932	360,720	1.19E+6	3.08E+7	5.0
	supernodal	24	78.5	-6.234	4E-10	4E-9	4E-9	121,932	360,720	1.19E+6	4.40E+7	3.3
3DtunheadLBS	multifrontal	0	11.1	0.000	3E-1	1E+0	1E+0	246,167	324,792	2.91E+6	5.55E+7	-
	supernodal	12	67.9	-21.229	7E-5	2E-4	2E-4	246,167	324,792	2.91E+6	6.89E+7	5.7
3DtunheadUBS	multifrontal	44	251.1	-33.432	7E-9	5E-11	5E-11	190,151	561,528	1.85E+6	4.18E+7	5.7
	supernodal	36	128.4	-33.432	3E-8	4E-9	4E-9	190,151	561,528	1.85E+6	5.20E+7	3.6

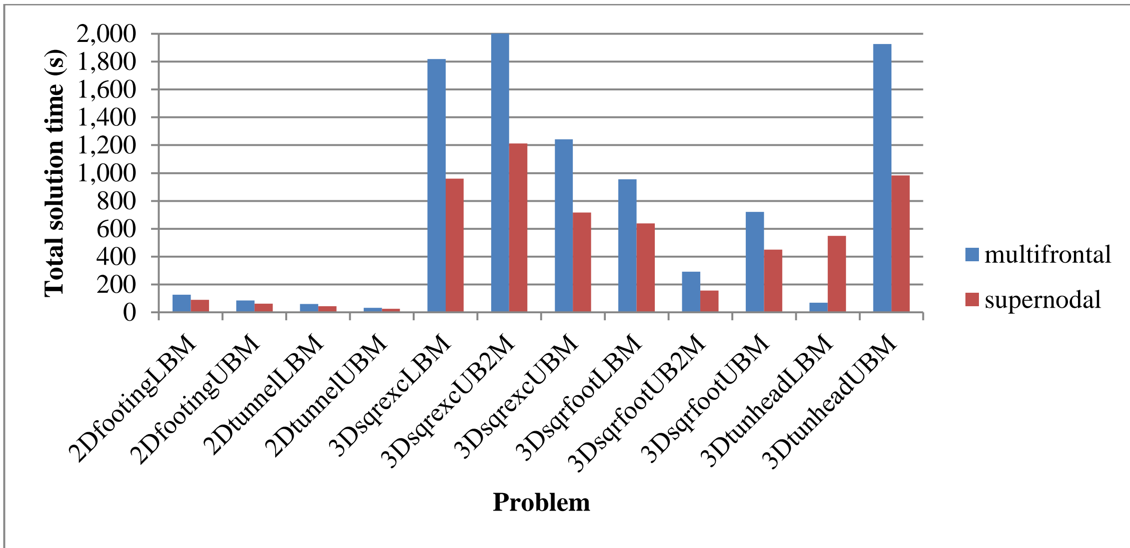


Figure 25. Comparison of the total solution time on the medium problem set between the multifrontal and supernodal direct solvers.

Table 8. Comparison of multifrontal and supernodal factorisation on medium problem set.

Problem	Solver	nit	t_T	pobj	pfeas	dfeas	gap	m	n	nnz(A)	nnz(L)	t_T/nit
2DfootingLBM	multifrontal	24	127.3	-14.832	1E-8	2E-8	2E-8	1,050,840	1,183,035	6.30E+6	6.86E+7	5.3
	supernodal	25	90.0	-14.832	4E-8	1E-8	1E-8	1,050,840	1,183,035	6.30E+6	8.44E+7	3.6
2DfootingUBM	multifrontal	23	86.2	-14.885	9E-9	8E-9	8E-9	788,130	1,577,625	4.33E+6	5.13E+7	3.7
	supernodal	23	63.3	-14.890	8E-9	7E-9	7E-9	788,130	1,577,625	4.33E+6	6.28E+7	2.8
2DtunnelLBM	multifrontal	44	60.5	-0.798	5E-9	7E-9	7E-9	476,099	649,440	2.98E+6	1.90E+7	1.4
	supernodal	41	43.6	-0.798	7E-9	1E-8	1E-8	476,099	649,440	2.98E+6	2.51E+7	1.1
2DtunnelUBM	multifrontal	25	33.4	-0.816	6E-9	7E-9	7E-9	432,239	864,600	2.38E+6	1.58E+7	1.3
	supernodal	24	26.9	-0.816	5E-9	6E-9	6E-9	432,239	864,600	2.38E+6	2.03E+7	1.1
3DsqrexcLBM	multifrontal	21	1817.6	-125.524	3E-9	7E-9	7E-9	554,048	787,968	5.91E+6	2.29E+8	86.6
	supernodal	19	961.1	-125.524	9E-10	8E-9	8E-9	554,048	787,968	5.91E+6	3.09E+8	50.6
3DsqrexcUB2M	multifrontal	60	15941.1	-135.578	6E-9	1E-8	1E-8	273,389	866,604	6.63E+6	3.45E+8	265.7
	supernodal	61	1211.7	-135.578	8E-9	2E-8	2E-8	273,389	866,604	6.63E+6	1.54E+8	19.9
3DsqrexcUBM	multifrontal	23	1242.8	-148.410	3E-9	5E-9	5E-9	639,359	1,649,592	4.49E+6	1.89E+8	54.0
	supernodal	22	717.8	-148.409	2E-9	2E-8	2E-8	639,359	1,649,592	4.49E+6	2.50E+8	32.6
3DsqrfootLBM	multifrontal	26	956.2	-5.557	5E-10	5E-9	5E-9	363,904	492,672	4.25E+6	1.24E+8	36.8
	supernodal	27	639.3	-5.557	9E-10	9E-9	9E-9	363,904	492,672	4.25E+6	1.77E+8	23.7
3DsqrfootUB2M	multifrontal	33	293.4	-6.048	5E-9	6E-9	6E-9	132,149	502,668	4.70E+6	4.14E+7	8.9
	supernodal	30	156.9	-6.048	4E-9	7E-9	7E-9	132,149	502,668	4.70E+6	5.24E+7	5.2
3DsqrfootUBM	multifrontal	23	721.3	-6.112	4E-10	4E-9	4E-9	289,728	856,320	2.83E+6	1.09E+8	31.4
	supernodal	25	451.2	-6.112	2E-10	2E-9	2E-9	289,728	856,320	2.83E+6	1.46E+8	18.0
3DtunheadLBM	multifrontal	0	70.1	0.000	2E-1	1E+0	1E+0	587,983	774,304	7.00E+6	2.03E+8	-
	supernodal	16	550.0	-22.334	2E-4	9E-5	1E-4	587,983	774,304	7.00E+6	2.55E+8	34.4
3DtunheadUBM	multifrontal	48	1925.7	-32.297	1E-8	3E-11	3E-11	454,879	1,342,528	4.43E+6	1.59E+8	40.1
	supernodal	42	983.6	-32.297	8E-7	1E-9	1E-9	454,879	1,342,528	4.43E+6	2.04E+8	23.4

3.4.2 Matrix reordering

While the AMD ordering performs satisfactorily for the two dimensional problems, the size of the factorisations in the three dimensional problems grows significantly. The graph partitioning ordering is used in `MOSEK` for all of the three-dimensional problems because of the fewer non-zeros in the factorisation. The two orderings were compared using the supernodal Cholesky solver and split free variables. The AMD ordering was computed by HSL `MC50` while the ND ordering is computed by `METIS` [116]. The total solution time with the two solvers is shown in Figure 26 with the non-zero counts in Figure 27, and complete results are shown in Table 9.

The number of iterations was very similar, except for the small and medium-sized lower bound tunnel heading problems, which showed large differences, although in opposite directions. An obvious difference, though, can be seen between the two-dimensional and three-dimensional problems in Figure 26, with the two-dimensional problems barely showing at the appropriate scale for the three-dimensional problems. The

difference in the solution time is greater than that between the non-zero counts shown in Figure 27 because the required floating point operations is approximately the sum of the square of the column (or row) counts, resulting in noticeably larger runtime differences than storage differences. Specifically, the number of non-zeros in the Cholesky factor ranged from 15% more to 180% more with the AMD ordering. On average, the AMD ordering led to over 60% larger factors than the ND ordering. For the large three dimensional problems, the range was 65% to 180% more non-zeros, with an average of 110%. This led to the IPM with the AMD ordering running 1.0 to 7.7× times slower, with an average of 2.5× slower, than the ND ordering. For the large three dimensional problems, the minimum speedup was 2.4× and the average 4.7×. A performance profile of the results on the test is shown in Figure 28.While not all problems were solved to the desired tolerance, the only problems that did not finish with a primal infeasibility of less than 10^{-6} were the lower bound tunnel headings.

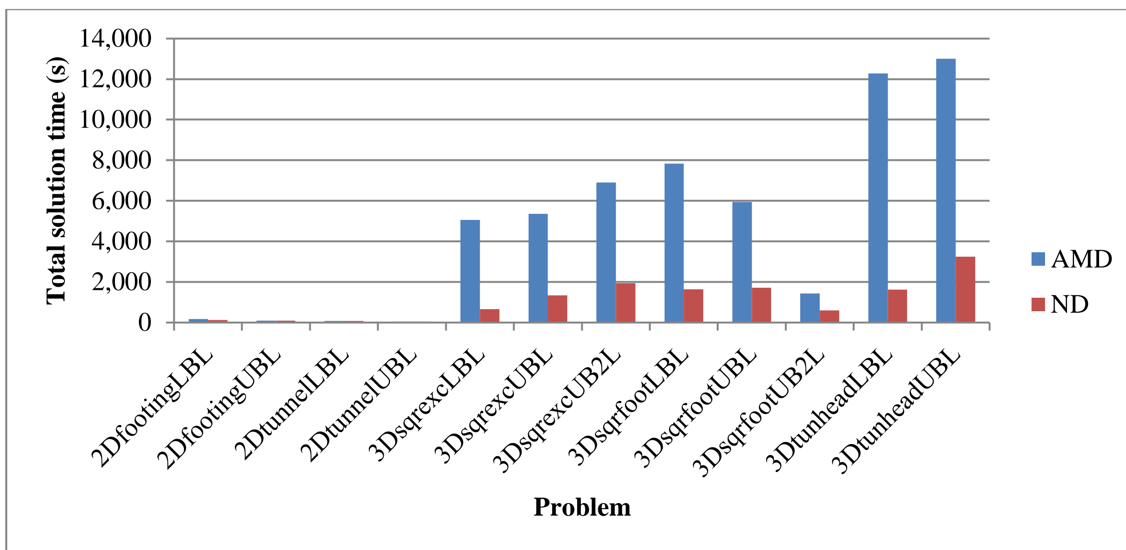


Figure 26. Comparison of the total solution time on the large problem set between the AMD and ND orderings.

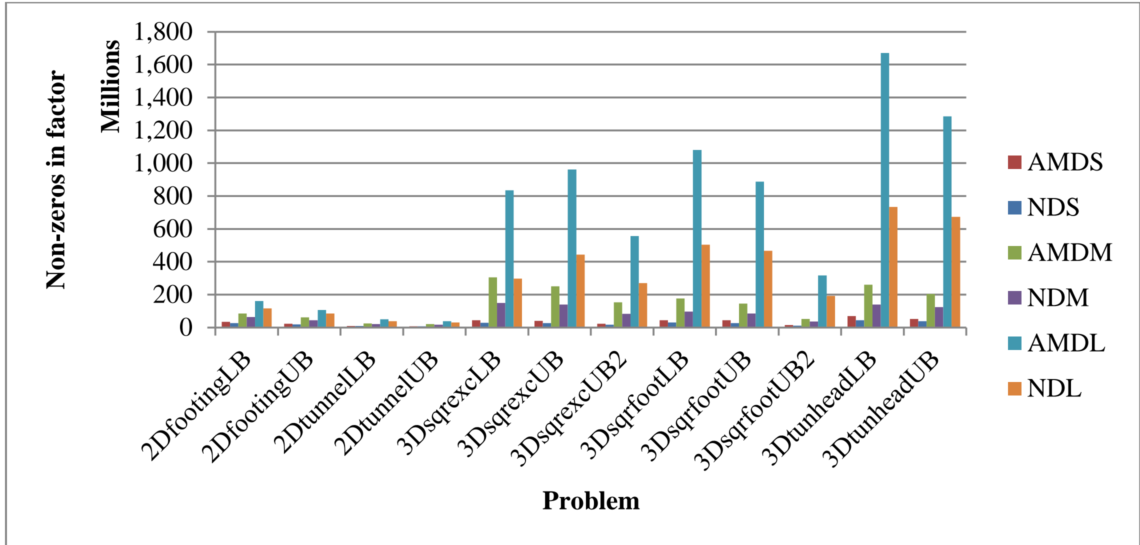


Figure 27. Comparison of the number of non-zeros between the AMD and ND orderings. The last letter of the ordering indicates whether the problem is from the small (S), medium (M), or large (L) test set.

Table 9. Comparison of AMD and ND ordering with supernodal Cholesky solver.

		S			M			L		
Problem	**Ordering**	**nit**	**t_T**	**nnz(L)**	**nit**	**t_T**	**nnz(L)**	**nit**	**t_T**	**nnz(L)**
2DfootingLB	AMD	27	35.3	34,176,470	27	98.3	84,432,074	23	169.2	161,724,772
	ND	28	30.7	26,597,626	26	71.4	64,196,268	24	123.8	116,193,876
2DfootingUB	AMD	23	23.1	23,492,620	22	61.3	62,771,568	19	91.1	106,148,904
	ND	23	21.1	18,859,208	22	49.0	44,888,168	19	84.5	85,687,344
2DtunnelLB	AMD	36	14.5	10,262,970	41	41.0	25,075,799	41	82.5	50,116,055
	ND	36	14.5	8,764,682	41	39.0	20,923,476	41	73.8	38,946,661
2DtunnelUB	AMD	23	9.6	8,182,072	22	22.9	20,276,016	22	42.7	38,467,957
	ND	23	9.7	7,017,616	22	21.8	16,556,341	22	39.9	30,413,594
3DsqrexcLB	AMD	19	64.9	44,426,757	20	1013.1	305,968,704	24	5058.0	835,631,569
	ND	19	28.0	29,637,786	20	226.5	148,895,546	24	657.2	297,682,560
3DsqrexcUB	AMD	23	59.3	41,126,846	22	710.4	250,321,643	23	5350.1	961,216,058
	ND	22	32.7	26,818,343	22	258.8	139,726,394	23	1332.3	444,358,967
3DsqrexcUB2	AMD	45	80.0	23,802,678	61	1194.5	153,633,398	55	6900.6	556,984,986
	ND	45	45.4	16,374,115	61	414.8	83,756,367	61	1942.3	270,925,139
3DsqrfootLB	AMD	22	76.6	44,029,048	26	608.8	177,254,045	24	7833.7	1,079,674,994
	ND	22	34.9	31,095,777	26	182.3	96,346,187	24	1636.0	503,500,942
3DsqrfootUB	AMD	22	71.9	44,031,780	23	405.8	145,811,716	25	5947.1	886,908,573
	ND	22	32.3	26,597,381	23	153.1	85,655,921	25	1721.1	467,076,084
3DsqrfootUB2	AMD	24	26.7	14,846,231	25	128.9	52,368,502	25	1435.5	317,849,414
	ND	24	19.0	11,543,046	25	71.1	36,142,232	25	598.9	192,568,328
3DtunheadLB	AMD	27	149.9	69,603,801	24	837.7	260,683,703	22	12277.3	1,670,110,569
	ND	15	32.4	45,049,383	49	482.3	140,127,684	18	1617.7	733,578,413
3DtunheadUB	AMD	30	110.1	51,998,007	31	728.2	203,885,049	39	12998.9	1,284,339,958
	ND	30	63.5	38,770,079	31	293.4	123,913,631	35	3249.8	673,234,352

Performance profiles will be used extensively in the following sections to visualise the relative effectiveness of different solution approaches. The performance profile was presented by Dolan and Moré [206] and show, for each solver, what percentage of problems is solved within α of the best performing solver, from 0% to 100%. In this case, the profile shows that the ND ordering provides a significant performance benefit over the AMD ordering, easily outweighing the greater cost of computing the ND ordering. The profile indicates that two thirds of the problems solved using the AMD ordering require at least $2.7\times$ the runtime when using the ND ordering.

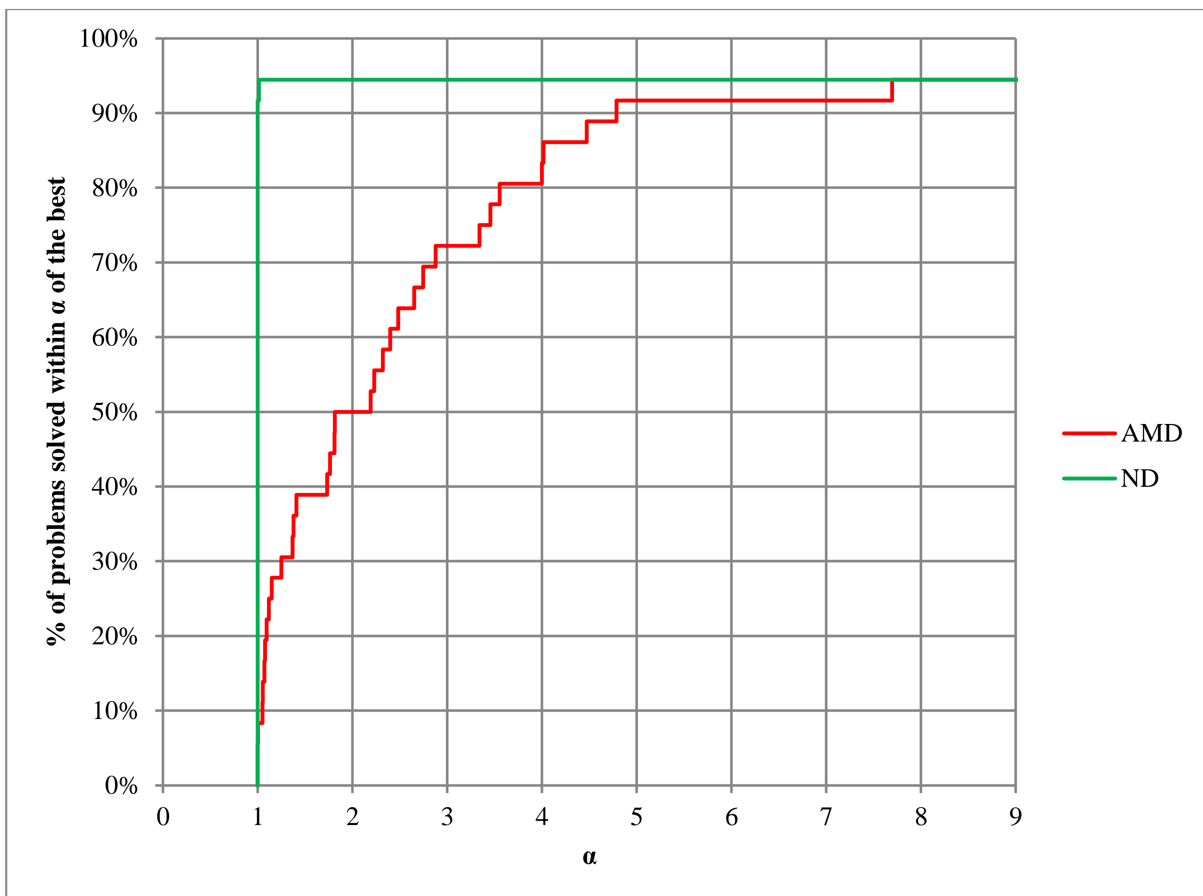


Figure 28. Performance profile of IPM runtime by orderings. Both methods used supernodal Cholesky, no presolve, and free variables were split.

3.4.3 Dealing with free variables

As discussed above in Section 1.3.3, there are various ways of dealing with free variables. It cannot be expected that these methods will perform the same or even similarly due the quite different way the problem is formulated. It is thus reasonable to compare the performance of the different approaches. In the following, free variables

are either split as the difference between two linear variables, embedded in a second-order cone, or the diagonal entries of the $(1,1)$ block in the augmented equations associated with the free variables are perturbed with a small value (10^{-10} is used) to make the block non-singular. The cone embedding uses a single cone per variable, which is not the only way. For example, `SeDuMi` embeds all free variables in a single cone, but this means the $(1,1)$ block in the augmented equations has a very large dense diagonal block when there is many free variables in the problem, and increases the density of the coefficient system in the Schur complement equation. Runtime comparisons are provided in Figure 29 and Figure 30, with complete results in Table 10, and the performance profile in Figure 31.

Interestingly, Figure 29 shows quite similar performance in terms of the total solution time between the three methods across the small problem set, although the differences are magnified in the large problems shown in Figure 30. The regularised approach does not work for the *3DtunheadLB* problems and has trouble on the medium and large versions of *3DsqrexcLB*. Moreover, the regularised approach appears to struggle on the *3DsqrexcUB* problems also, where approximately one third of the variables are free variables, compared with around one sixth for the other two three-dimensional upper bound problems. The quadratic upper bound formulation of the square excavation does not present the same difficulty, however, with the regularised approach performing significantly better than splitting the variables or embedding them in a quadratic cone. Moreover, the regularised approach often outperforms the other two approaches when there is a small proportion of free variables, suggesting that it is likely to be the preferred approach if the number of free variables can be kept low or modified to be so.

Embedding the free variables in a quadratic cone exhibits generally increases the required runtime and iterations taken to converge over the splitting of variables, with an average runtime average runtime of 1231.9s and 31.1 iterations on the large problems compared to 1089.8s and 28.4 iterations when splitting the free variables on the large problems. Table 10 shows the complete results across the three approaches. The columns are as previously described and $\phi = \max\left(pinf, dinf, relgap\right)$.

The performance profile in Figure 31 supports the notion that the regularised approach is generally the fastest, but struggles on the problems with a high proportion of free variables. Splitting free variables performs satisfactorily compared with the regularisation approach, and consistently outperforms the quadratic cone embedding approach.

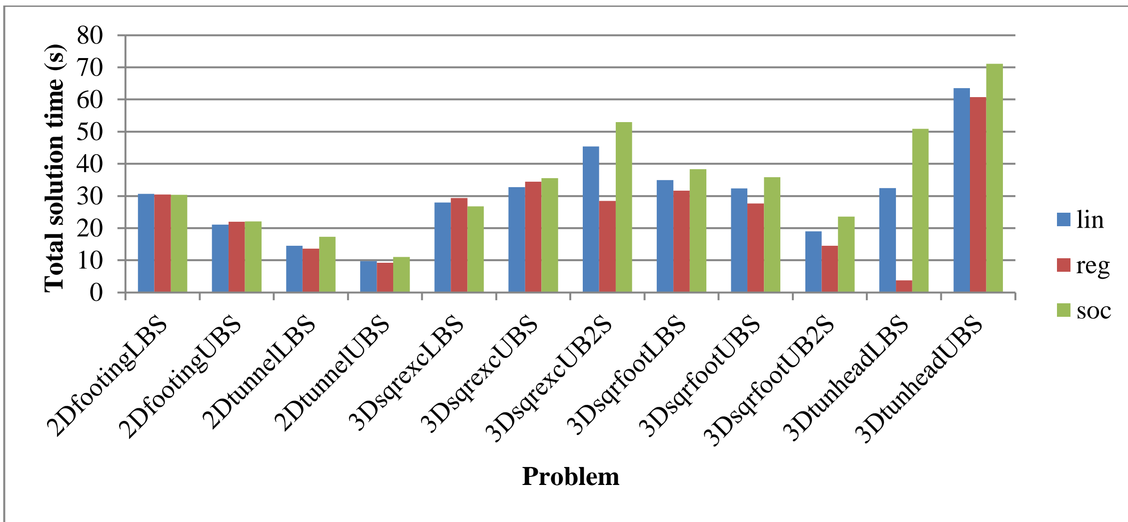


Figure 29. Comparison of the total solution time on the small problem set between the three approaches to handling free variables considered.

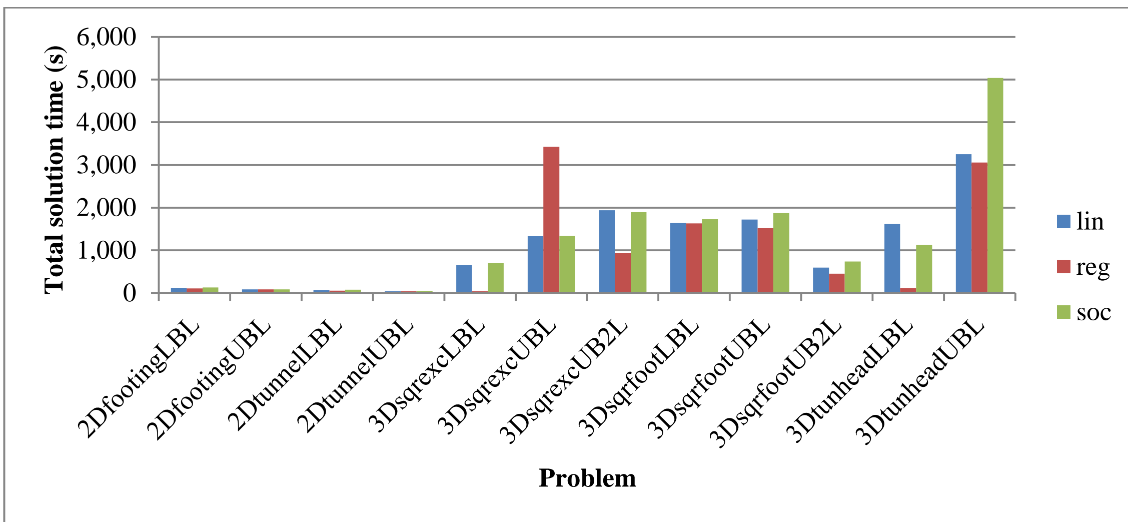


Figure 30. Comparison of the total solution time on the large problem set between the three approaches to handling free variables considered.

Table 10. Comparison of approaches to handle free variables. The three methods compared are the split into two linear variables (LIN), regularisation (REG), and embedding within a second-order cone (SOC).

		S			M			L		
Problem	**Method**	**nit**	**t_T**	**ϕ**	**nit**	**t_T**	**ϕ**	**nit**	**t_T**	**ϕ**
2DfootingLB	lin	28	30.7	1E-08	26	71.4	3E-08	24	123.8	3E-08
	reg	28	30.5	1E-08	24	67.0	2E-08	21	110.4	3E-08
	soc	28	30.4	1E-08	25	69.3	5E-08	25	129.0	4E-08
2DfootingUB	lin	23	21.1	7E-09	22	49.0	8E-09	19	84.5	7E-09
	reg	24	22.0	7E-09	23	50.7	8E-09	19	84.0	8E-09
	soc	24	22.1	7E-09	23	51.0	8E-09	19	84.3	8E-09
2DtunnelLB	lin	36	14.5	9E-09	41	39.0	8E-09	41	73.8	8E-09
	reg	36	13.6	2E-07	33	30.1	2E-06	31	52.8	6E-05
	soc	40	17.3	8E-09	41	42.3	1E-08	41	79.1	8E-09
2DtunnelUB	lin	23	9.7	6E-09	22	21.8	9E-09	22	39.9	7E-09
	reg	24	9.2	7E-09	19	17.6	1E-07	25	41.7	6E-09
	soc	24	11.0	7E-09	24	25.7	6E-09	24	47.5	7E-09
3DsqrexcLB	lin	19	28.0	6E-09	20	226.5	9E-09	24	657.2	8E-09
	reg	20	29.4	1E-08	1	16.6	1E+00	1	38.1	1E+00
	soc	19	26.8	9E-09	19	213.9	8E-09	26	702.2	5E-09
3DsqrexcUB	lin	22	32.7	9E-09	22	258.8	7E-09	23	1332.3	7E-09
	reg	24	34.4	5E-09	32	358.1	1E-07	61	3427.8	5E-06
	soc	23	35.5	1E-08	23	274.4	5E-09	24	1341.4	1E-07
3DsqrexcUB2	lin	45	45.4	6E-09	61	414.8	5E-08	61	1942.3	9E-08
	reg	30	28.5	2E-07	30	199.4	2E-07	30	935.7	1E-07
	soc	51	53.0	6E-09	61	425.6	2E-08	61	1897.7	4E-07
3DsqrfootLB	lin	22	34.9	8E-09	26	182.3	3E-09	24	1636.0	1E-08
	reg	20	31.7	9E-09	22	154.2	6E-09	24	1632.4	9E-09
	soc	24	38.3	5E-09	27	185.2	9E-09	26	1729.2	8E-09
3DsqrfootUB	lin	22	32.3	6E-09	23	153.1	5E-09	25	1721.1	6E-09
	reg	19	27.7	6E-09	20	132.3	3E-09	22	1521.6	4E-09
	soc	24	35.8	4E-09	25	168.5	2E-09	27	1872.2	5E-09
3DsqrfootUB2	lin	24	19.0	7E-09	25	71.1	8E-09	25	598.9	6E-09
	reg	19	14.5	4E-09	19	53.1	6E-09	19	451.4	3E-09
	soc	30	23.6	5E-09	30	88.3	7E-09	33	739.2	6E-09
3DtunheadLB	lin	15	32.4	6E-05	49	482.3	2E-05	18	1617.7	3E-04
	reg	1	3.8	1E+00	1	14.1	1E+00	1	114.5	1E+00
	soc	22	50.9	6E-05	25	245.5	6E-04	13	1126.6	8E-04
3DtunheadUB	lin	30	63.5	9E-09	31	293.4	1E-08	35	3249.8	2E-08
	reg	29	60.7	9E-09	22	198.2	1E-05	32	3056.8	8E-09
	soc	33	71.1	1E-08	41	394.5	2E-08	54	5035.2	5E-08

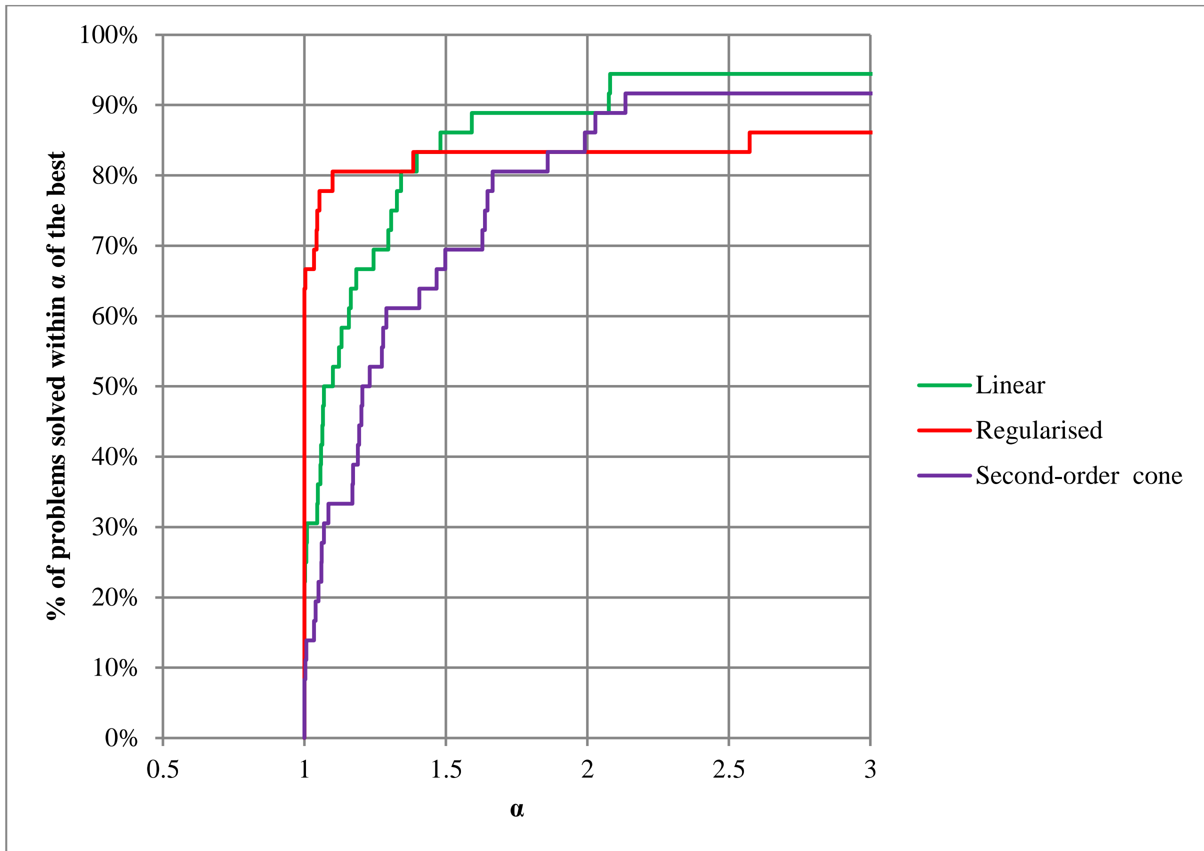


Figure 31. Performance profile of IPM runtime by free variable approach.

3.4.4 Presolving

An important phase in solving optimisation problems arises before the solver algorithm begins. This presolve phase can often reduce solve times considerably [207]. Thus, much effort (see, for example, [73], [208], [209]) has been spent seeking approaches to automatically improve the problem formulation before attempting to solve the mathematical program. As a result, a wide range of cheap heuristics have been developed that sometimes result in a significant improvement in computational speed. This presolve often involves manipulation of the upper and lower bounds on each of the linear variables looking for fixed and implied free variables, and the elimination of free variables. Unfortunately, much of the work on LP presolving does not carry over to second-order cone and semidefinite cone based programming because of the difference introduced by the conic constraints. However, performance improvements can still be gained by using an effective presolve procedure.

The three main components that benefit the solution of FELA problems are the elimination of free variables, the treatment of fixed variables subject to conic

constraints, and handling dense columns in the constraint matrix efficiently. While various other strategies may be employed, these do not appear to provide much payoff. A description of these presolve procedures is outlined below, including details on the efficient implementation of the methods.

3.4.4.1 Eliminating free variables

While most authors mention the difficulties posed by free variables in the FELA problem formulation, few provide beneficial methods of dealing with them. Makrodimopoulos and Martin [25] describe how the free variables may be eliminated from a lower bound formulation, but only achieve around 10% improvement in performance. This is likely to be a result of `MOSEK` being able to efficiently handle the free variables. In the following, an outline is given of the efficient elimination of some (possibly all) free variables from the problem before solving the optimisation problem using purely algebraic conditions. A direct comparison is then made between the solver performance with and without this presolve procedure. Note that a post-solve is also required if free variables are eliminated in order to return a solution suitable for interpretation of the results by the calling program. This matter involves a few cheap operations which are shown in Section 1.3.3.

The common method for eliminating free variables consists of iteratively checking for columns in the constraint matrix associated with a free variable that has only one non-zero in it; this is known as a column singleton. A column singleton associated with a free variable can be used as a pivot to eliminate one equation (the equation containing the non-zero in the singleton column) from the constraint matrix without causing any fill-in. Additionally, and perhaps more importantly with regards to the efficiency of the number of iterations of the IPM, the free variable is also eliminated from the problem. Other "tricks" involving concepts including doubleton equations (equations with just two non-zero coefficients), the removal of fixed variables, and sparsity increasing constraint equation manipulations [208], [209] can lead to columns associated with the free variables being reduced to singleton columns, allowing additional free variables to be eliminated from the problem without fill-in. Another approach used by some authors seeks to find a full rank row-set of as many free variables as possible on which it is possible to block-pivot without causing too much fill-in and eliminating as many free

variables as possible. Because it may not be possible to find a suitable sparsity-preserving and stable block pivot for all free variables, the partitioning leads to

$$\begin{bmatrix} \mathbf{E}_B & \mathbf{E}_{BR} & \mathbf{A}_N \\ \mathbf{E}_N & \mathbf{E}_{NR} & \mathbf{A}_N \end{bmatrix},$$

where the row set B contains the equations to be eliminated from the problem and the remaining rows are in the set N, $\mathbf{E}_B$ is the non-singular block pivot, the $\mathbf{E}_{*R}$ matrices contain the set of columns of the constraint matrix associated with free variables not being eliminated (possibly empty). This method was outlined by Mészáros [73], who used a Markowitz criterion with a maximum Markowitz count of four. This means that a free variable was only eliminated if one less than the number of entries in the pivot row, multiplied by one less than the number of entries in the pivot column, was less than or equal to four. Note that this approach does not consider the actual fill-in, but instead uses an upper bound on the fill-in as a heuristic. An alternative interpretation is that it minimises the number of updates to the active submatrix when the rank-one outer product update is made to the active submatrix upon eliminating the free variable. The use of linked lists provides a simple way to find the next pivot of lowest or low Markowitz-count subject to the stability threshold at each step [210]. The process continues until a suitable pivot can no longer be found.

This approach can be modified to use an approximate minimum local fill-count at each step, similar to that outlined for the reordering of the Schur complement system in [115] (except that the ordering is not computed *a priori* in contrast with the static ordering for the SPD system). A limited number of columns are searched in order of increasing column count [211] for a pivot satisfying the stability threshold $|a_{ij}| \geq \tau \times \max\left(|\mathbf{A}_j|\right)$ for some scalar $\tau \in [0,1]$ which produces a minimum fill score upon elimination of this pivot. The fill score is calculated as the amount of fill-in produced by using the pivot less the number of non-zeros remaining in the pivot column and pivot row in the active submatrix plus one (because the pivot occurs in both the row and the column). An efficient procedure for finding such a pivot checks for stability of the potential pivot, and then computes the amount of fill-in caused by the stable potential pivot. The fill-in can be computed by negating the row pointers in the CSR structure of the active

submatrix corresponding to a non-zero in the potential pivot column, and then comparing it with all the other columns containing an entry in the potential pivot row. In each of these columns, the amount of fill-in can be computed by scanning the column and keeping track of the number of common entries. The amount of fill-in in this column is the length of the potential pivot column less the number of common entries. At this point, it is also simple to check the number of entries that would exist in this column if the potential pivot were chosen. This is important for some problems if no method to handle "dense" columns (generally those columns which lead to high cost in forming and factorising the Schur complement system) in the constraint matrix is used and the Schur complement system is used to obtain the search direction. In such a case, it is possible to add an additional constraint on the pivot selection that limits the maximum growth of any column in the constraint matrix. Significant performance gains can be achieved by monitoring the progressive fill score, and stopping the fill-count computation as soon as it is greater than the maximum score allowed. After all of the columns have been checked, the fill score is then computed by subtracting the number of entries in the potential pivot row and potential pivot column and adding one. Note that if a potential pivot is found to be stable and exists on a singleton row or singleton column, it is chosen immediately, without further searching. After a pivot is chosen, the $\mathbf{LU}$ factors of $\mathbf{E}_B$ are updated, as is the active submatrix.

A less sophisticated but cheaper approach that avoids the use of the Markowitz table searches each free-variable column in some predetermined order, with a stable pivot chosen if its fill score is less than the maximum or it occurs on a singleton row or singleton column. To demonstrate the effectiveness of each approach, both the approximate minimum local fill-in and the static order minimum local fill-in schemes were used to eliminate as many free variables from each of the small test problems as possible, and then solved using `Mix8` with the supernodal Cholesky solver. Note that the primal and dual objective constants are added to the objective function value computed in the IPM, and the primal objective function $\mathbf{c}$ and constraint right-hand side $\mathbf{b}$ are modified as shown in Section 1.3.3

3.4.4.2 Handling dense columns

A column with a large number of non-zeros in the constraint matrix, relative to the other columns, leads to a considerably more dense Schur complement system than would be the case if the column did not contribute to the Schur complement system. For this reason, various approaches for reducing the impact of dense columns have been developed [212], [213]. The approach used here is based on Andersen's [213] modified Schur complement method and is detailed below.

Let $\mathbf{A}_D$ contain the dense columns of the constraint matrix, $\mathbf{A}$, and the remaining columns be $\mathbf{A}_S$. If the $(1,1)$ block of the augmented equations, $-\mathbf{F}^2$ (including diagonal scaling for linear variables and perturbations for free variables along), is partitioned accordingly, then the system to be solved is

$$\begin{bmatrix} -\mathbf{F}_D^2 & & \mathbf{A}_D^T \\ & -\mathbf{F}_S^2 & \mathbf{A}_S^T \\ \mathbf{A}_D & \mathbf{A}_S & \end{bmatrix} \begin{Bmatrix} \mathbf{x}_D \\ \mathbf{x}_S \\ \mathbf{y} \end{Bmatrix} = \begin{Bmatrix} \mathbf{p}_D \\ \mathbf{p}_S \\ \mathbf{q} \end{Bmatrix}.$$

Eliminating $\mathbf{x}_S$ gives

$$\begin{bmatrix} -\mathbf{F}_D^2 & \mathbf{A}_D^T \\ \mathbf{A}_D & \mathbf{A}_S \mathbf{F}_S^{-2} \mathbf{A}_S^T \end{bmatrix} \begin{Bmatrix} \mathbf{x}_D \\ \mathbf{y} \end{Bmatrix} = \begin{Bmatrix} \mathbf{p}_D \\ \mathbf{q} + \mathbf{A}_S \mathbf{F}_S^{-2} \mathbf{p}_S \end{Bmatrix}.$$

This differs from Andersen's approach in that $\mathbf{F}_D^2$ need not be the identity matrix, indeed, it is not even assumed to be non-singular or contain any non-zero entries. This approach allows dense columns associated with free variables to be treated explicitly, without any perturbations or modification to a conic variable. In contrast, Andersen's method will require any dense columns that are free variables to be treated by some method considering them as a conic variable, increasing the amount of work needed to be done and memory requirements, and introducing the difficulties known to be caused by splitting free variables. If $\mathbf{L}\mathbf{L}^T = \mathbf{A}_S \mathbf{F}_S^{-2} \mathbf{A}_S^T$ and $\mathbf{y}$ is eliminated from the first block row, the resulting equation is

$$-\left(\mathbf{F}_D^2 + \mathbf{A}_D^T \mathbf{L}^{-T} \mathbf{L}^{-1} \mathbf{A}_D\right) \mathbf{x}_D = \mathbf{p}_D + \mathbf{A}_D \mathbf{L}^{-T} \mathbf{L}^{-1} \left(\mathbf{q} + \mathbf{A}_S \mathbf{F}_S^{-2} \mathbf{p}_S\right).$$

This suggests the following steps compute the solution:

1. compute the Cholesky factorisation $\mathbf{L}\mathbf{L}^T = \mathbf{A}_S\mathbf{F}^{-2}\mathbf{A}_S^T$;
2. solve $\mathbf{L}\mathbf{V} = \mathbf{A}_D$;
3. solve $\mathbf{L}\mathbf{r} = \left(\mathbf{q} + \mathbf{A}_S\mathbf{F}_S^{-2}\mathbf{p}_S\right)$;
4. solve $\left(\mathbf{F}_D^2 + \mathbf{V}^T\mathbf{V}\right)\mathbf{x}_D = -\left(\mathbf{p}_D + \mathbf{V}^T\mathbf{r}\right)$;
5. solve $\mathbf{L}^T\mathbf{y} = \left(\mathbf{r} - \mathbf{V}\mathbf{x}_D\right)$; and
6. solve $-\mathbf{F}_S^2\mathbf{x}_S = \mathbf{p}_S + \mathbf{A}_S^T\mathbf{y}$.

