PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'École Normale Supérieure de Paris

Primal-Dual Proximal Augmented Lagrangian Methods for Quadratic Programming: Theory & Implementation

Soutenue par
Antoine Bambade

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité
Informatique

Composition du jury :

Aude Rondepierre INSA de Toulouse	*Président*
Paul Goulart University of Oxford	*Rapporteur*
Ludovic Righetti New York University	*Rapporteur*
Edouard Pauwels Toulouse School of Economics	*Examaniteur*
Jean Ponce Ecole Normale Supérieure-PSL	*Directeur de thèse*
Adrien Taylor Inria Paris	*Co-encadrant invité*
Justin Carpentier Inria Paris	*Co-encadrant invité*

ENS | PSL

Remerciements

Je souhaite d'abord exprimer ma plus chaleureuse gratitude au jury qui a accepté la responsabilité d'évaluer ma thèse. J'ai souhaité en effet étudier durant mon doctorat des sujets à l'interface de différents domaines, que ce soit l'optimisation numérique, la robotique, ou l'apprentissage automatique. J'apprécie donc sincèrement que des experts de renom de tous ces domaines comme vous, Aude Rondepierre, Édouard Pauwels, Ludovic Righetti, Paul Goulart, aient accepté d'évaluer ma thèse.

Je tiens également à remercier mon directeur de thèse, Jean Ponce, pour avoir accepté ce rôle, et aussi évidemment mes encadrants académiques Justin Carpentier et Adrien Taylor. Merci de m'avoir pris comme étudiant. Je n'aurais jamais pu rêver de sujets de thèse qui me plaisent autant. Je voudrais aussi vous remercier pour vos standards absolument intransigeants en termes de rigueur, qu'ils soient sur les plans des mathématiques ou de l'implémentation algorithmique et logiciel. Votre engagement inébranlable pour mieux comprendre ce que vous étudiez constitue une ressource extraordinaire. Et votre ambition de toujours faire mieux que ce que nous pouvons déjà faire est à la fois impressionnante et inspirante. Merci donc pour tout ce que vous m'avez appris.

A ma famille, papa, maman et à ma sœur Macha, je souhaite aussi vous remercier pour votre soutien indéfectible, même dans les moments difficiles, et aussi pour votre amour inconditionnel qui m'a définitivement permis de réaliser ce parcours académique. Papa et maman, sans vous comme modèles, comme chercheurs, je n'aurais peut-être pas eu cette chance de saisir toutes ces merveilleuses opportunités. Je vous remercie donc aussi pour tout ce que vous m'avez offert.

Mon doctorat n'aurait jamais été aussi passionnant sans toutes les discussions que j'ai eu la chance d'avoir avec les autres étudiants et chercheurs des équipes Willow et Sierra de l'INRIA Paris. J'ai vraiment apprécié de discuter et de collaborer avec vous, Stéphane, Louis, Fabian, Yann, Wilson, Oumayma, Etienne, Thomas, Quentin, Benjamin, et Éloïse. Ces années intenses, stimulantes et amusantes resteront gravées dans ma mémoire.

Pouvoir échanger avec autant de personnes et dans un environnement aussi incroyable est une expérience inouïe. Francis et Jean, même si je n'ai pas directement travaillé avec vous, je tiens malgré tout à vous exprimer ma plus sincère gratitude d'avoir créé un environnement aussi exceptionnel.

Je suis aussi reconnaissant envers le corps des Ponts, des Eaux et des Forêts qui m'a permis de réaliser ma thèse dans les meilleures conditions matérielles en tant qu'ingénieur des IPEF. Enfin, je voudrais remercier l'administration du laboratoire de l'INRIA Paris pour l'aide essentielle qu'elle nous apporte à tous.

Contents

Remerciements i

1 Résumé en français 1
- 1.1 Motivations de cette thèse . . . 1
  - 1.1.1 Les applications diverses de la programmation quadratique . . . 1
  - 1.1.2 Nécessité de solutions polyvalentes et fiables . . . 3
- 1.2 Contributions . . . 4
  - 1.2.1 Résumé des contributions . . . 4
  - 1.2.2 Organisation de cette thèse . . . 5
  - 1.2.3 Publications associées et logiciels . . . 5

2 Introduction and summary of the contributions 7
- 2.1 Motivation . . . 7
  - 2.1.1 The various applications of Quadratic Programming . . . 7
  - 2.1.2 Necessity of efficient and reliable solution methods . . . 9
- 2.2 Summary of contributions . . . 10
  - 2.2.1 Organisation of this thesis . . . 10
  - 2.2.2 Associated publications and software . . . 11

3 Background on Quadratic Programming 13
- 3.1 QPs: problem statement and optimality conditions . . . 14
  - 3.1.1 Problem statement . . . 14
  - 3.1.2 Duality . . . 14
  - 3.1.3 Certificates of optimality . . . 15
  - 3.1.4 Certificates of infeasibility . . . 16
- 3.2 Active-set methods . . . 16
  - 3.2.1 The Simplex Method . . . 17
  - 3.2.2 Primal Active-set Methods . . . 18
  - 3.2.3 Dual Active-Set Methods . . . 20
  - 3.2.4 Parametric Active-Set Methods . . . 25
- 3.3 Interior-Point method . . . 28
- 3.4 ADMM based-methods . . . 30
  - 3.4.1 Primal version . . . 31
  - 3.4.2 Homogeneous self-dual embedding version . . . 33
- 3.5 Proximal Augmented Lagrangian-based methods . . . 35

3.5.1 Method of Multipliers . . . 35
3.5.2 Proximal-Augmented Lagrangian Method . . . 37
3.5.3 Primal-Dual Proximal-Augmented Lagrangian Method . . . 39
3.5.4 Discussions . . . 41
3.6 Summary . . . 43

4 ProxQP algorithm 45
4.1 ProxQP . . . 46
4.1.1 The ProxQP algorithm . . . 46
4.1.2 Globalization strategy for scheduling ϵ^k and μ^k . . . 47
4.1.3 Infeasible QPs . . . 48
4.1.4 Solving proximal sub-problems . . . 48
4.2 Convergence properties . . . 51
4.2.1 Organisation of the section . . . 51
4.2.2 Further notations and technical properties . . . 51
4.2.3 Convergence of ProxQP . . . 52
4.2.4 Proof of Theorem 3 . . . 58
4.2.5 Proof of Lemma Lemma 1 . . . 59
4.3 Software Implementation . . . 59
4.3.1 Common QP solvers . . . 59
4.3.2 The ProxSuite library . . . 60
4.4 Applications to robotics and beyond . . . 62
4.4.1 Benchmark setup and scenarios . . . 63
4.4.2 Sparse problems . . . 63
4.4.3 Dense problems . . . 68
4.4.4 Generic QPs . . . 71

5 Preliminaries on differentiable optimization for convex QPs 79
5.1 QP layer: problems statement . . . 79
5.2 QP layers design under differentiability assumptions . . . 81
5.2.1 Unrolled differentiation . . . 81
5.2.2 Implicit differentiation . . . 83
5.2.3 Alternate differentiation . . . 85
5.3 Some alternative "informative gradients" . . . 86
5.3.1 Least square estimate . . . 86
5.3.2 Randomized-smoothing . . . 87
5.3.3 Other regularizations of the cost function . . . 87

6 Differentiable Optimization for QPs 89
6.1 Introduction . . . 90
6.2 The extended conservative Jacobian for convex QPs . . . 91
6.2.1 Problem formulation . . . 92
6.2.2 The closest feasible QP . . . 92
6.2.3 The extended conservative Jacobian . . . 93
6.2.4 Deriving an extended conservative Jacobian . . . 93
6.2.5 Future work and potential improvements . . . 95
6.3 Experiments . . . 96
6.3.1 Pedagogical examples of parametric QPs . . . 96
6.3.2 Learning capabilities . . . 102
6.3.3 Timing benchmarks on standard learning models . . . 104
6.3.4 Training with large learning rates . . . 107
6.4 Proofs . . . 112

6.4.1 Proof of Lemma 5 . . . 112
6.4.2 Proof of Lemma 6 . . . 113
6.4.3 Forward and backward AD for computing ECJs . . . 114
6.4.4 Proof of Lemma 7 . . . 119
6.4.5 Experimental setting . . . 120

7 Conclusion **123**
7.1 Summary of the Thesis . . . 123
7.2 Perspectives . . . 124

Chapter 1

Résumé en français

Chapter content

1.1 Motivations de cette thèse 1
1.1.1 Les applications diverses de la programmation quadratique 1
1.1.2 Nécessité de solutions polyvalentes et fiables 3
1.2 Contributions 4
1.2.1 Résumé des contributions 4
1.2.2 Organisation de cette thèse 5
1.2.3 Publications associées et logiciels 5

1.1 Motivations de cette thèse

1.1.1 Les applications diverses de la programmation quadratique

La programmation quadratique (PQ) convexe vise à optimiser une fonction quadratique convexe tout en respectant des contraintes linéaires. Elle représente l'une des formes fondamentales de la programmation non linéaire convexe, et son importance est évidente en tant que pierre angulaire dans la boîte à outils moderne de l'ingénierie. Ci-dessous, nous présentons une compilation non exhaustive des applications de la programmation quadratique dans différents domaines scientifiques, suivie d'un résumé plus détaillé de son rôle en robotique et en apprentissage automatique.

Applications en Science

La programmation quadratique trouve une utilité particulière dans la résolution de tâches exigeant l'allocation de ressources (par exemple, la conception de structure en ingénierie, le contrôle de centrale énergétique, la planification économique, l'optimisation de portefeuille) et la reconstruction de données (par exemple, via des techniques de filtrage). Quelques applications exemplaires sont résumées ci-dessous. Le lecteur intéressé peut consulter davantage de références dans l'aperçu bibliographique suivant [Gould and Toint, 2000].

Conception de structure en ingénierie. La programmation quadratique joue un rôle essentiel pour concevoir des structures ou systèmes d'ingéniérie. Elle facilite notamment l'identification des paramètres de conception optimaux en minimisant une fonction objective quadratique tout en respectant des contraintes de structure [Deb, 2012, Shi et al., 2016, Zhou et al., 2012].

Traitement du Signal. La programmation quadratique est utilisée en traitement du signal, par exemple dans les techniques visant sa reconstruction ou son filtrage [Abdelmalek, 1983, Adams et al., 1994, Hassanien et al., 2008, Nordebo et al., 1994, Bitmead et al., 1986, Simon and Simon, 2006, Mattingley and Boyd, 2010].

Allocation de Ressources. La programmation quadratique aide à identifier des allocations optimales sous des contraintes de ressources. Des exemples notables couvrent notamment l'optimisation de la génération d'énergie [Momoh et al., 1999, Nanda et al., 1989, Zhang et al., 2018], l'optimisation de la logistique [Liu and Zhang, 2016, Matskul et al., 2021], la planification économique et managériale [SHIM, 1983, Reid and Hasdorff, 1973, Rao, 1961, Fan and Zhang, 1998, McCarl et al., 1977], la planification agricole ou la sélection des cultures [McFarquhar, 1961, Wiens, 1976, Marques et al., 2010], et l'optimisation de portefeuille pour la réduction des risques en finance [Markowitz, 1952, Cornuejols and Tütüncü, 2006, Boyd et al., 2017, Boyd and Vandenberghe, 2004].

Applications en Robotique

L'optimisation est devenue un élément clé pour simplifier et systématiser la programmation de mouvements de robots complexes. De nos jours, de nombreux problèmes robotiques, de la simulation au contrôle, en passant par la planification et l'estimation, sont formulés sous forme de problèmes d'optimisation.

La programmation quadratique est utilisée en robotique pour la planification et le contrôle des mouvements. Elle permet de trouver la trajectoire optimale ou les entrées de contrôle qui minimisent une fonction de coût quadratique tout en respectant les dynamiques et les contraintes du système.

Notamment, elle permet entre autres de traiter la modélisation de contact unilatéral sans frottement [Redon et al., 2002], la dynamique directe sous contrainte [Carpentier et al., 2021], la cinématique et la dynamique inverses pour le contrôle de tâches [Escande et al., 2014, Herzog et al., 2016, Kuindersma et al., 2016], et la locomotion bipède [Wieber, 2006a, Carpentier and Wieber, 2021, Wensing et al., 2022]. La programmation quadratique est également couramment utilisé comme sous-routine pour résoudre des problèmes plus complexes, par exemple, dans le contexte de problèmes de contrôle optimal contraints [Leineweber et al., 2003, Houska et al., 2011, Tassa et al., 2014, Jallet et al., 2022a].

Applications en Apprentissage Automatique

Dans le domaine de l'apprentissage automatique, la programmation quadratique trouve des applications dans une gamme d'algorithmes dédiés à la résolution de problèmes d'ajustement. Nous détaillons ci-dessous quelques exemples marquants de cette utilisation.

Les **Machines à Vecteurs de Support** (SVM) ont gagné une reconnaissance substantielle pour leur efficacité dans les tâches de classification. La programmation quadratique est un outil fondamental dans les SVM [Cortes and Vapnik, 1995], car elle est utilisée pour déterminer l'hyperplan optimal qui sépare efficacement les points de données de différentes classes tout en maximisant la marge entre ces classes. Cette approche contribue à obtenir des résultats de classification robustes et précis.

Les techniques statistiques de **Lasso** [Tibshirani, 1996, Candes et al., 2008] et de **raccords** via les méthodes proposées par P.J. Huber [Huber, 1992, Huber, 2004] sont d'autres exemples notables où la programmation quadratique joue un rôle essentiel. Le Lasso, abréviation de "*Least Absolute Shrinkage and Selection Operator*", est utilisé dans les tâches de régression pour encourager la parcimonie dans l'espace des solutions. La programmation quadratique assiste le processus d'optimisation, résultant en des modèles mettant exclusivement l'accent sur les caractéristiques importantes des données. D'autre part, la technique de raccord statistique proposée par P.J.

Huber, offre une robustesse exemplaire contre des valeurs aberrantes présentes dans des données d'entraînement. La programmation quadratique aide à déterminer les paramètres de l'ajustement optimal qui trouvent un équilibre entre la précision de l'ajustement et la résistance aux valeurs aberrantes.

Couches de Programmation Quadratique. L'incorporation de problèmes d'optimisation différentiables en tant que couches au sein des réseaux neuronaux est récemment devenue pratique et efficace pour résoudre certaines tâches d'apprentissage automatique de manière plus performante, voir par exemple [Geng et al., 2020, Lee et al., 2019, Le Lidec et al., 2021, Donti et al., 2017, de Avila Belbute-Peres et al., 2018, Amos et al., 2018, Bounou et al., 2021]. En effet, de telles couches permettent implicitement de capturer des connaissances spécifiques au domaine ou des connaissances a priori, contrairement aux réseaux neuronaux conventionnels. Parmi ces types de couches, les couches de programmation quadratique offrent une grande puissance de modélisation [Amos and Kolter, 2017].

1.1.2 Nécessité de solutions polyvalentes et fiables

Les applications significatives de la programmation quadratique ont été renforcées par des efforts de recherche dédiés et le développement d'outils numériques robustes. Résoudre efficacement les problèmes de programmation quadratique nécessite des considérations couvrant à la fois les méthodes d'optimisation sous-jacentes et les détails de son implémentation. Nous présentons ci-dessous un aperçu de certaines caractéristiques clés qui contribuent à l'efficacité pratique de la programmation quadratique.

Vers des méthodes d'optimisation polyvalentes

Les provrammes quadratiques convexes exigent des algorithmes spécifiques qui respectent certaines hypothèses sur le QP d'entrée fournie par les utilisateurs finaux. Pour répondre à un large éventail d'applications, un algorithme efficace doit générer des solutions sous **des hypothèses minimales**. De plus, des "hypothèses favorables" doivent donner lieu à un algorithme **polyvalent**, permettant une utilisation flexible. Considérons les scénarios suivants :

- Souvent, les programmes quadratiques servent de sous-routines pour résoudre des problèmes plus complexes, tels que la programmation quadratique séquentielle (PQS) [Nocedal and Wright, 2006, Chapitre 16]. Fréquemment, ces programmes quadratiques sont étroitement liés. Par conséquent, tirer parti des solutions antérieures pour *initialiser* les processus de résolution de PQs ultérieurs émerge comme une propriété puissante.

- Dans des contextes tels que les applications robotiques et les tâches de cinématiques inverses, obtenir une précision de solution au niveau du millimètre suffit. Par conséquent, des solutions de PQs excessivement précises peuvent s'avérer inutiles. L'incorporation de stratégies d'*arrêt anticipé* offre une autre utilité pratique.

- Troisièmement, l'approche utilisée devrait exploiter efficacement la *structure* inhérente du PQ, telle que la parcimonie. En effet, certaines applications impliquent la résolution de problèmes à grande échelle, tandis que d'autres traitent de problèmes plus compacts.

De plus, une méthode d'optimisation "idéale" devrait garantir de fournir des solutions relativement **rapidement**. De plus, cette méthode devrait offrir des formulations mathématiquement **robustes** en ce qui concerne les entrées du PQ, permettant des procédures de factorisation efficaces ou des manipulations algébriques évitant la division par des valeurs potentiellement faibles.

De plus, les méthodes d'optimisation peuvent posséder des **caractéristiques** souhaitables pour résoudre efficacement certains types de problèmes. Par exemple, le formulation intrinsèque de la méthode d'optimisation peut naturellement encourager la parallélisation, ce qui permet de prendre en charge efficacement des applications à grande échelle. Enfin, il convient de souligner que pour une adoption généralisée, la **simplicité** d'une méthode d'optimisation est essentielle. Cela implique un pseudo-code concis, une interprétabilité aisée, un nombre minimal d'hyperparamètres sous-jacents et peu d'heuristiques.

Vers des implémentations efficaces

Les *solveurs* de programmation quadratique efficaces exploitent la puissance de **routines d'algèbre linéaire hautement optimisées** avec des langages de programmation bas niveau tels que C, C++ ou Rust. Ils privilégient des conceptions **sans allocation de mémoire** pour minimiser les déplacements de mémoire pendant le processus de résolution. De plus, une "bonne implémentation" présente une **flexibilité** à travers divers aspects :

- **Programmation par modèles** : Une utilisation intelligente de la *programmation par modèles* s'adapte automatiquement à la précision des nombres à virgule flottante de l'utilisateur final, garantissant une intégration efficace et une précision transparente.
- **Capacités d'Interfaçage** : Une interface aisée avec différents systèmes d'exploitation, des langages de programmation de plus haut niveau (par exemple, Python ou Julia) ou diverses plateformes numériques (par exemple, CVXPY [Diamond and Boyd, 2016], CASADI [Andersson et al., 2019], TSID [Del Prete et al., 2016]) améliore l'utilisabilité et la polyvalence d'un *software* de programmation quadratique.
- **Fonctionnalités Pratiques** : Offrir des fonctionnalités pratiques, telle que la parallélisation pour les tâches d'apprentissage, élargit également le champ d'application.

De plus, un solveur de PQs bien conçu présente une **API intuitive et bien documentée** qui simplifie l'intégration et l'utilisation. Enfin, il est important de souligner qu'un environnement en libre accès et doté d'une licence flexible et d'une **communauté active et réactive** dote tous les praticiens des outils essentiels pour améliorer les capacités d'un solveur de programmation quadratique.

1.2 Contributions

1.2.1 Résumé des contributions

Dans cette thèse, notre attention se concentre sur la **résolution** et la **différenciation** efficaces de programmes quadratiques convexes (PQ). Nous portons un accent particulier sur les applications en robotique et en contrôle.

Nous commençons par introduire l'algorithme PROXQP. C'est une méthode de résolution de PQs capable d'exploiter différentes structures de PQ grâce à l'utilisation de techniques de Lagrangien augmenté primal-dual. Nous illustrons ses performances pratiques sur diverses expériences standards en robotique et en contrôle, avec notamment une application de contrôle en boucle fermée sur un robot réel. Bien que initialement conçu pour la robotique, nous démontrons que PROXQP rivalise également avec les performances de *solvers* de pointe pour des problèmes génériques, ce qui en fait un solveur polyvalent et applicable au-delà du domaine de la robotique. De plus, nous établissons que pour des PQ convexes, PROXQP est garanti d'avoir une convergence globale vers le PQ faisable le plus proche, une caractéristique examinée dans les applications de contrôle en boucle fermée. Remarquablement, cette propriété améliore la stabilité et la sécurité dans les techniques de contrôle en boucle-fermée.

Nous exploitons ensuite cette caractéristique pour enrichir les capacités expressives des couches de programmation quadratique. Plus précisément, nous montrons comment la différentiation à travers les solutions de PQ faisables les plus proches permet d'étendre la portée des couches de PQ apprenables. Nous montrons notamment que les nouvelles types de couches apprenables présentent des performances prédictives supérieures dans des tâches d'apprentissage conventionnelles. De plus, nous présentons des formulations alternatives qui améliorent la robustesse numérique, la vitesse et la précision pour l'entraînement de telles couches.

Sur le plan pratique, nous proposons en complément de ces contributions la bibliothèque ProxSuite comprenant deux logiciels en libre accès écrits en C++:

- ProxQP : Un solveur de PQ efficace.
- QPLayer : Une couche de PQ différentiable dotée de capacités d'apprentissage améliorées.

1.2.2 Organisation de cette thèse

Ce manuscrit est organisé comme suit. Dans Chapter 3, après avoir exposé l'énoncé du problème de programmation quadratique et les conditions pour qu'il soit bien-posé, nous donnons une introduction concise aux principales méthodes d'optimisation utilisées pour résoudre des PQs. Dans Chapter 4, nous présentons d'abord les différentes composantes de l'algorithme ProxQP, y compris sa preuve de convergence et les détails de son implémentation. Ensuite, nous présentons un parangonnage approfondis sur des problèmes robotiques et des PQs standard. Dans Chapter 5, nous présentons un résumé des principales techniques algorithmiques utilisées pour concevoir des couches de PQ différentiables efficaces. Dans Chapter 6, nous présentons une approche unifiée pour aborder la différentiabilité à la fois des PQs faisables et infaisables en introduisant notamment la notion de *Jacobien Conservatif Étendu*. Nous proposons des moyens efficaces pour le calculer dans les modes de différentiation automatique direct et indirect, et nous montrons enfin comment cette méthode permet d'entraîner une gamme plus large de couches de PQs. Nous illustrons ensuite à travers différentes comparaisons que (i) cette technique performe que les approches traditionnelles pour résoudre certaines tâches, (ii) en utilisant les approches d'apprentissage courantes QPLayer s'exécute plus rapidement que les approches actuelles de l'état de l'art et est également numériquement plus robuste. Enfin, Chapter 7 esquisse des perspectives globales et offre un point de vue personnel sur de futures directions de recherche.

1.2.3 Publications associées et logiciels

Cette thèse a donné lieu à plusieurs publications, toutes traitant de l'optimisation numérique et différentiable.

Articles de conférence

- **Antoine Bambade**, Sarah El-Kazdadi, Adrien Taylor, Justin Carpentier. ProxQP: Yet another Quadratic Programming Solver for Robotics and beyond. In *Robotics: Science and System (RSS)*, 2022;
- Wilson Jallet, **Antoine Bambade**, Nicolas Mansard, Justin Carpentier. Constrained differential dynamic programming: A primal-dual augmented lagrangian approach. In *EEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022;
- Louis Montaut, Quentin Le Lidec, **Antoine Bambade**, Vladimir Petrik, Josef Sivic, Justin Carpentier. Differentiable collision detection: a randomized smoothing approach. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Article de *Workshop*

- Wilson Jallet, **Antoine Bambade**, Nicolas Mansard, Justin Carpentier. ProxNLP: a primal-dual augmented Lagrangian solver for nonlinear programming in Robotics and beyond. In *6th Legged Robots Workshop*, 2022.

Articles soumis

- **Antoine Bambade**, Fabian Schramm, Adrien Taylor, Justin Carpentier. QPLayer: efficient differentiation of convex quadratic optimization. Submitted to *Advances in Neural Information Processing Systems*, 2023.
- **Antoine Bambade**, Fabian Schramm, Sarah El-Kazdadi, Stéphane Caron, Adrien Taylor, Justin Carpentier. ProxQP: an Efficient and Versatile Quadratic Programming Solver for Real-Time Robotics Applications and Beyond. Submitted to IEEE Transactions on Robotics (TRO) (September 2023).
- Wilson Jallet, **Antoine Bambade**, Etienne Arlaud, Sarah El-Kazdadi, Nicolas Mansard, Justin Carpentier. ProxDDP: Proximal Constrained Trajectory Optimization. Submitted to IEEE Transactions on Robotics (TRO) (September 2023).
- Wilson Jallet, **Antoine Bambade**, Fabian Schramm, Quentin Le Lidec, Nicolas Mansard, Justin Carpentier. Notes on importance sampling of the first-order gradient estimator. Communication item submitted to IEEE Transactions on Robotics (TRO) (September 2023).

Logiciel

Cette thèse a également conduit au développement de ProxSuite[1], une bibliothèque C++ et en libre accès offrant le solveur ProxQP et QPLayer pour résoudre et différencier efficacement des programmes quadratiques. Cette nouvelle bibliothèque compte déjà environ 250 000 téléchargements et est désormais présente dans de nombreuse autres plateformes logicielles (par exemple, Cvxpy, Casadi, TSID).

[1] https://github.com/Simple-Robotics/proxsuite

Chapter 2

Introduction and summary of the contributions

Chapter content

2.1 Motivation **7**
2.1.1 The various applications of Quadratic Programming 7
2.1.2 Necessity of efficient and reliable solution methods 9
2.2 Summary of contributions **10**
2.2.1 Organisation of this thesis 10
2.2.2 Associated publications and software 11

2.1 Motivation

2.1.1 The various applications of Quadratic Programming

Convex Quadratic Programming (QP) is concerned with optimizing a convex quadratic objective function while adhering to linear constraints. It represents one of the fundamental forms of convex non-linear programming, and its significance is evident as a cornerstone in the modern engineering toolkit. Below, we outline a non-exhaustive compilation of Quadratic Programming applications across various scientific domains, followed by a more detailed summary of its role in Robotics and Machine Learning, the two application domains at the core of this thesis.

Applications in Science

Quadratic Programming finds extensive utility in addressing tasks that demand resource allocation (e.g., engineering design, power generation, economic planning, portfolio optimization) and data reconstruction (e.g., filtering techniques). A few exemplar applications are summarized below. The interested reader can get more references in the following bibliography overview [Gould and Toint, 2000].

Engineering Design. Quadratic Programming plays a pivotal role in engineering design challenges, such as optimizing the design of structures or systems. It facilitates the identification of optimal design parameters by minimizing a quadratic objective function while adhering to design constraints [Deb, 2012, Shi et al., 2016, Zhou et al., 2012].

Signal Processing. Quadratic Programming is harnessed in signal processing endeavors, encompassing signal reconstruction and filtering applications [Abdelmalek, 1983, Adams et al., 1994, Hassanien et al., 2008, Nordebo et al., 1994, Bitmead et al., 1986, Simon and Simon, 2006, Mattingley and Boyd, 2010].

Resource Allocation. Quadratic Programming aids in identifying optimal allocations under resource limitations. Noteworthy instances span diverse domains, including power generation optimization for electrical utilities [Momoh et al., 1999, Nanda et al., 1989, Zhang et al., 2018], logistics optimization [Liu and Zhang, 2016, Matskul et al., 2021], corporate and economic planning [SHIM, 1983, Reid and Hasdorff, 1973, Rao, 1961, Fan and Zhang, 1998, McCarl et al., 1977], agricultural decisions such as crop selection [McFarquhar, 1961, Wiens, 1976, Marques et al., 2010], and portfolio optimization for risk reduction in finance [Markowitz, 1952, Cornuejols and Tütüncü, 2006, Boyd et al., 2017, Boyd and Vandenberghe, 2004].

Applications in robotics

Optimization has become a key enabler to simplify and systematize the programming of complex robot movements. Many modern robotic problems ranging from simulation, control, planning, and estimation are framed as optimization problems.

Quadratic programming is used in robotics for motion planning and control. It helps find the optimal trajectory or control inputs that minimize a quadratic cost function while satisfying system dynamics and constraints. Among others, it allows to model friction-less unilateral contact modelling [Redon et al., 2002], constrained forward dynamics [Carpentier et al., 2021], inverse kinematics and dynamics for task control [Escande et al., 2014, Herzog et al., 2016, Kuindersma et al., 2016], and legged locomotion [Wieber, 2006a, Carpentier and Wieber, 2021, Wensing et al., 2022]. QPs are also commonly used as subroutines for solving more complex problems, for instance, in the context of constrained optimal control problems [Leineweber et al., 2003, Houska et al., 2011, Tassa et al., 2014, Jallet et al., 2022a]

Applications in Machine learning

In the realm of machine learning, quadratic programming finds application in a range of algorithms dedicated to solving fitting problems. Some prominent instances include **Support Vector Machines** (SVMs), **Lasso**, and **Huber fitting** techniques.

Support Vector Machines (SVMs) have gained substantial recognition for their efficiency in classification tasks. Quadratic programming is a fundamental tool in SVMs [Cortes and Vapnik, 1995], employed to determine the optimal hyperplane that effectively separates data points from different classes while maximizing the margin between these classes. This approach helps achieving robust and accurate classification outcomes.

Lasso [Tibshirani, 1996, Candes et al., 2008] and **Huber fitting** [Huber, 1992, Huber, 2004] are additional noteworthy examples where quadratic programming plays a pivotal role. Lasso, short for "Least Absolute Shrinkage and Selection Operator," is employed in regression tasks to encourage sparsity in the solution space. Quadratic programming assists in the optimization process, resulting in models that emphasize important features while discarding less significant ones. On the other hand, Huber fitting offers robustness against outliers in data by applying an appropriate loss function. Quadratic programming aids in determining the optimal fitting parameters that strike a balance between fitting accuracy and resistance to outliers.

Incorporating differentiable optimization problems as layers within neural networks has recently become practical and effective for solving certain machine learning tasks, see, for instance [Geng et al., 2020, Lee et al., 2019, Le Lidec et al., 2021, Donti et al., 2017, de Avila Belbute-Peres et al., 2018, Amos et al., 2018, Bounou et al., 2021]. Indeed, such layers implicitly allow for capturing useful domain-specific knowledge or priors contrary to conventional neural networks.

Among such types of layers, **Quadratic Programming layers** offer a rich modeling power [Amos and Kolter, 2017].

2.1.2 Necessity of efficient and reliable solution methods

Dedicated research efforts and the development of robust numerical tools have bolstered the significant applications of quadratic programming. Effectively solving Quadratic Programming problems necessitates considerations spanning both underlying optimization methods and implementation details. We present below an overview of some key attributes that contribute to the efficiency of practical Quadratic Programming.

Versatile optimization methods

Convex Quadratic Programs (QPs) demand specific algorithms that adhere to certain assumptions about the input QP from end users. To cater to a wide array of applications, an efficient algorithm should generate solutions under **minimal assumptions**. Additionally, "favorable assumptions" should yield a **versatile** algorithm, allowing for flexible use. Consider the following scenarios:

- Often, Quadratic Programs serve as subroutines in solving more complex problems, such as Sequential Quadratic Programming (SQP) [Nocedal and Wright, 2006, Chapter 16]. Frequently, these QPs are closely related. Consequently, leveraging prior solutions for *warm-starting* subsequent QP-solving processes emerges as a potent feature.
- In contexts like Robotic applications and inverse kinematic tasks, achieving millimeter-level solution precision is sufficient. Hence, overly accurate QP solutions may prove unnecessary. Incorporating *early-stopping* strategies offers practical utility.
- Thirdly, the employed approach should effectively exploit the inherent *structure* of the QP, such as sparsity. Indeed, some applications involve solving vast-scale problems, while others deal with more compact ones.

Furthermore, an "ideal" optimization method should guarantee relatively **fast** solution provision. Additionally, this method should offer mathematically **robust** formulations regarding QP entries, enabling efficient factorization procedures or algebraic manipulations avoiding division by potentially small values.

Moreover, optimization methods can possess desirable **features** that facilitate efficient problem-solving. For instance, inherent calculus may naturally prompt parallelization, accommodating large-scale applications. Finally, it is worth emphasizing that for widespread adoption, an optimization method's **simplicity** is key. This entails concise pseudo-code, interpretability, minimal hyper-parameters, and few heuristics.

Efficient and clean implementation

Efficient practical solvers for quadratic programming harness the power of **highly optimized linear algebra routines** with low-level programming languages such as C, C++, or Rust. They prioritize **memory allocation-free** designs to minimize memory movement during the solving process. Moreover, a "good implementation" exhibits **flexibility** through various aspects:

- **Template Programming**: Smart use of *template programming* automatically adapts to end-user floating-point precision, ensuring seamless integration and accuracy.
- **Interfacing Capabilities**: Effortless interfacing with different Operating Systems, higher-level programming languages (e.g., Python, Julia), or diverse numerical frameworks (e.g., CVXPY [Diamond and Boyd, 2016], CASADI [Andersson et al., 2019], TSID [Del Prete et al., 2016]) enhances usability and versatility.

- **Practical Features**: Offering practical functionalities, such as parallelization for learning tasks, broadens the scope of applications.

In addition, a well-designed QP solver boasts an **intuitive and well-documented API** that simplifies integration and use. Lastly, it is important to highlight that an open-source environment featuring a flexible license and an **active and responsive community** equips all practitioners with the essential tools to enhance the capabilities of a Quadratic Programming solver.

2.2 Summary of contributions

In this thesis, our focus centers on **solving** and **differentiating** efficiently convex QPs, with a particular emphasis on robotics and machine learning applications.

We start by introducing the ProxQP algorithm, a highly efficient QP solver adept at exploiting diverse QP structures through the utilization of primal-dual augmented Lagrangian techniques. We evaluate its practical performance on various standard robotic and control experiments, including a real-world closed-loop control application. While initially tailored for robotics, we demonstrate that ProxQP also rivals the state-of-the-art performance in generic QP problems, thereby rendering it a versatile off-the-shelf solver applicable beyond the realms of robotics. Additionally, we show that the ProxQP algorithm features a global convergence guarantee, as well as a few other advantageous numerical properties. Furthermore, we highlight that the ProxQP algorithm, and more generally primal-dual Proximal augmented Lagrangian methods, actually solves the *closest* primal-feasible QP—in a classical ℓ_2 sense that we detail in the sequel—if the original QP appears to be primally infeasible. Remarkably, this feature enhances stability and safety in high-speed control techniques.

We then harness this feature to enrich the expressive capabilities of existing quadratic programming layers. More precisely, we show how differentiation through the closest feasible QP solutions extends the scope of learnable QP layers, notably demonstrating superior predictive performance in conventional learning tasks. Additionally, we present alternative formulations that enhance numerical robustness, speed, and accuracy for training such layers.

On the practical side, we provide the ProxSuite library with two open-source C++ software packages as companions to these contributions:

- ProxQP: An efficient QP solver tailored for robotics and beyond.
- QPLayer: A QP layer endowed with enhanced learning capabilities.

2.2.1 Organisation of this thesis

This manuscript is organized as follows. In Chapter 3, after mathematically defining QP problems and the conditions for well-posedness, we give a concise introduction of the main optimization methods used for solving QPs. In Chapter 4 we first present the various components of ProxQP algorithm, including its proof of convergence and implementation details. We then present extensive benchmarks on robotic and standard QP problems. In Chapter 5 we provide a summary of the main algorithmic techniques used for designing efficient QPs layers. In Chapter 6, we present a unified approach to tackle the differentiability of both feasible and infeasible QPs by notably introducing the notion of Extended Conservative Jacobian. We propose efficient ways to compute it in both forward and backward automatic differentiation modes and finally demonstrate how this method enables training a broader range of QP layers. We notably illustrate on standard benchmarks that this technique performs better than traditional approaches for solving some tasks. We also show that when using standard learning approaches, our QP layer formulation performs faster than current state-of-the-art QP layers and is also numerically more

robust. Finally, Chapter 7 draws global perspectives and gives a personal view on future research directions.

2.2.2 Associated publications and software

This thesis has led to several publications, all of them dealing with numerical and differentiable optimization. We also mention other contributions not directly related to this thesis.

Conference articles

- **Antoine Bambade**, Sarah El-Kazdadi, Adrien Taylor, Justin Carpentier. ProxQP: Yet another Quadratic Programming Solver for Robotics and beyond. In *Robotics: Science and System (RSS)*, 2022;
- Wilson Jallet, **Antoine Bambade**, Nicolas Mansard, Justin Carpentier. Constrained differential dynamic programming: A primal-dual augmented lagrangian approach. In *EEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022;
- Louis Montaut, Quentin Le Lidec, **Antoine Bambade**, Vladimir Petrik, Josef Sivic, Justin Carpentier. Differentiable collision detection: a randomized smoothing approach. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Workshop articles

- Wilson Jallet, **Antoine Bambade**, Nicolas Mansard, Justin Carpentier. ProxNLP: a primal-dual augmented Lagrangian solver for nonlinear programming in Robotics and beyond. In *6th Legged Robots Workshop*, 2022.

Submitted articles

- **Antoine Bambade**, Fabian Schramm, Adrien Taylor, Justin Carpentier. Leveraging augmented-Lagrangian techniques for differentiating over infeasible quadratic programs in machine learning. Submitted to *International Conference on Learning Representations*, 2024.
- **Antoine Bambade**, Fabian Schramm, Sarah El-Kazdadi, Stéphane Caron, Adrien Taylor, Justin Carpentier. ProxQP: an Efficient and Versatile Quadratic Programming Solver for Real-Time Robotics Applications and Beyond. Submitted to IEEE Transactions on Robotics (TRO) (September 2023).
- Wilson Jallet, **Antoine Bambade**, Etienne Arlaud, Sarah El-Kazdadi, Nicolas Mansard, Justin Carpentier. ProxDDP: Proximal Constrained Trajectory Optimization. Submitted to IEEE Transactions on Robotics (TRO) (September 2023).
- Wilson Jallet, **Antoine Bambade**, Fabian Schramm, Quentin Le Lidec, Nicolas Mansard, Justin Carpentier. Notes on importance sampling of the first-order gradient estimator. Communication item submitted to IEEE Transactions on Robotics (TRO) (September 2023).

This thesis has also led to the development of ProxSuite[1], an open-source C++ library offering the ProxQP solver and QPLayer for efficiently solving and differentiating through Quadratic Programs. This novel library counts already about 300k downloads and is now present in many software frameworks (e.g., Cvxpy, Casadi, TSID).

[1] https://github.com/Simple-Robotics/proxsuite

Chapter 3

Background on Quadratic Programming

Abstract. *In this chapter we review the Quadratic Programming (QP) problem statement as well as conditions under which it is well-posed. We then give a concise introduction of the main algorithmic solution methods for solving QPs. More precisely, we review Active-Set methods, Primal-Dual Interior-Point methods, and Augmented Lagrangian (AL) methods. For the last family of methods, we distinguish those based on the Alternate Direction Method of Multipliers (ADMM) from those based on the Method of Multipliers (MM). Indeed, although ADMM is constructed from AL, it fundamentally manipulates a different splitting operation, thereby changing its convergence properties. For each of these methods, we propose a summary of a representative algorithm, its associated convergence guarantees, typical complexity if available, practical pros and cons, references to existing implementations of the approach, and some details about their practical implementations. This introduction is based on the reference books by [Boyd and Vandenberghe, 2004, Nocedal and Wright, 2006] and the following seminal works [Ferreau et al., 2014, Goldfarb and Idnani, 1983, Mehrotra, 1992, Stellato et al., 2020, O'Donoghue et al., 2016, Rockafellar, 1976a, Marchi., 2021]. The introduction also mentions other specific references.*

Chapter content

3.1 QPs: problem statement and optimality conditions 14
3.1.1 Problem statement 14
3.1.2 Duality 14
3.1.3 Certificates of optimality 15
3.1.4 Certificates of infeasibility 16
3.2 Active-set methods 16
3.2.1 The Simplex Method 17
3.2.2 Primal Active-set Methods 18
3.2.3 Dual Active-Set Methods 20
3.2.4 Parametric Active-Set Methods 25
3.3 Interior-Point method 28
3.4 ADMM based-methods 30
3.4.1 Primal version 31
3.4.2 Homogeneous self-dual embedding version 33
3.5 Proximal Augmented Lagrangian-based methods 35
3.5.1 Method of Multipliers 35
3.5.2 Proximal-Augmented Lagrangian Method 37
3.5.3 Primal-Dual Proximal-Augmented Lagrangian Method 39
3.5.4 Discussions 41

3.6 Summary 43

3.1 QPs: problem statement and optimality conditions

In this section we set the Quadratic Program statement and the conditions for being well-posed. We then review practical standard conditions used for evaluating the well-posedness of the problem or one of its solutions.

3.1.1 Problem statement

Formally, a Quadratic Program (QP) corresponds to the minimization of a convex quadratic cost under linear inequality constraints. It is mathematically described as follows:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \left\{ f(x) \stackrel{\text{def}}{=} \frac{1}{2} x^\top H x + g^\top x \right\} \\ \text{s.t.} \quad & Cx \leq u, \end{aligned} \tag{QP}$$

where $H \in \mathbb{R}^{n\times n}$ is a symmetric positive semi-definite matrix (notation $\mathbb{S}_+(\mathbb{R}^n)$), $g \in \mathbb{R}^n$, $C \in \mathbb{R}^{m\times n}$, and $u \in \mathbb{R}^m$. The problem dimension is n, and m corresponds to the number of inequality constraints. (QP) is said to be feasible if there exists at least one $x \in \mathbb{R}^n$ such that $Cx \leq u$.

3.1.2 Duality

Lagrangian of the problem: The Lagrangian function $\mathcal{L}$ associated to (QP) is defined by:

$$\mathcal{L}(x, z) \stackrel{\text{def}}{=} f(x) + z^\top (Cx - u), \tag{3.1}$$

where $x \in \mathbb{R}^n$ is the primal variable and $z \in \mathbb{R}^m$ corresponds to the dual variable.

Dual function: The dual function for (QP) is defined as follows [Boyd and Vandenberghe, 2004, Section 5.1.2]

$$\delta(z) \stackrel{\text{def}}{=} \inf_{x\in\mathbb{R}^n} \mathcal{L}(x, z). \tag{3.2}$$

When the Lagrangian is unbounded below in x, the dual function takes on the value $-\infty$. Since the dual function is the pointwise infimum of a family of affine functions of z, it implies that δ is concave.

The dual function yields lower bounds on the optimal value p^* of the problem (QP). For any $z \geq 0$, we have [Boyd and Vandenberghe, 2004, Section 5.1.3] that:

$$\delta(z) \leq p^* \tag{3.3}$$

This motivates us to consider the supremum of (3.3) to obtain the best possible lower bound of (QP).

Dual problem: The dual problem for (QP) is defined [Boyd and Vandenberghe, 2004, Section 5.2] as

$$\sup_{z\in\mathbb{R}^m_+} \delta(z), \tag{Dual-QP}$$

which leads from (3.3) to so-called weak-duality inequality

$$d^* \stackrel{\text{def}}{=} \sup_{z\in\mathbb{R}^m_+} \delta(z) \leq p^*. \tag{3.4}$$

(Dual-QP) is said to be **feasible** if $\delta < +\infty$.

Strong duality: For linearly constrained convex optimization problems such as (QP), strong duality holds, i.e., $p^* = d^*$, provided (QP) or (Dual-QP) is feasible [El Ghaoui, 2012, Theorem 2]. This property notably implies that the order of the minimization over x and the maximization over $z \geq 0$ of the Lagrangian function $\mathcal{L}$ can be switched without affecting the result.

3.1.3 Certificates of optimality

The KKT conditions: The Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for feasibility and optimality of a primal-dual point (x, z) with zero duality gap [Boyd and Vandenberghe, 2004, Section 5.5.3]. For (QP), the KKT system is given by the set of conditions:

$$\begin{cases} Hx + g + C^\top z = 0, \\ z \geq 0, \\ Cx \leq u, \\ z \odot (Cx - u) = 0, \end{cases} \tag{KKT}$$

where $\odot$ denotes the Hadamard product (i.e., for two vectors $u, v \in \mathbb{R}^m$, $u \odot v \in \mathbb{R}^m$ is the vector whose ith entry is $u_i v_i$). The latter condition is also known as the **complementary slackness**. It can be equivalently replaced by the zero duality gap condition which writes down for (QP) [O'Donoghue et al., 2016, Section 2.1]

$$x^\top H x + g^\top x + u^\top z = 0. \tag{3.5}$$

Active constraints: For a component index $i \in [1, m]$, the i^{th} constraint is said to be **active** if

$$C_i^\top x = u_i, \tag{3.6}$$

where C_i refers to the ith row of C. The complementary slackness condition [Boyd and Vandenberghe, 2004, Section 5.5.2] implies that:

$$z_i > 0 \implies C_i^\top x = u_i, \tag{3.7}$$

$$C_i^\top x < u_i \implies z_i = 0. \tag{3.8}$$

The **active-set** for KKT refers to the set I for which index elements point to the active constraints

$$I \stackrel{\text{def}}{=} \{i \in [1, m] | C_i^\top x = u_i\}. \tag{3.9}$$

Practical stopping criterion: In practice, we look for a tuple (x, z) satisfying the optimality conditions (KKT) up to a certain level of predefined accuracy $\epsilon_{\text{abs}} > 0$, (whose value typically depends on the application at hand), in the following sense [O'Donoghue et al., 2016, Section 3.5]:

$$\begin{cases} \|Hx + g + C^\top z\|_\infty \leq \epsilon_{\text{abs}}, \\ \|[Cx - u]_+\|_\infty \leq \epsilon_{\text{abs}}, \\ |x^\top H x + g^\top x + u^\top z| \leq \epsilon_{\text{abs}}. \end{cases} \tag{3.10}$$

In this work, we prefer the ℓ_∞ norm to the ℓ_2 norm as it is independent of the dimension d. It is also common to consider relative convergence criteria for early stopping, as absolute targets might not be reached due to the lack of numerical precision. In such cases, an additional relative tolerance level $\epsilon_{\text{rel}} > 0$ is introduced [Stellato et al., 2020, Section 5.1]

$$\begin{cases} \|Hx + g + C^\top z\|_\infty \leq \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max(\|Hx\|_\infty, \|g\|_\infty, \|C^\top z\|_\infty), \\ \|[Cx - u]_+\|_\infty \leq \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max(\|\Pi_{]-\infty, u]}(Cx)\|_\infty, \|Cx\|_\infty), \\ |x^\top H x + g^\top x + u^\top z| \leq \epsilon_{\text{abs}} + \epsilon_{\text{rel}}(|x^\top H x| + |g^\top x| + |u^\top z|), \end{cases} \tag{3.11}$$

where $z \mapsto \Pi_{]-\infty, u]}(z)$ refers to the projection of z onto the set $[-\infty, u]$.

3.1.4 Certificates of infeasibility

From the *theorem of strong alternatives* [Boyd and Vandenberghe, 2004, Section 5.8], [Banjac et al., 2019, Proposition 3.1] exactly one of the following set is nonempty if strong duality holds:

$$\begin{aligned} \mathcal{P} &\stackrel{\text{def}}{=} \{x \in \mathbb{R}^n | Cx - u \leq 0\}, \\ \mathcal{D} &\stackrel{\text{def}}{=} \{z \in \mathbb{R}^m | C^\top z = 0,\ z \geq 0,\ u^\top z < 0\}. \end{aligned} \tag{3.12}$$

Since $\mathcal{P}$ encodes primal feasibility, this implies that any dual variable $z \in \mathcal{D}$ serves as a *certificate* that the set $\mathcal{P}$ is empty, i.e., that the problem is **primal infeasible**.

Furthermore, it has been shown in [Banjac et al., 2019, Proposition 3.1], that a vector $x \in \mathbb{R}^n$ satisfying the following condition is a certificate of dual infeasibility

$$\begin{aligned} &Hx = 0, \\ &g^\top x < 0, \\ &Cx \in \mathcal{C}^\infty, \end{aligned} \tag{3.13}$$

where $\mathcal{C}^\infty \stackrel{\text{def}}{=} \{z \in \mathbb{R}^m | Cx - u + \tau z \geq 0, x \in \mathbb{R}^n, \tau \geq 0\}$ is the *recession cone* of the constraint set. The last condition splits as follows for $i \in [1, m]$ when dealing with linear constraints:

$$\begin{aligned} &(Cx)_i = 0 \text{ if } u_i \in \mathbb{R}, \\ &(Cx)_i \geq 0 \text{ if } u_i = +\infty. \end{aligned} \tag{3.14}$$

Practical conditions for infeasibility: In practice, (QP) is considered being primal infeasible at tolerance level $\epsilon_{\text{pinf}} > 0$ if for a nonzero vector $\mathrm{d}z \geq 0$ the following two conditions hold [Hermans et al., 2021, Section 5.4.2]

$$\begin{aligned} &\|C^\top \mathrm{d}z\|_\infty \leq \epsilon_{\text{pinf}} \|\mathrm{d}z\|_\infty, \\ &u^\top \mathrm{d}z \leq -\epsilon_{\text{pinf}} \|\mathrm{d}z\|_\infty. \end{aligned} \tag{3.15}$$

Moreover, the problem is determined to be dual infeasible at tolerance level $\epsilon_{\text{dinf}} > 0$ if for some vector $\mathrm{d}x \neq 0$ [Hermans et al., 2021, Section 5.4.2] the following holds for all $i \in [1, m]$

$$\begin{aligned} &(C\mathrm{d}x)_i \leq \epsilon_{\text{dinf}} \|\mathrm{d}x\|_\infty \text{ if } u_i \in \mathbb{R}, \\ &(C\mathrm{d}x)_i \geq -\epsilon_{\text{dinf}} \|\mathrm{d}x\|_\infty \text{ if } u_i \geq 0, \end{aligned} \tag{3.16}$$

and

$$\begin{aligned} &\|H\mathrm{d}x\|_\infty \leq \epsilon_{\text{dinf}} \|\mathrm{d}x\|_\infty, \\ &g^\top \mathrm{d}x \leq -\epsilon_{\text{dinf}} \|\mathrm{d}x\|_\infty. \end{aligned} \tag{3.17}$$

3.2 Active-set methods

The basic idea behind active-set methods is to iteratively update an active set of constraints that are likely to be active at the optimal solution. At each iteration, the algorithm solves a reduced QP problem that only considers the active constraints as equality constraints. If the current active set is optimal, the algorithm terminates. Otherwise, it updates the active set by adding or removing constraints based on certain optimality conditions.

Active-set methods are particularly efficient when the number of constraints is small, and the problem is not degenerate (e.g., constraint matrices are full rank, or f is strictly convex). We start reviewing these methods with the Simplex Method since it is a well-known and efficient active-set algorithm for solving linear programs (LP). We then review primal and dual active-set methods (which are direct extensions of the Simplex Method applied to the primal and dual of (QP)). Finally, we detail Parametric Active-Set Methods, which are particularly efficient for solving Sequential Quadratic Programs (SQP).