From the above it is apparent that an additional solve with the triangular factor plus the solution with the coefficient system $\mathbf{F}_D^2 + \mathbf{V}^T\mathbf{V}$ is required. Note also that even if $\mathbf{A}$ is of full row rank, $\mathbf{A}_S$ may not be and so there is no guarantee that the factorisation $\mathbf{L}\mathbf{L}^T = \mathbf{A}_S\mathbf{F}_S^{-2}\mathbf{A}_S^T$ exists. Although Andersen [213] suggests using the method described by Stewart [205], it was found that this procedure was not necessary for the FELA test problems, in which there is no more than a single dense column as a result of the elimination of free variables, and no dense columns in the original constraint matrices are present.

3.4.4.3 Eliminating fixed variables subject to a conic constraint

A fixed variable arises from a row singleton in the constraint matrix. If the variable is a free variable, it may be substituted out of the problem immediately (see Section 3.4.4.1). Similarly, linear variables may be eliminated if they are non-negative (if they are negative, the primal problem is infeasible). If, however, the fixed variable is part of a second-order cone constraint (or semidefinite constraint), then they may not be substituted out so easily. If the first variable associated with the k th second-order cone constraint is fixed, e.g. $x_j = b_i$ (that is, the j th variable has the only non-zero coefficient in the i th constraint and is in the k th second-order cone), there are three cases:

- $x_j = 0$, in which case all the other variables in the k th constraint must also be zero (by definition of the second-order cone), and so all of the values may be substituted out of the problem;

- $x_j \neq 0$ and the optimal value of $\mathbf{x}^k$ lies on the boundary of the second-order cone; or
- $x_j \neq 0$ and the optimal value of $\mathbf{x}^k$ lies on the interior of the second-order cone.

Because the values of the unfixed variables associated with the k th second-order cone are not known in the second and third cases, the variable may not be removed from the problem. Similar reasoning prevents the remaining variables from being removed if they are fixed. If the situation arises in which both the first variable and one of the other variables in the same cone are both fixed, then the variables may be "aggregated" and the cone size reduced by one for each aggregation [36]. For example, if the first variable of the k th cone $x_j = 4$, and the third variable in the same cone $x_{j+2} = 4$. The variable x_{j+2} may be eliminated from the problem after setting $x'_j = \sqrt{x_j^2 - x_{j+2}^2} = 3$, reducing the cone size by one [36].

Andersen *et al.* [57] show that the fixed variables constrained by a second-order cone may be further exploited to reduce the computational effort required to compute the search direction. For each of the fixed variables, they may be easily eliminated from every other constraint. Then the fixed variables may be symmetrically permuted in the augmented form of the equations such that

$$\mathbf{P}\begin{bmatrix} -\mathbf{H} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix}\mathbf{P}^T\mathbf{P}\begin{Bmatrix} \mathbf{x} \\ \mathbf{y} \end{Bmatrix} = \begin{bmatrix} -\mathbf{H}_{11} & -\mathbf{H}_{12} & \mathbf{0} & \mathbf{I} \\ -\mathbf{H}_{21} & -\mathbf{H}_{22} & \hat{\mathbf{A}}^T & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{A}} & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}\begin{Bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{y}_1 \\ \mathbf{y}_2 \end{Bmatrix} = \begin{Bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \end{Bmatrix} = \mathbf{P}\begin{Bmatrix} \mathbf{p} \\ \mathbf{q} \end{Bmatrix},$$

where $\hat{\mathbf{A}}$ are the constraints with no fixed variables (possibly containing dense columns), $\mathbf{x}$ and $\mathbf{y}$ are the unknowns, and $\mathbf{p}$ and $\mathbf{q}$ represent the right-hand side. From the fourth block equation, it is obvious that $\mathbf{x}_1 = \mathbf{q}_2$. This can be substituted into the second block equation which can then be solved for $\mathbf{x}_2$ and $\mathbf{y}_1$:

$$\begin{bmatrix} -\mathbf{H}_{22} & \hat{\mathbf{A}}^T \\ \hat{\mathbf{A}} & \mathbf{0} \end{bmatrix}\begin{Bmatrix} \mathbf{x}_2 \\ \mathbf{y}_1 \end{Bmatrix} = \begin{Bmatrix} \mathbf{p}_2 + \mathbf{H}_{21}\mathbf{q}_2 \\ \mathbf{q}_1 \end{Bmatrix}.$$

This may be reduced to the SPD form

$$\hat{\mathbf{A}}\mathbf{H}_{22}^{-1}\hat{\mathbf{A}}^T\mathbf{y}_1 = \mathbf{q}_1 + \mathbf{A}\mathbf{H}_{22}^{-1}\left(\mathbf{p}_2 + \mathbf{H}_{21}\mathbf{q}_2\right) \tag{3.1}$$

to compute $\mathbf{y}_1$, which can then be used to obtain $\mathbf{x}_2$ as

$$\mathbf{x}_2 = -\mathbf{H}_{22}^{-1}\left(\mathbf{p}_2 + \mathbf{H}_{21}\mathbf{q}_2 - \hat{\mathbf{A}}\mathbf{y}_1\right). \tag{3.2}$$

Finally, $\mathbf{y}_2$ can be found from

$$\mathbf{y}_2 = \mathbf{p}_1 + \mathbf{H}_{11}\mathbf{x}_1 + \mathbf{H}_{12}\mathbf{x}_2. \tag{3.3}$$

These operations may all be performed easily by making a few modifications and implementing some additional routines for the matrix-vector multiplies and solves involving the partitioned $\mathbf{H}$. Some details are provided below. The partitioning of $\mathbf{H}$ as

$$\theta^{-2}\begin{bmatrix}\mathbf{H}_{11} & \mathbf{H}_{12}\\ \mathbf{H}_{21} & \mathbf{H}_{22}\end{bmatrix} = -\begin{bmatrix}\mathbf{Q}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{Q}_2\end{bmatrix} + 2\begin{Bmatrix}\mathbf{w}_1\\ \mathbf{w}_2\end{Bmatrix}\begin{Bmatrix}\mathbf{w}_1\\ \mathbf{w}_2\end{Bmatrix}^T$$

(where the $\mathbf{Q}$ blocks are diagonal with entries ± 1), allows the inverse of $\mathbf{H}_{22}$ to be easily obtained using the Sherman-Morrison-Woodbury formula, giving

$$\mathbf{H}_{22}^{-1} = \theta^{-2}\left(-\mathbf{Q}_2 - \frac{2}{1-\mathbf{w}_2^T\mathbf{Q}_2\mathbf{w}_2}\mathbf{Q}_2\mathbf{w}_2\mathbf{w}_2^T\mathbf{Q}_2\right) \tag{3.4}$$

(note that $\mathbf{Q}_i = \mathbf{Q}_i^{-1}$ holds for any submatrix of a symmetric permutation of $\mathbf{Q}$ for a second-order cone). This is needed for matrix-vector multiplication as well as construction of the Schur complement matrix. Both operations require the factor $\left(1-\mathbf{w}_2^T\mathbf{Q}_2\mathbf{w}_2\right)^{-1}$ of the right-hand side in (3.4), so it is useful to compute this once for each iteration. The matrix-vector multiplication is easily applied with the inverse by only treating those variables whose index is not that of a fixed variable. By maintaining a compressed list of unfixed variables in each cone (with a pointer to the start of a list of each cone's unfixed variables, much like a CSR or CSC structure), it is trivial to perform the multiplication with the corresponding entries in $\mathbf{w}_2$. The construction of the Schur complement matrix is simplified by only storing $\hat{\mathbf{A}}$ (storing it in the conventional CSC structure), and scaling each outer-product associated with a fixed variable by the factor discussed above. Note that because the columns in $\hat{\mathbf{A}}$

corresponding to the fixed variables are zero, no further modifications to the implementation of the construction of the Schur complement matrix is necessary. This does require the modification of each matrix-vector product involving the matrix $\mathbf{A}$, which is easily performed with a list of the fixed variables (note that storing value of the fixed variable in the right-hand side vector $\mathbf{b}$ removes the need to store the coefficients in the fixed constraints as they are all simply one). The multiplications with $\mathbf{H}_{21}$ and $\begin{bmatrix}\mathbf{H}_{11} & \mathbf{H}_{12}\end{bmatrix}$ are most naturally performed with a list of the fixed variables, which are already stored for the matrix-vector products.

3.4.4.4 Presolve results

The problem set was tested again with an initial check for fixed variables before attempting to eliminate as many free variables as possible and checking for dense columns. Note that in some cases, additional fixed variables were present after the elimination, but were not exploited. Marginal improvements could be expected by repeating the presolve process until no more fixed variables can be identified nor free variables eliminated. The free variable elimination restricted the fill-in to ensure that any single elimination would not increase the size of the effective constraint matrix, i.e. if a free variable has n_r other variables in its row and n_c other variables in it column in the active part of the constraint matrix, then elimination will be allowed if it results in no more than $n_r + n_c$ fill-in entries. The Markowitz-based search searched a maximum of one column with a suitable free variable for elimination, and the pivot stability criterion for potential pivot a_{ij} used was $a_{ij} \geq 10^{-1}\max\left(\left|a_j\right|\right)$, where a_j is the j th column of $\mathbf{A}$. A column was considered dense if it contained more than $5\times$ the average number of entries per column. The complete results of the presolve process are shown in Table 11 and the IPM results are shown in Table 12 alongside the results from Section 3.4.4.2 with split and regularised free variables for comparison.

Presolving appears to reduce the number of iterations, the number of non-zeros in the Cholesky factor, and the total runtime when compared to both of the other approaches in the table. The iteration counts are shown in Figure 32 where the only problem in which the presolved approach does not converge with fewer iterations than either of the other approaches is on *3DtunheadLB*, where it solves the problem to a smaller tolerance

than that of the split free variables. Figure 33 shows that the presolve approach does reduce the number of non-zeros in the Cholesky factor, but not significantly so given the reduction in the dimension of the linear system seen in Table 12. This is because the Gaussian elimination of the columns associated with free variables results in additional entries in the constraint matrix. But, because of the factorisations computation time being proportional to the square of the column counts, this leads to a more pronounced improvement in the runtime between the approaches as shown in Figure 34.

On the large problems, the total time to solve all problems was 8617s with the presolve method and 13,077s without (split free variables). Note that the lower bound tunnel heading was solved to the desired accuracy using the presolve method, but was not by the other approaches. For the large three dimensional problems, except the *3DtunheadLB* problem (that was not solved to satisfactory tolerance without presolving), using a presolve method led to a $1.75\times$ speedup over the approach splitting free variables and without presolving. In many cases, presolving also led to a significant reduction in the size of the Cholesky factor and a consequent reduction in runtime over even the regularised free variable approach. presents the performance profile and displays the clear benefit of presolving the problems in the test set.

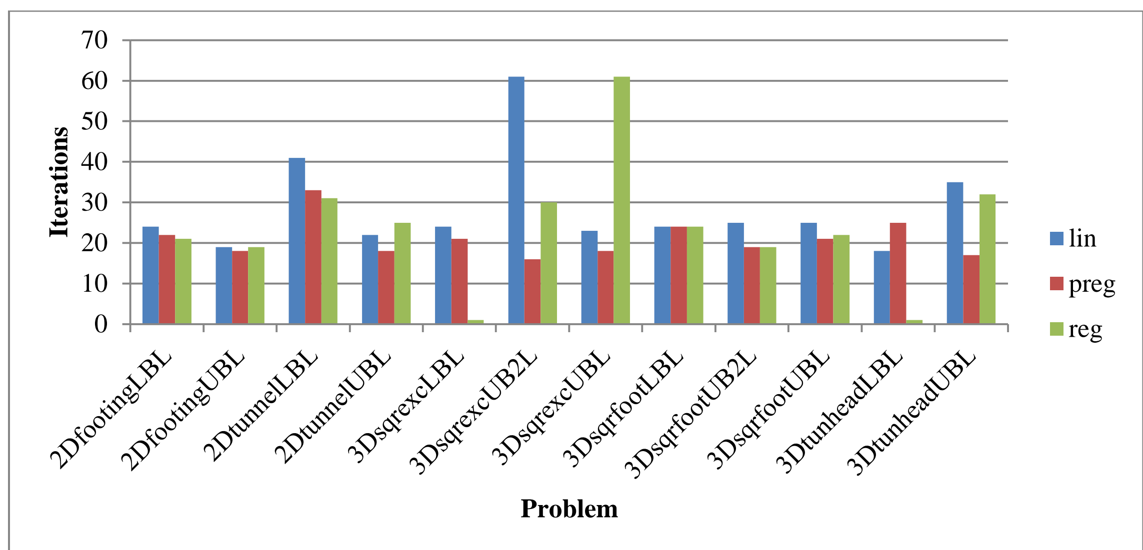


Figure 32. Comparison of the iteration count on the large problem set between the presolved problems and two approaches for handling free variables.

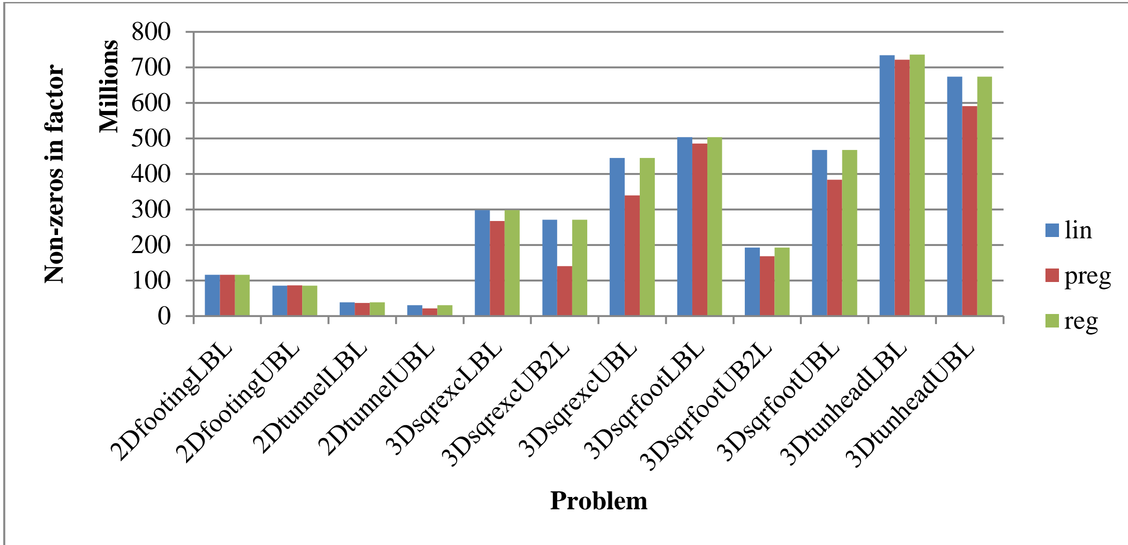


Figure 33. Comparison of the number on non-zeros in the factor on the large problem set between the presolved problems and two approaches for handling free variables.

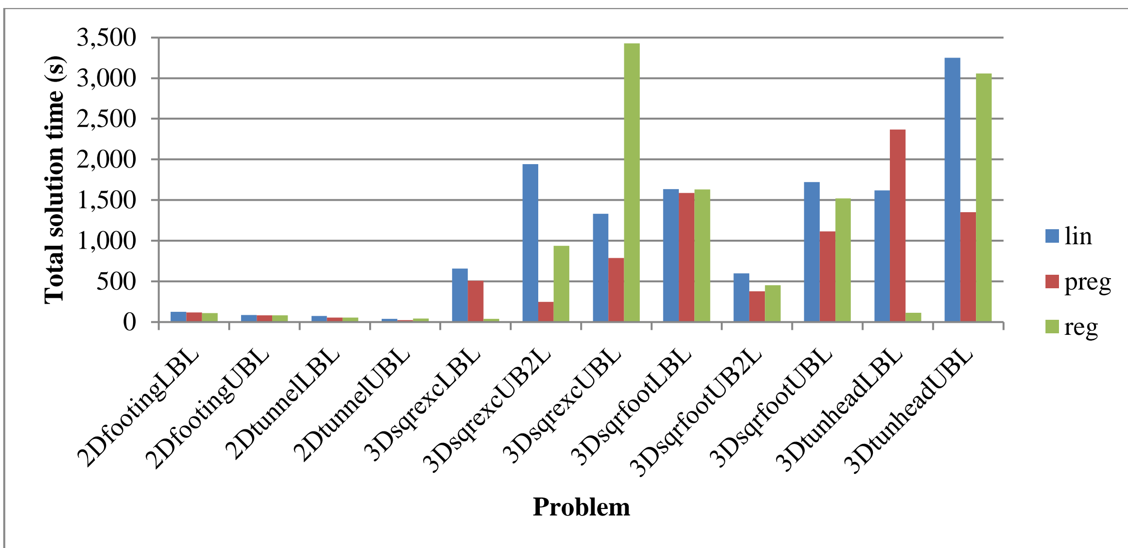


Figure 34. Comparison of the total solution time on the large problem set between the presolved problems and two approaches for handling free variables.

Table 11. Presolve results.

Problem	n_F	Eliminations	Fixed variables identified	Dense columns
2DfootingLBS	280	280	0	0
2DfootingLBM	420	420	0	0
2DfootingLBL	560	560	0	0
2DfootingUBS	980	980	0	0
2DfootingUBM	1,470	1,470	0	0
2DfootingUBL	1,960	1,960	0	0
2DtunnelLBS	58,080	1,505	57,600	1
2DtunnelLBM	130,320	2,257	129,600	1
2DtunnelLBL	231,360	3,011	230,400	1
2DtunnelUBS	77,480	51,404	76,480	1
2DtunnelUBM	173,820	115,162	172,320	1
2DtunnelUBL	308,560	204,710	306,560	1
3DsqrexcLBS	43,008	43,007	0	1
3DsqrexcLBM	145,152	145,151	0	1
3DsqrexcLBL	271,488	271,487	0	1
3DsqrexcUBS	119,280	119,279	0	1
3DsqrexcUBM	399,708	399,707	0	1
3DsqrexcUBL	944,064	944,063	0	1
3DsqrexcUB2S	56,726	56,725	0	1
3DsqrexcUB2M	184,470	184,469	0	1
3DsqrexcUB2L	429,158	429,157	0	1
3DsqrfootLBS	26,244	756	25,920	0
3DsqrfootLBM	62,016	1,344	61,440	0
3DsqrfootLBL	208,656	3,024	207,360	0
3DsqrfootUBS	48,492	12,070	43,956	0
3DsqrfootUBM	113,088	25,662	105,024	0
3DsqrfootUBL	375,408	76,702	357,264	0
3DsqrfootUB2S	29,094	3,149	25,920	0
3DsqrfootUB2M	67,014	5,533	61,440	0
3DsqrfootUB2L	219,750	12,305	207,360	0
3DtunheadLBS	42,300	42,299	0	1
3DtunheadLBM	99,920	99,919	0	1
3DtunheadLBL	335,736	335,735	0	1
3DtunheadUBS	77,400	77,399	0	1
3DtunheadUBM	180,848	180,847	0	1
3DtunheadUBL	600,984	600,983	0	1

Table 12. Comparison of presolve performance. Note that nnz(L) presents thousands of non-zeros.

		S				M				L			
Problem	**Approach**	**nit**	**t_T**	**nnz(L)**	**ϕ**	**nit**	**t_T**	**nnz(L)**	**ϕ**	**nit**	**t_T**	**nnz(L)**	**ϕ**
2DfootingLB	lin	28	30.7	26,598	1E-8	26	71.4	64,196	3E-8	24	123.8	116,194	3E-8
	reg	28	30.5	26,598	1E-8	24	67	64,196	2E-8	21	110.4	116,194	3E-8
	preg	28	30	25,792	2E-8	26	71.4	63,350	3E-8	22	117.7	116,090	5E-8
2DfootingUB	lin	23	21.1	18,859	7E-9	22	49	44,888	8E-9	19	84.5	85,687	7E-9
	reg	24	22	18,859	7E-9	23	50.7	44,888	8E-9	19	84	85,687	8E-9
	preg	23	21	18,333	8E-9	22	49.6	45,089	8E-9	18	82.2	86,104	8E-9
2DtunnelLB	lin	36	14.5	8,765	9E-9	41	39	20,923	8E-9	41	73.8	38,947	8E-9
	reg	36	13.6	8,765	2E-7	33	30.1	20,923	2E-6	31	52.8	38,947	6E-5
	preg	42	14.2	7,859	9E-8	39	32.3	19,463	2E-7	33	53.3	36,573	3E-7
2DtunnelUB	lin	23	9.7	7,018	6E-9	22	21.8	16,556	9E-9	22	39.9	30,414	7E-9
	reg	24	9.2	7,018	7E-9	19	17.6	16,556	1E-7	25	41.7	30,414	6E-9
	preg	19	5.6	4,595	6E-9	18	13	11,482	8E-9	18	24.5	21,637	6E-9
3DsqrexcLB	lin	19	28	29,638	6E-9	20	226.5	148,896	9E-9	24	657.2	297,683	8E-9
	reg	20	29.4	29,638	1E-8	1	16.6	148,896	1E+0	1	38.1	297,683	1E+0
	preg	17	19.9	24,149	7E-9	17	164.1	126,627	9E-9	21	506.8	267,285	1E-8
3DsqrexcUB	lin	22	32.7	26,818	9E-9	22	258.8	139,726	7E-9	23	1332.3	444,359	7E-9
	reg	24	34.4	26,818	5E-9	32	358.1	139,726	1E-7	61	3427.8	444,359	5E-6
	preg	19	18.4	18,521	4E-9	19	158.7	100,836	5E-9	18	787.5	339,284	8E-9
3DsqrexcUB2	lin	45	45.4	16,374	6E-9	61	414.8	83,756	5E-8	61	1942.3	270,925	9E-8
	reg	30	28.5	16,374	2E-7	30	199.4	83,756	2E-7	30	935.7	270,925	1E-7
	preg	17	9.3	7,932	9E-9	16	55.6	42,613	6E-9	16	247.9	139,934	6E-9
3DsqrfootLB	lin	22	34.9	31,096	8E-9	26	182.3	96,346	3E-9	24	1636	503,501	1E-8
	reg	20	31.7	31,096	9E-9	22	154.2	96,346	6E-9	24	1632.4	503,501	9E-9
	preg	20	29.7	29,161	1E-8	24	174.9	95,396	4E-9	24	1588.3	485,720	6E-9
3DsqrfootUB	lin	22	32.3	26,597	6E-9	23	153.1	85,656	5E-9	25	1721.1	467,076	6E-9
	reg	19	27.7	26,597	6E-9	20	132.3	85,656	3E-9	22	1521.6	467,076	4E-9
	preg	19	21.1	20,768	3E-9	19	92.7	67,046	8E-9	21	1113.6	383,195	6E-9
3DsqrfootUB2	lin	24	19	11,543	7E-9	25	71.1	36,142	8E-9	25	598.9	192,568	6E-9
	reg	19	14.5	11,543	4E-9	19	53.1	36,142	6E-9	19	451.4	192,568	3E-9
	preg	19	11.2	8,930	6E-9	19	43.7	30,064	8E-9	19	378.7	168,067	5E-9
3DtunheadLB	lin	15	32.4	45,049	6E-5	49	482.3	140,128	2E-5	18	1617.7	733,578	3E-4
	reg	1	3.8	45,049	1E+0	1	14.1	140,128	1E+0	1	114.5	735,899	1E+0
	preg	22	48.3	43,493	6E-9	34	321	135,606	9E-9	25	2366.7	720,932	6E-9
3DtunheadUB	lin	30	63.5	38,770	9E-9	31	293.4	123,914	1E-8	35	3249.8	673,234	2E-8
	reg	29	60.7	38,770	9E-9	22	198.2	123,914	1E-5	32	3056.8	673,234	8E-9
	preg	20	32.9	31,340	7E-9	19	146	106,303	9E-9	17	1349.7	590,632	5E-9

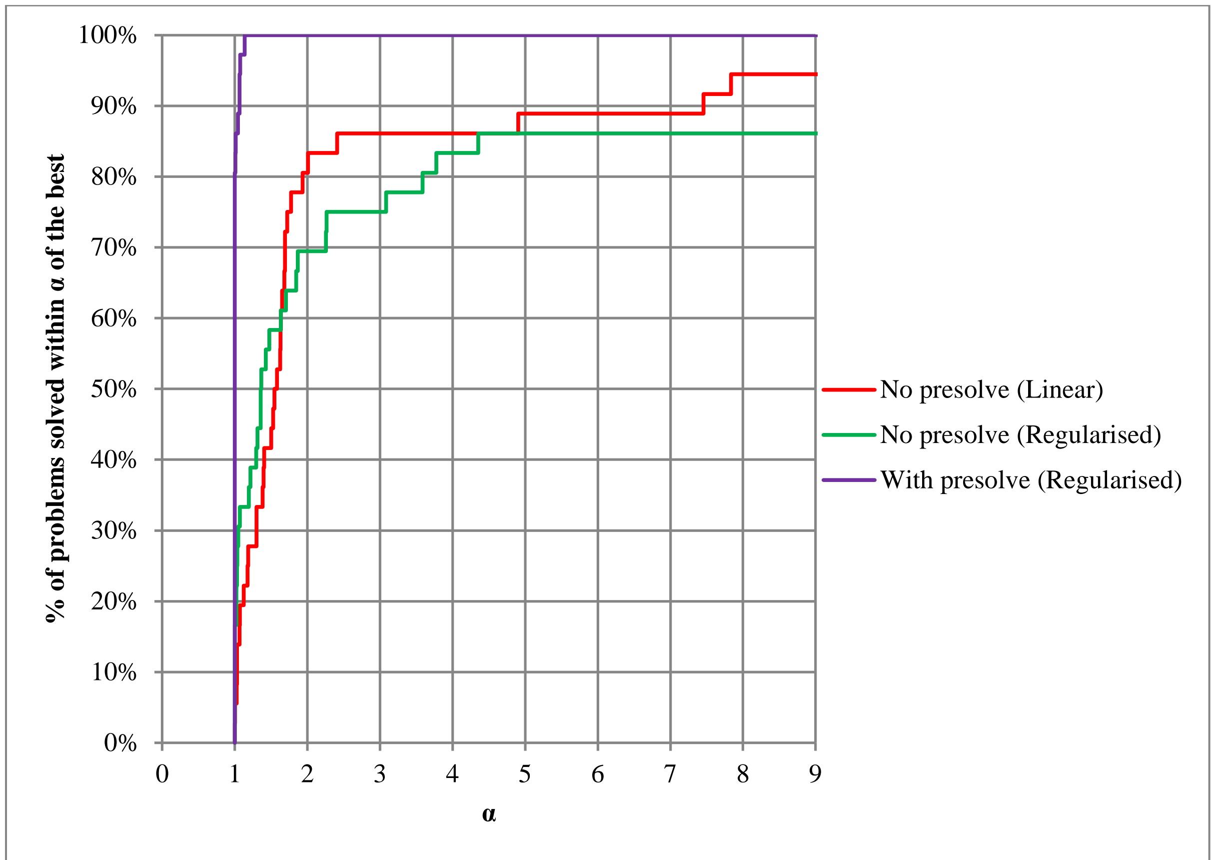


Figure 35. Performance profile of IPM runtime with and without presolve.

3.4.5Improvement summary

The preceding sections make it clear that there is enormous benefit to be gained through using different orderings and solvers, appropriately dealing with free variables, and presolving the problem. The improved IPM implementation with the supernodal Cholesky is subsequently referred to as `Mixup8`.

On the small problems alone, a $2\times$ speedup has been achieved overall from 525s with `Mix8` to 262s with the ND ordering, supernodal Cholesky, and presolve. This was more pronounced on the medium problems, where the total solve time was reduced from 23,275s to 1323s, a $17.5\times$ improvement, as well as solving *3DtunheadLB* to a satisfactory tolerance. Note that the bulk of the time in the medium test set was spent by `Mix8` in *3DsqrexcUB2*, 15,941s. Without the *3DsqrexcUB2* and *3DtunheadLB* problems, the improvement is from 7264s to 946s, or $7.7\times$ faster.

The improved performance is now comparable to the better of the two commercial solvers tested, `MOSEK`. Figure 36 shows that the iteration counts are still slightly above

that of MOSEK whose average of 18.8 iterations per problem on the large set is just over two iterations better than that of Mixup8 with 21.0 iterations per problem, although the average is slightly distorted with *2DtunnelLBL* taking a substantially greater number of iterations to converge. The complete results are shown in Table 13. The generally greater number of iterations also results in solutions with smaller infeasibilities, and yet is still completed in less time. The total solution time on the large problem set is shown in Figure 37. Both solvers converge to required tolerances or halt satisfactorily close to convergence except for MOSEK on the lower bound tunnel heading. Table 14 shows the total improvement over MOSEK on the entire test set, with a $1.7\times$ improvement on the large problem set, and the performance profile in Figure 38 shows the superiority of the improved implementation over MOSEK. These results show that there is benefit in developing a solver to ensure that the problems are solved with the greatest efficiency yet while the improvements have provided the capability to solve the test problems faster than MOSEK, there is still a distinct difference between the two-dimensional and three-dimensional problems.

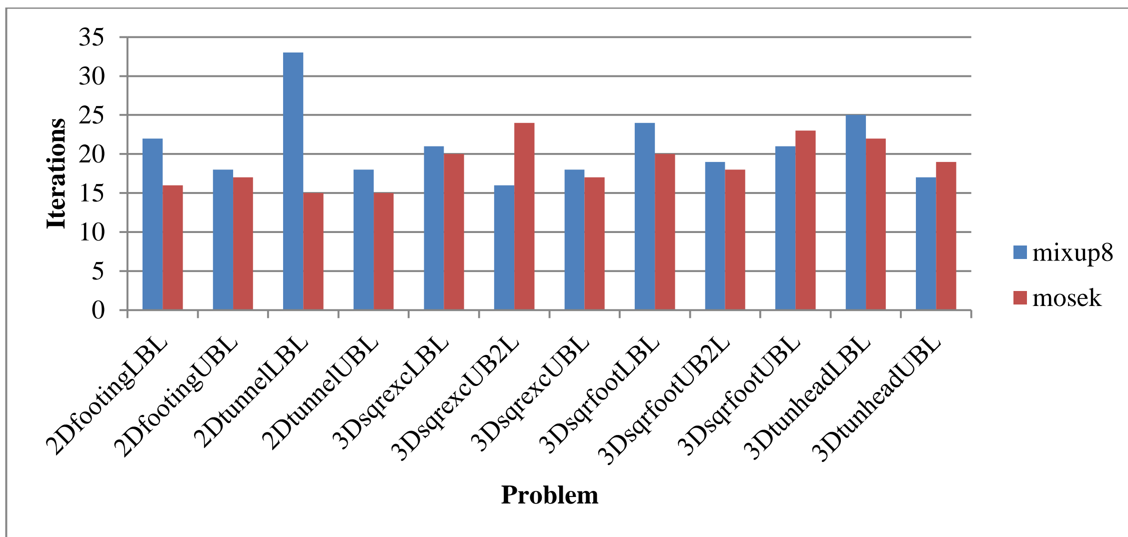


Figure 36. Comparison of the IPM iteration count between MOSEK and Mixup8.

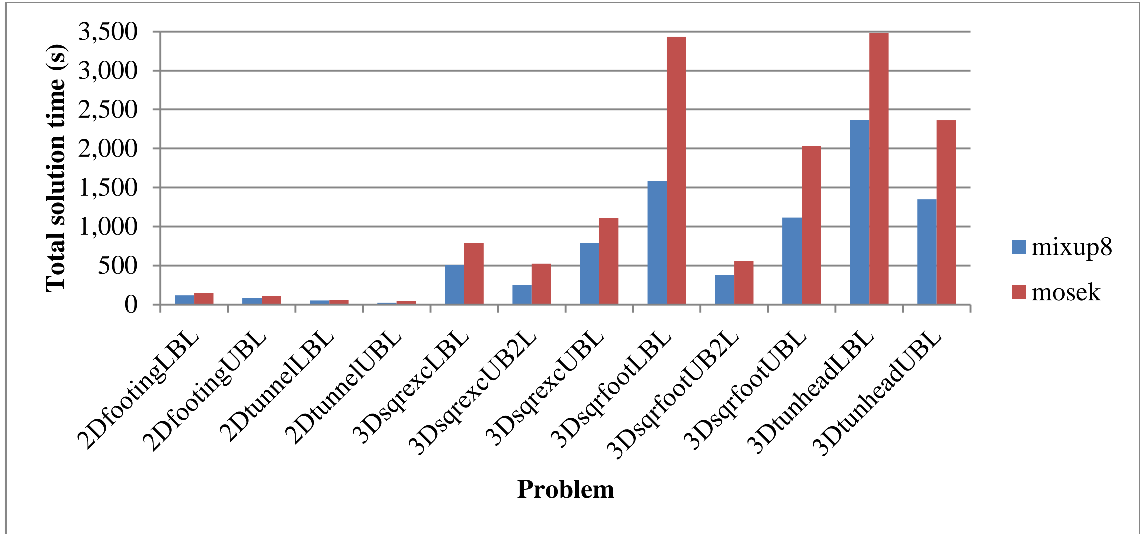


Figure 37. Comparison of the total solution time on the large problem set between `MOSEK` and the presolved IPM with a supernodal solver using a nested dissection ordering.

Table 13. Performance results compared with `MOSEK`.

		S			M			L		
Problem	**Method**	**nit**	**t_T**	**ϕ**	**nit**	**t_T**	**ϕ**	**nit**	**t_T**	**ϕ**
2DfootingLB	mosek	25	41.2	6E-09	20	90.0	8E-09	16	148.3	8E-09
	mixup8	28	30.0	2E-08	26	71.4	3E-08	22	117.7	5E-08
2DfootingUB	mosek	20	26.4	7E-09	20	62.3	5E-09	17	108.3	6E-09
	mixup8	23	21.0	8E-09	22	49.6	8E-09	18	82.2	8E-09
2DtunnelLB	mosek	19	14.5	5E-08	16	33.3	4E-08	15	57.8	5E-08
	mixup8	42	14.2	9E-08	39	32.3	2E-07	33	53.3	3E-07
2DtunnelUB	mosek	17	9.5	4E-08	15	22.5	2E-08	15	44.8	4E-08
	mixup8	19	5.6	6E-09	18	13.0	8E-09	18	24.5	6E-09
3DsqrexcLB	mosek	17	29.3	3E-08	16	245.8	3E-08	20	785.3	3E-08
	mixup8	17	19.9	7E-09	17	164.1	9E-09	21	506.8	1E-08
3DsqrexcUB	mosek	19	25.5	4E-08	18	197.8	3E-08	17	1108.8	4E-08
	mixup8	19	18.4	4E-09	19	158.7	5E-09	18	787.5	8E-09
3DsqrexcUB2	mosek	26	19.1	2E-07	23	110.4	1E-07	24	523.2	2E-07
	mixup8	17	9.3	9E-09	16	55.6	6E-09	16	247.9	6E-09
3DsqrfootLB	mosek	21	78.2	2E-08	21	375.3	3E-08	20	3432.3	3E-08
	mixup8	20	29.7	1E-08	24	174.9	4E-09	24	1588.3	6E-09
3DsqrfootUB	mosek	18	29.1	2E-08	21	179.2	2E-08	23	2030.6	3E-08
	mixup8	19	21.1	3E-09	19	92.7	8E-09	21	1113.6	6E-09
3DsqrfootUB2	mosek	19	15.0	3E-08	19	60.7	3E-08	18	556.1	3E-08
	mixup8	19	11.2	6E-09	19	43.7	8E-09	19	378.7	5E-09
3DtunheadLB	mosek	24	88.3	2E-05	28	414.5	2E-06	22	3482.4	1E-05
	mixup8	22	48.3	6E-09	34	321.0	9E-09	25	2366.7	6E-09
3DtunheadUB	mosek	20	46.1	4E-08	20	217.1	3E-08	19	2363.7	3E-08
	mixup8	20	32.9	7E-09	19	146.0	9E-09	17	1349.7	5E-09

Table 14. Total test set runtime.

	mosek	mixup8	mosek/mixup8
Small	422.2	261.6	1.6
Medium	2,008.9	1,323.1	1.5
Large	14,641.6	8,616.8	1.7

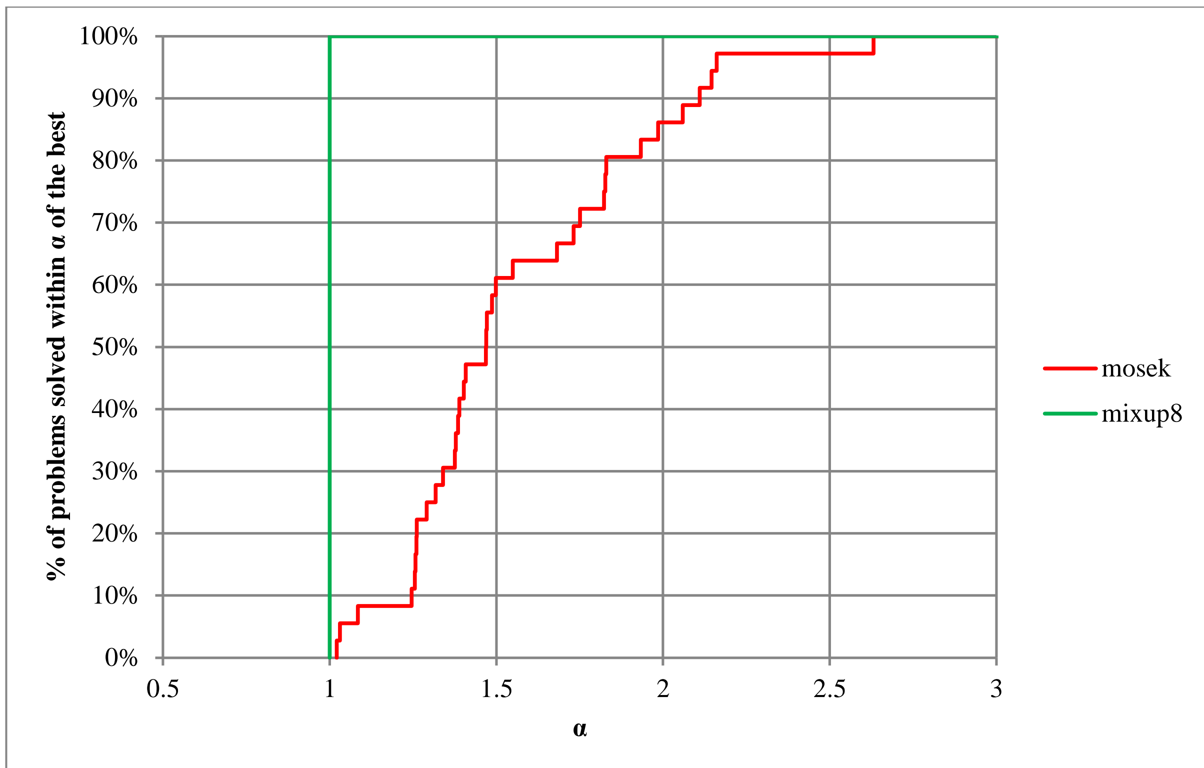


Figure 38. Performance profile of runtime by IPM solver with `MOSEK` *and the improved implementation.*

Chapter 4 Iterative solver approaches

4.1 Solving the normal equations

Because of the higher sensitivity to the problem and solution method parameters of iterative solvers compared with direct methods, it is expected that significant variation will be seen across the range of options. For this reason, it is important to systematically determine the parameters that lead to the most robust and efficient solution method. Among the options available, the choices may be split into three areas: options at the optimisation problem formulation and IPM level; preconditioning options; and choices around iterative solver parameters and methods.

Questions faced in formulating and solving the optimisation problem that are likely to affect the system defining the search direction include:

- What effect does the use of a presolve phase to exploit any opportunities to eliminate fixed and free variables have on the iterative solver's performance?
- Does the way in which remaining free variables are treated have an impact?
- What effect does the step length relaxation factor, γ, have on the iterative solver's performance?
- In choosing among the available preconditioning methods, the main questions include:
- How much variation is there between the different styles of preconditioner?
- How sensitive are the preconditioners to parameter choices, and what parameters are optimal?
- Do matrix permutations make a difference and which permutations are the most beneficial?
- For the actual iterative solvers, the following questions need to be considered:
- Is there a difference between the available iterative solvers?

- What tolerance on the residual norm is sufficient in solving each system, and should this tolerance be an absolute tolerance or and adaptive tolerance, changing as the IPM approaches a solution?