3.2.1 The Simplex Method

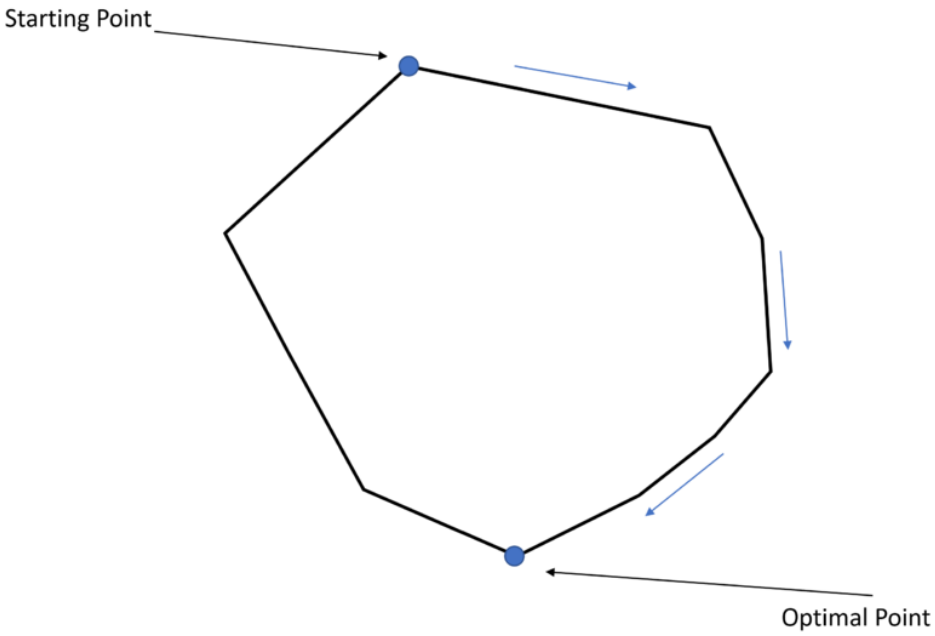

Figure 3.1: Geometric illustration of solving an LP using the Simplex method.

Algorithm summary: The Simplex Method [Dantzig, 1990] (and its revised version see Algorithm 1) solves Linear Programs (LP) of the following standard form, assuming $m \leq n$,

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} g^\top x, \\ &\text{s.t., } Cx = u, \\ &x \geq 0. \end{aligned} \tag{3.18}$$

It uses the property that for a well-posed and not degenerate LP, there always exists a vertex, which is a solution to its constrained polyhedron. More precisely, The Simplex Method starts from a vertex $\hat{x}$ of the feasible set and then calculates at each iteration a new vertex $\hat{x}^+$ unless it is observed that $\hat{x}^+$ is a solution or that the problem is unbounded (see Figure 3.1).

Convergence properties: The standard and revised Simplex Method assumes that (see e.g., [Nocedal and Wright, 2006, Section 13])

$$\mathcal{A}_P \stackrel{\text{def}}{=} \{x \in \mathbb{R}^n | Cx = u,\ x \geq 0\}, \tag{3.19}$$

is a non-empty polyhedron and that $C \in \mathbb{R}^{m \times n}$ has full row rank. Then, provided that the linear program is not degenerate (we review this notion hereafter) and bounded, the Simplex Method terminates at an optimal point [Nocedal and Wright, 2006, Theorem 13.4]. Regarding its convergence speed, it has been proven that its worst-case complexity is exponential [Klee and Minty, 1972].

Pros and cons: Despite its exponential worst-case complexity, the Simplex Method is remarkably efficient in practice, which explains why it is widely used. Historically, it constituted a great improvement over earlier methods (such as Fourier–Motzkin elimination; see for more details [Nocedal and Wright, 2006, Section 13.4]). It has notably been shown that when the inputs to the algorithm are slightly randomly perturbed, the expected running time of the method

is polynomial for any inputs [Dadush and Huiberts, 2018, Spielman and Teng, 2004]. This means that for any problem, there is a "nearby" one that the Simplex Method will efficiently solve, covering most real-world linear programs.

Yet, it may require some arduous preprocessing phase to match the convergence assumptions (e.g., to try removing redundancies in the constraints of C). Furthermore, C being full rank still does not prevent cycling effects due to the degeneracy of some solutions [Nocedal and Wright, 2006, Section 13.5].

Detailed implementation: We define the matrix $C_{\bullet B} = [C_i]_{i \in B}$, where C_i is the ith column of C.

Definition 1. *A vector $x \in \mathbb{R}^n$ is a **basic feasible point** if it is feasible and if there exists a subset B of m indexes in $[1, n]$ such that $C_{\bullet B}$ is nonsingular and such that $i \notin B \implies x_i = 0$. Such a set is called a **basis** for the problem. It is said to be **degenerate** if $x_i = 0$ for some $i \in B$. The linear program (3.18) is said to be **degerenate** if it has at least a degenerate basis. The complementary set of B within $[1, n]$ is noted N.*

Algorithm 1: The revised Simplex Method

Inputs: a basis B, its complementary set $N, x_B = C_{\bullet B}^{-1} u \geq 0, x_N = 0$ provided by (3.20),
Initialization:
Solve $C_{\bullet B}^\top \lambda = g_B$ for λ,
Compute *the reduced cost*: $r = g_N - C_N^\top \lambda$,
while $r < 0$ **do**
  Select $q \in N$ with $r_q < 0$ as the entering index;
  Solve $C_{\bullet B} d = -C_{\bullet j}$ for d;
  If $d \geq 0$ stop since the problem is unbounded;
  Compute $k = \arg\min\{-\frac{x_i}{d_i} | i \in B | d_i < 0\}$ and $\hat{\alpha} = -\frac{x_k}{d_k}$;
  $x_B \leftarrow x_B + \hat{\alpha} d,\ x_N \leftarrow (0, ..., 0, -\hat{\alpha}, 0, ..., 0)$;
  $B \leftarrow (B \cup \{j\}) \setminus \{k\}, N \leftarrow (B \cup \{k\}) \setminus \{j\}$;
  $r \leftarrow g_N - C_N^\top C_{\bullet B}^{-\top} g_B$;
end

To determine a starting basic feasible point, it is possible to solve what is called the following "Phase I" program

$$\begin{aligned} &\min_{x \in \mathbb{R}^n, z \in \mathbb{R}^m} \sum_{i=1}^{m} z_i, \\ &Ax + Dz = u, \\ &x \geq 0,\ z \geq 0, \end{aligned} \tag{3.20}$$

where D is the diagonal matrix for which $D_{ii} = 1$ if $u_i \geq 0$ and $D_{ii} = -1$ otherwise. To solve (3.20), it is possible to use the Simplex Method starting from $(0, Du)$ since it is by construction a basic feasible point [Nocedal and Wright, 2006, Section 13.5]. Alternative solutions exist, such as the "Big M" approach, which will be reviewed for the Primal Active-Set method.

3.2.2 Primal Active-set Methods

Algorithm summary: Primal Active-Set Methods (see Algorithm 2) are an extension of the Simplex Method for QPs. They differ fundamentally in the sense that optimal solutions for (QP) do not necessarily lie anymore at the vertices of the constraint polyhedron. Yet, similarly to the Simplex Method, Primal Active-Set methods first find a feasible point using a Phase I

approach. They then fix a *working set*, a maximal linearly independent subset of the active constraints, and solve the subproblem for which the inequality constraints of the working set are imposed as equalities. This resulting equality-constrained QP can be efficiently solved through a linear system. This procedure generates primal feasible iterates and is repeated until dual feasibility and, hence, an optimal solution is obtained.

Convergence properties: Standard assumptions require the constraint matrix C to be of full row rank. Furthermore, it is shown that under strict convexity of f, primal active set methods converge globally [Nocedal and Wright, 2006, Section 16.5].

Pros and Cons: Similarly to the Simplex Method, Primal Active-Set Methods can work very well in practice, especially on small-sized QPs. On the positive side, hot-start strategies (reuse of the previous factorization) can easily be incorporated in this setting. This is usually key for solving cascades of similar QPs, which is common in applications such as sequential quadratic programming (SQP) and model predictive control (MPC). The QPA module in the open-source software GALAHAD [Gould et al., 2017] provides a primal active-set-based implementation (handling also ℓ_1 penalties in the cost function).

Yet, the initial Phase I can be costly and reduce the possibility of warm-starting. Furthermore, Primal Active-Set Methods have limited scalability, struggle to exploit sparsity, and are hard to *early-stop* at inexact solutions satisfying (3.10) at accuracy ϵ_{abs} (they tend to stop only when a global solution is found, i.e., with $\epsilon_{\text{abs}} = 0$). Moreover, they may require strong assumptions, such as the QP being strictly convex or the constraint matrix being of full rank. These assumptions can reduce their applicability and may cause robustness issues, such as "active-set cycling" [Nocedal and Wright, 2006, sections 13.5 and 16.5]

Detailed implementation: Formally, considering a feasible iterate x^k and a working set $\mathcal{W}_k$, solving

$$\begin{aligned} &\min_{p\in\mathbb{R}^n} \tfrac{1}{2} p^\top H p + g_k^\top p \\ &\text{s.t., } C_i^\top p = 0,\ i \in \mathcal{W}_k, \end{aligned} \tag{3.21}$$

with $g_k \overset{\text{def}}{=} Hx^k + g$ corresponds to minimizing $f(x^k + p)$ along the current working set. Solving (3.21) amounts to finding a solution to the linear system

$$\begin{bmatrix} H & C_{\mathcal{W}_k}^\top \\ C_{\mathcal{W}_k} & 0 \end{bmatrix} \begin{bmatrix} p \\ \lambda \end{bmatrix} = \begin{bmatrix} -g_k \\ 0 \end{bmatrix}, \tag{3.22}$$

for some multiplier vector λ. Under the regularity conditions above, it can be handled efficiently using one of the methods detailed in [Nocedal and Wright, 2006, Sections 16.2 and 16.3][1]. Noting p^k a solution for (3.21), by construction this **direction step** guarantees that

$$C_i^\top (x^k + \alpha p^k) = u_i,\ \forall i \in \mathcal{W}_k, \forall \alpha \in \mathbb{R}. \tag{3.23}$$

Supposing for the moment that the optimal p^k from (3.21) is nonzero, the update

$$x^{k+1} = x^k + \alpha^k p^k, \tag{3.24}$$

with

$$\alpha^k = \min(1, \min_{i \notin \mathcal{W}^k, C_i^\top p^k > 0} \frac{u_i - C_i^\top x^k}{C_i^\top p^k}), \tag{3.25}$$

[1] Namely, null-space approach, Schur complement method, direct factorization of the system, Conjugate Gradient or Projected Conjugate Gradient methods.

guarantees **an optimal step-length direction**. Indeed, for $i \in \mathcal{W}_k$ the constraint is satisfied regardless of α^k. Otherwise, for $i \notin \mathcal{W}_k$, if $C_i^\top p^k \leq 0$, then for all $\alpha^k \geq 0$, $C_i^\top (x^k + \alpha^k p^k) \leq C_i^\top x^k \leq u_i$. Hence, the constraint i will be satisfied for all nonnegative choices of the step-length parameter. For $i \notin \mathcal{W}_k$ and $C_i^\top p^k > 0$, the inequality $C_i^\top (x^k + \alpha^k p^k) \leq u_i$ happens only if

$$\alpha^k \leq \frac{u_i - C_i^\top x^k}{C_i^\top p^k}. \tag{3.26}$$

If $\alpha^k < 1$, that is the step along p^k is blocked by some constraint not in $\mathcal{W}_k$, a new working set $\mathcal{W}_{k+1}$ is then constructed by adding one of the blocking constraints to $\mathcal{W}_k$.

We continue to iterate in this manner, adding constraints to the working set until we reach a point $\hat{x}$ that minimizes the quadratic objective function over its current working set $\hat{\mathcal{W}}$. It is easy to recognize such a point because the subproblem (3.21) has solution $p = 0$. When it eventually happens, the dual multipliers $\hat{z}$ can be derived by solving

$$\sum_{i \in \hat{\mathcal{W}}} \hat{z}_i C_i = H\hat{x} + g, \tag{3.27}$$

and by setting $\hat{z}_i = 0$, $\forall i \notin \hat{\mathcal{W}}$. The multipliers sign must then be checked to satisfy the full complementarity conditions:

- If $\hat{z}_i \geq 0, \forall i \in \hat{\mathcal{W}} \cap I$, then $\hat{x}$ is global solution for (QP),
- Otherwise, one or more of the multipliers $\hat{z}_i$, $i \in \hat{\mathcal{W}} \cap I$ is negative, and the objective function f may be decreased by dropping one of these constraints from the working set and solving the new resulting sub-problem ([Nocedal and Wright, 2006, Section 12.3] propose several choices for removing such constraint). This strategy produces a direction p at the next iteration that is feasible with respect to the dropped constraint.

Various techniques can be used to determine an **initial feasible point**. A "Phase I" approach can be used following what was described for the Simplex Method (3.18). An alternative that would also work for the Simplex Method is a penalty method (also referred to as "big M"), which includes a measure of infeasibility in the objective that is guaranteed to be zero at the solution. That is, we introduce a scalar artificial variable η into (QP) to measure the constraint violation, and solve the problem

$$\begin{aligned} &\min_{x,\eta} f(x) + M\eta, \\ &\text{s.t., } Cx - u \leq \eta,\ 0 \leq \eta \end{aligned} \tag{3.28}$$

for some large positive value of M. It can be shown that whenever there exist feasible points for the original problem (QP), then for all M sufficiently large, the solution of (3.28) will have $\eta = 0$, with an x component that is a solution for (QP).

3.2.3 Dual Active-Set Methods

Algorithm summary: Dual Active-Set Methods (see Algorithm 3) are an extension of the Dual Simplex Method for QPs [Nocedal and Wright, 2006, Section 13.6]. Similarly to the Dual Simplex Method, Dual Active-Set methods fix a *working set*, a maximal linearly independent subset of the active constraints, and solve the subproblem for which the inequality constraints of the working set are imposed as equalities. This resulting equality-constrained QP can be efficiently solved through a linear system. This procedure generates dual feasible iterates and is repeated until primal feasibility, and hence, an optimal solution is obtained.

Convergence properties: Standard assumptions require the constraint matrix C to be of full row rank and strict convexity of f. It ensures global convergence of the method [Goldfarb and Idnani, 1983, Theorem 3].

Algorithm 2: Primal Active-Set Methods

Inputs:
Compute $\hat{x}$ a feasible starting point solving the broader problem (3.28),
Set $\mathcal{W}_0$ to be a subset of the active constraints at x_0.
for $k = 0, 1, ...$ **do**
  Solve (3.21) to find p^k;
  if $p^k = 0$ **then**
    Find $\hat{z}$ solving (3.27) with $\hat{\mathcal{W}} = \mathcal{W}_k$;
    if $\hat{z}_i \geq 0 \; \forall i \in \mathcal{W}_k \cap I$ **then**
      stop with solution $(x^k, \hat{z})$
    else
      $j \leftarrow \arg\min_{j \in \mathcal{W}_k \cap I} \hat{z}_j$;
      $x^{k+1} \leftarrow x^k$, $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \setminus \{j\}$;
    end
  else
    Compute α^k from (3.25);
    $x^{k+1} \leftarrow x^k + \alpha^k p^k$;
    if *blocking constraints* **then**
      Add one blocking constraints to $\mathcal{W}_k$ to form $\mathcal{W}_{k+1}$;
    else
      $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k$;
    end
  end
end

Pros and Cons: Similarly to the Dual Simplex Method, Dual Active-Set Methods can work very well in practice, especially on small-sized QPs. On the positive side, hot-start (typically reusing the previous factorization) strategies can easily be incorporated in this setting. This is usually key for solving cascades of similar QPs, which is common in applications such as sequential quadratic programming (SQP) and model predictive control (MPC). Finally, contrary to Primal Active-Set Methods, no "Phase I" is required anymore. Popular Dual Active-Set-based solvers include the open-source QUADPROG [Goldfarb and Idnani, 1983], QPNNLS [Bemporad, 2015] and DAQP [Arnström et al., 2022].

Yet, Dual Active-Set Methods have limited scalability, struggle to exploit sparsity, and are hard to *early-stop* at inexact solutions satisfying (3.10) at accuracy ϵ_{abs} (they tend to stop only when a global solution is found, i.e., with $\epsilon_{\text{abs}} = 0$). Moreover, they may require strong assumptions, such as the QP being strictly convex or the constraint matrix being of full rank. These assumptions can reduce their applicability and may cause robustness issues, such as "active-set cycling" [Nocedal and Wright, 2006, sections 13.5 and 16.5].

Detailed implementation of quadprog [Goldfarb and Idnani, 1983]: In this section, we focus more specifically about the implementation of QUADPROG algorithm.

Definition 2. *Considering a subset J of $[1, m]$, the **Moore-Penrose generalized inverse** of C_J in the space of variables obtained under the transformation $\hat{x} = H^{1/2}x$ is noted N^J and defined as [Goldfarb and Idnani, 1983, Equation 2.1]*

$$N^J = (C_J H^{-1} C_J^\top)^{-1} C_J H^{-1}. \tag{3.29}$$

*Furthermore, the **reduced inverse Hessian** operator for f subject to the active set of constraints*

C_J is noted G and is defined as [Goldfarb and Idnani, 1983, Equation 2.2]

$$G = H^{-1}(I - C_J N^J) = H^{-1} - H^{-1}C_H(C_J H^{-1} N^J)^{-1} C_J H^{-1}. \tag{3.30}$$

Dual Active-Set Methods generate dual feasible points at each iteration. For this reason, they do not initially need a Phase I approach to generate primal feasible points. Furthermore, it can be seen that starting from an empty feasible set $\mathcal{W}^0 = \emptyset$, it always holds that $x^0 = -H^{-1}g$ (with $z^0 = \emptyset$) satisfies the dual feasibility requirement with respect to the equality constrained QP formed without any constraints. So it is always possible to efficiently initialize Dual Active-Set Methods provided $H \in \mathcal{S}_{++}(\mathbb{R}^n)$.

Assume that we have a working set $\mathcal{W}^k \neq \emptyset$ and iterates $(x^k, z^k_{\mathcal{W}^k} \geq 0)$ which solves the corresponding equality constrained QP:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &\text{s.t., } C_i^\top x \leq u_i,\ i \in \mathcal{W}_k. \end{aligned} \tag{3.31}$$

If the set $V = \{i \in [1, m] \setminus \mathcal{W}_k | C_i^\top x^k - u_i > 0\}$ is empty, then (x^k, z^k) is a solution of (QP) by extending z^k to be zero on the complementary set of $\mathcal{W}^k$. Otherwise, let's pick a violating constraint index $i \in V$.

The fundamental idea built on Dual Active-Set Methods relies on the fact that, starting from x^k, the primal step direction $d = GC_i^\top$ ensures that $\forall t \in \mathbb{R}^n, \hat{x} = x^k + td$ satisfies

$$G(H\hat{x} + g) = 0, \tag{3.32}$$

$$C_j^\top \hat{x} - u_j = 0,\ \forall j \in \mathcal{W}^k, \tag{3.33}$$

$$\hat{z} = N^{\mathcal{W}^k \cup \{i\}}(H\hat{x} + g) = N^{\mathcal{W}^k \cup \{i\}} x^k + t \begin{bmatrix} r \\ 1 \end{bmatrix}, \tag{3.34}$$

$$C_i^\top \hat{x} - u_i = C_i^\top x^k - u_i + td^\top (C_i^\top). \tag{3.35}$$

with $r = N^{\mathcal{W}^k} C_i^\top$ [Goldfarb and Idnani, 1983, Lemma 1]. It follows immediately that provided $d \neq 0$ the point $\hat{x} = x + t_p d$ minimizes the quadratic f over $\{x \in \mathbb{R}^n | C_j^\top x = u_j, j \in \mathcal{W}^k \cup \{i\}\}$, where

$$t_p = -\frac{C_i^\top x^k - u_i}{d^\top C_i^\top}. \tag{3.36}$$

If $\hat{z} \geq 0$, then $(\hat{x}, \hat{z})$ is a solution to the QP subproblem formed with the inequality constraints indexed by $\mathcal{W}^k \cup \{i\}$. Otherwise, there exists $t_d < t_p$, where

$$t_d = \min \left\{ \min_{j \text{ s.t., } r_j > 0} \frac{N^{\mathcal{W}^k \cup \{i\}}(x^k)_j}{N^{\mathcal{W}^k}(x^k)_j}, \infty \right\}, \tag{3.37}$$

such that there always exists some components of $N^{\mathcal{W}^k \cup \{i\}}(x^k + td) < 0$ for $t \in (t_d, t_p)$ [Goldfarb and Idnani, 1983, Theorem 1]. If we drop one such component i_d from the working set $\mathcal{W}^k$, then it holds that $\hat{x} = x^k + t_d d$ satisfies the following with the set $\mathcal{W}^k$ [Goldfarb and Idnani, 1983, Theorem 1]

$$C_j^\top \hat{x} - u_{i_d} > 0, \tag{3.38}$$

$$C_l^\top \hat{x} = u_l,\ \forall l \in \mathcal{W}^k \setminus \{i_d\}, \tag{3.39}$$

$$G(H\hat{x} + g) = 0 \tag{3.40}$$

$$N^{\mathcal{W}^k} \hat{x} \geq 0. \tag{3.41}$$

$\hat{x}$ thus corresponds to the optimal solution of the QP subproblem indexed by $\mathcal{W}^k$ and for which the i_dth constraint is replaced by $C_{i_d}^\top x \leq C_{i_d}^\top \hat{x}$. We are then back to the situation of (3.31).

The following procedure shows that starting from x^k, $\mathcal{W}^k$ and a violating constraint index i, we either obtain an optimal solution $\hat{x}$ for $\mathcal{W}^k \cup \{i\}$, or a new point $\hat{x}$ solving subproblem indexed by $\mathcal{W}^k \setminus \{i_d\}$ with a dropped constraint i_d. Notably, under the regularity assumptions provided ($H \in \mathbb{S}^+_{++}$, full rank of C), it is guaranteed to find in a finite number of steps a global solution or problem infeasibility (i.e., $t_p = t_d = \infty$) [Goldfarb and Idnani, 1983, Theorem 3]. The detailed algorithm is provided in Algorithm 3.

Finally, let us mention that rather than computing G and N^k, current implementation store and update the matrices $J_1 = Q_1^\top L^{-1}$, $J_2 = Q_2^\top L^{-1}$ and R obtained from the Cholesky and QR factorizations of $H = LL^\top$ and $L^{-1}C_{\mathcal{W}_k} = [Q_1\ Q_2]\begin{bmatrix} R \\ 0 \end{bmatrix}$. Indeed, such decompositions provide the required operators through

$$\begin{aligned} G &= J_2 J_2^\top, \\ N^k &= R^{-1} J_1^\top. \end{aligned} \tag{3.42}$$

G and N^k can then be efficiently updated using QR decomposition updates (or downdates) following, for example [Goldfarb and Idnani, 1983, Section 5].

Algorithm 3: Dual Active-Set Methods

Initialization:

- initial S-pair: $x^0 = -H^{-1}g$, $\mathcal{W}_0 = \emptyset$, $z^0 = 0$
- temporary variables: $q = 0$ (cardinality of $\mathcal{W}_0$), $G = H^{-1}$, $v = \frac{1}{2}g^\top x^0$

for $k = 0, 1, ...$ **do**
  Derive $V = \{i \in [1, m] \setminus \mathcal{W}_k | C_i^\top x^k - u_i > 0\}$;
  if $V \neq \emptyset$ **then**
    Choose a violated constraint $i \in V$;
    If $q = 0$, $\hat{z} = 0$, otherwise $\hat{z} = \begin{bmatrix} z^k \\ 0 \end{bmatrix}$;
    $\hat{\mathcal{W}} = \mathcal{W}_k \cup \{i\}$;
    Derive $d = GC_i^\top$ (step direction in primal space);
    If $q > 0$, derive $r = N^k C_i^\top$ (negative of the step direction in dual space);
    Compute t_p (minimum step in primal space) following (3.36) (taking ∞ is $d = 0$);
    Compute t_d (maximum step length in dual space) and an associated dropping constraint index $i_d \in [1, m]$ following (3.37) (taking ∞ if the set involved is empty);
    Compute step length $t = \min(t_p, t_d)$;
    if $t = \infty$ *(no step in primal or dual space)* **then**
      stop, (QP) is infeasible.
    else
      if $t_p = \infty$ *(step in dual space)* **then**
        $\hat{z} \leftarrow \hat{z} + t \begin{bmatrix} -r \\ 1 \end{bmatrix}$;
        Drop constraint i_d: $\hat{\mathcal{W}} \leftarrow \hat{\mathcal{W}} \setminus \{i_d\}$, $q \leftarrow q - 1$;
        Update G and N^k (see (3.42)) ;
        Compute a new step direction d;
      else
        $x^{k+1} \leftarrow \hat{x} + td$ (primal step);
        $v \leftarrow v + td^\top C_i(\frac{1}{2}t + \hat{z}_{q+1})$;
        $\hat{z} \leftarrow \hat{z} + t \begin{bmatrix} -r \\ 1 \end{bmatrix}$ (dual step);
        if *If* $t = t_p$ *(full primal step)* **then**
          $z^{k+1} \leftarrow \hat{z}$;
          Add constraint i: $\mathcal{W}_{k+1} \leftarrow \hat{\mathcal{W}} \cup \{i\}$, $q \leftarrow q + 1$;
          New step: $k \leftarrow k + 1$ and update V;
        else
          Drop constraint i_d: $\mathcal{W}_{k+1} \leftarrow \hat{\mathcal{W}} \setminus \{i_d\}$, $q \leftarrow q - 1$;
          Update G and N^k (see (3.42)) ;
          Compute a new step direction;
        end
      end
    end
  else
    stop, x^k and z^k forms a global optimal solution.
  end
end

3.2.4 Parametric Active-Set Methods

Algorithm summary: A variant of the Primal or Dual Active-Set Methods are Parametric Active-Set Methods. They are centered around the idea of tracing the solution of a linear homotopy parameterized by $\tau \in [0,1]$ between a QP problem with known solution ($\tau = 0$) and the QP problem to be solved ($\tau = 1$)

$$\begin{aligned} &\min_{x(\tau)\in\mathbb{R}^n} \tfrac{1}{2}x(\tau)^\top H x(\tau) + g(\tau)^\top x(\tau), \\ &\text{s.t., } Cx(\tau) \leq u(\tau), \end{aligned} \tag{QP(τ))}$$

where $g(\tau)$ and $u(\tau)$ are affine-linear functions of the homotopy parameter τ:

$$g(\tau) = (1-\tau)g^0 + \tau g, \tag{3.43}$$
$$u(\tau) = (1-\tau)u^0 + \tau u, \tag{3.44}$$

where $g^0 = 0$ or $u^0 = 0$ by default. It can be shown that the optimal solutions $x(\tau)$ depend piecewise affine on τ. From solution $x(\tau)$ and a working set $\mathcal{W}(\tau)$, the method consists of finding appropriate step direction and step length (i.e., $\mathrm{d}\tau$, $\mathrm{d}x$) to iteratively reach a solution of the QP to be solved ($\tau = 1$). Hence, similarly to previous active-set methods, Parametric Active-Set Methods solve as subproblems equality constrained QPs (the subproblem considering the active constraints of the current homotopy as equality constraints). Yet, compared to primal or dual active set methods, each new iteration of parametric methods is neither primal nor dual feasible by construction, except at convergence.

Convergence properties: Under the assumption that the matrix C is of full row rank, H positive definite on the null-space of C, and the first multiplier solution $z(0) > 0$, then Parametric Active-Set Methods are ensured to find either a global solution or infeasible certificates in a finite number of steps [Best, 1996, Assumption 2.1 and Theorem 4.1].

Pros and Cons: A considerable advantage of Parametric Active-Set Methods is that they are by design meant to use hot-starting and warm-starting strategies, which can be less straightforward for Primal or Dual Active-Set Methods. Furthermore, they do not necessarily require the full strict convexity of f, which makes them applicable to a wider range of QP problems than other active-set methods. Such features make them particularly suited for solving sequential quadratic programming (SQP) and model predictive control (MPC) tasks. A famous Parametric Active-Set-based solver is the open-source qpOASES [Ferreau et al., 2014].

Yet, similarly to other active-set methods, Parametric Active-Set Methods have limited scalability, struggle to exploit sparsity and are not robust to early termination. Moreover, the assumption of C being of full rank may still cause robustness issues such as "active-set cycling" [Nocedal and Wright, 2006, sections 13.5 and 16.5].

Detailed implementation of qpOASES [Ferreau et al., 2014]: Starting from a solution point $x(\tau), z(\tau)$ of (QP(τ)) with current workspace $\mathcal{W}(\tau)$ it is possible to determine a step direction $(\mathrm{d}x, \mathrm{d}z)$ towards (QP(τ)) solution with $\tau = 1$ by solving [Ferreau, 2006, Ferreau et al., 2008]

$$\begin{bmatrix} H & C_{\mathcal{W}(\tau)}^\top \\ C_{\mathcal{W}(\tau)} & 0 \end{bmatrix} \begin{bmatrix} \mathrm{d}x \\ -\mathrm{d}z_{\mathcal{W}(\tau)} \end{bmatrix} = \begin{bmatrix} -(g(1) - g(\tau)) \\ u_{\mathcal{W}(1)} - u_{\mathcal{W}(\tau)} \end{bmatrix}, \tag{3.45}$$

and by letting $\mathrm{d}z_{\mathcal{W}^c(\tau)} = 0$ with $\mathcal{W}^c(\tau)$ the complementary set of $\mathcal{W}(\tau)$. For solving (3.45) saddle-point problem, an efficient option consists of using, for example, a null-space method

(through a Cholesky factorization of the null-sparse projection of the Hessian, see [Ferreau et al., 2014, Section 2.3.1] for example for practical details).

It is then possible to follow $x(\tau), z(\tau)$ in the direction $(\mathrm{d}x, \mathrm{d}z)$ along the current active set until either an inactive constraint becomes active (primal blocking), or a dual variable of an active constraint changes its sign (dual blocking). The step length $\mathrm{d}\tau$ onto the first blocking constraint can be determined using the ratio test function RT for $u, v \in (\mathbb{R}^m)^2$ [Ferreau et al., 2014, Equations 11 and 12]

$$\mathrm{RT}(u, v) \mapsto \min\{u_i/v_i | 1 \leq i \leq m, v_i > 0\}. \tag{3.46}$$

The first blocking constraint is then given by

$$t_p = \mathrm{RT}(u(\tau) - Cx(\tau), C\mathrm{d}x), \tag{3.47}$$

and the first blocking sign is changed by

$$t_d = \mathrm{RT}(z(\tau)_{\mathcal{W}(\tau)}, \mathrm{d}z(\tau)_{\mathcal{W}(\tau)}). \tag{3.48}$$

The step length is then defined as $\mathrm{d}\tau = \min(t_p, t_d)$. The optimal solution is found if $\mathrm{d}\tau = 1$ since there are no blocking constraints. Otherwise, if $\mathrm{d}\tau = t_d$, there is dual blocking constraint for some constraint index $j \in \mathcal{W}(\tau)$ realizing the ratio test (3.48). We consider removing it from the new working set in such a case. If $\mathrm{d}\tau = t_p$ similarly to the primal active set method, we add a blocking constraint index i, realizing the ratio test (3.47).

For either case, improvements exist for efficiently testing whether the new working sets formed are linearly dependent and trying to find adequate exchange variables alternatively. We detail them below.

If the new working set $\mathcal{W}(\tau)^+$ is formed by the addition of a new constraint i, it can lead to a rank deficiency of the matrix $C_{\mathcal{W}(\tau)^+}$ and thus loss of invertibility for (3.45). Linear dependence of C_i can then be tested by solving

$$\begin{bmatrix} H & C_{\mathcal{W}(\tau)}^\top \\ C_{\mathcal{W}(\tau)} & 0 \end{bmatrix} \begin{bmatrix} p \\ q_{\mathcal{W}(\tau)} \end{bmatrix} = \begin{bmatrix} C_i^\top \\ 0 \end{bmatrix}, \tag{3.49}$$

and C_i is linearly dependent on $C_{\mathcal{W}(\tau)}$ if and only if $p = 0$. If it happens, one strategy consists in obtaining q from $q_{\mathcal{W}(\tau)}$ by letting $q_l = 0$ if $l \in \mathcal{W}(\tau^+)^c$. Then we try updating the dual multiplier considering the ratio test

$$\lambda^\star = \mathrm{RT}(-\mathcal{W}_i^+(\mathcal{W} \odot y(\tau^+)), \mathcal{W}_i^+({}_{\mathcal{W}}\odot q)), \tag{3.50}$$

where $\mathcal{W}_i^+(\mathcal{W} \odot v)$ is derived component-wise with the vector $\mathcal{W} \odot v$ which equals v_k if $k \in \mathcal{W}$ and 0 otherwise. If $\lambda^* = +\infty$, the parametric QP is infeasible. Otherwise, we can get a minimizing exchange index for (3.50) noted i_e and then update the constraint with

$$z(\tau^+)_i = -\lambda^\star \tag{3.51}$$

$$z(\tau^+)_k = z(\tau^+)_k + \lambda^\star q_k \text{ for } k \text{ s.t., } \mathcal{W}_k \neq 0. \tag{3.52}$$

By construction $z(\tau^+)_{i_e} = 0$ and constraint i_e can be removed from $\mathcal{W}(\tau^+)$ to restore linear independence.

Finally, if the new working set $\mathcal{W}(\tau)^+$ is formed by removing a constraint, it can lead to the exposition of a direction of zero curvature in the null-space of $C_{\mathcal{W}(\tau^+)}$. It can again cause loss of invertibility in (3.45). Directions of zero curvature can be detected by solving

$$\begin{bmatrix} H & C_{\mathcal{W}(\tau)}^\top \\ C_{\mathcal{W}(\tau)} & 0 \end{bmatrix} \begin{bmatrix} s \\ \xi_{\mathcal{W}(\tau)} \end{bmatrix} = \begin{bmatrix} 0 \\ -(e_k)_{\mathcal{W}(\tau)} \end{bmatrix}, \tag{3.53}$$

with $e_k \in \mathbb{R}^m$ the kth unit vector. H is singular on the null-space of $C_{\mathcal{W}(\tau)^+)}$ if and only if $\xi_{\mathcal{W}(\tau^+)} = 0$. Then, s solves

$$Hs = 0, \quad Aks = -1, \quad C_{\mathcal{W}} + s = 0, \tag{3.54}$$

and all points $\tilde{x} = x(\tau^+) + \sigma s, \sigma > 0$ are also optimal solutions if $\tilde{x}$ is primal feasible. The largest σ can be determined from the ratio test

$$\sigma = \mathrm{RT}(u - Cx(\tau^+), Cs). \tag{3.55}$$

If $\sigma = \infty$, then the parametric QP is unbounded beyond τ^+, and in particular, in $\tau = 1$. Otherwise, let i_l be a minimizing index of a ratio set that delivered the minimizer σ in (3.55), and let $x(\tau^+) := x(\tau^+) + \sigma s$. By construction of σ, the constraint row i_l is active in $x(\tau^+)$ and can be added to the working set via $\mathcal{W}(\tau^+) \leftarrow \mathcal{W}(\tau^+) \cup \{i_l\}$.

Algorithm 4: Parametric active set method

Initialization:

- $x(0), z(0)$ primal-dual input from previous solved QP (or initialized at 0),
- $\mathcal{W}_0$ associated working set,
- $\tau = 0$, and $g(0), u(0)$ initialized at previous solved QP (or 0).

for $k = 0, 1, \ldots$ **do**
  Solve (3.45) to find $\mathrm{d}x$ and $\mathrm{d}z$;
  Find maximum homotopy step length $\mathrm{d}\tau$ and possibly the index of a blocking constraint i or a blocking multipliers sign change j from (3.47) and (3.48);
  if $d\tau \geq 1 - \tau$ **then**
    stop with solution $(x(\tau) + (1-\tau)\mathrm{d}x, z(\tau) + (1-\tau)\mathrm{d}z)$;
  else
    $\tau^+ \leftarrow \tau + \mathrm{d}\tau$;
    $x(\tau^+) \leftarrow x(\tau) + \mathrm{d}\tau\mathrm{d}x$;
    $z(\tau^+) \leftarrow z(\tau) + \mathrm{d}\tau\mathrm{d}z$;
    $\mathcal{W}(\tau^+) \leftarrow \mathcal{W}(\tau)$;
    if *i is blocking* **then**
      $\mathcal{W}(\tau^+) \leftarrow \mathcal{W}(\tau^+) \cup \{i\}$;
      If $\mathcal{W}(\tau^+)$ is linearly dependent, try finding an exchange variable or stop due to infeasibility (following (3.50));
    else
      if *sign change of j is blocking* **then**
        $\mathcal{W}(\tau^+) \leftarrow \mathcal{W}(\tau^+) \setminus \{j\}$;
        If H has nonpositive curvature on the null-space of $\mathcal{W}(\tau^+)$, find an exchange constraint index (following (3.55)); Otherwise stop due to unboundedness of (QP);
      else
        $\tau \leftarrow \tau^+$, $\mathcal{W}(\tau) \leftarrow \mathcal{W}(\tau^+)$;
        Find new step direction;
      end
    end
  end
end

3.3 Interior-Point method

This section presents the most popular interior-point method for convex QPs. It is based on a direct extension of Mehrotra's predictor-corrector originally developed for linear programs [Mehrotra, 1992]. Interior-point methods solve (QP) by applying Newton's method to a sequence of modified versions of the KKT conditions. Such a sequence follows a specific central path for ensuring global convergence. The main steps are summarized below.

The KKT conditions (KKT) can be rewritten by introducing the slack vector $s \geq 0$

$$\begin{aligned} &Hx + C^\top z + g = 0, \\ &Cx - s - u = 0, \\ &s_i z_i = 0, \quad i = 1, 2, ..., m, \\ &(s, s) \geq 0. \end{aligned} \tag{3.56}$$

Given a current iterate (x, s, z) that satisfies $(s, z) > 0$, we can define a complementarity measure μ by

$$\mu = \frac{s^\top z}{m}. \tag{3.57}$$

We can then derive path-following primal-dual methods by considering the perturbed KKT conditions given by

$$F(x, s, z; \sigma, \mu) = \begin{bmatrix} Hx - A^T\lambda + c \\ Cx - s - u \\ Y\Lambda e - \sigma\mu e \end{bmatrix} = 0, \tag{3.58}$$

where

$$Y = \text{diag}(y_1, y_2, ..., y_m), \quad \Lambda = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_m), \quad e = (1, 1, ..., 1)^T,$$

and $\sigma \in [0, 1]$. The solutions to (3.58) for all positive values of σ and μ define the central path, which is a trajectory that leads to the solution of the quadratic program as $\sigma\mu$ tends to zero.

By fixing μ and applying Newton's method to (3.58), we obtain the linear system

$$\begin{bmatrix} H & 0 & -C^\top \\ C & -I & 0 \\ 0 & \Lambda & Y \end{bmatrix} \begin{bmatrix} \mathrm{d}x \\ \mathrm{d}s \\ \mathrm{d}z \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -\Lambda Y e + \sigma\mu e \end{bmatrix}, \tag{3.59}$$

where $\mathrm{d}x$, $\mathrm{d}s$, and $\mathrm{d}z$ are the step sizes for x, s, and z respectively, and r_d, r_p equal

$$\begin{aligned} r_d &= Hx - C^\top z + g, \\ r_p &= Cx - s - u. \end{aligned} \tag{3.60}$$

We obtain the next iterate by setting

$$(x^+, s^+, z^+) = (x, s, z) + \alpha(\mathrm{d}x, \mathrm{d}s, \mathrm{d}z), \tag{3.61}$$

where α is chosen to retain the inequality $(s^+, z^+) > 0$ and possibly to satisfy various other conditions (see, e.g., the different variants in [Nocedal and Wright, 2006, Chapter 16]).

Convergence properties: Primal-Dual Interior point methods converge globally under the existence of a solution for (QP) [Monteiro, 1994]. The complexity of the method is typically quadratic [Potra and Wright, 2000] and [Boyd and Vandenberghe, 2004, Chapter 11].

Pros and Cons: Primal-dual interior-point methods [Wright, 1997, Chapter 16] became popular in the 90s due to their good practical performances across a wide range of problems. On the positive side, unlike active-set methods, they benefit from early termination and can easily exploit the sparsity structure of the problem. Furthermore, they converge under far more robust guarantees than active-set methods, and by construction, they also control the primal-dual gap. Moreover, another considerable advantage is that the primal-dual interior point algorithm is easy to implement and generalizes to a wide range of cones (linear, SOCP, SDP, exponential, etc.). Its certainly explains its widespread use and implementations among numerous popular solvers such as commercial solvers Gurobi [Optimization, 2020] and Mosek [Mosek, 2022], closed-source BPMPD [Mészáros, 1999], and open-source solvers OOQP [Gertz and Wright, 2003], ECOS [Domahidi et al., 2013], CVXOPT [Andersen et al., 2013], qpSWIFT [Pandala et al., 2019], HPIPM [Frison and Diehl, 2020], and Clarabel[2].

Yet, because of the homotopy-based structure of this family of methods, one of their main drawbacks is the difficulty of using them with warm-starting procedures when solving a sequence of related QPs (e.g., for solving SQP or MPC tasks).

Detailed implementation (from [Nocedal and Wright, 2006]): In the predictor-corrector algorithm, first an affine scaling step $(\mathrm{d}x^{\mathrm{aff}}, \mathrm{d}s^{\mathrm{aff}}, \mathrm{d}z^{\mathrm{aff}})$ is derived from (3.59) with $\sigma = 0$. It is then improved upon by computing a corrector step. To do so, the centering parameter is estimated as $\sigma = \left(\frac{\mu_{\mathrm{aff}}}{\mu}\right)^3$ with

$$\mu = \frac{(s^k)^\top z^k}{m}, \tag{3.62}$$

$$\mu_{\mathrm{aff}} = (s^k + \hat{\alpha}_{\mathrm{aff}} \mathrm{d}s^{\mathrm{aff}})^\top (z^k + \hat{\alpha}_{\mathrm{aff}} \mathrm{d}z^{\mathrm{aff}})/m, \tag{3.63}$$

for $\hat{\alpha}_{\mathrm{aff}} = \max\left\{\alpha \in (0,1] \mid (s^k, z^k) + \alpha(\mathrm{d}s^{\mathrm{aff}}, \mathrm{d}z^{\mathrm{aff}}) \geq 0\right\}$.

The full correction step $\mathrm{d}x, \mathrm{d}s, \mathrm{d}z$ is then obtained as a solution of the following system:

$$\begin{bmatrix} H & 0 & -C^T \\ C & -I & 0 \\ 0 & \Lambda & Y \end{bmatrix} \begin{bmatrix} \mathrm{d}x \\ \mathrm{d}s \\ \mathrm{d}z \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -\Lambda Y e - \mathrm{d}Y^{\mathrm{aff}} \mathrm{d}\Lambda^{\mathrm{aff}} e + \sigma\mu e \end{bmatrix}, \tag{3.64}$$

The step lengths are finally selected via $\alpha = \min(\alpha_\tau^{\mathrm{pri}}, \alpha_\tau^{\mathrm{dual}})$, where

$$\begin{aligned} \alpha_\tau^{\mathrm{pri}} &= \max\{\alpha \in (0,1] : s + \alpha \mathrm{d}s \geq (1-\tau)s\}, \\ \alpha_\tau^{\mathrm{dual}} &= \max\{\alpha \in (0,1] : z + \alpha \mathrm{d}z \geq (1-\tau)\lambda\}. \end{aligned} \tag{3.65}$$

The parameter $\tau \in (0,1)$ controls how far we are back off from the maximum step for which the conditions $s + \alpha \mathrm{d}s \geq 0$ and $z + \alpha \mathrm{d}z \geq 0$ are satisfied (usually $\tau \approx 0.99$). There exist other variants with different step lengths for the primal and dual steps, which have demonstrated numerically better performance (see for more details, e.g., [Nocedal and Wright, 2006, Chapter 16]).

[2] https://github.com/oxfordcontrol/Clarabel.rs

Algorithm 5: Predictor-corrector Algorithm

Inputs:
$\hat{x}$, $\hat{s}$, $\hat{z}$ given by the user;
Initialization:
Set $x^0 = \hat{x}$, $s^0 = \max(1, |\hat{s} + \mathrm{d}s^{\text{aff}}|)$, $z^0 = \max(1, |\hat{z} + \mathrm{d}z^{\text{aff}}|)$, solving (3.59) with $\sigma = 0$ for $(\mathrm{d}s^{\text{aff}}, \mathrm{d}z^{\text{aff}})$;
for $k = 0, 1, \ldots$ **do**
  Solve (3.59) with $\sigma = 0$ for $(\mathrm{d}x^{\text{aff}}, \mathrm{d}s^{\text{aff}}, \mathrm{d}z^{\text{aff}})$;
  Calculate $\mu = \frac{(s^k)^\top z^k}{m}$;
  Calculate $\hat{\alpha}_{\text{aff}} = \max\left\{\alpha \in (0, 1] \mid (s^k, z^k) + \alpha(\mathrm{d}s^{\text{aff}}, \mathrm{d}z^{\text{aff}}) \geq 0\right\}$;
  Calculate $\mu_{\text{aff}} = (s^k + \hat{\alpha}_{\text{aff}}\mathrm{d}s^{\text{aff}})^\top (z^k + \hat{\alpha}_{\text{aff}}\mathrm{d}z^{\text{aff}})/m$;
  Set centering parameter $\sigma = \left(\frac{\mu_{\text{aff}}}{\mu}\right)^3$;
  Solve (3.64) for $(\mathrm{d}x, \mathrm{d}s, \mathrm{d}z)$;
  Choose $\tau_k \in (0, 1)$ and set $\hat{\alpha} = \min(\alpha_\tau^{\text{primal}}, \alpha_\tau^{\text{dual}})$ (see (3.65));
  Set $(x^{k+1}, s^{k+1}, z^{k+1}) = (x^k, s^k, z^k) + \hat{\alpha}(\mathrm{d}x, \mathrm{d}s, \mathrm{d}z)$;
end

3.4 ADMM based-methods

The alternating direction method of multipliers (ADMM) [Glowinski and Marroco, 1975, Gabay and Mercier, 1976, Fortin and Glowinski, 2000, Eckstein, 1989, Parikh and Boyd, 2014] is an operator splitting method, which can be viewed as a simple variation of the classical alternating projections algorithm for finding a point in the intersection of two convex sets. Roughly speaking, ADMM adds a dual-state variable to the basic method, which substantially improves convergence. Fundamentally, it stems from making ADMM a method equivalent to the Douglas-Rachford spliting (DRS) [Gabay, 1983]. The latter method benefits from strong convergence guarantees under relatively weak assumptions since it is known to be an averaged operator [Lions and Mercier, 1979].

More formally, ADMM solves convex problems of the form

$$\begin{aligned} \min_{x \in \mathbb{R}^n, z \in \mathbb{R}^m} \; & v(x) + w(z), \\ \text{s.t.}, \; & x = z, \end{aligned} \tag{3.66}$$

where v and w may be nonsmooth or take on infinite values to encode implicit constraints. The basic ADMM algorithm is

$$\begin{aligned} x^{k+1} &= \arg\min_{x \in \mathbb{R}^n} v(x) + \rho/2 \|x - z^k - \lambda^k\|_2^2, \\ z^{k+1} &= \arg\min_{z \in \mathbb{R}^m} w(z) + \rho/2 \|x^{k+1} - z - \lambda^k\|_2^2, \\ \lambda^{k+1} &= \lambda^k - x^{k+1} + z^{k+1}, \end{aligned} \tag{ADMM}$$

where $\rho > 0$ is a step-size parameter and λ is the scaled dual variable associated with the constraint $x = z$.

Overall this method can reliably provide solutions to modest accuracy after a relatively small number of iterations and scale extremely well to solve large problems. Its benefits of being very simple to implement (i.e., only a few lines of code) and applicable for a variety of conic constraint types (see, e.g., [Garstka et al., 2021, O'Donoghue et al., 2016]). It is also naturally parallelizable. In the rest of the section, we will detail two famous practical implementations of the ADMM algorithm: (i) a primal version at the core of the OSQP solver [Stellato et al., 2020]; and (ii)

another one that solves the homogeneous self-dual embedding problem from which stems the SCS solver [O'Donoghue et al., 2016].

3.4.1 Primal version

Algorithm summary: A primal version of ADMM would consider solving directly the primal problem (QP). It can equivalently be written with a supplementary slack variable z as

$$\begin{aligned} \min_{x\in\mathbb{R}^n, z\in\mathbb{R}^m} & f(x), \\ & Cx = z, \\ & z \leq u. \end{aligned} \tag{3.67}$$

v and w can be chosen as follows for fitting to (3.67) formalism,

$$\begin{aligned} v(x,z) &= f(x) + \mathcal{I}_{\text{eq}}(x,z), \\ w(x,z) &= \mathcal{I}_{\mathcal{C}}(z), \end{aligned} \tag{3.68}$$

where $\mathcal{I}_{\text{eq}}$ and $\mathcal{I}_{\mathcal{C}}$ are the indicator functions given by the equations:

$$\mathcal{I}_{\text{eq}}(x,z) = \begin{cases} 0 & \text{if } Cx = z, \\ +\infty & \text{otherwise,} \end{cases} \quad \mathcal{I}_{\mathcal{C}}(z) = \begin{cases} 0 & \text{if } z \leq u, \\ +\infty & \text{otherwise,} \end{cases} \tag{3.69}$$

The problem (3.67) can then be rewritten as the equivalent

$$\begin{aligned} \min_{\substack{(\hat{x},\hat{z})\in\mathbb{R}^n\times\mathbb{R}^m \\ (x,z)\in\mathbb{R}^n\times\mathbb{R}^m}} & v(\hat{x},\hat{z}) + w(x,z), \\ & (\hat{x},\hat{z}) = (x,z). \end{aligned} \tag{3.70}$$

An iteration of ADMM consists of the following steps:

$$(\tilde{x}^{k+1}, \tilde{z}^{k+1}) \leftarrow \arg\min_{(\tilde{x},\tilde{z}):C\tilde{x}=\tilde{z}} \left(f(\tilde{x}) + \frac{\sigma}{2}\|\tilde{x} - x^k + \frac{1}{\sigma}w^k\|_2^2 + \frac{\rho}{2}\|\tilde{z} - z^k + \frac{1}{\rho}y^k\|_2^2 \right), \tag{3.71a}$$

$$x^{k+1} \leftarrow \alpha\tilde{x}^{k+1} + (1-\alpha)x^k + \frac{1}{\sigma}w^k, \tag{3.71b}$$

$$z^{k+1} \leftarrow \Pi_{\mathcal{C}}\left(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k + \frac{1}{\rho}y^k \right), \tag{3.71c}$$

$$w^{k+1} \leftarrow w^k + \sigma\left(\alpha\tilde{x}^{k+1} + (1-\alpha)x^k - x^{k+1} \right), \tag{3.71d}$$

$$y^{k+1} \leftarrow y^k + \rho\left(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k - z^{k+1} \right), \tag{3.71e}$$

where $\sigma > 0$ and $\rho > 0$ are the step-size parameters, $\alpha \in (0,2)$ is a relaxation parameter, and $\Pi_{\mathcal{C}}$ denotes the Euclidean projection onto the set defined by $\mathcal{I}_{\mathcal{C}}$. The introduction of the splitting variable $\tilde{x}$ ensures that the subproblem in (3.71a) is always solvable.

Convergence properties: It has been shown [Banjac et al., 2019] that the sequence provided by (3.71) converges globally if there exists a solution for (QP). Moreover, ADMM-based methods are known to typically have a linear rate of convergence [Nishihara et al., 2015][3].

[3] Note that the mentioned result is shown for strictly convex QP.