An understanding of which choices or range of choices provide robust and efficient methods is sought. This is achieved in the following by first answering questions regarding the iterative solvers and then testing a range of preconditioning and ordering options before considering choices at the problem formulation and IPM level. This approach enables the number of possible permutations to be significantly reduced while indicating which choices are most beneficial.

4.1.1 Test problems

In order to focus more specifically on the performance of the preconditioner and iterative solver, individual systems have been extracted from the IPM using a direct solver. For each one of the small problems, three systems will be considered: the first system, the first system in which $\rho_p \leq 10^{-4}$ and $\mu \leq 10^{-4}$, and the first system encountered when $\rho_p \leq 10^{-8}$ and $\mu \leq 10^{-7}$. Note that these problems were solved using the supernodal Cholesky with `METIS` ordering, step length relaxation factor of 0.95, no free variables were eliminated, no fixed variables or dense columns were exploited, and each free variable was considered as two linear variables. The number of non-zeros and order of the Schur complement system are shown, along with an upper bound on the forward error from `MA57` [214] and `MATLAB` condition estimate of each system in Table 15. The same details are provided for the augmented equations in Table 16. The rank deficiency is as reported by `MA57`.

As reported in Table 15 and Table 16, the conditioning of the linear system deteriorates in almost every problem the closer to convergence the IPM. The condition numbers on the third system for each problem indicate severely ill-conditioned matrices, in particular, the lower bound systems. Furthermore, it would not be unexpected for a direct solver to have difficulty in computing a stable factor that allows a relatively accurate solution to be obtained, with an upper bound on the forward error in many cases greater than 1.0. This difficulty was observed in Chapter 3 whereby optimisation packages considered robust and mature struggled to achieve convergence.

Table 15. Schur complement equation conditioning reported by `MA57` *and* `MATLAB`.

Problem	Iteration	ω_1	ω_2	κ_{ω_1}	κ_{ω_2}	UB on Forward error	Rank deficiency	MATLAB condition estimate
2DfootingLBS	1	3E-16	0E+00	2E+09	0E+00	6E-07	0	4E+10
	5	3E-16	0E+00	1E+10	0E+00	4E-06	0	1E+11
	26	7E-16	2E-23	2E+17	1E+14	1E+02	0	2E+18
2DfootingUBS	1	4E-16	0E+00	1E+08	0E+00	5E-08	0	2E+07
	7	3E-16	0E+00	2E+10	0E+00	5E-06	0	3E+08
	23	4E-16	1E-23	9E+16	1E+09	3E+01	0	5E+13
2DtunnelLBS	1	3E-16	0E+00	2E+07	0E+00	6E-09	0	3E+07
	7	3E-16	0E+00	3E+09	0E+00	8E-07	0	3E+08
	35	3E-16	9E-24	6E+15	5E+12	2E+00	0	9E+15
2DtunnelUBS	1	3E-16	0E+00	1E+07	0E+00	4E-09	0	2E+07
	8	3E-16	0E+00	8E+09	0E+00	3E-06	0	2E+08
	22	4E-16	1E-23	5E+15	4E+10	2E+00	0	3E+12
3DsqrexcLBS	1	4E-16	0E+00	8E+08	0E+00	3E-07	0	1E+08
	9	4E-16	5E-24	4E+12	4E+04	1E-03	0	2E+10
	16	4E-16	9E-24	4E+16	3E+09	1E+01	0	1E+13
3DsqrexcUBS	1	5E-16	1E-23	2E+10	5E+02	1E-05	0	7E+08
	10	5E-16	9E-24	6E+13	8E+05	3E-02	0	3E+10
	19	5E-16	1E-23	3E+18	5E+10	2E+03	0	7E+13
3DsqrexcUB2S	1	6E-16	0E+00	3E+06	0E+00	2E-09	0	4E+06
	8	6E-16	0E+00	2E+10	0E+00	1E-05	0	3E+08
	38	7E-16	7E-24	3E+15	4E+08	2E+00	0	7E+12
3DsqrfootLBS	1	2E-16	2E-28	3E+16	5E+06	5E+00	0	1E+17
	11	3E-16	6E-28	3E+18	6E+08	8E+02	0	1E+19
	19	1E-15	3E-19	1E+18	4E+12	1E+03	0	7E+19
3DsqrfootUBS	1	3E-16	0E+00	5E+05	0E+00	2E-10	0	1E+04
	12	4E-16	0E+00	3E+07	0E+00	1E-08	0	5E+06
	20	5E-16	6E-24	2E+09	3E+05	1E-06	0	2E+10
3DsqrfootUB2S	1	3E-16	0E+00	9E+04	0E+00	3E-11	0	5E+03
	11	5E-16	0E+00	1E+06	0E+00	5E-10	0	2E+06
	21	5E-16	6E-25	4E+08	4E+00	2E-07	0	3E+10
3DtunheadLBS	1	8E-14	6E-19	1E+19	2E+14	1E+06	0	6E+18
	12	2E-14	2E-18	3E+18	2E+15	6E+04	0	6E+30
	21	6E-14	2E-19	3E+17	4E+15	2E+04	0	2E+28
3DtunheadUBS	1	4E-16	0E+00	3E+06	0E+00	1E-09	0	2E+06
	13	5E-16	1E-23	2E+11	1E+03	1E-04	0	1E+09
	25	5E-16	2E-23	1E+16	1E+10	5E+00	0	1E+13

4.1.2 Choices related to the iterative solver

4.1.2.1 Solver parameters

The two parameters that are generally required to be passed to an iterative solver are the maximum number of iterations and the convergence tolerance. Obviously, only one of these two parameters will actually be used to terminate an iterative solver (except in the rare circumstance in which a method achieves convergence on the final iteration, although it may be argued that the method will actually only terminate because it converges or because it has reached the maximum number of iterations, but not both).

The convergence tolerance indicates the quality of the solution required by the application, while the maximum number of iterations indicates the maximum amount of work permissible to obtain the solution. In this case, we need the convergence threshold to ensure that sufficient progress will be made towards a solution of the optimisation problem, including reduction, or at least maintenance, of the primal and dual infeasibilities. The maximum number of iterations, however, must allow the solver to converge, if possible, while cutting off the solver in cases where it appears unlikely that the method will converge in a reasonable amount of time.

Table 16. Augmented equation conditioning as reported by `MA57`.

Problem	Iteration	w1	w2	Kw1	Kw2	UB on forward error	Rank deficiency
	1	3E-16	2E-23	3E+05	2E+06	9E-11	0
2DfootingLBS	5	4E-16	1E-23	2E+06	3E-02	7E-10	0
	26	3E-16	5E-23	1E+05	8E+08	3E-11	0
	1	4E-16	2E-23	1E+04	2E+00	4E-12	0
2DfootingUBS	7	3E-16	0E+00	2E+05	6E+00	6E-11	0
	23	3E-16	6E-23	3E+07	3E+09	1E-08	0
	1	3E-16	0E+00	3E+03	0E+00	1E-12	0
2DtunnelLBS	7	3E-16	0E+00	4E+04	0E+00	1E-11	0
	35	3E-16	2E-23	1E+08	3E+08	5E-08	0
	1	3E-16	6E-24	8E+03	1E+00	3E-12	0
2DtunnelUBS	8	3E-16	0E+00	1E+05	0E+00	3E-11	0
	22	4E-16	2E-23	1E+07	1E+09	4E-09	0
	1	3E-16	2E-23	2E+05	8E+05	5E-11	0
3DsqrexcLBS	9	3E-16	0E+00	1E+06	0E+00	4E-10	0
	16	3E-16	0E+00	5E+07	0E+00	2E-08	0
	1	4E-16	4E-23	4E+05	3E+07	1E-10	0
3DsqrexcUBS	10	3E-16	0E+00	3E+07	0E+00	1E-08	0
	19	4E-16	3E-23	2E+10	6E+06	7E-06	0
	1	4E-16	0E+00	6E+02	0E+00	2E-13	0
3DsqrexcUB2S	8	3E-16	0E+00	2E+04	0E+00	6E-12	0
	38	5E-16	2E-23	8E+07	2E+08	4E-08	0
	1	2E-16	2E-17	6E+01	4E+18	7E+01	11
3DsqrfootLBS	11	3E-16	3E-15	1E+04	4E+17	1E+03	2
	19	3E-16	3E-16	2E+07	9E+17	3E+02	24
	1	3E-16	0E+00	1E+04	0E+00	4E-12	0
3DsqrfootUBS	12	3E-16	0E+00	3E+03	0E+00	9E-13	0
	20	3E-16	2E-23	9E+05	6E+08	3E-10	0
	1	4E-16	0E+00	4E+03	0E+00	2E-12	0
3DsqrfootUB2S	11	3E-16	0E+00	6E+02	0E+00	2E-13	0
	21	3E-16	1E-23	1E+03	5E+07	4E-13	0
	1	3E-15	7E-17	5E+01	6E+18	4E+02	6
3DtunheadLBS	12	4E-15	6E-14	4E+03	4E+17	3E+04	2
	21	7E-16	2E-14	6E+05	4E+16	6E+02	4
	1	3E-16	8E-24	1E+03	2E+00	4E-13	0
3DtunheadUBS	13	3E-16	0E+00	8E+04	0E+00	3E-11	0
	25	3E-16	3E-23	8E+07	8E+07	3E-08	0

In the majority of the simulations to follow, an absolute convergence tolerance of 1×10^{-8} is used with a maximum number of iterations of 20,000. Such a convergence tolerance is necessary for the later iterations in the IPM to ensure that the primal and dual infeasibility is not increased preventing convergence.

4.1.3 Preconditioning the normal equations

4.1.3.1 Comparing the symmetric Krylov subspace solvers

When using a diagonal preconditioner, it is simple to explicitly precondition the linear system before passing it to the Krylov solver. This is achieved by taking the inverse of the square root of each diagonal entry, and performing row-scaling and column-scaling of the coefficient matrix such that the diagonal entries are one. Note that this affects both the solution vector and the right-hand-side. The approach is computed three steps:

With $d_{ii} = a_{ii}$, solve $\mathbf{Ax} = \mathbf{b}$:
Compute $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$
Solve $\tilde{\mathbf{A}}\mathbf{z} = \mathbf{D}^{-1/2}\mathbf{b}$
Compute $\mathbf{x} = \mathbf{D}^{-1/2}\mathbf{z}$

In the case of the conjugate gradient solver, this saves n multiplications per iteration (replacing the preconditioner step) and a vector update. It also reduces the storage requirements by a vector of length n. This scaling (resulting in unit values on the diagonal) maintains an approximately equal number of non-zeros in the incomplete factorisations as the IPM converges to a solution; without this scaling, the large number of small eigenvalues in the NT scaling matrix lead to many very large entries in the Schur complement system that do not get dropped in an incomplete-style preconditioner, leading to significantly increasing factorisation sizes in the latter iterations of the IPM. Interestingly, this approach converges in fewer iterations than when using the diagonal preconditioner at each iteration of a preconditioned Krylov solver. Thus, the following results for PCG, MINRES, and SymQMR are unpreconditioned algorithms with explicit preconditioning of the coefficient matrix before being passed to the Krylov solver.

4.1.3.2 The effect of matrix permutations

For direct methods, a sparsity-preserving ordering can have a significant impact on the relative performance of the method employed. This is because a poor ordering is likely

to result in many more non-zero entries in the factor as is clear from the comparison of available solvers in Section 3.4.2. In the case of incomplete factorisations, however, the effect of any reordering used may be much more subtle given that the quality of the incomplete factorisation may not necessarily depend upon a good sparsity-preserving ordering.

The tests performed here use an absolute convergence tolerance in the residual norm of 10^{-8} with the PCG solver and the Ajiz-Jennings RIC1 incomplete factorisation as a preconditioner with a drop tolerance of 10^{-2} and 10^{-4} relative to the column diagonal entry. The four permutations compared are the reverse Cuthill-McKee (RCM) ordering, Sloan's profile reducing ordering, approximate minimum degree (AMD) ordering, and a nested dissection (ND) ordering. The AMD and ND are the same as those tested for the direct methods in Section 3.4.2; HSL's `MC47` AMD and the `METIS` ND routines. The RCM implementation is that contained in `SPARSEPAK` [72], [215], while the Sloan ordering is the HSL routine `MC40` [216]. Both of these codes use pseudo-peripheral nodes [72], instead of a "node which might be a node of minimum degree" as suggested for the original Cuthill-McKee ordering [117], as the starting node for each connected component in the graph of the matrix, but whereas RCM was designed to minimise the bandwidth of positive-definite finite element matrices, the Sloan ordering sought to minimise their profile.

As stated above, the ordering need only be constructed once for each problem and so generally has a negligible impact on the overall performance in solving the problem. For completeness, however, the ordering construction times are included here in Table 17. As expected, the RCM ordering is the fastest to construct and the nested dissection ordering the slowest for every problem in the small test set, the only exception being *2DfootingLBS*, in which Sloan's ordering is considerably slower than all the other methods. Interestingly, the AMD ordering is no more than double the quick construction time of RCM, while Sloan's ordering falls in between the AMD and ND method times.

As can be seen in Table 18, with a drop threshold of $\tau = 10^{-2}$ the number of non-zeros in the factor is quite similar across the different orderings, with the density, on average, increasing from the RCM ordering, the Sloan ordering, to the nested dissection, and

then the approximate minimum degree. The Sloan ordering is quite inconsistent, with 15 systems of the 36 in which it has fewer non-zeros than RCM, but then has over two and a half times more entries on some other systems. For the systems considered here, the factorisation with the RCM ordering had between approximately $0.5\times$ to $2\times$ the non-zeros in the coefficient matrix.

Table 17. Schur complement ordering time. These times are taken from the analysis of each problem with $\tau = 10^{-2}$.

Problem	RCM	Sloan	ND	AMD
2DfootingLBS	0.08	0.39	2.03	0.11
2DfootingUBS	0.05	0.23	1.33	0.07
2DtunnelLBS	0.03	0.12	1.03	0.05
2DtunnelUBS	0.03	0.11	0.71	0.04
3DsqrexcLBS	0.08	2.10	1.27	0.10
3DsqrexcUBS	0.05	0.53	1.60	0.08
3DsqrexcUB2S	0.05	0.80	1.32	0.13
3DsqrfootLBS	0.06	0.46	0.79	0.08
3DsqrfootUBS	0.03	0.25	0.73	0.04
3DsqrfootUB2S	0.04	0.25	0.59	0.07
3DtunheadLBS	0.09	1.01	1.28	0.12
3DtunheadUBS	0.06	0.58	1.19	0.10

The number of iterations required to solve each system with the PCG solver is shown in Table 19. On average, the RCM ordering leads to the quickest preconditioner construction times, followed by the ND and then the AMD orderings, with the Sloan ordering suffering from considerable variance in the performance from on par with the RCM ordering to around $5\times$ slower. The number of iterations required to solve the systems, however, is almost the opposite of this situation.

The number of iterations spent before achieving the convergence criterion $\|\mathbf{b}-\mathbf{Ax}\|_2 \le 10^{-8} = \varepsilon$ (on the diagonally scaled system) is shown in Table 20. The maximum number of iterations was set at $20,000$, indicating that many of the solution attempts failed to achieve convergence. This was especially so for the last system tested in each problem, where only the two upper bounds on the square footing problem saw the solver reach convergence and only with the ND and AMD orderings. Interestingly, none of the approaches converged for any of the *3DtunheadLBS* systems. This is likely to be due to the near rank-deficiency of the constraint matrix, which caused problems for standard direct Cholesky and multifrontal $\mathbf{LDL}^T$ factorisations. For the majority of the middle systems in each problem, the AMD ordering often led to the most effective

preconditioner, often followed by the ND ordering. The RCM and Sloan orderings varied from competitive up to $6\times$ worse than the AMD/ND-based preconditioners on some of the lower bound problems. The iteration count approximately corresponds to the time spent in the solver, with per iteration time generally similar across the compared approaches. The time per 100 iterations is shown in Figure 39 and is fairly consistent across the orderings. The performance profile of iteration counts is provided in Figure *40*, and shows the AMD ordering to be the most effective ordering when used with RIC1. It should be noted that the performance profile of the iteration count is very similar to the efficiency profile.

Table 18. Non-zeros in the RIC1 factorisation with $\tau = 10^{-2}$.

Problem	Iteration	RCM	Sloan	ND	AMD
	1	5,351,560	5,021,559	7,232,403	7,332,105
2DfootingLBS	5	5,344,679	5,097,573	7,390,588	7,497,593
	26	6,093,082	5,632,803	7,947,492	8,712,208
	1	3,086,158	3,514,126	4,473,881	4,641,495
2DfootingUBS	7	3,091,083	3,664,354	4,914,935	5,208,697
	23	3,203,845	3,798,060	5,756,218	7,175,940
	1	1,615,035	1,578,850	2,100,652	2,055,544
2DtunnelLBS	7	2,081,038	1,961,639	2,811,970	2,846,691
	35	2,225,347	2,107,063	2,974,596	3,273,011
	1	1,114,250	1,315,277	1,539,955	1,541,035
2DtunnelUBS	8	1,561,960	3,977,735	2,097,501	2,222,921
	22	1,582,770	4,145,997	2,222,805	2,531,137
	1	2,382,344	2,218,662	2,796,623	2,912,654
3DsqrexcLBS	9	3,359,524	3,012,265	3,767,826	4,055,837
	16	3,341,677	3,008,978	3,742,613	4,051,321
	1	1,157,530	1,188,779	1,389,075	1,434,230
3DsqrexcUBS	10	1,674,244	2,497,296	2,163,265	2,331,207
	19	1,708,513	2,601,577	2,254,724	2,478,762
	1	931,614	975,751	956,892	959,317
3DsqrexcUB2S	8	1,786,814	1,837,004	1,699,006	1,744,273
	38	1,914,976	1,938,824	1,807,734	1,923,425
	1	2,274,027	2,177,695	2,814,157	2,918,339
3DsqrfootLBS	11	2,768,058	2,686,452	3,464,683	3,623,474
	19	2,789,795	2,713,795	3,518,760	3,712,695
	1	1,058,662	1,141,424	1,334,937	1,391,460
3DsqrfootUBS	12	1,452,138	2,107,675	1,754,673	1,839,161
	20	1,478,222	2,238,431	1,817,334	1,933,840
	1	968,700	882,465	988,047	989,937
3DsqrfootUB2S	11	1,681,667	1,439,689	1,549,504	1,562,729
	21	1,742,187	1,462,455	1,571,703	1,589,830
	1	4,096,957	6,471,163	4,852,909	4,909,300
3DtunheadLBS	12	4,853,125	10,167,924	6,040,053	6,485,309
	21	4,986,376	11,605,757	6,148,181	6,721,038
	1	1,904,763	2,300,756	2,217,165	2,289,268
3DtunheadUBS	13	2,230,723	3,704,995	2,635,020	2,776,475
	25	2,302,339	4,349,070	2,886,347	3,189,039

Table 19. RIC1 preconditioner construction time with $\tau = 10^{-2}$.

Problem	Iteration	RCM	Sloan	ND	AMD
	1	0.38	0.40	0.52	0.60
2DfootingLBS	5	0.38	0.40	0.53	0.59
	26	0.46	0.46	0.59	0.84
	1	0.20	0.20	0.31	0.35
2DfootingUBS	7	0.20	0.21	0.33	0.40
	23	0.21	0.23	0.40	0.62
	1	0.12	0.13	0.16	0.17
2DtunnelLBS	7	0.14	0.14	0.20	0.22
	35	0.16	0.16	0.21	0.28
	1	0.09	0.10	0.12	0.13
2DtunnelUBS	8	0.10	0.46	0.14	0.18
	22	0.10	0.49	0.15	0.20
	1	0.29	0.22	0.30	0.35
3DsqrexcLBS	9	0.41	0.33	0.42	0.53
	16	0.39	0.32	0.42	0.54
	1	0.13	0.16	0.16	0.19
3DsqrexcUBS	10	0.17	0.46	0.23	0.29
	19	0.17	0.48	0.24	0.31
	1	0.16	0.24	0.17	0.17
3DsqrexcUB2S	8	0.28	0.53	0.24	0.26
	38	0.42	0.60	0.25	0.29
	1	0.34	0.23	0.51	0.34
3DsqrfootLBS	11	0.28	0.26	0.36	0.42
	19	0.29	0.27	0.37	0.43
	1	0.12	0.13	0.16	0.18
3DsqrfootUBS	12	0.14	0.31	0.18	0.21
	20	0.14	0.33	0.18	0.23
	1	0.17	0.15	0.17	0.16
3DsqrfootUB2S	11	0.26	0.18	0.19	0.20
	21	0.27	0.18	0.20	0.21
	1	0.58	0.93	0.69	0.58
3DtunheadLBS	12	0.51	1.56	0.65	0.78
	21	0.53	1.84	0.66	0.89
	1	0.19	0.34	0.23	0.29
3DtunheadUBS	13	0.21	0.61	0.25	0.33
	25	0.21	0.70	0.28	0.39

The behaviour seen for the low-fill factorisations is quite different to the higher-fill preconditioners. Reducing the drop tolerance to $\tau = 10^{-4}$ significantly increases the number of fill-in entries that are kept in the factorisation process and consequently significantly increases the time required to construct the preconditioner. This is usually offset by a reduction in the number of iterations required to achieve convergence in the solve phase. The number of non-zeros in the factorisation with the reduced drop tolerance is shown in Table 21. For the second two systems in each problem, the RIC1 factorisation with the AMD ordering achieves up to a 50% reduction in the size of the

factor over the MOSEK Cholesky factorisation. Many problems see approximately the same number of entries comparing the direct method and the incomplete factorisation, and for the *3DsqrexcUB2* systems, the incomplete factorisation with all orderings significantly underperforms the direct method with 25% to $14\times$ more entries than the seven million entries in MOSEK's full factorisation. This is possibly because of the significant savings achieved by eliminating free variables, avoiding dense columns, and exploiting fixed variables.

The much greater size of the factors also incurs a significantly greater construction time cost, as can be seen in Table 22. Consistent with the amount of fill-in noted previously, these times are not competitive with the factorisation times of the direct methods in the high-performance conic program software packages.

Table 20. Iterations required with RIC1 preconditioned CG ($\tau = 10^{-2}$ and $\varepsilon = 10^{-8}$). † indicates convergence did not occur within the maximum 20,000 iterations.

Problem	Iteration	RCM	Sloan	ND	AMD
	1	†	†	17,578	15,592
2DfootingLBS	5	†	†	13,415	15,347
	26	†	†	†	†
	1	1,014	640	792	755
2DfootingUBS	7	1,134	721	871	866
	23	†	†	†	†
	1	2,329	2,058	1,005	520
2DtunnelLBS	7	3,596	3,139	1,513	1,001
	35	†	†	†	†
	1	346	389	345	304
2DtunnelUBS	8	701	1,520	734	528
	22	†	†	†	†
	1	3,044	1,563	1,298	1,295
3DsqrexcLBS	9	†	8,811	6,317	3,258
	16	†	†	†	†
	1	1,520	2,839	1,526	1,375
3DsqrexcUBS	10	3,463	10,042	3,396	2,773
	19	†	†	†	†
	1	242	804	294	147
3DsqrexcUB2S	8	1,335	3,724	1,575	946
	38	†	†	†	†
	1	240	197	173	160
3DsqrfootLBS	11	†	†	†	†
	19	†	†	†	†
	1	79	88	87	84
3DsqrfootUBS	12	440	792	491	402
	20	15,688		17,793	14,081
	1	61	58	56	52
3DsqrfootUB2S	11	595	439	385	306
	21	†	18,202	17,768	14,079
	1	†	†	†	†
3DtunheadLBS	12	†	†	†	†
	21	†	†	†	†
	1	202	392	184	153
3DtunheadUBS	13	2,090	5,766	1,685	990
	25	†	†	†	†

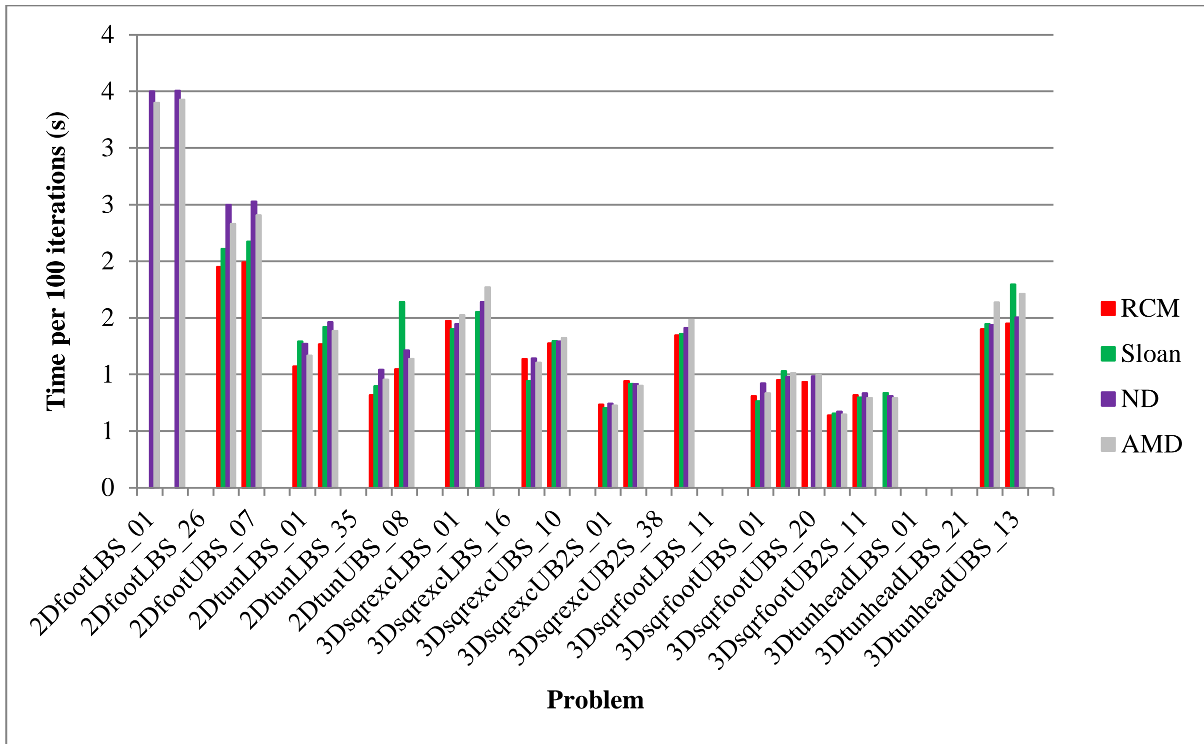


Figure 39. Time per 100 iterations with RIC1 ($\tau = 10^{-2}$) preconditioned CG.

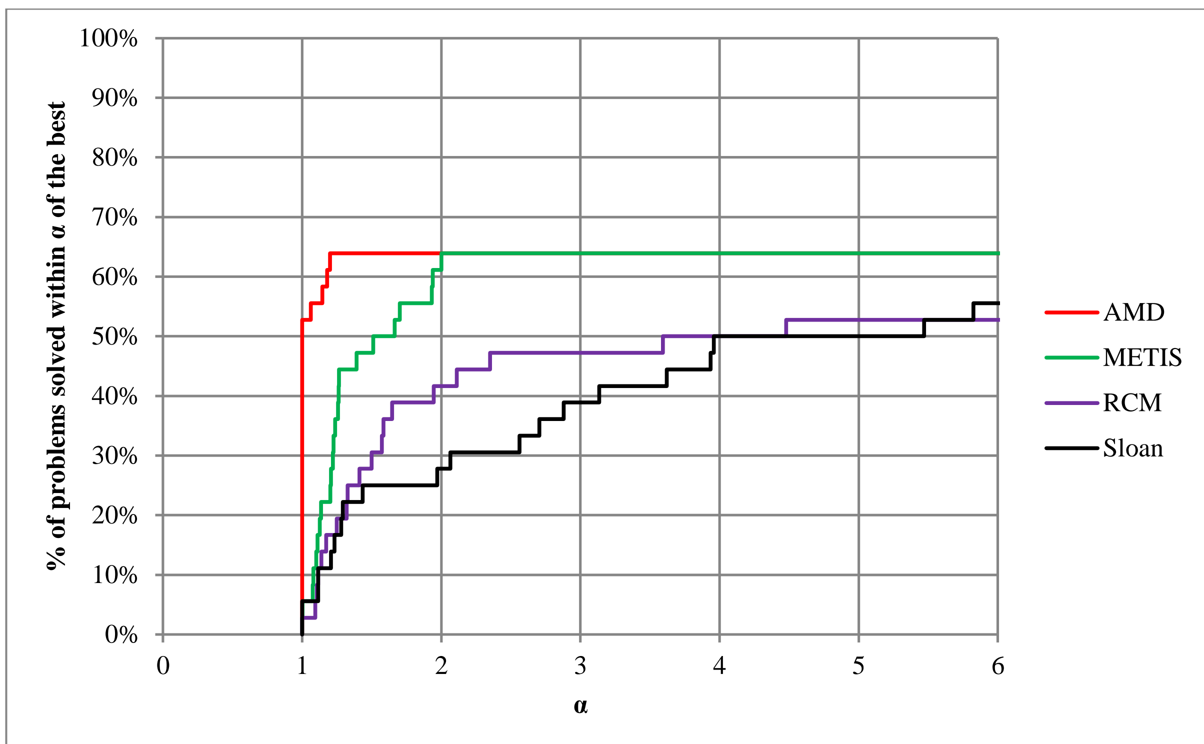


Figure 40. Performance profile of iteration counts by ordering method with RIC (with $\tau = 10^{-2}$).

Table 21. Non-zeros in the RIC1 factorisation with $\tau = 10^{-4}$.

Problem	Iteration	RCM	Sloan	ND	AMD
2DfootingLBS	1	33,692,140	27,168,805	15,655,565	18,257,763
	5	33,762,490	28,453,869	16,054,870	19,015,901
	26	48,636,953	44,601,271	17,365,679	22,775,499
2DfootingUBS	1	15,175,693	13,007,502	10,398,601	12,097,548
	7	15,309,907	13,440,530	11,156,000	13,717,719
	23	20,170,032	17,654,841	11,975,128	16,737,184
2DtunnelLBS	1	8,840,380	7,596,094	4,461,745	4,881,983
	7	12,609,642	10,155,077	5,154,352	5,901,792
	35	16,149,373	13,741,086	5,159,112	6,197,963
2DtunnelUBS	1	5,343,264	4,377,259	3,397,486	3,454,875
	8	7,174,963	76,738,405	3,972,181	4,155,104
	22	8,146,368	79,522,252	3,953,276	4,247,076
3DsqrexcLBS	1	24,669,271	18,630,509	13,374,402	12,500,731
	9	46,325,896	34,091,946	17,222,833	22,932,096
	16	46,037,355	34,751,389	16,462,910	22,486,943
3DsqrexcUBS	1	9,214,213	23,161,388	8,405,736	7,889,490
	10	15,749,067	104,673,027	12,102,548	14,221,679
	19	17,138,959	113,094,148	12,135,964	15,586,904
3DsqrexcUB2S	1	8,563,275	36,005,597	6,925,040	6,516,542
	8	18,062,951	100,841,446	8,871,473	10,543,303
	38	18,838,845	98,254,858	8,636,873	11,237,777
3DsqrfootLBS	1	22,880,118	18,211,879	13,843,980	12,185,233
	11	28,130,231	23,447,179	16,347,371	15,724,039
	19	27,101,576	23,324,930	16,186,892	16,077,724
3DsqrfootUBS	1	8,866,821	9,748,974	8,709,581	8,285,553
	12	10,387,000	47,030,545	9,817,532	9,749,364
	20	10,275,326	52,277,929	9,663,466	9,829,626
3DsqrfootUB2S	1	6,730,887	6,578,043	4,662,536	4,472,196
	11	12,128,029	8,537,004	5,980,154	6,166,816
	21	12,807,654	8,781,651	5,884,983	6,307,611
3DtunheadLBS	1	43,217,352	92,280,177	22,985,628	22,626,345
	12	57,267,713	213,969,882	27,467,599	30,964,250
	21	61,267,081	313,090,153	27,127,675	31,813,138
3DtunheadUBS	1	14,246,984	41,270,583	11,824,650	11,616,517
	13	14,884,890	91,538,263	13,533,102	14,119,504
	25	15,340,515	123,106,100	15,557,591	18,815,249

Table 22. RIC1 preconditioner construction time with $\tau = 10^{-4}$.

Problem	Iteration	RCM	Sloan	ND	AMD
2DfootingLBS	1	4.06	2.95	1.89	2.97
	5	4.04	3.24	2.05	3.37
	26	6.86	6.40	2.92	6.10
2DfootingUBS	1	1.40	1.05	1.01	1.49
	7	1.43	1.12	1.18	2.07
	23	2.34	1.81	1.50	4.01
2DtunnelLBS	1	0.87	0.72	0.41	0.57
	7	1.27	1.00	0.50	0.78
	35	1.97	1.70	0.52	0.90
2DtunnelUBS	1	0.49	0.36	0.30	0.31
	8	0.67	48.09	0.35	0.40
	22	0.75	53.57	0.35	0.42
3DsqrexcLBS	1	6.73	4.49	4.10	4.85
	9	18.86	15.55	7.64	23.37
	16	18.96	16.96	7.13	22.37
3DsqrexcUBS	1	1.87	17.28	1.91	1.86
	10	4.20	119.60	4.28	8.36
	19	4.67	136.60	4.34	10.85
3DsqrexcUB2S	1	2.50	40.26	2.25	2.30
	8	7.89	182.00	3.81	6.67
	38	8.75	189.10	3.62	7.74
3DsqrfootLBS	1	7.51	3.88	7.52	4.49
	11	7.23	5.67	6.75	8.22
	19	6.91	5.76	6.76	8.91
3DsqrfootUBS	1	1.70	1.97	2.43	2.26
	12	1.85	36.89	3.01	3.09
	20	1.83	44.09	2.89	3.16
3DsqrfootUB2S	1	1.84	1.88	1.59	1.51
	11	3.40	2.47	2.02	2.36
	21	3.81	2.73	2.00	2.52
3DtunheadLBS	1	14.94	61.00	10.63	9.68
	12	17.41	309.40	12.10	23.32
	21	20.41	765.90	11.92	26.01
3DtunheadUBS	1	2.50	20.67	2.68	2.76
	13	2.59	81.90	3.58	4.26
	25	2.74	138.40	5.25	9.82

As expected, the number of iterations taken to convergence is greatly reduced by the much more accurate preconditioner. The iteration counts, again limited to 20,000, are shown in Table 23. Importantly, more of the systems were solved to tolerance with all orderings, although some of the lower bound systems were still unable to be solved with the AMD and ND-based preconditioners which achieved convergence on 30 of the 36 systems tested. The RCM and Sloan orderings are clearly outperformed in their preconditioning effectiveness as well as the quality of the sparsity-preservation in the more dense factorisations. The RCM ordering-based preconditioner solved 25 systems, while the Sloan ordering-based preconditioner solved only 23. In all cases, the

factorisation with the AMD ordering outperforms the ND ordering-based factorisation. This difference was significant for some systems, indicating that the AMD ordering was unquestionably the most robust choice among the options considered for the systems tested with the lower drop tolerance. The time per 100 iterations is displayed in Figure 41, showing the effect of the large number of non-zeros in the preconditioner. Similar to RIC1 with $\tau = 10^{-2}$, the performance profile of the iteration counts in Figure 42 shows AMD to be the ordering with the highest likelihood of providing an ordering to produce an incomplete factorisation preconditioner among the orderings tested.

These results make it clear that to obtain the search direction with an iterative solver in the later iterations of the IPM, more accurate preconditioners must be constructed. If one is to construct these more dense factorisations, then the banded preconditioners are not suitable, necessitating a choice between the ND and AMD orderings. While the ND ordering resulted in fewer non-zeros in the factorisation and often faster construction times, the effectiveness of the preconditioner was clearly inferior to the AMD-based preconditioner. Following these results, all incomplete Cholesky factorisations in the study use the AMD ordering.

Table 23. Iterations required with RIC1 preconditioned CG ($\tau = 10^{-4}$ and $\varepsilon = 10^{-8}$). † indicates convergence did not occur within the maximum 20,000 iterations.

Problem	Iteration	RCM	Sloan	ND	AMD
	1	9,202	5,942	1,118	412
2DfootingLBS	5	5,535	3,671	828	357
	26	†	†	†	†
	1	176	143	63	62
2DfootingUBS	7	202	170	74	71
	23	†	†	11,401	3,708
	1	473	362	88	19
2DtunnelLBS	7	750	604	89	19
	35	†	†	†	†
	1	67	62	23	17
2DtunnelUBS	8	157	388	39	18
	22	19,472	†	4,442	1,700
	1	738	253	102	37
3DsqrexcLBS	9	4,740	1,382	312	92
	16	†	†	11,432	3,904
	1	485	2,656	192	103
3DsqrexcUBS	10	1,256	7,401	404	256
	19	†	†	9,707	6,566
	1	110	507	46	18
3DsqrexcUB2S	8	500	1,951	111	63
	38	†	†	6,923	2,526
	1	47	38	17	13
3DsqrfootLBS	11	†	†	57	39
	19	†	†	†	†
	1	19	22	15	15
3DsqrfootUBS	12	94	239	52	36
	20	3,137	8,473	1,646	869
	1	11	13	8	8
3DsqrfootUB2S	11	72	63	20	12
	21	2,718	3,400	968	196
	1	†	†	†	†
3DtunheadLBS	12	†	†	†	†
	21	†	†	†	†
	1	53	170	28	22
3DtunheadUBS	13	499	1,958	181	63
	25	19,403	20,000	6,825	1,725

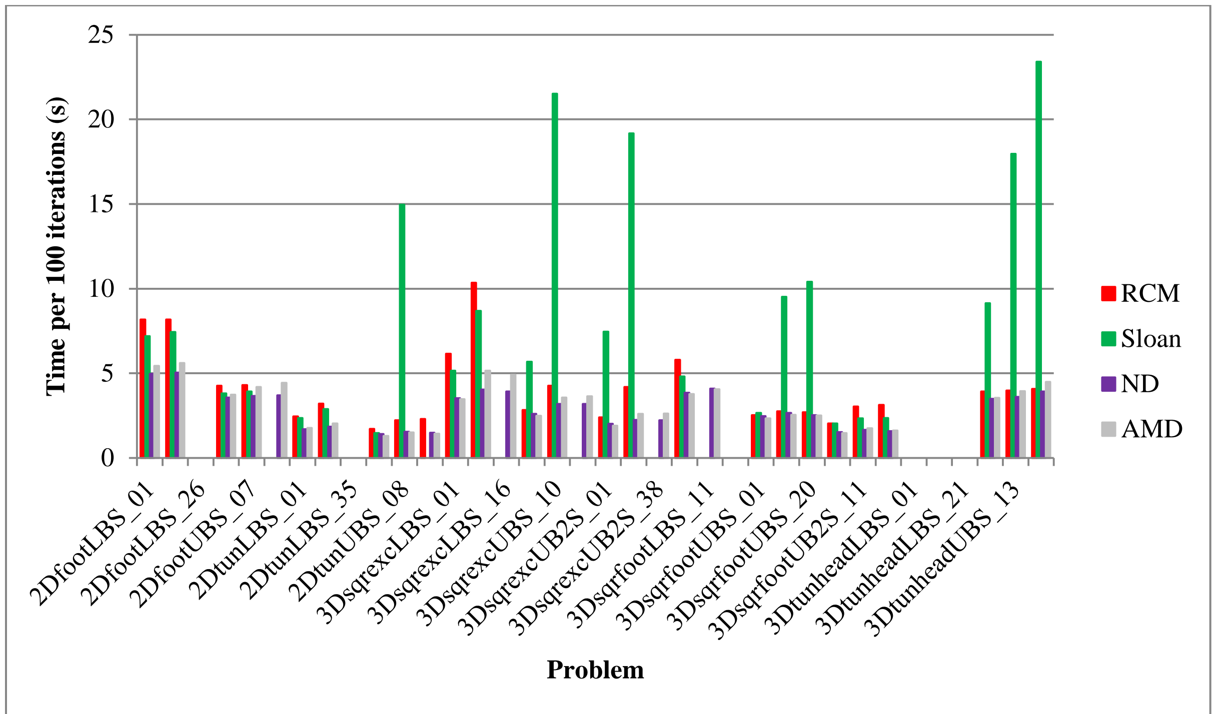


Figure 41. Time per 100 iterations with RIC1 ($\tau = 10^{-4}$) preconditioned CG.

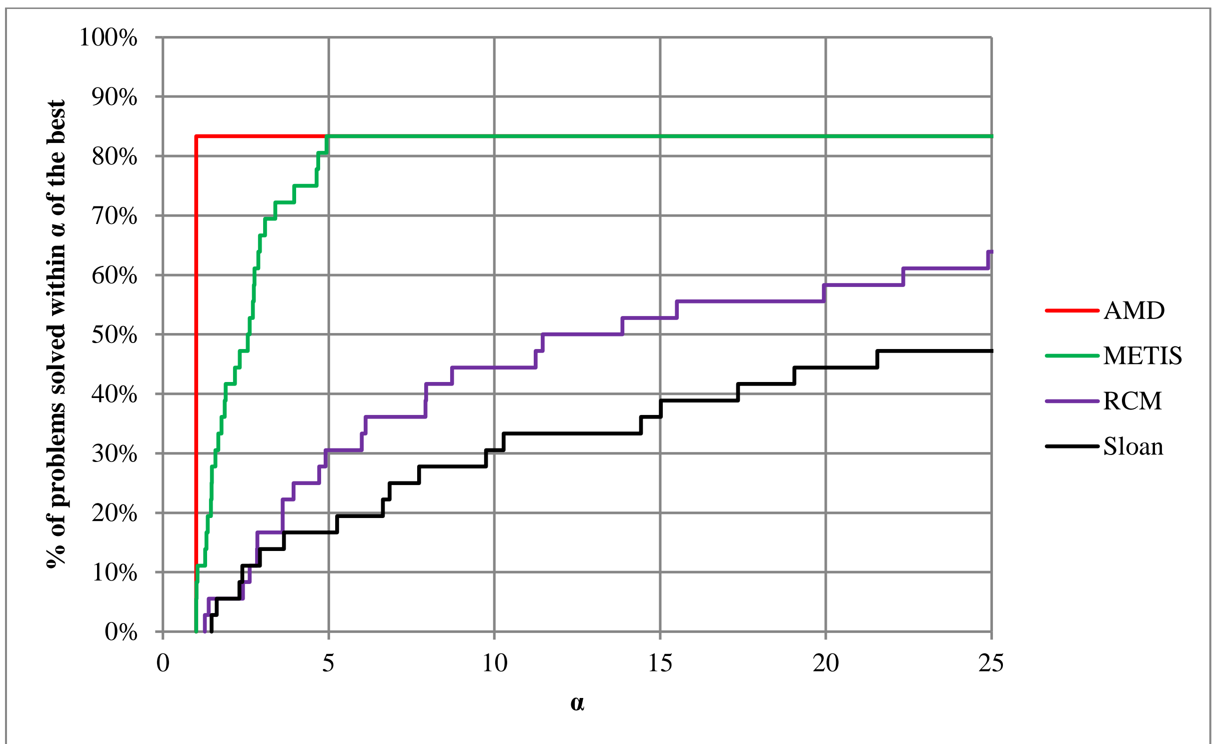


Figure 42. Performance profile of iteration counts by ordering method with RIC1 ($\tau = 10^{-4}$). Note that the RCM and Sloan ordering profiles extend beyond an Alpha value of 25 but are not shown.