Pros and Cons: Apart from the previously mentioned advantages of ADMM-based methods (i.e., simple, scalable, strong convergence properties), it can naturally exploit the sparsity structure of the problem. It can also easily be warm-started and hot-started with previous matrix factorization, which makes it particularly efficient for MPC or SQP-type tasks. The method also benefits from an early termination strategy, which can be useful for some tasks, such as embedded applications. All these advantages certainly explain its widespread use. A popular open-source implementation of this algorithm is the well-known OSQP [Stellato et al., 2020].

Yet, this method depicts a relatively slow convergence rate, particularly for reaching high accuracies. Moreover, the primal ADMM-based version presented here has only an asymptotic control over the primal-dual gap, making it practically less attractive for some applications requiring accurate control over it.

Detailed implementation of OSQP [Stellato et al., 2020]: We detail in this part how solving (3.71a). Evaluating the ADMM step (3.71a) involves solving the equality constrained QP

$$\begin{aligned} \min_{\substack{\hat{x}\in\mathbb{R}^n \\ \hat{z}\in\mathbb{R}^m}} & f(\hat{x}) + \frac{\sigma}{2}\|\tilde{x} - x^k\|^2 + \frac{\rho}{2}\|\tilde{z} - z^k + \rho^{-1}y^k\|^2 \\ \text{s.t.,} & \; C\tilde{x} = \tilde{z}. \end{aligned} \tag{3.72}$$

The optimality conditions for this equality-constrained QP are:

$$H\tilde{x}^{k+1} + g + \sigma(\tilde{x}^{k+1} - x^k) + C^\top \nu^{k+1} = 0, \tag{3.73a}$$
$$\rho(\tilde{z}^{k+1} - z^k) + y^k - \nu^{k+1} = 0, \tag{3.73b}$$
$$C\tilde{x}^{k+1} - \tilde{z}^{k+1} = 0, \tag{3.73c}$$

where $\nu^{k+1} \in \mathbb{R}^m$ is the Lagrange multiplier associated with the constraint $Cx = z$. By eliminating the variable $\tilde{z}^{k+1}$ from (3.73b), the above linear system reduces to

$$\begin{bmatrix} H + \sigma I & C^\top \\ C & -\rho^{-1}I \end{bmatrix} \begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - g \\ z^k - \rho^{-1}y^k \end{bmatrix}, \tag{3.74}$$

with $\tilde{z}^{k+1}$ recoverable as

$$\tilde{z}^{k+1} = z^k + \rho^{-1}(\nu^{k+1} - y^k). \tag{3.75}$$

The linear system (3.74) can be solved using a **direct method** by first factoring the KKT matrix (with for example an LDLT factorization) and then performing forward and backward substitution. Since the KKT matrix remains the same for every iteration of ADMM, only one factorization is needed. The factors can be cached and reused in subsequent iterations. This approach is very efficient when the factorization cost is considerably higher than the cost of forward and backward substitutions, so that each iteration is computed quickly. Note that if ρ or σ change, the KKT matrix needs to be factored again.

With large-scale QPs, factoring linear system (3.74) might be prohibitive. In these cases it might be more convenient to use an **indirect method** by solving instead the linear system

$$(H + \sigma I + \rho C^\top C)\hat{x}^{k+1} = \sigma x^k - g + C^\top(\rho z^k - y^k), \tag{3.76}$$

obtained by eliminating ν^{k+1} from (3.74). Note that the coefficient matrix in the above linear system is always positive definite. The linear system can therefore be solved with an iterative scheme such as the conjugate gradient method [Van Loan and Golub, 1996, Nocedal and Wright, 2006].

Algorithm 6: OSQP algorithm

Inputs: x^0, z^0, y^0 and parameters $\rho > 0$, $\sigma > 0$, $\alpha \in (0, 2)$,
while *termination criterion is satisfied* **do**
  $\hat{x}^{k+1}, \hat{z}^{k+1}$ as solution to (3.74);
  $\hat{z}^{k+1} \leftarrow z^k + \rho^{-1}(\nu^{k+1} - y^k)$;
  $x^{k+1} \leftarrow \alpha\hat{x}^{k+1} + (1-\alpha)x^k$;
  $z^{k+1} \leftarrow \Pi_{\mathcal{C}}(\alpha\hat{z}^{k+1} + (1-\alpha)z^k - z^{k+1})$;
  $y^{k+1} \leftarrow y^k + \rho(\alpha\hat{z}^{k+1} + (1-\alpha)z^k - z^{k+1})$;
end

3.4.2 Homogeneous self-dual embedding version

Algorithm summary: The algorithm considers solving more general conic constrained problems which writes down in standard form

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} g^\top x, & \max\limits_{z\in\mathbb{R}^m} -u^\top z, \\ \text{s.t., } Cx + s = u, & \text{s.t., } -C^\top z + r = g, \\ (x,s) \in \mathbb{R}^n \times \mathcal{K}, & (r,z) \in \{0\}^n \times \mathcal{K}^\star, \end{array} \tag{3.77}$$

Here, $x \in \mathbb{R}^n$ and $s \in \mathbb{R}^m$ are the primal variables, and $r \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ are the dual variables. We refer to x as the primal variable, s as the primal slack variable, y as the dual variable, and r as the dual residual. The set K is a nonempty, closed, convex cone with dual cone $\mathcal{K}^\star$, and $\{0\}^n$ is the dual cone of $\mathbb{R}^n$, so the cones $\mathbb{R}^n \times \mathcal{K}$ and $\{0\}^n \times \mathcal{K}^*$ are duals of each other. The problem data are $C \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $g \in \mathbb{R}^n$, and the cone $\mathcal{K}$.

The original pair of problems (3.77) can be converted into a single homogeneous self-dual embedding problem [O'Donoghue et al., 2016, Section 2.3]:

$$\underbrace{\begin{bmatrix} r \\ s \\ \kappa \end{bmatrix}}_{\stackrel{\text{def}}{=} t} = \underbrace{\begin{bmatrix} 0 & C^T & g \\ -C & 0 & u \\ -g^T & -u^T & 0 \end{bmatrix}}_{\stackrel{\text{def}}{=} Q} \underbrace{\begin{bmatrix} x \\ y \\ \tau \end{bmatrix}}_{\stackrel{\text{def}}{=} h}, \quad (x, y, \tau, r, s, \kappa) \in \underbrace{\mathbb{R}^n \times \mathcal{K}^\star \times \mathbb{R}^+}_{\stackrel{\text{def}}{=} \mathcal{C}} \times \underbrace{\{0\}^n \times \mathcal{K} \times \times \mathbb{R}^+}_{\stackrel{\text{def}}{=} \mathcal{C}^\star}, \tag{3.78}$$

which is hence equivalent to

$$\text{Find } (t, h) \in \mathcal{C} \times \mathcal{C}^\star \text{ s.t., } t = Qh. \tag{3.79}$$

This embedding introduces two new variables, τ and κ, that are nonnegative and complementary, i.e., at most one is nonzero. Any solution of the self-dual embedding $(x, s, r, y, \tau, \kappa)$ falls into one of three cases:

- $\tau > 0$, $\kappa = 0$: $(x/\tau, y/\tau, s/\tau)$ is a primal dual solution,
- $\tau = 0$, $\kappa > 0$:
  - if $u^\top y < 0$ then the (QP) is primal infeasible,
  - if $g^\top x < 0$, then the (QP) is dual infeasible,
- $\tau = \kappa = 0$: the problem is primal or dual infeasible.

Problem (3.79) can thus be written under the formalism (3.66) by choosing $v = \mathcal{I}_{\mathcal{C}\times\mathcal{C}^\star}$ and $w = \mathcal{I}_{Qh=t}$

$$\begin{aligned} &\min_{\substack{t,h\in\mathcal{C}\times\mathcal{C}^\star \\ \hat{t},\hat{h}\in\mathcal{C}\times\mathcal{C}^\star}} \mathcal{I}_{\mathcal{C}\times\mathcal{C}^\star}(t,h) + \mathcal{I}_{Qt=h}(\hat{t},\hat{h}), \\ &(t,h) = (\hat{t},\hat{h}), \end{aligned} \tag{3.80}$$

where $\mathcal{I}_{\mathcal{S}}$ is the indicator function of the set $\mathcal{S}$. Hence the ADMM iterations write down

$$(\hat{t}^{k+1},\hat{h}^{k+1}) \leftarrow \Pi_{Qt=h}(t^k+\lambda^k, h^k+\mu^k), \tag{3.81a}$$
$$t^{k+1} \leftarrow \Pi_{\mathcal{C}}(\hat{t}^{k+1}-\lambda^k), \tag{3.81b}$$
$$h^{k+1} \leftarrow \Pi_{\mathcal{C}^\star}(\hat{h}^{k+1}-\mu^k), \tag{3.81c}$$
$$\lambda^{k+1} \leftarrow \lambda^k - \hat{t}^{k+1} + t^{k+1}, \tag{3.81d}$$
$$\mu^{k+1} \leftarrow \mu^k - \hat{h}^{k+1} + h^{k+1}, \tag{3.81e}$$

with $\Pi_{\mathcal{S}}$ the Euclidean projection onto set $\mathcal{S}$, and λ and μ are dual variables for the equality constraints on t and h respectively.

Convergence properties: It has been shown [O'Donoghue et al., 2016] that the sequence provided by (3.81) converges globally as soon as there exists a solution for (3.79). Moreover ADMM-based methods are known to have typically a linear rate of convergence [Nishihara et al., 2015][4].

Pros and Cons: Apart from previously mentioned advantages of ADMM-based methods (simple, scalable, strong convergence properties), it can naturally exploit the sparsity structure of the problem. It can also easily be warm-started and hot-started with previous matrix factorization which makes it particularly efficient for MPC or SQP type tasks. The method benefits as well of early termination strategy which can be use-full for some tasks such as embedded applications. Contrary to the previous one, this ADMM-based version works for a broader range of cones. Furthermore, by construction, it directly controls the primal-dual gap. All these advantages certainly explain its widespread use. A popular open-source implementation of this algorithm is the well-known SCS [O'Donoghue et al., 2016].

Yet, this method depicts a relatively slow rate of convergence, particularly for reaching high accuracies. Moreover, since the primal ADMM-based version exploits more the structure of QPs, this more generic method can be slower for solving some standard QP problems.

Detailed implementation of SCS [O'Donoghue et al., 2016]: In this part we detail how solving (3.81a) and simplifying other iterations considering the problem conic problem structure.

Step (3.81a) amounts to compute a projection onto Q by solving

$$\min_{t,h} \left[\frac{1}{2}\|t-t^k-h^k\|_2^2 + \frac{1}{2}\|h-t^k-h^k\|_2^2\right] \quad \text{s.t.} \quad h = Qt, \tag{3.82}$$

with variables t and h. The KKT conditions for this problem are

$$\begin{bmatrix} I & Q^\top \\ Q & -I \end{bmatrix} \begin{bmatrix} t \\ \mu \end{bmatrix} = \begin{bmatrix} t^k + h^k \\ t^k + h^k \end{bmatrix}, \tag{3.83}$$

where $\mu \in \mathbb{R}^{m+n+1}$ is the dual variable associated with the equality constraint $Qt = h = 0$. By eliminating μ, we obtain

$$\tilde{t}^{k+1} = (I + Q^\top Q)^{-1}(I - Q)(t^k + h^k). \tag{3.84}$$

[4]Note that the mentioned result is shown for strictly convex QP.

Since the matrix Q is skew-symmetric, it thus simplifies to

$$\hat{t}^{k+1} = (I + Q)^{-1}(t^k + h^k). \tag{3.85}$$

The linear system in (3.85) can be efficiently solved using direct methods or indirect methods (e.g., conjugate gradient (CG) methods). For more details we refer the reader to [O'Donoghue et al., 2016, Section 4.1]. Combining these simplifications, the final algorithm is summarized in Algorithm 7.

Algorithm 7: SCS algorithm

Inputs: t^0, h^0,
while *termination criterion is satisfied* **do**
  $\hat{t}^{k+1} = (I + Q)^{-1}(t^k + h^k)$;
  $t^{k+1} = \Pi_{\mathcal{C}}(\hat{t}^{k+1} - h^k)$;
  $h^{k+1} = h^k - \hat{t}^{k+1} + t^{k+1}$;
end

3.5 Proximal Augmented Lagrangian-based methods

This family of methods is primarily based on the idea of Lagrangian relaxations with an additional quadratic penalization term (possibly piecewise) to encourage the feasibility of the iterate (see, e.g., [Nocedal and Wright, 2006, Section 17.3]). This kind of technique emerged in the 70s through the works of Hestenes and Powell [Powell, 1969, Hestenes, 1969] and then later with those of Rockafellar [Rockafellar, 1976a], which emphasized their link with the so-called "proximal-point method". Indeed, Augmented Lagrangian (AL)-type methods naturally arise by applying proximal-point methods on either the dual or saddle-point formulations to (QP), thereby offering the advantage, similarly to the DRS operator, of converging under relatively weak assumptions. Yet, these methods differ from ADMM-based methods in at least two practical structural ways: (i) AL-based methods can converge at a sharper rate than ADMM-based approaches, and they do not either struggle to converge to high accuracy levels; (ii) yet AL-based approaches have less scalability than ADMM-based approaches: among others, they do not benefit from obvious parallelization, and they require a priori case by case methods for solving subproblems the best efficient way.

We review in this section the Method of Multipliers (MM), the Proximal Method of Multipliers and a Primal-Dual Proximal-Augmented Lagrangian Method of Multipliers. Finally, we end the section with a discussion about practical ingredients that are key when implementing AL-based approaches.

3.5.1 Method of Multipliers

Algorithm summary: The method of multipliers (MM) is based on the minimization of the augmented Lagrangian (AL) penalty function. The so-called AL function [Rockafellar, 1976a, Section 4] relies on augmenting the cost function f with a shifted ℓ_2 penalization of the constraint (*i.e.*, $Cx - u$) with respect to a weighted dual multipliers μz:

$$\mathcal{L}_A(x, z; \mu) \overset{\text{def}}{=} f(x) + \tfrac{1}{2\mu}\left(\|[Cx - u + \mu z]_+\|_2^2 - \|\mu z\|_2^2\right),$$

where $\mu > 0$ is a positive penalty parameter. In the case where we replace the inequality constraints $Cx \leq u$ of (QP) by equalities $Cx = u$, this would be equivalent to augmenting the standard Lagrangian $\mathcal{L}$ with a weighted penalization:

$$\mathcal{L}_A(x, z; \mu) = \mathcal{L}(x, z) + \tfrac{1}{2\mu}\|Cx - u\|_2^2.$$

Many optimization strategies rely on alternating (i) minimizing $\mathcal{L}_A$ with respect to the primal variables, and (ii) maximizing $\mathcal{L}_A$ with respect to the dual variables. MM [Hestenes, 1969, Powell, 1969] is one such strategy, where the maximization step (ii) is computed in closed-form. More precisely, considering some sequence $\{\mu^k\}_k$ bounded below by some $\mu^\infty > 0$:

$$\begin{aligned} x^{k+1} &= \arg\min_{x\in\mathbb{R}^n} \mathcal{L}_A(x, z^k; \mu^k), \\ z^{k+1} &= \arg\max_{z\in\mathbb{R}^m} \mathcal{L}_A(x^{k+1}, z; \mu^k) = \left[z^k + \tfrac{1}{\mu^k}(Cx^{k+1} - u)\right]_+. \end{aligned}$$

Yet, in the presence of inequality constraints, the exact minimization of $\mathcal{L}_A(x, z^k; \mu^k)$ with respect to x is generally hard to compute in closed-form since $x \mapsto \mathcal{L}_A(x, z^k; \mu^k)$ is piece-wise quadratic [Sun, 1997]. For this reason, practical MM solvers typically rely on ϵ^k-approximate solutions for computing x^{k+1}:

$$\begin{aligned} x^{k+1} &\approx_{\epsilon^k} \arg\min_{x\in\mathbb{R}^n} \mathcal{L}_A(x, z^k; \mu), \\ z^{k+1} &= \left[z^k + \tfrac{1}{\mu}(Cx^{k+1} - u)\right]_+, \end{aligned} \tag{3.86}$$

for some notion of "$\approx_{\epsilon^k}$". Note, that since $\mathcal{L}_A$ is by construction a Moreau envelop of the Lagrangian with respect to the dual variable it is a C^1 function [Parikh and Boyd, 2014]. Hence, for instance, we can use:

$$\|\nabla_x \mathcal{L}_A(x^{k+1}, z^k; \mu^k)\|_\infty \leq \epsilon^k.$$

For ensuring convergence of the numerical scheme, one typically needs to enforce certain properties on ϵ^k and $\{\mu^k\}_k$. For instance, it is clear that one must have $\epsilon^k \to 0$; other conditions include summability of the sequence $\{\epsilon^k\}_k$, see, e.g., [Conn et al., 1991, Nocedal and Wright, 2006, Rockafellar, 1976a]. The boundedness condition on $\{\mu^k\}_k$ turns out also to be necessary for ensuring convergence [Rockafellar, 1976a].

Convergence properties: It turns out that MM can be seen as a proximal point method applied to (Dual-QP), see [Rockafellar, 1976a, Section 4]. Therefore, it is guaranteed that $\{z^k\}_k$ converges to an optimal solution to (QP) if one exists [Rockafellar, 1976a, Theorem 4]. Furthermore, $\{x^k\}_k$ is a minimizing sequence, *i.e.*, $f(x^k) \to \min_{x\in\mathbb{R}^n} f(x)$. Note that $\{x^k\}_k$ does not necessarily converge to a solution to (QP) [Rockafellar, 1976a]. This equivalence between MM and the dual proximal point algorithm also allows MM to inherit a key property for handling primal infeasible problems. Indeed, assuming (Dual-QP) to be feasible, the dual iterates of MM actually converge linearly to a solution of the dual to the following hierarchical problem [Chiche and Gilbert, 2016]:

$$\begin{aligned} s^\star = \arg\min_{s\in\mathbb{R}^m} \ & \tfrac{1}{2}\|s\|_2^2 \\ \text{s.t. } & x^\star, z^\star \in \arg\min_{x\in\mathbb{R}^n} \max_{z\in\mathbb{R}^m_+} L(x, z, s), \end{aligned} \tag{QP-H}$$

with $L(x, z, s) \overset{\text{def}}{=} f(x) + z^\top(Cx - u - s)$ (namely the Lagrangian of (QP) augmented with a shift variable s). Under strict convexity of f, it results that as soon as (QP) is primal infeasible (i.e., there is no x s.t. $Cx \leq u$), then MM will converge towards a solution $(x^\star, z^\star)$ of (QP-H). Such a solution satisfies the following KKT optimality conditions:

$$\begin{aligned} &Hx^\star + g + C^\top z^\star = 0, \\ &Cx^\star \leq u + s^\star, \\ &(x^\star)^\top Hx^\star + g^\top x^\star + (u + s^\star)^\top z^\star = 0. \end{aligned}$$

In other words, s is the smallest possible (measured in ℓ_2 norm) shift on the constraints that makes (QP) feasible.

T. Rockafellar's general theory on the proximal point algorithms has established that under the existence of a unique solution for the dual of (QP), the primal iteration sequence generated by the MM enjoys an asymptotic Q-superlinear convergence rate. It has also been shown that under some assumptions of a variant of the MM, it exhibits asymptotically one inner iteration per outer iteration [Conn et al., 1992b]. More recent work [Cui et al., 2016] extended T. Rockafellar's results in the context of both the existence of multiple solution for the problem and a wider class of conic problems (namely Semi-Definite Programs, yet under stronger assumptions). It has also been established the asymptotic R-superlinear convergence of the dual iterate.

Pros and Cons: The MM benefits from being efficient in practice for a wide class of problems. Along with its sharp rate of convergence, this method enjoys a number of practical features for solving MPC or SQP tasks: warm-starting, hot-starting factorization, early-stopping, exploiting sparsity. A popular open-source implementation of a MM-based approach is the LANCELOT solver designed for solving also non linear programs [Conn et al., 1992a].

Yet, the MM does not guarantee convergence of the primal iterates, which reduces its robustness. Furthermore, since the inner loops are also only convex, the subproblems are not necessarily that simple to solve (even-though they are already unconstrained). Finally, setting the penalization parameter ϵ^k and μ can be tedious while it considerably affects the practical performance. Several strategies have been proposed in this sense [Bertsekas, 1982, Birgin and Martínez, 2014, Conn et al., 1991].

Detailed implementation: The sequential, approximate minimization of the Augmented Lagrangian penalty function can be performed in a trust region framework [Conn et al., 1992b]. We refer the reader to this specific work and more generally to [Nocedal and Wright, 2006, Chapters 3 and 4] for other strategies dedicated for solving unconstrained problems.

3.5.2 Proximal-Augmented Lagrangian Method

Algorithm summary: The proximal method of multipliers (PMM) is an extension of the MM with stronger convergence guarantees. For a certain $\rho > 0$ ($1/\rho$ corresponds to a step-size), PMM relies on an additional proximal term on the primal variables with [Rockafellar, 1976a, Equation 1.9], considering an additional sequence $\{\mu^k\}_k$ bounded below by some $\mu^\infty > 0$:

$$(x^{k+1}, z^{k+1}) = \underset{x\in\mathbb{R}^n}{\arg\min} \; \underset{z\in\mathbb{R}^m_+}{\max} \mathcal{L}(x,z) + \tfrac{\rho}{2}\|x - x^k\|_2^2 - \tfrac{\mu^k}{2}\|z - z^k\|_2^2. \tag{3.87}$$

Assuming that one can solve (3.87), one can generate a sequence $\{(x^{k+1}, z^{k+1})\}_k$ that converges to a primal-dual optimal pair for (QP). However, (3.87) involves a convex-concave saddle-point problem, for which numerous numerical methods exist yet closed-form solutions can seldom be found. For solving this problem, a natural strategy consists in splitting the optimization procedure in two parts:

$$\begin{cases} x^{k+1} = \arg\min_{x\in\mathbb{R}^n} \Phi^k_{\rho,\mu^k}(x), \\ z^{k+1} = \arg\max_{z\in\mathbb{R}^m_+} \mathcal{L}(x,z) - \frac{\mu^k}{2}\|z - z^k\|_2^2, \end{cases}$$

where $\Phi^k_{\rho,\mu^k}(x) \overset{\text{def}}{=} \mathcal{L}_A(x, z^k; \mu^k) + \frac{\rho}{2}\|x - x^k\|_2^2$ is often referred to as the Proximal augmented Lagrangian [Hermans et al., 2019], and where the second line can be computed in closed form as $z^{k+1} = \left[z^k + \frac{1}{\mu^k}(Cx^{k+1} - u)\right]_+$. As with MM, one generally needs to rely on numerical methods

for the first minimization stage, which we can only solve approximately:

$$\begin{cases} x^{k+1} \approx_{\epsilon^k} \arg\min_{x\in\mathbb{R}^n} \Phi^k_{\rho,\mu^k}(x), \\ z^{k+1} = \left[z^k + \frac{1}{\mu^k}(Cx^{k+1} - u)\right]_+, \end{cases} \tag{3.88}$$

for some notion of "$\approx_{\epsilon^k}$". For instance, we can use:

$$\left\|\nabla_x \Phi^k_{\rho,\mu^k}(x^{k+1})\right\|_\infty \leq \epsilon^k. \tag{3.89}$$

Again, it is clear that convergence of $\{(x^k, z^k)\}$ towards a primal-dual solution requires certain properties of the approximation error, such as $\epsilon^k \to 0$. Other conditions include summability of $\{\epsilon^k\}_k$, and $\{\mu^k\}$ being bounded from below by some $\mu^\infty > 0$, see, e.g., [Rockafellar, 1976a, Theorem 5].

Convergence properties: PMM enjoys the key advantage of guaranteed primal-dual convergence of its iterates under relatively weak assumptions, namely the existence of an optimal primal-dual pair with zero duality gap [Luque, 1984, Proposition 1.2]. This guarantee is stronger than that of MM, for which only the dual variables are guaranteed to converge [Rockafellar, 1976a, Theorem 4]. Furthermore, the proximal augmented Lagrangian function is a piece-wise quadratic strongly convex function, unlike the augmented Lagrangian $x \mapsto \mathcal{L}_A(x, z^k; \mu^k)$ which is only guaranteed to be convex. This allows accelerating the computation of the iterates, e.g., using quadratic Newton-type methods [Hermans et al., 2021]. We leverage this key feature in ProxQP, as further detailed in Section 4.1.4. Finally, Remark 1 highlights that more generally to MM methods, PMM both towards the primal-dual pair of the closest feasible QP solution, which is useful when (QP) is infeasible.

Remark 1 (Convergence towards solutions of (QP-H))**.** *It can be proven following [Chiche and Gilbert, 2016] that if* (QP) *is primal infeasible, its dual* (Dual-QP) *is feasible, and* $\{\mu^k\}_k$ *is a sequence satisfying* $\forall k \in \mathbb{N}$, $\mu^k \geq \mu^\infty$ *for some* $\mu^\infty > 0$*, then, the sequence* $\{(x^k, z^k)\}$ *generated by* (3.87) *converge to a solution of* (QP-H)*.*

PMM also exhibits a typical linear global rate of convergence for their approach [Rockafellar, 1976a, Hermans et al., 2021]. [Luque, 1984] proposes a more general analysis of the asymptotic convergence of the proximal point algorithm (PPA) for the solution of equations of type $0 \in Tz$, where T is a multivalued maximal monotone operator in a real Hilbert space (PMM is hence a particular case of PPA). The convergence rate depends on how rapidly T^{-1} grows away from the set of solutions $\overline{W}$. When this growth is bounded by a power function with exponent s, then for a sequence $\{w^k\}$ generated by PPA, $\{\|z^k - \overline{W}\|\}_k$ converges to zero like $o(k^{-s/2})$, linearly, superlinearly, or in a finite number of iterations according to whether $s \in (0, 1)$, $s = 1$, $s \in (1, +\infty)$, or $s = +\infty$. For PMM applied to QPs, the KKT conditions form a polyhedral variational inequality, so we know the solution map is outer Lipschitz continuous and its constant typically depends on the QP at hand. Hence, that's why we can conclude that the convergence rate is typically at least linear.

Pros and Cons: A considerable advantage from PMM is that is converges globally under weak conditions (i.e., existence of a solution for (QP)). Furthermore, similarly to MM, this method enjoys a number of practical features for solving MPC or SQP tasks: warm-starting, hot-starting factorization, early-stopping, exploiting sparsity. An open-source implementation of a PMM-based approach is the QPALM solver designed for solving also non convex programs [Hermans et al., 2021].

Even-though the PMM converges globally under weak assumptions, it can be sensitive to ill-conditioning [Bambade et al., 2022, Section IV-B] because of quadratic matrix products (such as $C^\top C$) involved for solving (3.88). This will be studied with more details in Section 3.5.3.

Detailed implementation: We detail in this part how solving the inner loop involved in the first step of (3.88).

The Proximal Augmented Lagrangian penalty function Φ^k_{ρ,μ^k} is a strictly convex piece-wise quadratic, which is hence semi-smooth [Mifflin, 1977]. Thus its unique minimum can be found in finite time using a semi-smooth Newton method with exact line-search (see, e.g., convergence proof in [Sun, 1997, Theorem 3] and algorithm in [Hermans et al., 2019, Section IV.C]). Practically speaking, the semi-smooth Newton method is initialized at $\hat{x}^{(0)} = x^k$ and generates a sequence $\hat{x}^{(1)}, \hat{x}^{(2)}, \dots$ via the update rule:

$$\hat{x}^{(l+1)} = \hat{x}^{(l)} + \alpha^\star \mathrm{d}x, \tag{3.90}$$

where the step-size $\alpha^\star$ is computed via an exact line-search:

$$\alpha^\star = \underset{\alpha \geq 0}{\arg\min}\, \Phi^k_{\rho,\mu^k}(\hat{x}^{(l)} + \alpha \mathrm{d}x), \tag{3.91}$$

(note that $\alpha \mapsto \Phi^k_{\rho,\mu^k}(\hat{x}^{(l)} + \alpha \mathrm{d}x)$ is a continuous piecewise quadratic function with a finite number of breaking points), and where $\mathrm{d}x$ is found by solving a linear system of equations [Hermans et al., 2021, Equation 3.4]:

$$\begin{bmatrix} H + \rho I & C^\top_{I_k(\hat{x}^{(l)})} \\ C_{I_k(\hat{x}^{(l)})} & -\mu^k I_{I_k(\hat{x}^{(l)})} \end{bmatrix} \begin{bmatrix} \mathrm{d}x \\ \mathrm{d}z \end{bmatrix} = \begin{bmatrix} -\nabla_x \Phi^k_{\rho,\mu^k}(\hat{x}^{(l)}) \\ 0 \end{bmatrix}, \tag{3.92}$$

where $I_k(\hat{x}^{(l)}) \overset{\text{def}}{=} \{i \in [1,m] \,|\, C_i \hat{x}^{(l)} - u_i + z^k_i \mu^k > 0\}$ refers to the active set of the current subproblem, and $C_{I_k(\hat{x}^{(l)})}$ corresponds to a reduced version of the matrix C containing the active rows indexed by $\mathcal{I}_k(\hat{x}^{(l)})$.

This iterative process is repeated until reaching the accuracy requirement (3.89); that is, as soon as $\|\nabla \Phi^k_{\rho,\mu^k}(\hat{x}^{(l)})\|_\infty \leq \epsilon^k$ it outputs $x^{k+1} \leftarrow \hat{x}^{(l)}$ as an approximate solution. In practice, the linear system (4.1.4) can be solved efficiently via LDLT factorization of this indefinite system. Furthermore ϵ^k typically evolves at an exponential decay rate (i.e., $\epsilon^{k+1} = \tau \epsilon^k$ for some $\tau \in (0,1)$ [Hermans et al., 2021]), and μ^k is decreased only when the primal violation $\|[Cx^{k+1} - u]_+\|_\infty$ is not small enough compared to previous outer-iterate primal violation $\|[Cx^k - u]_+\|_\infty$ (see for more details [Birgin and Martínez, 2014, Hermans et al., 2021]).

3.5.3 Primal-Dual Proximal-Augmented Lagrangian Method

Primal-dual proximal methods of multipliers (PDPMM) is an alternative formulation of PMM. Its name stems from its choice of a primal-dual penalty function for evaluating the quality of the iterates of the sequence of proximal sub-problems. When intermediary optimization subproblems are solved exactly, PMM and PDPMM produce the same iterates. Thereby, PDPMM benefits from the strong convergence properties of PMM. However, the alternate formulation of PMM is convenient and can be leveraged at computation time, and we leverage it in our algorithms.

Algorithm summary: The author of [Marchi., 2021] shows that (3.87) can be equivalently reformulated as a minimization of the following primal-dual merit function:

$$\mathcal{M}^k_{\rho,\mu}(x,z) \overset{\text{def}}{=} f(x) + \tfrac{1}{\mu} \left\| [Cx - u + \mu(z^k - \tfrac{z}{2})]_+ \right\|_2^2 + \tfrac{\rho}{2}\|x - x^k\|_2^2 + \tfrac{\mu}{4}\|z\|_2^2. \tag{3.93}$$

Hence (3.87) is equivalent to:

$$(x^{k+1}, z^{k+1}) = \underset{x \in \mathbb{R}^n, z \in \mathbb{R}^m}{\arg\min}\, \mathcal{M}^k_{\rho,\mu^k}(x,z). \tag{3.94}$$

Of course, solving this optimization step still requires using numerical schemes, and can therefore only be approximated:

$$(x^{k+1}, z^{k+1}) \approx_{\epsilon^k} \underset{x\in\mathbb{R}^n, z\in\mathbb{R}^m}{\arg\min}\ \mathcal{M}^k_{\rho,\mu^k}(x, z), \tag{3.95}$$

for some notion of "$\approx_{\epsilon^k}$". For instance, [Marchi., 2021] uses:

$$\left\| \begin{bmatrix} \nabla_x f(x^{k+1}) + C^\top z^{k+1} + \rho(x^{k+1} - x^k) \\ [Cx^{k+1} - u + \mu^k(z^k - z^{k+1}/2)]_+ - \mu^k z^{k+1}/2 \end{bmatrix} \right\|_\infty \leq \epsilon^k.$$

The ProxQP algorithm relies on a similar error criterion which will be detailed in Section 4.1.4.

Convergence properties: As for PMM, the recurrence (3.95) guarantees that $\{(x^k, z^k)\}_k$ converges to an optimal primal-dual pair under the relatively weak assumptions (the same as in (3.88): as soon as there exists a solution to (QP), and $\{\mu^k\}_k$ is bounded below by some $\mu^\infty > 0$). Furthermore, $\mathcal{M}^k_{\rho,\mu^k}$ is both strongly convex and piece-wise quadratic with respect to both the primal and the dual variables. Hence, as for $\Phi^k_{\rho,\mu}$, its minimization can be performed efficiently using semi-smooth Newton-type methods. As we show in the "Detailed Implementation" section below, the linear systems induced by the minimization of $\mathcal{M}^k_{\rho,\mu^k}$ are naturally better conditioned than those arising from Φ^k_{ρ,μ^k}, which is due to the fact that this formulation naturally avoids quadratic matrix multiplications (such as $C^\top C$). As for PMM it can be shown that the convergence rate is typically at least linear. Finally, Remark 1 also guarantees that more generally to MM methods, PMM and PDPMM both converge towards the primal-dual pair of the closest feasible QP solution, which is useful when (QP) is infeasible.

Pros and Cons: As for PMM, a considerable advantage from PDPMM is that it converges globally under weak conditions (i.e., the existence of a solution for (QP)). Furthermore, as MM this method enjoys a number of practical features for solving MPC or SQP tasks: warm-starting, hot-starting factorization, early-stopping, exploiting sparsity. An open-source implementation of a PMM-based approach is the QPDO solver [Marchi., 2021]. Contrary to PMM, PDPMM enables better structure for being less sensitive to ill-conditioning. Its stems from the fact that it can eliminate quadratic matrix products involved in the internal Newton semi-smooth method used. We detail the calculation in the next paragraph.

Detailed implementation: The inner loop for PDPMM uses exactly the same strategy as for PMM. So we refer the reader to Section 3.5.2 for its detailed implementation. We give below more insights about the numerical conditioning of the linear systems obtained with PDPMM.

For PMM, the gradient involved at l^{th} iteration of a semi-smooth Newton method applied to Φ^k_{ρ,μ^k} is expressed as:

$$\nabla_x \Phi^k_{\rho,\mu^k}(\hat{x}^l) = H\hat{x}^{(l)} + g + \rho(\hat{x}^{(l)} - x^k) + C^\top \left[z^k + \tfrac{1}{\mu^k}(C\hat{x}^{(l)} - u)\right]_+. \tag{3.96}$$

For either low values of μ^k or ill-conditioning of $C_{I_k(\hat{x}^{(l)})}$, the evaluation of the gradient is numerically challenged by the value $\frac{1}{\mu^k} C^\top_{I_k(\hat{x}^{(l)})} C_{I_k(\hat{x}^{(l)})}$ (see, e.g., [Nocedal and Wright, 2006, Chapter 17.4] which gives examples about ill-conditioning resulting from such structures).

When working with $\mathcal{M}^k_{\rho,\mu}$, it is possible to eliminate such square matrix products. A semi-smooth Newton step applied to $\mathcal{M}^k_{\rho,\mu,}$, and initialized at $(\hat{x}^{(0)}, \hat{z}^{(0)}) = (x^k, z^k)$ involves finding at l^{th} inner itertion $\mathrm{d}w = (\mathrm{d}x, \mathrm{d}z)$ satisfying:

$$\nabla^2 \mathcal{M}^k_{\rho,\mu^k}(\hat{x}^{(l)}, \hat{z}^{(l)})\mathrm{d}w + \nabla \mathcal{M}^k_{\rho,\mu^k}(\hat{x}^{(l)}, \hat{z}^{(l)}) = 0, \tag{3.97}$$

More precisely, it reads:

$$\begin{bmatrix} H+\rho I+\frac{2}{\mu}C^\top(I-P^{(k,l)})C & -C^\top(I-P^{(k,l)}) \\ -(I-P^{(k,l)})C & \mu[I-P^{(k,l)}] \end{bmatrix} \begin{bmatrix} \mathrm{d}x \\ \mathrm{d}z \end{bmatrix} = -\begin{bmatrix} \nabla_x f(\hat{x}^{(l)})+\rho(x^{(l)}-x^k)+\frac{2}{\mu}C^\top[w_k^{(l)}-u]_+ \\ -[w_k^{(l)}-u]_+ + \mu/2\hat{z}^{(l)} \end{bmatrix}, \tag{3.98}$$

where $P^{(k,l)}$ stands for one element of the generalized Jacobian of $[.]_+$ at $w_k^{(l)} \overset{\text{def}}{=} Cx^{(l)}+\mu z^k - \mu/2 z^{(l)}$, which consists of a diagonal matrix with entries

$$P_{jj}^{(k,l)} \overset{\text{def}}{=} \begin{cases} 1 & \text{if } C_j^\top x^{(l)}+\mu z_j^k+\mu/2\hat{z}_j^{(l)} \leq u_j, \\ 0 & \text{otherwise.} \end{cases} \tag{3.99}$$

When $P_{jj}^{(k,l)}=1$, it implies (see [Marchi., 2021, Section 3.2] for more details)

$$\mathrm{d}z_j = -(\hat{z}^{(l)})_j. \tag{3.100}$$

The linear system can hence be equivalently formulated as:

$$\begin{cases} \begin{bmatrix} H+\rho I & C^\top_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})} \\ C_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})} & -\mu I_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})} \end{bmatrix} \begin{bmatrix} \mathrm{d}x \\ \mathrm{d}z_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})} \end{bmatrix} = \\ -\begin{bmatrix} H\hat{x}^{(l)}+g+\rho(\hat{x}^{(l)}-x^k)+C^\top_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}\hat{z}^{(l)}_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})} \\ [Cx^{(l)}+\mu z^k-\mu\hat{z}^{(l)}-u]_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})} \end{bmatrix}, \\ \mathrm{d}z_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})^c} = -\hat{z}^{(l)}_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})^c}, \end{cases} \tag{3.101}$$

where:

$$J_k(x,z) \overset{\text{def}}{=} \left\{ j \in [1,n_i] | C_j^\top x+\mu z_j^k+\mu/2 z_j - u_j > 0 \right\}$$

refers to the primal-dual active set at (x,z) for the current k^{th} sub-problem, and $J_k(x,z)^c$ stands for its complementary set (i.e., the set of inactive constraints). Contrary to PMM, one can notice that this new linear system does not explicitly contain any square matrix multiplications.

3.5.4 Discussions

There are three practical ingredients that are key when implementing AL-based approaches.

1 – Setting ϵ^k Appropriate choices for ϵ^k are fundamental to any AL method.

- On the one hand, picking values for ϵ^k that are too small leads to unnecessary good approximations of the intermediate optimization problems, even when the iterates are still far away from solutions to (QP). Practically, this scenario corresponds to requiring the internal solver to unnecessarily run for too long when solving the intermediary problems.
- On the contrary, picking ϵ^k too large results in faster computations of the intermediary subproblems at the cost of requiring much more iterations of the AL method, which might not even converge.

There are many different techniques for fixing ϵ^k in practice [Hermans et al., 2021, Rockafellar, 1976a, Birgin and Martínez, 2014]. For ProxQP, we chose to rely on an adaptive strategy similar in spirit to [Conn et al., 1991].

2 – Setting $\boldsymbol{\mu}^k$ Using different values for μ^k has a direct impact on the practical convergence speed (along with the value ρ for PMM and PDPMM) (see, e.g., [d'Aspremont et al., 2021, Chapter 5])

- On the one hand, lower values of μ^k lead to faster convergence (i.e., fewer iterations).
- On the other hand, larger values of μ^k render the intermediary subproblems structurally simpler to solve (by increased strong convexity).

Practically speaking, let us mention that in many solvers (including ProxQP), updating μ^k corresponds to re-factorizing some internal matrices. Hence, we must generally limit the number of updates to μ^k for best practical performances.

3 – Efficient method for computing $(\boldsymbol{x^{k+1}, z^{k+1}})$ There are several different ways to come up with ϵ^k-approximations for (x^{k+1}, z^{k+1}). The choice of the internal routine for this approximation is thereby crucial both for timing and accuracy requirements.

3.6 Summary

Table 3.1: Summary of the different optimization methods reviewed for solving QPs, with associated software libraries and references.

Method	Sub-Family	Solver	Backend	Warm-starting	Hot-starting	Early termination	Primal-dual gap
ACTIVE-SET	Primal	GALAHAD	Dense	✗	✗	✗	✓
	Dual	QUADPROG	Dense	✗ [1]	✗ [1]	✗	✓
	Dual	DAQP	Dense	✓	✓	✓	✓
	Dual	QPNNLS	Dense	✓	✓	✓	✓
	Parametric	QPOASES	Dense	✓	✓	✗	✓
INTERIOR-POINT	Primal-Dual	GUROBI	Sparse	✗	✗	✓	✓
	Primal-Dual	MOSEK	Sparse	✗	✗	✓	✓
	Primal-Dual	CVXOPT	Dense	✗	✗	✓	✓
	Primal-Dual	ECOS	Sparse	✗	✗	✓	✓
	Primal-Dual	QPSWIFT	Sparse	✗	✗	✓	✓
	Primal-Dual	HPIPM	Dense	✗	✗	✓	✓
	Primal-Dual	CLARABEL	Sparse	✗	✗	✓	✓
	Primal-Dual	BPMPD	Sparse	✗	✗	✓	✓
	Primal-Dual	OOQP	Sparse	✗	✗	✓	✓
ADMM	Primal	OSQP	Sparse	✓	✓	✓	✗
	Self-dual embedding	SCS	Sparse	✓	✓	✓	✓
AUGMENTED LAGRANGIAN	MM	LANCELOT	Sparse	✓	✓	✓	✗
	PMM	QPALM	Sparse	✓	✓	✓	✗
	PDPMM	QPDO	Sparse	✗ [1]	✗ [1]	✓	✗
	PDPMM	PROXQP	Sparse & Dense	✓	✓	✓	✓

[1] In their original implementation.

Chapter 4

ProxQP algorithm

Abstract. *In this chapter we first present the various components of ProxQP algorithm, including its convergence properties and implementation details. In particular, since ProxQP is an AL-based method, we discuss how to handle three key aspects for ensuring the computational efficiency of this AL-based approach (see notably the discussion in Section 3.5.4). We then present extensive benchmarks on robotic and standard QP problems.*

This chapter is based on these two works:

- ProxQP: Yet another Quadratic Programming Solver for Robotics and beyond, with Sarah El-Kazdadi, Adrien Taylor, Justin Carpentier, *Robotics: Science and System (RSS)*, 2022;
- ProxQP: an Efficient and Versatile Quadratic Programming Solver for Real-Time Robotics Applications and Beyond, with Fabian Schramm, Sarah El-Kazdadi, Stéphane Caron, Adrien Taylor, Justin Carpentier. Submitted to IEEE Transactions on Robotics (TRO).

Chapter content

4.1 ProxQP . . . 46
4.1.1 The ProxQP algorithm . . . 46
4.1.2 Globalization strategy for scheduling ϵ^k and μ^k . . . 47
4.1.3 Infeasible QPs . . . 48
4.1.4 Solving proximal sub-problems . . . 48
4.2 Convergence properties . . . 51
4.2.1 Organisation of the section . . . 51
4.2.2 Further notations and technical properties . . . 51
4.2.3 Convergence of ProxQP . . . 52
4.2.4 Proof of Theorem 3 . . . 58
4.2.5 Proof of Lemma Lemma 1 . . . 59
4.3 Software Implementation . . . 59
4.3.1 Common QP solvers . . . 59
4.3.2 The ProxSuite library . . . 60
4.4 Applications to robotics and beyond . . . 62
4.4.1 Benchmark setup and scenarios . . . 63
4.4.2 Sparse problems . . . 63
4.4.3 Dense problems . . . 68
4.4.4 Generic QPs . . . 71

4.1 ProxQP

4.1.1 The ProxQP algorithm

ProxQP is detailed in Algorithm 8. It combines a proximal augmented Lagrangian technique (approximated using an internal primal-dual method) with a globalization strategy for scheduling the internal step size and accuracy parameters. More precisely, at iteration k, ProxQP computes candidates $x^{k+1}, \hat{z}^{k+1}$ satisfying

$$(x^{k+1}, \hat{z}^{k+1}) \approx_{\epsilon^k} \underset{x\in\mathbb{R}^n, z\in\mathbb{R}^m}{\arg\min}\ \mathcal{M}^k_{\rho,\mu^k,\alpha}(x,z), \tag{4.1}$$

for a merit function $\mathcal{M}^k_{\rho,\mu^k,\alpha}$ that we detail in Section 4.1.4. The following steps build on the bound-constrained Lagrangian (BCL) strategy (see, e.g., [Conn et al., 1991] and [Nocedal and Wright, 2006, Algorithm 17.4]) for scheduling both ϵ^k and μ^k.

The main idea consists in a fast decrease of ϵ^k when the primal residual $\|[C\hat{x}^{k+1} - u]_+\|_\infty$ is small enough. In this case, we accept the approximation with $z^{k+1} = \hat{z}^{k+1}$ and leave μ^k unchanged with $\mu^{k+1} = \mu^k$. Otherwise, we slightly decrease ϵ^k, reject the approximation with $z^{k+1} = z^k$ and decrease μ^k. Decreasing μ^k corresponds to a heavier penalization of the feasibility constraint within the augmented Lagrangian-based function at the next step. Such a strategy has been proved to perform well in optimization packages such as LANCELOT [Conn et al., 1992c] as well as in robotics for solving constrained optimal control problems [Plancher et al., 2017, El Kazdadi et al., 2021, Jallet et al., 2021, Jallet et al., 2022a]. We detail this globalization technique alongside its resulting global convergence properties for ProxQP in Section 4.1.2. The method used to minimize $\mathcal{M}^k_{\rho,\mu^k,\alpha}$ is explained in Section 4.1.4.

Algorithm 8: ProxQP (practical version)

Inputs:

- initial states: x^0, z^0,
- initial parameters: ϵ^0_{bcl}, ϵ^0, ϵ_{abs}, ρ, $\mu^0 > 0$
- hyper-parameters: $\mu_f, \beta_{\text{bcl}} \in (0,1)$, $\alpha_{\text{bcl}} \in (0, 1/2)$, with $\alpha_{\text{bcl}} + \beta_{\text{bcl}} < 1$, $\mu_{\min} > 0$, $k_{\max} \in \mathbb{N} \cup \{+\infty\}$.

Initialization:

- preconditioning (see Table 4.3.2)
- optional initialization (see Table 4.3.2) of x^0,z^0.

while *Stopping criterion* (3.10) *not satisfied* **do**
  Compute $(x^{k+1}, \hat{z}^{k+1})$ satisfying (4.5) (ϵ^k-approximation to proximal subproblem (4.4)) using Section 4.1.4;
  if $||[Cx^{k+1} - u]_+||_\infty < \epsilon^k_{bcl}$ *using* (4.2) *OR* $k \geq k_{\max}$ **then**
    $\mu^{k+1} = \mu^k$, $\epsilon^{k+1} = \epsilon^k \mu^{k+1}$, $\epsilon^{k+1}_{\text{bcl}} = \epsilon^k_{\text{bcl}} (\mu^{k+1})^{\beta_{\text{bcl}}}$
    $z^{k+1} = \hat{z}^{k+1}$
  else
    $\mu^{k+1} = \max(\mu_{\min}, \mu_f \mu^k)$
    $\epsilon^{k+1} = \epsilon^0 \mu^{k+1}$, $\epsilon^{k+1}_{\text{bcl}} = \epsilon^0_{\text{bcl}} (\mu^{k+1})^{\alpha_{\text{bcl}}}$
    $z^{k+1} = z^k$
  end
  $k \leftarrow k + 1$
end

Output: A (x^k, z^k) satisfying the ϵ_{abs}-approximation criterion (3.10) for problem (QP).

4.1.2 Globalization strategy for scheduling ϵ^k and μ^k

The globalization strategy in ProxQP consists in updating the dual variables z^k—obtained from minimizing a primal-dual proximal augmented Lagrangian merit function (described in Section 4.1.4)—only when the corresponding primal inequality constraint violation (denoted by p^k hereafter) is small enough. More precisely, we use a second internal sequence of tolerances denoted by ϵ^k_{bcl} (which is also tuned within this BCL strategy) and update the dual multipliers only when $p^k \leq \epsilon^k_{\text{bcl}}$, where p^k denotes the ℓ_∞ primal feasibility violation:

$$p^k \overset{\text{def}}{=} \left\| [Cx^{k+1} - u]_+ \right\|_\infty . \tag{4.2}$$

This BCL strategy allows searching for appropriate values for the hyper-parameters ϵ^k, ϵ^k_{bcl}, μ^k. For the constraint penalization parameter μ^k, it proceeds as follows:

- **If $p^k \leq \epsilon^k_{\text{bcl}}$:** the primal inequality constraint violation is satisfactory, keep μ^k as is.
- **Otherwise:** the primal inequality constraint violation is too large, decrease μ^k for subsequent proximal subproblems, which has the effect of leading to heavier penalization of infeasibility, enforcing their satisfaction.

About the accuracy parameters ϵ^k and ϵ^k_{bcl}, the update rules are more technical. The motivation underlying those choices is to ensure global convergence: a geometric-decay type update when the primal inequality constraint violation is good enough, and see [Conn et al., 1991, Lemma 4.1] for when the infeasibility is too large. The detailed strategy is summarized in Algorithm 8.

Convergence properties We analyze a simpler algorithm (see Algorithm 9), whose small variations are more effective in practice. More precisely, it assumes that at each iteration k of the algorithm $\rho^k = \mu^k$ compared to Algorithm 8, where ρ is fixed. Updating ρ is not efficient in practice since it would require a numerical factorization at each iteration. In this setup and under the assumption that there exists a primal-dual solution to (QP) (see Theorem 3 and Remark 2), the iterates of ProxQP are guaranteed to converge to an optimal primal-dual pair.