4.1.3.3 Incomplete factorisation comparison

Incomplete Cholesky factorisations are the most common form of preconditioners for SPD systems [78]. Because of their popularity, many developments have been made under the incomplete Cholesky umbrella. The variants tested here were a dual-threshold incomplete Cholesky, ICTP, that controls fill through both a drop tolerance and fill-control, a robust incomplete Cholesky, RIC1, by Ajiz and Jennings [217] that ensures the incomplete factor exists for any SPD system, and a second order stabilised incomplete Cholesky, RIC2S, by Kaporin [218].

In order to allow a direct comparison between the effectiveness of the different approaches, these three incomplete factorisation methods were all implemented based on the left-looking Cholesky factorisation described by Davis [85]. These implementations were then compared against well-known codes from the HSL to measure their effectiveness against high performance solvers. Because of the difficulty in factorising the systems which define the search direction in the IPM (see Section 3.4.1 for a discussion on computing the full Cholesky factor), it was essential to use a diagonal shift framework to ensure that an incomplete factor could be computed. This and a general outline of the incomplete Cholesky factorisation method used are described next.

4.1.3.3.1 Implementation of the incomplete factorisations

In order to obtain the most efficient preconditioner construction and iterative solution process for each linear system, the implementation details of the various incomplete factorisation algorithms are crucial. The implementation of these algorithms also provides a more direct comparison between the different approaches, as opposed to comparing solvers from available libraries. The incomplete factorisation methods implemented include a conventional incomplete Cholesky factorisation, the Ajiz-Jennings robust incomplete factorisation [217], and Kaporin's second-order stabilised robust incomplete factorisation [218]. Following Benzi and Tůma [219], we label these methods ICT, RIC1, and RIC2S, respectively. ICT is implemented with fill-control and threshold dropping, while RIC1 and RIC2S follow the algorithms provided by Kaporin [218], except that in sparsifying the row accumulator vector in RIC2S, ζ is set to $|v_j|/\sqrt{d_i d_j}$ instead of $|v_j|/\sqrt{d_j}$ (this is a correction, see footnote on p. 392 in [219]).

The implementations also compute $\mathbf{L}$ one column at a time as opposed to one row at a time described by Kaporin [218]. All three methods are based on the same column-oriented left-looking sparse Cholesky factorisation in which the columns of $\mathbf{L}$ are stored in a CSC structure as they are computed using the concepts from Davis's up-looking Cholesky factorisation described in Reference [85]. The basic algorithm is shown below.

```
d(i) = A(i,i) ∀i = 1,...,n
head(1:n) = 0 ! Initialise header pointers for linked lists
for k = 1,n
   v = A(:,k) ! Scatter the kth column of A
   j = head(k) ! For each entry in the kth row
   do while (j ≠ 0)
      ! Column j has an entry in row k
      v := v − L(k, j)×L(k+1:n, j)
      jnext = next(j)
      ! Update linked list for next entry in column j
      j = jnext
   end do
   ! Perform any dropping, and sort column entries
   L(k,k) = √d(k)
   L(k+1:n,k) = v/L(k,k)
   d(k+1:n) := d(k+1:n) − L(k+1:n,k)^2
   ! Update next and head with first off-diagonal entry in column k of L
end do
```

Algorithm 1 - Left-looking incomplete Cholesky framework.

A linked list is used to allow the partially computed factor to be accessed by row which is necessary in the update step. The linked list comprises three n-vectors, only two of which are shown in the outline above; **head** holds the head of the linked list and **next** points to the next entry in the linked list. The third vector holds a pointer to the location in the row index and value arrays for $\mathbf{L}$, so that $\mathbf{L}(k:n,k)$ may be easily accessed if the column entries are sorted. This pointer vector is initialised for each column in the last line before the end of the main loop by checking whether there are any off-diagonal

entries in the k th column of $\mathbf{L}$, and, if there is, setting the pointer vector to the location of the first sub-diagonal in column k in the CSC structure for $\mathbf{L}$, and adding the column to the head of the linked list for the row containing this entry. Modifying the linked list is performed similarly after the update by incrementing the pointer vector by one position to the next entry in the column (the columns are sorted as described next) and inserting the column at the head of the linked list associated with the row index of that next entry. For RIC2S, the linked list is used for entries in both $\mathbf{L}$ and $\mathbf{R}^T$ (where $\mathbf{R}^T$ is the strictly lower triangular matrix containing the entries discarded after the factorisation finishes). The column index is simply negated to indicate that the entry in row k is an entry of $\mathbf{R}^T$. The initialisation and update for the linked lists must also check entries in both $\mathbf{L}$ and $\mathbf{R}^T$ to determine which holds the next off-diagonal entry in the column.

Because the sparsity pattern is not known beforehand for the incomplete factorisations, a $\mathbf{flag}$ vector is maintained that will have the i th component set to k if $\mathbf{v}(i)$ will be the non-zero entry corresponding to $\mathbf{L}(i,k)$. If fill-in is encountered in row i, $\mathbf{flag}(i)$ is set to k and $\mathbf{v}(i)$ is initialised. The use of the $\mathbf{flag}$ vector avoids checking a floating point number for equality with zero and any associated duplication of entries which have cancelled. It also avoids the need to zero out the accumulation vectors after each column has been computed. The row indices are held in unordered form during the update, with the threshold dropping being applied after the update and then the column is sorted in order of increasing row indices using quicksort. If fill-control is being used, then only the fill-control parameter times the number of non-zero entries in the lower triangular part of the k th column of $\mathbf{A}$ are stored; these entries are found using quicksplit (described by Saad in his implementation details for ILUT [136]) before being ordered with quicksort. After the dropping and sorting is performed, the diagonal entry is first checked for non-positivity. If the entry is no longer positive, the factorisation halts, and then restarted after performing a diagonal shift so that $\mathbf{A} := \mathbf{A} + \alpha \operatorname{diag}(\mathbf{A})$ as suggested by Manteuffel [220] or $\mathbf{A} := \mathbf{A} + \beta \mathbf{I}$. In all cases the second approach with β was found to provide better performance and was used as follows:

1. Set β to zero and attempt to construct the incomplete factor. If successful, exit.
2. If β is zero, then set it to a small value such as 10^{-12} or 10^{-16}, otherwise set β to its square root.
3. Return to step 1.

This process is modified slightly such that if the incomplete factorisation fails around a similar pivot in consecutive attempts, then the square root is taken twice, i.e. $\beta := \sqrt[4]{\beta}$.

4.1.3.3.2 Incomplete Cholesky

The ICTP preconditioner was tested with a range of drop tolerances from $\tau = 10^{-2}$ to 10^{-4} and a fill control from $p = 20$ to 80. In addition, ICTP was tested without a fill control. This meant that whenever the factorisation needed more space, it attempted to allocate memory and transfer the previously computed portion of the preconditioner. This approach did not fail on any of the problems tested, but is likely to require modification if the initial allocation is too small when solving very large systems with a small drop tolerance. The performance profile comparing the parameter settings is shown in Figure 44.

Clearly, introducing a fill control strongly affects the effectiveness of the preconditioner. The fill control does significantly impact the preconditioner with $\tau = 10^{-3}$ until p is reduced to 20, and none of the fill control values has an impact when $\tau = 10^{-2}$. As expected, the use of fill control does not result in an improvement in any case. Furthermore, it appears that rather than introducing a fill control to reduce the number of entries in the factorisation one should increase the drop tolerance. For example, rather than introducing a fill control of 40 to limit the number of entries in the preconditioner with $\tau = 10^{-4}$, increasing $\tau = 10^{-3}$ will generally result in fewer entries, faster factorisation time, and a more effective preconditioner, thus dominating the smaller drop threshold with a fill control.

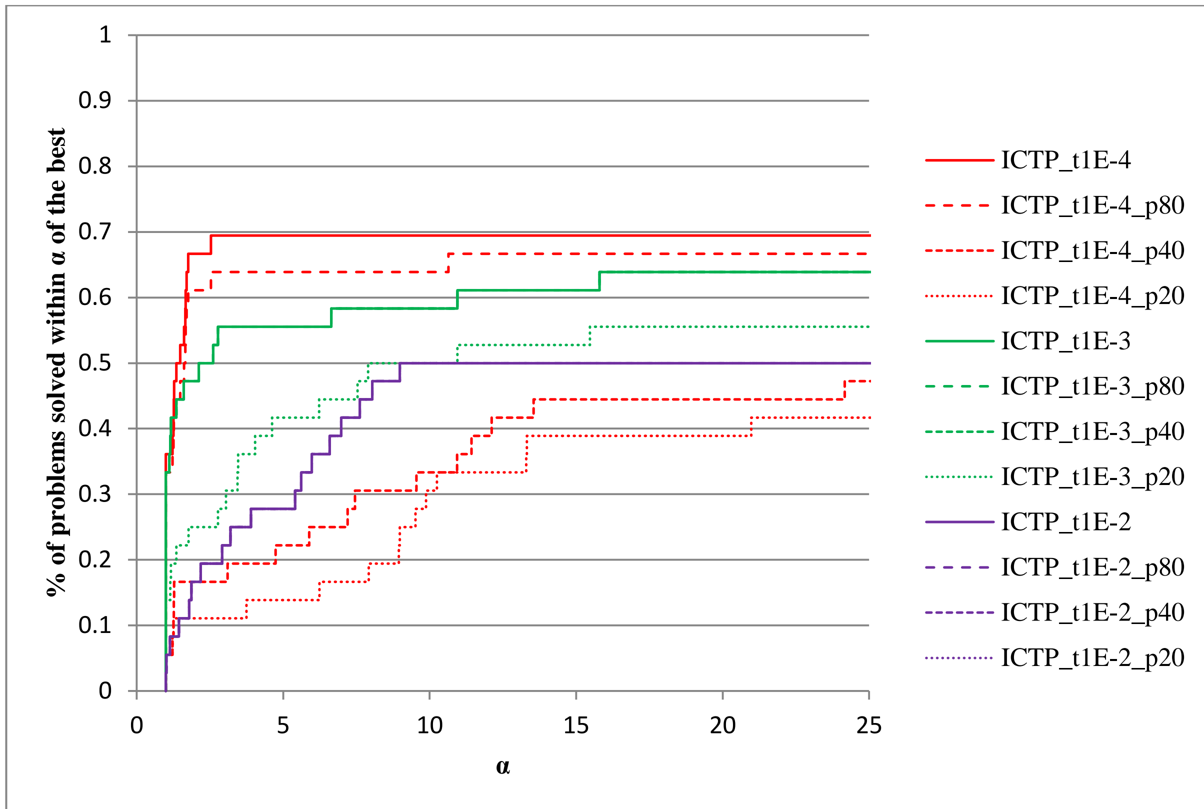


Figure 43. Performance profile of efficiency by ICTP parameter choice.

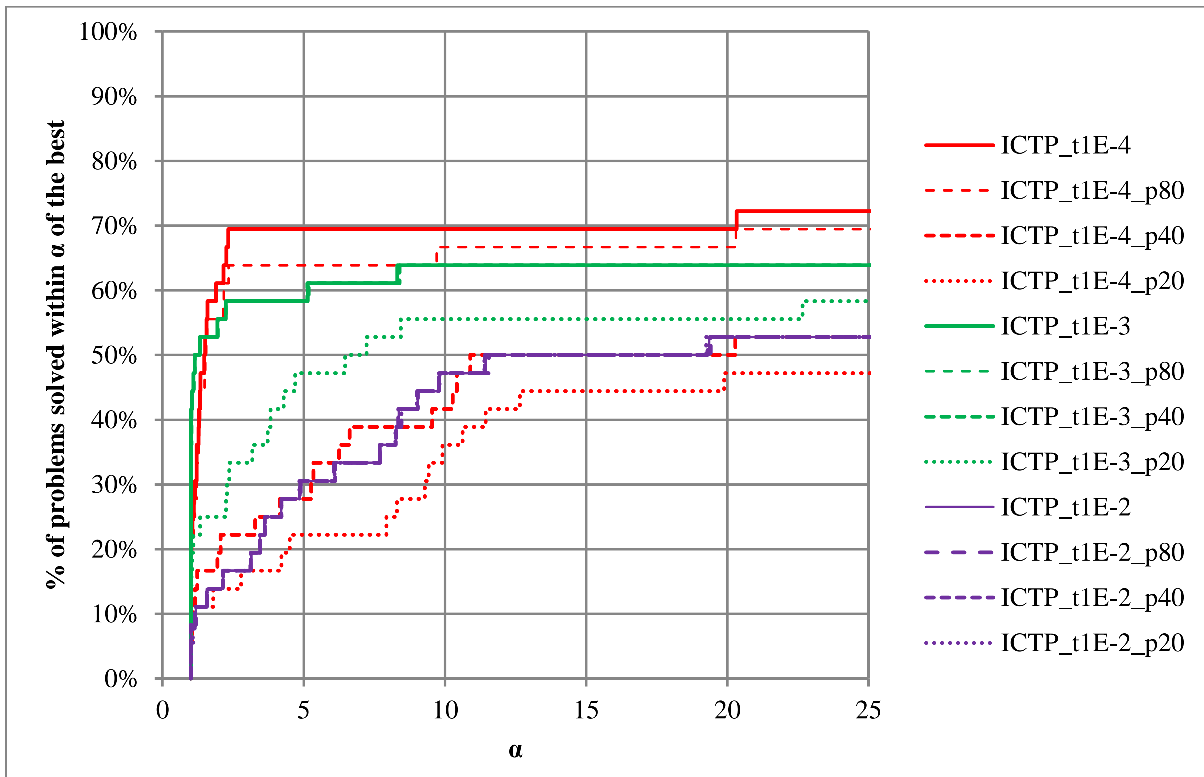


Figure 44. Performance profile of factor time plus $3\times$ solve time by ICTP parameter choice.

The performance profile of efficiency is given in Figure 43 and the runtime in Figure 44, with each preconditioners drop tolerance shown with the prefix *t* and the fill control with a prefix *p*. The figures show that building the preconditioner with $\tau = 10^{-3}$ leads to better performance than with either the higher or lower drop tolerance for more than 50% of the problems tested. This suggests that an optimal tradeoff between preconditioner accuracy and fill-in exists, but that as the IPM approaches a solution it is likely to be beneficial to reduce the drop tolerance to increase the effectiveness of the preconditioner.

4.1.3.3.3 Robust incomplete Cholesky

The RIC1 method differs from the ICTP solver in that the drop tolerance of 10^{-3} does not appear to hold any advantage over the lower value of 10^{-4} for any of the problems tested as seen in the performance profile of expected per IPM iteration runtime in Figure 46, which includes three solves and the factorisation. This difference lies in the increase of the diagonal perturbations as more entries are dropped, reducing the effectiveness of the preconditioner.

It should be noted that not all of the systems tested required diagonal modifications to ensure the existence of the incomplete factor, yet the RIC1 solver has no way of exploiting this. Consequently, the RIC1 solver operates at a significant disadvantage to the ICTP solver on any system in which the incomplete factor exists without modifying the diagonal. Furthermore, because RIC1 modifies the diagonal whenever an entry is dropped, the more entries are dropped the greater the diagonal modifications. This will result in the larger drop tolerances performing much more poorly on the better-conditioned systems than would be the case if the impact of these diagonal modifications could be avoided. To achieve this, it is possible to scale the diagonal modifications by a fixed factor, ν. By starting with $\nu = 0$ and increasing ν whenever the factorisation fails because of a non-positive pivot (to a maximum of 1), the preconditioner can avoid reducing the effectiveness of the preconditioner for the better conditioned systems but avoid failure as the systems become more ill-conditioned. This approach mirrors the process used with the ICTP solver where instead of factorising $\mathbf{A}$, the system $\mathbf{A} + \alpha \operatorname{diag}(\mathbf{A})$ or $\mathbf{A} + \beta \mathbf{I}$ is factorised. This modified version of RIC1 is labelled *nuRIC1*.

The performance profiles of efficiency and runtime are shown in Figure 45 and Figure 46, where the preconditioner has its drop tolerance appended with a prefix t. These profiles show that the smaller the drop tolerance, the better the robust incomplete Cholesky preconditioner performs. This is expected for the RIC1 preconditioner, where the diagonal modifications are less when fewer entries are dropped. The use of the relaxation parameter ν clearly improved the efficiency of the preconditioner overall, but for $\tau = 10^{-4}$ solved one less of the systems. The greater efficiency of nuRIC1 over RIC1 did not coincide with a similar difference in the runtime because of the greater preconditioner construction costs.

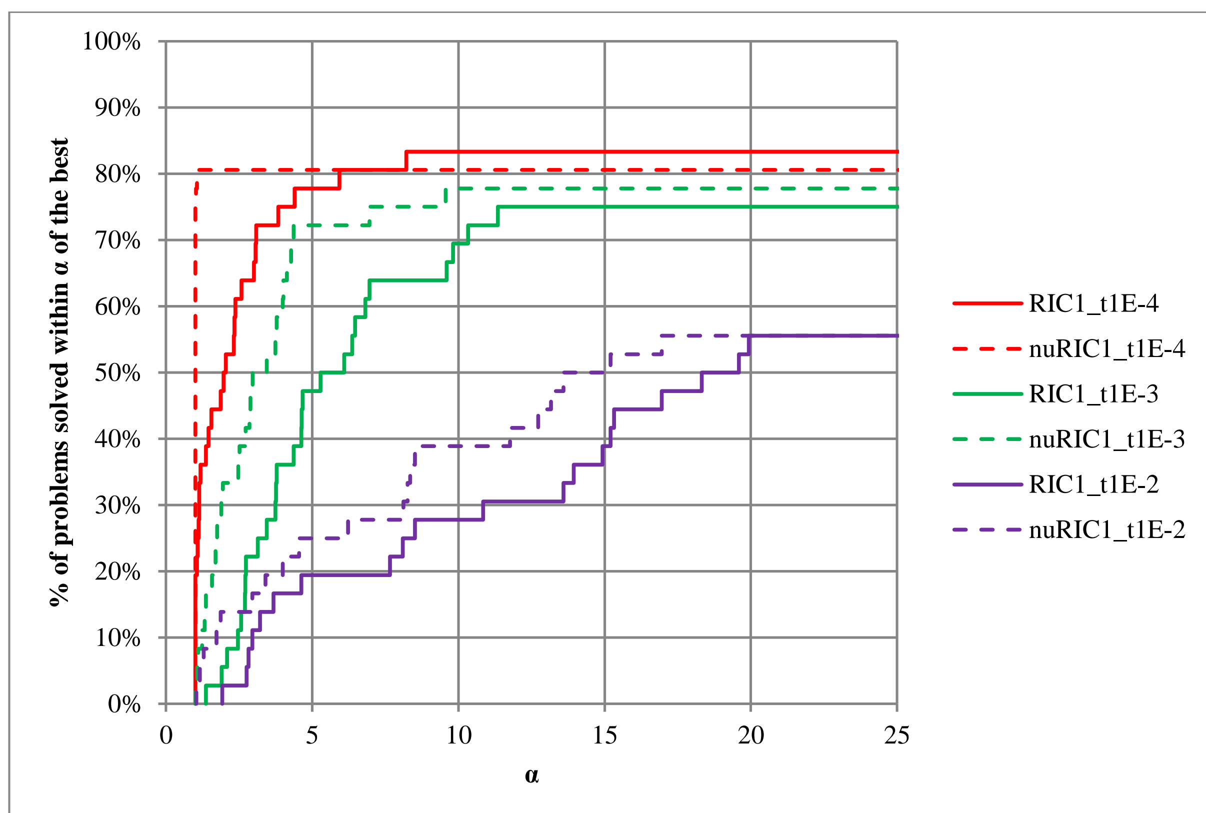


Figure 45. Performance profile of efficiency by RIC1 drop tolerance.

4.1.3.3.4 Incomplete Cholesky with second-order corrections

With preconditioner efficiency defined as the iteration count multiplied by the number of non-zeros in the incomplete factor, the second-order incomplete Cholesky preconditioner will generally produce highly efficient preconditioners. This does hide, however, the often large intermediate storage that is necessary to build the preconditioner. For this reason, no efficiency profiles are shown, just the runtime profiles without the diagonal modifications with dropped entries in Figure 47 and with

stabilisation in Figure 48. The two flavours of the preconditioner are labelled RIC2 and RIC2S for with and without the diagonal perturbations, respectively. Again, the drop tolerances are shown prefixed with t and the fill control for the second-order update matrix $\mathbf{R}$ is prefixed with r.

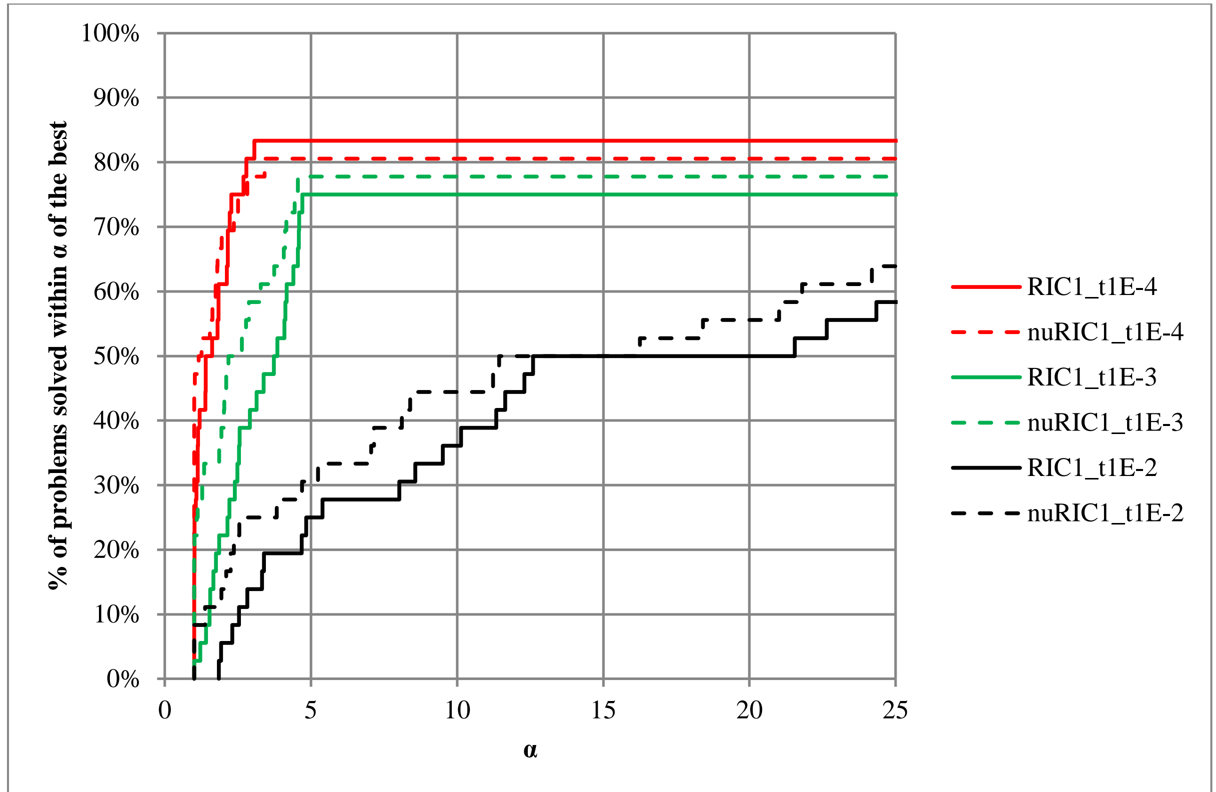


Figure 46. Performance profile of factor plus $3\times$ solve time by RIC1 drop tolerance.

As with ICTP and RIC1, it is clear that the smaller the drop tolerance, the more effective the preconditioner in terms of both runtime and the number of systems solved. There appears to be little difference between the two high fill control values of $p=40$ and 80 for $\tau=10^{-3}$, but the tighter fill controls adversely impact the preconditioner quality on a number of the systems. For the largest drop tolerance of $\tau=10^{-1}$, the fill control makes negligible difference for RIC2. For RIC2S, this effect is apparent for both $\tau=10^{-1}$ and 10^{-2}.

Comparing the preconditioner with and without stabilisation in Figure *49* gives a clear indication that the diagonal perturbations for dropped entries in RIC2S makes a significant improvement over RIC2 for the problems in the test set.

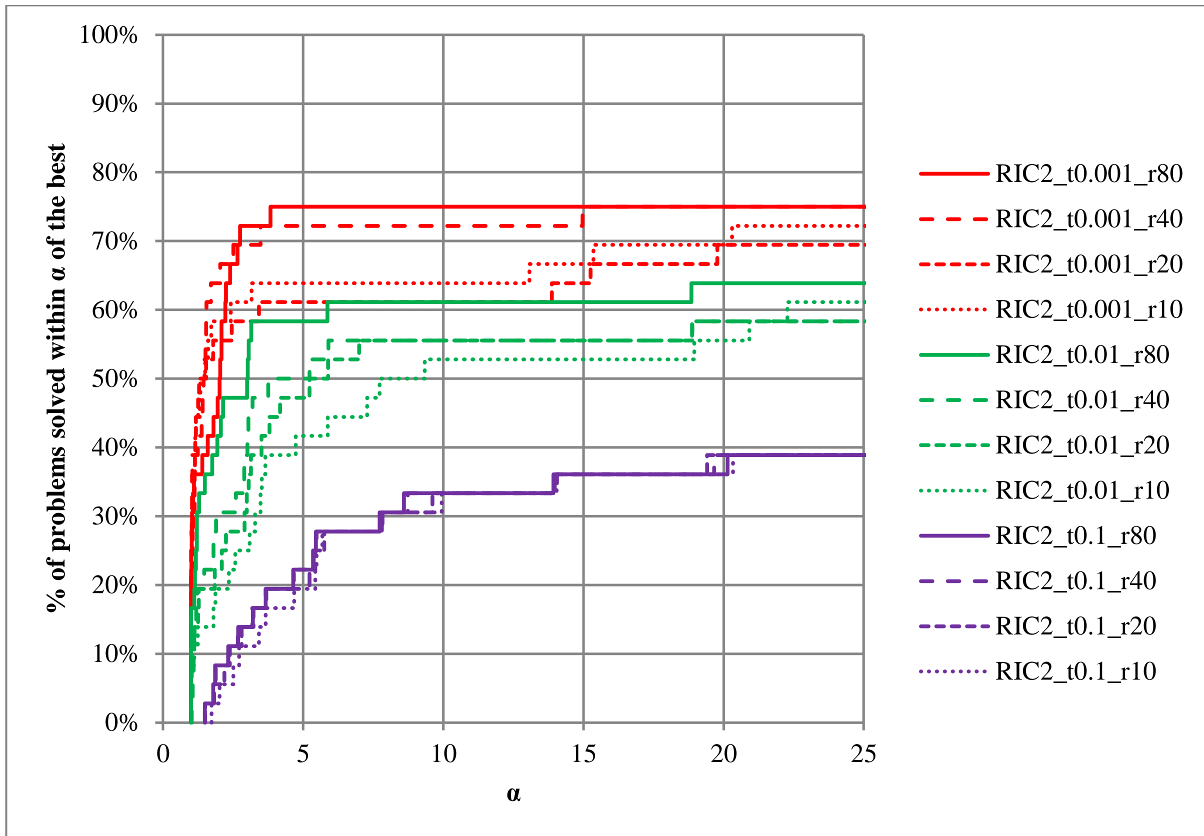


Figure 47. Performance profile of factor plus $3\times$ *solve time by RIC2 parameter choice.*

4.1.3.3.5 Comparison with available incomplete Cholesky packages

A comparison between these implementations and those from some highly-regarded libraries was performed. The previously discussed incomplete Cholesky preconditioners are compared with the incomplete Cholesky factorisations constructed by the `MA61` and `MI28` packages in the HSL. HSL's `MA61` is a relatively conventional right-looking incomplete Cholesky factorisation, while `MI28` is a state-of-the-art second-order stabilised robust incomplete Cholesky implementation similar to the RIC2S implementation used above. The `MA61` and `MI28` packages are both available in source form. `MA61` is a right-looking $\mathbf{LDL}^T$ factorisation. The incomplete factorisation in `MI28` is a left-looking Cholesky implementation using linked lists of ancestors for each node in the elimination tree, and attempts to minimise the diagonal shift parameter, α, that allows an incomplete factorisation to be computed by increasing/decreasing α and recomputing the factorisation. While `MI28` has many options, the default values for all parameters except τ, p, and r were used. The second drop threshold for entries dropped from $\mathbf{L}$ and not to be included in $\mathbf{R}$ was set at $\sqrt{\tau}$. This implementation can

be expected to achieve very similar performance to the RIC2S implementation developed here, while MA61 is compared only on the more easily-solved systems as its diagonal perturbation approach to non-positive pivots is insufficient to effectively precondition many of the numerically harder systems in the test set. These results are displayed in Table 24.

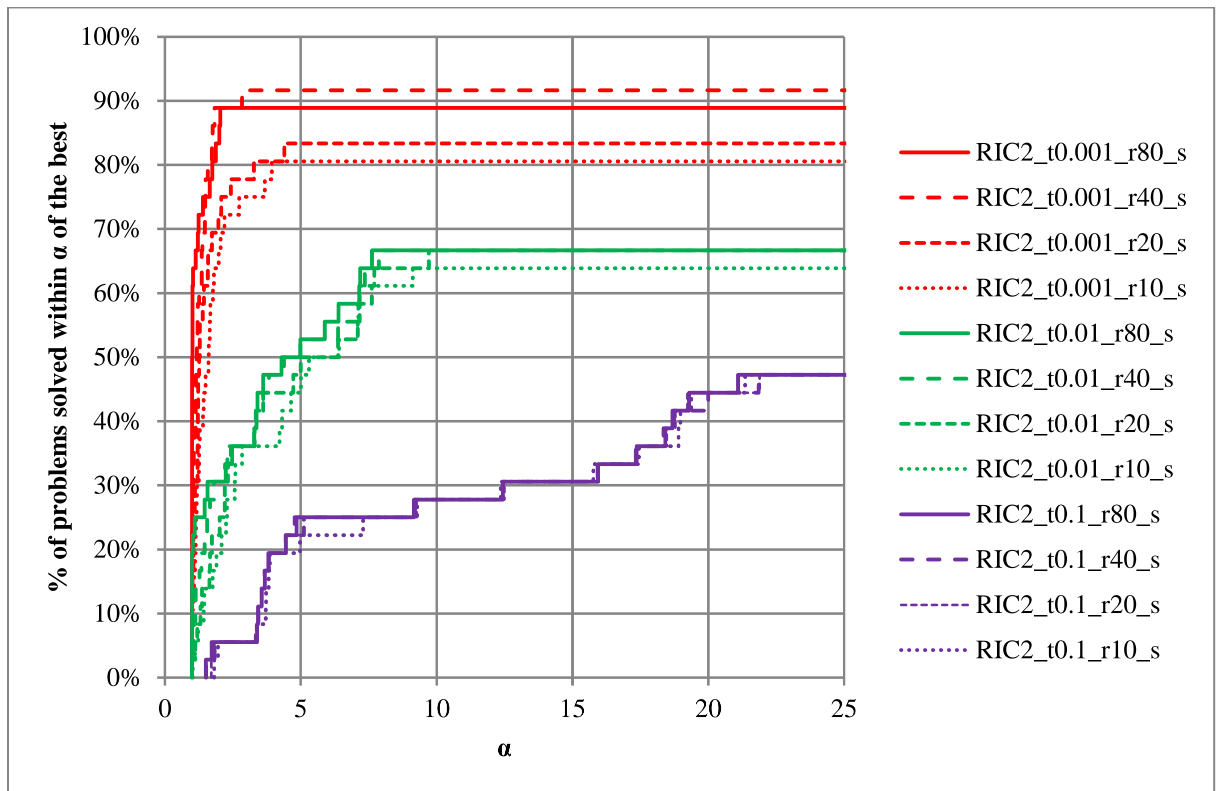


Figure 48. Performance profile of factor plus $3\times$ solve time by RIC2S parameter choice.

On these problems, the ICTP implementation described here dominates MA61 in factorisation time and solve time, being faster for all but one of the systems (*3DsqrfootUB2_01*, with 0.10s and 0.09s being the respective solve times). For the same drop tolerance, fill-in is roughly the same for the middle IPM iteration systems as expected, but the first system creates more fill-in with ICTP. Even with greater fill-in, ICTP is a significantly faster factorisation. Furthermore, for a given drop tolerance, ICTP builds a very competitive and effective preconditioner as measured by iteration counts and solve times. For example, solving *3DsqrexcUB2_01* with $\tau = 10^{-3}$ leads to approximately the same amount of fill-in between the two methods, but the ICTP preconditioner takes roughly a quarter of the time to build and requires only 16

iterations to converge compared with `MA61`'s 263 iterations. `MA61` also has some unexpectedly large iteration counts for $\tau = 10^{-3}$ on *3DsqrexcUB2_08* and *3DsqrfootUB_12*. Based on these results, `MA61` was not considered any further in this study.

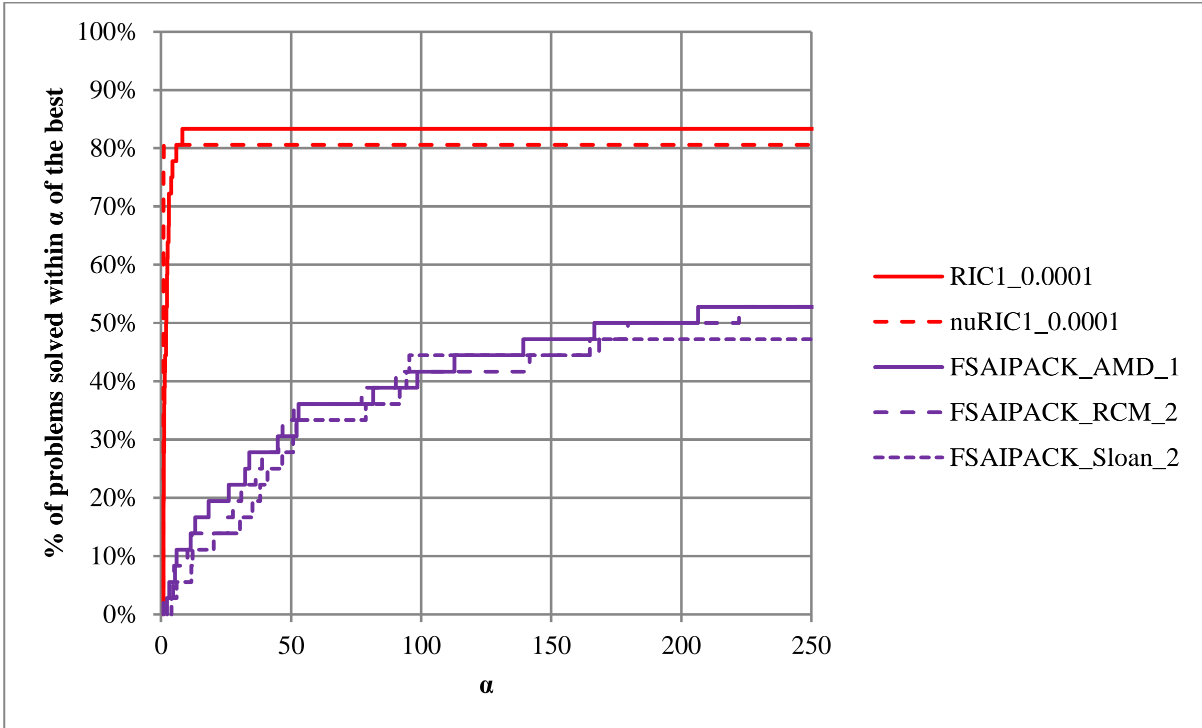


Figure 49. Performance profile of factor plus $3\times$ solve time with RIC2 and RIC2S for some of the more accurate parameter settings.

MI28 was tested both with and without the no-fill updates from $\mathbf{RR}^T$, and the performance profiles are shown without the updates in Figure 50 and in Figure 51 with the updates. The figures appear almost identical, and this is confirmed in Figure 52 where $\tau = 10^{-3}$ with an $\mathbf{L}$ fill control of 40 and $\mathbf{R}$ fill controls of 40 and 80. The same behaviour as with the implemented preconditioners exists here, with a reduced drop tolerance leading to more systems solved in less time.

The top performing parameter settings for ICTP, RIC1, nuRIC1, and RIC2S are compared with the top performing MI28 preconditioner in Figure 53. The nuRIC1 preconditioner appears to be the best performing approach for 75% of the problems in the test set. The RIC1 and RIC2S solve more problems than nuRIC1, but at a slight performance disadvantage. ICTP is clearly not competitive, and the MI28 preconditioner does not perform as well as the other preconditioners tested. Table 25

shows the number of non-zeros in the incomplete factorisation, including the intermediate $\mathbf{R}$ entries for the second-order methods. When it converges, ICTP generally has the lowest non-zero count. Note that the different drop criterion in RIC1 (drop v_{ij} if $v_{ij} \leq \tau\sqrt{d_i^{(j)} d_j^{(j)}}$, where v_{ij} is an entry that is yet to be divided by the diagonal d_j in its column when computing column j) causes it generally to have more non-zeros than ICTP (drop v_{ij} if $\left|v_{ij}\right| \leq \tau\sqrt{d_j^{(j)}}$) for the same drop threshold. For most systems, the RIC1 and nuRIC1 preconditioners with $\tau = 10^{-4}$ are of similarly size to the second-order methods with $\tau = 10^{-3}$ and $\mathbf{R}$ containing 40 to 80 entries per column.

Table 24. Comparison of MA61 and ICTP on some upper bound 3D systems. Both methods are using the same AMD ordering.

System	Prec	beta	p	tau	L_{nz}	nit	||r||	t_{factor}	t_{solve}
3DsqrexcUB2_01	ICTP	0		0.001	3,107,095	16	8.187E-09	0.49	0.20
3DsqrexcUB2_01	MA61		8	0.001	3,070,004	263	7.99E-09	1.93	3.19
3DsqrexcUB2_01	ICTP	0		0.0001	6,159,195	7	3.651E-09	1.93	0.14
3DsqrexcUB2_01	MA61		8	0.0001	4,884,377	51	9.45E-09	6.70	0.85
3DsqrexcUB2_08	ICTP	0.001		0.001	4,648,726	262	9.648E-09	1.74	3.87
3DsqrexcUB2_08	MA61		8	0.001	5,029,849	11320	9.811E-09	6.80	186.60
3DsqrexcUB2_08	ICTP	0.001		0.0001	8,866,606	187	9.364E-09	13.35	3.99
3DsqrexcUB2_08	MA61		8	0.0001	9,751,167	189	9.997E-09	31.92	5.02
3DsqrfootUB_01	ICTP	0		0.001	3,629,324	17	8.293E-09	0.44	0.22
3DsqrfootUB_01	MA61		8	0.001	3,334,297	16	4.194E-09	1.93	0.25
3DsqrfootUB_01	ICTP	0		0.0001	7,885,251	7	6.591E-09	1.87	0.17
3DsqrfootUB_01	MA61		8	0.0001	6,052,946	8	4.853E-09	7.12	0.21
3DsqrfootUB_12	ICTP	0.001		0.001	4,231,361	71	6.794E-09	1.16	1.02
3DsqrfootUB_12	MA61		8	0.001	4,412,107	1635	9.945E-09	2.84	31.71
3DsqrfootUB_12	ICTP	0		0.0001	8,833,194	13	3.094E-09	2.22	0.33
3DsqrfootUB_12	MA61		8	0.0001	7,273,197	34	9.635E-09	9.12	0.91
3DsqrfootUB2_01	ICTP	0		0.001	2,306,657	10	1.517E-09	0.36	0.10
3DsqrfootUB2_01	MA61		8	0.001	2,324,502	10	3.123E-09	1.46	0.09
3DsqrfootUB2_01	ICTP	0		0.0001	4,264,378	4	9.746E-09	1.20	0.06
3DsqrfootUB2_01	MA61		8	0.0001	3,610,849	6	1.492E-09	4.41	0.08
3DsqrfootUB2_11	ICTP	0		0.001	3,297,940	25	5.05E-09	0.49	0.29
3DsqrfootUB2_11	MA61		8	0.001	3,523,282	135	8.622E-09	2.19	1.64
3DsqrfootUB2_11	ICTP	0		0.0001	5,560,940	7	9.597E-09	1.65	0.12
3DsqrfootUB2_11	MA61		8	0.0001	5,154,269	12	1.903E-09	7.05	0.20

There are large discrepancies, however, on *3DsqrexcLB* where the second-order methods have around half the non-zeros of RIC1 and nuRIC1, and on *3DtunheadUB*, where the opposite is the case (although, in both cases, RIC1 and nuRIC1 outperformed the more sophisticated preconditioners in runtime performance).

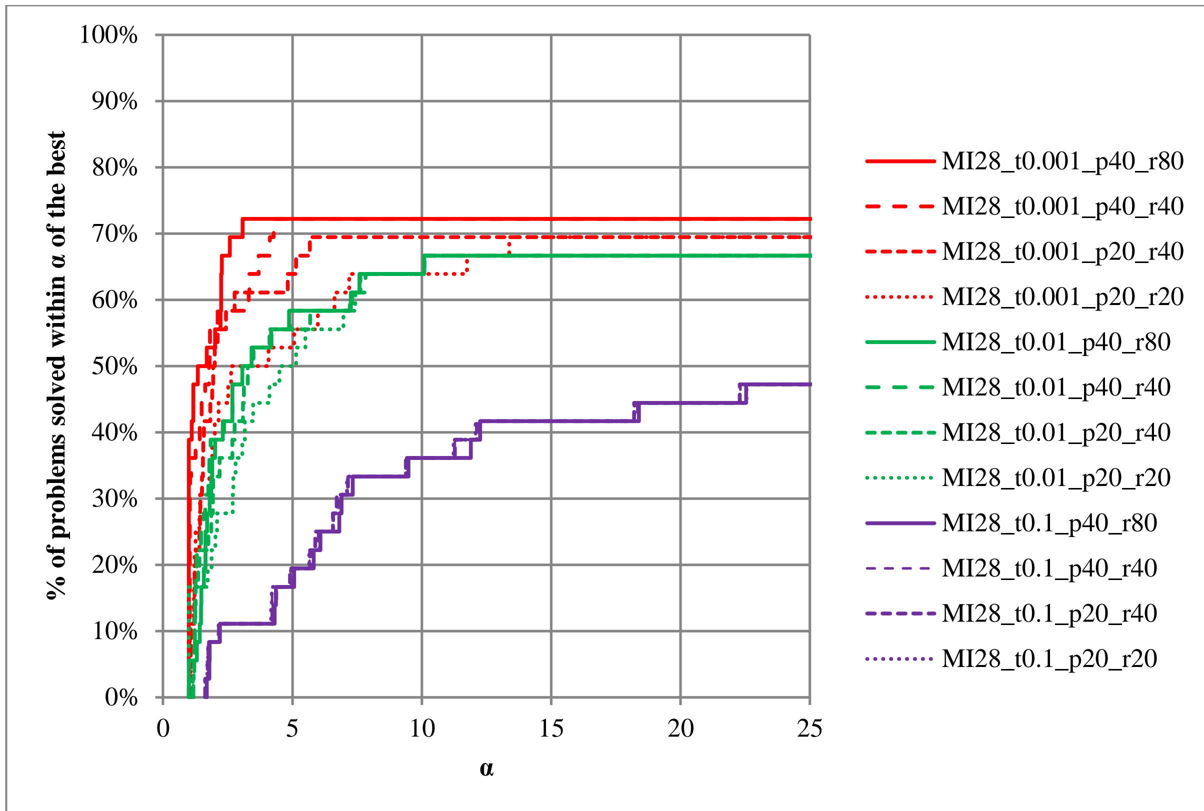


Figure 50. Performance profile of factor plus $3\times$ solve time with `MI28` *by parameter choice.*

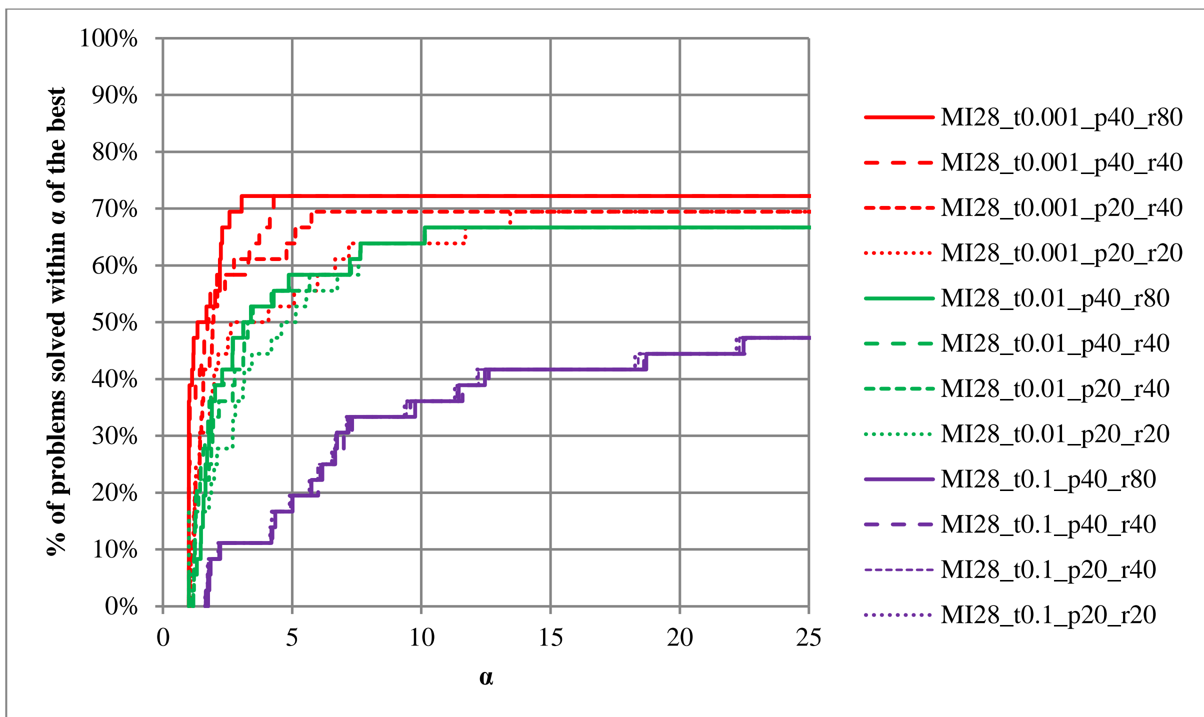


Figure 51. Performance profile of factor plus $3\times$ solve time with `MI28` *by parameter choice with no-fill* $\mathbf{RR}^T$ *updates.*

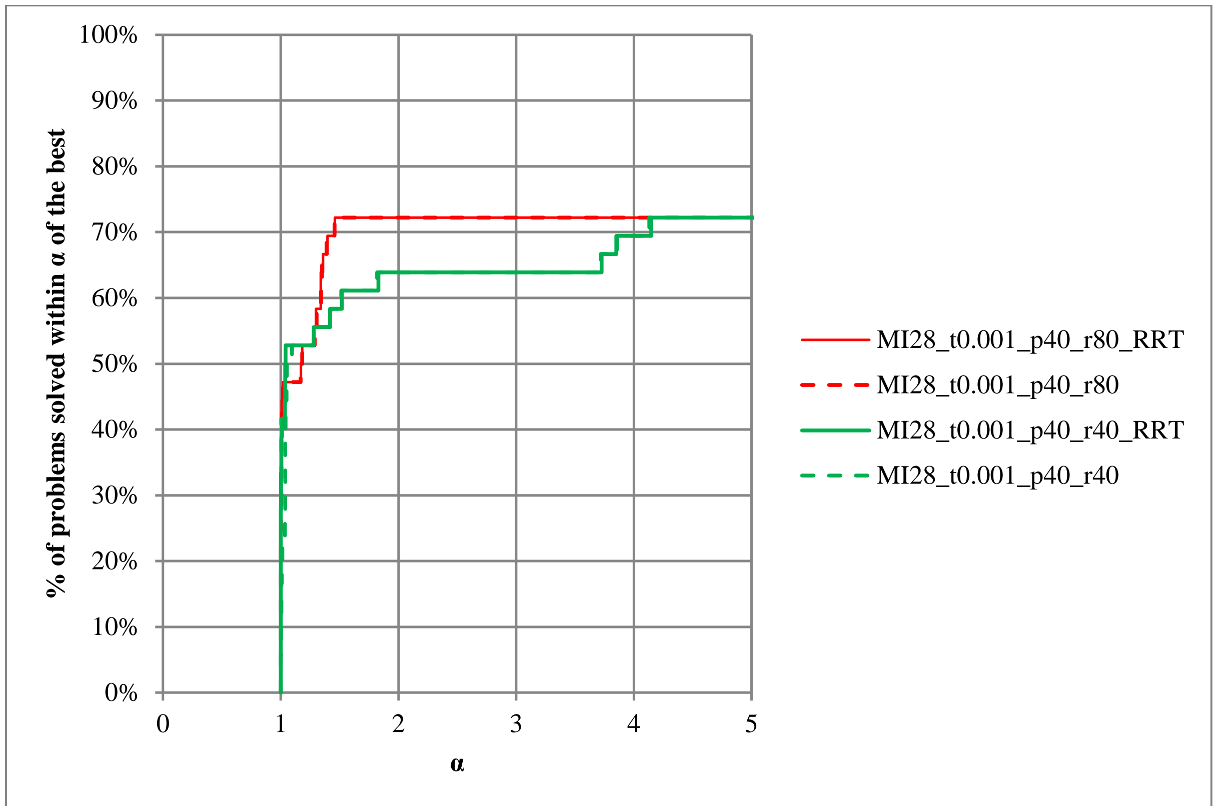


Figure 52. Performance profile of factor plus $3\times$ solve time with MI28 comparing with and without no-fill $\mathbf{RR}^T$ updates.

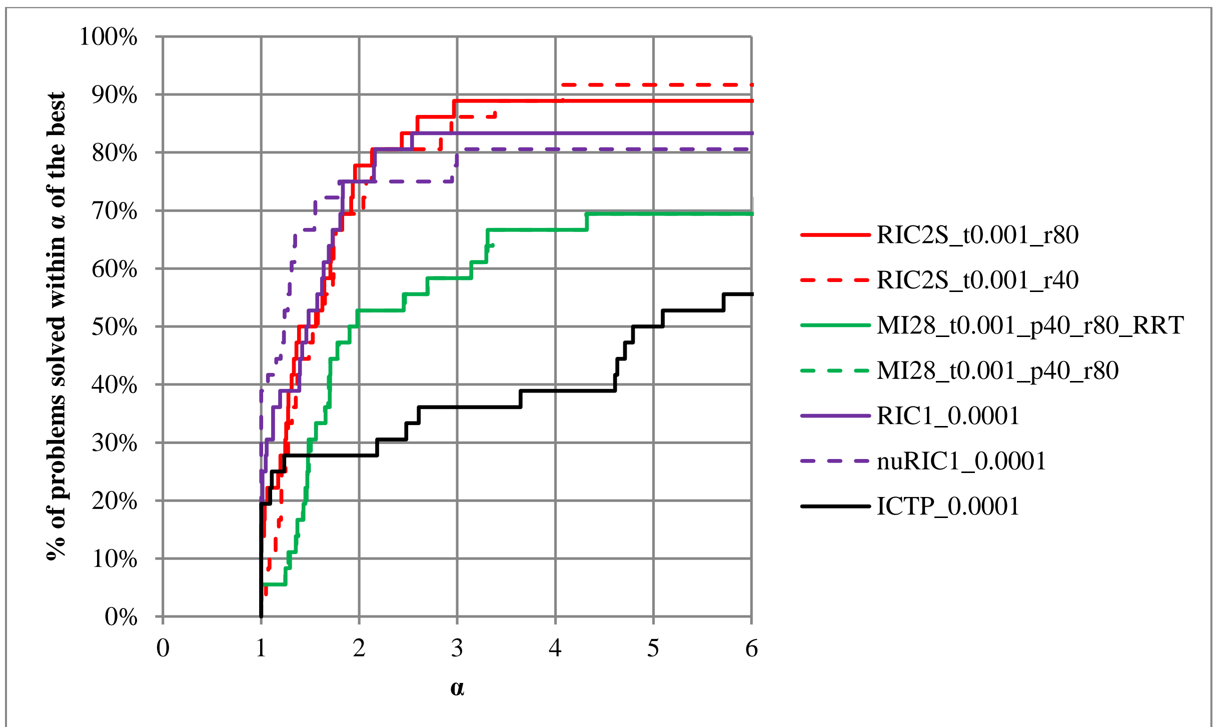


Figure 53. Performance profile of factor plus $3\times$ solve time with best incomplete Cholesky preconditioners.

From these tests, it is obvious that the standard incomplete Cholesky preconditioner, even with quite small drop tolerances and diagonal shifts, is not sufficiently robust to compute the search direction in an IPM consistently. It is difficult to draw general conclusions about the RIC1, nuRIC1, and RIC2S preconditioners, with mixed rankings across the problem set.

4.1.3.4 Sparse approximate inverses

The factorised sparse approximate inverse (FSAI) method was tested using the `FSAIPACK` software package [221] designed for multicore systems. While the package implements numerous sophisticated approximate inverse features, none of these appeared to offer any advantage over the basic method during preliminary testing. The same four re-ordering methods used for the IC factorisations were tested for `FSAIPACK`, with the sparsity pattern for the approximate inverse being either the first, second, or third power of $\mathbf{A}$.

The performance profiles of efficiency and runtime for the FSAI preconditioners tested are shown in Figure 54 and Figure 55. The FSAI-based methods solve between 50% and 65% of the problems in the test set, with little difference in performance among the orderings in general. The worst-performing methods used the dense sparsity patterns of $\operatorname{tril}\left(\mathbf{A}^3\right)$. While the efficiency profile generally favoured the sparsity pattern of the original matrix, the banded orderings with a power of two sparsity pattern achieved the best runtime performance across the widest range of the problems tested. Interestingly, the different orderings resulted in different iteration counts for the original sparsity pattern (i.e. a power of 1).

Figure 56 and Figure 57 compare the efficiency and runtime performance against some of the robust incomplete Cholesky preconditioners. The high α values in both figures are indicative of the poor relative performance of the FSAI approach on these problems, with `FSAIPACK` taking an order of magnitude or more longer than the IC preconditioners for almost all the test systems. The `FSAIPACK` preconditioners were not considered further.

Table 25. Incomplete Cholesky preconditioner non-zeros. Note that RIC2S and MI28 values include the number of entries in $\mathbf{R}$ *. System best values are in bold.*

Problem	Iterat	RIC2S t1E-3 r80	RIC2S t1E-3 r40	MI28 RRT t1E-3 p40 r80	MI2 t1E-3 p40 r80	RIC1 t1E-4	nuRIC1 t1E-4	ICTP t1E-4
2DfootingLB	1	48,181,944	29,567,544	47,763,010	29,148,610	**18,257,763**	**18,257,763**	-
	5	48,460,693	29,846,293	47,937,439	29,323,039	**19,015,901**	**19,015,901**	-
	26	-	-	-	-	-	-	-
2DfootingUB	1	35,863,514	21,902,714	35,697,251	21,736,451	12,097,548	12,426,641	**10,094,413**
	7	36,563,155	22,602,355	36,252,173	22,291,373	13,717,719	14,068,063	**10,870,684**
	23	36,053,109	22,092,309	35,439,227	21,478,427	**16,737,184**	**16,737,184**	-
2DtunnelLB	1	20,266,670	11,794,710	20,223,439	11,751,479	4,881,983	4,915,762	**4,177,734**
	7	20,987,352	12,515,392	20,910,020	12,438,060	5,901,792	5,904,992	**4,996,552**
	35	-	-	-	-	-	-	-
2DtunnelUB	1	17,969,636	10,283,276	17,999,158	10,312,798	3,454,875	3,488,087	**3,342,097**
	8	18,524,172	10,837,812	18,546,363	10,860,003	4,155,104	4,145,446	**3,850,849**
	22	18,352,489	10,666,129	18,345,865	10,659,505	**4,247,076**	**4,247,076**	-
3DsqrexcLB	1	19,069,324	12,414,480	18,091,705	11,487,151	12,500,731	13,143,543	**10,741,933**
	9	20,410,118	13,759,088	18,493,103	**11,884,427**	22,932,096	23,160,451	14,669,871
	16	19,863,716	**13,241,902**	-	-	22,486,943	22,505,497	-
3DsqrexcUB	1	18,983,144	11,346,357	18,473,866	10,912,090	7,889,490	8,244,551	**7,100,785**
	10	20,680,025	12,988,199	19,546,404	11,959,050	14,221,679	14,498,513	**10,930,285**
	19	-	12,924,519	-	-	15,586,904	15,638,940	**11,559,253**
3DsqrexcUB2	1	9,772,718	6,395,018	9,127,936	**5,795,155**	6,516,542	6,734,619	6,159,195
	8	11,152,032	7,736,616	9,776,503	**6,373,640**	10,543,303	10,769,277	8,866,606
	38	10,820,472	**7,516,326**	-	-	11,237,777	11,287,605	8,804,988
3DsqrfootLB	1	18,185,705	11,965,697	17,220,198	11,072,771	12,185,233	12,185,233	**10,759,061**
	11	19,172,459	12,947,414	17,894,544	11,748,154	15,724,039	-	**9,100,987**
	19	18,945,357	12,724,609	-	-	-	-	**12,569,669**
3DsqrfootUB	1	13,529,897	8,566,333	12,814,436	7,934,600	8,285,553	8,667,294	**7,885,251**
	12	14,174,177	9,198,159	13,289,190	**8,409,910**	9,749,364	10,150,660	8,833,194
	20	14,106,530	9,151,021	13,304,387	**8,416,287**	9,829,626	9,971,620	8,803,047
3DsqrfootUB2	1	6,827,491	4,541,262	6,406,752	**4,138,776**	4,472,196	4,556,316	4,264,378
	11	7,808,367	5,511,313	7,236,426	**4,970,413**	6,166,816	6,242,172	5,560,940
	21	7,719,621	5,429,459	7,187,095	**4,951,597**	6,307,611	6,311,431	5,611,009
3DtunheadLB	1	30,020,565	20,027,864	-	-	-	-	-
	12	31,337,618	21,362,894	-	-	-	-	-
	21	-	-	-	-	-	-	-
3DtunheadUB	1	20,839,358	13,109,129	19,948,353	12,335,895	11,616,517	12,124,717	**10,904,381**
	13	21,426,661	13,687,739	20,302,463	12,691,146	14,119,504	15,007,448	**11,618,902**
	25	21,554,924	13,880,374	-	-	18,815,249	18,890,351	**12,982,614**

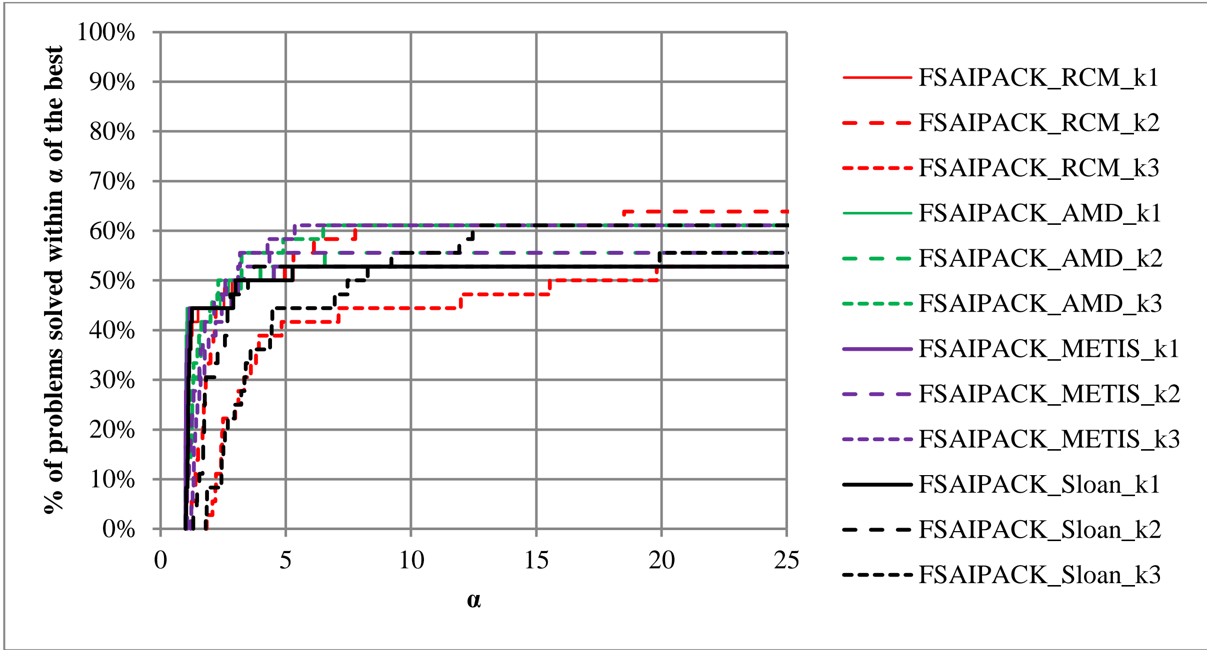


Figure 54. Performance profile of `FSAIPACK` *preconditioner efficiency.*

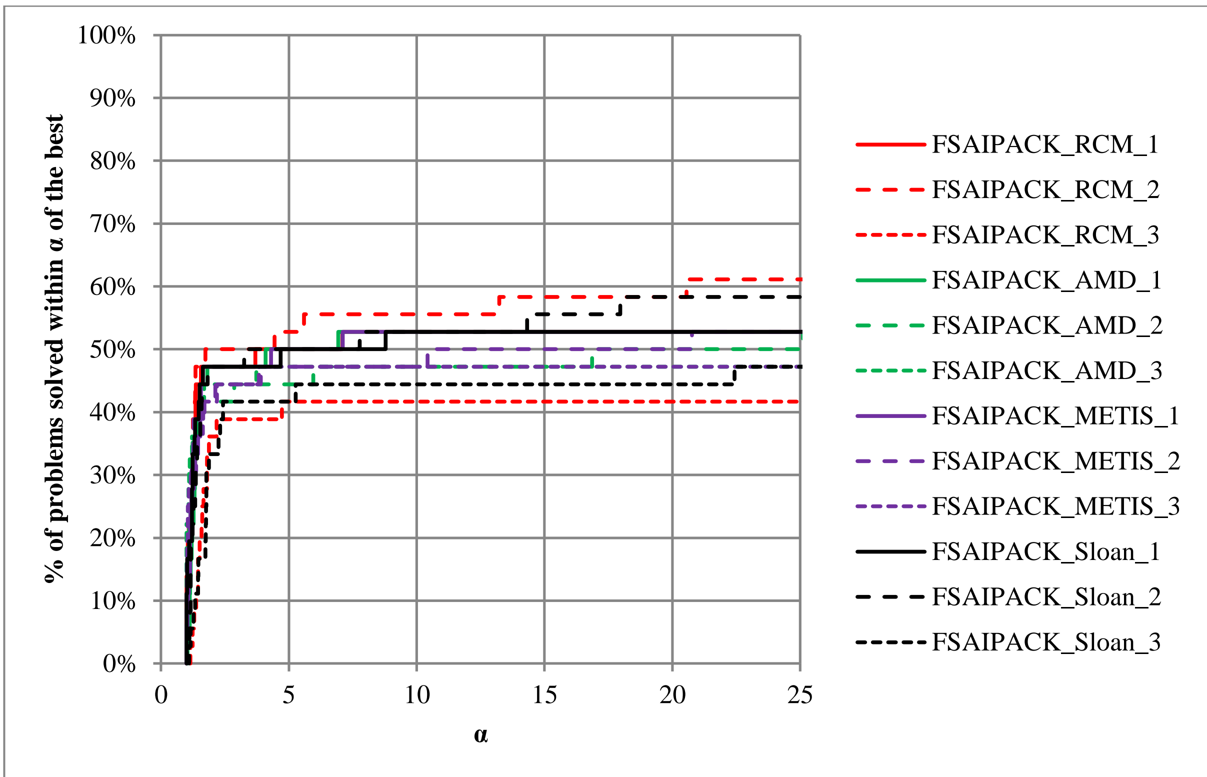


Figure 55. Performance profile of build plus $3\times$ *solve time with* `FSAIPACK` *by ordering and pattern power.*

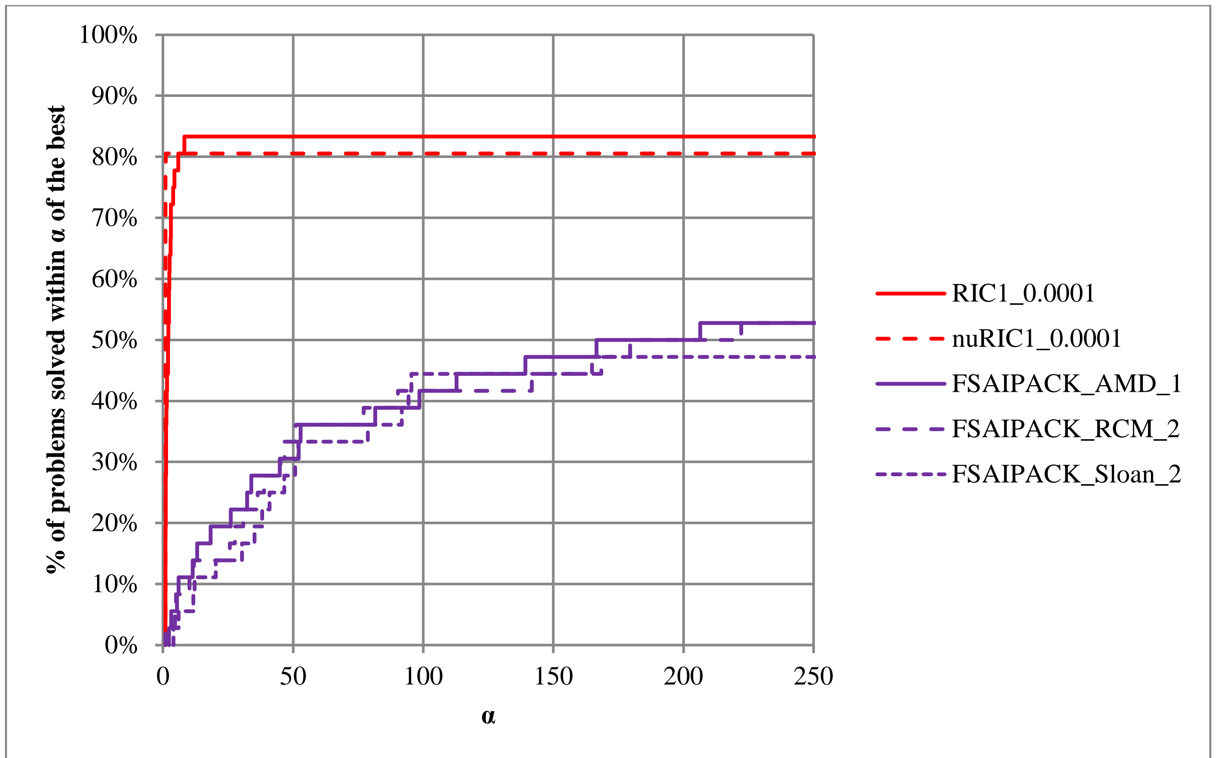


Figure 56. Performance profile comparison of efficiency between some of the robust incomplete Cholesky factorisations and the better FSAI preconditioners.

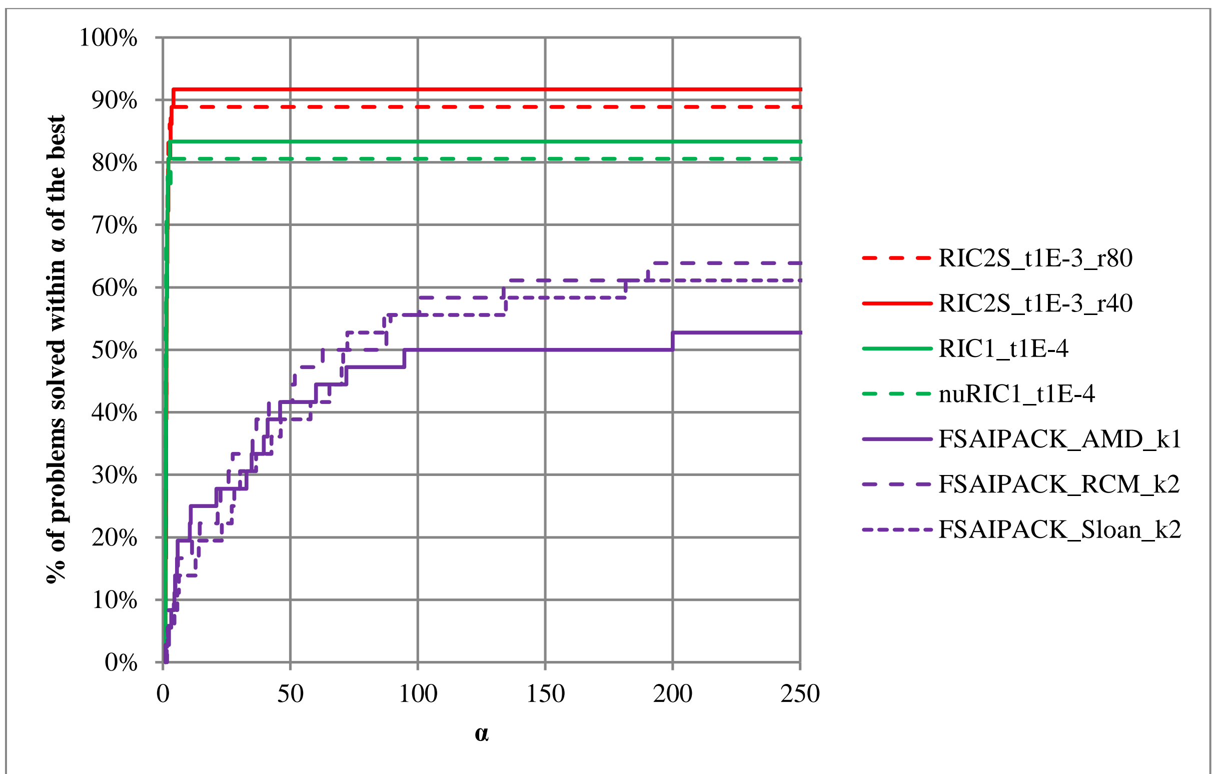


Figure 57. Performance profile of preconditioner build plus solve with the best incomplete Cholesky and FSAI preconditioners and parameter settings.

4.2 Solving the augmented equations

The two basic preconditioners for saddle point systems are the block analytic inverse (or block $\mathbf{LU}$) factorisation and the block diagonal or block triangular preconditioner with the Schur complement. Note that the block inverse can be considered to be a generalisation of the oft-cited constraint preconditioner (see Reference [222] for a discussion). Because of the better conditioning of the augmented equations (note the substantially smaller upper bound on the error in Table 16 than the corresponding systems in Table 15), it is possible that the Krylov solvers may be able to improve on the performance on the Schur complement system. In the following, the block analytical inverse

$$\begin{bmatrix} \mathbf{H}^{-1} - \mathbf{H}^{-1}\mathbf{A}^T\hat{\mathbf{S}}^{-1}\mathbf{A}\mathbf{H}^{-1} & \mathbf{H}^{-1}\mathbf{A}^T\hat{\mathbf{S}}^{-1} \\ \hat{\mathbf{S}}^{-1}\mathbf{A}\mathbf{H}^{-1} & -\hat{\mathbf{S}}^{-1} \end{bmatrix},$$

the block-diagonal Schur preconditioner

$$\begin{bmatrix} \mathbf{H} & \mathbf{0} \\ \mathbf{0} & \alpha\hat{\mathbf{S}} \end{bmatrix},$$

and the block-triangular Schur preconditioner

$$\begin{bmatrix} \mathbf{H} & \\ k\mathbf{A} & \alpha\mathbf{S} \end{bmatrix},$$

are compared where $\mathbf{H} = (\boldsymbol{\theta}\mathbf{W})^2$, $\hat{\mathbf{S}} \approx \mathbf{L}\mathbf{L}^T$ is an incomplete factorisation of $\mathbf{A}(\boldsymbol{\theta}\mathbf{W})^{-2}\mathbf{A}^T$, and α and k are scalars. Note that α is commonly set at 1 or -4 and both values are tested with the block-diagonal preconditioner [222]. The scalar k is set to 1.

The iteration counts using these preconditioners are shown in *Table 26* and the solve time multiplied by three plus the factorisation time are shown in Table 27. To enable a direct comparison, the associated runtimes from the better of both the RIC1 and RIC2S solvers tested are included. As expected, the number of iterations required to converge to a residual 2-norm of 10^{-8} or less generally grew from the 1st system through to the last system. The exception here was the *2DfootingLB* problem in which the first system

took longer to converge than the second system tested. For the problems where it converged, the triangular preconditioner converged in fewer iterations than the other approaches. It should be noted, however, that each iteration of BiCGSTAB requires twice the amount of work as PCG, but can be expected to converge approximately twice as fast. With all these preconditioners, there was no improvement over the Schur complement system approach when attempting to solve the last system for each problem, with the block diagonal Schur ($\alpha = -4$) and the block triangular Schur preconditioners each solving three of the 12 last systems, and the other two preconditioners not solving any. Interestingly, the two earlier systems from the *3DtunheadLB* were solved, but took a large number of iterations to converge. The runtime performance differs from the iteration count due to the operation count difference between SymQMR and BiCGSTAB. The fastest of the four preconditioners tested was the analytic block inverse, with the rest presenting mixed results. The two block-diagonal Schur preconditioners performed similarly, with α set to -4 solving more of the problems than $\alpha = 1$. The block triangular Schur preconditioner often outperformed the block diagonal preconditioner in the problems it solved, but failed on more of the problems than the other preconditioners. While different systems were being solved and, due to the way scaling was used to precondition the Schur complement system, the convergence criterion represented different levels of accuracy, and the results indicate a strong disadvantage in terms of runtime by solving the augmented equations instead of the Schur complement system. Furthermore, very little progress was made towards a solution for the later systems that were not solved within the 20,000 iteration limit, with some even diverging with the block triangular preconditioner. The performance profile in Figure 58 shows that not only were the Schur complement based methods able solve the problems considerably more quickly, they also solved considerably more of the systems in the test set.

Table 26. Saddle point preconditioner iteration counts. † indicates that convergence was not achieved within the maximum 20,000 iterations.

Problem	Iteration	Block inverse SymQMR	Block diagonal Schur (1) SymQMR	Block diagonal Schur (-4) SymQMR	Block triangular Schur (1) BiCGSTAB
2DfootingLB	1	812	1,523	1,548	418
	5	780	1,506	1,397	756
	26	†	†	†	†
2DfootingUB	1	51	96	101	30
	7	70	128	128	42
	23	†	†	†	†
2DtunnelLB	1	37	72	74	21
	7	77	146	144	59
	35	†	†	†	†
2DtunnelUB	1	25	48	50	16
	8	49	90	90	35
	22	†	†	17,420	†
3DsqrexcLB	1	91	173	177	49
	9	758	1,340	1,330	599
	16	†	†	†	†
3DsqrexcUB	1	141	264	266	119
	10	764	1,489	1,379	1,517
	19	†	†	†	†
3DsqrexcUB2	1	75	140	143	52
	8	453	832	830	350
	38	†	†	†	17,743
3DsqrfootLB	1	27	50	56	16
	11	69	128	125	39
	19	†	†	†	†
3DsqrfootUB	1	28	52	58	17
	12	119	215	215	73
	20	†	†	8,947	14,898
3DsqrfootUB2	1	18	34	39	11
	11	65	114	116	38
	21	†	†	3,042	2,696
3DtunheadLB	1	3,916	7,143	7,254	3,050
	12	8,979	14,666	14,476	†
	21	†	†	†	†
3DtunheadUB	1	43	80	84	27
	13	324	592	590	308
	25	†	†	†	†

Table 27. Saddle point preconditioner factor plus $3\times$ solve time. RIC1 was used with $\tau = 10^{-4}$ and RIC2S with $\tau = 10^{-3}$ and $p = 40$. † indicates that convergence was not achieved within the maximum 20,000 iterations. The fastest augmented equation times are in bold.

Problem	Iteration	Block inverse	Block diagonal Schur (1)	Block diagonal Schur (-4)	Block triangular Schur (1)	RIC1	RIC2S
2DfootingLB	1	173.4	267.3	276.4	**144.2**	70.3	85.7
	5	**168.6**	272.5	257.7	262.5	63.6	67.3
	26	†	†	†	†	†	†
2DfootingUB	1	12.1	17.4	18.3	**11.2**	8.5	5.9
	7	16.2	23.2	23.2	**14.9**	11.0	7.0
	23	†	†	†	†	497.2	631.3
2DtunnelLB	1	4.1	6.1	6.3	**3.7**	1.6	2.1
	7	**8.0**	12.2	12.3	9.5	1.9	3.5
	35	†	†	†	†	†	†
2DtunnelUB	1	**2.9**	4.1	4.3	†	1.0	1.2
	8	**5.3**	7.5	7.6	†	1.2	1.6
	22	†	†	**1298.6**	†	73.6	126.1
3DsqrexcLB	1	12.6	18.1	18.7	**11.6**	8.7	8.6
	9	**86.9**	128.7	130.1	115.5	37.6	44.9
	16	†	†	†	†	596.9	1212.4
3DsqrexcUB	1	**19.2**	27.6	28.2	24.1	9.6	7.9
	10	**102.9**	165.1	153.6	315.9	35.8	39.9
	19	†	†	†	†	730.3	1133.3
3DsqrexcUB2	1	**9.0**	11.4	11.9	†	3.3	4.1
	8	**49.0**	65.6	66.5	60.7	11.6	17.2
	38	†	†	†	†	206.2	585.3
3DsqrfootLB	1	5.3	6.9	7.3	**5.1**	6.0	4.2
	11	10.7	14.9	14.7	**10.2**	13.0	15.6
	19	†	†	†	†	†	389.4
3DsqrfootUB	1	4.1	5.4	5.8	**3.8**	3.3	2.5
	12	13.8	19.2	19.4	**13.3**	5.9	5.6
	20	†	†	**712.8**	†	68.5	62.5
3DsqrfootUB2	1	2.5	2.9	3.1	**2.3**	1.9	1.3
	11	7.2	8.5	8.7	**6.5**	3.0	2.6
	21	†	†	**187.3**	360.5	12.1	23.7
3DtunheadLB	1	**636.5**	992.2	1026.2	849.2	†	317.5
	12	**1529.7**	2180.8	2196.7	†	†	492.3
	21	†	†	†	†	†	†
3DtunheadUB	1	9.4	12.5	13.4	**8.9**	5.1	5.0
	13	54.5	77.7	78.6	78.1	11.7	18.8
	25	†	†	†	†	242.6	879.9

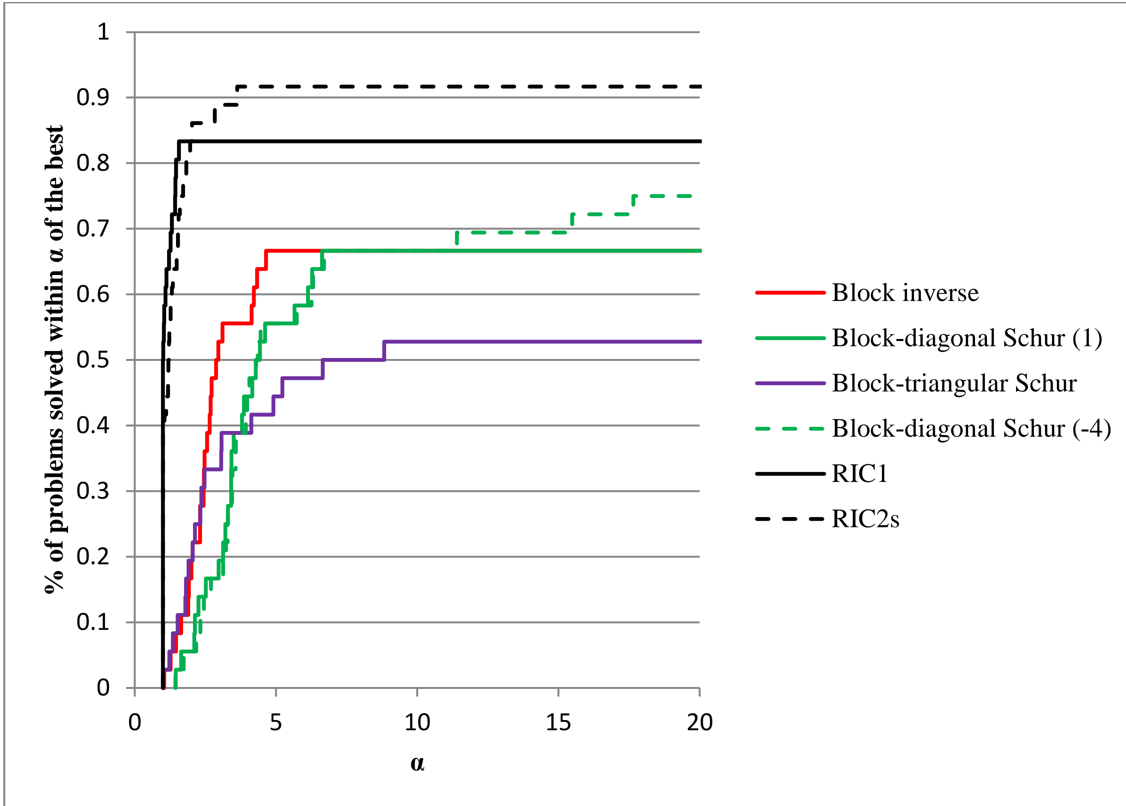


Figure 58. Performance profile of runtime plus $3\times$ solve time with conventional saddle point preconditioners. The RIC1 solver used $\tau = 10^{-4}$, and RIC2S used $\tau = 10^{-3}$ and $r = 40$.