Algorithm 9: ProxQP (idealized version)

Inputs:

- initial states: x^0, z^0,
- initial parameters: ϵ^0_{bcl}, $\epsilon^0 > 0$, $\rho^0 = \mu^0 \in (0, 1)$
- hyper-parameters: $\mu_f < 1$, $\alpha_{\text{bcl}} \in (0, 1/2)$,$\beta_{\text{bcl}} \in (0, 1)$ with $\alpha_{\text{bcl}} + \beta_{\text{bcl}} < 1$, $k_{\max} \in \mathbb{N} \cup \{+\infty\}$.

for $k = 0, 1, \ldots$ **do**
  Compute $(x^{k+1}, \hat{z})$ satisfying (4.5) at accuracy ϵ^k with $\rho^k = \mu^k$;
  if $p^k \leq \epsilon^k_{bcl}$ *OR* $k \geq k_{\max}$ **then**
    $\mu^{k+1} = \mu^k$, $\epsilon^{k+1} = \epsilon^k \mu^{k+1}$, $\epsilon^{k+1}_{\text{bcl}} = \epsilon^k_{\text{bcl}} (\mu^{k+1})^{\beta_{\text{bcl}}}$,
    $z^{k+1} = \hat{z}^{k+1}$
  else
    $\mu^{k+1} = \mu_f \mu^k$,
    $\epsilon^{k+1} = \epsilon^0 \mu^{k+1}$, $\epsilon^{k+1}_{\text{bcl}} = \epsilon^0_{\text{bcl}} (\mu^{k+1})^{\alpha_{\text{bcl}}}$,
    $z^{k+1} = z^k$.
  end
end

Assumption 1. *The problem* (QP) *is feasible.*

Assumption 2. *The iterates* $\{(x^k, z^k)\}_k$ *generated by* ProxQP *are bounded.*

Theorem 3 (Convergence of ProxQP)**.** *Under Assumption 1 (existence of a solution to QP), Assumption 2 (bounded iterates), and with all parameters set as required in the inputs of Algorithm 9, the iterates* $\{(x^k, z^k)\}$ *of* ProxQP *converge to a solution* $(x^\star, z^\star)$ *of* (QP)*.*

Proof. See Section 4.2.3. □

Lemma 1. *Under Assumption 1 (existence of solution to* (QP)*) and Assumption 2 (bounded iterates), and with all parameters set as required in the inputs of Algorithm 9,* $\exists K \in \mathbb{N}$*, and* $\mu \in \mathbb{R}$ *such that* $\forall k \geq K \ \mu^k = \mu$*. Hence,* $\forall k \geq 0, p^{K+k} \leq \epsilon_{bcl}^K (\mu^K)^{\beta_{bcl} k}$*.*

Proof. See Section 4.2.3. □

Remark 2 (On the necessity of Assumption 2)**.** *The BCL strategy [Conn et al., 1991, Algorithm 1 and Algorithm 2] was originally developed for solving more general optimization problems. BCL relies on an augmented Lagrangian strategy. In other words, we build on BCL for constructing an algorithm that exploits the structural properties of simpler optimization problems (namely QPs). Furthermore, rather than being based on a purely AL strategy, we rely on a proximal AL strategy. We managed to get rid of most of the strong assumptions required for ensuring global convergence of BCL [Conn et al., 1991]. Yet, Assumption* (2) *still appeared necessary in our analysis. We introduced the safeguard parameter* $k_{\max} \in \mathbb{N}$ *to always enforce the algorithm to ultimately always accept the candidates* (4.1) *if Assumption 2 does not hold. This simple trick allows inheriting some nice properties from PMM that include convergence under mild assumptions [Rockafellar, 1976a, Theorem 5].*

Remark 3 (Advantageous numerical properties)**.** *As a by-product of the proof of Theorem 3, the method is guaranteed to use constant penalization parameters after a finite number of iterations. This is advantageous insofar as (1) we inherit favorable properties of proximal point methods on saddle point, and (2) it limits the number of numerical matrix factorizations.*

4.1.3 Infeasible QPs

Finally, Corollary 4 guarantees that, similarly to MM, PMM and PDPMM methods, ProxQP converges towards the primal-dual pair to the closest feasible QP solution, which is useful when (QP) is infeasible.

Corollary 4. *Let* (QP) *be primal infeasible, let its dual* (Dual-QP) *be feasible, let* $\mu > 0$ *and* $k_{\max} \in \mathbb{N}$*. The sequence* $\{(x^k, z^k)\}$ *generated by* ProxQP *with exact sub-problem minimization converges towards a solution of* (QP-H)*.*

Proof. This is a direct consequence of the fact that, after a finite number of iterations, the iterations of ProxQP correspond to those of PMM. □

4.1.4 Solving proximal sub-problems

We now explicit the primal-dual proximal augmented Lagrangian merit function used for generating candidates $(x^{k+1}, \hat{z}^{k+1})$ in (4.1). Then, we explain how to minimize it in quadratic time.

The ProxQP primal-dual proximal augmented Lagrangian merit function We use the same procedure as in [Marchi., 2021] for building our primal-dual proximal augmented Lagrangian merit function $\mathcal{M}^k_{\rho,\mu^k,\alpha}$. By construction, it guarantees that minimizing $\mathcal{M}^k_{\rho,\mu^k,\alpha}$ is equivalent to solving (3.87). Yet, contrary to $\mathcal{M}_{\rho,\mu^k}$, we make a different choice for the parameter $\alpha \in (0,1)$[1], tuned for better practical performance,

$$\mathcal{M}^k_{\rho,\mu^k,\alpha}(x,z) \overset{\text{def}}{=} f(x) + \tfrac{1}{2\alpha\mu^k}\|[Cx-u+\mu^k(z^k+(\alpha-1)z)]_+\|_2^2 + \tfrac{\rho}{2}\|x-x^k\|_2^2 + \tfrac{(1-\alpha)\mu^k}{2}\|z\|_2^2. \tag{4.3}$$

This leads to the following inexact proximal scheme

$$(x^{k+1},z^{k+1}) \approx_{\epsilon^k} \underset{x\in\mathbb{R}^n, z\in\mathbb{R}^m}{\arg\min}\ \mathcal{M}^k_{\rho,\mu^k,\alpha}(x,z), \tag{4.4}$$

where $\approx_{\epsilon^k}$ corresponds to

$$\left\| \begin{bmatrix} \nabla_x f(x^{k+1}) + C^\top z^{k+1} + \rho(x^{k+1}-x^k) \\ [Cx^{k+1}-u+\mu^k(z^k+(\alpha-1)z^{k+1})]_+ - \alpha\mu^k z^{k+1} \end{bmatrix} \right\|_\infty \leq \epsilon^k. \tag{4.5}$$

Since $\mathcal{M}^k_{\rho,\mu^k,\alpha}$ is strongly convex and piece-wise quadratic, it can be minimized exactly in finite time using a semi-smooth Newton method with exact linesearch [Sun, 1997, Theorem 3].

Semi-smooth Newton method For convenience, we omit the index k for the iterates of the Newton method arising at iteration k of ProxQP. A semi-smooth Newton method applied to $\mathcal{M}^k_{\rho,\mu^k,\alpha}$, and initialized at $(\hat{x}^{(0)},\hat{z}^{(0)}) = (x^k,z^k)$ involves finding at the l^{th} inner iteration $\mathrm{d}w \overset{\text{def}}{=} (\mathrm{d}x,\mathrm{d}z)$ satisfying:

$$\nabla^2\mathcal{M}^k_{\rho,\mu^k,\alpha}(\hat{x}^{(l)},\hat{z}^{(l)})\mathrm{d}w + \nabla\mathcal{M}^k_{\rho,\mu^k,\alpha}(\hat{x}^{(l)},\hat{z}^{(l)}) = 0, \tag{4.6}$$

where $\nabla\mathcal{M}^k_{\rho,\mu^k,\alpha}(\hat{x}^{(l)},\hat{z}^{(l)})$ and $\nabla^2\mathcal{M}^k_{\rho,\mu^k,\alpha}(\hat{x}^{(l)},\hat{z}^{(l)})$ stand respectively for one element of the generalized Jacobian and the generalized Hessian of $\mathcal{M}_{\rho,\mu^k,\alpha}$ [Facchinei and Pang, 2003, Section 7.1].

Once a semi-smooth Newton step $(\mathrm{d}x,\mathrm{d}z)$ has been obtained, the exact line-search procedure consists in finding the unique t^* such that:

$$t^* = \underset{t\geq 0}{\arg\min}\ \mathcal{M}^k_{\rho,\mu^k,\alpha}(\hat{x}^{(l)}+t\mathrm{d}x,\hat{z}^{(l)}+t\mathrm{d}z).$$

Similarly to [Hermans et al., 2021, Marchi., 2021, Bambade et al., 2022] the function $t \mapsto \mathcal{M}^k_{\rho,\mu^k,\alpha}(\hat{x}^{(l)}+t\mathrm{d}x,\hat{z}^{(l)}+t\mathrm{d}z)$ is continuous and piece-wise quadratic with a finite number of breaking points[2]. Hence, one can exactly compute t^*, an optimal solution to this one-dimensional problem. Finally, the semi-smooth Newton method initialized at $(\hat{x}^{(0)},\hat{z}^{(0)}) = (x^k,z^k)$ generates a sequence $(\hat{x}^{(l)},\hat{z}^{(l)})$ via the update rule:

$$\begin{aligned} \hat{x}^{(l+1)} &= \hat{x}^{(l)} + t^*\mathrm{d}x, \\ \hat{z}^{(l+1)} &= \hat{z}^{(l)} + t^*\mathrm{d}z. \end{aligned}$$

Solving linear systems In this section, we specify the linear systems induced by the minimization of $\mathcal{M}^k_{\rho,\mu^k,\alpha}$. We show that they are naturally better conditioned than those arising from Φ^k_{ρ,μ^k} since they naturally avoid quadratic matrix multiplications (such as $C^\top C$).

[1] Notice that with $\alpha = \frac{1}{2}$ we recover $\mathcal{M}_{\rho,\mu^k}$ as in (3.93).

[2] This feature is very specific to linear and quadratic types of problems and does help reducing the overall computational burden.

Equation (4.6) reads:

$$\begin{aligned}&\begin{bmatrix} H+\rho I+\frac{1}{\alpha\mu^k}C^\top(I-P^{(k,l)})C & \frac{\alpha-1}{\alpha}C^\top(I-P^{(k,l)}) \\ \frac{\alpha-1}{\alpha}(I-P^{(k,l)})C & \mu^k(\alpha-1)[-\frac{1}{\alpha}I-\frac{\alpha-1}{\alpha}P^{(k,l)}] \end{bmatrix}\begin{bmatrix}\mathrm{d}x\\ \mathrm{d}z\end{bmatrix}\\ &=-\begin{bmatrix}\nabla_x f(\hat{x}^{(l)})+\rho(x^{(l)}-x^k)+\frac{1}{\alpha\mu^k}C^\top[w_k^{(l)}-u]_+\\ \frac{\alpha-1}{\alpha}[w_k^{(l)}-u]_+ + \mu^k(1-\alpha)\hat{z}^{(l)}\end{bmatrix},\end{aligned} \tag{4.7}$$

where $P^{(k,l)}$ stands for one element of the generalized Jacobian of $[.]_+$ at $w_k^{(l)} \overset{\text{def}}{=} Cx^{(l)} + \mu^k z^k + \mu^k(\alpha-1)z^{(l)}$. $P^{(k,l)}$ consists of a diagonal matrix with entries

$$P_{jj}^{(k,l)} \overset{\text{def}}{=} \begin{cases} 1 & \text{if } C_j^\top \hat{x}^{(l)} + \mu^k z_j^k + \mu^k(\alpha-1)\hat{z}_j^{(l)} \leq u_j, \\ 0 & \text{otherwise.}\end{cases}$$

When $P_{jj}^{(k,l)} = 1$, it implies (see [Marchi., 2021, Section 3.2] for more details)

$$\mathrm{d}z_j = -(\hat{z}^{(l)})_j.$$

The linear system (4.7) can hence be equivalently formulated as:

$$\begin{cases} \begin{bmatrix} H+\rho I & C_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}^\top \\ C_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})} & -\mu^k I_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}\end{bmatrix}\begin{bmatrix}\mathrm{d}x\\ \mathrm{d}z_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}\end{bmatrix} = \\ -\begin{bmatrix} H\hat{x}^{(l)}+g+\rho(\hat{x}^{(l)}-x^k)+C_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}^\top \hat{z}_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}^{(l)} \\ [Cx^{(l)}+\mu^k z^k-\mu^k\hat{z}^{(l)}-u]_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}\end{bmatrix}, \\ \mathrm{d}z_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})^c} = -\hat{z}_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})^c}^{(l)}, \end{cases} \tag{4.8}$$

where:

$$J_k(x,z) \overset{\text{def}}{=} \left\{ j \in [1,n_i] | C_j^\top x + \mu^k z_j^k + \mu^k(\alpha-1)z_j - u_j > 0 \right\}$$

refers to the primal-dual active set at (x,z) for the current k^{th} sub-problem, and $J_k(x,z)^c$ stands for its complementary set (i.e., the set of inactive constraints). The linear system (4.8) is symmetric and non-singular. Thus it can be efficiently solved using LDLT factorization. Furthermore, one can notice it does not explicitly contain any square matrix multiplications. We will see it is not the case when working with Φ_{ρ,μ^k}^k.

At the l^{th} iteration, a semi-smooth Newton method applied to Φ_{ρ,μ^k}^k, and initialized at $\hat{x}^{(0)} = x^k$ involves finding $\mathrm{d}x$ satisfying $\nabla^2\Phi_{\rho,\mu^k}^k(\hat{x}^{(l)})\mathrm{d}x + \Phi_{\rho,\mu^k}^k(\hat{x}^{(l)}) = 0$. One ends-up solving [Hermans et al., 2021, Equation 3.4]:

$$\begin{bmatrix} H+\rho I & C_{I_k(\hat{x}^{(l)})}^\top \\ C_{I_k(\hat{x}^{(l)})} & -\mu^k I\end{bmatrix}\begin{bmatrix}\mathrm{d}x\\ \mathrm{d}z\end{bmatrix} = \begin{bmatrix}-\nabla_x\Phi_{\rho,\mu^k}^k(\hat{x}^{(l)})\\ 0\end{bmatrix},$$

where $I_k(\hat{x}^{(l)}) \overset{\text{def}}{=} \{i \in [1,n_i] \,|\, C_i\hat{x}^{(l)} - u_i + z_i^k\mu^k > 0\}$ refers to the active set of the current proximal sub-problem, and $C_{I_k(\hat{x}^{(l)})}$ corresponds to a reduced version of the matrix C containing the active rows indexed by $\mathcal{I}_k(\hat{x}^{(l)})$. The gradient on the right-hand side of this equation equals:

$$\nabla_x\Phi_{\rho,\mu^k}^k(\hat{x}^l) = H\hat{x}^{(l)} + g + \rho(\hat{x}^{(l)} - x^k) + C^\top\left[z^k + \tfrac{1}{\mu^k}(C\hat{x}^{(l)} - u)\right]_+.$$

For either low values of μ^k or ill-conditioning of $C_{I_k(\hat{x}^{(l)})}$, the evaluation of the gradient is numerically challenged due to the value of $\frac{1}{\mu^k}C_{I_k(\hat{x}^{(l)})}^\top C_{I_k(\hat{x}^{(l)})}$ (see, e.g., [Nocedal and Wright, 2006, Chapter 17.4] which gives examples illustrating ill-conditioning resulting from such structures). In the next section, we establish the convergence properties stated above.

4.2 Convergence properties

4.2.1 Organisation of the section

Section 4.2.3 establishes the global convergence of ProxQP (Theorem 3). It illustrates that the BCL algorithm [Conn et al., 1991] finds appropriate proximal steps to match a desired convergence speed for primal feasibility p^k. More precisely, BCL decreases μ^k until the inequality $p^k \leq \epsilon^k_{\text{bcl}}$ holds for all iteration starting from k. At this stage of the algorithm, ProxQP amounts to a primal-dual proximal point algorithm, and we rely on [Luque, 1984, Proposition 1.2] to ensure the global convergence of ProxQP. We study the simplified Algorithm 9, which assumes that at each iteration k of the algorithm $\rho^k = \mu^k$ compared to Algorithm 8, where ρ is fixed. Updating ρ is not desirable in practice since it would require a numerical factorization at each iteration.

Section	Content
Section 4.2.2	Notations and technical properties.
Section 4.2.3 Missing technical elements.	Proof of Lemma 2. Proof of Lemma 4.
Section 4.2.4	Proof of Theorem 3.
Section 4.2.5	Proof of Lemma 1.

Table 4.1: Organization of the section.

4.2.2 Further notations and technical properties

This section provides a few additional technical ingredients that are necessary for establishing a ProxQP convergence proof. We introduce a maximal monotone operator $T_{\mathcal{L}}^{-1}(v,t)$ (see for more details [Rockafellar, 1976b]), that encodes the set of solutions of the shifted problem (QP) by (t,v):

$$T_{\mathcal{L}}(x,z) \overset{\text{def}}{=} \{(v,t) | (v,-t) \in \partial\mathcal{L}(x,z)\},$$
$$T_{\mathcal{L}}^{-1}(v,t) \overset{\text{def}}{=} \underset{x\in\mathbb{R}^n}{\arg\min} \max_{z\in\mathbb{R}^m} \{\mathcal{L}(x,z) - x^\top v + z^\top t\},$$

where $\partial\mathcal{L}$ refers to

$$\partial\mathcal{L}(x,z) \overset{\text{def}}{=} \begin{pmatrix} \nabla f(x) + C^\top z \\ Cx - \partial I_u^\star(z) \end{pmatrix}, \tag{4.9}$$

with I_u the indicator function of the constraint set $Cx \leq u$. The Lagrangian function and $T_{\mathcal{L}}(x,z)$ encodes the saddle sub-differential for (QP) [Ryu and Boyd, 2016, Section 4.3]. $T_{\mathcal{L}}$ and $T_{\mathcal{L}}^{-1}$ are generally "point to set" operators. From $T_{\mathcal{L}}$ and for $\mu > 0$ we also introduce the resolvent of $T_{\mathcal{L}}$ and $T_{\mathcal{L}}^{-1}$

- the resolvent of $T_{\mathcal{L}}$ [Rockafellar, 1976b]:

$$P_\mu \overset{\text{def}}{=} (I + \frac{1}{\mu} T_{\mathcal{L}})^{-1}, \tag{4.10}$$

- the resolvent of $T_{\mathcal{L}}^{-1}$ [Rockafellar, 1976b]:

$$Q_\mu \stackrel{\text{def}}{=} (I + \mu T_{\mathcal{L}}^{-1})^{-1} = I - P_\mu. \tag{4.11}$$

More precisely, when $T_{\mathcal{L}}^{-1}(0,0)$ is non-empty, it encodes the set of solutions to (QP) . When (QP) is feasible, the KKT conditions for (QP) form a polyhedral variational inequality [Dontchev et al., 2009a, Section 3D]. Hence, $\exists a \geq 0, \tau > 0$, such that $\forall (u,v) \in \mathbb{R}^{n+m}$:

$$\begin{array}{c} \|(u,v)\| \leq \tau \\ \Longrightarrow \\ \text{dist}_{T_{\mathcal{L}}^{-1}(0,0)}(x,z) \leq a\|(u,v)\|, \forall (x,z) \in T_{\mathcal{L}}^{-1}(u,v) \end{array} \tag{4.12}$$

a and τ are constants describing the geometry of the QP. Rockafellar [Rockafellar, 1970, Thm. 2] has established that it holds as soon as $T_{\mathcal{L}}(0,0)^{-1} \neq \emptyset$. That is, $\forall \mu > 0$, $\forall (x,z) \in \mathbb{R}^n \times \mathbb{R}^m$:

$$\begin{array}{c} \mu\|Q_\mu(x,z)\| \leq \tau \\ \Longrightarrow \\ \text{dist}_{T_{\mathcal{L}}(0,0)^{-1}}(P_\mu(x,z)) \leq \frac{a\mu}{\sqrt{1+a^2\mu^2}} \text{dist}_{T_{\mathcal{L}}(0,0)^{-1}}(x,z). \end{array} \tag{4.13}$$

4.2.3 Convergence of ProxQP

The main convergence result for ProxQP is provided by Theorem 3. To prove this theorem (in Section 4.2.4), we proceed with the following strategy. This strategy relies on a slightly different formulation of ProxQP provided by Algorithm 9 (it is necessary for the proof to consider the conceptual infinite sequence of iterates generated by the algorithm).

- First, Lemma 2 shows that the "if condition" of ProxQP (see Algorithm 9) is satisfied at infinitely many iterations.
- Second, Lemma 4 shows that the "else condition" of ProxQP (see Algorithm 9) is entered only for a finite number of iterations.
- It follows from the second point that ProxQP corresponds to a primal-dual proximal point method, and hence inherits all its nice convergence properties. This is formally stated in Section 4.2.4.
- Finally, one can conveniently deduce from this construction that ProxQP asymptotically corresponds to a fixed-step proximal method of multiplier (PMM, i.e., there exists $K > 0$ such that for all $k \geq K$, μ^k is constant). This is formally stated in Section 4.2.5.

We enter the "if" of Algorithm 9 an infinite number of times

Lemma 2. *Under Assumption 1 and Assumption 2, $\forall k \in \mathbb{N}, \exists K \geq 0$ such that the condition $\|[Cx^{k+K+1} - u]_+\|_\infty \leq \epsilon_{bcl}^{k+K}$ of Algorithm 9 is satisfied.*

Proof. The following proof proceeds by contradiction. That is, we show that there cannot exist a $k \geq 0$ such that $\forall K > 0$ it holds that $p^{k+K} \stackrel{\text{def}}{=} \|[Cx^{k+K+1} - u]_+\|_\infty > \epsilon_{\text{bcl}}^{k+K}$ (as defined in (4.2)). For convenience, and without loss of generality, we assume $k = 0$.

By contradiction hypothesis, we thus have that $p^K > \epsilon_{\text{bcl}}^K$ holds $\forall K \geq 0$. Therefore, $\mu^K = \mu^0(\mu_f)^K$ and $\epsilon_{\text{bcl}}^K = \epsilon_{\text{bcl}}^0(\mu^0)^{\alpha_{bcl}}(\mu_f)^{\alpha_{\text{bcl}}K}$ and hence $\mu^K \to 0$ and $\epsilon_{\text{bcl}}^K \to 0$ as $K \to \infty$, since $\mu_f \in (0,1)$ and $\alpha_{\text{bcl}} \in (0, 1/2)$. For establishing the desired statement, we show that under the contradiction hypothesis, one has

$$\epsilon_{\text{bcl}}^K < \|[Cx^{K+1} - u]_+\|_\infty \leq A\sqrt{\mu^K}, \tag{4.14}$$

for all $K \geq 0$, thereby reaching a contradiction. Indeed, with $\alpha_{\text{bcl}} \in (0, 1/2)$, the term $\sqrt{\mu^K}$ goes faster to zero (as a function of K) than ϵ^K_{bcl}. More precisely, the hyperparameter choices within ProxQP ensure that $\exists K_0 \geq 0$ such that $\forall K \geq K_0$, $\epsilon^K_{\text{bcl}} > A\sqrt{\mu^K}$.

For proving our claim, we, therefore, focus on proving (4.14) in the following lines.

First, at iteration $K+1$, the pair $(x^{K+1}, \hat{z}^{K+1})$ is an approximate solution to (4.4). In other words, by defining intermediary variables, we have:

$$\begin{aligned} u^K &\overset{\text{def}}{=} \nabla f(x^{K+1}) + C^\top \hat{z}^{K+1} + \mu^K(x^{K+1} - x^K), \\ v^K &\overset{\text{def}}{=} [Cx^{K+1} - u + \mu^K(z^0 + (\alpha - 1)\hat{z}^{K+1}]_+ - \hat{z}^{K+1}\alpha\mu^K, \end{aligned}$$

with $\|(u^K, v^K)\|_\infty \leq \epsilon^K$. Following [Marchi., 2021, Section 3.1], one can equivalently look at $(x^{k+1}, \hat{z}^{k+1})$ as the unique minimum of the strongly convex function $\hat{\mathcal{M}}^K_{\rho,\mu^K,\alpha}$:

$$\begin{aligned} \hat{\mathcal{M}}^K_{\rho,\mu^K,\alpha}(x, z) \overset{\text{def}}{=} f(x) &+ \tfrac{\mu^K}{2}\|x - (x^K + \tfrac{1}{\mu^K}u^K)\|_2^2 \\ &+ \tfrac{1}{2\alpha\mu^K}\|[Cx - u + \mu^K(z^K + (\alpha - 1)z)]_+\|_2^2 \\ &+ \tfrac{(1-\alpha)\mu^K}{2}\|z + \tfrac{1}{\alpha\mu^k}v^K\|_2^2. \end{aligned}$$

Therefore, for any $x \in \mathbb{R}^n$ satisfying $Cx - u \leq 0$ (such a feasible x exists by Assumption Assumption 1), we get:

$$\hat{\mathcal{M}}^K_{\rho,\mu^K,\alpha}(x^{K+1}, \hat{z}^{K+1}) \leq \hat{\mathcal{M}}^K_{\rho,\mu^K,\alpha}(x, 0).$$

As $\{x^K\}$ is bounded by assumption 2, then $\exists M > 0$ independent of K such that

$$\hat{\mathcal{M}}^K_{\rho,\mu^K,\alpha}(x^{K+1}, \hat{z}^{K+1}) \leq f(x) + M\mu^K, \tag{4.15}$$

where we used the facts that for a feasible point x, $[Cx - u + z^0\mu^K]_+ \leq [z^0]_+\mu^K$ (as $Cx - u \leq 0$), and that $\|(u^K, v^K)\|_\infty \leq \epsilon^K$ implies that $\|u^K\|_\infty/\mu^K \leq \epsilon^0$ and that $\|v^K\|_\infty/\mu^K$.

It follows from the definition of $\hat{\mathcal{M}}^K_{\rho,\mu^K,\alpha}$ (and nonnegativity of the $\|.\|_2$ terms) that:

- we can bound

  $$\hat{\mathcal{M}}^K_{\rho,\mu^K,\alpha}(x^{K+1}, \hat{z}^{K+1}) \geq f(x^{K+1}) + \tfrac{1}{2\alpha\mu^k}\|[Cx^{K+1} - u + \mu^k(z^0 + (\alpha - 1)\hat{z}^{k+1})]_+\|_2^2$$

  which, using (4.15), can be used to reach

  $$\|[Cx^{K+1} - u + \mu^K(z^0 + (\alpha - 1)\hat{z}^{K+1})]_+\|_2 \leq \sqrt{2\mu^K\alpha(f(x) - f(x^{K+1}) + M\mu^K)}. \tag{4.16}$$

  From (4.16), since $\{x^K\}_K$ is bounded and f is a quadratic, we deduce that there exists some $M_1 > 0$ independent of K such that

  $$\|[Cx^{K+1} - u + \mu^K(z^0 + (\alpha - 1)\hat{z}^{K+1})]_+\|_2 \leq \sqrt{\mu^K}M_1. \tag{4.17}$$

- We can bound

  $$\hat{\mathcal{M}}^K_{\rho,\mu^K,\alpha}(x^{K+1}, \hat{z}^{K+1}) \geq f(x^{K+1}) + \tfrac{(1-\alpha)\mu^K}{2}\|z + \tfrac{1}{\alpha\mu^k}v^K\|_2^2.$$

  which, using (4.15), can be used to reach

  $$\sqrt{\mu^K}\|\hat{z}^{K+1}\|_2 \leq \tfrac{\sqrt{2}}{\sqrt{1-\alpha}}(\sqrt{f(x) - f(x^{K+1}) + M\mu^K} + \tfrac{\sqrt{1-\alpha}}{\sqrt{\alpha}}\|v^K\|_2).$$

  It shows that $\sqrt{\mu^K}\|\hat{z}^{K+1}\|_2$ is bounded since $\{x^K\}$ is bounded and f is a quadratic function.

We are now in a position to guarantee that $\forall K > 0$:

$$\|[Cx^{K+1} - u]_+\|_\infty \leq A\sqrt{\mu^K}. \tag{4.18}$$

for some $A > 0$. Indeed, it holds component-wise

$$[Cx^{K+1} - u]_+ - \mu^K|(z^0 + (\alpha - 1)\hat{z}^{K+1})| \leq [Cx^{K+1} - u + \mu^K(z^0 + (\alpha - 1)\hat{z}^{K+1})]_+, \tag{4.19}$$

where $|.|$ is the absolute value taken component-wise. We can deduce from (4.19) that

$$\begin{aligned}\|[Cx^{K+1} - u]_+\|_2 \leq \|[Cx^{K+1} - u + \mu^K(z^0 + (\alpha - 1)\hat{z}^{K+1})]_+\|_2 \\ + \sqrt{\mu^K}\|\sqrt{\mu^K}(z^0 + (\alpha - 1)\hat{z}^{K+1})\|_2\end{aligned} \tag{4.20}$$

Hence, from (4.20), using (4.17) and the fact that $\{\sqrt{\mu^K}(z^0 + (\alpha - 1)\hat{z}^{K+1})\}_K$ is bounded by some $M_2 > 0$ (since $\{\sqrt{\mu^K}\hat{z}^{K+1}\}$ is bounded and $\{\sqrt{\mu^K}\}_K$ converges to zero) we get

$$\|[Cx^{K+1} - u]_+\|_2 \leq \underbrace{(M_1 + M_2)}_{\stackrel{\text{def}}{=} A > 0}\sqrt{\mu^K}.$$

To reach the desired claim, we use (4.18) together with the contradiction hypothesis that $\epsilon_{\text{bcl}}^K < \|[Cx^{K+1} - u]_+\|_\infty$ which allows reaching the contradiction that $\forall K > 0$:

$$\epsilon_{\text{bcl}}^K \leq A\sqrt{\mu^K}.$$

□

We enter the "else" of Algorithm 9 a finite number of times

As previewed in Section 4.2.3, this section establishes that the "else" condition of Algorithm 9 is only entered a finite number of times. The proof can be summarized as follows.

- If the "else" condition is never satisfied, then the result directly holds.
- Otherwise, since Lemma Lemma 2 ensures the "if" condition is entered an infinite number of times, we consider the following setting. Without loss of generality, we can assume that the "else" condition is entered at iteration k for the Kth time and that it is followed by a "if" condition. It implies that

  $$\mu^k = \mu^0(\mu_f)^K,\ \epsilon_{\text{bcl}}^k = \epsilon_{\text{bcl}}^0(\mu^k)^{\alpha_{\text{bcl}}}.$$

  We denote by $N \geq 2$ the number of consecutive iterations from k for which we enter the "if" condition, meaning that

  $$\mu^{k+N} = \mu^0(\mu_f)^K,\ \epsilon_{\text{bcl}}^{k+N-1} = \epsilon_{\text{bcl}}^0(\mu^k)^{\alpha_{\text{bcl}}+(N-1)\beta_{\text{bcl}}}.$$

  In this situation, one can establish that for QPs, there exists constants $M > 0$ and $b \geq 1$ (describing the geometry of the problem at hand) such that the primal infeasibility can be upper-bounded for $N \geq 2$ by

  $$\|[Cx^{k+N} - u]_+\|_\infty \leq M\left(b\mu^k\right)^{N-1}, \tag{4.21}$$

  For entering the "else" condition in this scenario at the $(k+N)$th step, it is therefore necessary to be in the situation where

  $$\epsilon_{\text{bcl}}^{k+N-1} < \|[Cx^{k+N} - u]_+\|_\infty \leq M\left(b\mu^k\right)^{N-1},$$

Substituting the expressions for $\epsilon_{\text{bcl}}^{k+N-1}$ and μ^k, we arrive to the following necessary condition for entering the "else" at iteration $(k+N)$:

$$\epsilon_{\text{bcl}}^0(\mu^0(\mu_f)^K)^{\alpha_{\text{bcl}}+(N-1)\beta_{\text{bcl}}} \leq M(b\mu^0(\mu_f)^K)^{N-1}.$$

On the contrary, it is therefore sufficient to satisfy

$$M(b\mu^0(\mu_f)^K)^{N-1} \leq \epsilon_{\text{bcl}}^0(\mu^0(\mu_f)^K)^{\alpha_{\text{bcl}}+(N-1)\beta_{\text{bcl}}}$$

for not entering the "else". Given that $\alpha_{\text{bcl}} \in (0, 1/2)$, $\beta_{\text{bcl}} \in (0,1)$ with $\beta_{\text{bcl}} + \alpha_{\text{bcl}} < 1$, and $\mu_f \in (0,1)$ it can be established that this condition is met for all $N \geq 2$ as soon as

$$\max(\lceil \frac{\log(\frac{b(\mu^0)^{1-\alpha_{\text{bcl}}-\beta_{\text{bcl}}}M}{\epsilon_{\text{bcl}}^0})}{-\log(\mu_f)(1-\alpha_{\text{bcl}}-\beta_{\text{bcl}})} \rceil, \lceil \frac{\log(b(\mu^0)^{1-\beta_{\text{bcl}}})}{-\log(\mu_f)(1-\beta_{\text{bcl}})} \rceil, K_\tau) \leq K, \tag{4.22}$$

for some $K_\tau \in \mathbb{N}$ (for the pair (a, τ) from (4.13), which describes the geometry of the QP at hand). It shows that the "else" condition is entered only a finite number of times, thereby arriving to the desired conclusion.

Our main target is thus to establish that (4.22) holds, as the desired conclusion nicely follows from this fact. For reaching this target, Lemma 3 provides an intermediary result which ensures that for K sufficiently large (i.e., larger than K_τ mentioned in (4.22)), the technical property (4.13) holds for all the iterates of ProxQP.

Lemma 3. *Under Assumption 1, Assumption 2, and if $\mu^k \to 0$ (i.e., we enter an infinite number of times the "else" condition), for any $\tau > 0$ $\exists K_\tau \in \mathbb{N}$ such that for all $\forall k \geq K_\tau$, $\mu^k \|Q_{\mu^k}(x^k, z^k)\|_2 \leq \tau$.*

Proof. In this setting $\epsilon^k = \epsilon^0 \mu^k$. Then, by definition of Q_{μ^k} (4.11)

$$\|Q_{\mu^k}(x^k, z^k)\|_2 = \|(x^k, z^k) - P_{\mu^k}(x^k, z^k)\|_2 \leq \left\| \begin{pmatrix} x^k - x^{k+1} \\ z^k - z^{k+1} \end{pmatrix} \right\|_2 + \|(x^{k+1}, z^{k+1}) - P_{\mu^k}(x^k, z^k)\|_2. \tag{4.23}$$

$P_{\mu^k}(x^k, z^k)$ corresponds to the solution pair to (3.87). Since $\mathcal{M}_{\mu^k,\mu^k,\alpha}$ is strongly convex with parameter at least $(1-\alpha)\mu^k$, it follows that

$$\begin{aligned} &(1-\alpha)\mu^k \|(x^{k+1}, z^{k+1}) - P_{\mu^k}(x^k, z^k)\|_2 \\ &\leq \|\nabla \mathcal{M}_{\mu^k,\mu^k,\alpha}(x^{k+1}, z^{k+1}) - \nabla \underbrace{\mathcal{M}_{\mu^k,\mu^k,\alpha}(P_{\mu^k}(x^k, z^k))}_{=0} \|_2 \\ &\leq \max(1, \tfrac{1-\alpha}{\alpha})\epsilon^k, \end{aligned}$$

where the last inequality comes from the fact that

$$\nabla_z \mathcal{M}_{\mu^k,\mu^k,\alpha}(x^{k+1}, z^{k+1}) = \tfrac{1-\alpha}{\alpha}([Cx^{k+1} - u + \mu^k(z^k + (\alpha-1)z^{k+1})]_+ - \alpha\mu^k z^{k+1}),$$

and (x^{k+1}, z^{k+1}) satisfies (4.5) at accuracy ϵ^k. Hence, it holds

$$\|(x^{k+1}, z^{k+1}) - P_{\mu^k}(x^k, z^k)\|_2 \leq \underbrace{\max(\tfrac{1}{1-\alpha}, \tfrac{1}{\alpha})}_{\text{def}= w} \epsilon^k/\mu^k. \tag{4.24}$$

As $\epsilon^k = \epsilon^0\mu^k$, combining (4.23) with (4.24) allows obtaining the upper bound

$$\begin{aligned} \mu^k \|Q_{\mu^k}(x^k, z^k)\|_2 &\leq \mu^k \left\| \begin{pmatrix} x^k - x^{k+1} \\ z^k - z^{k+1} \end{pmatrix} \right\|_2 + w\epsilon^k \\ &\leq \mu^k \left\| \begin{pmatrix} x^k - x^{k+1} \\ z^k - z^{k+1} \end{pmatrix} \right\|_2 + \mu^k w\epsilon^0. \end{aligned} \tag{4.25}$$

Since $\{(x_k, z_k)\}$ is bounded (see Assumption 1), and since $\mu^k \to 0$ (exponentially in the number of times we enter the "else" condition), we conclude from (4.25) that $\forall \tau > 0$, there $\exists K_\tau \in \mathbb{N}$, such that $\forall k \geq K_\tau$, $\mu^k \|Q_{\mu^k}(x^k, z^k)\|_2 \leq \tau$. □

Finally, Lemma Lemma 4 establishes our main result. More precisely, we make use of Lemma 3 to establish first the inequality (4.21) and then (4.22).

Lemma 4. *Under Assumption 1 and Assumption 2, $\exists N_{\max} \in \mathbb{N}$ such that for all $k \geq N_{\max}$ the condition $p^{k+1} \leq \epsilon_{bcl}^k$ of Algorithm 9 is satisfied (i.e., we enter the "if condition").*

Proof. We split the proof in several cases.

- If we enter the "else condition" a finite number of times, the result holds.
- If we enter the "else condition" an infinite number of times, Lemma Lemma 2 ensures that each time we enter the "else", there is a subsequent iteration for which we enter the "if".

  Therefore, we can without loss of generality assume that we enter the "else" at iteration k for the the Kth time, and that we enter the "if" at iteration $k+1$.

  In this setup (we enter an infinite number of times the "else", $\mu^k \to 0$ and Lemma 3 applies. Therefore, we can also pick without loss of generality $k \geq K_\tau$ where $K_\tau \in \mathbb{N}$ is such that (4.13) holds (for a specific (a, τ) describing the geometry of the QP at hand).

  In what follows, we consider this situation and show that it cannot happen (i.e., "else" is entered a finite number of times).

Since (QP) is feasible by assumption, it follows that $T_{\mathcal{L}}^{-1}(0,0) \neq \emptyset$. For $l \in [0, N]$ we denote by $(\bar{x}^{k+l}, \bar{z}^{k+l})$ the projection of (x^{k+l}, z^{k+l}) onto the set of solutions $T_{\mathcal{L}}^{-1}(0,0)$. As $\bar{x}^{k+N}$ is feasible, it follows that

$$\|[Cx^{k+N} - u]_+\|_2 = \|[Cx^{k+N} - u]_+ - [C\bar{x}^{k+N} - u]_+\|_2. \tag{4.26}$$

As $[.]_+$ is a non-expansive operator (as any projection operator), it follows that (4.26) can be upper bounded by

$$\begin{aligned} &\|[Cx^{k+N} - u]_+\|_2 \leq \|C(x^{k+N} - \bar{x}^{k+N})\|_2 \\ &\leq \|C\|_2 \|x^{k+N} - \bar{x}^{k+N}\|_2 \\ &\leq \|C\|_2 \underbrace{\left\| \begin{pmatrix} x^{k+N} - \bar{x}^{k+N} \\ z^{k+N} - \bar{z}^{k+N} \end{pmatrix} \right\|_2}_{\stackrel{\text{def}}{=} \text{dist}_{T_{\mathcal{L}}(0,0)^{-1}}(x^{k+N}, z^{k+N})}. \end{aligned} \tag{4.27}$$

The following lines therefore focus on upper bounding the last term in different ways. First,

$$\begin{aligned} &\left\| \begin{pmatrix} x^{k+N} - \bar{x}^{k+N} \\ z^{k+N} - \bar{z}^{k+N} \end{pmatrix} \right\|_2 \\ &\leq \left\| \begin{pmatrix} x^{k+N} \\ z^{k+N} \end{pmatrix} - \overline{P_{\mu^k}}(x^{k+N-1}, z^{k+N-1}) \right\|_2, \end{aligned} \tag{4.28}$$

which follows from the definition of $(\bar{x}^{k+N}, \bar{z}^{k+N})$ as the projection of (x^{k+N}, z^{k+N}) and $\overline{P_{\mu^k}}(x^{k+N-1}, z^{k+N-1})$ corresponds to the projection of $P_{\mu^k}(x^{k+N-1}, z^{k+N-1})$ onto $T_{\mathcal{L}}^{-1}(0,0)$. which can be further upper bounded via a triangle inequality:

$$\begin{aligned} &\| \begin{pmatrix} x^{k+N} \\ z^{k+N} \end{pmatrix} - \overline{P_{\mu^k}}(x^{k+N-1}, z^{k+N-1})\|_2 \\ &\leq \| \begin{pmatrix} x^{k+N} \\ z^{k+N} \end{pmatrix} - P_{\mu^k}(x^{k+N-1}, z^{k+N-1})\|_2 \\ &+ \|P_{\mu^k}(x^{k+N-1}, z^{k+N-1}) - \overline{P_{\mu^k}}(x^{k+N-1}, z^{k+N-1})\|_2. \end{aligned} \tag{4.29}$$

Hence we also have:

$$\begin{aligned}&\left\|\begin{pmatrix}x^{k+N}-\bar{x}^{k+N}\\ z^{k+N}-\bar{z}^{k+N}\end{pmatrix}\right\|_2\\ &\leq \|\begin{pmatrix}x^{k+N}\\ z^{k+N}\end{pmatrix}-P_{\mu^k}(x^{k+N-1},z^{k+N-1})\|_2\\ &+\|P_{\mu^k}(x^{k+N-1},z^{k+N-1})-\overline{P_{\mu^k}}(x^{k+N-1},z^{k+N-1})\|_2.\end{aligned}\tag{4.30}$$

Now, using $k\geq K_\tau$ from Lemma 3 (for a specific (a,τ) such that (4.13) holds for the QP at hand), then using the property (4.13) provides

$$\begin{aligned}&\|P_{\mu^k}(x^{k+N-1},z^{k+N-1})-\overline{P_{\mu^k}}(x^{k+N},z^{k+N})\|_2\\ &\leq \frac{a\mu^k}{\sqrt{1+a^2(\mu^k)^2}}\mathrm{dist}_{T_{\mathcal{L}}(0,0)^{-1}}(x^{k+N-1},z^{k+N-1})\\ &\leq a\mu^k\left\|\begin{pmatrix}x^{k+N-1}-\bar{x}^{k+N-1}\\ z^{k+N-1}-\bar{z}^{k+N-1}\end{pmatrix}\right\|_2.\end{aligned}\tag{4.31}$$

Hence incorporating (4.31) in (4.30) leads to

$$\left\|\begin{pmatrix}x^{k+N}-\bar{x}^{k+N}\\ z^{k+N}-\bar{z}^{k+N}\end{pmatrix}\right\|_2\leq\frac{w\epsilon^{k+N-1}}{\mu^k}+a\mu^k\left\|\begin{pmatrix}x^{k+N-1}-\bar{x}^{k+N-1}\\ z^{k+N-1}-\bar{z}^{k+N-1}\end{pmatrix}\right\|_2\tag{4.32}$$

where we also used the fact that (x^{k+N},z^{k+N}) satisfies (4.5) at accuracy ϵ^{k+N-1} similarly to (4.24). Iteratively expanding the upper bound (4.32) $N-1$ times leads to

$$\left\|\begin{pmatrix}x^{k+N}-\bar{x}^{k+N}\\ z^{k+N}-\bar{z}^{k+N}\end{pmatrix}\right\|_2\leq w\sum_{l=0}^{N-1}\frac{\epsilon^{k+N-1-l}}{\mu^k}(a\mu^k)^l+(a\mu^k)^N\left\|\begin{pmatrix}x^k-\bar{x}^k\\ z^k-\bar{z}^k\end{pmatrix}\right\|_2.\tag{4.33}$$

Because $\epsilon^{k+N-1-l}=\epsilon^0(\mu^k)^{N-l}$, we deduce from (4.33) that

$$\begin{aligned}&\left\|\begin{pmatrix}x^{k+N}-\bar{x}^{k+N}\\ z^{k+N}-\bar{z}^{k+N}\end{pmatrix}\right\|_2\\ &\leq w(\mu^k)^{N-1}\sum_{l=0}^{N-1}(a)^l+(a\mu^k)^N\left\|\begin{pmatrix}x^k-\bar{x}^k\\ z^k-\bar{z}^k\end{pmatrix}\right\|_2\\ &\leq w(\mu^k)^{N-1}\sum_{l=0}^{N-1}(a)^l+B(a\mu^k)^N,\end{aligned}\tag{4.34}$$

for some $B>0$, since $\{(x^k,z^k)\}_k$ is bounded (by Assumption 2). Finally, incorporating (4.34) into (4.27) provides the bound

$$\|[Cx^{k+N}-u]_+\|_\infty\leq e\|C\|_2\left(w(\mu^k)^{N-1}\sum_{l=0}^{N-1}(a)^l+B(a\mu^k)^N\right),\tag{4.35}$$

for some constant $e>0$ (from the norm equivalences between $\|.\|_\infty$ and $\|.\|_2$ in $\mathbb{R}^m$). Depending on the value of a, we distinguish three different cases to show the inequality (4.21):

- if $a\in(0,1)$, then, since $\{\mu^k\}_k$ is bounded by 1 (since $\mu^0\in(0,1)$ and $\mu_f\in(0,1)$), we can upper bound the geometric sum in (4.35) with

$$\|[Cx^{k+N}-u]_+\|_\infty\leq\underbrace{e\|C\|_2\left(w\tfrac{1}{1-a}+B\right)}_{=M}(\mu^k)^{N-1},$$

  and hence (4.21) holds with $b=1$.

- if $a > 1$, then we can upper bound the geometric sum with (using again that $\{\mu^k\}_k$ is bounded by 1)

$$\|[Cx^{k+N} - u]_+\|_\infty \leq \underbrace{e\|C\|_2 \left(w\tfrac{1}{a-1} + aB\right)}_{=M}(a\mu^k)^{N-1},$$

  and hence (4.21) holds with $b = a$.

- if $a = 1$, then we can upper bound the geometric sum with (using again that $\{\mu^k\}_k$ is bounded by 1)

$$\begin{aligned}\|[Cx^{k+N} - u]_+\|_\infty &\leq e\|C\|_2 \left(wN + B\right)(\mu^k)^{N-1} \\ &\leq \underbrace{4e\|C\|_2 \max(w, B)}_{=M}(2\mu^k)^{N-1},\end{aligned}$$

  where we used that $\forall N \in \mathbb{N}, N \leq 2^N$. Hence (4.21) holds with $b = 2$.

Thus in any case, there exists some $M > 0$ and $b \geq 1$, such that (4.21) holds. To conclude, in order to enter at least $N \geq 2$ consecutive times in the "if" condition it is sufficient to satisfy

$$M(b\mu^0(\mu_f)^K)^{N-1} \leq \epsilon_{\text{bcl}}^0(\mu^0(\mu_f)^K)^{\alpha_{\text{bcl}}+(N-1)\beta_{\text{bcl}}}, \tag{4.36}$$

which is equivalent to

$$\begin{aligned}&\log(\tfrac{b(\mu^0)^{1-\alpha_{\text{bcl}}-\beta_{\text{bcl}}}M}{\epsilon_{\text{bcl}}^0}) + (N-2)\log(b(\mu^0)^{1-\beta_{\text{bcl}}}) \\ &\leq -\log(\mu_f)(1-\alpha_{\text{bcl}}-\beta_{\text{bcl}})K \\ &\quad -\log(\mu_f)(1-\beta_{\text{bcl}})(N-2)K.\end{aligned}$$

Since $\mu_f \in (0,1)$, $\alpha_{\text{bcl}} \in (0, 1/2)$, $\beta_{\text{bcl}} \in (0,1)$ with $\alpha_{\text{bcl}} + \beta_{\text{bcl}} < 1$, then (4.36) holds for any $N \geq 2$ as soon as

$$\max(\lceil \tfrac{\log(\frac{b(\mu^0)^{1-\alpha_{\text{bcl}}-\beta_{\text{bcl}}}M}{\epsilon_{\text{bcl}}^0})}{-\log(\mu_f)(1-\alpha_{\text{bcl}}-\beta_{\text{bcl}})} \rceil, \lceil \tfrac{\log(b(\mu^0)^{1-\beta_{\text{bcl}}})}{-\log(\mu_f)(1-\beta_{\text{bcl}})} \rceil, K_\tau) \leq K,$$

which establishes (4.22). In other words, we enter the "else condition" only a finite number of times (which is bounded above by the value of the previous max). □

4.2.4 Proof of Theorem 3

Theorem 3 (Convergence of ProxQP)**.** *Under Assumption 1 (existence of a solution to QP), Assumption 2 (bounded iterates), and with all parameters set as required in the inputs of Algorithm 9, the iterates $\{(x^k, z^k)\}$ of ProxQP converge to a solution $(x^\star, z^\star)$ of (QP).*

Proof. The proof consists in showing that ProxQP asymptotically corresponds to a proximal point algorithm (PPA). Then, we leverage the classical global convergence guarantees for PPA, which is automatically inherited by ProxQP. To show this link between ProxQP and PPA, (i) we first recall PPA global convergence properties, and then (ii) show that in a finite number of iterations ProxQP is a fixed-step PPA.

When (QP) is feasible, the KKT conditions $0 \in T_{\mathcal{L}}(x, z)$ forms a polyhedral variational inequality [Dontchev et al., 2009a, Section 3D]. In such settings, if we consider a sequence $\{\mu^k\}_k$, bounded below by some $\mu^\infty > 0$, and $\{\eta^k\}_k$ some summable sequence, then the inexact proximal point iteration

$$(x^{k+1}, z^{k+1}) \approx_{\eta^k} P_{\mu^k}(x^k, z^k), \tag{4.37}$$

with $\approx_{\eta^k}$ corresponding to

$$\|(x^{k+1}, z^{k+1}) - P_{\mu^k}(x^k, z^k)\|_2 \leq \eta^k, \tag{4.38}$$

is guaranteed to converge globally to some $(x^\star, z^\star) \in T_{\mathcal{L}}^{-1}(0,0)$ [Luque, 1984, Proposition 1.2].

Depending on the value of $k_{\max}$, we distinguish two different cases to show that ProxQP is asymptotically equivalent to PPA. As we will see, it amounts to show that ProxQP always enters the "if" condition in a finite number of iterations.

- If $k_{\max} < +\infty$, then after $k_{\max}$ number of iterations, ProxQP iterates always enter the "if" condition.
- Otherwise, under Assumption (2) and Assumption (1), Lemma (4) ensures that there exists some $N_{\max} \in \mathbb{N}$, after which ProxQP enter only the "if" condition.

Hence, for $k \geq N \overset{\text{def}}{=} \min(k_{\max}, N_{\max})$, ProxQP always accepts the iterates $(x^{k+1}, \hat{z}^{k+1})$ from (4.4). Furthermore, for $l \geq 0$

$$\begin{aligned}\mu^{N+l} &= \mu^N, \\ \epsilon^{N+l} &= \epsilon^k (\mu^N)^l.\end{aligned}$$

Hence the iterates of Algorithm 9 amount to a fixed step PPA, namely for $k \geq N$

$$(x^{k+1}, \hat{z}^{k+1}) \approx_{e\epsilon^k} P_{\mu^N}(x^k, z^k),$$

for some constant $e > 0$ which can be provided following the same strategy as detailed in (4.24). As $\{\epsilon^k\}_k$ is summable for $k \geq N$, it concludes our claim. □

4.2.5 Proof of Lemma Lemma 1

Lemma 1. *Under Assumption 1 (existence of solution to (QP)) and Assumption 2 (bounded iterates), and with all parameters set as required in the inputs of Algorithm 9, $\exists K \in \mathbb{N}$, and $\mu \in \mathbb{R}$ such that $\forall k \geq K$ $\mu^k = \mu$. Hence, $\forall k \geq 0, p^{K+k} \leq \epsilon_{bcl}^K (\mu^K)^{\beta_{bcl} k}$.*

Proof. By Lemma 4, $\exists \mu \in \mathbb{R}$, $\exists K \in \mathbb{N}$ such that for all $k \geq K$, it holds that $p^k < \epsilon_{\text{bcl}}^k$ and $\mu^k = \mu$ is constant. □

4.3 Software Implementation

In this section, we list a range of common QP solvers, along with their underlying optimization strategy, and then detail the software implementation of ProxQP within the ProxSuite library.