4.3 Addressing the ill-conditioning in the search direction

In an attempt to overcome the increasingly unfavourable behaviour exhibited by the preconditioned iterative schemes for the majority of the FELA problems in the test set as the IPM converges towards a solution, a range of approaches were considered. These generally sought some method of dealing with the ill-conditioning in the search direction, but were not found to perform well in preliminary testing and so were not considered further. The approaches included the augmented preconditioner [199] that tries to solve KKT equations of the form

$$\begin{bmatrix} \mathbf{F} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{x} \\ \mathbf{y} \end{Bmatrix} = \begin{Bmatrix} \mathbf{p} \\ \mathbf{q} \end{Bmatrix}$$

with an iterative solver using the block-triangular preconditioner

$$\begin{bmatrix} \mathbf{F} + \mathbf{A}^T \mathbf{W}^{-1} \mathbf{A} & k\mathbf{A}^T \\ \mathbf{0} & \mathbf{W} \end{bmatrix}$$

or the block-diagonal preconditioner

$$\begin{bmatrix} \mathbf{F} + \mathbf{A}^T \mathbf{W}^{-1} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{W} \end{bmatrix}.$$

The majority of the computational effort required to use this preconditioner lies in solving a system with the $(1,1)$ block $\mathbf{F} + \mathbf{A}^T \mathbf{W}^{-1} \mathbf{A}$, which should be better conditioned than $\mathbf{F}$ for an appropriate choice of $\mathbf{W}$, but becomes significantly larger with many more non-zeros than the Schur complement system. The augmented Lagrangian Uzawa method was found to perform poorly for the same reason, with an almost identical system requiring solution. Similarly, the reduced augmented equations [76] described in Section 2.4.3.6became much larger and more dense than the Schur complement system. The final approach considered was along the lines of that used by Al-Jeiroudi [128] for solving linear programs. This scheme seeks a nonsingular basis in the constraint matrix that is associated with the small eigenvalues of the $(1,1)$ block in the augmented equations (note that this is diagonal for linear programming). While very low-fill $\mathbf{LU}$ factors could be found for the upper bound constraint matrices, the block-diagonal nature of the $(1,1)$ block in the second-order cone programs meant that in order to identify which columns should be considered for inclusion in the basis through diagonalisation of the $(1,1)$ block, the accompanying modification of the constraint matrix made it significantly more dense than the original (very few of the cones have eigenvalues which are all small, with most having at least one eigenvalue $O(1)$).

4.4 Using PCG to compute the search direction in an IPM

Having considered a range of Krylov subspace solvers for representative systems from the IPM and selected the best performing methods, these iterative solvers were then used inside the `Mixup8` IPM solver by replacing the supernodal Cholesky factorisation. Note that no presolving was done as it was found to degrade the performance when used in conjunction with the Krylov subspace solvers.

Unfortunately, the Krylov subspace methods may exhibit erratic convergence which makes it difficult to establish reliable tests that discern between idling or divergent behaviour [120]. The approach used here was to set a maximum number of iterations for

the Krylov solver when computing the first search direction, and then limit each subsequent search direction computation to use a maximum of 150% of the iterations used at the previous IPM step. A limit of 200% was used for nuRIC1 to account for the sudden deterioration of the preconditioner quality as ν was increased when non-positive pivots were encountered. This appeared to work well for the problems tested, limiting the number of iterations to a reasonable number as the IPM progressed without having too much of an adverse impact on the accuracy of the search direction.

In contrast to the maximum iteration count, the convergence tolerance is likely to have a more clearly defined impact on the ability of the IPM to obtain a solution and reduce the primal and dual infeasibilities. Two obvious strategies are to set an absolute convergence tolerance, which remains fixed, or adopt an adaptive convergence tolerance that becomes tighter as the IPM advances towards the solution. The absolute tolerance will provide better quality search directions early in the IPM iterations, but at a higher cost, while the adaptive approach permits lower accuracy to reduce the number of iterations performed by the iterative solver. On 64-bit personal computers, IEEE machine precision is approximately 10^{-16}, and it is not uncommon for direct factorisations to be able to achieve a residual norm $\left\|\mathbf{r}^{(k)}\right\| = \left\|\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}\right\|$ of around 10^{-14} for well-conditioned SPD systems. If one were seeking to simply substitute an iterative solver for a direct method, this provides a good starting point for setting the convergence tolerance. Alternatively, the convergence tolerance for the optimisation problems (primal and dual infeasibilities, and the normalised complementarity gap) are usually set to 10^{-8}, suggesting a minimum convergence accuracy because perturbations in the search direction of the order 10^{-8} or bigger are likely to adversely impact progress towards a feasible solution. It thus seems reasonable to test convergence tolerances between 10^{-8} and 10^{-14}. Considering the need to approach a feasible solution as well as an optimal solution, an adaptive tolerance should seek to avoid having an adverse impact on the reduction of the primal and dual infeasibilities, while still allowing for a reduction in the accuracy necessary at each iteration. By choosing a minimum required accuracy, and then reducing that to be some multiple of the minimum of the primal and dual infeasibilities, a certain minimum solution accuracy can be obtained in the early phase of the IPM, with an adaptive choice being used later.

Unfortunately, preliminary testing indicated that lower accuracy search directions in early IPM iterations had an adverse impact on the IPM convergence towards the optimisation problem solution. The convergence tolerance is thus not modified throughout the IPM. The problem with both the adaptive and absolute convergence tolerances is the failure to take into account the effect of system conditioning on solution accuracy. As recommended by Barret et al. [120], a convergence tolerance of $\left\|\mathbf{r}^{(k)}\right\|_\infty \le \varepsilon \max\left(1, \left\|\mathbf{A}\right\| \left\|\mathbf{x}^{(k)}\right\|_\infty + \left\|\mathbf{b}\right\|_\infty\right)$, where $\left\|\mathbf{A}\right\|$ was approximated by $\max\left(\left|a_{ij}\right|\right)$, was used with $\varepsilon = 10^{-13}$. Larger values of ε were found to be insufficiently accurate in the later iterations of the IPM, while smaller values were too difficult to satisfy for some systems.

For the iterative solver-based IPMs the primal infeasibility, dual infeasibility, and relative gap convergence tolerances were set to 10^{-7} to avoid the severely ill-conditioned matrices in the last two to three optimisation iterations. The step length relaxation factor was set at 0.95 and the free variables were split into positive and negative linear variables.

4.4.1.1 Small problem set

The preconditioners nuRIC1 ($\tau = 10^{-4}$), RIC1 ($\tau = 10^{-4}$), and RIC2S ($\tau = 10^{-3}$ and $r = 40$) were used to compute the search direction at each iteration of the IPM for the small problems in the test set. The results are shown in Table 28. The best performing approach from Chapter 3 is included here for comparison, and runtimes and factor non-zeros as a multiple of the direct solver-based IPM are shown in Table 29. Figure 59 shows the performance profile with the three preconditioners.

Overall, the iterative solver results for the small test set are discouraging. It is immediately clear that the iterative solver-based IPMs require much longer runtimes than the direct solver-based IPMs, ranging from 9 times to 181 times longer. The storage requirements for the incomplete factors for the three-dimensional problems were all less than that of the direct solver except for *3DsqrexcUB2S*, where the incomplete factor required 45% more storage than the full factor. This is due to the significant savings enabled by eliminating all but one of the free variables during the presolve when using the direct solver along with the better sparsity-preserving ordering.

The two dimensional problems are worse in relative terms, with RIC2S requiring 20% to 140% more storage, while the robust incomplete Cholesky preconditioners only saving 10 to 20%, except on 2DtunnelUB, where the storage requirements are practically the same as the direct solver. Coupled with the computational time of $15\times$ to over $200\times$ longer to solve, the best iterative approaches considered here are not suitable for two-dimensional finite element limit analysis.

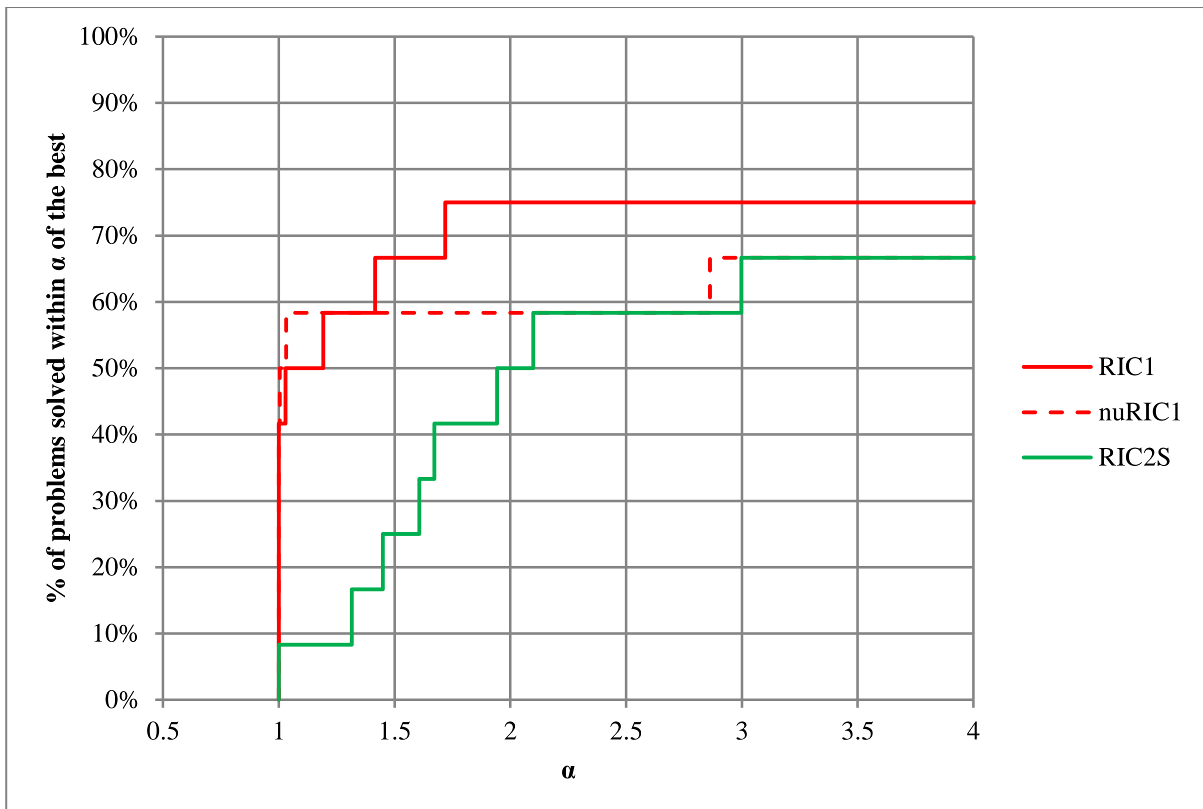


Figure 59. Performance profile of runtime on small problem set by incomplete Cholesky preconditioner.

4.4.1.2 The growth of computational requirements on the square footing problems

While the iterative solver-based IPMs do not perform well compared to those based on a direct solver for the small problem set, it was hoped that the iterative solvers could show favourable growth characteristics as the problems grow in size. To perform these tests, the small, medium, and large square footing problems were solved with the better performing iterative solvers being used to compute the search directions.

*Table 28. IPM results on small problem set with Krylov subspace solvers. * estimated.*

Problem	Solver	nit_{IPM}	nit_{PCG}	t_T	t_F	t_S	pobj	pinf	dinf	relgap	nnz(L)
2DfootingLB	mixup8	28		30.0	14.2	7.6	-14.833	2E-8	8E-9	8E-9	2.58E+7
	nuRIC1		49,450								2.31E+7
	RIC1	12	30,837	1855.7	50.4	1802.7	-14.784	3E-6	4E-7	3E-7	2.31E+7
	RIC2S	11	44,078	1784.1	29.4	1752.1	-14.741	5E-5	7E-7	6E-7	1.22E+7
2DfootingUB	mixup8	23		21.0	7.5	5.8	-14.916	8E-9	7E-9	6E-9	1.83E+7
	nuRIC1	17	13,795	608.7	40.8	563.2	-14.909	9E-8	8E-8	8E-8	1.46E+7
	RIC1	17	14,881	626.6	34.0	587.9	-14.909	9E-8	8E-8	8E-8	1.46E+7
	RIC2S	17	26,080	800.6	27.4	768.5	-14.909	9E-8	8E-8	8E-8	9.09E+6
2DtunnelLB	mixup8	42		14.2	6.1	4.5	-0.791	9E-8	6E-9	6E-9	7.86E+6
	nuRIC1	29	169,969	3360.3	26.9	3330.7*	-0.790	6E-8	9E-8	9E-8	6.36E+6
	RIC1	29	165,948	3254.7	24.6	3227.5	-0.790	6E-8	9E-8	9E-8	6.36E+6
	RIC2S		219,010								4.27E+6
2DtunnelUB	mixup8	19		5.6	1.7	1.5	-0.823	3E-9	6E-9	6E-9	4.59E+6
	nuRIC1	22	15,597	247.3	13.2	231.6	-0.824	8E-8	1E-8	1E-8	4.72E+6
	RIC1	19	5,026	86.4	9.9	74.4	-0.823	5E-8	8E-8	7E-8	4.73E+6
	RIC2S	19	9,331	125.2	10.0	113.1	-0.823	6E-8	8E-8	7E-8	3.31E+6
3DsqrexcLB	mixup8	17		19.9	15.8	2.2	-121.988	1E-9	7E-9	7E-9	2.41E+7
	nuRIC1	16	24,293	1388.7	252.6	1134.7	-121.987	4E-8	7E-8	7E-8	2.25E+7
	RIC1	13	3,367	312.6	160.6	150.8	-121.922	1E-5	3E-6	3E-6	2.24E+7
	RIC2S	17	87,527	2698.8	72.5	2624.8	-121.987	2E-7	7E-8	7E-8	7.30E+6
3DsqrexcUB	mixup8	19		18.4	13.5	2.2	-155.149	5E-10	4E-9	4E-9	1.85E+7
	nuRIC1	19	40,152	1659.4	162.8	1494.3	-155.148	7E-9	7E-8	7E-8	1.60E+7
	RIC1	19	61,735	2351.0	121.2	2227.5	-155.148	4E-8	7E-8	7E-8	1.58E+7
	RIC2S	19	137,020	2665.6	60.0	2603.4	-155.148	3E-8	7E-8	7E-8	5.53E+6
3DsqrexcUB2	mixup8	17		9.3	4.4	1.3	-138.246	1E-9	9E-9	9E-9	7.93E+6
	nuRIC1	37	43,349	1401.9	290.8	1103.5	-138.196	9E-8	1E-7	9E-8	1.15E+7
	RIC1	38	57,903	1671.3	240.9	1422.9	-138.196	2E-7	1E-7	9E-8	1.15E+7
	RIC2S		245,586								4.49E+6
3DsqrfootLB	mixup8	20		29.7	24.8	2.8	-5.492	1E-9	1E-8	1E-8	2.92E+7
	nuRIC1		58,306								1.50E+7
	RIC1		72,733								1.55E+7
	RIC2S	18	9,346	265.6	54.4	209.8	-5.492	1E-6	4E-7	4E-7	6.86E+6
3DsqrfootUB	mixup8	19		21.1	16.7	2.2	-6.234	3E-10	3E-9	3E-9	2.08E+7
	nuRIC1	20	4,255	177.6	66.5	109.0	-6.234	7E-9	5E-8	5E-8	1.03E+7
	RIC1	20	9,890	305.3	58.0	245.1	-6.234	9E-9	5E-8	5E-8	1.00E+7
	RIC2S	20	23,645	372.7	39.1	331.7	-6.234	5E-8	5E-8	5E-8	4.31E+6
3DsqrfootUB2	mixup8	19		11.2	5.9	1.3	-6.170	6E-10	6E-9	6E-9	8.93E+6
	nuRIC1	21	2,283	103.8	59.8	39.6	-6.169	2E-8	9E-8	8E-8	6.47E+6
	RIC1	21	3,064	103.4	48.1	50.7	-6.169	9E-9	9E-8	8E-8	6.47E+6
	RIC2S	21	12,995	172.9	27.2	141.2	-6.169	1E-8	9E-8	8E-8	3.27E+6
3DtunheadLB	mixup8	22		48.3	39.8	5.0	-22.394	1E-9	6E-9	6E-9	4.35E+7
	nuRIC1		53,121								2.57E+7
	RIC1		30,180								2.05E+7
	RIC2S		90,326								1.13E+7
3DtunheadUB	mixup8	20		32.9	25.7	3.9	-33.431	1E-9	7E-9	7E-9	3.13E+7
	nuRIC1	19	816	143.5	105.8	34.7	-33.375	2E-5	4E-6	4E-6	1.71E+7
	RIC1	25	24,380	1154.1	125.4	1024.6	-33.431	8E-8	7E-8	7E-8	1.85E+7
	RIC2S	25	154,719	3459.2	76.8	3378.3	-33.431	9E-8	7E-8	7E-8	6.33E+6

Table 29. Runtime and incomplete factorisation size as a multiple of `Mixup8` results.

Problem	Solver	t_T / t_{mixup8}	nnz(LIC) / nnz(L)
2DfootingLB	nuRIC1	-	90%
	RIC1	62	90%
	RIC2S	-	120%
2DfootingUB	nuRIC1	29	79%
	RIC1	30	79%
	RIC2S	38	126%
2DtunnelLB	nuRIC1	237	81%
	RIC1	230	81%
	RIC2S	-	162%
2DtunnelUB	nuRIC1	44	103%
	RIC1	15	103%
	RIC2S	22	239%
3DsqrexcLB	nuRIC1	70	93%
	RIC1	-	93%
	RIC2S	135	57%
3DsqrexcUB	nuRIC1	90	86%
	RIC1	128	86%
	RIC2S	145	71%
3DsqrexcUB2	nuRIC1	152	145%
	RIC1	181	145%
	RIC2S	-	99%
3DsqrfootLB	nuRIC1	-	51%
	RIC1	-	53%
	RIC2S	9	45%
3DsqrfootUB	nuRIC1	8	50%
	RIC1	14	48%
	RIC2S	18	44%
3DsqrfootUB2	nuRIC1	9	72%
	RIC1	9	72%
	RIC2S	15	62%
3DtunheadLB	nuRIC1	-	59%
	RIC1	-	47%
	RIC2S	-	49%
3DtunheadUB	nuRIC1	-	55%
	RIC1	35	59%
	RIC2S	105	44%

The results show that the lower bounds are especially difficult to compute when using the Krylov solvers. It is promising that the total PCG iterations experience only moderate growth as the problem size grows. Similarly, the number of IPM iterations does not increase with problem size for the Krylov solver-based analyses, However, as with the small problem set, the IPM took more iterations to converge using an iterative solver than with a direct solver, even when a looser tolerance was used. All the primal objective function values were the same or very close for the upper bound problems, but failure through the time cut-off and lack of convergence occurred on many of the lower bound problems.

Figure 60 shows the growth in the number of non-zeros in the incomplete factors for *3DsqrfootUB2*, Figure 61 shows the growth in total solution time for *3DsqrfootUB2*, and Figure 62 and Figure 63 show the same for *3DsqrfootUB*. As can be seen in Figure 60 and Figure 62, the preconditioned iterative solvers show very close to linear growth in the factor size across these problems, making them ideal for use when the problems are too large to factorise with a direct method.

Unfortunately, the iterative solvers exhibit nonlinear growth in the runtime as the problem size increases. This is shown in Figure 61 and Figure 63. It should be kept in mind that this behaviour is being observed with convergence tolerances that were relaxed for the iterative solver and on the two problems for which the iterative solver-based approaches attained the best relative performance. Thus, it is likely that the only situation in which an iterative solver should be used to compute the IPM search direction for large-scale finite element analysis problems is when memory limitations prevent the full Cholesky factor from being computed. Even then, extreme care must be taken with the parameter settings to ensure that the preconditioner is not too large while still being accurate enough to accelerate convergence.

Table 30. Results for square footing problems.

Problem	Solver	nit$_{IPM}$	nit$_{PCG}$	t$_T$	t$_F$	t$_S$	pobj	pinf	dinf	relgap	m	n	nnz(A)	nnz(L)
	mixup8	20		29.7	24.8	2.8	-5.492	1E-09	1E-08	1E-08	126,972	181,008	1,461,776	29,160,870
3DsqrfootLBS	nuRIC1		58306								153,648	208,008	1,785,434	15,001,179
	RIC1		72733								153,648	208,008	1,785,434	15,520,104
	mixup8	24		174.9	159.6	9.7	-5.557	1E-09	4E-09	4E-09	301,120	429,312	3,482,024	95,396,160
3DsqrfootLBM	nuRIC1		101265								363,904	492,672	4,246,600	40,799,574
	RIC1	14	37290	4036.8	238.2	3795.6	-5.549	7E-05	1E-05	1E-05	363,904	492,672	4,246,600	42,538,197
	mixup8	24		1588.3	1526.1	42.1	-5.629	3E-09	6E-09	6E-09	1,016,784	1,449,792	11,807,552	485,720,287
3DsqrfootLBL	nuRIC1		8112								1,227,168	1,661,472	14,379,178	125,108,516
	RIC1		14487								1,227,168	1,661,472	14,379,178	154,199,318
	mixup8	19		21.1	16.7	2.2	-6.234	3E-10	3E-09	3E-09	65,906	299,498	722,552	20,767,502
3DsqrfootUBS	nuRIC1	20	4255	177.6	66.5	109.0	-6.234	7E-09	5E-08	5E-08	121,932	360,720	1,190,052	10,280,051
	RIC1	20	9890	305.3	58.0	245.1	-6.234	9E-09	5E-08	5E-08	121,932	360,720	1,190,052	10,012,930
	mixup8	19		92.7	81.1	5.9	-6.112	8E-10	8E-09	8E-09	159,042	716,402	1,758,648	67,045,990
3DsqrfootUBM	nuRIC1	21	2074	388.4	235.3	148.2	-6.112	9E-07	3E-07	3E-07	289,728	856,320	2,833,728	27,723,951
	RIC1	22	13034	1081.2	200.9	874.9	-6.112	3E-08	3E-08	2E-08	289,728	856,320	2,833,728	26,552,870
	mixup8	21		1113.6	1061.6	30.7	-5.991	7E-10	6E-09	6E-09	546,242	2,439,674	6,091,328	383,195,411
3DsqrfootUBL	nuRIC1	22	9132	3778.0	1268.5	2491.0	-5.991	1E-08	8E-08	8E-08	980,208	2,894,400	9,607,248	107,277,965
	RIC1	22	21574	6506.2	907.3	5580.3	-5.991	8E-09	8E-08	8E-08	980,208	2,894,400	9,607,248	101,618,223
	mixup8	19		11.2	5.9	1.3	-6.170	6E-10	6E-09	6E-09	27,480	181,440	1,025,482	8,929,792
3DsqrfootUB2S	nuRIC1	21	2283	103.8	59.8	39.6	-6.169	2E-08	9E-08	8E-08	56,549	213,708	1,995,855	6,468,086
	RIC1	21	3064	103.4	48.1	50.7	-6.169	9E-09	9E-08	8E-08	56,549	213,708	1,995,855	6,465,459
	RIC2S	21	12995	172.9	27.2	141.2	-6.169	1E-08	9E-08	8E-08	56,549	213,708	1,995,855	3,266,057
	mixup8	19		43.7	30.6	3.7	-6.048	8E-10	8E-09	8E-09	65,176	430,080	2,497,586	30,064,290
3DsqrfootUB2M	nuRIC1	23	3290	383.6	220.7	151.2	-6.048	1E-08	5E-08	4E-08	132,149	502,668	4,700,321	18,465,436
	RIC1	23	4244	396.7	200.3	184.9	-6.048	1E-08	5E-08	4E-08	132,149	502,668	4,700,321	18,158,845
	RIC2S	23	15059	504.8	83.3	410.0	-6.048	8E-09	5E-08	4E-08	132,149	502,668	4,700,321	8,125,643
	mixup8	19		378.7	330.4	15.4	-5.949	5E-10	5E-09	5E-09	220,092	1,451,520	8,658,898	168,066,502
3DsqrfootUB2L	nuRIC1	22	3146	2020.9	1413.0	569.8	-5.949	1E-08	5E-08	4E-08	439,757	1,683,660	15,761,109	77,203,689
	RIC1	22	4676	1842.1	1004.2	783.0	-5.949	8E-09	5E-08	4E-08	439,757	1,683,660	15,761,109	73,625,152
	RIC2S	22	30272	3263.2	312.8	2912.2	-5.949	2E-08	5E-08	4E-08	439,757	1,683,660	15,761,109	28,815,402

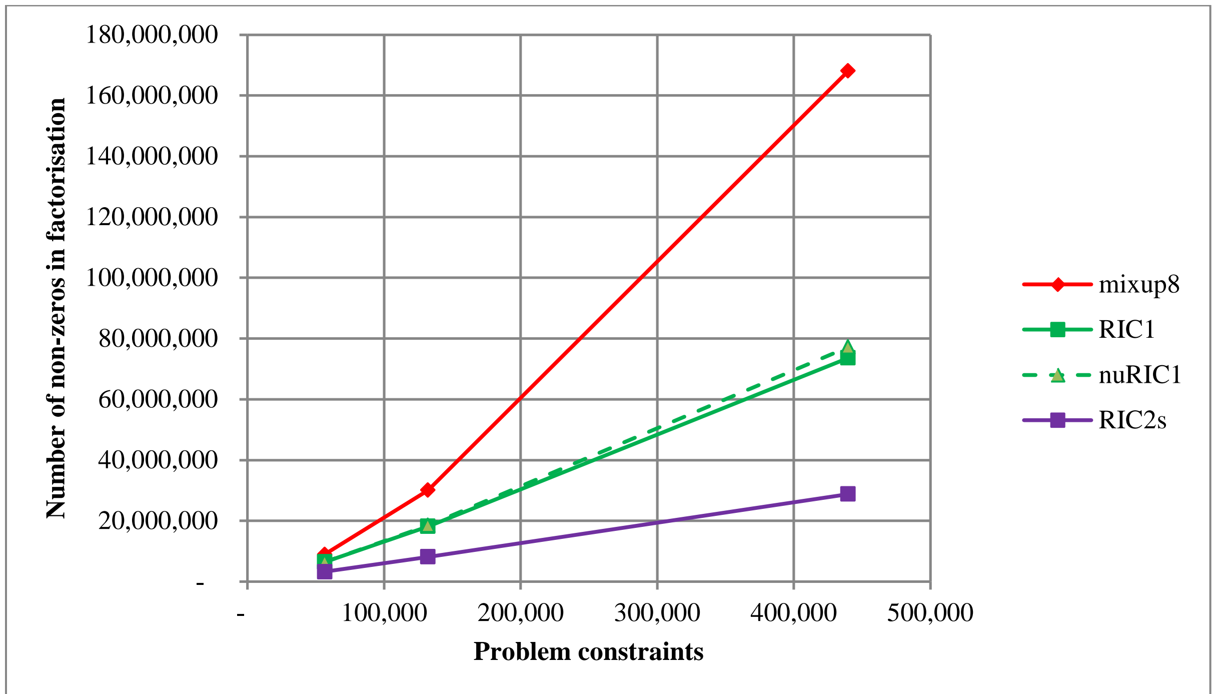


Figure 60. Number of non-zeros in factorisation versus number of constraints for 3DsqrfootUB2. Note that this includes the memory allocated for **R** *in RIC2S.*

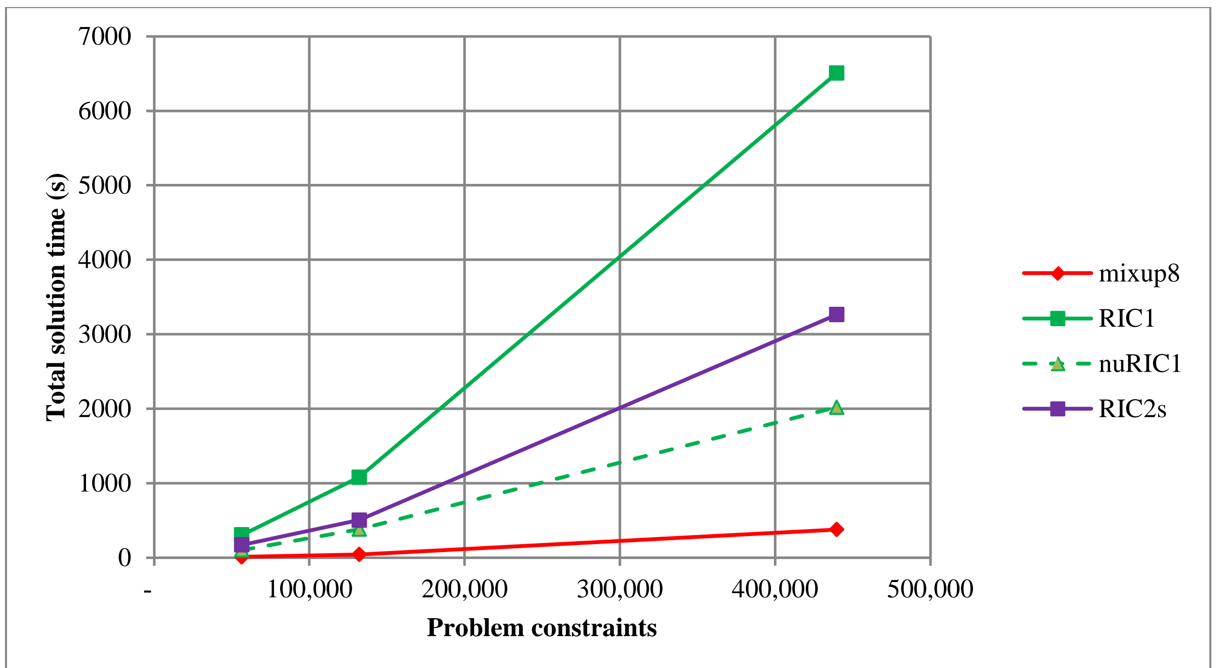


Figure 61. Total solution time versus number of constraints for 3DsqrfootUB2.

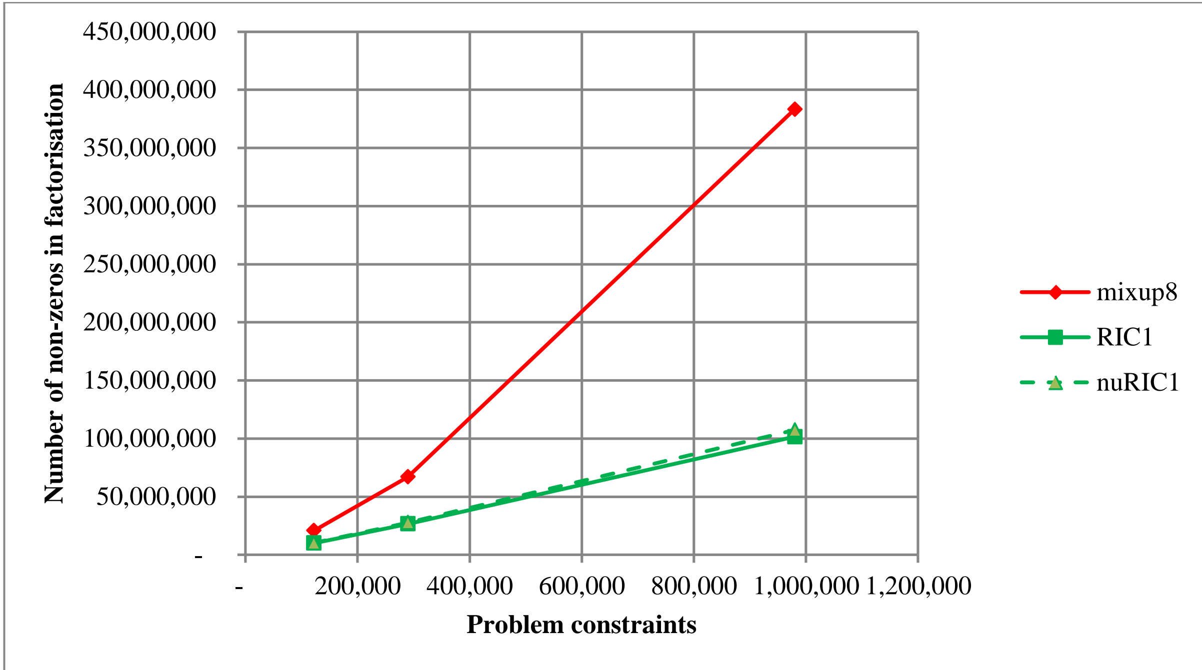


Figure 62. Number of non-zeros in factorisation versus number of constraints for 3DsqrfootUB.

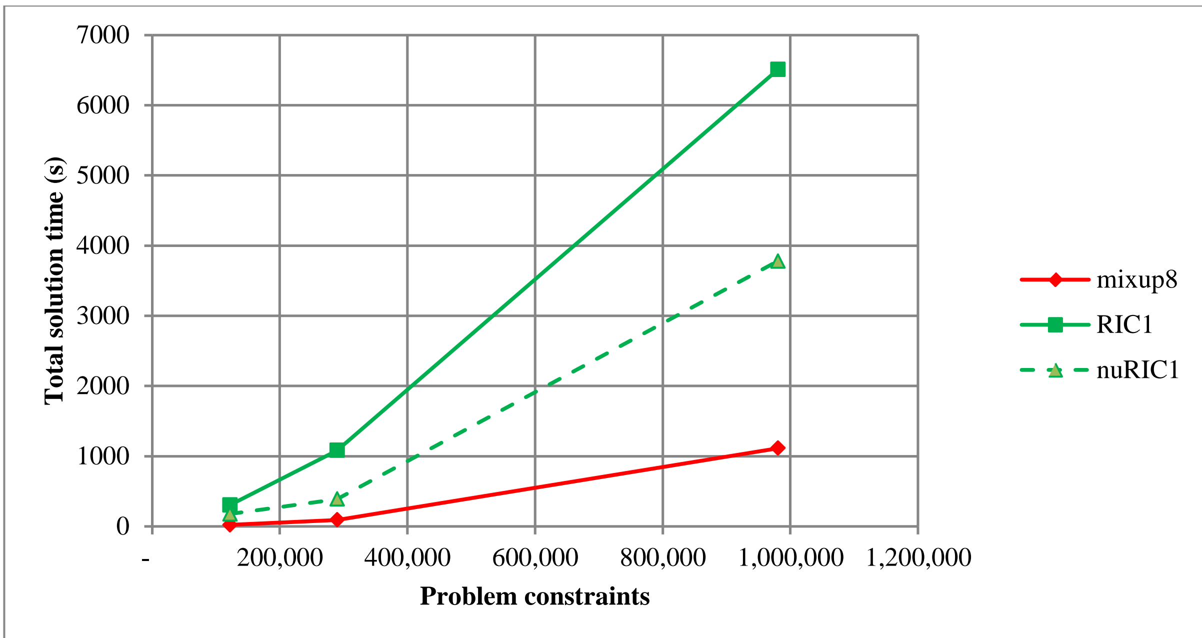


Figure 63. Total solution time versus number of constraints for 3DsqrfootUB.

Chapter 5 Parallelisation of the solution scheme

In order to solve very large FELA problems in a practical manner, it is necessary to minimise the associated wall-clock time required. As processor manufacturers move away from increasing the speed of single processing cores to introducing more cores, it is crucial that performance-critical portions of the IPM code are parallelised where possible. In the following, a brief overview of parallel computing is provided, before describing an IPM designed to take advantage of parallel processing capability and presenting the performance results obtained with it. Because of the relatively poor floating point operation speed of sparse matrix-vector multiplications, as well as the lack of their robustness and serial performance, the Krylov subspace solvers are not considered further in this Thesis. Instead, we aim to exploit the high performance obtained by dense matrix multiplication kernels for direct solvers.

5.1 Overview of parallel computing

Although a common interpretation of Moore's Law finds the increase in computing performance has slowed since the early 2000s in terms of processor clock rate, the widespread introduction of parallel computing architectures has continued to support Moore's Law, and is likely to continue to do so [223]. As a result of microprocessor chips being released with increasing number of cores, application performance will no longer experience the performance improvements gained historically through uniprocessor development. In addition to the multicore processors, add-on cards conventionally designed for rendering graphics (known as graphical processing units or GPUs) are now being exploited for general-purpose parallel computation. GPUs today have thousands of cores per device, and some are built specifically for computation. To increase the total computational power further, individual machines are grouped together in a cluster and messages are passed between the machines via a network. Such parallelism is often referred to as distributed memory parallelism because of the individual memory owned by each machine that is not directly accessible by any other machine. This is in contrast to shared memory parallelism, such as that on a multicore machine, where each core has direct access to the same memory. GPUs have their own memory and require data to be copied between the host machine and the device.

Obviously, in network clusters, there are both the shared and the distributed memory architectures to work with.

Interestingly, many of the clusters built recently that rank among the fastest supercomputers in the world utilise general purpose GPUs. As an example, the Titan, a heterogeneous computing system containing 18,688 networked CPUs (central processing units), includes as many NVIDIA Tesla GPUs (graphics processing units) as there are CPUs along with 710TB of RAM. Each CPU is a 16 core AMD Opteron, while each Tesla GPU has 2688 processing cores. Solving a dense system of linear equations, the system has performed at 17.59×10^{15} floating point operations per second. As of 2013, the list of the top 500 supercomputers in the world no longer features any systems with less than 2,000 processing cores.

In order to achieve high fractions of the peak performance from a given machine, all of the machine's capabilities must be exploited as much as possible. On modern machines, this includes vectorisation, data locality, and interprocess communication. There are also other important aspects that are largely invisible from software and so will not be addressed here (for example, pipelining and multithreading; see [224] for an in-depth treatment). Vector, or SIMD (single instruction multiple data), processing allows the same operation to be performed on multiple data items simultaneously. While modern CPU chips often include vector capabilities for 2 double precision floating point operations at a time, GPUs are SIMD machines that can process many more operations concurrently. The performance of both, however, is usually determined by memory access and data movement. Modern CPUs contain multiple levels of cache in a hierarchy, starting with a hard drive on the lower level with high latency but large amounts of storage, and progressing to caches with lower latencies but smaller storage capacity through to the on-chip registers. Much effort is put into minimising the amount and impact of data movement across the memory hierarchy, especially for operations with significant data re-use such as matrix-matrix multiplication. As sending, waiting for, and receiving messages adds to the overhead of a parallel program, interprocess communication should also be minimised. Such communication may be between, among others, threads of a multithreaded application on a modern CPU, co-operative work performed by a cluster of computers, or between a host computer and a GPU.

5.2 Parallelisation of the IPM

In determining where effort should be focussed within the IPM for the LPs and SOCPs solved here, guidance is provided by Amdahl's Law: the execution time after an improvement will be equal to the execution time affected by the improvement divided by the amount of improvement plus the execution time that is unaffected. This makes it clear that unless the improvement will affect a large fraction of the overall runtime, then limited improvement is possible. With the majority of the IPM solution time in the factorisation of the coefficient system defining the search direction, improving the linear solver has the most potential to make the biggest impact on reducing the overall runtime. Table 28 and Table 30 report the factorisation time for some of the test problems, showing that in most cases over 75% of the total time is spent in the factorisation routine. While other areas are able to be parallelised and are likely to yield some improvement [27], [59], [225], the benefit is not expected to be significant for FELA problems. It is thus expected that, through the use of parallel dense linear algebra subroutines, notably the level 3 BLAS matrix-matrix multiplication routine that allows significant reuse of data as well as exhibiting naturally independent operations that can be run in parallel, the runtime of the solvers can be significantly reduced. The main hurdle to be overcome for sparse linear equation solvers is ensuring that the dense subproblems are large enough to fully exploit the available performance of the hardware. Fortunately, many of the supernodes, especially as one moves towards the root of the elimination tree, are large enough to expect a major improvement if a machine's parallel computing resources can be exploited. This approach also enables pre-compiled and highly optimised BLAS libraries to be used on single machines. These parallel libraries have been tuned for the hardware and provide a high fraction of peak hardware performance. It should be noted that sparse linear equation solvers have been developed for distributed systems, including the solver in IBM's Watson Sparse Matrix Package (WSMP) [226], [227]. Most parallel linear equation solvers, however, do not involve clusters of networked machines and instead rely on the parallel processing capability of an individual machine.

For the `Mixup8` implementation described in Section 3.4, the parallel Intel MKL 11.0.5 was used for the dense BLAS operations in the supernodal Cholesky factorisation. To exploit the highly parallel capability of recent GPU hardware, a modified version of

CHOLMOD (version 3.0.3) was used. CHOLMOD was modified by discarding the functionality to restart the factorisation when a non-positive pivot is encountered and the same modified DPOTRF routine as Mixup8 uses is called in place of the BLAS library routine. To test the improvement in performance, the problem set was solved using MOSEK with multiple threads, Mixup8 using multiple threads, and Mixup8 utilising a GPU. Note that it is not known what specific areas in MOSEK have been parallelised, while the two Mixup8 programs are only exploiting parallelism in the factorisation of the Schur complement system and subsequent solves, as well as any calls to the BLAS library.

The parallel MOSEK code is labelled *mospar*, the parallel Mixup8 code *mixpar*, and the code exploiting the GPU *mixgpu*. Both Mixup8 codes use the presolve process, exploit fixed variables, and handle dense columns explicitly. Any free variables that are not eliminated are regularised by setting the diagonal in the $(1,1)$ block of the augmented equations to 10^{-10}, with the exception of any free variable associated with a dense column in which the augmented equation system is solved without regularisation. The results presented in this chapter summarise simulations performed on an Intel Xeon E5-1620 @ 3.60 GHz with 64GB RAM and an NVIDIA Tesla K20c GPU.

Figure 64 and Table 31 presents a comparison of the runtime between the parallel solvers and the complete results on the small problem set, respectively. Figure 65 and Table 32 present the same information but for the medium problems, and Figure 66 shows the iteration counts on the large problem set with Figure 67 and Table 32 containing the runtimes and complete results, respectively, for the large problems.

The two Mixup8 codes generally compute the same objective values, with slight differences due to rounding differences in the solver that stem from different accumulation processes when computing the contribution from the child supernodes. On the GPU version, this process is scheduled dynamically, which means that small differences may result from run to run. As expected, the iteration counts are almost identical between the two Mixup8 solvers.

MOSEK computes similar objective values for most problems, but, as with the sequential solver, it has trouble with the *2Dtunnel*B* problems, converging on different solutions

in all three sizes. While the reason is not known, it is likely due to the deletion of a constraint thought to be redundant. The commercial solver also has fewer non-zeros in the factorisation, but this difference may be a result of the non-zero count reported for the `Mixup8` codes including the unused space in each supernode. It should be noted that the GPU version of Mixup8 also converges on a different objective value for *2DtunnelLBL* and appears to be caused by the slight difference arising from rounding described above.

The difference in runtime on the small problem set shown in Figure 64 is not great nor consistent across all problems, with the GPU version being slowest on all four of the two-dimensional problems, and showing negligible improvement over the parallel `Mixup8` solver. Both Mixup8 solvers compare favourably against MOSEK on the three-dimensional problems, being faster on all the small test problems except *3DsqrfootUB2S*. This improvement is more pronounced and consistent across all of the three-dimensional problems in the medium problem set as shown in Figure 65. Moreover, the GPU version of `Mixup8` is clearly faster than the CPU-only version on all of the three-dimensional problems but again offers no improvement in the two-dimensional cases. This is similar to the large problem set, although, to determine that the two-dimensional performance results are mixed, one must look at Table 33 because the scale required for the large three-dimensional problems obscure the details in Figure 67.

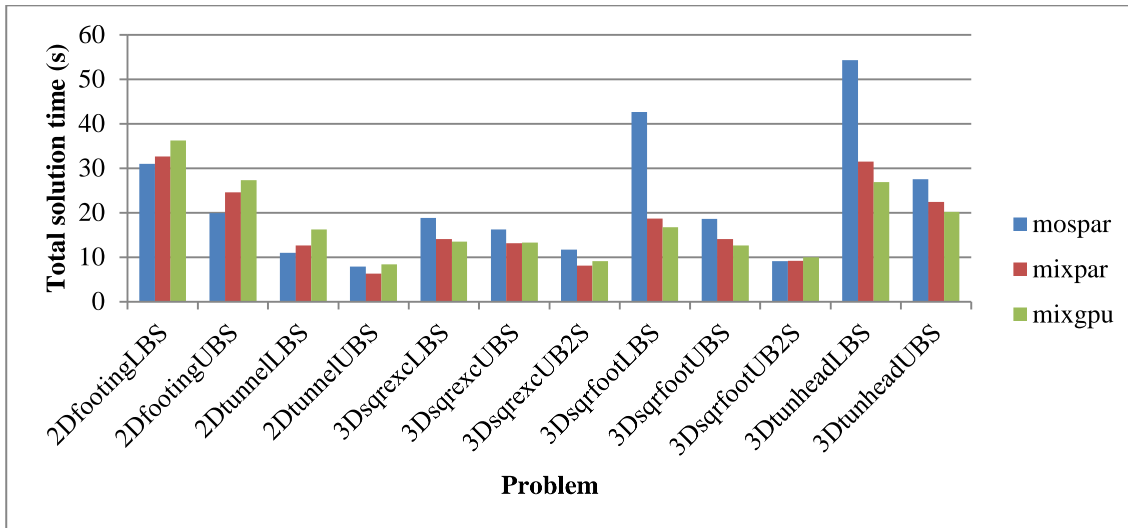


Figure 64. Comparison of the total solution time on the small problem set between the parallel solvers.

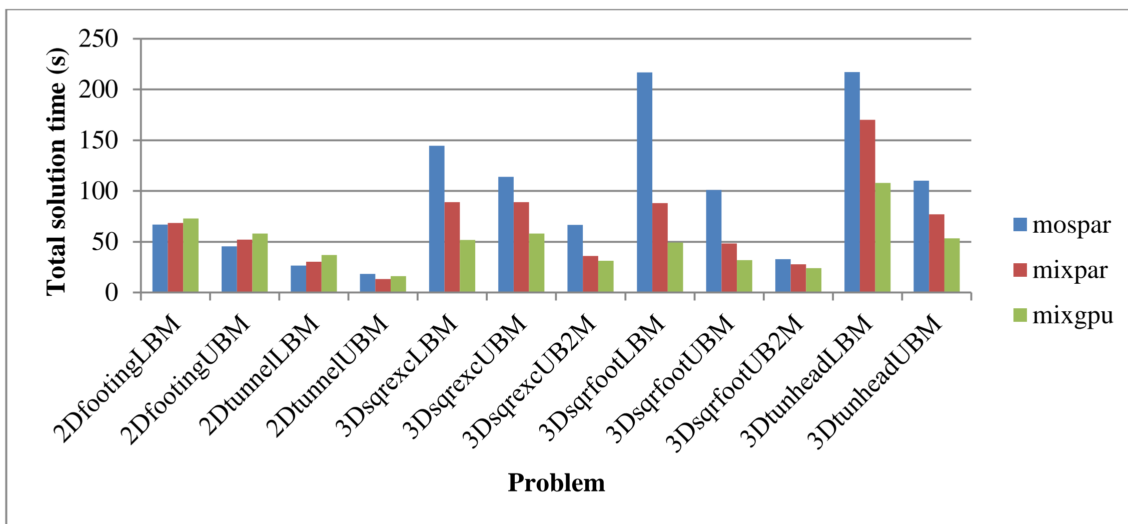


Figure 65. Comparison of the total solution time on the medium problem set between the parallel solvers.

Table 31. Parallel solver results on small problems.

Problem	Solver	nit	t_T	t_P	t_O	pobj	pfeas	dfeas	relgap	m	n	nnz(A)	nnz(L)
	mospar	25	31.0	0.3	8.1	-14.831	5E-9	6E-9	6E-9	464,940	523,391	2.8E+6	1.8E+7
2DfootingLB	mixpar	29	32.6	0.2	2.2	-14.833	5E-8	8E-9	7E-9	465,080	523,530	2.8E+6	2.6E+7
	mixgpu	27	36.2	0.3	3.6	-14.833	1E-8	8E-9	8E-9	465,080	523,530	2.8E+6	2.6E+7
	mospar	20	19.9	0.5	4.9	-14.914	5E-10	7E-9	3E-9	348,040	696,990	1.9E+6	1.4E+7
2DfootingUB	mixpar	23	24.6	0.2	1.4	-14.916	8E-9	7E-9	6E-9	348,040	696,990	1.9E+6	1.8E+7
	mixgpu	23	27.3	0.3	2.8	-14.916	8E-9	7E-9	6E-9	348,040	696,990	1.9E+6	1.8E+7
	mospar	19	11.0	0.8	3.1	-0.767	5E-8	3E-8	3E-8	106,462	182,924	1.6E+6	6.3E+6
2DtunnelLB	mixpar	32	12.6	0.2	0.6	-0.790	8E-8	3E-8	3E-8	152,694	229,375	9.2E+5	7.9E+6
	mixgpu	37	16.2	0.2	2.0	-0.791	1E-6	5E-9	5E-9	152,694	229,375	9.2E+5	7.9E+6
	mospar	16	7.9	0.9	1.2	-0.763	4E-8	6E-8	6E-8	51,916	243,158	7.7E+5	3.9E+6
2DtunnelUB	mixpar	19	6.3	0.2	0.4	-0.823	3E-9	6E-9	6E-9	64,275	255,516	6.2E+5	4.6E+6
	mixgpu	19	8.4	0.3	1.8	-0.823	3E-9	6E-9	6E-9	64,275	255,516	6.2E+5	4.6E+6
	mospar	17	18.8	0.2	2.6	-121.987	1E-8	3E-8	3E-8	121,348	147,458	1.4E+6	2.1E+7
3DsqrexcLB	mixpar	17	14.1	0.2	0.6	-121.988	9E-10	7E-9	7E-9	121,348	147,457	1.4E+6	2.4E+7
	mixgpu	17	13.5	0.2	2.0	-121.988	8E-10	7E-9	7E-9	121,348	147,457	1.4E+6	2.4E+7
	mospar	19	16.2	0.3	1.3	-155.147	3E-8	4E-8	4E-8	69,648	248,834	7.9E+5	1.5E+7
3DsqrexcUB	mixpar	19	13.1	1.0	0.2	-155.149	5E-10	4E-9	4E-9	69,648	248,833	7.4E+5	1.9E+7
	mixgpu	19	13.3	1.1	1.8	-155.149	5E-10	4E-9	4E-9	69,648	248,833	7.4E+5	1.9E+7
	mospar	25	11.7	0.6	1.3	-138.247	1E-7	2E-8	6E-9	31,227	152,221	1.2E+6	6.9E+6
3DsqrexcUB2	mixpar	17	8.1	0.7	0.2	-138.246	1E-9	9E-9	9E-9	26,464	147,457	9.8E+5	7.9E+6
	mixgpu	17	9.1	0.6	1.6	-138.246	1E-9	9E-9	9E-9	26,464	147,457	9.8E+5	7.9E+6
	mospar	21	42.6	0.8	4.7	-5.492	9E-9	2E-8	2E-8	109,786	163,895	2.1E+6	3.3E+7
3DsqrfootLB	mixpar	20	18.7	0.1	0.6	-5.492	2E-9	1E-8	1E-8	126,972	181,008	1.5E+6	2.9E+7
	mixgpu	20	16.7	0.1	2.1	-5.492	2E-9	1E-8	1E-8	126,972	181,008	1.5E+6	2.9E+7
	mospar	18	18.6	0.7	2.3	-6.234	9E-9	2E-8	2E-8	36,430	270,023	1.0E+6	1.5E+7
3DsqrfootUB	mixpar	19	14.1	0.2	0.4	-6.234	3E-10	3E-9	3E-9	65,906	299,498	7.2E+5	2.1E+7
	mixgpu	19	12.6	0.2	1.9	-6.234	3E-10	3E-9	3E-9	65,906	299,498	7.2E+5	2.1E+7
	mospar	19	9.1	0.5	1.1	-6.169	1E-8	3E-8	3E-8	27,882	181,843	1.1E+6	7.9E+6
3DsqrfootUB2	mixpar	19	9.2	0.5	0.2	-6.170	6E-10	6E-9	6E-9	27,480	181,440	1.0E+6	8.9E+6
	mixgpu	19	10.0	0.5	1.7	-6.170	6E-10	6E-9	6E-9	27,480	181,440	1.0E+6	8.9E+6
	mospar	24	54.3	0.5	5.0	-22.394	2E-5	7E-8	6E-7	203,063	239,581	2.4E+6	3.8E+7
3DtunheadLB	mixpar	22	31.5	0.2	0.9	-22.394	1E-9	6E-9	6E-9	203,868	240,193	2.4E+6	4.3E+7
	mixgpu	22	26.9	0.3	2.4	-22.394	1E-9	5E-9	5E-9	203,868	240,193	2.4E+6	4.3E+7
	mospar	20	27.5	0.4	2.0	-33.430	4E-8	4E-8	4E-8	112,752	406,730	1.2E+6	2.6E+7
3DtunheadUB	mixpar	20	22.4	0.3	0.4	-33.431	1E-9	7E-9	7E-9	112,752	406,729	1.2E+6	3.1E+7
	mixgpu	20	20.2	0.3	1.9	-33.431	1E-9	7E-9	7E-9	112,752	406,729	1.2E+6	3.1E+7

Table 32. Parallel solver results on medium problems.

Problem	Solver	nit	t_T	t_P	t_O	pobj	pfeas	dfeas	relgap	m	n	nnz(A)	nnz(L)
2DfootingLB	mospar	19	66.8	0.7	20.6	-14.824	9E-9	8E-9	8E-9	1,050,210	1,181,986	6.3E+6	4.6E+7
	mixpar	25	68.5	0.5	5.4	-14.832	2E-8	8E-9	8E-9	1,050,420	1,182,195	6.3E+6	6.3E+7
	mixgpu	23	72.8	0.6	6.8	-14.831	2E-8	1E-8	1E-8	1,050,420	1,182,195	6.3E+6	6.3E+7
2DfootingUB	mospar	20	45.4	1.1	12.4	-14.886	5E-10	5E-9	2E-9	786,660	1,574,685	4.3E+6	3.4E+7
	mixpar	22	52.1	0.5	3.3	-14.889	8E-9	7E-9	7E-9	786,660	1,574,685	4.3E+6	4.5E+7
	mixgpu	22	58.2	0.6	4.8	-14.889	8E-9	7E-9	7E-9	786,660	1,574,685	4.3E+6	4.5E+7
2DtunnelLB	mospar	16	26.7	2.0	7.3	-0.649	4E-8	4E-8	4E-8	267,653	439,945	3.2E+6	1.6E+7
	mixpar	36	30.4	0.3	1.6	-0.798	2E-7	9E-8	9E-8	344,242	516,863	2.1E+6	1.9E+7
	mixgpu	37	37.1	0.3	3.1	-0.798	2E-7	8E-8	8E-8	344,242	516,863	2.1E+6	1.9E+7
2DtunnelUB	mospar	15	18.5	2.6	3.4	-0.718	2E-8	2E-8	2E-8	144,197	575,059	1.7E+6	1.0E+7
	mixpar	18	13.3	0.6	1.0	-0.816	4E-9	8E-9	8E-9	144,757	575,618	1.4E+6	1.1E+7
	mixgpu	18	16.3	0.6	2.4	-0.816	4E-9	8E-9	8E-9	144,757	575,618	1.4E+6	1.1E+7
3DsqrexcLB	mospar	16	144.5	0.5	11.1	-125.520	1E-8	3E-8	3E-8	408,897	497,666	4.8E+6	1.1E+8
	mixpar	17	89.1	1.5	2.1	-125.523	1E-9	9E-9	9E-9	408,897	497,665	4.7E+6	1.3E+8
	mixgpu	17	51.9	1.6	3.6	-125.523	1E-9	9E-9	9E-9	408,897	497,665	4.7E+6	1.3E+8
3DsqrexcUB	mospar	18	113.9	0.9	4.8	-148.403	2E-8	3E-8	3E-8	239,652	850,178	2.7E+6	8.5E+7
	mixpar	19	89.0	10.5	1.0	-148.408	6E-10	5E-9	5E-9	239,652	850,177	2.6E+6	1.0E+8
	mixgpu	19	58.1	11.2	2.4	-148.408	6E-10	5E-9	5E-9	239,652	850,177	2.6E+6	1.0E+8
3DsqrexcUB2	mospar	24	66.6	2.2	5.4	-135.582	2E-7	3E-8	3E-8	104,910	513,656	4.0E+6	3.7E+7
	mixpar	16	36.0	3.1	0.7	-135.584	8E-10	6E-9	6E-9	88,920	497,665	3.4E+6	4.3E+7
	mixgpu	16	31.3	3.2	2.1	-135.584	8E-10	6E-9	6E-9	88,920	497,665	3.4E+6	4.3E+7
3DsqrfootLB	mospar	21	216.7	1.9	12.9	-5.557	1E-8	3E-8	3E-8	261,560	389,881	5.0E+6	1.1E+8
	mixpar	24	88.2	0.3	1.5	-5.557	2E-9	4E-9	4E-9	301,120	429,312	3.5E+6	9.5E+7
	mixgpu	24	49.3	0.3	3.0	-5.557	2E-9	5E-9	5E-9	301,120	429,312	3.5E+6	9.5E+7
3DsqrfootUB	mospar	21	100.9	2.0	7.1	-6.112	8E-9	2E-8	2E-8	91,185	648,546	2.5E+6	5.7E+7
	mixpar	19	48.4	0.4	1.1	-6.112	8E-10	8E-9	8E-9	159,042	716,402	1.8E+6	6.7E+7
	mixgpu	19	31.9	0.4	2.5	-6.112	8E-10	8E-9	8E-9	159,042	716,402	1.8E+6	6.7E+7
3DsqrfootUB2	mospar	19	33.0	1.3	3.0	-6.048	1E-8	3E-8	3E-8	65,970	430,875	2.6E+6	2.7E+7
	mixpar	19	27.7	1.2	0.5	-6.048	8E-10	8E-9	8E-9	65,176	430,080	2.5E+6	3.0E+7
	mixgpu	19	24.1	1.3	1.9	-6.048	8E-10	8E-9	8E-9	65,176	430,080	2.5E+6	3.0E+7
3DtunheadLB	mospar	30	217.0	1.3	12.8	-22.736	7E-8	1E-7	1E-6	486,495	573,281	5.8E+6	1.2E+8
	mixpar	34	170.1	0.6	2.4	-22.736	7E-9	9E-9	9E-9	488,064	574,465	5.8E+6	1.4E+8
	mixgpu	34	108.0	0.6	3.9	-22.736	1E-8	9E-9	9E-9	488,064	574,465	5.8E+6	1.3E+8
3DtunheadUB	mospar	20	110.3	1.0	5.6	-32.293	3E-8	3E-8	3E-8	274,032	980,834	3.0E+6	9.0E+7
	mixpar	19	77.0	0.6	1.1	-32.295	1E-9	9E-9	9E-9	274,032	980,833	3.0E+6	1.1E+8
	mixgpu	19	53.5	0.7	2.6	-32.295	1E-9	9E-9	9E-9	274,032	980,833	3.0E+6	1.1E+8

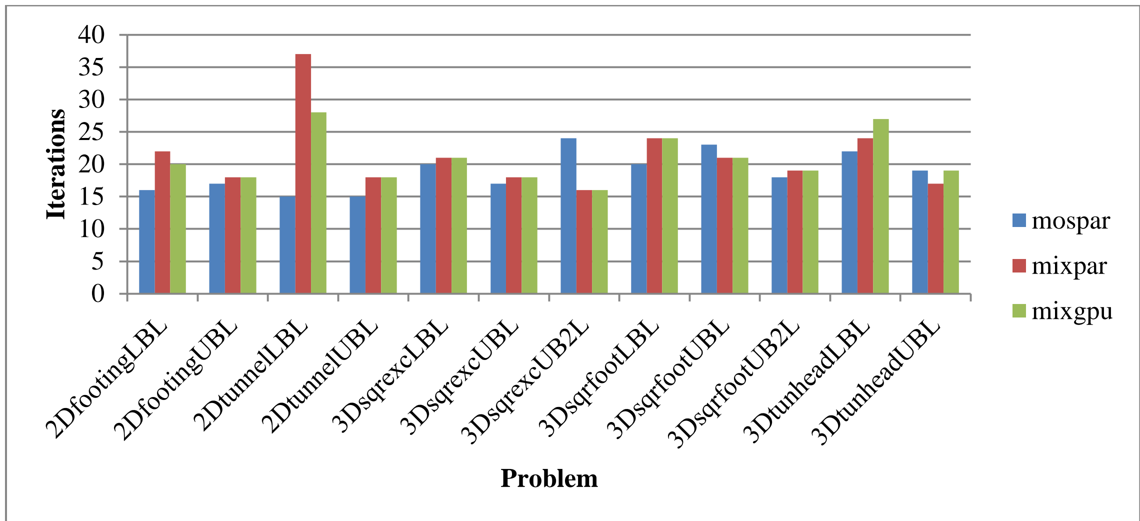


Figure 66. Comparison of the iteration count on the large problem set between the parallel solvers.

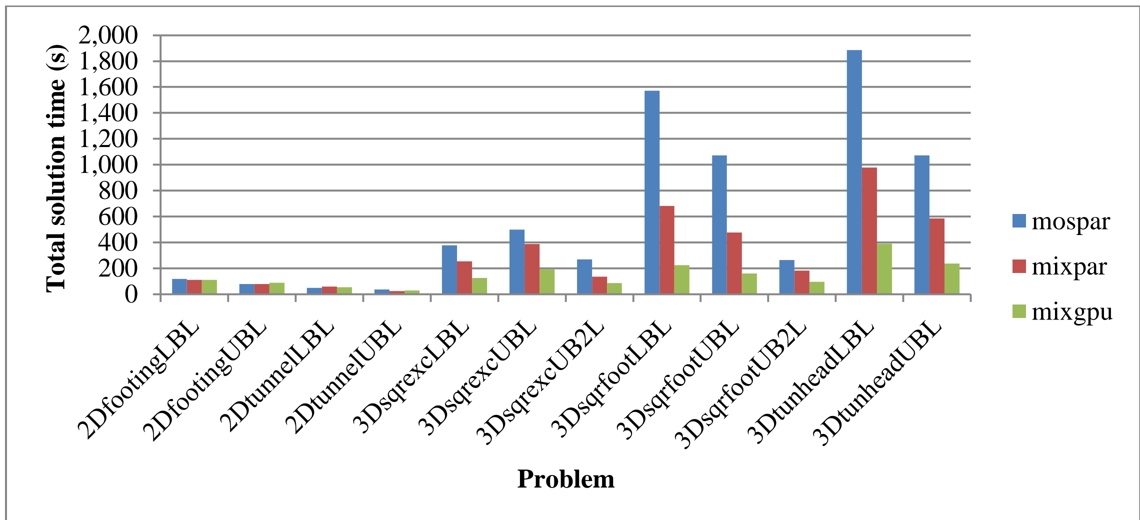


Figure 67. Comparison of the total solution time on the large problem set between the parallel solvers.

Table 33. Parallel solver results on large problems.

Problem	Solver	nit	t_T	t_P	t_O	pobj	pfeas	dfeas	relgap	m	n	nnz(A)	nnz(L)
2DfootingLB	mospar	16	117.8	1.2	39.3	-14.820	5E-9	7E-9	7E-9	1,870,680	2,105,181	1.1E+7	8.6E+7
	mixpar	22	110.5	0.9	11.3	-14.830	3E-8	1E-8	1E-8	1,870,960	2,105,460	1.1E+7	1.2E+8
	mixgpu	20	109.4	1.0	11.5	-14.830	3E-8	1E-8	1E-8	1,870,960	2,105,460	1.1E+7	1.2E+8
2DfootingUB	mospar	17	79.3	1.9	24.2	-14.868	4E-10	6E-9	3E-9	1,401,680	2,805,180	7.7E+6	6.5E+7
	mixpar	18	78.8	0.9	6.2	-14.876	8E-9	7E-9	7E-9	1,401,680	2,805,180	7.7E+6	8.6E+7
	mixgpu	18	87.5	1.1	7.8	-14.876	8E-9	7E-9	7E-9	1,401,680	2,805,180	7.7E+6	8.6E+7
2DtunnelLB	mospar	15	48.8	3.5	13.2	-0.480	5E-8	5E-8	5E-8	513,831	820,353	5.2E+6	2.8E+7
	mixpar	37	58.5	0.5	3.0	-0.800	4E-6	7E-8	7E-8	612,588	919,549	3.7E+6	3.7E+7
	mixgpu	28	52.5	0.6	4.4	-0.694	5E-6	7E-6	7E-6	612,588	919,549	3.7E+6	3.7E+7
2DtunnelUB	mospar	15	36.0	4.6	6.9	-0.581	4E-8	2E-8	2E-8	301,965	1,068,447	3.0E+6	2.0E+7
	mixpar	18	24.7	1.0	1.8	-0.812	3E-9	6E-9	6E-9	257,049	1,023,530	2.5E+6	2.2E+7
	mixgpu	18	29.1	1.1	3.2	-0.812	3E-9	6E-9	6E-9	257,049	1,023,530	2.5E+6	2.2E+7
3DsqrexcLB	mospar	20	376.5	1.0	22.6	-123.869	1E-8	3E-8	3E-8	763,216	930,818	8.9E+6	2.4E+8
	mixpar	21	254.9	4.7	4.2	-123.872	4E-9	1E-8	1E-8	763,216	930,817	8.8E+6	2.7E+8
	mixgpu	21	125.8	5.1	5.7	-123.872	3E-9	1E-8	1E-8	763,216	930,817	8.8E+6	2.7E+8
3DsqrexcUB	mospar	17	498.6	2.1	13.7	-144.429	3E-8	4E-8	4E-8	573,504	2,027,522	6.6E+6	2.9E+8
	mixpar	18	386.5	57.2	2.5	-144.443	9E-10	8E-9	8E-9	573,504	2,027,521	6.2E+6	3.4E+8
	mixgpu	18	191.4	61.6	4.0	-144.443	9E-10	8E-9	8E-9	573,504	2,027,521	6.2E+6	3.4E+8
3DsqrexcUB2	mospar	24	269.2	5.2	13.9	-134.439	2E-7	2E-8	2E-10	246,277	1,215,623	9.6E+6	1.3E+8
	mixpar	16	135.0	12.1	1.7	-134.442	7E-10	6E-9	6E-9	210,304	1,179,649	8.2E+6	1.4E+8
	mixgpu	16	85.1	12.4	3.2	-134.442	7E-10	6E-9	6E-9	210,304	1,179,649	8.2E+6	1.4E+8
3DsqrfootLB	mospar	20	1571.4	7.6	51.3	-5.628	2E-8	3E-8	3E-8	914,109	1,347,406	1.6E+7	5.6E+8
	mixpar	24	681.0	1.0	5.8	-5.629	3E-9	6E-9	6E-9	1,016,784	1,449,792	1.2E+7	4.9E+8
	mixgpu	24	223.2	1.0	7.3	-5.629	4E-9	6E-9	6E-9	1,016,784	1,449,792	1.2E+7	4.9E+8
3DsqrfootUB	mospar	23	1071.4	7.7	27.1	-5.991	1E-8	3E-8	3E-8	394,485	2,287,918	7.9E+6	3.4E+8
	mixpar	21	475.5	1.3	4.2	-5.991	7E-10	6E-9	6E-9	546,242	2,439,674	6.1E+6	3.8E+8
	mixgpu	21	159.9	1.4	5.6	-5.991	7E-10	6E-9	6E-9	546,242	2,439,674	6.1E+6	3.8E+8
3DsqrfootUB2	mospar	18	264.2	4.7	12.5	-5.949	1E-8	3E-8	3E-8	221,686	1,453,115	8.9E+6	1.5E+8
	mixpar	19	183.3	4.2	1.8	-5.949	5E-10	5E-9	5E-9	220,092	1,451,520	8.7E+6	1.7E+8
	mixgpu	19	95.3	4.3	3.3	-5.949	5E-10	5E-9	5E-9	220,092	1,451,520	8.7E+6	1.7E+8
3DtunheadLB	mospar	22	1885.2	4.4	56.5	-22.737	4E-6	1E-6	1E-5	1,658,879	1,953,217	2.0E+7	6.6E+8
	mixpar	24	978.9	1.9	9.4	-22.747	7E-9	9E-9	9E-9	1,662,720	1,956,097	2.0E+7	7.2E+8
	mixgpu	27	388.6	2.2	11.0	-22.751	7E-9	1E-8	1E-8	1,662,720	1,956,097	2.0E+7	7.2E+8
3DtunheadUB	mospar	19	1071.9	3.4	24.9	-30.768	3E-8	3E-8	3E-8	948,024	3,367,442	1.0E+7	5.2E+8
	mixpar	17	584.1	2.1	4.3	-30.741	9E-10	5E-9	5E-9	948,024	3,367,441	1.0E+7	5.9E+8
	mixgpu	19	236.1	2.3	5.7	-30.774	1E-9	7E-9	7E-9	948,024	3,367,441	1.0E+7	5.9E+8

The performance profile shown in Figure 68 demonstrates that using a GPU to accelerate the factorisation can significantly reduce the time compared with parallel approaches on common multi-core processors. While not shown here, compared with the original `Mix8` solver on the medium problem set, `Mixup8` with the GPU was $39\times$ faster in total runtime. Furthermore, it is apparent that significant improvements can be achieved over commercial optimisation packages for solving large-scale finite element limit analysis problems. Table 34 shows the total time on the problem sets by the parallel solvers, with the serial `Mixup8` performance included for comparison. Using the GPU accelerated the IPM to $4.8\times$ faster than the serial `Mixup8` and over $4\times$ faster than

`MOSEK` with 4 threads on the large problem set. This increases to a speedup of $5.5\times$ over serial `Mixup8` and $4.65\times$ over `MOSEK` on the large three dimensional problems, and it is expected that the performance benefit will increase with problem size beyond the large problems tested here. The GPU results are over $2\times$ faster than the parallel MKL version.

Table 34. Total solution time using parallel IPM solvers on the test set.

	mixup8	**mospar**	**mixpar**	**mixgpu**
Small	261.6	268.5	207.3	210.3
Medium	1,323.1	1,160.2	789.9	592.5
Large	8,616.8	7,290.2	3,951.7	1,784.0

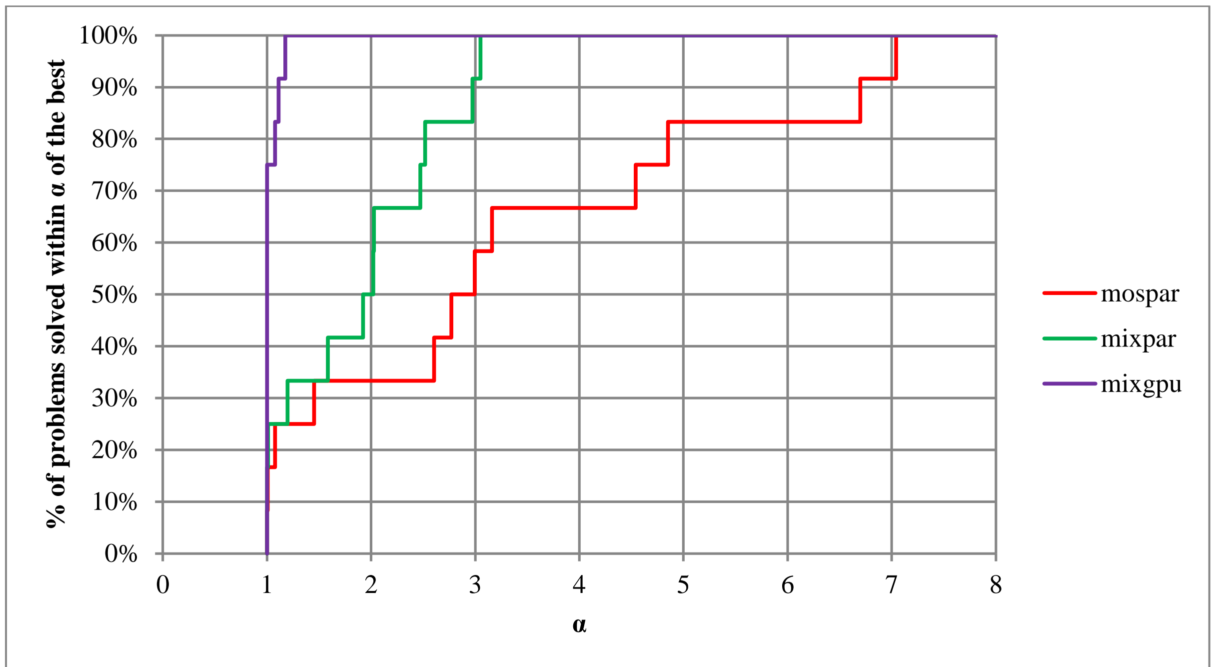


Figure 68. Performance profile of parallel solvers on large problems.

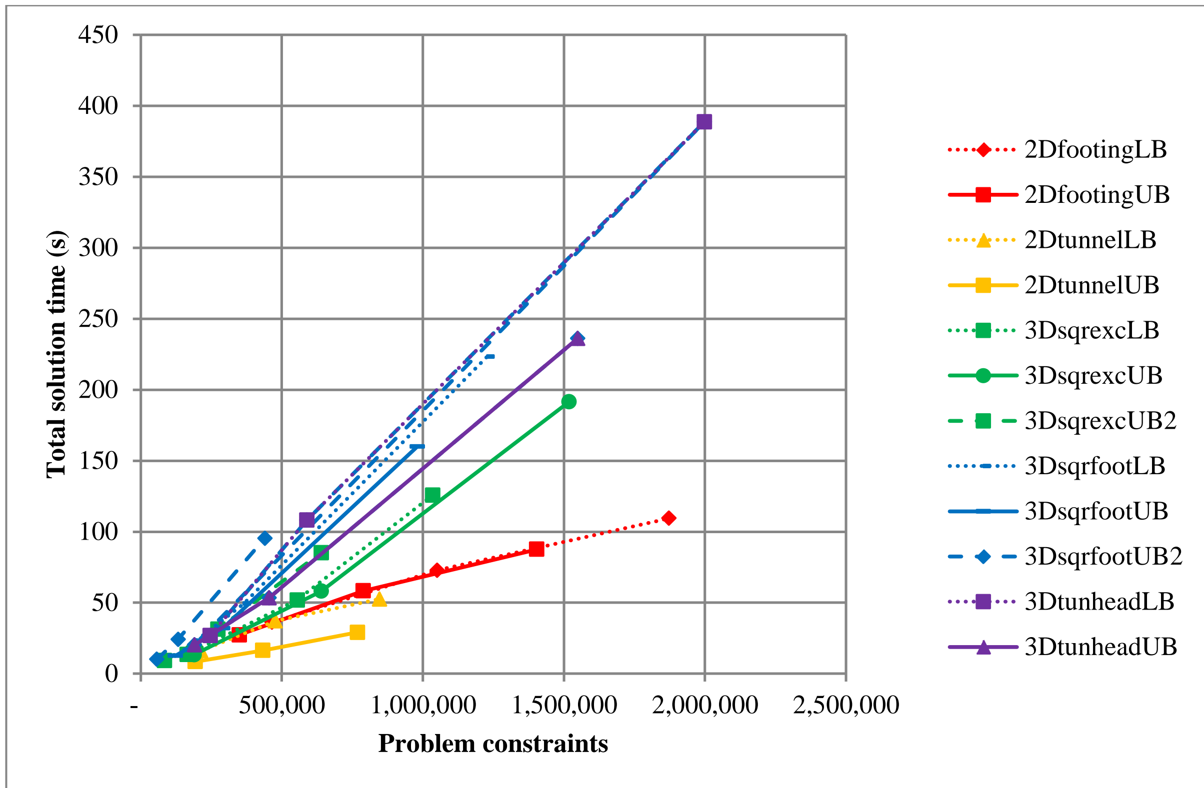


Figure 69. Total solution time versus number of constraints in original problem with `Mixup8` using the GPU with `CHOLMOD`.

Chapter 6 Conclusions and future work

This Thesis has developed and presented new methods to solve very large scale finite element limit analysis problems efficiently, including those in three dimensions, and has provided large performance gains on realistic problems against the latest state-of-the-art packages available. Improvements to the solution of the ill-conditioned linear equations, which are embedded in IPM formulations for FELA problems, including presolve routines and different sparsity-preserving orderings, led to speedups of approximately $1.5\times$ over the fastest available solver. This improvement increased to a speedup of over $4\times$ by exploiting modern highly parallel hardware. The *ab initio* development of a range of advanced preconditioned iterative linear solvers was also pursued, but ultimately proved unfruitful.

It was shown that, especially for three dimensional problems, the computational and storage burden increases rapidly as the problem size increases for FELA, presenting a major challenge to the analysis of larger and more complex designs. Comparing with the best state-of-the-art commercial solvers, substantial efficiencies could be realised through the use of a good presolve process eliminating free and fixed variables where beneficial, dealing with dense columns, and utilising a high quality sparsity-preserving nested dissection ordering in combination with a high-performance supernodal direct solver. The presolve process identified free variables that could be eliminated without incurring an increase in the number of non-zeros in the constraint matrix, which reduced the size of the problem as well as avoided the computational difficulties presented by the free variables. This elimination was performed by modifying an efficient sparse $\mathbf{LU}$ factorisation with the ability to search for pivots by degree or actual fill-in. The free variable elimination was supported with the efficient handling of any dense columns, which often arose through the free variable elimination. This allowed the sparse Cholesky factorisation to be completed without the dense columns and accounted for them afterwards. The nested dissection (ND) ordering was found to provide superior results in terms of the fill-in during factorisation at a small and amortisable increase in time over the approximate minimum degree (AMD) method. Similarly, the supernodal Cholesky factorisation provided a performance benefit over the symmetric multifrontal method. The supernodal factorisation arranged all floating point computations so that

highly optimised dense linear algebra kernels could be used, avoiding the slower accesses inherent in dealing with sparse matrices.

In seeking a reduction in the exhibited growth in computational demands as the problems became larger, especially in three dimensions, a range of iterative solvers and preconditioning approaches were tested. They were found to be very sensitive to the ordering, the drop tolerance used, and the conditioning of the system. Although some of the approaches reduced the storage demands and computational complexity, their lack of robustness and uneven performance across the problem set, particularly when approaching the optimal solution, made them unsuitable for further study. The reverse Cuthill-McKee (RCM), Sloan, ND, and AMD permutations were compared using Ajiz-Jenning's robust incomplete Cholesky factorisation (RIC1) and preconditioned conjugate gradient (PCG) on the Schur complement equation. Contrary to many published results, the approximate minimum degree ordering was found to approximately match or outperform the RCM and Sloan orderings on the test systems. A new robust incomplete method (nuRIC1) was developed to exploit the fact that the diagonal modifications of RIC1 reduce the rate of convergence and are not always necessary to compute an incomplete factor. The new method instead restarts the factorisation with an increased fraction of the diagonal modifications used if a non-positive pivot was encountered in the aim of minimising the perturbation but allowing a Cholesky factor to be computed. The two robust incomplete factorisation methods were then tested against a conventional incomplete Cholesky with threshold-based dropping and maximum fill control, and a second-order stabilised incomplete factorisation (RIC2S). The three robust methods were preferable over the conventional incomplete Cholesky, especially for the more ill-conditioned systems. Factorised sparse approximate inverses with the RCM, Sloan, or AMD orderings were not found to be competitive with the incomplete factorisations. The symmetric quasi-minimal residual (SymQMR) solver was used to test the block inverse, block-diagonal Schur, block-triangular Schur, block-diagonal augmented, and block-triangular augmented preconditioners on the augmented equations, but underperformed the methods targeting the Schur complement equation. A range of other methods aimed at reducing the impact of the increasing ill-conditioning as the IPM nears optimality were also considered but were not found to be worth developing beyond a preliminary stage. These methods

included the augmented Lagrangian Uzawa method, reduced augmented equations, and an approach that sought a sparse non-singular basis in the constraint matrix associated with small entries in the $(1,1)$ block of the augmented equations. The three best-performing approaches, RIC1, nuRIC1, and RIC2S, were compared with the best approach using a direct solver and were found to yield a reduction in the amount of storage required, but the corresponding growth in computation time and the inability to solve a number of the test problems, especially the lower bound problems, deemed them unsuitable for further development.

The best performing direct method was then used as a basis to harness the powerful features of modern parallel machines and add-on GPU devices. With the arrangement of all floating point computations into dense algebra operations, high performance kernels were exploited to reduce the time spent in computing the search direction at each iteration of the IPM. These speedups were obtained with no loss in robustness or solution quality.

6.1 Future work

While the methods outlined in this Thesis have provided a significant improvement over existing approaches, it is likely that further gains can be made through further software developments and better utilisation of available hardware.

The direct solvers which yield the greatest benefit are not designed to handle systems that are not positive-definite. Two key approaches are likely to improve the robustness of a direct solver. The first is to develop a highly parallel, symmetric indefinite solver that uses 2×2 block pivots when a zero or negative pivot is encountered, even though this leads to a non-static sparsity pattern and may result in significant non-zero growth in the factor. The second, more simple, approach is to follow the approach of Stewart [205] and modifying the pivot to a suitable value and then correct for it during the solve phase. Careful consideration would need to be given to either of these approaches so as not to destroy the parallelism exploited in the Cholesky solver.

Further benefit may be gained by seeking parallelism through the elimination tree. As all nodes on the same level of the elimination tree are independent, the updates to each of these nodes or supernodes may be carried out simultaneously. This will yield a high

number of independent tasks early on in the factorisation. The suggested approach would also provide tasks for multiple computers and GPUs, and therefore would be especially suitable for exploiting machines with greater parallel capability.

References

[1] D. C. Drucker, H. J. Greenberg, and W. Prager, "Extended limit design theorems for continuous media," *Q. Appl. Math.*, vol. 9, pp. 381–389, 1952.

[2] D. C. Drucker and W. Prager, "Soil mechanics and plastic analysis or limit design," *Q. Appl. Math.*, vol. 10, pp. 157–165, 1952.

[3] W.-F. Chen, *Limit Analysis and Soil Plasticity*. Elsevier, 1975.

[4] S. W. Sloan, "Geotechnical stability analysis," *Géotechnique*, vol. 63, no. 7, pp. 531–572, 2013.

[5] J. Lysmer, "Limit analysis of plane problems in soil mechanics," *J. Soil Mech. Found. Div.*, vol. 96, no. SM4, pp. 1311–1334, 1970.

[6] G. B. Dantzig, *Linear Programming and Extensions*. Princeton, N.J.: Princeton University Press, 1963.

[7] E. Anderheggen and H. Knöpfel, "Finite element limit analysis using linear programming," *Int. J. Solids Struct.*, vol. 8, pp. 1413–1431, 1972.

[8] J. Pastor, "Analyse limite : détermination numerique de solutions statistiques completes. Application au talus vertical," *J. Mécanique appliquée*, vol. 2, no. 2, pp. 167–196, 1978.

[9] A. Bottero, R. Negre, J. Pastor, and S. Turgeman, "Finite element method and limit analysis theory for soil mechanics problems," *Comput. Methods Appl. Mech. Eng.*, vol. 22, pp. 131–149, 1980.

[10] S. W. Sloan, "Lower bound limit analysis using finite elements and linear programming," *Int. J. Numer. Anal. Methods Eng.*, vol. 12, pp. 61–77, 1988.

[11] S. W. Sloan, "Upper bound limit analysis using finite elements and linear programming," *Int. J. Numer. Anal. Methods Eng.*, vol. 13, no. January 1987, pp. 263–282, 1989.

[12] S. W. Sloan, "A steepest edge active set algorithm for solving sparse linear programming problems," *Int. J. Numer. Methods Eng.*, vol. 26, pp. 2671–2685, 1988.

[13] S. W. Sloan and P. W. Kleeman, "Upper bound limit analysis using discontinuous velocity fields," *Comput. Methods Appl. Mech. Eng.*, vol. 127, pp. 293–314, 1995.