4.3.1 Common QP solvers

We provide below a summary of different start-of-the-art solvers implementing iterative optimization methods previously described in Chapter 3, and summarized in Table 4.2, with a short description of their practical features for embedded optimization.

Active-set solvers Popular active set-based convex QP solvers include the open-source qpOASES [Ferreau et al., 2014], quadprog [Goldfarb and Idnani, 1983], DAQP [Arnström et al., 2022], and the QPA module in the open source software GALAHAD [Gould et al., 2017].

Interior-point solvers Standard solvers using an interior-point strategy are commercial solvers GUROBI [Optimization, 2020] and MOSEK [Mosek, 2022], closed-source BPMPD [Mészáros, 1999], and open-source solvers OOQP [Gertz and Wright, 2003], HPIPM [Frison and Diehl, 2020], ECOS [Domahidi et al., 2013], CVXOPT [Andersen et al., 2013] and QPSWIFT [Pandala et al., 2019].

Augmented Lagrangian-based solvers Common solvers based on augmented Lagrangians for solving QPs include OSQP [Stellato et al., 2020] and SCS [O'Donoghue et al., 2016] (via an alternating direction method of multipliers) and QPALM [Hermans et al., 2021].

Table 4.2: Comparison of different optimization methods and software libraries for solving QPs.

Method	Solver	Backend	Warm-starting	Hot-starting	Early termination	Primal-dual gap	Infeasibility solving
ACTIVE-SET	QPOASES	Dense	✓	✓	✗	✓	✗
	QUADPROG	Dense	✗ [1]	✗ [1]	✗	✓	✗
	DAQP	Dense	✗ [1]	✗ [1]	✗	✓	✗
INTERIOR-POINT	GUROBI	Sparse	✗	✗	✓	✓	✗
	MOSEK	Sparse	✗	✗	✓	✓	✗
	CVXOPT	Dense	✗	✗	✓	✓	✗
	ECOS	Sparse	✗	✗	✓	✓	✗
	QPSWIFT	Sparse	✗	✗	✓	✓	✗
AUGMENTED LAGRANGIAN	OSQP	Sparse	✓	✓	✓	✗	✗ [2]
	SCS	Sparse	✓	✓	✓	✓	✗ [2]
	PROXQP	Sparse & Dense	✓	✓	✓	✓	✓

[1] In their original implementation. [2] Not established nor implemented.

4.3.2 The ProxSuite library

PROXQP is implemented in C++ within the PROXSUITE library and builds extensively on the Eigen library [Guennebaud et al., 2010] for efficient linear algebra. The current implementation is tailored to both dense and sparse matrix operations and leverages recent CPU architectures equipped with advanced vectorization instructions. PROXSUITE is publicly available at https://github.com/Simple-Robotics/proxsuite under a permissive open source BSD-2-Clause license. It comes with out-of-the-box interfaces for the C++, Julia, and Python languages inspired from OSQP [Stellato et al., 2020]. The documentation of PROXSUITE's API is available online at `https://simple-robotics.github.io/proxsuite/`. In the following, we describe some key features of PROXSUITE, list our default parameters in Table 4.3 as well as a code snippet in Listing 4.1.

Cholesky factorization In PROXSUITE, we have developed dedicated dense and sparse LDLT Cholesky factorizations to explicitly account for the specific calculations. In particular, this Cholesky factorization implements advanced rank update routines to efficiently account for the change of active sets when solving (4.8) while lowering the overall memory footprint. Furthermore, when the number of constraints is too large compared to the number of (primal) variables, we add an option to factorize, using the dense backend, the matrix

$$H + \rho I + \tfrac{1}{\mu} C_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}^{\top} C_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}, \tag{4.39}$$

instead of the one involved in (4.8) (the latter choice is called PRIMALDUALLDLT, whereas the first PRIMALLDLT). A heuristic (comparing "typical" algorithmic complexity for factorizing,

updating, and solving (4.8) using one of the mentioned matrices) is also available for automatically choosing which matrix to factorize.

Moreover, a similar heuristic is available for the sparse backend when the system is large-scale. In such case, a sparse matrix-free routine then solves (4.39) in the same spirit as [Hermans et al., 2021, Section 4.1.3]. The sparse and dense Cholesky factorizations are also freely available within the ProxSuite library.

Table 4.3: Default hyperparameters for ProxQP.

Parameter	Value	Description
α	0.95	primal-dual interpolation
$\epsilon_{\text{bcl}}^0 = \epsilon^0$	1	accuracy
ρ	10^{-6}	primal proximal penalization
μ_{min}	10^{-9}	minimal penalization
μ_f	0.1	penalization decay factor
α_{bcl}, β_{bcl}	0.09, 0.9	BCL hyperparameters
k_{max}	10^4	safeguard
μ_e^0	10^{-3}	penalization for equality constraints
μ_i^0	10^{-1}	penalization for inequality constraints

Warm-start Since ProxQP is built on an augmented Lagrangian-based method, the algorithm is readily warm-startable. For example, it is a useful feature for solving a cascade of QPs that change slightly (e.g., for Model Predictive Control, Inverse Kinematics, Inverse Dynamics, SQPs, etc.). Default initial guesses are also available as options. Among others, by default and motivated by the fact that equality constraints are always active at an optimal solution, we use the following initialization for primal and dual variables:

$$\begin{bmatrix} x^0 \\ z^0 \end{bmatrix} = \begin{bmatrix} H + \rho I & C^\top \\ C & -\mu^0 I \end{bmatrix}^{-1} \begin{bmatrix} -g \\ u \end{bmatrix},$$

provided C is an equality constraint matrix and u and equality constraint vector.

Hot-start ProxQP has a dedicated feature, similar to the OSQP, SCS, and qpOASES solvers [Stellato et al., 2020, Ferreau et al., 2014, O'Donoghue et al., 2016], where we can *hot-start* the solver with a factorization from a previous related problem. More precisely, if QP updates concern only linear terms (e.g., u or g), then the updates are realized allocation-free in ProxQP. Furthermore, ProxQP has settings that combine warm-starts with a reuse of the whole previous data workspace (including the Cholesky factorization) called WARM_START_WITH_PREVIOUS_RESULT. This feature has been specially designed for solving control or SQP problems with changes concerning only linear parts.

Preconditioning ProxQP contains several preconditioning strategies, enhancing the overall numerical stability and convergence of the optimization process. The default preconditioner used in our current implementation is often referred to as the Ruiz equilibration [Ruiz, 2001]; see, e.g., [Stellato et al., 2020, Algorithm 2], also used in OSQP and QPALM.

Solving closest feasible problems ProxQP can solve the closest feasible QP problem if it is detected to be primal infeasible (following the same procedure described as in OSQP [Stellato

et al., 2020, Section 3.4]). If the option is activated, then the following stopping condition will be tracked:

$$\begin{cases} \|Hx^\star + g + C^\top z^\star\|_\infty \leq \epsilon_{\text{abs}} \\ \|C^\top [Cx^\star - u]_+\|_\infty \leq \|C^\top 1\|_\infty \epsilon_{\text{abs}} \\ |(x^\star)^\top Hx^\star + g^\top x^\star + (u + s^\star)^\top [z^\star]_+| \leq \epsilon_{\text{abs}} \end{cases}$$

Dealing with nonconvex QPs Since ProxQP is based on a primal-dual proximal method of multipliers, it is also able to find local minima of nonconvex QPs, provided that the primal proximal step size ρ is larger than the lowest negative eigenvalue of the cost Hessian H [Hermans et al., 2021, Theorem 2.6]. Following the methodology described in [Hermans et al., 2021, Section 4.2], ProxQP also provides routines for evaluating this minimal eigenvalue (using a power iteration-like algorithm, or methods from the Eigen library) from the and automatically sets ρ to make the intermediary quadratic sub-problems well-defined and convex.

Solving batches of QPs ProxSuite includes a dedicated data structure for parallel solving of batches of dense or sparse QPs.

```python
import proxsuite.proxqp as solver
n = 10    # primal variable
n_eq = 2  # equality constraints
n_in = 2  # generic type of ineq. constraints

# Create a qp object and change settings
qp = solver.dense.QP(n, n_eq, n_in)
qp.settings.primal_infeasibility_solving = True
qp.settings.initial_guess = solver.WARM_START_WITH_PREVIOUS_RESULT
qp.init(...)  # fill with QP data
qp.solve()

# Optionally set specific backend
qp1 = solver.dense.QP(n, n_eq, n_in, solver.dense.DenseBackend.PrimalDualLDLT)
qp2 = solver.dense.QP(n, n_eq, n_in, solver.dense.DenseBackend.PrimalLDLT)
qp3 = solver.dense.QP(n, n_eq, n_in, solver.dense.DenseBackend.Automatic)

# Solving N QPs in parallel
qp_vector = solver.dense.VectorQP()
for i in range(N):
    qp = qp_vector.init_qp_in_place(n, n_eq, n_in)
    qp.init(...)  # fill with data of i-th QP
solver.dense.solve_in_parallel(n_threads, qp_vector)
```

Listing 4.1: Code example for using the ProxSuite python API.

4.4 Applications to robotics and beyond

In this section, we evaluate ProxQP against other state-of-the-art approaches on classic control problems and representative robotic tasks. These practical problems illustrate how ProxQP takes advantage of structural properties (sparsity structure, warm-starts, hot-starts, early stopping). We also show that ProxQP's ability to solve primal infeasible problems can be leveraged in closed-loop MPC scenarios accounting for perturbations and measurement errors that yield infeasible problems. Finally, we conclude benchmarking ProxQP against alternative solvers of the state-of-the-art on more generic types of QPs [Maros and Mészaros, 1999], [Stellato et al., 2020].

4.4.1 Benchmark setup and scenarios

Our benchmarks focus on relatively small and medium-sized QPs, typically used for embedded applications such as in robotics (less than a thousand variables and constraints), with sparse or dense problem structures.

In the first set of experiments (see Section 4.4.2), we consider sparse convex MPC scenarios: the benchmark proposed by [Stellato et al., 2020, Section 8.4] and the one proposed by [Wang and Boyd, 2009, Section 5.A]. We also explain with these synthetic examples how primal infeasibility solving can be used in the context of closed-loop MPC : (i) first, we initialize one instance with a primal infeasible point with respect to the dynamics bounds $x_{\min}$ and $x_{\max}$; (ii) in the latter, we add an infeasible random perturbation to the dynamics.

In the second set of experiments, we focus on dense problems (see Section 4.4.3): inverse kinematics tasks, a closed-loop dense MPC tasks for controlling the center of mass of a humanoid robot, and a real-world closed-loop MPC scenario with the Upkie wheeled-biped robot [tasts-robots, 2023]. Through an extensive benchmark with the humanoid closed-loop MPC task, we also highlight that ProxQP is more robust than other state-of-the-art solvers against random perturbations that render the approximate solution infeasible.

Finally, we evaluate ProxQP on more generic and standard QPs of the literature from the optimization community.

For all experiments, ProxQP is compared against other state-of-the-art approaches: active-set methods (qpOASES, quadprog), interior-point methods (MOSEK, qpSWIFT) and augmented Lagrangian-based methods (SCS, OSQP). We exploit all available features (see Table 4.2) of each solver considering the problem structure. More precisely, OSQP, qpOASES, and SCS use warm-starting and hot-starting when possible. Box constraints on the state or control variables are specified when the solver API allows to do so. Otherwise, they are listed as parts of the inequality constraints. Moreover, interior-point and augmented Lagrangian-based methods exploit their underlying early-stopping strategy if it suits the task. The experiments were conducted on a standard laptop CPU, an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz. The benchmark is open-source and available at https://github.com/Bambade/proxqp_benchmark with an easy-to-use interface strongly inspired by the one proposed by OSQP [Stellato et al., 2020].

4.4.2 Sparse problems

In this subsection, we consider two sets of convex MPC problems: an existing convex-MPC benchmark from [Stellato et al., 2020, Section 8.4], and a sparse chain of oscillating masses from [Wang and Boyd, 2009, Section 5A]. In both cases, we run systems over 100 simulation steps. Their initial states x_{init} are uniformly chosen between reasonable bounds. We thus measure the average of the full-time needed for solving all 100 simulation steps. For each separate randomization seed, the full simulation steps' is also averaged over 10 consecutive runs in order to mitigate the impact of loading costs. Furthermore, we consider two different scenarios: (i) one when early-stopping is relevant ($\epsilon_{\text{abs}} = 10^{-3}$); and another (ii) where higher accuracy is required ($\epsilon_{\text{abs}} = 10^{-6}$). In this test set, only the vectors terms in (QP) change in this test case, the practitioner can use warm-start and hot-start strategies if available. These features are exploited by OSQP, qpOASES, SCS and ProxQP. Furthermore, inequality constraints in these problems only consist of box constraints, a specificity that SCS leverages with a custom feature. Finally, since the problems have a sparse structure, ProxQP uses its sparse back-end.

Convex MPC test set

Experiment setup In this set of experiments, we consider the MPC benchmark proposed in [Stellato et al., 2020, Section 8.4]. This benchmark consists of LQR problems with additional

Table 4.4: Sparse convex MPC benchmark (total runtimes in ms for solving 100 simulation steps)

Problem type	Sizes	ProxQP	quadprog	OSQP	qpOASES	SCS	qpSWIFT	MOSEK
Convex MPC $\epsilon_{\text{abs}} = 10^{-3}$	n=76,m=142	**5.7±0.2**	✗[2]	6.7±0.7	37.6±23.9	11.9±2.2	13.5±0.5	82.1±3.5[1]
	n=162,m=294	**16.4±1.7**	✗[2]	24.1±6.0	393.9±109.9	51.0±15.7	47.1±4.6	211.5±13.3
	n=216,m=392	**27.6±4.8**	✗[2]	48.2±29.4	1434.2±480	65.6±30.9	80.0±3.6	311.3±15.4[1]
	n=270,m=490	**43.2±8.3**	✗[2]	77.5±65.6	2345.5±608	104.7±66.0	131.0±7.4	454.2±18.6
Convex MPC $\epsilon_{\text{abs}} = 10^{-6}$	n=76,m=142	**8.5±1.4**	✗[2]	14.1±8.4	27.1±15.4	25.5±14.6	16.5±0.7	84.0±3.8[1]
	n=162,m=294	**21.1±2.3**	✗[2]	94.0±101.9	452.9±98.3	101.8±47.9	57.8±5.3	216.3±14.1[1]
	n=216,m=392	**36.1±4.3**	✗[2]	150.1±136.0	1626.6±650.1	162.0±95.0	99.5±4.9	314.1±17.4[1]
	n=270,m=490	**57.0±9.0**	✗[2]	346.7±362.0	2670.3±842.4	311.1±122.5	164.1±9.9	459.6±21.0[1]
Chain of mass $\epsilon_{\text{abs}} = 10^{-3}$	n=462,m=834	**45.2±0.8**	(28.3±0.6)E3	59.1±1.7	(80.2±1.6)E3	161.1±11.1	215.3±1.3	566.8±14.1
Chain of mass $\epsilon_{\text{abs}} = 10^{-6}$	n=462,m=834	**67.8±11.1**	(28.3±0.6)E3	104.9±9.6	(80.2±1.6)E3	172.6±9.8	248.9±3.1	575.5±16.8[1]

[1] The solution does not satisfy the desired absolute accuracy. [2] quadprog cannot solve QPs that are not strictly convex.

state and input constraints, as described in [Stellato et al., 2020, Appendix A3], with a horizon $T = 10$ starting from $x_0 = x_{\text{init}} \sim \mathcal{U}(-0.5\bar{x}, 0.5\bar{x})$ with $\bar{x} \sim \mathcal{U}(1, 2)$:

$$\begin{aligned} \min_{x_t \in \mathbb{R}^{n_x}, u_t \in \mathbb{R}^{n_u}} \; & x_T^\top Q_T x_T + \sum_{t=0}^{T-1} x_t^\top Q x_t + u_t^\top R u_t, \\ \text{s.t. } & x_{t+1} = A x_t + B u_t, \\ & -\bar{x} \leq x_t \leq \bar{x}, \; -\bar{u} \leq u_t \leq \bar{u}, \end{aligned} \tag{4.40}$$

with $Q, Q_T \in \mathbb{S}_+(\mathbb{R}^{n_x})$, $R \in \mathbb{S}_{++}(\mathbb{R}^{n_u})$, and $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$. More precisely, we solve this optimal control problem in a receding horizon fashion, computing an optimal input sequence $u_0, \ldots, u_{T-1}$, applying the first input u_0 to the system and propagating the state to the next time step. The whole procedure is repeated 100 times. We choose the state variable dimension n_x and control input dimension n_u for the context of small embedded systems, concretely $n_x = 6, 12, 16, 20$ and $n_u = n_x/4$. This benchmark generates sparse problems (i.e., the KKT matrix of the QP has about 5% of nonzeros elements) and QPs whose dimensions (number of primal variables and constraints) range from 76 to 490.

Runtime results The first and second-row blocks of Table 4.4 show the results of this test case. We can see that in both scenarios ProxQP is the fasted method with a speed-up ranging from 1.2 to 1.8 in the early-stopping scenario (with respect to OSQP, the second fastest approach); and from 1.7 to 2.9 in the high accuracy scenario. We can also observe that sparse solvers (ProxQP, OSQP, SCS, qpSWIFT, MOSEK) obviously provide better timings than dense solver (qpOASES). Furthermore, one can notice that methods based on ADMM (i.e., OSQP and SCS) are slower to converge to high-accuracy solutions.

Solving MPC from primal infeasible points In order to solve an MPC problem starting from $x_{\text{init}} \sim \bar{x} + \mathcal{U}(0, 0.01)$, we consider the following strategy: the solution found at the first simulation step provides a solution trajectory $\hat{x}_t, \hat{u}_t$ (over a horizon of T time steps for control inputs $\hat{u}_t$ and $T + 1$ for the state $\hat{x}_t$) for the closest feasible QP of (4.40) (see orange line in Figure 4.1). ProxQP also provides optimal shifts $\hat{s}_t^u$, $\hat{s}_t^x$ for the inequality constraints on x and u; and $\hat{s}_t^e$ for equality constraints. We make the forward updates as follows:

- $u_0 = \hat{u}_0 - \hat{s}_0^u$: we bring back the closest feasible input solution to the solution space of the initial QP (so u_0 satisfies the input constraints since $\hat{u}_0 \leq \bar{u} + \hat{s}_0^u$ by construction);
- $x_1 = A\hat{x}_0 + Bu_0 - \hat{s}_0^e$: we update the trajectory from a feasible control input u_0 and optimal dynamic $\hat{x}_0$ and shift $\hat{s}_0^e$. It lies on the green line drawn in Figure 4.1. Compared to the

closest feasible optimal solution $\hat{x}_1 = A\hat{x}_0 + B\hat{u}_0 - \hat{s}_0^e$, we make a drift

$$\begin{aligned} x_1 - \hat{x}_1 &= A\hat{x}_0 + Bu_0 - \hat{s}_0^e - [A\hat{x}_0 + B\hat{u}_0 - \hat{s}_0^e], \\ &= B(u_0 - \hat{u}_0) = -B\hat{s}_0^u, \end{aligned}$$

which tends towards zero, as soon as the optimal shift over the control input tends towards zero.

- We repeat the closed-loop process starting from x_1 to derive $\hat{x}_1, \hat{u}_1$ and then $x_2 = A\hat{x}_1 + B(\hat{u}_1 - \hat{s}_1^u) - \hat{s}_1^e$.

The following update rule is motivated by the following intuition. Assume the steady state x_{goal} is still reachable by an unconstrained LQR scheme (see the grey line in Figure 4.1)

$$\begin{aligned} \min_{x_t \in \mathbb{R}^{n_x}, u_t \in \mathbb{R}^{n_u}} \; & x_T^\top Q_T x_T + \sum_{t=0}^{T-1} x_t^\top Q x_t + u_t^\top R u_t, \\ \text{s.t. } & x_{t+1} = Ax_t + Bu_t, \; x_0 = x_{\text{init}}, \end{aligned} \tag{4.41}$$

with a solution sequence $\{x_t^{\text{LQR}}\}_t$, $\{u_t^{\text{LQR}}\}_t$. As the constraints are linear, strong duality holds, and the primal of problem (4.41) p^{LQR} equals its dual δ^{LQR} for some dual multipliers $\{y_t^{\text{LQR}}\}_t$. Noting $\mathcal{L}^{\text{LQR}}$ the Lagrangian of problem (4.41), we have

$$\begin{aligned} -\infty < \delta^{\text{LQR}} &= \mathcal{L}^{\text{LQR}}(\{x_t^{\text{LQR}}\}_t, \{u_t^{\text{LQR}}\}_t, \{y_t^{\text{LQR}}\}_t) \\ &= \mathcal{L}(\{x_t^{\text{LQR}}\}_t, \{u_t^{\text{LQR}}\}_t, \{y_t^{\text{LQR}}\}_t, 0, 0), \\ &= \inf_{x_t, u_t} \mathcal{L}(\{x_t\}_t, \{u_t\}_t, \{y_t^{\text{LQR}}\}_t, 0, 0), \end{aligned}$$

with $\mathcal{L}$ the Lagrangian of the original problem (4.40), and 0 corresponds to the value of the dual multipliers associated with the box constraints with respect to x_t and u_t. It thus guarantees the dual of the original problem has a non-empty domain and that there exists a closest feasible solution sequence $\{\hat{x}_t\}_t, \{\hat{u}_t\}_t$ (see the orange line in Figure 4.1) with associated optimal shifts $\{\hat{s}_t^e\}_t, \{\hat{s}_t^x\}_t, \{\hat{s}_t^u\}_t$.

If we compare now $\|x_1^{\text{LQR}} - x_{\text{goal}}\|_2^2$ with $\|x_1 - x_{\text{goal}}\|_2^2$ assuming that both $x_1^{\text{LQR}} > \bar{x}$ and $\hat{x}_1 > \bar{x}$ we see

$$\begin{aligned} \|x_1^{\text{LQR}} - x_{\text{goal}}\|_2^2 = \|x_1^{\text{LQR}} - \bar{x}\|_2^2 + \|\bar{x} - x_{\text{goal}}\|_2^2 + \\ 2(x_1^{\text{LQR}} - \bar{x})^\top (\bar{x} - x_{\text{goal}}), \end{aligned}$$

and

$$\begin{aligned} \|x_1 - x_{\text{goal}}\|_2^2 &= \|x_1 - \bar{x}\|_2^2 + \|\bar{x} - x_{\text{goal}}\|_2^2 + 2(x_1 - \bar{x})^\top (\bar{x} - x_{\text{goal}}), \\ &= \|\hat{x}_1 - B\hat{s}_0^u - \bar{x}\|_2^2 + \|\bar{x} - x_{\text{goal}}\|_2^2 + \\ &2(\hat{x}_1 - B\hat{s}_0^u - \bar{x})^\top (\bar{x} - x_{\text{goal}}) \\ &= \|\underbrace{\hat{x}_1 - \bar{x}}_{=\hat{s}_1^x}\|_2^2 + \|B\hat{s}_0^u\|_2^2 - 2(B\hat{s}_0^u)^\top (\hat{s}_1^x) + \\ &\|\bar{x} - x_{\text{goal}}\|_2^2 + 2(\hat{s}^x - B\hat{s}_0^u)^\top (\bar{x} - x_{\text{goal}}). \end{aligned}$$

Thus we have:

$$\begin{aligned} &\|x_1^{\text{LQR}} - x_{\text{goal}}\|_2^2 - \|x_1 - x_{\text{goal}}\|_2^2 = \|x_1^{\text{LQR}} - \bar{x}\|_2^2 + \\ &2(x_1^{\text{LQR}} - \bar{x})^\top (\bar{x} - x_{\text{goal}}) - \|\hat{s}_1^x\|_2^2 - \|B\hat{s}_0^u\|_2^2 - \\ &2(\hat{s}_1^x - B\hat{s}_0^u)^\top (\bar{x} - x_{\text{goal}}) + 2(B\hat{s}_0^u)^\top (\hat{s}_1^x). \end{aligned} \tag{4.42}$$

Since $x_1^{\text{LQR}} > \bar{x}$ and $\bar{x} > x_{\text{goal}}$, then $(x_1^{\text{LQR}} - \bar{x})^\top (\bar{x} - x_{\text{goal}}) > 0$. Hence, if the optimal shifts $\hat{s}_0^u$ and $\hat{s}_1^x$ are sufficiently small, Equation (4.42) shows that $\|x_1^{\text{LQR}} - x_{\text{goal}}\|_2 > \|x_1 - x_{\text{goal}}\|_2$. The same calculus shows in this setting that $\|x_1^{\text{LQR}} - \bar{x}\|_2 > \|x_1 - \bar{x}\|_2$. Consequently, this update rule guarantees in this setting that if x_{init} is not too far in the sense that the optimal shifts are small enough, then the update rule provides a point from which the feasible target x_{goal} is both closer to reach and closer of the feasible frontier. Our intuition is that the property of this update rule may eventually lead to a feasible update, with demonstration further investigated in future work.

We illustrate the application of this update rule with an instance of the convex MPC scenario for $n_x = 6$ and $n_u = 1$ in Figure 4.2. The bottom figure shows the control trajectory u_t of the system. As expected, it lies in the control bounds $-\bar{u}$ and $\bar{u}$. The six upper figures show the 6 components of the dynamics x_t. All these components are initially infeasible for a fixed number of simulation steps since they are above $\bar{x}$. Since the initial infeasible perturbation is not too large (i.e., the optimal shifts are sufficiently small), the dynamics eventually converge to the feasible steady-state position.

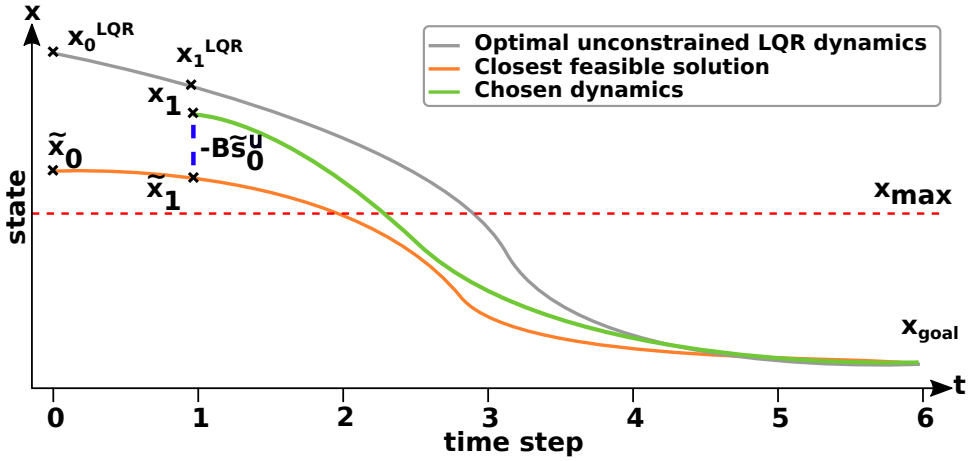

*Figure 4.1: A convex MPC problem is initialized with an infeasible state $x_0 = x_{init} > x_{\max}$. **Grey:** Yet, $x_0^{LQR} = x_0$ enables reaching a horizon T a feasible state x_{goal} with an unconstrained LQR scheme (see (4.41)). **Orange:** It guarantees that a closest feasible trajectory $\hat{x}_t$ exists with optimal shifts $\hat{s}_t^x, \hat{s}_t^u, \hat{s}_t^e$. **Green:** We choose the update $x_1 \overset{def}{=} \hat{x}_1 - B\hat{s}_0^u$, which is guaranteed not moving away x_1 from x_{goal} for sufficiently small optimal shifts $\hat{s}_1^x, \hat{s}_0^u, \hat{s}_1^e$. More precisely, $\|x_1 - x_{goal}\|_2 \leq \|x_1^{LQR} - x_{goal}\|_2$ and $\|x_1 - x_{\max}\|_2 \leq \|x_1^{LQR} - x_{goal}\|_2$.*

Sparse chain of oscillating masses

Experiment setup We consider the following MPC problem involving a chain of six masses interconnected with spring-dampers [Wang and Boyd, 2009, Section 5A]

$$\begin{aligned} \min_{x_t \in \mathbb{R}^{n_x}, u_t \in \mathbb{R}^{n_u}} \ & \frac{1}{2} x_T^\top Q_T x_T + \frac{1}{2} \sum_{t=0}^{T-1} x_t^\top Q x_t + u_t^\top R u_t, \\ \text{s.t. } & x_{t+1} = A x_t + B u_t, \ x_0 = x_{\text{init}} \sim \mathcal{U}(-0.5, 0.5), \\ & -4 \leq x_t \leq 4, \ -0.5 \leq u_t \leq 0.5, \end{aligned}$$

with $R = I$ and $Q = I$. The problem dimensions are $n_x = 12, n_u = 3$ and the horizon is $T = 30$ [Wang and Boyd, 2009, Section 5A]. It generates QPs with 462 variables and 864 constraints.

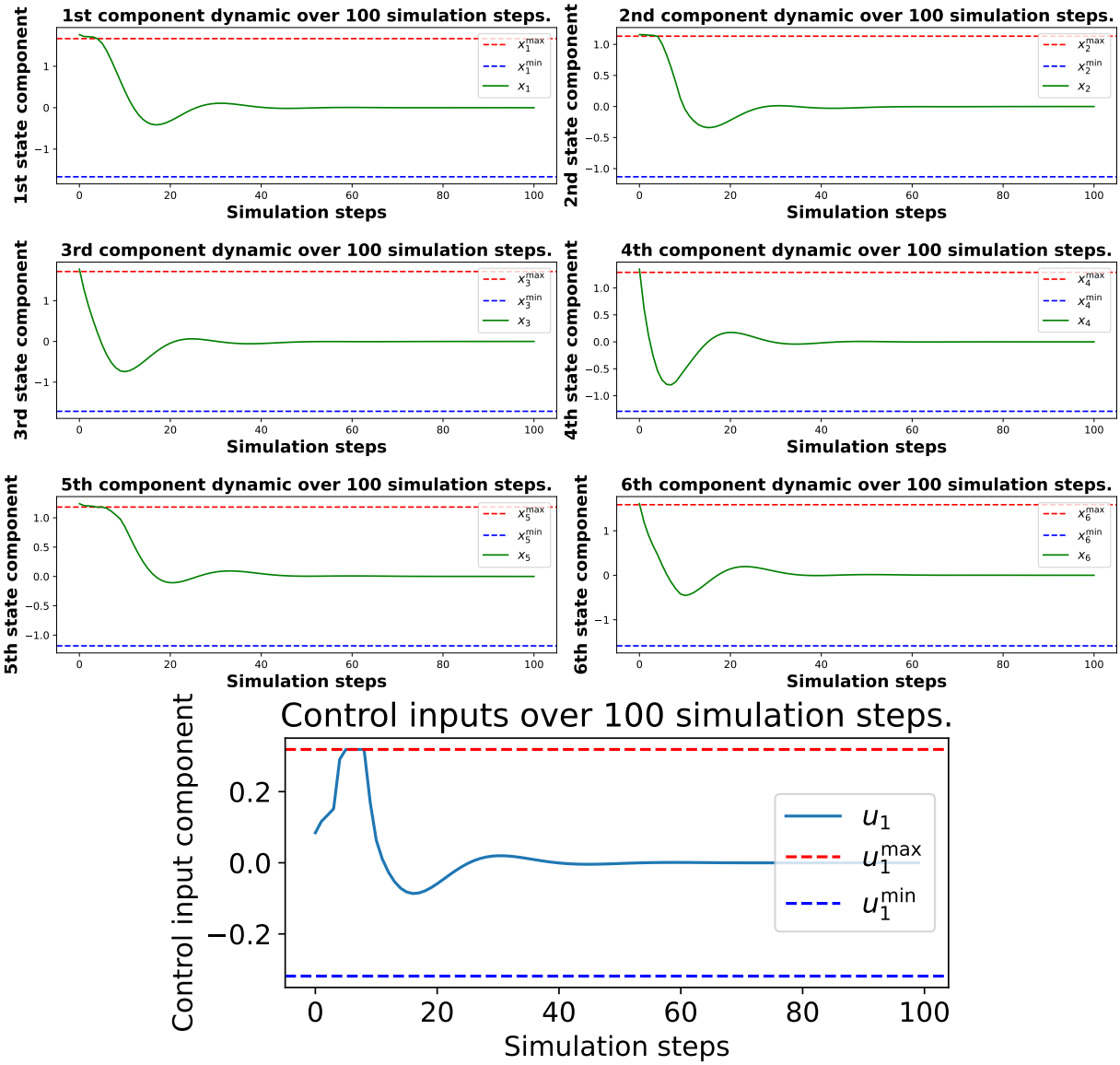

Figure 4.2: Closed-loop convex MPC simulation over 100 simulating steps with $n_x = 6$ and $n_u = 1$ starting from an infeasible initial state x_0 using ProxQP. At each new simulation step t, the new control input equals $u_t = \hat{u}_t - \hat{s}_t^u$, so it is guaranteed to be feasible, while the dynamics follow $x_t = \hat{x}_t - B\hat{s}_t^u$. Since the initial infeasible perturbation is not too large, the dynamics converge after a fixed number of steps to a feasible regime.

Runtime results The last two-row blocks of Table 4.4 show the runtime results. We can see that ProxQP is the fastest method with a speed-up ranging from 1.3 (early stopping scenario) to 1.5 (high accuracy scenario).

Solving MPC with infeasible perturbations To illustrate another possible scenario where primal infeasibility solving might be useful, we consider the following setup: at about one fourth of the simulation (i.e., the 25^{th} simulation step), a random perturbation $w_t \sim 4.8 + \mathcal{U}(0, 0.01)$ disturbs the usual dynamic update

$$x_{t+1} = Ax_t + Bu_t + w_t,$$

which is thus guaranteed at the next LQR simulation step to create an infeasible problem since x_t is already close to the steady state for such a system after 25 steps. We apply the same update

rule as exposed in the previous MPC problem to tackle this infeasible perturbation since the problem structure is similar. The upper figure in Figure 4.3 shows the 12 components of the chain system. It becomes infeasible after the 25^{th} closed-loop update. Nevertheless, we can see that it eventually recovers to the steady state solution after a finite number of time steps. The bottom figure in Figure 4.3 shows the 3 control input components. We can see that, as expected, they all satisfy the constraint bounds thanks to the update rule $u_t = \hat{u}_t - \hat{s}_t^u$.

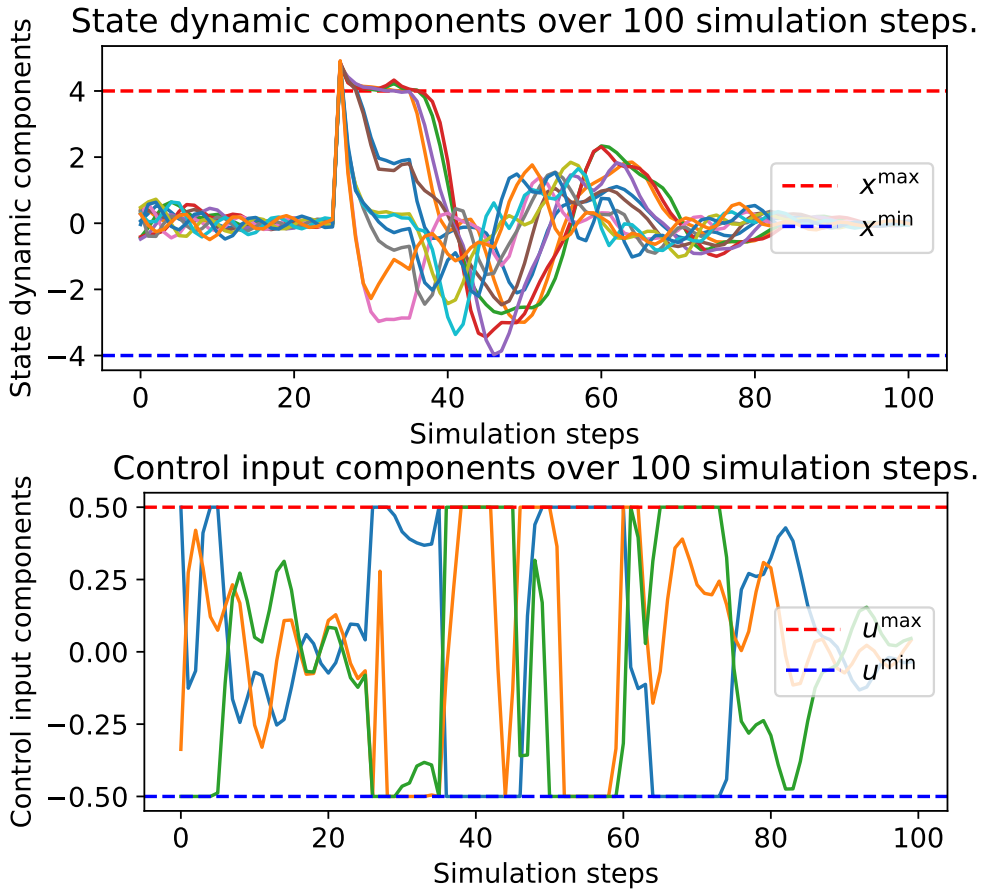

Figure 4.3: Convex MPC problem over 100 simulation steps for controlling a chain of masses with an infeasible dynamic perturbation at the fourth of the simulation. $n_x = 12$ and $n_u = 3$. The control inputs and the dynamics are updated following Equation 4.4.2. After a fixed number of time steps, the system manages to recover its asymptotic steady state while satisfying during the whole process feasible control bounds.

4.4.3 Dense problems

Inverse kinematics In this set of experiments, we consider different inverse kinematic (IK) tasks proposed for motion control of articulated robots using Pinocchio [Carpentier et al., 2019] and the open-source Pink library `https://github.com/tasts-robots/pink`. The resulting motion controller is run for 5 s (i.e., solving 1,000 consecutive QPs with a time-step of 5 ms). Each such run is repeated 50 times, and we measure the runtime per timestep average for achieving each IK task. Considering the structure of the problem, OSQP, SCS, qpOASES, and ProxQP benefit from warm-starting, and ProxQP uses its dense-backend. Furthermore, from a practitioner's point of view, we consider the minimal accuracy needed ϵ_{abs} for achieving these tasks (i.e., solving each IK task while maintaining sufficient smoothness in the resulting motions). We thus imposed $\epsilon_{\text{abs}} = 10^{-5}$ for each task, except when using STRETCH which required a lower $\epsilon_{\text{abs}} = 10^{-7}$ in order to satisfy to be numerically stable while enforcing joint limits.

Table 4.5: Dense QP benchmarks (average runtime per time-step (IK) and total simulation runtimes).

Task name	ProxQP	quadprog	OSQP	qpOASES	SCS	qpSWIFT	MOSEK
UR3 IK	**13.2±0.1μs**	17.0±3.3μs	15.8±0.2μs	21.7±0.3μs	✗ [1]	52.2±1.2μs	323.1±74μs [2]
UR5 IK	**11.9±0.1μs**	16.8±0.2μs	16.5±0.2μs	21.4±0.2μs	✗ [1]	53.8±0.8μs	310.8±5.3μs [2]
DUAL ARMS IK	**17.2±0.1μs**	23.3±0.2μs	40.4±0.6μs	330.4±5.8μs	81.5±0.6μs	152.1±1.1μs	554.4±25.1μs [2]
KINOVAGEN2 IK	**15.8±0.1μs**	18.9±0.2μs	17.0±0.2μs	31.4±0.4μs	46.5±1.3μs	53.7±0.4μs	375.4±14.9μs [2]
SIGMABAN IK	**14.1±0.2μs**	25.2±0.4μs	45.2±0.9μs	523.6±4.8μs	68.6±0.5μs	224.9±2.0μs	452.5±8.8μs [2]
STRETCH IK	**26.9±0.3μs**	36.7±0.6μs	53.1±0.9μs	212.6±0.8μs	455.3±23.8μs [2]	152.8±1.5μs	✗ [3]
CHAIN80 SQP	**15.6±0.3ms**	355.8±0.9ms	456.5±2.6ms	182±2.9ms	837±16.0ms	2193.9±22.3ms	1554.6±19.5ms [2]
CHAIN80W SQP	**265.7±2.9ms**	610.1±5.7ms	2141.9±22.4ms	467.4±3.3ms [1]	✗ [4]	✗ [4]	3444.2±30.3ms [2]
HUMANOID MPC $\epsilon_{\text{ABS}} = 10^{-2}$	**1.6±0.01ms**	4.5±1.8ms	1.8±0.01ms	18.0±5.3ms	433.0±21.7ms[5]	✗[3]	70.7±0.7ms
HUMANOID MPC $\epsilon_{\text{ABS}} = 10^{-4}$	**2.6±0.02ms**	4.5±1.8ms	2.7±0.03ms	18.0±5.3ms	80.2±2.9ms[5]	✗[3]	68.3±0.3ms
HUMANOID MPC $\epsilon_{\text{ABS}} = 10^{-6}$	**4.4±0.05ms**	4.5±1.8ms	4.4±0.05ms	18.0±5.3ms	64.3±2.4ms[5]	✗[3]	69.7±0.8ms

[1] The solver throws a factorization error (of non-convexity). [2] The solution does not match the desired accuracy [3] The solver does not manage to satisfy configuration limits. [4] The solver does not manage to handle upper bound constraints and outputs infeasibility errors.
[5] Low accuracy iterates provokes with SCS warm starts more difficult QPs to solve in a closed-loop strategy.

Table 4.6: Humanoid locomotion MPC problems with perturbations.

Noise Level	ProxQP	quadprog	OSQP	qpOASES	SCS	qpSWIFT	MOSEK
10.0	**11.9±9.7%**	1.0±0.2%	1.9±0.9%	1.0±0.2%	1.0±0.2%	0±0.0%	1.0±0.2%
5.0	**58.38±36.4%**	1.1±0.3%	2.1±1.1%	1.1±0.3%	1.1±0.4%	0±0.0%	1.1±0.3%
1.0	**100±0.0%**	1.4±0.8%	3.5±2.4%	1.4±0.8%	1.5±1.1%	0±0.0%	1.4±0.9%
0.5	**100±0.0%**	1.8±1.2%	5.5±3.8%	1.9±1.5%	2.1±1.6%	0±0.0%	1.8±1.2%
0.1	**100±0.0%**	3.3±2.6%	51.6±36.7%	4.3±3.8%	4.9±4.3%	0±0.0%	3.3±2.6%
0.05	**100±0.0%**	3.5±3.2%	97.6±13.5%	5.0±6.9%	7.7±6.5%	0±0.0%	3.5±3.2
0.01	**100±0.0%**	4.4±4.4%	**100±0.0%**	7.7±9.8%	60.2±37.8%	0±0.0%	4.4±4.5%
10^{-3}	**100±0.0%**	5.0±5.2%	**100±0.0%**	11.4±12.5%	**100±0.0%**	0±0.0%	5.0±5.2%
10^{-4}	**100±0.0%**	5.0±5.2%	**100±0.0%**	15.5±16.8%	**100±0.0%**	0±0.0%	5.0±5.2%
10^{-5}	**100±0.0%**	5.0±5.2%	**100±0.0%**	83.0±36.5%	99.1±8.9%	0±0.0%	5.1±5.3%
10^{-7}	**100±0.0%**	5.0±5.2%	**100±0.0%**	**100±0.0%**	97±14.8%	0±0.0%	44.8±34.2%
10^{-9}	**100±0.0%**	5.0±5.2%	**100±0.0%**	**100±0.0%**	**100±0.0%**	0±0.0%	**100±0.0%**
0.0	**100±0.0%**	**100±0.0%**	**100±0.0%**	**100±0.0%**	**100±0.0%**	0±0.0%	**100±0.0%**

Table 4.5 reports runtimes in its second-row block. It shows that PROXQP is the fastest method with a speed-up ranging from 1.2 to 1.8 with respect to OSQP or QUADPROG, the next two fastest approaches. Apart from OSQP, we note how dense solvers are faster than sparse ones, which can be explained by the fact that IK problems depict a relatively high ratio of nonzero elements with respect to the problem size (the ratios of sparsity of the Hessian cost matrix H ranges from 38% to 100%, and between 4% and 16% for the constraint matrix).

Chain of oscillating masses In this set of experiments, we consider a dense version of the chain of oscillating masses MPC problem proposed by [Ferreau et al., 2014, Section 6.3] as an SQP problem. The first test problem CHAIN80 aims at regulating a chain of nine masses connected by springs into a certain steady state. One end of the chain is fixed on a wall while the three velocity components of the other end are used as control input with fixed lower and upper bounds. The prediction horizon of 16 seconds is divided into 80 control intervals. The model equations are derived from the linearisation of the non-linear ODE model (with 57 states) at the steady state. Deviation from the steady state, the velocities of all masses, and the control action are penalized via the objective function. It forms a sequence of 101 QPs with 240 variables with only box constraints over the input variables. The second test problem CHAIN80W considers the same settings by adding also state constraints into the optimization problem in order to ensure that the chain does not hit a vertical wall close to the steady state. It results thus in a sequence of 101 QPs with 240 variables and 709 constraints. These two SQPs are run 10 times, and we measure the total runtime necessary for solving these tasks to an accuracy $\epsilon_{\text{abs}} = 10^{-6}$.

We can see the runtime results in the first-row block of Table 4.5. It shows that PROXQP is

the fastest method with a speed-up ranging from 1.8 to 11.6 with respect to qpOASES, the second fastest approach. We can notice again that dense solvers are faster than the other sparse solvers.

Closed-loop MPC

In this last set of experiments, we consider dense convex MPC problems for controlling a walking humanoid robot in simulations, and the real robot Upkie [tasts-robots, 2023] in a balancing scenario. We evaluate again the robustness of the solvers to perturbations.

Humanoid locomotion via cart-table model The task consists in controlling a reduced model of a humanoid robot, consisting of its center-of-mass c and zero-tilting moment point (ZMP) z, using the center-of-mass jerk $u = \dddot{c}$ as control input [Kajita et al., 2001, Caron et al., 2019]. Figure 4.4 depicts a trajectory of this system. The goal is to keep the ZMP inside the support polygon of the feet while achieving a prescribed walking velocity. The resulting MPC problem can be cast as a quadratic program [Wieber, 2006a] of the form:

$$\begin{aligned} \min_{\substack{x_t \in \mathbb{R}^{n_x} \\ u_t \in \mathbb{R}^{n_u}}} \; & w_T \|x_T - x_{\text{goal}}\|_2^2 + \sum_{t=0}^{T-1} w_x \|x_t - x_{\text{goal}}\|_2^2 + w_u \|u_t\|_2^2, \\ \text{s.t. } & x_{t+1} = Ax_t + Bu_t, \; x_0 = x_{\text{init}}, \\ & Cx_t + Du_t \leq e_t, \end{aligned} \tag{4.43}$$

where $n_x = 3$ (for center-of-mass position c, velocity $\dot{c}$ and acceleration $\ddot{c}$), $n_u = 1$ and $T = 16$. As in [Wieber, 2006a], horizontal coordinates can be decoupled, so that we can focus on single-dimensional coordinates $c, z \in \mathbb{R}$. For the sake of this example we choose lateral coordinates and a target velocity of zero corresponding to walking in place.

The problem is solved with respect to the stacked vector $U = [u_0 \dots u_{T-1}]$ of control inputs, deriving states x_t linearly from U by recursive expansion of the dynamics $x_{t+1} = Ax_t + Bu_t$ [Audren et al., 2014]. This leads to a series of dense QPs with 16 variables and inequality constraints. The experiment is run for 100 simulation steps with a duration of $0.1s$ each, and the results are averaged over 100 trials. We measure the average total runtime for solving the task. Since the system is time-invariant, OSQP, SCS, qpOASES, and ProxQP benefit from warm-starting and hot-starting. In our experiments, setting $\epsilon_{\text{abs}} = 10^{-2}$ is sufficient to solve the task for all solvers based on IP and AL-based methods. Yet, they do not benefit the same way from this early-stopping feature. For this reason, we execute the runtime benchmark for $\epsilon_{\text{abs}} = 10^{-2}$, $\epsilon_{\text{abs}} = 10^{-4}$ and $\epsilon_{\text{abs}} = 10^{-6}$.

The runtime results are displayed in the last row block of Table 4.5. ProxQP is slightly faster than OSQP or quadprog, the best performing state-of-the-art solvers on this test set, where we note again how, as expected, dense solvers are generally faster than sparse solvers. Finally, we can see that SCS benefits less from early stopping here. Timings for SCS improve when the iterates are more accurate.

Robustness to perturbations We apply random (Gaussian) noise at each simulation step to the center-of-mass acceleration with an amplitude σ modeling state observation inaccuracies or unmodeled dynamics. When the ZMP is close to the edge of its support polygon, a small perturbation in center-of-mass acceleration can cause it to cross the edge, which would break the foot contact. For this reason, it is common practice to add safety margins to ZMP support areas in real systems, sometimes reducing them to a square well within the full area [Scianca et al., 2020]. We include such margins in our QP formulation, as depicted by the light-dotted lines in Figure 4.4. In order to quantify the robustness of the different solvers to various noise amplitudes, we measure the percentage of the full trajectory (100 simulation steps) each solver manages to solve and average over 100 seeds. This closed-loop task is interrupted if the primal

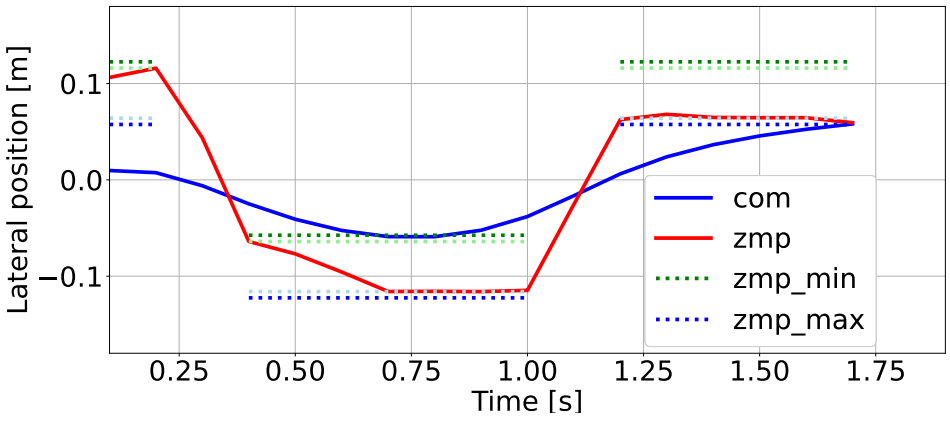

Figure 4.4: Controlling the lateral center-of-mass trajectory (blue) to maintain the ZMP (red) within the real support polygon (dotted dark green and blue). Light dotted lines are more conservative ZMP bounds used during solving QP. Random perturbation of the current center-of-mass acceleration can lead to a ZMP outside the support polygon and a primal infeasible QP.

feasibility violation with respect to the original support polygon (dark dotted lines) exceeds the accuracy of $\epsilon_{abs} = 10^{-2}$, or if the solver does not return any solution. For all solvers, we set the primal infeasibility tolerance to the minimum possible value in order to solve as many problems as possible. In Table 4.6, ProxQP solves the entire trajectory for noise amplitudes lower than 1.0 m/s^2, while the second best solver OSQP reaches this level of performance for a noise amplitude of 10^{-2} m/s^2. All other solvers that are based on interior-point or active-set methods exhibit a less robust behavior on this task.

Real-robot balancing We test ProxQP on real hardware with closed-loop model predictive control experiments on the Upkie wheeled biped. The controller runs entirely on the robot's embedded computer, a Raspberry Pi 4 Model B equipped with a quad-core ARM Cortex-A72 64-bit SoC @ 1.8 GHz. Note that this CPU has fewer cores and a lower frequency than the standard laptop CPU used in other test cases. We formulate the MPC problem as a quadratic program using the same strategy as in (4.43), for a discretization of the linearized cart-pole model (for wheels) rather than cart-table (for flat feet). States observed from IMU Kalman filtering and wheel odometry are fed directly as an initial state x_{init} to the MPC problem. We set $T = 50$ timesteps with a time discretization of 20 ms, resulting in a receding horizon duration of 1 s. Task weights are chosen as $w_T = 1$, $w_x = 10^{-3}$ and $w_u = 10^{-3}$. The resulting QPs have dimension $n = 50$ with $m = 100$ inequality constraints. They are solved by ProxQP with hot-starting in 0.8 ± 0.02 ms, fitting well within the budget of a 200 Hz control loop. Figure 4.5 reports the resulting computation times alongside observed states for a sample run of the balancing controller on the robot. The source code for this experiment is available in the `mpc_balancer` directory of the robot's repository [tasts-robots, 2023].

4.4.4 Generic QPs

Maros-Mészáros problems The Maros-Mészáros test set [Maros and Mészaros, 1999] is composed of 138 "hard" QPs. Most of them are sparse and ill-conditioned problems, and they contain up to 90597 variables and 180895 constraints. About 83% of the problems have a sparsity level lower than 10% for the Hessians H, as well as for the constraint matrices. We restrict the benchmark to problems whose dimensions (constraints and variables) are below or equal to 10^3. This results in a subset of 62 problems, about 45% of the Maros-Mészáros test set, for which two-third of the problems have a sparsity level no larger than 20%.

We report performance profiles [Stellato et al., 2020, Hermans et al., 2021, Bambade et al., 2022] on Figure 4.6 comparing ProxQP with its dense backend against the other solvers of the

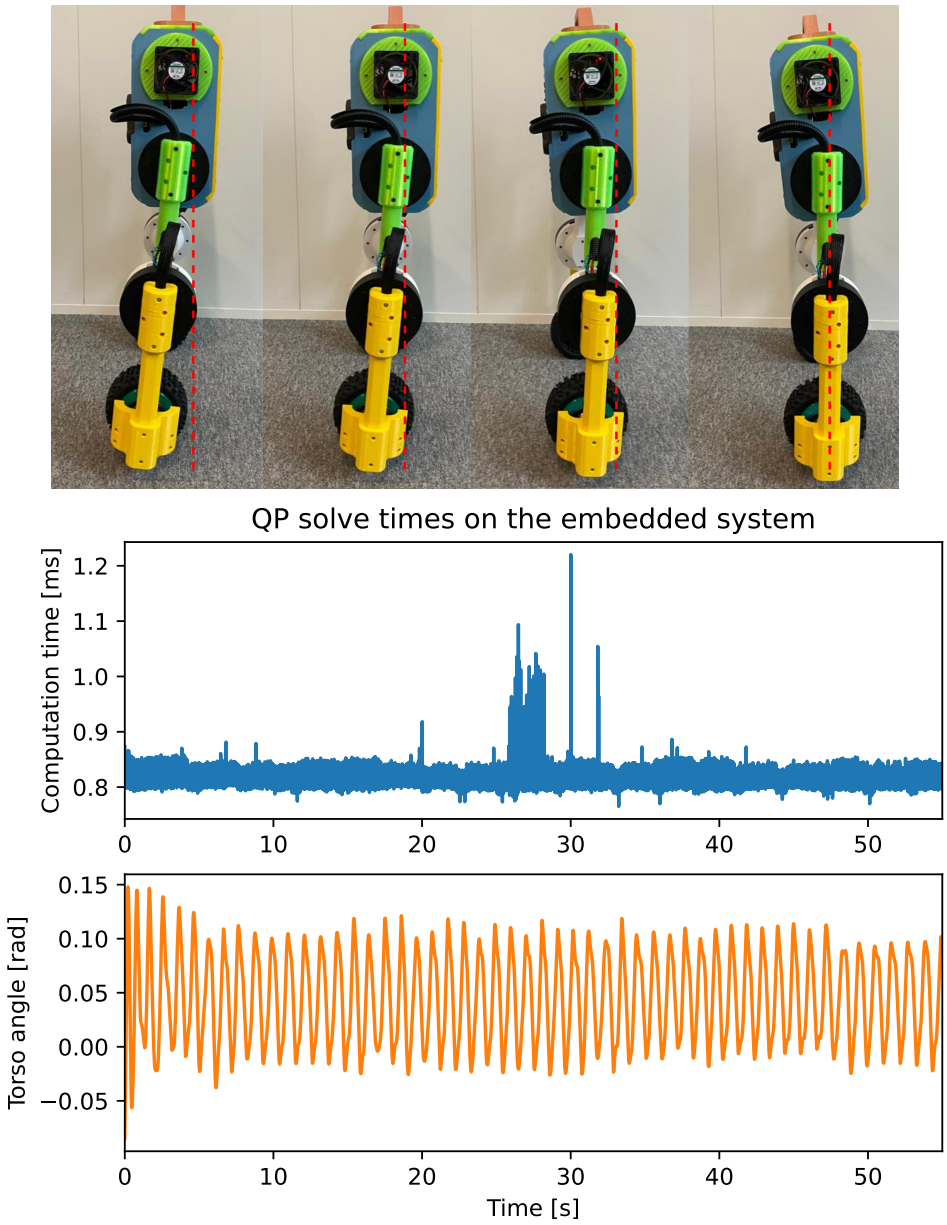

Figure 4.5: Closed-loop model predictive control with ProxQP on the Upkie wheeled biped. The top plot shows the time taken by the hot-started solver to solve MPC problems at each control cycle on the robot's embedded computer. The bottom plot shows the angle between the robot's torso and the vertical at the corresponding times, illustrating how it balances upright.

testbed. Performance profiles correspond to the fraction of problems solved as a function of a certain runtime (measured in terms of a multiple of the runtime of the fastest solver for that problem). Figure 4.6 depicts that even-though the problems are sparse, ProxQP always depicts the best performance profile. Hence it is the fastest approach for solving the problems of this testbed.

Random QPs with finite upper and lower inequality constraints. We generated random QPs with 15% of sparsity for both matrices H and C in the spirit of the benchmark API from OSQP [Stellato et al., 2020, Hermans et al., 2021, Bambade et al., 2022]. The dimension d of those problems is in the range from 10 to 1000. For each dimension, we generated 1000 problem instances with different random seeds and averaged the runtime of each algorithm on 10

Table 4.7: Shifted geometric means (SGM) and failure rates (FR) for solving sparse QPs. The lower the better.

	ProxQP	quadprog	OSQP	qpOASES	SCS	qpSWIFT	MOSEK
SGM	**1**	18.8	2.6	36.7	9.0	261.2	3279.8
FR	0.0%	0.02%	0.0%	0.0%	0.02%	3.31%	25.38%

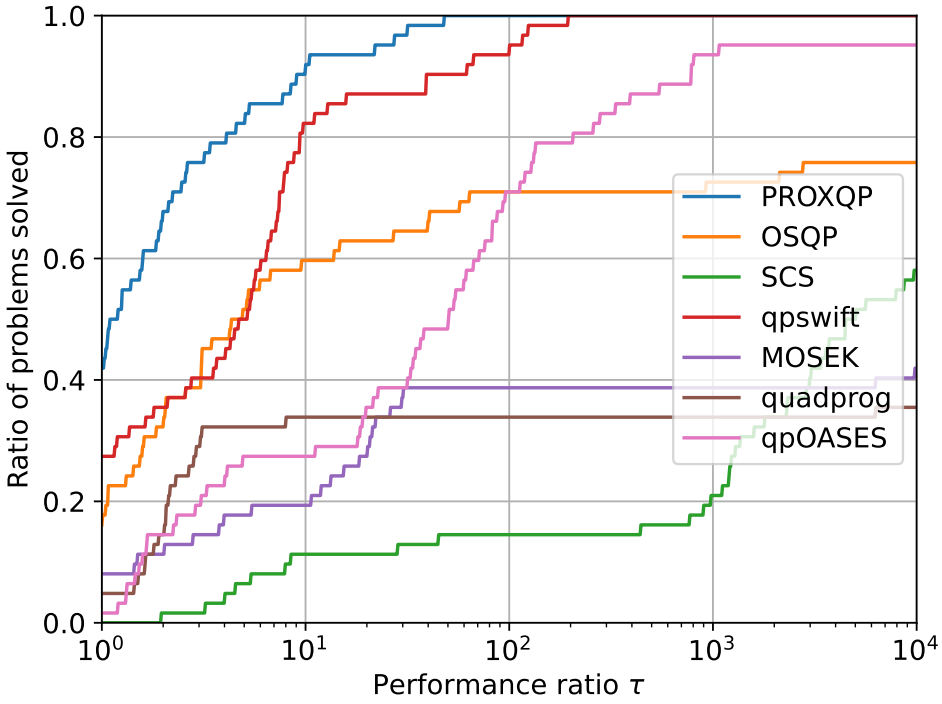

Figure 4.6: Performance profiles on small to medium-sized Maros-Mészáros problems, using a Core i5 - 5300U - 5th Generation @ 2,3 GHz processor. Target accuracy $\epsilon_{abs} = 10^{-6}$ and time limit set to 1000 seconds. The higher the better.

consecutive runs to limit the impact of loading costs. We then report the failure rate and the *shifted geometric means* [Bambade et al., 2022, Section VI-Db] in Table 4.7. ProxQP is about 2.6 times faster across all the problems than OSQP, the second fastest approach. Furthermore, ProxQP, OSQP, and qpOASES exhibit no failure rate.

For the last testbed of experiments, the level of accuracy required for termination is arbitrarily set to $\epsilon_{abs} = 10^{-9}$ (see criterion in (3.10)) so that to show across all these experiments which solvers manage to be the most accurate, the fastest, and the most robust within a large variety of QPs. Furthermore, the qpSWIFT is replaced by the Interior Point solver GUROBI and SCS is removed from the solver testbed.

Remark 4 (Accuracy choice)**.** *Choosing a low ϵ_{abs} has a few non-negligible advantages when comparing solvers. Indeed, the fact a solver might not reach every desired level of accuracy (within the available finite precision limits) is typically due to either (i) an algorithm that has a (very) slow convergence, or (ii) somehow "inappropriate" underlying subroutines (including linear algebra ones) with a limited working precision/stability range, worsening the effect of finite precision. Hence, such low accuracy level reveals which solvers provide on the same time (i) an algorithm whose capabilities allow to reach high accuracy and (ii) a set of underlying numerical routines allowing to reach high precisions in reasonable times.*

Random QPs with finite upper inequality constraints. Concerning random problems, two different levels of sparsity (0.15, 0.5) were used for generating the random matrices of the QPs in the spirit of the benchmark API from OSQP [Stellato et al., 2020]. The dimensions of those problems were picked from $d = 10$ to $d = 1000$. For each set of parameters (sparsity levels and dimensions), we generated 5 problem instances with different random seeds and averaged the running time of each algorithm on 10 consecutive runs (to limit the impact of loading costs). The results are provided in Figure Figure 4.7 using bar plots (including the median, the minimal and maximal execution timings). We also report base statistics in Table 4.8.

In particular, Figure 4.7 shows that even in a sparse configuration (we recall that our solver uses a dense back-end), our solver is around 4 to 5 times faster than OSQP (the second best solver from our test-bed) for examples of dimensions $d \approx 50$ (which is representative of typical robotic applications). When dimension grows to $d \approx 1000$, the speed-up reaches almost an order of magnitude.

The failure rates observed in Table 4.8 for MOSEK and GUROBI solvers come from the fact they are not able to reach the desired precision $\epsilon_{\text{abs}} = 10^{-9}$: solutions proposed are not accurate enough. Consequently, it also impacts their geometric means.

Table 4.8: Shifted geometric means (SGM) and failure rates (FR) for sparse equality and inequality constrained QPs with finite upper bounds and high accuracy target ($\epsilon_{abs} = 10^{-9}$). The lower the better.

	ProxQP	**quadprog**	**OSQP**	**qpOASES**	**GUROBI**	**MOSEK**
SGM	**1**	18.8	4.6	305.6	1146.9	40891.4
FR	0.0%	0.0%	0.0%	0.0%	16.0%	80.0%

Degenerate pure inequality-constrained problems. Figure 4.8 provides the results of our numerical experiments when generating pure inequality-constrained QPs where the matrix H is positive definite but for which LICQ conditions are no longer satisfied (by duplicating the constraints), a common type of degeneracy. We observe from Figure 4.8 that for problems of dimensions $d \approx 50$ our solver is about 3 times faster than OSQP (the second best solver in this set of experiments).

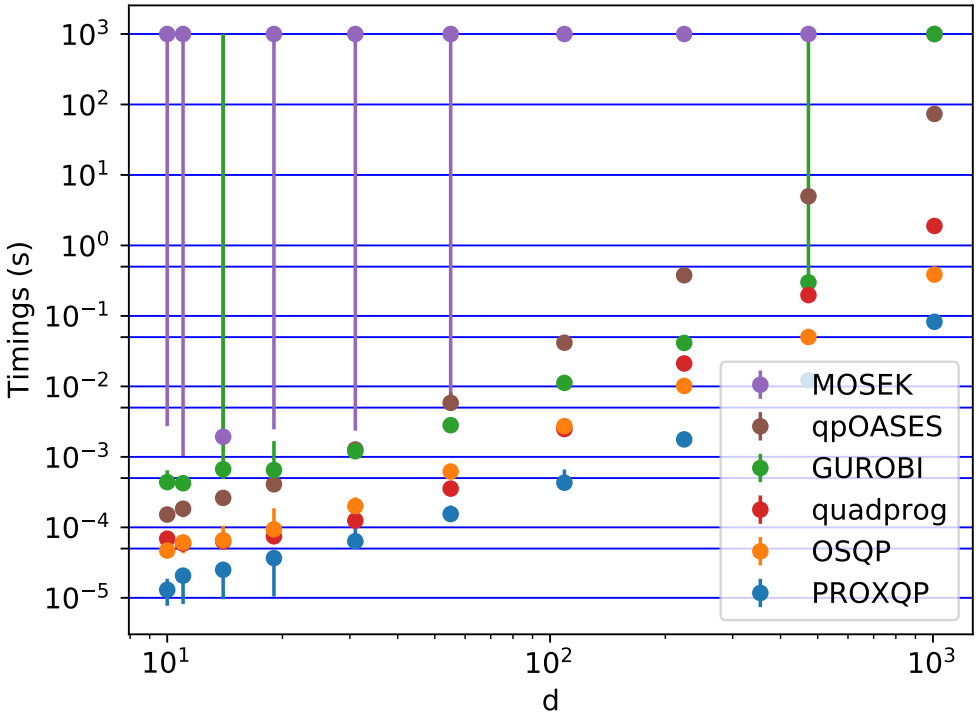

Figure 4.7: Solving times for sparse equality and inequality constrained QPs, using a Core i5 - 5300U - 5th Generation @ 2,3 GHz processor. For each set of executions, the median is shown with a dot.

Table 4.9: Shifted geometric means (SGM) and failure rates (FR) for sparse degenerate pure inequality constrained QPs with finite upper bounds and high accuracy target ($\epsilon_{abs} = 10^{-9}$). The lower the better.

	ProxQP	**quadprog**	**OSQP**	**qpOASES**	**GUROBI**	**MOSEK**
SGM	**1**	3.7	7.1	305.6	461.2	16065.0
FR	0.0%	0.0%	0.0%	0.0%	14%	76.0%

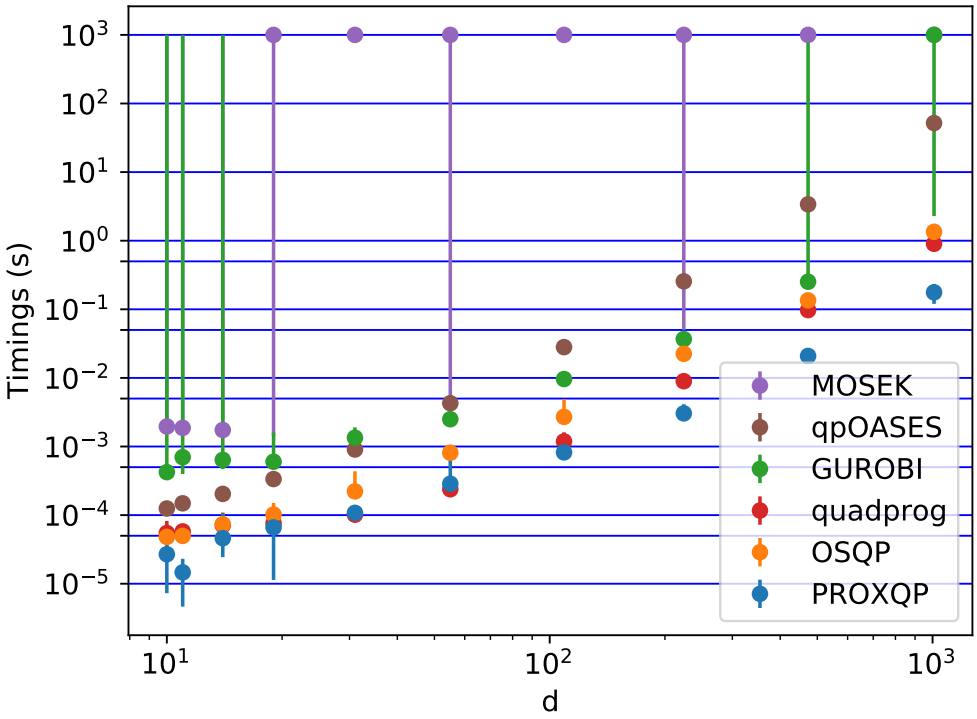

Figure 4.8: Solving times for sparse degenerate pure inequality constrained QPs, using a Core i5 - 5300U - 5th Generation @ 2,3 GHz processor. For each set of executions, the median is shown with a dot.

We report base statistics in Table 4.9. The failure rates observed for MOSEK and GUROBI solvers come from the fact, again, they are not able reaching the desired precision $\epsilon_{\text{abs}} = 10^{-9}$, their outputted solutions being not accurate enough.

Non-strongly convex QPs. As before, in the spirit of the OSQP [Stellato et al., 2020], we generate random QPs for which the Hessian H is not strictly positive definite. We can see in Figure 4.9 that when matrices have 15% of sparsity, OSQP and ProxQP have a similar speed for $d \leq 200$. For higher dimension, we observe that ProxQP is approximately 1.8 times faster than OSQP. When sparsity is about 50%, one can see in Figure 4.10 ProxQP is about 1.8 to 2 times faster for $d \approx 50$. When $d \approx 1000$, ProxQP is about four times faster.

We report base statistics in Table 4.10. The failure rates observed for MOSEK and GUROBI solvers come from the fact, again, their solutions not being precise enough.

Table 4.10: Shifted geometric means (SGM) and failure rates (FR) for sparse pure inequality constrained QPs with non-strictly positive definite Hessian with finite upper bounds and high accuracy target ($\epsilon_{abs} = 10^{-9}$). The lower the better.

	ProxQP	quadprog	OSQP	qpOASES	GUROBI	MOSEK
SGM	**1**	NA	2	70.1	917.2	6446.9
FR	0.0%	100.0%	0.0%	0.0%	34%	72.0%

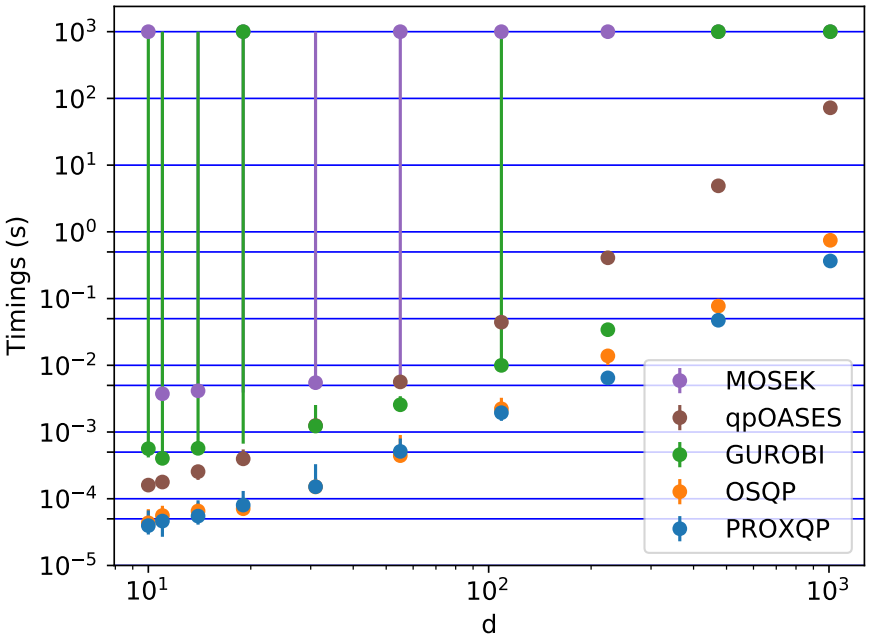

Figure 4.9: Solving times for sparse pure inequality constrained QPs with non-strictly positive definite Hessian, using a Core i5 - 5300U - 5th Generation @ 2,3 GHz processor. For each set of executions, the median is shown with a dot.

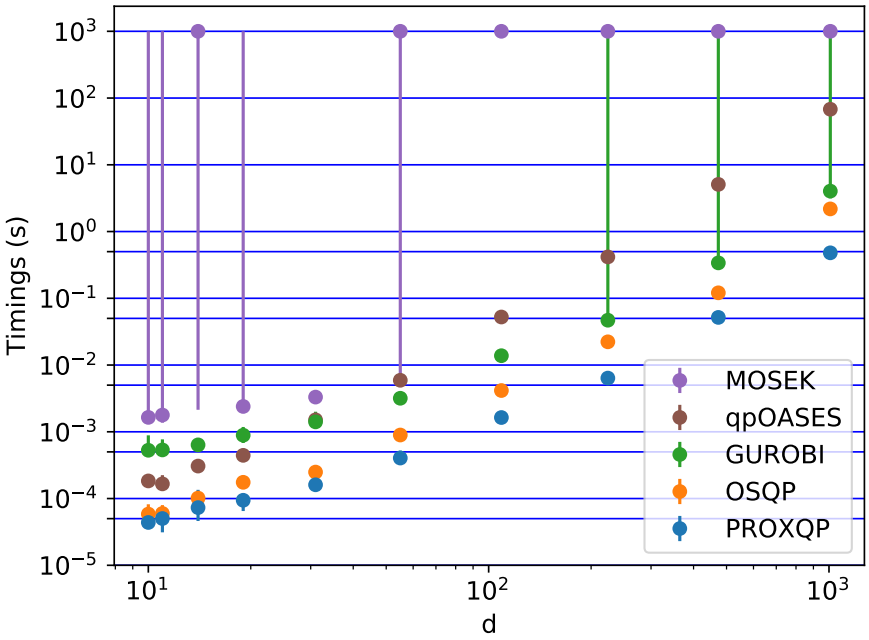

Figure 4.10: Solving times for pure inequality constrained QPs with non-strictly positive definite Hessian and 50% of sparsity, using a Core i5 - 5300U - 5th Generation @ 2,3 GHz processor. For each set of executions, the median is shown with a dot.

Chapter 5

Preliminaries on differentiable optimization for convex QPs

Abstract. *In this chapter, after setting the problem statements and the conditions for being well-posed, we provide a summary of the main algorithmic techniques used for designing efficient QPs layers. More precisely, we review the unrolling methods, implicit differentiation techniques, and the Alternate differentiation approach under the assumption that the solution for a QP is differentiable. For each of the methods reviewed, we propose a summary of a representative algorithm, practical pros and cons, references to existing implementations, and more details about practical implementations. In a second time, we provide a brief overview of the alternative "informative gradient" used by practitioners when a solution for a QP is not differentiable. This introduction is notably based on the reference books [Krantz and Parks, 2002, Dontchev et al., 2009b] and the seminal works by [Amos and Kolter, 2017, Agrawal et al., 2019, Sun et al., 2022, Blondel et al., 2022, Gould et al., 2021, Berthet et al., 2020, Wilder et al., 2019, Mandi and Guns, 2010]. The introduction also mentions other specific references.*

Chapter content

5.1 QP layer: problems statement 79
5.2 QP layers design under differentiability assumptions 81
5.2.1 Unrolled differentiation 81
5.2.2 Implicit differentiation 83
5.2.3 Alternate differentiation 85
5.3 Some alternative "informative gradients" 86
5.3.1 Least square estimate 86
5.3.2 Randomized-smoothing 87
5.3.3 Other regularizations of the cost function 87

5.1 QP layer: problems statement

In this chapter, we consider some loss $\mathcal{L} : \theta \in \mathbb{R}^d \mapsto \mathbb{R}$ which needs to be minimized in order to solve a learning task. We suppose further that there exists some differentiable $h : \mathbb{R}^n \mapsto \mathbb{R}$, such that $\mathcal{L}(\theta) \stackrel{\text{def}}{=} h(x^\star(\theta))$ where $x^\star(\theta)$ must be a solution of a convex QP parameterized by θ

$$x^\star(\theta) \in \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \left\{ f(x;\, \theta) \stackrel{\text{def}}{=} \frac{1}{2} x^\top H(\theta)\, x + x^\top g(\theta) \right\} \quad \text{s.t. } C(\theta)\, x \leq u(\theta), \tag{QP(θ)}$$

with $H(\theta) \in \mathcal{S}_+^n(\mathbb{R})$ a real symmetric positive semi-definite matrix of $\mathbb{R}^{n \times n}$, $g(\theta) \in \mathbb{R}^n$, $C(\theta) \in \mathbb{R}^{m \times n}$ and $u(\theta) \in \mathbb{R}^m$.

Incorporating such an optimization problem as a layer within neural networks has recently become practical and effective for solving certain machine learning tasks, see, for instance [Geng et al., 2020, Amos and Kolter, 2017, Lee et al., 2019, Le Lidec et al., 2021, Donti et al., 2017, de Avila Belbute-Peres et al., 2018, Amos et al., 2018, Bounou et al., 2021]. Such layers allow capturing useful domain-specific knowledge or priors. Unlike conventional neural networks, where the output of each layer is provided by a simple (explicit) function of its input, the input of an optimization layer is the parameter of an optimization problem, and its output is a solution to this problem.

We will denote H, g, C, and u without explicit dependence on θ when this dependence is clear from the context or does not generate any ambiguity. Throughout this chapter, we make the assumption that (QP(θ)) is well-posed: it is bounded below (i.e., dual feasible) and primal feasible (i.e., there exists at least x s.t., $C(\theta)x \leq u(\theta)$). Obtaining a solution $x^\star(\theta)$ for (QP(θ)) is what is commonly referred to as a **forward pass**.

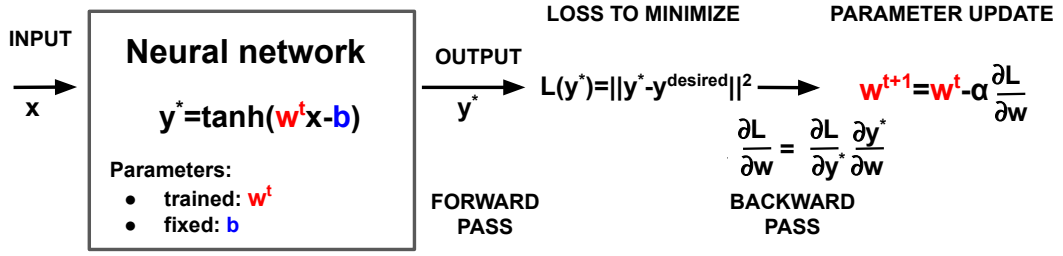

Figure 5.1: Example of a feed-forward neural network.

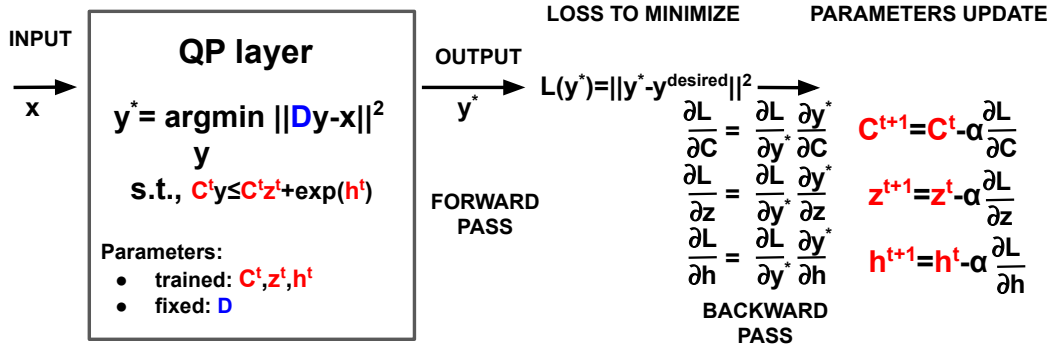

Figure 5.2: Example of a Quadratic Programming layer (with D nonsingular).

Figure 5.1 and Figure 5.2 provide two illustrative examples of a neural network and a QP layer. Both layers have potentially fixed (in blue) and trained (in red) parameters. The main difference is that the output $y^\star$ of the feed-forward neural network has a closed-form expression, whereas the output of the QP layer is the solution of a constrained QP. Finally note that to ensure the quadratic program is always well-posed, extra parameters (i.e., z^t and h^t) are trained.

First problem statement [minimizing $\mathcal{L}$ efficiently under differentiability assumptions]: In current learning pipelines, $\mathcal{L}$ is minimized via Gradient Descent (GD)-based approaches, i.e., it iteratively repeats starting from some $\theta^0 \in \mathbb{R}^d$

$$\theta^{t+1} \leftarrow \theta^t - \alpha \frac{\partial \mathcal{L}(\theta^t)}{\partial \theta}, \tag{5.1}$$

for some step-size $\alpha > 0$, and where $\frac{\partial \mathcal{L}(\theta^t)}{\partial \theta}$ refers to a gradient for $\mathcal{L}$. Furthermore, to efficiently compute at each epoch $\frac{\partial \mathcal{L}(\theta^t)}{\partial \theta}$, learning pipelines exploit the fact that $\mathcal{L}(\theta^t) \in \mathbb{R}$ to backpropagate

derivatives via the chain-rule using automatic differentiation in reverse mode

$$\frac{\partial \mathcal{L}(\theta^t)}{\partial \theta} = \frac{\partial \mathcal{L}(x^\star(\theta^t))}{\partial x} \frac{\partial x^\star(\theta^t)}{\partial \theta}. \tag{5.2}$$

Yet, this holds provided $x^\star(\theta)$ is differentiable with respect to θ, which is challenging for a few reasons. First, there is usually no practical way to compute a closed-form for $x^\star(\theta)$, even when QP(θ) is well-defined. Second, even when such an $x^\star(\theta)$ exists, there is no guarantee for it to be unique nor differentiable –at least in the classical sense– w.r.t. θ (see, e.g., the assumptions of the "classical" implicit function theorem with C^1 functions [Dontchev et al., 2009b, Theorem 1B.1], and extensions for nonsmooth functions [Robinson, 1991, Theorem 3.2], [Bolte et al., 2021, Theorem 2] or [Krantz and Parks, 2002, Section 5.2]).

Thus the first problem statement consists in providing a method for computing efficiently $\frac{\partial \mathcal{L}(\theta^t)}{\partial \theta}$. $\frac{\partial}{\partial \theta}$ should be a chainable notion of derivative which makes the associated GD-based approach (5.1) converge under standard assumptions.

Second problem statement [$\mathcal{L}$ has no differentiability assumptions]: By construction $\mathcal{L}$ is differentiable w.r.t. θ if and only if $x^\star$ is differentiable w.r.t. θ. Hence as soon as $x^\star$ has no notion of differentiability, then $\mathcal{L}$ has none either. In such a setting, the problem statement consists in providing a practical proxy for $\frac{\partial x^\star}{\partial \theta}$ which still feeds the learning pipeline (i.e., the chain rule (5.2) and then the GD-based update (5.1)) with "alternative informative gradients".

In the rest of the chapter, we will briefly review current methods used for addressing the first and second problem statements.

5.2 QP layers design under differentiability assumptions

5.2.1 Unrolled differentiation

A first approach for computing $\frac{\partial x^\star}{\partial \theta}$ consists of unrolling the iterations of the optimization algorithm outputting the last iterate x^k as a proxy for the optimization problem solution $x^\star$ [Domke, 2012, Monga et al., 2021]. This allows the explicit construction of a computational graph relating the algorithm output to the inputs. Then, $\frac{\partial x^\star}{\partial \theta}$ is approximated by $\frac{\partial x^k}{\partial \theta}$, which is obtained via forward or reverse automatic differentiation (see toy examples in Figure 5.3 and Figure 5.4).

This technique works particularly well for unconstrained optimization, where gradient descent is typically used as an optimization into the inference procedure [Belanger and McCallum, 2016, Belanger et al., 2017, Amos et al., 2017, Metz et al., 2019]. It is also very easy to implement within the PyTorch or TensorFlow frameworks, which naturally deal with forward or backward automatic differentiation.

However, this requires reimplementing the algorithm using the automatic differentiation system, and not all algorithms are necessarily autodiff-friendly. Notably, in the context of inequality-constrained QPs, iterative algorithms may use a projection operator that may be difficult to unroll through (e.g., ADMM-based methods). Moreover, if we note $O(M)$ the typical complexity needed for outputting $x^k(\theta)$, then

- Forward-mode autodiff has a time complexity that scales linearly with the number of variables in $\theta \in \mathbb{R}^d$. Hence, it typically has $O(Md)$ time complexity.
- Reverse-mode autodiff has memory complexity that scales linearly with the number of algorithm iterations. Hence, it typically has $O(M)$ time complexity and $O(Md)$ memory complexity.

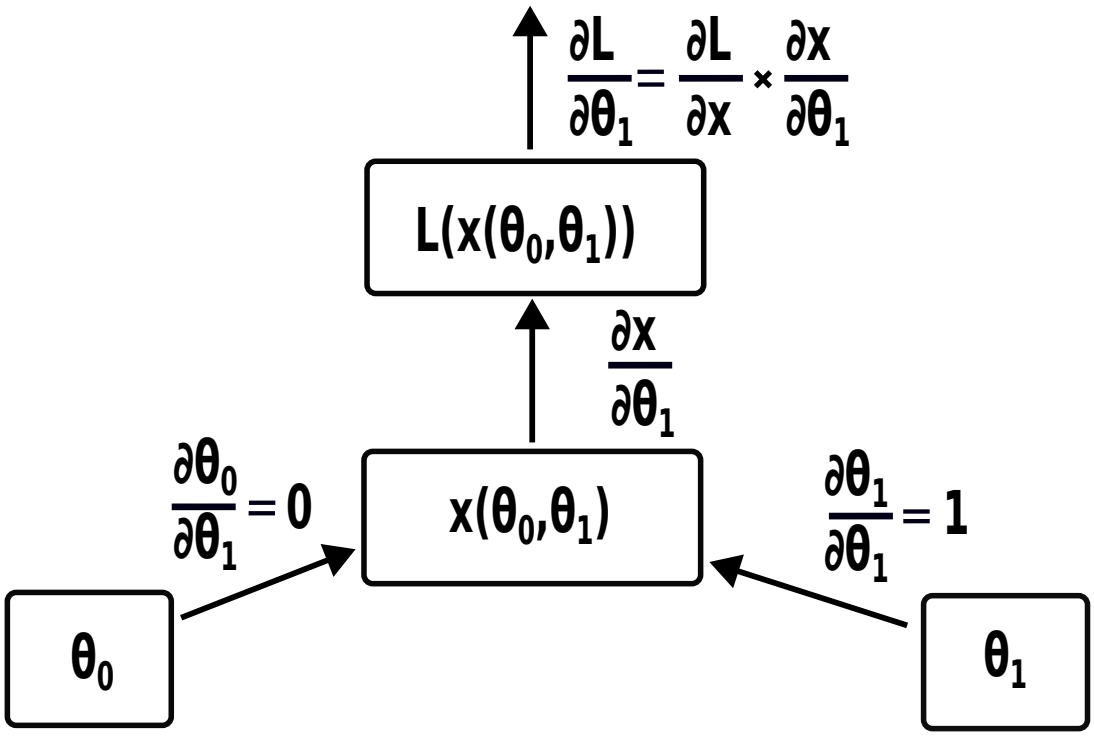

Figure 5.3: Example of a Forward Automatic differentiation scheme.

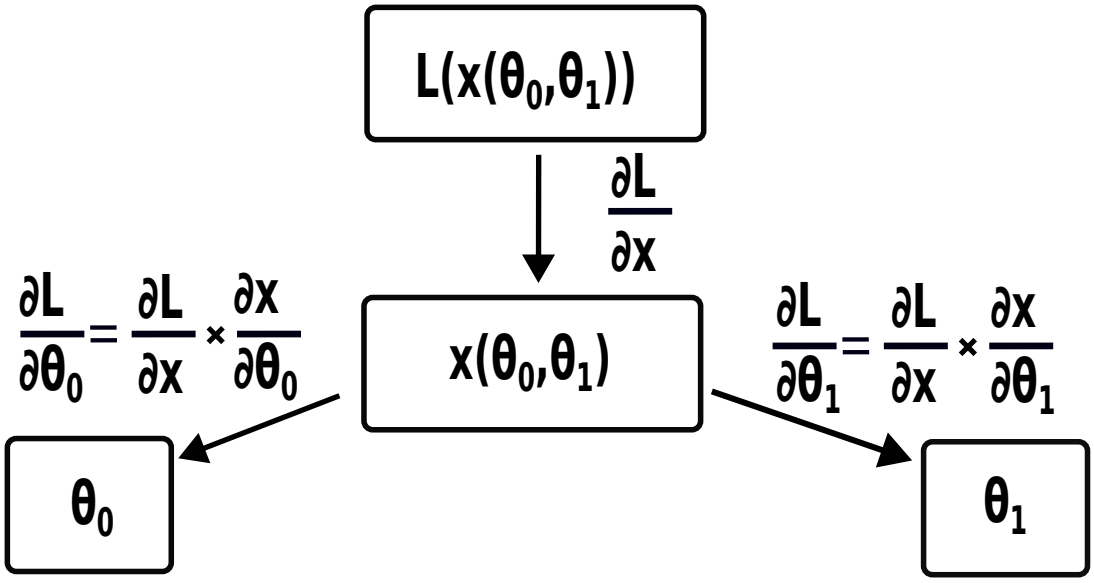

Figure 5.4: Example of a Backward Automatic differentiation scheme.

Finally, recently [Scieur et al., 2022] has highlighted the "curse of unrolling" by showing that for unconstrained quadratic optimization, there is a tradeoff between the convergence speed of the iterates and that of the Jacobian.

We will see that better complexity bounds can be obtained using implicit differentiation in the case of constrained QPs.

5.2.2 Implicit differentiation

A second approach consists of implicitly relating an optimization problem solution to its inputs using optimality conditions, which is possible under certain regularity conditions [Gould et al., 2016, Gilbert, 2021, Fiacco and McCormick, 1968, Robinson, 1980].

In a machine learning context, such implicit differentiation has been used for stationarity conditions [Bengio, 2000, Lorraine et al., 2020], KKT conditions [Chapelle et al., 2002, Gould et al., 2016, Niculae et al., 2018] and the proximal gradient fixed point [Niculae and Blondel, 2017, Bertrand et al., 2020]. An advantage of implicit differentiation is that a solver reimplementation is unnecessary, allowing it to build upon decades of state-of-the-art software. Notably, the OptNet layer [Amos and Kolter, 2017] differentiates through the KKT conditions for (QP(θ)). This technique has been then extended to convex conic problems with CvxpyLayer [Agrawal et al., 2019], and more generally to other types of implicit differentiation problems with JaxOpt framework [Blondel et al., 2022].

In the rest of this section, we detail the techniques used in OptNet and its current limitations for addressing the two problem statements.

The KKT optimality conditions for (QP(θ)) can be written for some $x^\star \in \mathbb{R}^n$ and $z^\star \in \mathbb{R}^m_+$

$$g(x^\star, z^\star, \theta) \stackrel{\text{def}}{=} \begin{bmatrix} H(\theta)x^\star + g(\theta) + C(\theta)^\top z^\star \\ D(z^\star)(C(\theta)x^\star - u(\theta)) \end{bmatrix} = 0, \tag{5.3}$$

where $D(z^\star)$ corresponds to a diagonal matrix formed from vector $z^\star$. If we assume that $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ are continuously differentiable w.r.t. θ, and that the set of points $\{i \in [1, m] | z_i^\star = 0, C(\theta)^\top x^\star = u_i(\theta)\}$ is empty, then the Implicit function theorem (see Theorem 5 from [Dontchev et al., 2009b, Theorem 1B.1]) holds and $\frac{\partial x^\star}{\partial \theta}$ and $\frac{\partial z^\star}{\partial \theta}$ can be found as solution from [Amos and Kolter, 2017, Equation 6]

$$\begin{bmatrix} H & C^\top \\ D(z^\star)C & D(Cx^\star - u) \end{bmatrix} \begin{bmatrix} \frac{\partial x^\star}{\partial \theta} \\ \frac{\partial z^\star}{\partial \theta} \end{bmatrix} = - \begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ D(z^\star)(\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) \end{bmatrix}, \tag{5.4}$$

provided that the matrix involved in this linear system is non-singular.

Theorem 5 (Implicit Function Theorem)**.** *Let $g : \mathbb{R}^{n+m} \times \mathbb{R}^d \to \mathbb{R}^{n+m}$ be continuously differentiable in a neighborhood of $(\bar{w}, \bar{\theta})$ and such that $g(\bar{w}, \bar{\theta}) = 0$, and let the partial Jacobian of g with respect to w at $(\bar{w}, \bar{\theta})$, namely $\frac{\partial g(\bar{w}, \bar{\theta})}{\partial w}$, be non-singular. Then the solution mapping $S(\theta) \stackrel{def}{=} \{w | g(w, \theta) = 0\}$ has a single-valued localization s around $\bar{\theta}$ for $\bar{w}$ which is continuously differentiable in a neighborhood Q of $\bar{\theta}$ with Jacobian satisfying*

$$\frac{\partial s(\theta)}{\partial \theta} = -\frac{\partial g(s(\theta), \theta)}{\partial w}^{-1} \frac{\partial g(s(\theta), \theta)}{\partial \theta}, \forall \theta \in Q. \tag{5.5}$$

Under the assumptions above, $\mathcal{L}$ is locally continuously differentiable with respect to θ. Hence, the chain rule (5.2) holds. [Amos and Kolter, 2017] propose to use the following computational

trick to derive the backward pass efficiently

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \theta} &= \frac{\partial \mathcal{L}}{\partial x^\star}^\top \frac{\partial x^\star}{\partial \theta} + \frac{\partial \mathcal{L}}{\partial z^\star}^\top \frac{\partial z^\star}{\partial \theta} \\
&= -(-\begin{bmatrix} \frac{\delta \mathcal{L}}{\delta x^\star} \\ \frac{\delta \mathcal{L}}{\delta z^\star} \end{bmatrix})^\top \begin{bmatrix} \frac{\partial x^\star}{\partial \theta} \\ \frac{\partial z^\star}{\partial \theta} \end{bmatrix} \\
&= -(\begin{bmatrix} H & C^\top \\ D(z^\star)C & D(Cx^\star - u) \end{bmatrix} \begin{bmatrix} b_1^\star \\ b_2^\star \end{bmatrix})^\top \begin{bmatrix} \frac{\partial x^\star}{\partial \theta} \\ \frac{\partial z^\star}{\partial \theta} \end{bmatrix} \\
&= -\begin{bmatrix} b_1^\star \\ b_2^\star \end{bmatrix}^\top (-\begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ D(z^\star)(\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) \end{bmatrix}) \\
&= (b_1^\star)^\top (\frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star) \\
&+ (b_2^\star)^\top D(z^\star)(\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) \\
&= (b_1^\star)^\top \frac{\partial H}{\partial \theta} x^\star + (\frac{\partial C}{\partial \theta} b_1^\star)^\top z^\star + (b_1)^\top \frac{\partial g}{\partial \theta} \\
&+ (D(z^\star) b_2^\star)^\top (\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}),
\end{aligned}$$

with $b_1^\star$ and $b_2^\star$ the solution of the linear system

$$\begin{bmatrix} H & C^\top D(z^\star) \\ C & D(Cx^\star - u) \end{bmatrix} \begin{bmatrix} b_1^\star \\ b_2^\star \end{bmatrix} = -\begin{bmatrix} \frac{\delta \mathcal{L}}{\delta x^\star} \\ \frac{\delta \mathcal{L}}{\delta z^\star} \end{bmatrix}. \tag{5.6}$$

In practice, since OptNet uses for its forward pass an Interior Point Method, internally they factorize a matrix K of the form

$$K = \begin{bmatrix} H & 0 & C^\top \\ 0 & D(z^\star) & D(Cx^\star - u) \\ C & I & 0 \end{bmatrix},$$

which is symetrized by scaling the second row block by $D(1/[Cx^\star - u])$. Hence, (5.6) can be solved using a unique backsolve from the forward pass by caching the previous factorization of K. Thus, the backward pass essentially amounts to cheap operations compared to the forward pass (linear system solving, matrix multiplication, and scalar products vs. matrix factorization). Compared to unrolling methods, Implicit differentiation for QPs can hence be achieved through only $O(M)$ operations.

Although remarkable for its reduced complexity, the approach proposed in the OptNet framework has a few limitations.

First, it is constrained by some assumptions of Theorem 5. Practically speaking, it requires H to be positive definite, which reduces the range of application of this approach. Note that this approach performs well in practice when (5.6) has a solution. Hence, the nonsingularity of the involved matrix is not in practice necessary (which can happen when C is not full rank).

Secondly, the approach is limited in practice by the KKT formulation used. Indeed, if some solution $x^\star$ lies on the boundary of some constraints, then the symmetrized matrix K formed through the scaling $D(1/[Cx^\star - u])$ has a condition number which blows up. It reduces in practice the applicability of the method.

Third, the method can only learn QP layers for which the primal feasibility is **structurally enforced**. Figure 5.5 provides such an example from [Amos and Kolter, 2017]. In order to learn the equality constraint matrix A during the training process, the authors add a supplementary variable z_0 to ensure that the resulting QP is always primal feasible during the training process.

Yet, other approaches could allow infeasibility during training while driving the layer to be feasible at test time (e.g., A is learned such that some fixed hard constraint b lies in its range space).

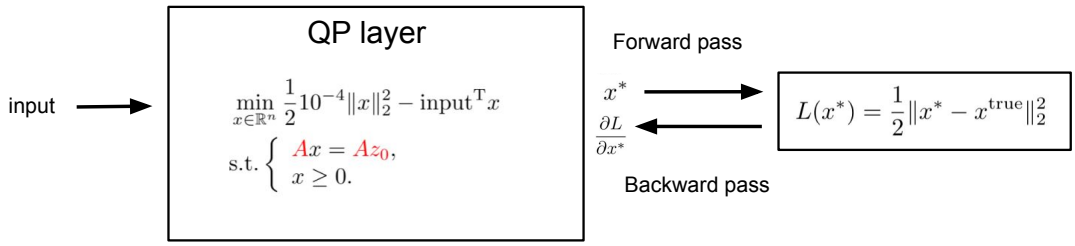

Figure 5.5: Strictly convex QP layer (as in OptNet [Amos and Kolter, 2017]). The constraint matrix A and an extra variable z_0 (strictly positive) are learned to ensure that the QP is always primal feasible (structural feasibility).

Finally, if $x^\star(\theta)$ or $z^\star(\theta)$ do not have any notion of differentiability, then the OptNet layer does not provide any other alternative informative gradients.

5.2.3 Alternate differentiation

H. Sun and co-authors [Sun et al., 2022] have recently proposed an ADMM-type method, called Alt-Diff, to alternatively solve a constrained convex optimization program and obtain approximate Jacobians at the current approximate solutions.

The ADMM steps can be written as

$$x^{k+1} = \arg\min_{x\in\mathbb{R}^n} \mathcal{L}_A(x, s^k, z^k; \theta, \mu), \tag{5.7a}$$

$$s^{k+1} = \arg\min_{s\in\mathbb{R}^m_+} \mathcal{L}_A(x^{k+1}, s, z^k; \theta, \mu), \tag{5.7b}$$

$$z^{k+1} = z^k + \mu(Cx^{k+1} + s^{k+1} - u), \tag{5.7c}$$

where $\mathcal{L}_A$ is the Augmented Lagrangian function

$$\mathcal{L}_A(x, s, z; \theta, \mu) \overset{\text{def}}{=} f(x; \theta) + z^\top(C(\theta)x + s - u(\theta)) + \tfrac{\mu}{2}\|C(\theta)x + s - u(\theta)\|_2^2. \tag{5.8}$$

Note that if $\nabla_x^2\mathcal{L}_A$ is strictly convex, then x^{k+1} is explicitly obtained as

$$x^{k+1} = -(H + \mu C^\top C)^{-1}(g + C^\top z^k + \mu C^\top(s^k - u)). \tag{5.9}$$

Furthermore, s^{k+1} has the closed form solution

$$s^{k+1} = \Pi_+(-\tfrac{1}{\mu} - Cx^{k+1} + u), \tag{5.10}$$

where Π_+ is the projection onto the positive orthant. The function is path differentiable since it is a semi-algebraic and monotone function [Bolte and Pauwels, 2020, Proposition 2(iv)]. Hence, its conservative Jacobian is a "chainable" gradient [Bolte et al., 2021].

Assuming $\nabla_x^2\mathcal{L}_A$ is strictly convex, and $H(\theta)$, $g(\theta)$, $C(\theta)$, $u(\theta)$ are continuously differentiable w.r.t. θ, the Implicit Function Theorem Theorem 5 then holds and can be applied to (5.7a). Hence $\frac{\partial x^{k+1}}{\partial \theta}$ can be derived through

$$\frac{\partial x^{k+1}}{\partial \theta} = -(\nabla_x^2\mathcal{L}_A(x^{k+1}))^{-1}\nabla_{x,\theta}\mathcal{L}_A(x^{k+1}), \tag{5.11}$$

Applying differentiation techniques to (5.7) leads then to [Sun et al., 2022, Equations 7]

$$\frac{\partial x^{k+1}}{\partial \theta} = -(\nabla_x^2 \mathcal{L}_A(x^{k+1}))^{-1} \nabla_{x,\theta} \mathcal{L}_A(x^{k+1}), \tag{5.12a}$$

$$\frac{\partial s^{k+1}}{\partial \theta} = -\frac{1}{\mu} \mathrm{sgn}(s^{k+1}) \cdot 1^\top \odot \left(\frac{\partial z^k}{\partial \theta} + \mu \frac{\partial (Cx^{k+1} - u)}{\partial \theta} \right), \tag{5.12b}$$

$$\frac{\partial z^{k+1}}{\partial \theta} = \frac{\partial z^k}{\partial \theta} + \mu \frac{\partial (Cx^{k+1} + s^{k+1} - u)}{\partial \theta}, \tag{5.12c}$$

Assuming L-smoothness for $\nabla_x^2 f$ and $\nabla_{x,\theta} \mathcal{L}_A$ and strict convexity for $\mathcal{L}_A$, then it has been shown [Sun et al., 2022, Theorem 4.1] that the Alt-Diff method (see Algorithm 10) ensures that $\frac{\partial x^k}{\partial \theta}$ converges towards $\frac{\partial x^\star}{\partial \theta}$ since x^k converges then to $x^\star$ and we have

$$\| \tfrac{\partial x^k}{\partial \theta} - \tfrac{\partial x^\star}{\partial \theta} \| \leq B \| x^k - x^\star \|, \tag{5.13}$$

for some constant $B > 0$.