[14] E. Christiansen and K. O. Kortanek, "Computation of the collapse state in limit analysis using the LP primal affine scaling algorithm," *J. Comput. Appl. Math.*, vol. 34, pp. 47–63, 1991.

[15] E. Christiansen, "Computation of limit loads," *Int. J. Numer. Methods Eng.*, vol. 17, pp. 1547–1570, 1981.

[16] N. Zouain, J. Herskovits, L. A. Borges, and R. A. Feijóo, "An iterative algorithm for limit analysis with nonlinear yield functions," *Int. J. Solids Struct.*, vol. 30, no. 10, pp. 1397–1417, 1993.

[17] A. J. Abbo and S. W. Sloan, "A smooth hyperbolic approximation to the Mohr-Coulomb yield criterion," *Comput. Struct.*, vol. 54, no. 3, pp. 427–441, 1995.

[18] A. V Lyamin, "Three-dimensional lower bound limit analysis using nonlinear programming," University of Newcastle, 1999.

[19] A. V Lyamin and S. W. Sloan, "Lower bound limit analysis using non-linear programming," *Int. J. Numer. Methods Eng.*, vol. 55, no. 5, pp. 573–611, Oct. 2002.

[20] A. V Lyamin and S. W. Sloan, "Upper bound limit analysis using linear finite elements

and non-linear programming," *Int. J. Numer. Anal. Methods Geomech.*, vol. 26, no. 2, pp. 181–216, Feb. 2002.

[21] J. Pastor, T.-H. Thai, and P. Francescato, "Interior point optimization and limit analysis: an application," *Commun. Numer. Methods Eng.*, vol. 19, pp. 779–785, Sep. 2003.

[22] K. Krabbenhøft, A. V Lyamin, M. Hjiaj, and S. W. Sloan, "A new discontinuous upper bound limit analysis formulation," *Int. J. Numer. Methods Eng.*, vol. 63, no. 7, pp. 1069–1088, Jun. 2005.

[23] H. C. Suárez, "Computation of upper and lower bounds in limit analysis using second-order cone programming and mesh adaptivity," Massachusetts Institute of Technology, 2004.

[24] H. Ciria and J. Peraire, "Computation of upper and lower bounds in limit analysis using second-order cone programming and mesh adaptivity," in *9th ASCE Specialty Conference on Probabilistic Mechanics and Structural Reliability*, 2004, pp. 1–13.

[25] A. Makrodimopoulos and C. M. Martin, "Lower bound limit analysis of cohesive-frictional materials using second-order cone programming," *Int. J. Numer. Methods Eng.*, vol. 66, pp. 604–634, 2006.

[26] A. Makrodimopoulos and C. M. Martin, "Upper bound limit analysis using simplex strain elements and second-order cone programming," *Int. J. Numer. Anal. Methods Eng.*, vol. 31, pp. 835–865, 2007.

[27] E. D. Andersen and K. D. Andersen, "The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm," in *High performance optimization*, H. Frenk, K. Roos, T. Terlaky, and S. Zhang, Eds. Springer US, 2000, pp. 197–232.

[28] K. Krabbenhøft, A. V Lyamin, and S. W. Sloan, "Formulation and solution of some plasticity problems as conic programs," *Int. J. Solids Struct.*, vol. 44, pp. 1533–1549, 2007.

[29] K. Krabbenhøft, A. V Lyamin, and S. W. Sloan, "Three-dimensional Mohr-Coulomb limit analysis using semidefinite programming," *Commun. Numer. Methods Eng.*, vol. 24, pp. 1107–1119, 2008.

[30] C. M. Martin and A. Makrodimopoulos, "Finite-element limit analysis of Mohr-Coulomb materials in 3D using semidefinite programming," *J. Eng. Mech.*, vol. 134, no. 4, pp. 339–347, 2008.

[31] M. V. da Silva and A. N. Antão, "A novel augmented Lagrangian-based formulation for upper-bound limit analysis," *Int. J. Numer. Methods Eng.*, 2011.

[32] M. V. da Silva, A. N. Antão, and M. Vicente da Silva, "Upper bound limit analysis with a parallel mixed finite element formulation," *Int. J. Solids Struct.*, vol. 45, no. 22–23, pp. 5788–5804, Nov. 2008.

[33] C. D. Bisbos and P. M. Pardalos, "Second-order cone and semidefinite representations of material failure criteria," *J. Optim. Theory Appl.*, vol. 134, pp. 275–301, 2007.

[34] G. H. Golub and C. van Loan, *Matrix Computations*. The John Hopkins University Press, 1996.

[35] K. Terzaghi, *Theoretical Soil Mechanics*. John Wiley and Sons, 1943.

[36] J. F. Sturm, "Implementation of interior point methods for mixed semidefinite and second order cone optimization problems," *Optim. Methods Softw.*, vol. 17, no. 6, pp. 1105–1154, 2002.

[37] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behaviour*, 60th

Anniv. New Jersey: Princeton University Press, 2007.

[38] V. Klee and G. J. Minty, “How good is the simplex method?,” *Inequalities III*. Academic Press, New York, 1972.

[39] K. H. Borgwardt, *The Simplex Method: A Probabilistic Analysis*. Berlin: Spring Berlin Heidelberg, 1987.

[40] S. J. Wright, *Primal-Dual Interior point Methods*. Philadelphia, PA: SIAM, 1997.

[41] Y. Nesterov and A. Nemirovskii, *Interior point Polynomial Algorithms in Convex Programming*. Philadelphia: SIAM, 1994.

[42] A. Ben-Tal and A. Nemirovski, *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. Philadelphia: SIAM, 2001.

[43] J. Peng, C. Roos, and T. Terlaky, *Self-Regularity: A New Paradigm for Primal-Dual Interior point Algorithms*. Princeton: Princeton University Press, 2002.

[44] L. G. Khachiyan, “A polynomial algorithm in linear programming,” *Sov. Math. Dokl.*, vol. 20, pp. 191–194, 1979.

[45] N. Karmarkar, “A new polynomial-time algorithm for linear programming,” *Combinatorica*, vol. 4, no. 4, pp. 373–395, 1984.

[46] P. E. Gill, W. Murray, M. A. Saunders, J. A. Tomlin, and M. H. Wright, “On projected Newton barrier methods for linear programming and an equivalence to Karmarkar’s projective method,” *Math. Program.*, vol. 36, pp. 183–209, 1986.

[47] C. Roos and J.-P. Vial, “A polynomial method of approximate centers for linear programming,” *Math. Program.*, vol. 54, pp. 295–305, 1992.

[48] J. Renegar, “A polynomial-time algorithm, based on Newton’s method, for linear programming,” *Math. Program.*, vol. 40, pp. 59–93, 1988.

[49] L. McLinden, “An analogue of moreau’s proximation theorem, with application to the nonlinear complementarity problem,” *Pacific J. Math.*, vol. 88, no. 1, pp. 101–162, 1980.

[50] N. Megiddo, “Pathways to the optimal set in linear programming,” in *Progress in Mathematical Programming*, N. Megiddo, Ed. New York: Springer-Verlag, 1989, pp. 131–158.

[51] R. D. C. Monteiro and I. Adler, “Interior path following primal-dual algorithms. Part I: Linear programming,” *Math. Program.*, vol. 44, pp. 27–41, 1989.

[52] K. Tanabe, “Centered Newton method for mathematical programming,” in *System Modelling and Optimization*, M. Iri and K. Yajima, Eds. Springer Berlin Heidelberg, 1988, pp. 197–206.

[53] M. Kojima, S. Mizuno, and A. Yoshise, “A primal-dual interior point algorithm for linear programming,” in *Progress in Mathematical Programming*, New York: Springer-Verlag, 1989, pp. 29–47.

[54] S. Mehrotra, “On the implementation of a primal-dual interior point method,” *SIAM J. Optim.*, vol. 2, no. 4, pp. 575–601, 1992.

[55] J. Gondzio, “Multiple centrality corrections in a primal-dual method for linear programming,” *Comput. Optim. Appl.*, vol. 6, pp. 137–156, 1996.

[56] J.-P. Haeberly, M. V Nayakkankuppam, and M. L. Overton, “Extending Mehrotra and Gondzio higher order methods to mixed semidefinite-quadratic-linear programming,” *Optim. Methods Softw.*, vol. 11, no. 1–4, pp. 67–90, Jan. 1999.

[57] E. D. Andersen, C. Roos, and T. Terlaky, “On implementing a primal-dual interior point method for conic quadratic optimization,” *Math. Program.*, vol. 95, no. 2, pp. 249–277,

Feb. 2003.

[58] J. F. Sturm, “Using SeDuMi 1.02 A MATLAB toolbox for optimization over symmetric cones,” *Optim. Methods Softw.*, vol. 11, pp. 625–653, 1999.

[59] E. D. Andersen, K. D. Andersen, A. Ben-Tal, B. Borchers, R. M. Freund, K. Fujisawa, M. Fukuda, Y. Lo Keung, M. Kojima, E. K. Lee, Z.-Q. Luo, T. Margalit, J. E. Mitchell, S. Mizuno, K. Nakata, A. Nemirovski, Y. Nesterov, J. Peng, K. Roos, J. F. Sturm, T. Terlaky, S. A. Vavasis, Y. Ye, and J. Zou, *High performance optimization techniques*. Boston: Kluwer Academic Publishers, 1999.

[60] M. Kojima, N. Megiddo, T. Noma, and Y. Akiko, *A Unified Approach to Interior Point Algorithms for Linear Complementarity Problems*, no. October. Springer-Verlag, 1991.

[61] Y. Ye, M. J. Todd, and S. Mizuno, “An $O(sqrt(n))$-iteration homogeneous and self-dual linear programming algorithm,” *Math. Oper. Res.*, vol. 19, no. 1, pp. 53–67, 1994.

[62] X. Xu, P.-F. Hung, and Y. Ye, “A simplified homogeneous and self-dual linear programming algorithm and its implementation,” *Ann. Oper. Res.*, vol. 62, pp. 151–171, 1996.

[63] M. J. Todd, “A study of search directions in primal-dual interior point methods for semidefinite programming,” *Optim. Methods Softw.*, vol. 11, pp. 517–534, 1999.

[64] I. Adler and F. Alizadeh, “Primal-dual interior point algorithms for convex quadratically constrained and semidefinite optimization problems,” Rutgers Center for Operations Research, Rutcor, 1995.

[65] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz, “An interior point method for semidefinite progamming,” *SIAM J. Optim.*, vol. 6, pp. 342–361, 1996.

[66] Y. E. Nesterov and M. J. Todd, “Self-scaled barriers and interior point methods for convex programming,” *Math. Oper. Res.*, vol. 22, pp. 1–42, 1997.

[67] Y. E. Nesterov and M. J. Todd, “Primal-dual interior point methods for self-scaled cones,” *SIAM J. Optim.*, vol. 8, pp. 324–362, 1998.

[68] T. Tsuchiya, “A polynomial primal-dual path-following algorithm for second-order cone programming,” The Institute for Statistical Mathematics, Tokyo, 1997.

[69] T. Tsuchiya, “A convergence analysis of the scaling-invariant primal-dual path-following algorithms for second-order cone programming,” *Optim. Methods Softw.*, vol. 11/12, pp. 141–182, 1999.

[70] K.-C. Toh and M. J. Todd, “Solving semidefinite-quadratic-linear programs using SDPT3,” *Optim. Methods Softw.*, vol. 11, pp. 545–581, 1999.

[71] M. Benzi, G. H. Golub, and J. Liesen, “Numerical solution of saddle point problems,” *Acta Numer.*, vol. 14, pp. 1–137, 2005.

[72] A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, vol. 39, no. 159. New Jersey: Prentice-Hall Inc., 1981.

[73] C. Mészáros, “On free variables in interior point methods,” *Optim. Methods Softw.*, vol. 4, pp. 121–139, 1998.

[74] K. Kobayashi, K. Nakata, and M. Kojima, “A Conversion of an SDP Having Free Variables into the Standard Form SDP,” Tokyo, 2005.

[75] MOSEK ApS, “The MOSEK command line tool Version 7.0 (Revision 106),” Denmark, 2013.

[76] Z. Cai and K.-C. Toh, “Solving second order cone programming via a reduced augmented system approach,” *SIAM J. Optim.*, vol. 17, no. 3, pp. 711–737, 2006.

[77] M. F. Anjos and S. Burer, “On Handling Free Variables in Interior point Methods for Conic Linear Optimization,” *SIAM J. Optim.*, vol. 18, no. 4, pp. 1310–1325, Jan. 2008.

[78] M. Benzi, “Preconditioning techniques for large linear systems: A survey,” *J. Comput. Phys.*, vol. 182, pp. 418–477, 2002.

[79] I. S. Duff and C. Rutherford, “MA57 - A code for the solution of sparse symmetric and indefinite systems,” *ACM Trans. Math. Softw.*, vol. 30, no. 2, pp. 118–144, 2004.

[80] M. Hestenes and E. Stiefel, “Methods of conjugate gradients for solving linear systems,” *J. Res. Natl. Bur. Stand. (1934).*, vol. 49, no. 6, pp. 409–436, 1952.

[81] K.-C. Toh, M. J. Todd, and R. H. Tutuncu, “SDPT3 — a Matlab software package for semidefinite programming,” *Math. Program.*, vol. 95, pp. 189–217, 2003.

[82] R. W. Freund and N. M. Nachtigal, “A new Krylov subspace method for symmetric indefinite linear systems,” in *Proceedings of the 14th IMACS World Congress*, 1994.

[83] C. Mészáros, “On numerical issues of interior point methods,” *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 1, pp. 223–235, 2008.

[84] I. S. Duff, A. Erisman, and J. Reid, *Direct Methods for Sparse Matrices*. Oxford University Press, 1986.

[85] T. A. Davis, *Direct Methods for Sparse Linear Systems*. Philadelphia: Society for Industrial and Applied Mathematics, 2006.

[86] L. Trefethen and D. Bau, *Numerical Linear Algebra*. SIAM, 1997.

[87] E. Polizzi and A. H. Sameh, “A parallel hybrid banded system solver: the SPIKE algorithm,” *Parallel Comput.*, vol. 32, pp. 177–194, 2006.

[88] C. Ashcraft, R. Grimes, and J. Lewis, “Accurate symmetric indefinite linear equation solvers,” *SIAM J. Matrix Anal. Appl.*, vol. 20, no. 2, pp. 513–561, 1998.

[89] R. C. Whaley, A. Petitet, and Jack, “Automated empirical optimization of software and the ATLAS project,” *Parallel Comput.*, vol. 27, no. 1–2, pp. 3–35, 2001.

[90] Q. Wang, X. Zhang, Y. Zhang, and Q. Yi, “AUGEM: Automatically Generate High Performance Dense Linear Algebra Kernels on x86 CPUs,” *Proc. Int. Conf. High Perform. Comput. Networking, Storage Anal.*, pp. 1–12, 2013.

[91] D. J. Rose, R. E. Tarjan, and G. S. Lueker, “Algorithmic aspects of vertex elimination on graphs,” *SIAM J. Comput.*, vol. 5, no. 2, pp. 266–283, 1976.

[92] R. Schreiber, “A New Implementation of Sparse Gaussian Elimination,” *ACM Trans. Math. Softw.*, vol. 8, no. 3, pp. 256–276, 1982.

[93] J. R. Gilbert and T. Peierls, “Sparse partial pivoting in time proportional to arithmetic operations,” *SIAM J. Sci. Stat. Comput.*, vol. 9, no. 5, pp. 862–874, 1988.

[94] S. Parter, “The use of linear graphs in Gauss elimination,” *SIAM Rev.*, vol. 3, no. 2, pp. 119–130, 1961.

[95] T. A. Davis and W. W. Hager, “Dynamic supernodes in sparse Cholesky update/downdate and triangular solves,” *ACM Trans. Math. Softw.*, vol. 35, no. 4, 2009.

[96] B. M. Irons, “A frontal solution program for finite element analysis,” *Int. J. Numer. Methods Eng.*, vol. 2, pp. 5–32, 1970.

[97] I. S. Duff and J. K. Reid, “The multifrontal solution of indefinite sparse symmetric linear equations,” *ACM Trans. Math. Softw.*, vol. 9, no. 3, pp. 302–325, 1983.

[98] I. S. Duff, “Parallel implementation of multifrontal schemes,” *Parallel Comput.*, vol. 3, pp. 193–204, 1986.

[99] P. R. Amestoy, I. Duff, and J. L'Excellent, "Multifrontal parallel distributed symmetric and unsymmetric solvers," *Comput. Methods Appl. Mech. Eng.*, vol. 184, pp. 501–520, 2000.

[100] O. Schenk and K. Gärtner, "Solving unsymmetric sparse systems of linear equations with PARDISO," *Futur. Gener. Comput. Syst.*, vol. 20, pp. 475–487, 2004.

[101] N. I. M. Gould, Y. Hu, and J. A. Scott, "A numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations," Council for the Central Labaratory of the Research Councils, 2005.

[102] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, "Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate," *ACM Trans. Math. Softw.*, vol. 35, no. 3, 2008.

[103] I. The MathWorks, "MATLAB and Statistics Toolbox Release 2012b." Natick, Massachusetts.

[104] C. Mészáros, "Fast Cholesky factorization for interior point methods of linear programming," *Comput. Math. with Appl.*, vol. 31, no. 4–5, pp. 49–54, 1996.

[105] N. I. M. Gould, J. A. Scott, and Y. Hu, "A numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations," *ACM Trans. Math. Softw.*, vol. 33, no. 2, Jun. 2007.

[106] F. Dobrian, G. Kumfert, and A. Pothen, "The design of sparse direct solvers using object-oriented techniques," in *Advances in Software Tools for Scientific Computing*, H. P. Langtangen, A. M. Brauset, and E. Quak, Eds. 2000, pp. 89–131.

[107] H. van der Vorst, *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, 2003.

[108] P. Matstoms, "Sparse QR factorization in MATLAB," *ACM Trans. Math. Softw.*, vol. 20, no. 1, pp. 136–159, 1994.

[109] P. R. Amestoy, I. S. Duff, and C. Puglisi, "Multifrontal QR factorization in a multiprocessor environment," *Numer. Linear Algebr. with Appl.*, vol. 3, no. 4, pp. 275–300, 1996.

[110] M. Yannakakis, "Computing the minimum fill-in is NP-Complete," *SIAM J. Algebr. Discret. Methods*, vol. 2, no. 1, pp. 77–79, 1981.

[111] H. M. Markowitz, "The elimination form of the inverse and its application to linear programming," *Manage. Sci.*, vol. 3, no. 3, pp. 255–269, 1957.

[112] A. George and J. W. H. Liu, "The evolution of the minimum degree ordering algorithm," *SIAM Rev.*, vol. 31, no. 1, pp. 1–19, 1989.

[113] P. R. Amestoy, Enseeiht-Irit, T. A. Davis, and I. S. Duff, "Algorithm 837: AMD, An approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 381–388, 2004.

[114] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM J. Matrix Anal. Appl.*, vol. 17, no. 4, pp. 886–905, 1996.

[115] C. Mészáros, "The efficient implementation of interior point methods for linear programming and their applications," Eötvös Loránd University of Sciences, 1996.

[116] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Jan. 1998.

[117] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 24th National Conference of the Association for Computing*

Machinery, 1969, pp. 157–172.

[118] J. A. George, “Computer implementation of the finite-element method,” Stanford University, 1971.

[119] R. W. Vuduc, “Automatic performance tuning of sparse matrix kernels,” University of California, Berkeley, 2003.

[120] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhour, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd ed. SIAM, 1994.

[121] D. Ruizf, M. Arioli, I. S. Duff, and D. Ruiz, “Stopping criteria for iterative solvers,” *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 1, pp. 138–144, 1992.

[122] R. D. Skeel, “Iterative refinement implies numerical stability for Gaussian elimination,” *Math. Comput.*, vol. 35, no. 151, pp. 817–832, 1980.

[123] A. Greenbaum, *Iterative Methods for Solving Linear Systems*. SIAM, 1997.

[124] S. Mizuno and F. Jarre, “Global and polynomial-time convergence of an infeasible interior point algorithm using inexact computation,” *Math. Program.*, vol. 84, pp. 105–122, 1999.

[125] R. W. Freund, F. Jarre, and S. Mizuno, “Convergence of a class of inexact interior point algorithms for linear programs,” *Math. Oper. Res.*, vol. 24, pp. 105–122, 1999.

[126] J. Korzak, “Convergence analysis of inexact infeasible-interior point algorithms for solving linear programming problems,” *SIAM J. Optim.*, vol. 11, pp. 133–148, 2000.

[127] R. D. S. Monteiro and J. W. O’Neal, “Convergence analysis of long-step primal-dual infeasible interior point LP algorithm based on iterative linear solvers,” School of ISyE, Georgia Institute of Technology, 2003.

[128] G. Al-jeiroudi, “On Inexact Newton Directions in Interior Point Methods for Linear Optimization,” University of Edinburgh, 2009.

[129] G. Al-Jeiroudi, J. Gondzio, and J. Hall, “Preconditioning indefinite systems in interior point methods for large scale linear optimisation,” *Optim. Methods Softw.*, vol. 23, no. 3, pp. 345–363, 2008.

[130] W. Wang and D. O’Leary, “Adaptive use of iterative methods in predictor-corrector interior point methods for linear programming,” *Numer. Algorithms*, vol. 25, pp. 387–406, 2000.

[131] L. Bergamaschi, J. Gondzio, M. Venturin, and G. Zilli, “Inexact constraint preconditioners for linear systems arising in interior point methods,” *Comput. Optim. Appl.*, vol. 36, pp. 137–147, 2007.

[132] M. Kojima, M. Shida, and S. Shindoh, “Search directions in the SDP and the monotone SDLCP: Generalization and inexact computation,” *Math. Program.*, vol. 85, pp. 51–80, 1999.

[133] G. Zhou and K.-C. Toh, “Polynomiality of an inexact infeasible interior point algorithm for semidefinite programming,” *Math. Program.*, vol. 99, pp. 261–282, 2004.

[134] K.-C. Toh, “Solving large scale semidefinite programs via an iterative solver on the augmented systems,” *SIAM J. Optim.*, vol. 14, no. 3, pp. 670–698, 2003.

[135] J. W. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart, “Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization,” *Math. Comput.*, vol. 30, no. 136, pp. 772–795, 1976.

[136] Y. Saad, *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[137] C. Lanczos, “An iteration method for the solution of the eigenvalue problem of linear differential and integral operators,” *J. Res. Natl. Bur. Stand. (1934).*, vol. 45, no. 4, pp. 255–282, 1950.

[138] K. J. Arrow, L. Hurwicz, and H. Uzawa, *Studies in Linear and Non-linear Programming*. Stanford: Stanford University Press, 1958.

[139] H. C. Elman and G. H. Golub, “Inexact and preconditioned {U}zawa algorithms for saddle point problems,” *SIAM J. Numer. Anal.*, vol. 31, no. 6, pp. 1645–1661, 1994.

[140] S.-X. Miao, “A modified inexact Uzawa algorithm for generalized saddle point problems,” *Int. J. Comput. Math. Sci.*, vol. 4, no. 7, pp. 349–351, 2010.

[141] Y. Saad and M. Schultz, “GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems,” *SIAM J. Sci. Stat. Comput.*, vol. 7, no. 3, pp. 856–869, 1986.

[142] A. Greenbaum and Z. Strakoš, “Predicting the behavior of finite precision Lanczos and conjugate gradient computations,” *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 1, pp. 121–137, 1992.

[143] H. S. Dollar, N. I. M. Gould, W. H. A. Schilders, and A. J. Wathen, “Implicit-factorization preconditioning and iterative solvers for regularized saddle-point systems,” *SIAM J. Matrix Anal. Appl.*, vol. 28, no. 1, pp. 170–189, 2006.

[144] Y. Saad, M. Yeung, J. Erhel, and F. GuyoMarc’h, “A deflated version of the Conjugate Gradient algorithm,” *SIAM J. Sci. Comput.*, vol. 21, no. 5, pp. 1909–1926, 2000.

[145] C. C. Paige and M. A. Saunders, “Solution of sparse indefinite systems of linear equations,” *SIAM J. Numer. Anal.*, vol. 12, no. 4, pp. 617–629, 1975.

[146] C. C. Paige, “Computational variants of the Lanczos method for the eigenproblem,” *J. Inst. Math. its Appl.*, vol. 10, pp. 373–381, 1972.

[147] G. L. G. Sleijpen, H. A. van der Vorst, and J. A. N. Modersitzki, “Differences in the effects of rounding errors in Krylov solvers for symmetric indefinite linear systems,” *SIAM J. Matrix Anal. Appl.*, vol. 22, no. 3, pp. 726–751, 2000.

[148] M. Embree, “The Tortoise and the Hare restart GMRES,” *SIAM Rev.*, vol. 45, no. 2, pp. 259–266, 2003.

[149] Y. Saad, “A flexible inner-outer preconditioned GMRES algorithm,” *SIAM J. Sci. Comput.*, vol. 14, no. 2, pp. 461–469, 1993.

[150] H. Walker and L. Zhou, “A simpler GMRES,” *Numer. Linear Algebr. with Appl.*, vol. 1, no. 6, pp. 571–581, 1994.

[151] A. H. Baker, E. R. Jessup, and T. A. Manteuffel, “A technique for accelerating the convergence of restarted GMRES,” *SIAM J. Matrix Anal. Appl.*, vol. 26, no. 4, pp. 962–984, 2005.

[152] H. Walker, “Implementation of the GMRES method using Householder transformations,” *SIAM J. Sci. Stat. Comput.*, vol. 9, no. 1, pp. 152–163, 1988.

[153] H. van der Vorst and C. Vuik, “The superlinear convergence behaviour of GMRES,” *J. Comput. Appl. …*, vol. 48, pp. 327–341, 1993.

[154] D. M. Young and K. Jea, “Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods,” *Linear Algebra Appl.*, vol. 34, pp. 159–194, 1980.

[155] O. Axelsson, “Conjugate Gradient type methods for unsymmetric and inconsistent systems of linear equations,” *Linear Algebra Appl.*, vol. 29, pp. 1–16, 1980.

[156] A. Greenbaum, V. Pták, and Z. Strakoš, “Any nonincreasing convergence curve is

possible for GMRES," *SIAM J. Matrix Anal. …*, vol. 17, no. 3, pp. 465–469, 1996.

[157] Y. Saad, "Further analysis of minimum residual iterations," Minnesota Supercomputer Institute, University of Minnesota, 1997.

[158] R. W. Freund and N. M. Nachtigal, "QMR: a quasi-minimal residual method for non-Hermitian linear systems," *Numer. Math.*, vol. 60, pp. 315–339, 1991.

[159] C. Lanczos, "Solution of systems of linear equations by minimized iterations," *J. Res. Natl. Bur. Stand. (1934).*, vol. 49, no. 1, pp. 33–53, 1952.

[160] N. M. Nachtigal, "A look-ahead variant of the Lanczos algorithm and its application to the Quasi-Minimal Residual method for non-Hermitian linear systems," Massachusetts Institute of Technology, 1991.

[161] R. W. Freund and N. M. Nachtigal, "An implementation of the QMR method based on coupled two-term recurrences," Research Institute for Advanced Computer Science, NASA Ames Research Center, 1992.

[162] R. W. Freund, "A transpose-free {Q}uasi-{M}inimal {R}esidual algorithm for non-{H}ermitian linear systems," *SIAM J. Sci. Comput.*, vol. 14, no. 2, pp. 470–482, 1993.

[163] K.-K. Phoon, K.-C. Toh, S. Chan, and F. Lee, "An efficient diagonal preconditioner for finite element solution of Biot's consolidation equations," *Int. J. Numer. Methods Eng.*, vol. 55, pp. 377–400, 2002.

[164] G. Gambolati, M. Ferronato, and C. Janna, "Preconditioners in computational geomechanics: A survey," *Int. J. Numer. Anal. Methods Eng.*, vol. 35, pp. 980–996, 2011.

[165] R. Weiss, "Error-minimizing Krylov subspace methods," *SIAM J. Sci. Comput.*, vol. 15, no. 3, pp. 511–527, 1994.

[166] H. A. van der Vorst, "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 13, no. 2, pp. 631–644, 1992.

[167] P. Sonneveld, "CGS, A fast Lanczos-type solver for nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 10, no. 1, pp. 36–52, 1989.

[168] D. Fokkema, G. Sleijpen, and H. van der Vorst, "Generalized Conjugate Gradient Squared," *J. Comput. Appl. Math.*, vol. 71, no. 1, pp. 125–146, 1996.

[169] G. L. G. Sleijpen, H. van der Vorst, and D. R. Fokkema, "BiCGstab(l) and other hybrid Bi-CG methods," *Numer. Algorithms*, vol. 7, pp. 75–109, 1994.

[170] S.-L. Zhang, "GPBi-CG: Generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems," *SIAM J. Sci. Comput.*, vol. 18, no. 2, pp. 537–551, 1997.

[171] K. Chen, *Matrix Preconditioning Techniques and Applications*. Cambridge University Press, 2005.

[172] E. Boman and B. Hendrickson, "Support theory for preconditioning," *SIAM J. Matrix Anal. Appl.*, vol. 25, no. 3, pp. 694–717, 2003.

[173] M. Bern, J. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo, "Support-graph preconditioners," *SIAM J. Matrix Anal. Appl.*, vol. 27, no. 4, pp. 930–951, 2006.

[174] S. Bocanegra, F. F. Campos, and A. R. L. Oliveira, "Using a hybrid preconditioner for solving large-scale linear systems arising from interior point methods," *Comput. Optim. Appl.*, vol. 36, pp. 149–164, 2007.

[175] J. A. Meijerink and H. A. van der Vorst, "An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix," *Math. Comput.*, vol. 31, no.

137, pp. 148–162, 1977.

[176] Y. Saad, “ILUT: A dual threshold incomplete LU factorization,” *Numer. Linear Algebr. with Appl.*, vol. 1, no. 4, pp. 387–402, 1994.

[177] E. Chow and Y. Saad, “Experimental study of ILU preconditioners for indefinite matrices,” Minnesota Supercomputer Institute, University of Minnesota, 1997.

[178] M. Bollhöfer, “A robust and efficient ILU that incorporates the growth of the inverse triangular factors,” *SIAM J. Sci. Comput.*, vol. 25, no. 1, pp. 86–103, 2003.

[179] Z.-Z. Bai, I. S. Duff, and A. J. Wathen, “A class of incomplete orthogonal factorization methods. I: Methods and theories,” *BIT*, vol. 41, no. 1, pp. 53–70, 2001.

[180] Y. Saad, “Preconditioning techniques for nonsymmetric and indefinite linear systems,” *J. Comput. Appl. Math.*, vol. 24, pp. 89–105, 1988.

[181] N. I. M. Gould and J. A. Scott, “Sparse approximate-inverse preconditioners using norm-minimization techniques,” *SIAM J. Sci. Comput.*, vol. 19, no. 2, pp. 605–625, 1998.

[182] M. J. Grote and T. Huckle, “Parallel preconditioning with sparse approximate inverses,” *SIAM J. Sci. Comput.*, vol. 18, no. 3, pp. 838–853, 1997.

[183] M. Benzi and M. Tůma, “A sparse approximate inverse preconditioner for nonsymmetric linear systems,” *SIAM J. Sci. Comput.*, vol. 19, no. 3, pp. 968–994, 1998.

[184] L. Kolotilina and A. Yeremin, “Factorized sparse approximate inverse preconditioners I. Theory,” *SIAM J. Matrix Anal. Appl.*, vol. 14, no. 1, pp. 45–58, 1993.

[185] M. Benzi and M. Tůma, “A comparative study of sparse approximate inverse preconditioners,” *Appl. Numer. Math.*, vol. 30, pp. 305–340, 1999.

[186] Y. A. Kuznetsov, “Efficient preconditioner for mixed finite element methods on nonmatching meshes,” *Russ. J. Numer. Anal. Math. Model.*, vol. 19, no. 2, pp. 163–172, 2004.

[187] M. Murphy, G. H. Golub, and A. Wathen, “A note on preconditioning for indefinite linear systems,” *SIAM J. Sci. Comput.*, vol. 21, no. 6, pp. 1969–1972, 2000.

[188] L. Bergamaschi, J. Gondzio, and G. Zilli, “Preconditioning indefinite systems in interior point methods for optimization,” *Comput. Optim. Appl.*, vol. 28, pp. 149–171, 2004.

[189] C. Keller, N. I. M. Gould, and A. J. Wathen, “Constraint preconditioning for indefinite linear systems,” *SIAM J. Matrix Anal. Appl.*, vol. 21, no. 4, pp. 1300–1317, 2000.

[190] J. C. Haws and C. Meyer, “Preconditioning KKT systems,” 2001.

[191] Z.-Z. Bai, M. Ng, and Z.-Q. Wang, “Constraint preconditioners for symmetric indefinite matrices,” *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 2, pp. 410–433, 2009.

[192] J. C. Haws, “Preconditioning KKT systems,” North Carolina State University, 2002.

[193] M. Rozložník and V. Simoncini, “Krylov Subspace Methods for Saddle Point Problems with Indefinite Preconditioning,” *SIAM J. Matrix Anal. Appl.*, vol. 24, no. 2, pp. 368–391, Jan. 2002.

[194] G. Gambolati, G. Pini, and M. Ferronato, “Numerical performance of projection methods in finite element consolidation models,” *Int. J. Numer. Anal. Methods Geomech.*, vol. 25, no. 14, pp. 1429–1447, Dec. 2001.

[195] M. Ferronato, C. Janna, and G. Gambolati, “Mixed constraint preconditioning in computational contact mechanics,” *Comput. Methods Appl. Mech. Eng.*, vol. 197, pp. 3922–3931, 2008.

[196] M. Ferronato, L. Bergamaschi, and G. Gambolati, “Performance and robustness of block

constraint preconditioners in finite element coupled consolidation problems," *Int. J. Numer. Methods Eng.*, vol. 81, pp. 381–402, 2010.

[197] H. Dollar, N. I. M. Gould, M. Stoll, and A. Wathen, "Preconditioning saddle-point systems with applications in optimization," *SIAM J. Sci. Comput.*, vol. 32, no. 1, pp. 249–270, 2010.

[198] G. H. Golub and C. Greif, "On solving block-structured indefinite linear systems," *SIAM J. Sci. Comput.*, vol. 24, no. 6, pp. 2076–2092, 2003.

[199] T. Rees and C. Greif, "A preconditioner for linear systems arising from interior point optimization methods," *SIAM J. Sci. Comput.*, vol. 29, no. 5, pp. 1992–2007, 2007.

[200] Y. Zeng and C. Li, "A new preconditioner with two variable relaxation parameters for saddle point linear systems with highly singular (1,1) blocks," *Am. J. Comput. Math.*, vol. 1, pp. 252–255, 2011.

[201] J.-S. Chai and K.-C. Toh, "Preconditioning and iterative solution of symmetric indefinite linear systems arising from interior point methods for linear programming," *Comput. Optim. Appl.*, vol. 36, no. 2–3, pp. 221–247, Mar. 2007.

[202] S. W. Sloan, "A FORTRAN program for profile and wavefront reduction," *Int. J. Numer. Methods Eng.*, vol. 28, no. 11, pp. 2651–2679, 1989.

[203] R. Bridson and W.-P. Tang, "A structural diagnosis of some IC orderings," *SIAM J. Sci. Comput.*, vol. 22, no. 5, pp. 1527–1532, 2000.

[204] M. Benzi and M. Tůma, "Orderings for factorized sparse approximate inverse preconditioners," *SIAM J. Sci. Comput.*, vol. 21, no. 5, pp. 1851–1868, 2000.

[205] G. W. Stewart, "Modifying pivot elements in Gaussian elimination," *Math. Comput.*, vol. 28, no. 126, pp. 537–542, 1974.

[206] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Math. Program.*, vol. 91, no. 2, pp. 201–213, 2002.

[207] E. D. Andersen, "On implementing a primal-dual interior point method for conic quadratic optimization," 2000.

[208] E. D. Andersen and K. D. Andersen, "Presolving in linear programming," *Math. Program.*, vol. 71, pp. 221–245, 1995.

[209] J. Gondzio, "Presolve analysis of linear programs prior to applying an interior point method," Geneva, Switzerland, 1994.

[210] I. S. Duff, "MA28 - A set of Fortran subroutines for sparse unsymmetric linear equations," Oxfordshire, 1980.

[211] Z. Zlatev, "On some pivotal strategies in Gaussian elimination by sparse technique," *SIAM J. Numer. Anal.*, vol. 17, no. 1, pp. 18–30, 1980.

[212] E. D. Andersen, J. Gondzio, C. Meszaros, and X. Xu, "Implementation of interior point methods for large scale linear programming," in *Interior point methods of mathematical programming*, Kluwer Academic Publishers, 1996, pp. 189–252.

[213] K. D. Andersen, "A modified Schur-complement method for handling dense columns in interior point methods for linear programming," *ACM Trans. Math. Softw.*, vol. 22, no. 3, pp. 348–356, 1996.

[214] M. Arioli, J. W. Demmel, and I. S. Duff, "Solving Sparse Linear Systems with Sparse Backward Error," *SIAM J. Matrix Anal. Appl.*, vol. 10, no. 2, pp. 165–190, 1989.

[215] A. George and J. W. Liu, "An optimal algorithm for symbolic factorization of symmetric matrices," *SIAM J. Comput.*, vol. 9, no. 3, pp. 583–593, 1980.

[216] "HSL. A collection of Fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk/." 2011.

[217] M. A. Ajiz and A. Jennings, "A robust incomplete Choleski-Conjugate Gradient algorithm," *Int. J. Numer. Methods Eng.*, vol. 20, no. April 1983, pp. 949–966, 1984.

[218] I. E. Kaporin, "High quality preconditioning of a general symmetric positive definite matrix based on its $U^TU+U^TR+R^TU$-decomposition," *Numer. Linear Algebr. with Appl.*, vol. 5, no. April 1997, pp. 483–509, 1998.

[219] M. Benzi and M. Tůma, "A robust incomplete factorization preconditioner for positive definite matrices," *Numer. Linear Algebr. with Appl.*, vol. 10, no. 5–6, pp. 385–400, Jul. 2003.

[220] T. A. Manteuffel, "An incomplete factorisation technique for positive definite linear systems," *Math. Comput.*, vol. 34, no. 150, pp. 473–497, Apr. 1980.

[221] C. Janna, M. Ferronato, F. Sartoretto, and G. Gambolati, "FSAIPACK: A software package for high-performance factored sparse approximate inverse preconditioning," *ACM Trans. Math. Softw.*, vol. 41, no. 2, pp. 10–26, 2015.

[222] K.-C. Toh, K.-K. Phoon, and S.-H. Chan, "Block preconditioners for symmetric indefinite linear systems," *Int. J. Numer. Methods Eng.*, vol. 60, pp. 1361–1381, 2004.

[223] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The Landscape of Parallel Computing Research: A View from Berkeley," 2006.

[224] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design*, 4th ed. Massachusetts: Elsevier, 2012.

[225] M. Yamashita, K. Fujisawa, and M. Kojima, "SDPARA: SemiDefinite Programming Algorithm paRAllel version," *Parallel Comput.*, vol. 29, no. 8, pp. 1053–1067, Aug. 2003.

[226] A. Gupta, S. Koric, and T. George, "Sparse matrix factorization on massively parallel computers," in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009.

[227] A. Gupta, "A shared- and distributed-memory parallel general," *Appl. Algebr. Eng. Commun. Comput.*, vol. 18, no. 3, pp. 263–277, 2007.

[228] L. Prandtl, "Über die Härte plasticher Körper," *Nachrichten von der K. Gesellschaft der Wissenschaften, Göttingen Math. Phys. Klasse*, pp. 74–85, 1920.

[229] E. H. Davis, M. J. Gunn, R. J. Mair, and H. N. Seneviratne, "The stability of shallow tunnels and underground openings in cohesive material," *Géotechnique*, vol. 30, no. 4, pp. 397–416, 1980.

[230] D. W. Wilson, A. J. Abbo, S. W. Sloan, and A. V Lyamin, "Undrained stability of a circular tunnel where the shear strength increases linearly with depth," *Canadian Geotechnical Journal*, vol. 48, no. 9. pp. 1328–1342, Sep-2011.

[231] K. Yamamoto, A. V Lyamin, D. W. Wilson, S. W. Sloan, and A. J. Abbo, "Bearing capacity of cohesive-frictional soils with a shallow circular tunnel," 2010.

[232] K. Yamamoto, A. V Lyamin, D. W. Wilson, S. W. Sloan, and A. J. Abbo, "Stability of a circular tunnel in cohesive-frictional soil subjected to surcharge loading," *Comput. Geotech.*, vol. 38, no. 4, pp. 504–514, Jun. 2011.

[233] A. Klar, A. S. Osman, and M. Bolton, "2D and 3D upper bound solutions for tunnel excavation using 'elastic' flow fields," *Int. J. Numer. Anal. Methods Geomech.*, vol. 31, pp. 1367–1374, 2007.

[234] H. S. Yu, S. W. Sloan, and P. W. Kleeman, “A quadratic element for upper bound limit analysis,” *Eng. Comput.*, vol. 11, no. 3, pp. 195–212, 1994.

[235] J. Pastor and S. Turgeman, “Limit analysis in axisymmetrical problems: numerical determination of complete statical solutions,” *Int. J. Mech. Sci.*, vol. 24, no. 2, pp. 95–117, 1982.

[236] S. Turgeman and J. Pastor, “Limit analysis: A linear formulation of the kinematic approach for axisymmetric mechanic problems,” *Int. J. Numer. Anal. Methods Eng.*, vol. 6, pp. 109–128, 1982.

[237] R. T. Shield and D. C. Drucker, “The application of limit analysis to punch-indentation problems,” *J. Appl. Mech. ASME*, vol. 75, pp. 453–460, 1953.

[238] P. A. Vermeer, N. Ruse, and T. Marcher, “Tunnel Heading Stability in Drained Ground,” *FELSBAU - Rock Soil Eng.*, vol. 20, no. 6, pp. 8–18, 2002.