Algorithm 10: Alt-Diff

Inputs: θ from previous layer, x^0, s^0, z^0
while *termination criterion is satisfied* **do**
  Forward update following (5.7);
  Primal update $\frac{\partial x^{k+1}}{\partial \theta}$ following (5.12a);
  Slack update $\frac{\partial s^{k+1}}{\partial \theta}$ following (5.12b);
  Dual update $\frac{\partial z^{k+1}}{\partial \theta}$ following (5.12c);
  $k \leftarrow k + 1$;
end

Alt-Diff requires to keep in memory previous derivative iterates $\frac{\partial x^k}{\partial \theta}$, $\frac{\partial z^k}{\partial \theta}$ and $\frac{\partial z^k}{\partial \theta}$ which has $O((2n+m)d)$ memory complexity. Furthermore, at each iteration of ADMM, it can require three-dimensional tensor vector operations, such as $\frac{\partial C}{\partial \theta} x^{k+1}$ (which has typically $O(dmn)$ dimension complexity). Thus, if ADMM time complexity $O(M)$ can be decomposed with M_{fact} for the initial factorization complexity and some cost M_{iter} for N typical iteration, then the new total time complexity would be of order: $O(M_{\text{fact}} + N(M_{\text{iter}} + dn(n+m))) = O(M + Ndn(n+m))$. It is, in general, larger than the complexity involved with OptNet, which does not require much for the backsolves of the backward pass (it typically costs $O((n+m)^2)$).

Alt-Diff approach is not guaranteed to reduce the computational burden of differentiable optimization in all cases. Notably, if θ includes differentiation w.r.t. elements of H and C, then $O(d) = O(n(n+m))$ and the approach is then slower. Furthermore, similarly to OptNet, this approach requires strict convexity for f and it does not enable learning specifically C while letting u fixed. Finally, it does not provide alternative informative "gradients" if $x^\star$ or $z^\star$ are not differentiable.

5.3 Some alternative "informative gradients"

In this section, we provide different practical alternative "gradients" proposed in the literature when a function is known to be not everywhere differentiable (i.e., in this setting $\frac{\partial x^\star}{\partial \theta}$). We also add guarantees, if there exist, and practical pros and cons.

5.3.1 Least square estimate

Algebraically speaking, non-differentiability often amounts to trying to solve a linear system involving a singular matrix. Practitioners often use, in such cases, a least-square estimate. In

the context of implicit differentiation, it amounts to solving instead

$$\partial_\theta^{\mathrm{LS}} x^\star, \partial_\theta^{\mathrm{LS}} z^\star \in \underset{\substack{\partial_\theta^{\mathrm{LS}} x \\ \partial_\theta^{\mathrm{LS}} z}}{\arg\min} \left\| \begin{bmatrix} H & C^\top \\ D(z^\star)C & D(Cx^\star - u) \end{bmatrix} \begin{bmatrix} \partial_\theta^{\mathrm{LS}} x \\ \partial_\theta^{\mathrm{LS}} z \end{bmatrix} + \begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ D(z^\star)(\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) \end{bmatrix} \right\|_2^2, \quad (5.14)$$

which always has a solution. The considerable advantage of Least square estimates is two-fold: (i) First, they are easy to implement since a lot of dedicated iterative methods can efficiently solve this problem (e.g., iterative refinement [Parikh and Boyd, 2014], conjugate gradient [Nocedal and Wright, 2006] applied to normal equations, or LSQR [Paige and Saunders, 1982]); (ii) Second, they perform practically well, notably when the linear system has a solution while not being unique (see e.g., the discussion of [Krantz and Parks, 2002, Section 5.4] which notably provides examples with non-differentiable function, singular associated linear systems and still the viable application of the implicit function theorem). This explains why such a heuristic is in practice widely used, notably in frameworks such as CvxpyLayer (see [Agrawal et al., 2019, Appendix B]), JaxOpt [Blondel et al., 2022, Section 2], or [Gould et al., 2021].

5.3.2 Randomized-smoothing

Another possible way to approximate a non-differentiable function by a differentiable one is to convolve it with a smooth function [Vlastelica et al., 2019, Berthet et al., 2020]. In our context, this corresponds to considering

$$x_\sigma^\star(\theta) = \mathbb{E}[x^\star(\theta + \sigma Z)], \quad (5.15)$$

where Z is a random variable with strictly positive and sufficiently differentiable density, and σ is a positive parameter. Notably, it has been shown that for MILPs with unique solutions, when θ corresponds to the linear cost g, then the following holds [Berthet et al., 2020, Proposition 2.3]

$$f(x^\star(\theta)) - f(x_\sigma^\star(\theta)) \leq B'\sigma, \quad (5.16)$$

for some $B' > 0$. Furthermore, the corresponding solution $x_\sigma^\star(\theta)$ is differentiable with respect to θ [Berthet et al., 2020, Proposition 2.2] and can be approximated with $\mathbb{E}[x^\star(\theta + \sigma Z)]$ (i.e., Zero order estimate) or $\mathbb{E}[f(x^\star(\theta + \sigma Z))\nabla_z \nu(Z)/\sigma]$ (i.e., First order estimate) where the distribution must satisfy $\mathrm{d}\mu(z) \sim \exp(-\nu(z))\mathrm{d}z$ with ν being twice differentiable. This strategy implies that following the direction of the perturbed optimizers leads to a solution that is not too far from a solution to the original problem. A temperature level σ essentially controls this distance.

The workaround Randomized Smoothing is relatively recent. Hence, its extensions from MILPs towards different settings, such as QPs, are still ongoing. Furthermore, its applications for control and robotics are relatively new [Le Lidec et al., 2021, Mensch and Blondel, 2018, Suh et al., 2022, Montaut et al., 2023].

5.3.3 Other regularizations of the cost function

Practitioners studying the differentiation of the MILP cost function have also proposed to perturb the cost function by adding another regularization function. We mention two types of techniques.

[Wilder et al., 2019, Blondel et al., 2020b, Martins and Astudillo, 2016] have proposed to consider the following perturbed objective $f_\lambda(x;\theta) \overset{\text{def}}{=} f(x;\theta) - \lambda\|x\|_2^2$ with $\lambda > 0$. Noting $x_\lambda^\star(\theta)$ a minimizer of the new perturbed problem, it then holds that [Wilder et al., 2019, Theorem 2]

$$f(x_\lambda^\star(\theta)) \geq f(x^\star) - \lambda D, \quad (5.17)$$

where $x^\star$ is a solution of the original problem and D is the diameter of the feasible set. Furthermore, $x_\lambda^\star(\theta)$ is differentiable w.r.t. θ almost everywhere [Wilder et al., 2019, Theorem 1].

Another more generic way of regularizing f consists in adding an appropriate smooth convex regularizer Ω, which should be typically $+\infty$ if x is close to zero[1]. [Mandi and Guns, 2010] proposes to consider for example the transformed objective $f_\lambda(x;\theta) \stackrel{\text{def}}{=} f(x;\theta) - \lambda \sum_{i=1}^{n} \ln(x_i)$ with $\lambda > 0$, which naturally fits to Interior Point methods. There are numerous other possible choices, including the family of regularizers forming Fenchel-Young losses (e.g., the Shanon entropy $\Omega(x) = -\sum_{i=1}^{n} x_i \ln(x_i)$) [Blondel et al., 2020a].

[1] Remember that in standard form $x \geq 0$ for linear programs.

Chapter 6

Differentiable Optimization for QPs

Abstract. *In this chapter, we present a unified approach to tackle the differentiability of both feasible and infeasible QPs by notably introducing the notion of* Extended Conservative Jacobian. *We propose efficient ways to compute it in both forward and backward automatic differentiation modes and finally demonstrate how this method enables training a broader range of QP layers. We notably illustrate on standard benchmarks that this technique performs better than traditional approaches for solving some tasks. We also show that using standard learning approaches, our QP layer formulation performs faster than current state-of-the-art QP layers and is also numerically more robust.*

This chapter is based on our work Leveraging augmented-Lagrangian techniques for differentiating over infeasible quadratic programs in machine learning, with Fabian Schramm, Adrien Taylor, and Justin Carpentier, submitted to *International Conference on Learning Representations*, 2024.

Chapter content

6.1 Introduction . . . 90
6.2 The extended conservative Jacobian for convex QPs . . . 91
6.2.1 Problem formulation . . . 92
6.2.2 The closest feasible QP . . . 92
6.2.3 The extended conservative Jacobian . . . 93
6.2.4 Deriving an extended conservative Jacobian . . . 93
6.2.5 Future work and potential improvements . . . 95
6.3 Experiments . . . 96
6.3.1 Pedagogical examples of parametric QPs . . . 96
6.3.2 Learning capabilities . . . 102
6.3.3 Timing benchmarks on standard learning models . . . 104
6.3.4 Training with large learning rates . . . 107
6.4 Proofs . . . 112
6.4.1 Proof of Lemma 5 . . . 112
6.4.2 Proof of Lemma 6 . . . 113
6.4.3 Forward and backward AD for computing ECJs . . . 114
6.4.4 Proof of Lemma 7 . . . 119
6.4.5 Experimental setting . . . 120

6.1 Introduction

Incorporating differentiable optimization problems as layers within neural networks has recently become practical and effective for solving certain machine learning tasks, see, for instance [Geng et al., 2020, Amos and Kolter, 2017, Lee et al., 2019, Le Lidec et al., 2021, Donti et al., 2017, de Avila Belbute-Peres et al., 2018, Amos et al., 2018, Bounou et al., 2021]. Such layers allow capturing useful domain-specific knowledge or priors. Unlike conventional neural networks, where the output of each layer is provided by a simple (explicit) function of its input, the input of an optimization layer is the parameter of an optimization problem, and its output is a solution to this problem. Figure 6.1 and Figure 6.2 provide two illustrative examples of a neural network and a QP layer. Both layers have potentially fixed (in blue) and trained (in red) parameters. The main difference is that the output of the feed-forward neural network has a closed-form expression, whereas the output of the QP layer is the solution of a constrained QP. Finally, note that extra parameters (i.e., z^t and h^t) are trained to ensure the quadratic program is always well-posed during training.

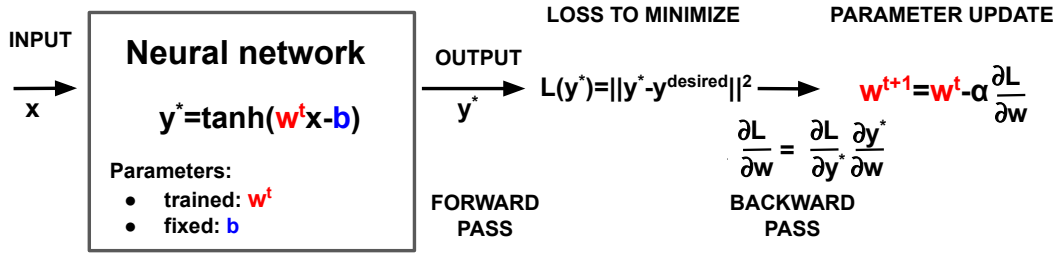

Figure 6.1: Example of a feed-forward neural network.

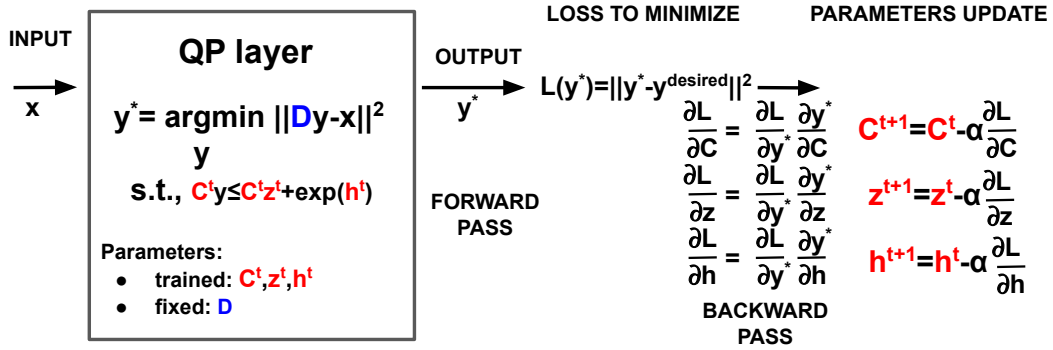

Figure 6.2: Example of a Quadratic Programming layer (with D nonsingular).

In this work, we focus on convex Quadratic Programming (QP) layers, a specific type of optimization layer that offers a rich modeling power [Amos and Kolter, 2017, Section 3.2]. A convex QP parameterized by θ is defined as follows

$$x^\star(\theta) \in \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \left\{ f(x;\, \theta) \stackrel{\text{def}}{=} \frac{1}{2} x^\top H(\theta)\, x + x^\top g(\theta) \right\} \quad \text{s.t. } C(\theta)\, x \leq u(\theta), \tag{QP(θ)}$$

where $H(\theta) \in \mathcal{S}_+^n(\mathbb{R})$ is a real symmetric positive semi-definite matrix of $\mathbb{R}^{n \times n}$, $g(\theta) \in \mathbb{R}^n$, $C(\theta) \in \mathbb{R}^{m \times n}$ and $u(\theta) \in \mathbb{R}^m$. n is the problem dimension, while m is the number of inequality constraints. We will abusively denote H, g, C, and u without explicit dependence on θ when this dependence is clear from the context or generates no ambiguity. In order to use QP(θ) as a learning tool that can be trained with standard optimization techniques, we need to be able to differentiate $x^\star(\theta)$ w.r.t. θ, which is challenging for a few reasons. First, there is usually no

practical way to compute a closed-form for $x^\star(\theta)$, even when QP(θ) is well-defined. Second, even when such an $x^\star(\theta)$ exists, there is no guarantee for it to be unique nor differentiable w.r.t. θ (see, e.g., the assumptions of the implicit function theorem [Dontchev et al., 2009b, Theorem 1B.1]). Consequently, concurrent approaches are generally based on architectures enforcing the satisfaction of some strong assumptions. In particular, to the best of our knowledge, previous approaches enforce the primal feasibility of the layer during training, which generally requires additional learning variables and limits the modeling power of those layers. For instance, as in [Amos and Kolter, 2017], learning QP(θ) requires imposing its feasible set to be non-empty. For imposing this while learning C, the authors also learn $z_0 \in \mathbb{R}^n$ and $s_0 \in \mathbb{R}^m_+$ and u of the form $u = Cz_0 + s_0$, thereby preventing, among others, u from being fixed independently of the learning.

This work makes the following contributions:

- We propose a unified approach to tackle the differentiability of both feasible and infeasible QPs. The main idea consists of extending the definition of $x^\star(\theta)$ to be either a solution to QP(θ) when it is feasible or a solution of the closest feasible QP (in the least-square sense) when it is not. By relying on the notion of a conservative Jacobian by [Bolte et al., 2021, Bolte and Pauwels, 2020], we notably show that the KKT map G of this extended problem is path differentiable w.r.t. θ and $x^\star$ (Section 6.2.2). In this context, the Jacobian $\frac{\partial x^\star(\theta)}{\partial \theta}$ is defined as the least-square solution of the linear system formed by applying the implicit function theorem to G (Section 6.2.3). We show that this definition consistently covers the differentiability of feasible QPs as with the traditional implicit differentiation [Amos and Kolter, 2017] when it is valid and with the least-square estimate proposed by [Agrawal et al., 2019, Appendix B] otherwise.

- In Section 6.2.4 we provide an efficient way to compute the Jacobian $\frac{\partial x^\star(\theta)}{\partial \theta}$ in both forward and backward automatic differentiation modes.

- In Section 6.3 we demonstrate how the approach enables dealing with possibly infeasible QP(θ), allowing to train a broader range of QP layers (e.g., learning QPs that are not generically feasible during training). More precisely, we will show how training towards feasibility at test time the QP layer provided in Figure 6.3.

Based on these developments, we provide QPLayer, an open-source implementation with efficient forward and backward passes. It takes advantage, among others, of recent advances in solving of QP problems to output in the forward pass the closest feasible QP solution in ℓ_2-sense as soon as the program is primal infeasible [Chiche and Gilbert, 2016]. Section 6.3 highlights for different learning tasks the numerical robustness, accuracy, and speed of our approach against other state-of-the-art methods.

6.2 The extended conservative Jacobian for convex QPs

This section introduces the main contribution of this work: an extended conservative Jacobian for the solutions to QP(θ) allowing to simultaneously deal with feasible and infeasible QPs, as provided in Section 6.2.2 and Section 6.2.3. Section 6.2.4 proposes efficient algorithms for computing them in forward and backward modes. For exposition purposes, Section 6.3.1 illustrates the concepts on a few simple examples.

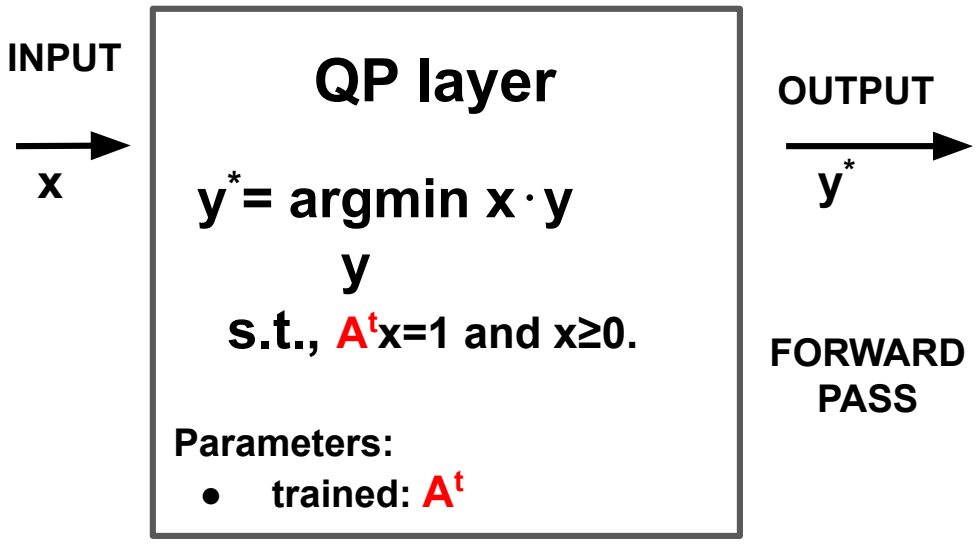

Figure 6.3: A Linear Programming layer. Nothing guarantees during training that the constrained vector of 1 *lies in the range space of the trained matrix* A^t*. Our approach enables to train* A^t *such that at test time the LP is feasible.*

6.2.1 Problem formulation

For differentiating QPs, we solve a hierarchic problem QP-H(θ) which is equivalent to QP(θ) when QP(θ) is primal feasible (i.e., there exists x s.t. $C(\theta)x \leq u(\theta)$)

$$\begin{aligned} s^\star(\theta) = \arg \min_{s\in\mathbb{R}^{n_i}} \ & \tfrac{1}{2}\|s\|_2^2 \\ \text{s.t. } & x^\star(\theta), z^\star(\theta) \in \arg \min_{x\in\mathbb{R}^n} \max_{z\in\mathbb{R}_+^{n_i}} L(x,z,s;\theta), \end{aligned} \tag{QP-H(θ))}$$

with $L(x,z,s;\theta) \stackrel{\text{def}}{=} \frac{1}{2}x^\top H(\theta)\, x + x^\top g(\theta) + z^\top(C(\theta)x - u(\theta) - s)$ (namely the Lagrangian of QP(θ) augmented with a slack variable s). The following assumption is necessary and sufficient for guaranteeing QP-H(θ) to have a solution. In this situation, QP-H(θ) is therefore well-posed and $s^\star$ is referred to as the optimal shift. It provides a measure of the distance of QP(θ) to be primal infeasible in ℓ_2-sense (hence $s^\star = 0$ iff QP(θ) is feasible).

Assumption 3. *$H(\theta)$ is symmetric positive definite in the direction of $g(\theta)$ or $g(\theta)$ is orthogonal to the recession cone of QP(θ), i.e., $g(\theta) \perp \mathcal{C}^\infty(\theta) \stackrel{def}{=} \{y \in \mathbb{R}^n | C(\theta)[x+\tau y] \leq u(\theta)$ s.t. $C(\theta)x \leq u(\theta), \tau \geq 0\}$.*

The existence of a solution $(x^\star(\theta), z^\star(\theta), s^\star(\theta))$ is also equivalent to the dual of QP(θ) having a non-empty domain (i.e., being proper), see [Chiche and Gilbert, 2016, Assumption 2.6 and Proposition 2.5]). So, the approach proposed here allows differentiating through dual feasible convex QPs.

6.2.2 The closest feasible QP

In what follows, we deal with QP-H(θ) via a nonlinear map G:

$$G(x,z,t;\ \theta) := \begin{bmatrix} H(\theta)x + g(\theta) + C(\theta)^\top z \\ C(\theta)x - u(\theta) - t \\ [[t]_- + z]_+ - z \\ C(\theta)^\top [t]_+ \end{bmatrix}, \tag{G}$$

where $[.]_+$ and $[.]_-$ respectively correspond to component-wise projections on the non-negative and non-positive orthants. The following lemma guarantees solutions to QP-H(θ) to be zeros of G (see proof in Section 6.4.1).

Lemma 5. *Let $H(\theta) \in \mathcal{S}_+^n(\mathbb{R})$, $g(\theta) \in \mathbb{R}^n$, $C(\theta) \in \mathbb{R}^{n_i \times n}$ and $u(\theta) \in \mathbb{R}^{n_i}$ be satisfying Assumption 3. It holds that $(x^\star, z^\star, s^\star)$ solves QP-H(θ) iff there exists $t^\star \in \mathbb{R}^{n_i}$ s.t. $G(x^\star, z^\star, t^\star;\ \theta) = 0$ and $s^\star = [t^\star]_+$.*

6.2.3 The extended conservative Jacobian

For differentiating through G, we rely on the notion of extended conservative Jacobian (ECJ). As provided by the following lemma, the nonlinear map $G(x, z, t;\ \theta)$ is path differentiable (see [Bolte and Pauwels, 2020, Definition 3]) w.r.t. x, z, t and also w.r.t. θ under the assumption that $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ are differentiable w.r.t. θ. This lemma is proved in Section 6.4.2.

Lemma 6. *G is path differentiable w.r.t. $x^\star$, $z^\star$ and $t^\star$. Furthermore, if $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ are differentiable w.r.t. θ, then G is path differentiable w.r.t. θ.*

Definition 6. *Let $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ be differentiable w.r.t. θ and satisfying Assumption 3. Let $v^\star = (x^\star, z^\star, t^\star) \in \mathbb{R}^n \times \mathbb{R}_+^{n_i} \times \mathbb{R}^{n_i}$ s.t. $G(x^\star, z^\star, t^\star;\ \theta) = 0$. We refer to the ECJs of $x^\star$, $z^\star$ and $t^\star$, respectively denoted by $\frac{\partial x^\star}{\partial \theta}, \frac{\partial z^\star}{\partial \theta}$ and $\frac{\partial t^\star}{\partial \theta}$, as solutions of the following problem*

$$\left(\frac{\partial x^\star}{\partial \theta}, \frac{\partial z^\star}{\partial \theta}, \frac{\partial t^\star}{\partial \theta}\right) \in \arg\min_w \left\| \frac{\partial G(x^\star, z^\star, t^\star;\ \theta)}{\partial v^\star} w + \frac{\partial G(x^\star, z^\star, t^\star;\ \theta)}{\partial \theta} \right\|_2^2. \tag{6.1}$$

Furthermore, we refer to an ECJ of $s^\star = [t^\star]_+$, denoted by $\frac{\partial s^\star}{\partial \theta}$, any element satisfying $\Pi \frac{\partial t^\star}{\partial \theta} \in \frac{\partial s^\star}{\partial \theta}$, with $\Pi \in \partial([.]_+)(t^\star)$ a subgradient of the positive orthant evaluated in $t^\star$.

As shown in the next section the ECJs match the definitions of standard Jacobians under standard assumptions guaranteeing differentiability (when the QP is feasible), as provided by [Amos and Kolter, 2017,Dontchev et al., 2009b]. When the QP is feasible but not differentiable, the ECJ corresponds to a least-square approximation specialized for QPs. A similar practical least-square estimate was proposed in [Agrawal et al., 2019, Appendix B] for differentiating primal solutions of second-order cones (SOCs)[1]. [Blondel et al., 2022, Section 2.1] proposed a similar estimate.

6.2.4 Deriving an extended conservative Jacobian

This section derives an ECJ and incorporates it in a backpropagation algorithm. It also shows how to efficiently compute this ECJ under primal feasibility.

General case: dealing with both feasible and infeasible QPs

In the following, we provide forward and backward pass algorithms to compute ECJs for both feasible and infeasible QPs.

Forward pass: Let $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ be differentiable w.r.t. θ and satisfying Assumption 3, and $x^\star, z^\star, t^\star$ s.t. $G(x^\star, z^\star, t^\star;\ \theta) = 0$. We show in Section 6.4.3 that we can efficiently

[1] More precisely, [Agrawal et al., 2019, Appendix B] relies on a series of assumptions allowing to simplify the computations. In particular, they assume that $\frac{\partial z^\star}{\partial \theta} = 0$ and $\frac{\partial t^\star}{\partial \theta} = 0$, where $t^\star$ is a slack variable.

derive ECJs of $x^\star$, $z^\star$ and $t^\star$ by solving the following QP using an augmented Lagrangian-based algorithm [Rockafellar, 1976a]

$$\frac{\partial x^\star}{\partial \theta}, \frac{\partial z^\star}{\partial \theta}, \frac{\partial t^\star}{\partial \theta} \in \underset{\frac{\partial x}{\partial \theta}, \frac{\partial z}{\partial \theta}, \frac{\partial t}{\partial \theta}}{\arg\min} \left\| \begin{bmatrix} H & C^\top & 0 \\ C & 0 & -I \\ 0 & \Pi_1 - I & \Pi_1\Pi_2 \\ 0 & 0 & C^\top(I-\Pi_2) \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \theta} \\ \frac{\partial z}{\partial \theta} \\ \frac{\partial t}{\partial \theta} \end{bmatrix} + \begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta} \\ 0 \\ \frac{\partial C^\top}{\partial \theta}[t^\star]_+ \end{bmatrix} \right\|_2^2, \tag{6.2}$$

where Π_1 and Π_2 are binary diagonal matrices respectively corresponding to the subdifferentials $\partial([.]_+)([t^\star]_- + z^\star)$ and $\partial([.]_-)(t^\star)$, with the following specific choices in zeros

$$(\Pi_1)_i = 0 \text{ when } [t_i^\star]_- + z_i^\star = 0, \qquad (\Pi_2)_i = 1 \text{ when } t_i^\star = 0. \tag{6.3}$$

Furthermore, an ECJ of $s^\star$ can be obtained via $(1-\Pi_2)\frac{\partial t^\star}{\partial \theta} \in \frac{\partial s^\star}{\partial \theta}$.

As $s^\star$ is a direct output of an augmented Lagrangian-based algorithm [Chiche and Gilbert, 2016], in what follows, we work with $s^\star$ instead of $t^\star$.

Backward pass: Let $h : \mathbb{R}^n \times (\mathbb{R}^{n_i})^2 \to \mathbb{R}$ be a differentiable function, and let $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ be differentiable w.r.t. θ and satisfying Assumption 3. Then, denoting $\mathcal{L}(\theta) \triangleq h(x^\star(\theta), z^\star(\theta), s^\star(\theta))$ and following the methodology provided in [Amos and Kolter, 2017, Section 3], we show in Section 6.4.3 that under assumptions of **??** a conservative Jacobian $\frac{\partial \mathcal{L}}{\partial \theta}$ can be obtained from the usual chain rule:

$$\frac{\partial \mathcal{L}}{\partial \theta} = (b_1^\star)^\top \frac{\partial H}{\partial \theta} x^\star + (b_1^\star)^\top \frac{\partial g}{\partial \theta} + (b_2^\star)^\top \frac{\partial C}{\partial \theta} x^\star + (z^\star)^\top \frac{\partial C}{\partial \theta} b_1^\star + (s^\star)^\top \frac{\partial C}{\partial \theta} b_4^\star - (b_2^\star)^\top \frac{\partial u}{\partial \theta}, \tag{6.4}$$

where $b_1^\star$, $b_2^\star$, $b_3^\star$ and $b_4^\star$ are solutions of the linear system

$$\begin{bmatrix} H & C^\top & 0 & 0 \\ C & 0 & (I-\Pi_1) & 0 \\ 0 & -I & -\Pi_1\Pi_2 & (1-\Pi_2)C \end{bmatrix} \begin{bmatrix} b_1^\star \\ b_2^\star \\ b_3^\star \\ b_4^\star \end{bmatrix} = - \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial x^\star} \\ \frac{\partial \mathcal{L}}{\partial z^\star} \\ \frac{\partial \mathcal{L}}{\partial s^\star} \end{bmatrix}. \tag{6.5}$$

If not, the methodology proposed in [Amos and Kolter, 2017, Section 3] cannot be used. Hence, ECJs of $x^\star, z^\star$ and $s^\star$ are derived using forward mode and $\frac{\partial \mathcal{L}}{\partial \theta}$ is recovered from

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial x^\star}^\top \frac{\partial x^\star}{\partial \theta} + \frac{\partial \mathcal{L}}{\partial z^\star}^\top \frac{\partial z^\star}{\partial \theta} + \frac{\partial \mathcal{L}}{\partial s^\star}^\top \frac{\partial s^\star}{\partial \theta}. \tag{6.6}$$

In practice, we provide a feasibility tolerance that can be set by the user and above which (6.5) is considered not accurate enough or infeasible. In such cases, the forward mode is used internally.

Exploiting primal feasibility of the QP

In this section, we exploit feasibility of the QP for simplifying the computations. First, for the forward pass, the QP needs only to be feasible for the value of θ under consideration. For the backward pass, we exploit the standard assumption (see [Amos and Kolter, 2017]) of the QP being constructively feasible for all values of θ (which is of course restrictive, but which can be exploited for efficiency).

Forward pass: When QP(θ) is feasible and $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ are differentiable w.r.t. θ and satisfy Assumption 3, we show in Lemma 6.4.3 that ECJs can be obtained as a solution to the simpler:

$$\begin{aligned}\frac{\partial x^\star}{\partial \theta}, \frac{\partial z^\star}{\partial \theta} &\in \underset{\frac{\partial x}{\partial \theta}, \frac{\partial z}{\partial \theta}}{\arg\min} \left\| \begin{bmatrix} H & C^\top \\ \frac{\Pi_1}{\sqrt{1+\Pi_1}} C & \Pi_1 - I \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \theta} \\ \frac{\partial z}{\partial \theta} \end{bmatrix} + \begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\Pi_1}{\sqrt{1+\Pi_1}} \left(\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta} \right) \end{bmatrix} \right\|_2^2, \\ \frac{\partial t^\star}{\partial \theta} &= (I + \Pi_1)^{-1} \left(C \frac{\partial x^\star}{\partial \theta} + \frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta} \right),\end{aligned} \tag{6.7}$$

where Π_1 is a binary diagonal matrices representing the subdifferential $\partial[.]_+(Cx^\star - u + z^\star)$ with the following specific choice:

$$(\Pi_1)_i = 0 \text{ when } C_i x^\star - u_i + z_i^\star = 0. \tag{6.8}$$

The following lemma (see proof in Equation 6.4.3) guarantees that, under standard assumptions, solutions to (6.7) correspond to standard Jacobians (see, e.g., [Amos and Kolter, 2017]).

Lemma 7. *If QP(θ) is feasible and $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ are differentiable w.r.t. θ and satisfy Assumption 3, and if the KKT matrix of active constraints is nonsingular and $x^\star$, $z^\star$ satisfy strict complementarity, then the ECJs matches the standard Jacobian, i.e., $\frac{\partial x^\star(\theta)}{\partial \theta} = \nabla x^\star(\theta)$ and $\frac{\partial z^\star(\theta)}{\partial \theta} = \nabla z^\star(\theta)$.*

Backward pass: If QP(θ) is by construction primal feasible for any θ, then for any θ, $s^\star(\theta) = 0$. We can exploit this result for considering simpler losses not depending of $s^\star(\theta)$. More precisely, let $h : \mathbb{R}^n \times (\mathbb{R}^{n_i}) \to \mathbb{R}$ be a differentiable function, and let $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ be differentiable w.r.t. θ and satisfying Assumption 3. Then, denoting $\mathcal{L}(\theta) \triangleq h(x^\star(\theta), z^\star(\theta))$, we show in Equation 6.4.3 that when assumptions of Lemma 7 hold the backward pass can be evaluated by solving the following linear system

$$\begin{bmatrix} H & C_J^\top \\ C_J & 0 \end{bmatrix} \begin{bmatrix} b_x^\star \\ b_{z_J}^\star \end{bmatrix} = - \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial x^\star} \\ \frac{\partial \mathcal{L}}{\partial z_J^\star} \end{bmatrix}, \qquad b_{z_{J^c}}^\star = \frac{\partial \mathcal{L}}{\partial z_{J^c}^\star}, \tag{6.9}$$

where J is the set of constraints for which $(\Pi_1)_i = 1$ and J^c the one for which $(\Pi_1)_i = 0$. $\frac{\partial \mathcal{L}}{\partial \theta}$ is then retrieved from the chain rule

$$\frac{\partial \mathcal{L}}{\partial \theta} = (b_x^\star)^\top \frac{\partial H}{\partial \theta} x^\star + (b_x^\star)^\top \frac{\partial g}{\partial \theta} + (\Pi_1 b_z^\star)^\top \frac{\partial C}{\partial \theta} x^\star + (z^\star)^\top \frac{\partial C}{\partial \theta} b_x^\star - (\Pi_1 b_z^\star)^\top \frac{\partial u}{\partial \theta}. \tag{6.10}$$

Note that the assumptions of Lemma 7 are not necessarily met. Therefore, if (6.9) is found to be infeasible we provide an option to use forward mode in the backward pass as in the generic case (see Equation 6.2.4). Such infeasibility can be detected easily using iterative refinement [Parikh and Boyd, 2014, Section 4.1.2]) as it converges to the least-square solution of (6.9) in the infeasible case (see [Güler, 1991, Theorem 2.3]).

6.2.5 Future work and potential improvements

Before moving to the experiments, let us mention a few potential directions for future work and improvements in our approach. There remain a few gaps in the theoretical foundations of our methodology, which we believe should be handled in the future. Indeed, we have not proved that we could apply the chain rule to ECJs in the general case in the spirit of CJs (see [Bolte et al., 2021]). Also, it should be confirmed that ECJ indeed reduces to CJ under relatively weak assumptions. While those problems are present in most frameworks [Agrawal et al., 2019, Section B], [Blondel et al., 2022, Section 2.1], using the least-square estimate provides good practical results when non-differentiability occurs.

6.3 Experiments

Our forward and backward mode differentiation of convex QP layers has been implemented in C++. We refer to it as QPLayer in what follows. Our code leverages the primal-dual augmented Lagrangian solver ProxQP, as its internal QP solver. This section illustrates through different classic learning tasks that QPLayer can drive towards feasibility infeasible QPs. We show in an example that it enables learning new types of layers with an improved prediction rate. Then we illustrate that when using standard layers (enforcing feasibility), QPLayer is also faster than other state-of-the-art approaches. QPLayer further allows also relaxing primal feasibility constraints, thereby enabling the training of simplified layers. We finally also show in Section 6.3.4 that QPLayer can be used for solving tasks with high learning rates.

6.3.1 Pedagogical examples of parametric QPs

A few numerical examples illustrate the concept of ECJ in different simple scenarios. The first example corresponds to a strictly convex parametric QP which can be either feasible or infeasible. In this example, a linear constraint depends on a parameter θ. Depending on the value of this parameter, the QP can either be feasible or infeasible.

The second example is a strictly convex QP with a parameterized linear cost. This problem is always feasible.

The last example is a parametric LP with possibly multiple solutions. For appropriate values of the parameters, the LP is feasible but not differentiable.

Strictly convex QP (parameterized constraints)

Consider the following strictly convex QP parameterized by a scalar value θ

$$\begin{aligned} x^\star(\theta) = \underset{x_1,x_2\in\mathbb{R}^2}{\arg\min} & \frac{1}{2}(x_1^2 + x_2^2) \\ \text{s.t. } & \theta \leq x_1 + x_2 \leq 1.55, \\ & 1.5 \leq 2x_1 + x_2 \leq 1.55 \end{aligned} \tag{6.11}$$

Notice that for $\theta > \theta_{\text{limit}} \triangleq 1.55$, the QP becomes primal infeasible. We use gradient descent to minimize two scalar losses $\mathcal{L}_1(\theta) = x_1^\star(\theta)$ and $\mathcal{L}_2(\theta) = x_2^\star(\theta)$, starting from a predefined value θ_0. More precisely we have launched gradient descent for 40 steps with a learning rate 5×10^{-4} starting from $\theta_0 = 1.54$. Figure 6.4 illustrates the results by showing the iterates of gradient descent for minimizing $x_1^\star(\theta)$ (as well as the search direction—minus the ECJs). By doing so θ increases and eventually becomes larger than θ_{limit}.

Figure 6.5 reports a similar experiment for when minimizing $x_2^\star(\theta)$.

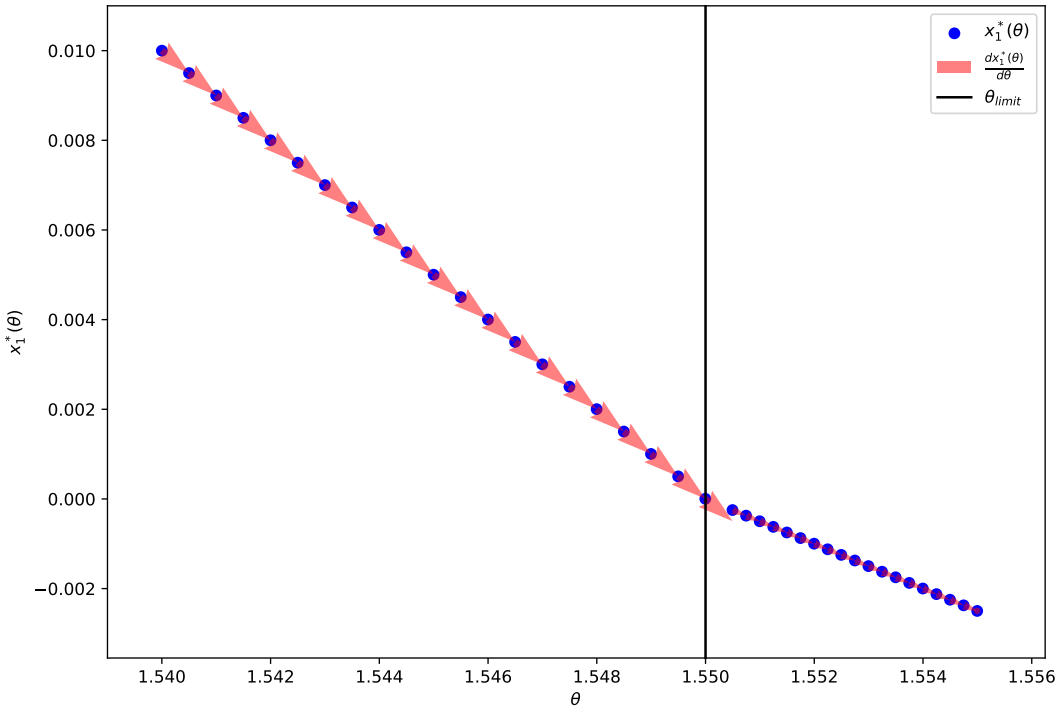

Figure 6.4: 40 steps of gradient descent for minimizing $x_1^\star(\theta)$ starting from $\theta_0 = 1.54$. When $\theta > 1.55$, (6.11) is differentiated though infeasible.

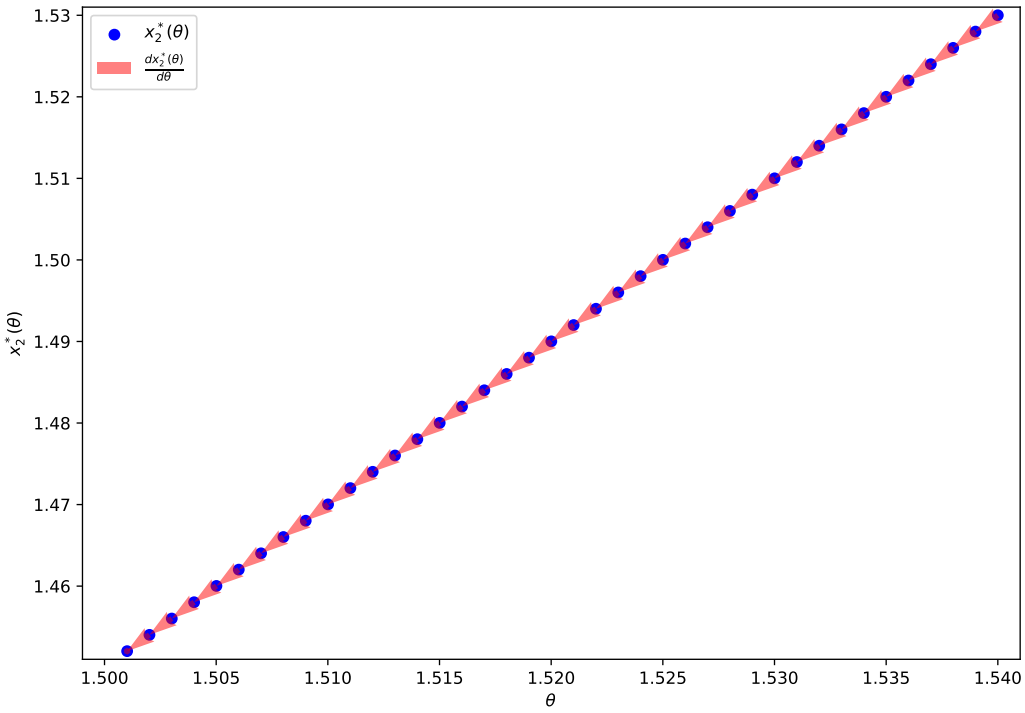

Figure 6.5: 40 steps of gradient descent for minimizing $x_2^\star(\theta)$ starting from $\theta_0 = 1.54$. The QPs remain feasible.

Strictly convex QP (parameterized objective)

Consider the following strictly convex QP parametrized by a scalar value θ

$$\begin{aligned} x^\star(\theta) = \underset{x_1,x_2\in\mathbb{R}^2}{\arg\min} \frac{1}{2} \left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \theta \\ -2 \end{bmatrix} \right\|_2^2 \\ \text{s.t.} \quad -300 \leq x_1 + x_2 \leq 400, \\ -200 \leq 2x_1 + x_2 \leq 500 \end{aligned} \tag{6.12}$$

We use gradient descent to minimize the loss $\mathcal{L}_1(\theta) = x_1^\star(\theta)$. More precisely, we run 40 iterations of gradient descent with learning rate 5×10^{-4} starting from $\theta_0 = 1.54$, as reported by Figure 6.6. As expected, we see that $x_1^\star(\theta) = -\theta$, hence increasing θ decreases $x_1^\star(\theta)$.

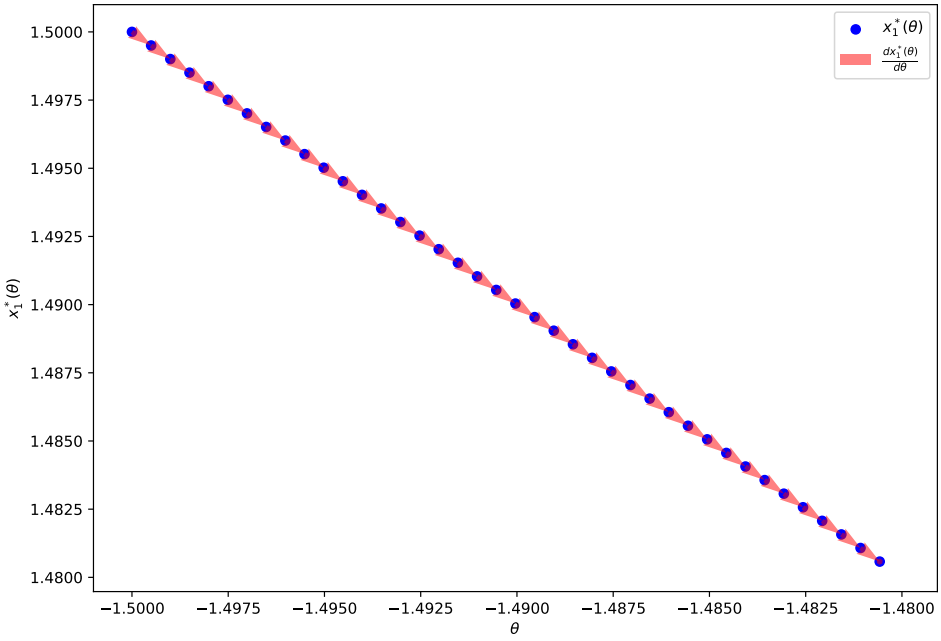

Figure 6.6: Gradient descent for minimizing $x_1^\star(\theta)$ (solution to (6.12)) starting from $\theta_0 = -1.5$.

Parameterized linear program

Consider the following LP parameterized by a scalar parameter $\theta > 0$

$$\begin{aligned} x^\star(\theta) \in \underset{x_1,x_2\in\mathbb{R}^2}{\arg\min}\, x_1 + x_2 \\ \text{s.t.} \quad \theta \leq x_1 + x_2, \\ 0 \leq x_1 \leq 1, \\ 0 \leq x_2 \leq 1. \end{aligned} \tag{6.13}$$

Note that this LP is always well-defined for any θ since the linear cost is orthogonal to the recession cone (which is empty), thereby satisfying the technical requirements from Assumption 3. We use gradient descent for minimizing two scalar losses $\mathcal{L}_1(\theta) = x_1^\star(\theta)$ and $\mathcal{L}_2(\theta) = x_2^\star(\theta)$.

Feasible case ($\theta \leq 2$). For any $\theta \in]0, 2]$, there are infinitely many solutions to (6.13) which are defined by the segment equation

$$\begin{aligned} x_1^\star + x_2^\star &= \theta, \\ 0 \leq x_1^\star &\leq 1, \\ 0 \leq x_2^\star &\leq 1. \end{aligned}$$

We can see in Figure 6.7 and Figure 6.8 that the forward pass chooses as solution $x_1^\star = x_2^\star = \frac{\theta}{2}$. Hence, only the constraint $\theta \leq x_1 + x_2$ is active. Following the formalism from Section 6.2.1 we have

$$C = \begin{bmatrix} -1 & -1 \\ 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}, \qquad u = \begin{bmatrix} -\theta \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

The ECJs of $\mathcal{L}_1$ and $\mathcal{L}_2$ w.r.t θ are the solutions to

$$\begin{bmatrix} (b_x^\star)_1 \\ (b_x^\star)_2 \\ b_z^\star \end{bmatrix} \in \underset{b_x, b_z}{\arg\min} \left\| \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & -1 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} (b_x)_1 \\ (b_x)_2 \\ b_z \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right\|_2^2,$$
$$\begin{bmatrix} (d_x^\star)_1 \\ (d_x^\star)_2 \\ d_z^\star \end{bmatrix} \in \underset{d_x, d_z}{\arg\min} \left\| \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & -1 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} (d_x)_1 \\ (d_x)_2 \\ d_z \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right\|_2^2.$$

As the corresponding linear systems involved within the ℓ_2 norm are infeasible, the least square estimates do not correspond to solutions to the linear system. That is, the corresponding least-square solutions are respectively the solutions of the following projected linear systems:

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} (b_x^\star)_1 \\ (b_x^\star)_2 \\ b_z^\star \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$
$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} (d_x^\star)_1 \\ (d_x^\star)_2 \\ d_z^\star \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

which leads to $\frac{\partial \mathcal{L}_1}{\partial \theta} = \frac{\partial \mathcal{L}_2}{\partial \theta} = b_z^\star = d_z^\star = \frac{1}{2}$. Figure 6.7 and Figure 6.8 show that those directions allow minimizing $\mathcal{L}_1$ and $\mathcal{L}_2$ through gradient descent.

Infeasible case ($\theta > 2$). For any $\theta > 2$, the LP is infeasible. The corresponding ECJs are the least-square solutions to

$$\underset{b_1, b_2, b_3, b_4}{\arg\min} \left\| \begin{bmatrix} 0 & C^\top & 0 & 0 \\ C & 0 & (I - \Pi_1) & 0 \\ 0 & -I & -\Pi_1 \Pi_2 & (1 - \Pi_2) C \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} + \begin{bmatrix} \frac{\delta \mathcal{L}_i}{\delta x} \\ 0 \\ 0 \end{bmatrix} \right\|_2^2 \text{ for } i \in \{1, 2\},$$

with $P_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$ and $P_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$. The linear system within the ℓ_2 norm is feasible and QPLayer outputs as ECJs $\frac{\partial \mathcal{L}_1}{\partial \theta} = \frac{\partial \mathcal{L}_2}{\partial \theta} = \frac{1}{3}$. Figure 6.9 and Figure 6.10 show that following such directions allows using gradient descent for minimizing $\mathcal{L}_1$ and $\mathcal{L}_2$.

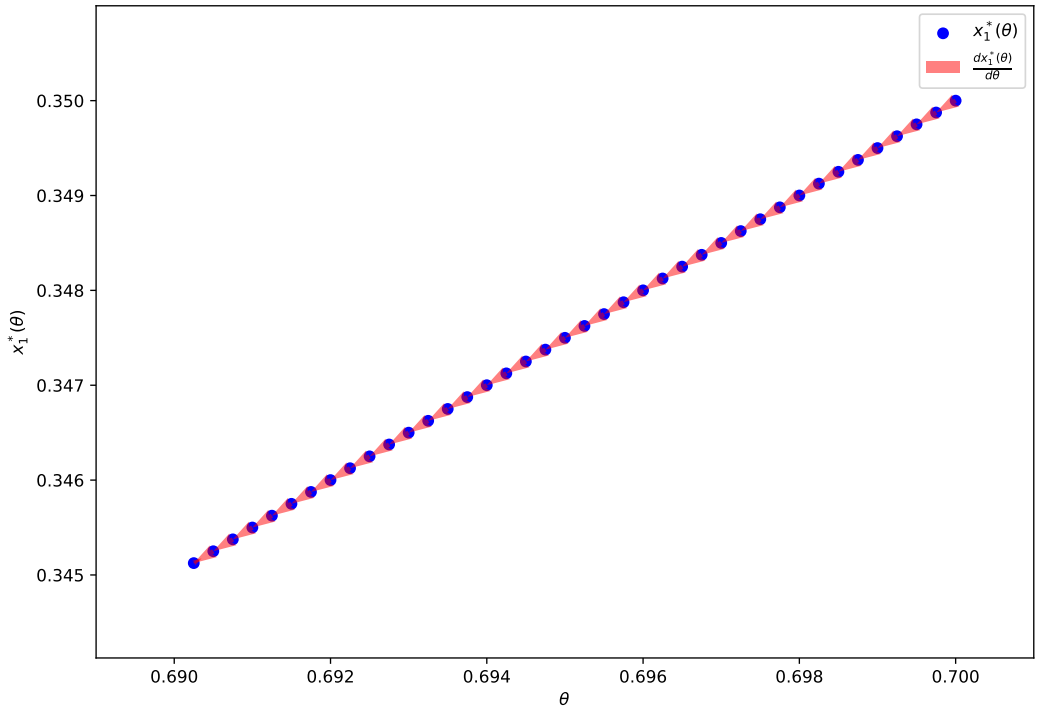

Figure 6.7: 40 iterations of gradient descent for minimizing $x_1^\star(\theta)$ for the problem (6.13).

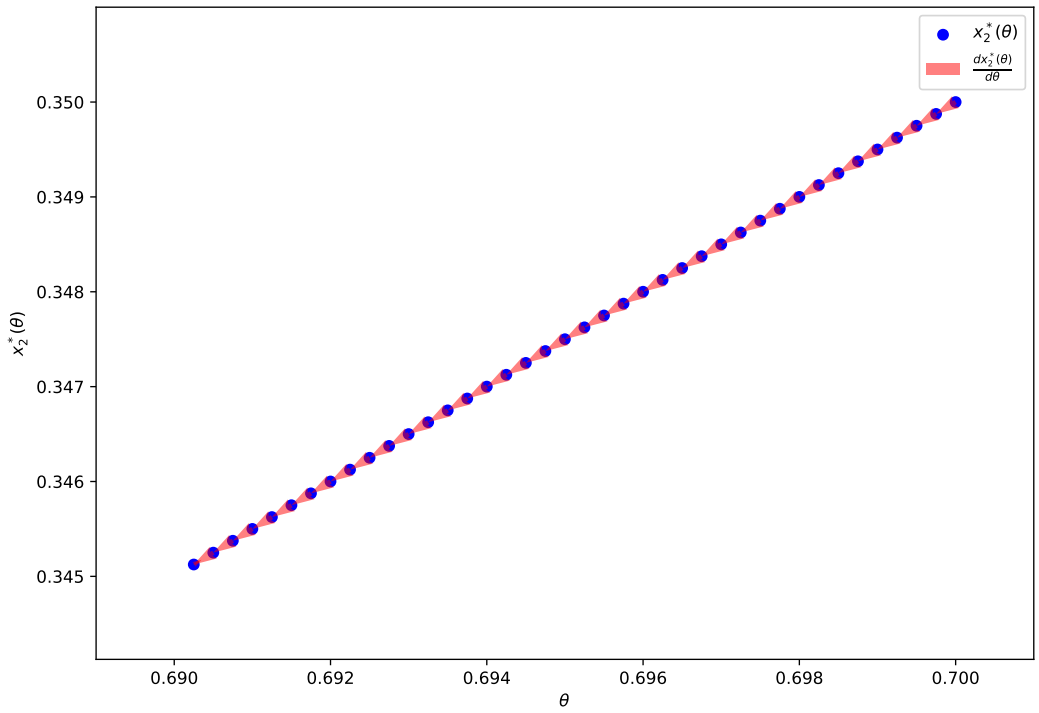

Figure 6.8: 40 iterations of gradient descent for minimizing $x_2^\star(\theta)$ for the problem (6.13).

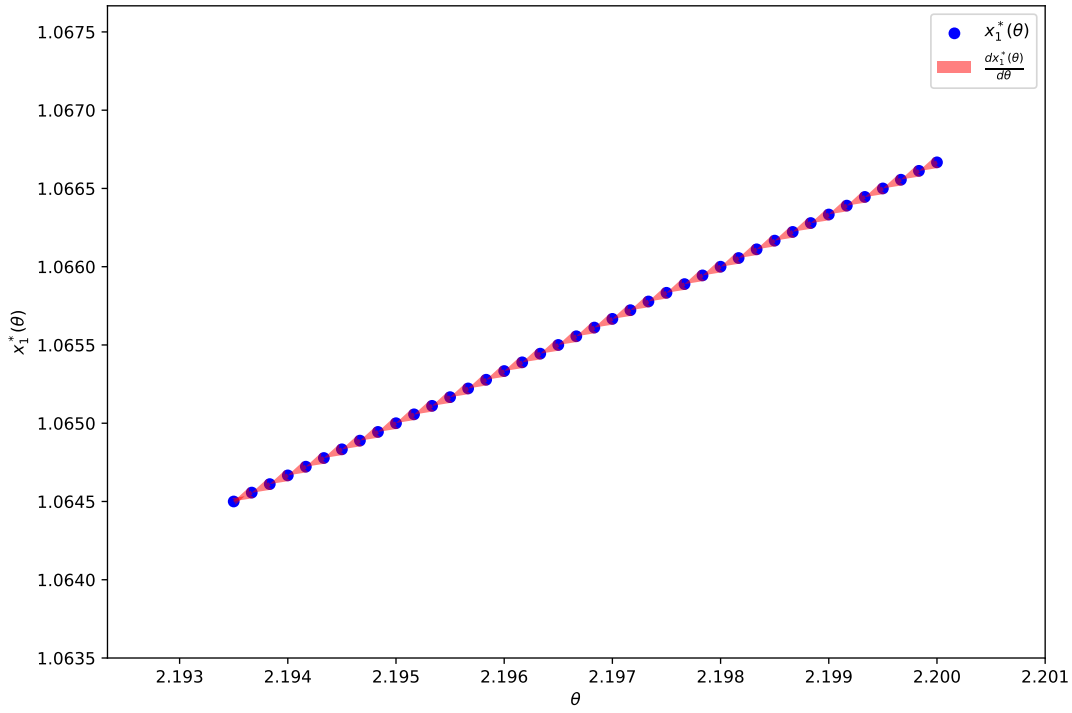

Figure 6.9: 40 steps of gradient descent applied to minimize $x_1^\star(\theta)$ starting from $\theta_0 = 2.2$, when considering the infeasible LP defined by (6.13).

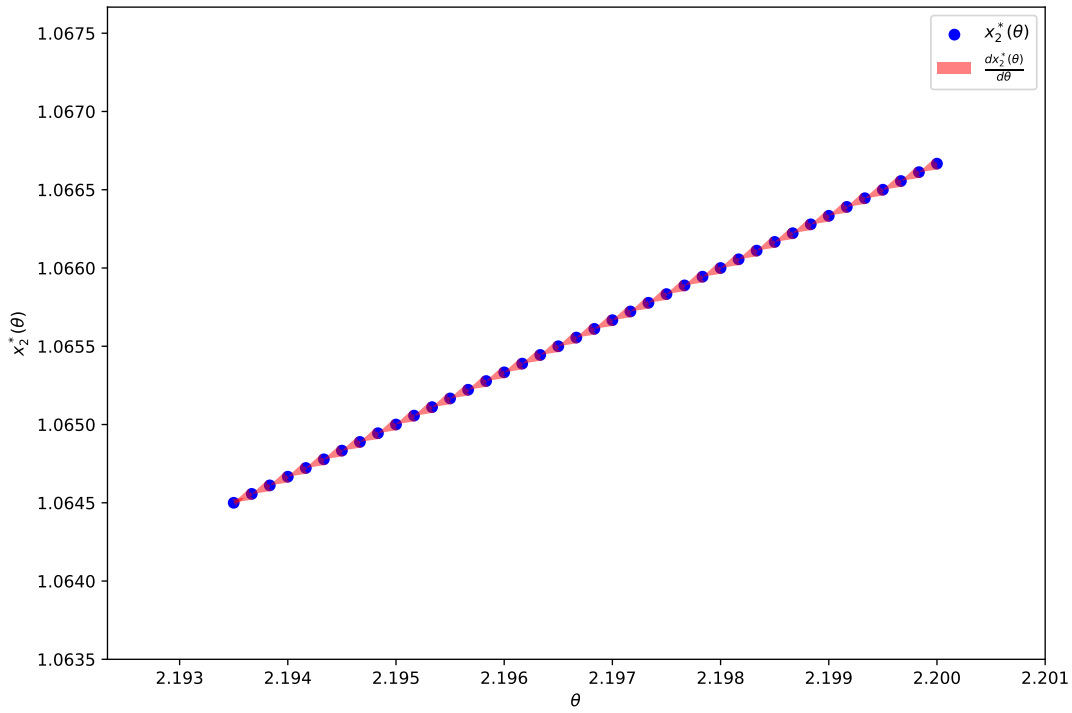

Figure 6.10: 40 steps of gradient descent applied to minimize $x_2^\star(\theta)$ starting from $\theta_0 = 2.2$, when considering the infeasible LP defined by (6.13).

6.3.2 Learning capabilities

Differentiable optimization for neural network layers has shown great representational power for learning problems that are fundamentally rooted in optimization. The Sudoku problem is one such problem, which can naturally be cast as a mixed integer linear program (MILP). For the Sudoku, OptNet recently showed better robustness and prediction accuracy results than traditional neural networks [Amos and Kolter, 2017, Section 4.4]. This section shows that QPLayer generalizes even better by exploiting the fact that it allows learning LPs (and not only QPs). Further, the ability of QPLayer to deal with possibly primal infeasible problems during the learning process appears to be key.

Learning linear programs

The Sudoku problem is detailed in [Amos and Kolter, 2017, Section 4.4]. We reproduce those experiments with OptNet and CvxpyLayer, while letting QPLayer learn linear programs (LPs) instead of strictly convex QPs. More precisely, OptNet and CvxpyLayer train models that are structurally feasible by learning an extra parameter z_0 (this layer is detailed in Figure 6.11). QPLayer enforces structural feasibility without considering the quadratic cost (i.e., it learns an LP).

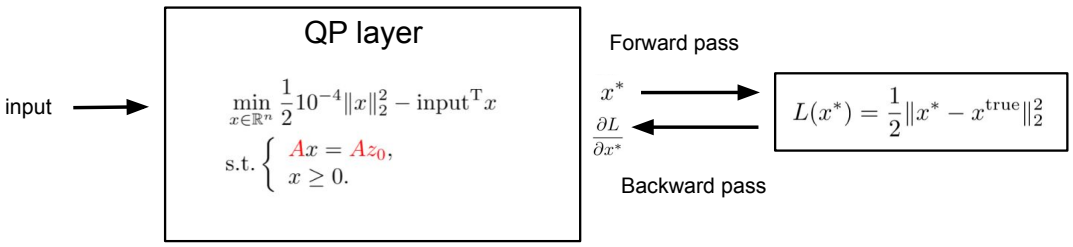

Figure 6.11: Strictly convex QP layer (as in OptNet [Amos and Kolter, 2017]). The constraint matrix and an extra variable z_0 are learned in order to be sure that the QP is always primal feasible (structural feasibility).

OptNet, CvxpyLayer, and QPLayer were trained using Adam with a batch of size 150 and a learning rate of 0.05 to minimize an MSE loss on the dataset created by [Amos and Kolter, 2017]. The dataset contains 9000 training puzzles and 1000 puzzles for testing. First, Figure 6.12 shows that QPLayer minimizes the training and test loss without ending up over-fitting to the training data, contrary to OptNet and CvxpyLayer which appears to saturate. Second, Figure 6.13 shows that QPLayer achieves significantly more accurate and robust training and test error predictions than OptNet and CvxpyLayer.

Handling primal infeasibility

As outlined in Section 6.3.2, forcing primal feasibility while learning is a common algorithmic strategy. For the Sudoku problems, those techniques enforcing primal feasibility typically involve neglecting a linear equality constraint $Ax = 1$ (we learn A) which corresponds to Sudoku rules. As shown in Figure 6.16, this means that the learning procedures do not respect the Sudoku rule constraint (see green and orange dashed lines—labeled "QPLayer; $Ax = 1$ violation" and "OptNet; $Ax = 1$ violation"). In the end, neglecting this constraint ultimately leads to learning a constraint matrix that is inconsistent with the Sudoku rules. Relaxing the primal feasibility imposed by differentiation procedures of previous solvers thereby appears to be key.

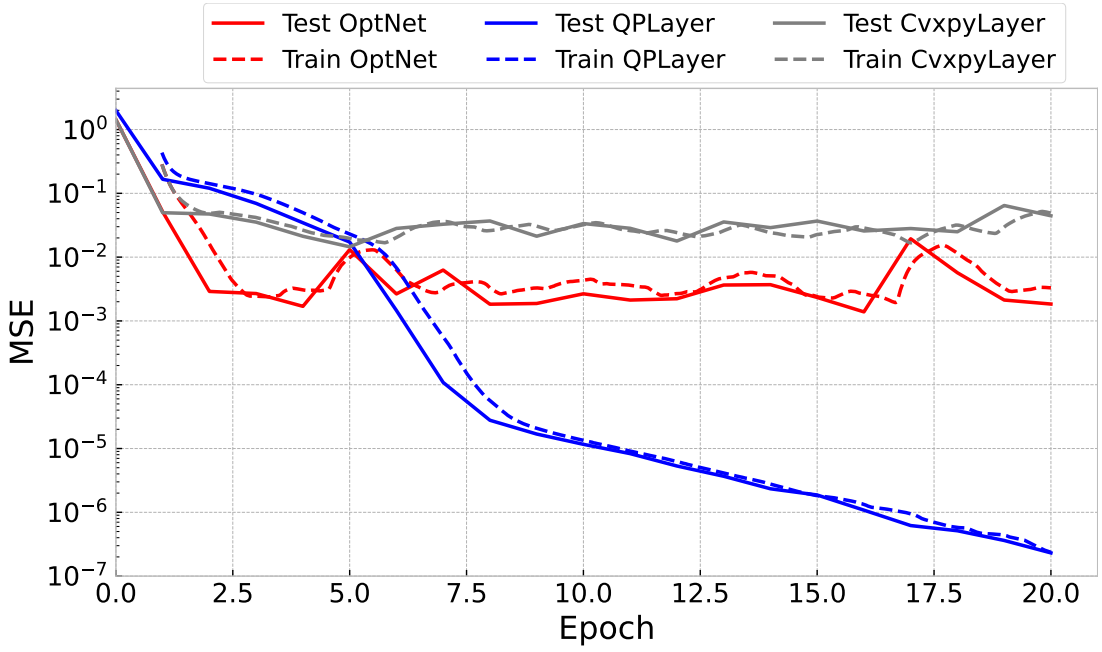

Figure 6.12: Test and training MSE losses of QPLayer, CvxpyLayer *and* OptNet *layers.*

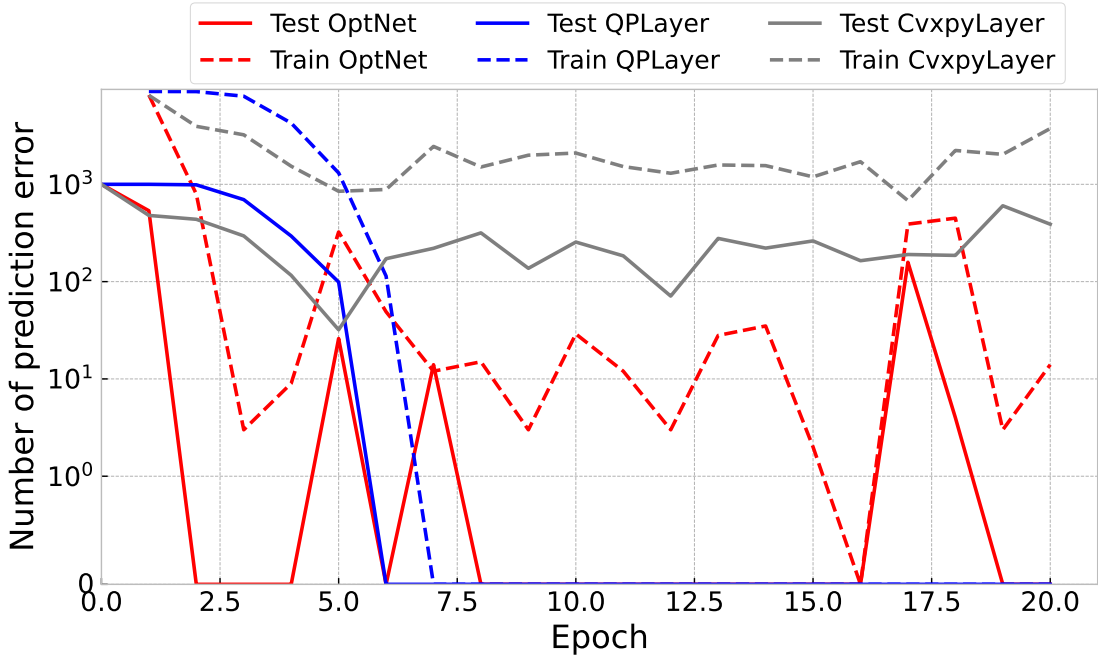

Figure 6.13: Test and training prediction errors of QPLayer, CvxpyLayer, *and* OptNet *over 1000 and 9000 puzzles.*QPLayer *can learn LPs (which appear more appropriate), whereas* OptNet *is restricted to strictly convex QPs.*

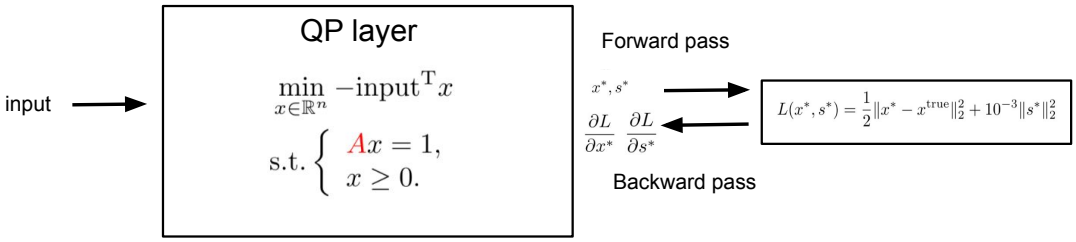

Figure 6.14: An LP layer (as allowed by QPLayer) allows for more flexibility in the problem to be learned (only the constraint matrix A is learned). The optimal shift s^* is a new output variable minimized in the loss, in order to learn at the end a feasible LP layer.

By incorporating a potential optimal shift $s^\star$ in its formulation (as exposed in Section 6.2.1), QPLayer allows dealing with infeasible problems. Numerical performances are reported in Figure 6.15 and Figure 6.16. It is apparent that the dark green curve labeled "QPLayer-learn A; $Ax = 1$ violation" converges after the end of the first epoch towards a model satisfying Sudoku rules. It also converges slightly faster towards a regime without any prediction errors. The steeper slope observed in the graph suggests that it might be worthwhile to train a layer that more accurately adheres to the Sudoku rules, as this could potentially lead to faster puzzle-solving and more interpretable outcomes. For comparison, the gray curves correspond to results obtained by training a strictly convex QP layer with relaxed primal feasibility constraint (labeled "OptNet-learn A; $Ax = 1$ violation) using OptNet. As expected, it fails to satisfy the primal feasibility constraint (i.e., the black dashed curve value is around 10) and displays a worse prediction error.

6.3.3 Timing benchmarks on standard learning models

Benchmark setup. In the first set of experiments (see Section 6.3.3), QPLayer is compared to OptNet, CvxpyLayer, JaxOpt, and Alt-Diff. For all the other experiments, QPLayer is benchmarked only against approaches available within the PyTorch framework (i.e., OptNet and CvxpyLayer[2]). The experiments were conducted using all threads of an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz. The benchmark API will be released upon acceptance of this work. It was inspired by `https://github.com/locuslab/optnet` and has been extended to cover all our experiments. It should be noted that all the benchmarks were obtained on CPU whereas we plan to release a GPU extension of our techniques in future work.

In this section, we report our numerical results and compare them against state-of-the-art frameworks on a set of standard experiments. We outline detailed timings for differentiating solutions on a set of different QPs. First, Table 6.1 shows the results for a few randomly generated QPs. Second, Table 6.2 reports the average time spent in the differentiation procedure on four different learning tasks.

Random QPs

In this first set of experiments, we generated random QPs with 100 variables, 50 equality, and 50 inequality constraints. We solve and backpropagate through all those QPs for different forward pass accuracies. We then average the results over 5 trails and report the timings in the spirit of [Sun et al., 2022].

For a batch size of 100, Table 6.1 shows that QPLayer is almost 4 times faster than OptNet (the second fastest approach). We can see similar performance for a batch size of 1 (see Table 6.3

[2]Alt-Diff exhibited too slow performances for a fair and reasonable comparison. Note that [Sun et al., 2022] have not yet proposed an Alt-Diff layer deriving all QP Jacobians. Therefore, we have included in our benchmark an open-source implementation of Alt-Diff based on their work.

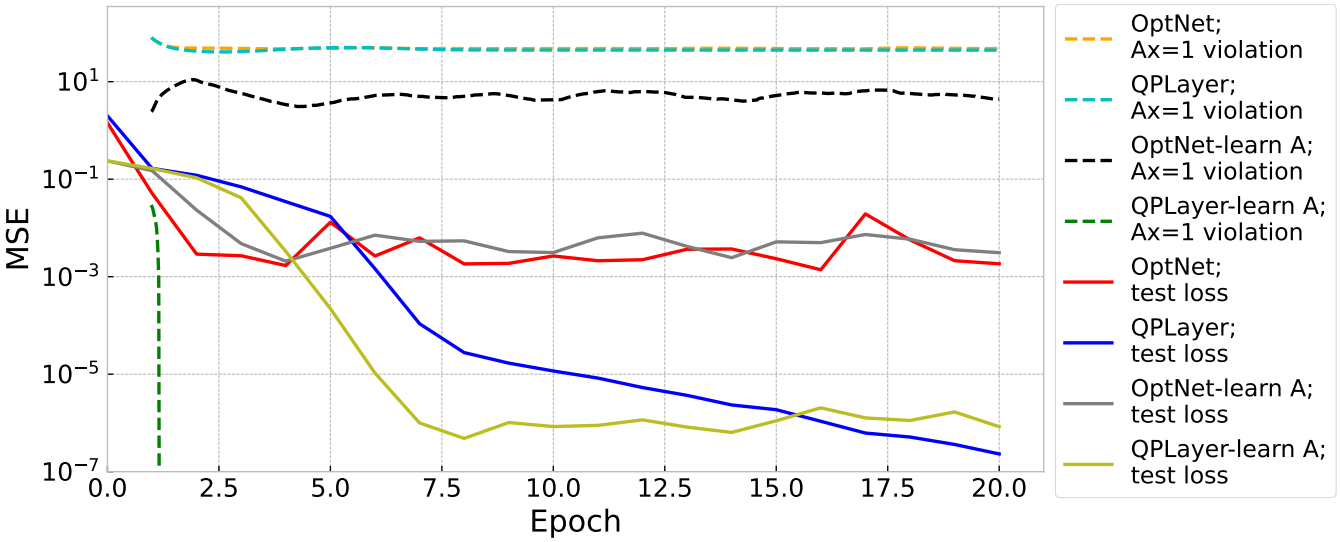

Figure 6.15: Test MSE loss of QPLayer, OptNet, QPLayer-learn A, and OptNet-learn A specialized for learning A. It includes Sudoku $Ax = 1$ violation.

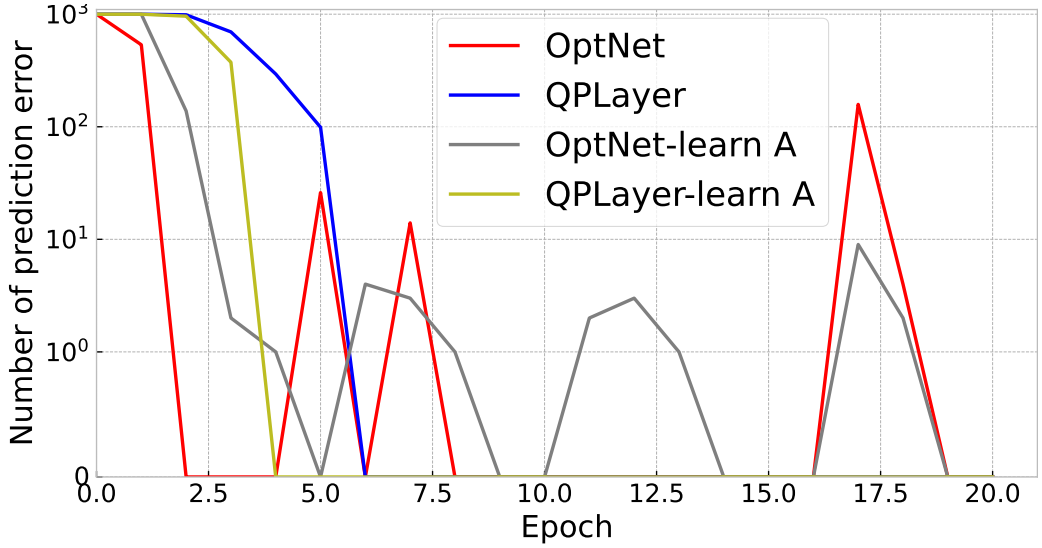

Figure 6.16: Test prediction errors over 1000 puzzles of OptNet, QPLayer, QPLayer-learn A and OptNet-learn A specialized for learning A. Contrary to OptNet, QPLayer can be specialized to learn models satisfying specific linear constraints.

in the appendix). We observe that the speed gain is mostly due to the forward pass speed-up, enabled by the use of ProxQP and thread parallelization. It is also confirmed by Table 6.4, which reproduces in the appendix the serial forward timing benchmark proposed in [Amos and Kolter, 2017, Section 4.1]. It exhibits from 5 to 9 times faster computation times.

Learning tasks

For this second set of experiments, we report the numerical results obtained on 4 traditional learning tasks (namely MNIST classification, signal denoising, Sudoku solving and cart-pole experiment). For all experiments, we report the average (over all epochs) time spent in the forward and backward passes. Table 6.2 reports that QPLayer is 3 to 10 times faster than the second fastest approach (i.e., 3 times faster on the classification task, 4 times faster on the Sudoku, about 10 times faster for the denoising and 7 times faster for the cart-pole experiments). In all cases, the test loss incurred using QPLayer is either similar (for the denoising and cart-pole tasks) or far better than its competitor layers (about 2 times smaller and 3 orders of magnitudes better for the classification and Sudoku experiments).

More precisely, the first three experiments reproduce the ones originally described in [Amos and Kolter, 2017, Sections 4.2 to 4.4]. A complete description of the cart-pole swing-up task is detailed in Section 6.4.5. These tasks involve learning convex feasible QPs using Adam optimizer [Kingma and Ba, 2014]. The first three experiments are run with the default batch sizes and the number of epochs fixed by the original authors (i.e., batch size equals 64 for classification, 150 for denoising and Sudoku tasks; 30 epochs for classification, and 20 for denoising and Sudoku tasks). We run the cart-pole example with batch size 1 for 800 epochs. For the first three experiments, we use the same QP layer models as [Amos and Kolter, 2017], except that we have changed the backends for executing the forward and backward passes (using either Qpth, QPLayer[3] or CvxpyLayer). Finally, we have used the following learning rates for running the benchmarks: 10^{-3} for classification, 10^{-5} for denoising, 5×10^{-2} for Sudoku, and 10^{-1} for cart-pole tasks.

[3]Two differences should be noted: QPLayer learns an LP for the Sudoku experiment. We have not imposed zero Hessian for the CvxpyLayer even if it could learn it, as it did display worse results. Furthermore, for the denoising experiment, QPLayer learns the lower and upper bounds simultaneously.

Forward tolerance ϵ	10^{-1}	10^{-2}	10^{-3}
Forward (ms)	**72.43** ($\pm$7.28)	**71.89** ($\pm$3.0)	**72.12** ($\pm$4.21)
Backward (ms)	18.14 ($\pm$1.01)	18.03 ($\pm$0.17)	18.12 ($\pm$0.37)
QPLayer total (ms)	**90.57**	**89.92**	**90.24**
Forward (ms)	340.62 ($\pm$7.06)	348.68 ($\pm$3.51)	349.44 ($\pm$2.44)
Backward (ms)	**6.39** ($\pm$0.16)	**6.61** ($\pm$0.41)	**6.66** ($\pm$0.38)
OptNet total (ms)	347.01	355.29	356.10
Forward (ms)	123.35 ($\pm$13.70)	196.75 ($\pm$38.13)	281.54 ($\pm$64.18)
Backward (ms)	546.91 ($\pm$55.58)	622.45 ($\pm$66.63)	723.34 ($\pm$66.26)
JaxOpt total (ms)	670.16	819.20	1004.88
Forward (ms)	1.16($\pm$0.038)$\times 10^3$	1.19 ($\pm$0.015)$\times 10^3$	1.24 ($\pm$0.017)$\times 10^3$
Backward (ms)	187.52($\pm$8.59)	187.74 ($\pm$11.54)	197.18 ($\pm$7.99)
CvxpyLayer total (ms)	1.35$\times 10^3$	1.38$\times 10^3$	1.43$\times 10^3$
Forward (ms)	Time limit	Time limit	Time limit
Backward (ms)	Time limit	Time limit	Time limit
Alt-Diff total (ms)	Time limit	Time limit	Time limit

Table 6.1: Timings for deriving all Jacobians of random QPs with different forward pass accuracies and batch size 100.

Learning Tasks	QPLayer	OptNet	CvxpyLayer
cart-pole			
Forward (ms)	**55.20** ($\pm$6.93)	615.84 ($\pm$16.15)	629.70 ($\pm$30.42)
Backward (ms)	**39.27** ($\pm$2.17)	61.26 ($\pm$2.84)	101.88 ($\pm$1.11)
Final test loss	**0.02556**	0.02604	0.02566
Sudoku			
Forward (ms)	**87.39** ($\pm$16.69)	454.48 ($\pm$22.22)	772.77 ($\pm$33.89)
Backward (ms)	20.80 ($\pm$1.70)	**8.07** ($\pm$0.42)	192.99 ($\pm$9.81)
Final test loss	**2.31**$\times 10^{-7}$	0.0017	0.389
denoising			
Forward (ms)	**247.89** ($\pm$17.03)	2827.48 ($\pm$618.78)	Error
Backward (ms)	52.99 ($\pm$2.75)	**40.57** ($\pm$2.98)	Error
Final test loss	**3281.84**	3529.2029	Error
classification			
Forward (ms)	**26.77**($\pm$3.63)	102.62 ($\pm$18.23)	Error
Backward (ms)	**11.12** ($\pm$3.01)	16.58 ($\pm$12.05)	Error
Final test loss	**0.1363**	0.3264	Error

Table 6.2: Timings and final loss of 4 learning tasks. CvxpyLayer errors arise due to failures in filling the disciplined parametrized programming (DPP) form of the quadratic cost.

6.3.4 Training with large learning rates

In this section, we assess the numerical robustness of QPLayer on traditional learning tasks by demonstrating that it can be trained with larger learning rates than other approaches, potentially resulting in improved basins of attraction.

More precisely, we ran the MNIST classification and denoising tasks described in Section 6.3.3 with SGD and larger learning rates and reported the corresponding results. We measured the

Forward tolerance ϵ	10^{-1}	10^{-2}	10^{-3}
Forward (ms)	**1.30** ($\pm$0.17)	**1.42** ($\pm$0.19)	**1.55** ($\pm$0.20)
Backward (ms)	0.77 ($\pm$0.01)	0.70 ($\pm$0.02)	0.71 ($\pm$0.02)
QPLayer total (ms)	**2.07**	**2.12**	**2.26**
Forward (ms)	6.93 ($\pm$0.82)	6.85 ($\pm$0.05)	7.49 ($\pm$0.04)
Backward (ms)	**0.75** ($\pm$12)	**0.70** ($\pm$0.01)	**0.70** ($\pm$0.01)
OptNet total (ms)	7.68	7.55	8.19
Forward (ms)	7.99 ($\pm$0.93)	12.82 ($\pm$2.67)	18.82 ($\pm$4.31)
Backward (ms)	24.71 ($\pm$2.53)	30.47 ($\pm$2.71)	36.43 ($\pm$3.93)
JaxOpt total (ms)	32.70	43.29	55.25
Forward (ms)	411.20 ($\pm$3.90)	415.94 ($\pm$2.63)	422.28 ($\pm$2.65)
Backward (ms)	6.13 ($\pm$0.02)	6.34 ($\pm$0.33)	6.26 ($\pm$0.06)
CvxpyLayer total (ms)	417.33	422.24	428.54
Forward (ms)	6.00 ($\pm$0.85)$\times 10^3$	38.66 ($\pm$7.11)$\times 10^3$	114.20 ($\pm$31.48)$\times 10^3$
Backward (ms)	0.99 ($\pm$0.21)	0.98 ($\pm$0.13)	1.05 ($\pm$0.14)
Alt-Diff total (ms)	6.00$\times 10^3$	36.66$\times 10^3$	114.20$\times 10^3$

Table 6.3: Averaged timings for the forward and backward passes when computing all Jacobians of randomly generated feasible QPs (with 100 primal variables, 50 equality constraints and 50 inequality constraints) considering different forward pass accuracies. The batch size is 1. QPLayer has the best total timings for all accuracies.

QP and Batch sizes	QPLayer (ms)	OptNet (ms)	CvxpyLayer (ms)
Batch = 1, $n = 100$, $n_e = 0$, $n_i = 50$	**0.88** ($\pm$0.06)	8.01 ($\pm$0.89)	334.06 ($\pm$7.83)
Batch = 1, $n = 100$, $n_e = 50$, $n_i = 50$	**1.09** ($\pm$0.10)	8.35 ($\pm$0.67)	406.18 ($\pm$1.44)
Batch = 1, $n = 100$, $n_e = 0$, $n_i = 100$	**1.22** ($\pm$0.08)	7.62 ($\pm$0.32)	422.59 ($\pm$4.34)
Batch = 1, $n = 100$, $n_e = 50$, $n_i = 100$	**1.75** ($\pm$0.17)	13.95 ($\pm$2.03)	522.80 ($\pm$3.96)
Batch = 1, $n = 100$, $n_e = 100$, $n_i = 50$	**1.26** ($\pm$0.14)	16.77 ($\pm$3.41)	521.18 ($\pm$16.15)
Batch = 1, $n = 100$, $n_e = 100$, $n_i = 100$	**1.45** ($\pm$0.28)	18.33 ($\pm$5.35)	621.25 ($\pm$9.67)
Batch = 64, $n = 100$, $n_e = 0$, $n_i = 50$	**30.16** ($\pm$5.51)	173.95 ($\pm$12.82)	771.49 ($\pm$59.74)
Batch = 64, $n = 100$, $n_e = 50$, $n_i = 50$	**48.60** ($\pm$9.19)	183.84 ($\pm$9.99)	818.75 ($\pm$5.42)
Batch = 64, $n = 100$, $n_e = 0$, $n_i = 100$	**44.68** ($\pm$5.95)	176.88 ($\pm$2.30)	820.85 ($\pm$12.38)
Batch = 64, $n = 100$, $n_e = 50$, $n_i = 100$	**55.77** ($\pm$6.38)	243.15 ($\pm$27.86)	1126.99 ($\pm$34.33)
Batch = 64, $n = 100$, $n_e = 100$, $n_i = 50$	**51.66** ($\pm$8.51)	231.11 ($\pm$13.75)	1177.89 ($\pm$132.99)
Batch = 64, $n = 100$, $n_e = 100$, $n_i = 100$	**62.52** ($\pm$7.84)	276.18 ($\pm$31.29)	1314.47 ($\pm$69.60)
Batch = 128, $n = 100$, $n_e = 0$, $n_i = 50$	**62.94** ($\pm$9.52)	620.91 ($\pm$3.52)	1202.61 ($\pm$88.32)
Batch = 128, $n = 100$, $n_e = 50$, $n_i = 50$	**78.01** ($\pm$4.06)	667.55 ($\pm$5.86)	1295.50 ($\pm$24.63)
Batch = 128, $n = 100$, $n_e = 0$, $n_i = 100$	**89.48** ($\pm$7.09)	653.32 ($\pm$5.54)	1267.83 ($\pm$30.85)
Batch = 128, $n = 100$, $n_e = 50$, $n_i = 100$	**111.01** ($\pm$8.21)	774.08 ($\pm$20.90)	1739.41 ($\pm$70.53)
Batch = 128, $n = 100$, $n_e = 100$, $n_i = 50$	**96.23** ($\pm$9.47)	811.44 ($\pm$25.29)	1896.52 ($\pm$284.89)
Batch = 128, $n = 100$, $n_e = 100$, $n_i = 100$	**119.13** ($\pm$10.17)	934.26 ($\pm$87.79)	2059.44 ($\pm$147.77)

Table 6.4: Timing benchmarks of different backends used for solving a forward pass for different batch sizes. n_e stands for the number of equality constraints.

final test loss and error reached after 30 epochs and the standard deviation over the last 10 epochs of the test loss and test error. The results are averaged over 10 seeds and the experiment is performed for different learning rates.

As described in Section 6.3.3, for those tasks, the QP layers need to learn all the model parameters (i.e., H, g, C, and u). We observe that it generates potentially very ill-conditioned problems when the forward or the backward passes are not solved accurately enough. This phenomenon appears to be amplified with larger learning rates. In those situations, it appears that robust solution methods (e.g., allowing for temporary infeasible, or ill-conditioned problems) are critical.

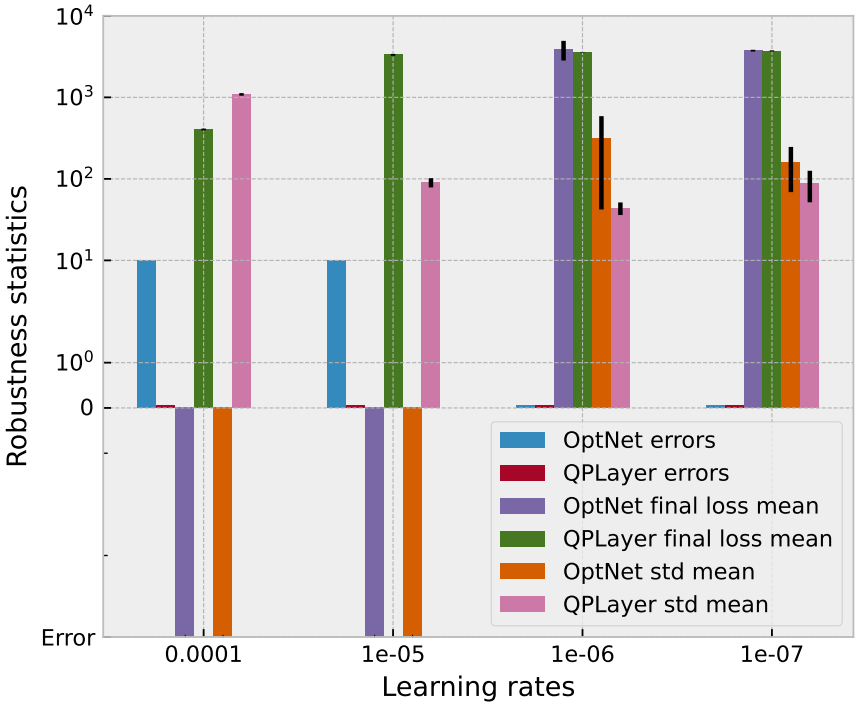

Figure 6.17: Robustness statistics for denoising task:number of errors (i.e., NaNs), averaged final loss reached after 30 epochs (with 95% confidence intervals), and averaged standard deviations over the last 10 epochs (with 95% confidence intervals). Results are averaged over 10 seeds. CvxpyLayer fails to be trained for all these tasks.

Figure 6.17 and Figure 6.18 show that for too high learning rates (i.e., 10^{-4} or 10^{-5} for denoising task and 10^{-2} for the classification task) the OptNet layer generate errors, whereas it is never the case for QPLayer. Furthermore, for low learning rate levels (i.e., 10^{-6} or 10^{-7} for the denoising task and 10^{-3} and 10^{-4} for the classification task), the final loss reached is similar but with a less important noise amplitude level when using QPLayer (Figure 6.19 provides robustness statistics of the classification task using the prediction error rate of the two layers). Finally, QPLayer is capable of being trained with a larger learning rate (i.e., 10^{-4} for the denoising task and 10^{-2} for the classification task). Also, note that CvxpyLayer fails to be run in all these robustness experiments because the Hessian part of the quadratic model fails to fit the required DPP form [Amos et al., 2018, Section 4.1].

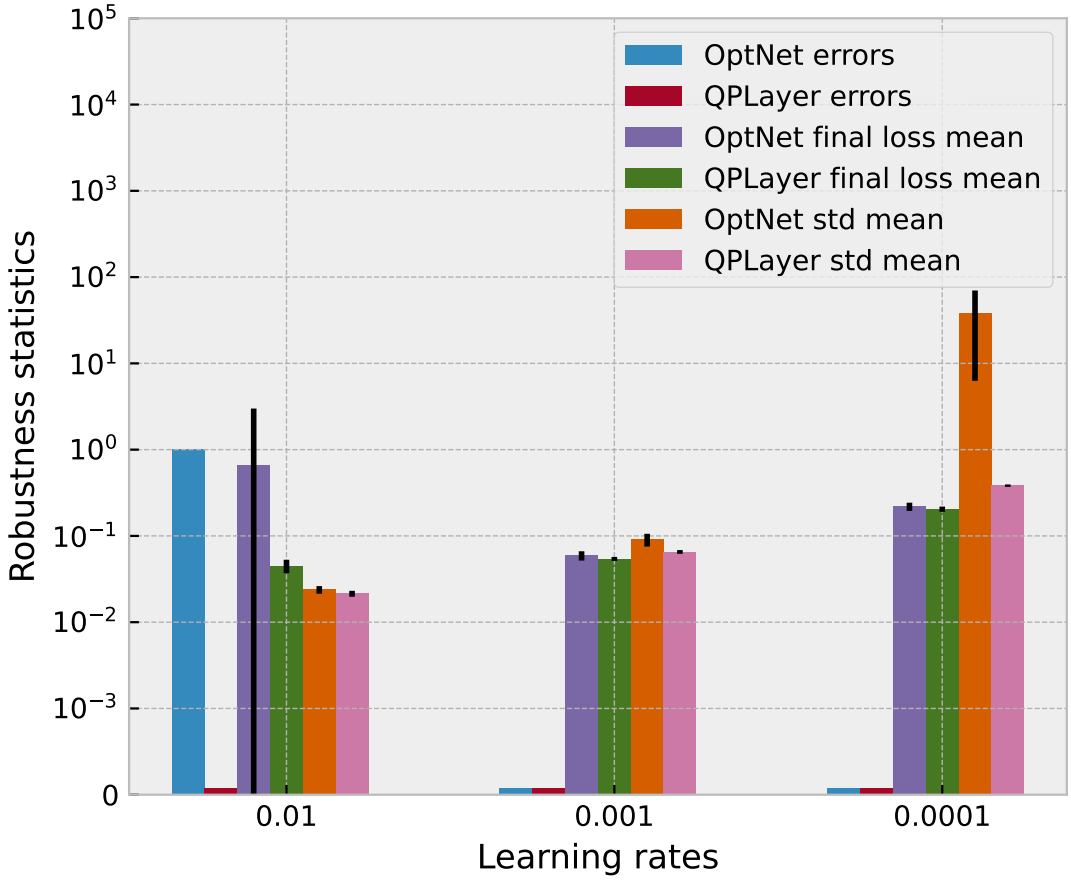

Figure 6.18: Robustness statistics for MNIST classification task:number of errors (i.e., NaNs), averaged final loss reached after 30 epochs (with 95% confidence intervals), and averaged standard deviations over the last 10 epochs (with 95% confidence intervals). Results are averaged over 10 seeds. CvxpyLayer fails to be trained for all these tasks.

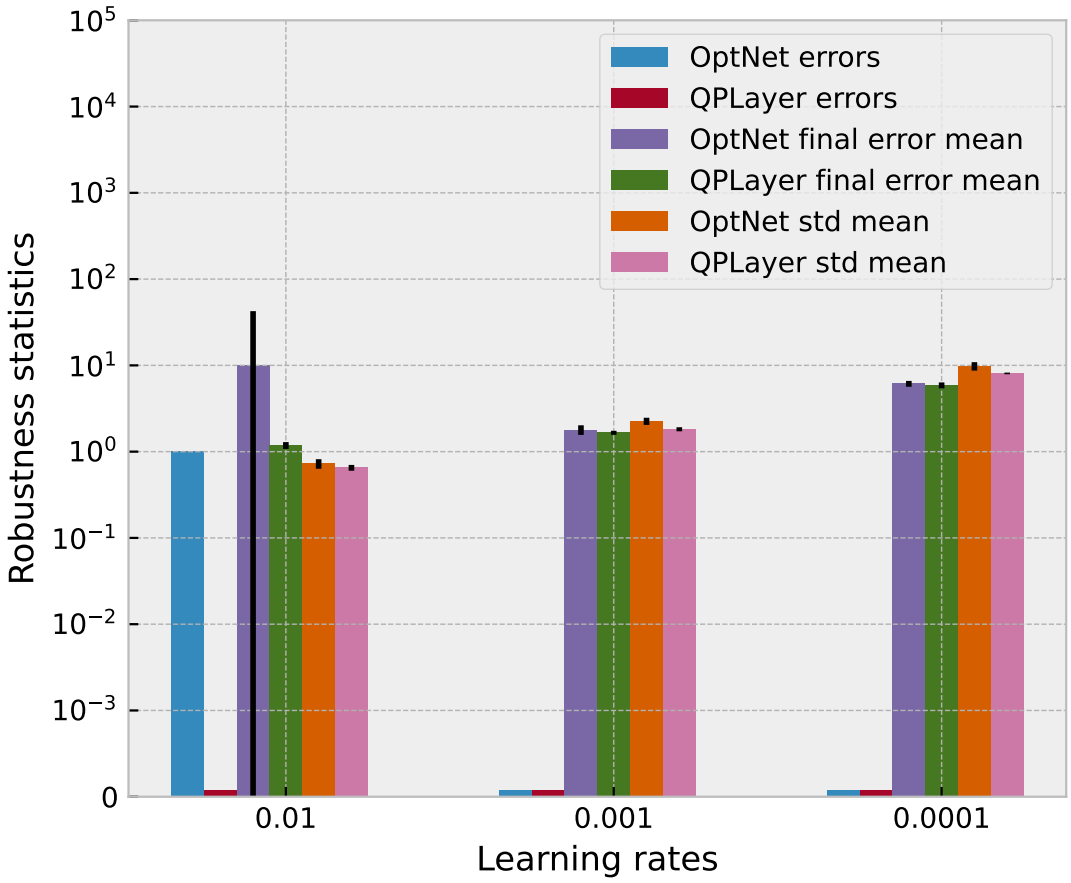

Figure 6.19: Robustness statistics for the MNIST classification: number of errors (i.e., NaNs), averaged final prediction error reached after 30 epochs (with 95% confidence intervals), and averaged standard deviations over the last 10 epochs (with 95% confidence intervals). Results are averaged over 10 seeds.

Remark 5 (Numerical differences with OptNet)**.** *Our approach offers a few numerical advantages compared to [Amos and Kolter, 2017]. In particular, a numerical matrix factorization is at the center of most popular techniques for differentiating through QPs. This factorization procedure represents one of the main bottlenecks in the computational costs. In our approach, we need to factorize smaller and better-conditioned symmetric matrices.*

The formulation exploited by OptNet consists of a larger linear system to compute its Jacobians. More precisely, they factorize a matrix of the form:

$$K = \begin{bmatrix} H & 0 & C^\top \\ 0 & D(z^\star) & D(t^\star) \\ C & I & 0 \end{bmatrix},$$

where $t^\star = Cx^\star - u$ and $D(z^\star)$ corresponds to a diagonal matrix whose diagonal entries correspond to $z^\star$. For obtaining a symmetrized version that can be factorized with efficient methods, the second row block is scaled by $D(1/t^\star)$ [Amos and Kolter, 2017, Section 3.1]. Yet, symmetrization comes at the price of being more sensitive to the localization of the solution w.r.t the constraints. Indeed, if $x^\star$ lies on the boundary, i.e., $C_I x^\star - u_I = 0$ for some component index I, the conditioning of the matrix is degraded, as OptNet needs to divide by zeros (or small clamped numbers in practice).

On our side, the formulation for feasible QPs relies on a smaller matrix, which is symmetric and better-conditioned (it does not require scaling rows by values that are potentially zeros, see (6.9)).

6.4 Proofs

Section	Content
Section 6.4.1	Lemma 5: solutions to QP-H.
ECJs and automatic differentiation.	Lemma 6: path differentiability of G (Section 6.4.2).
	Forward AD (general case, Section 6.4.3).
	Backward AD (general case, Section 6.4.3).
	Forward AD (feasible QPs, Lemma 6.4.3).
	Backward AD (structurally feasible QPs, Equation 6.4.3).
	Lemma 7: when do ECJs reduce to Jacobians? (Section 6.4.4).
Experimental setups.	Description of the cart-pole problem (Section 6.4.5).

Table 6.5: Organization of the appendix. QP stands for "quadratic programming", AD stands for "automatic differentiation", and (E)CJ stands for (extended) conservative Jacobian.

6.4.1 Proof of Lemma 5

To prove Lemma 5, we first show that solutions to (QP-H) are zeros of the map $\mathcal{G}$:

$$\mathcal{G}(x, z;\ \theta) \overset{\text{def}}{=} \begin{bmatrix} H(\theta)x + g(\theta) + C(\theta)^\top z \\ [[C(\theta)x - u(\theta)]_- + z]_+ - z \\ C(\theta)^\top [C(\theta)x - u(\theta)]_+ \end{bmatrix}. \tag{6.14}$$

Then, a suitable change of variable shows that finding a zero of $\mathcal{G}$ is equivalent to finding a zero of map G.

Lemma 5. *Let $H(\theta) \in \mathcal{S}_+^n(\mathbb{R})$, $g(\theta) \in \mathbb{R}^n$, $C(\theta) \in \mathbb{R}^{n_i \times n}$ and $u(\theta) \in \mathbb{R}^{n_i}$ be satisfying Assumption 3. It holds that $(x^\star, z^\star, s^\star)$ solves QP-H(θ) iff there exists $t^\star \in \mathbb{R}^{n_i}$ s.t. $G(x^\star, z^\star, t^\star;\ \theta) = 0$ and $s^\star = [t^\star]_+$.*

Proof. We first show that $(x^\star, z^\star, [Cx^\star - u]_+)$ solves (QP-H) if and only if $\mathcal{G}(x^\star, z^\star;\ \theta) = 0$.

The optimal shift $s^\star$ (that corresponds to the closest feasible QP) is equal to $[Cx^\star - u]_+$ and is characterized by the ℓ_2 optimality condition [Chiche and Gilbert, 2016, Lemma 2.13]:

$$C^\top [Cx^\star - u]_+ = 0.$$

Furthermore, for a feasible problem, the KKT conditions using a nonlinear complementarity formulation [Sun and Qi, 1999] reads [De Marchi, 2022, Section 2.1]:

$$\begin{aligned} &Hx^\star + g + C^\top z^\star = 0, \\ &[Cx^\star - u + z^\star]_+ - z^\star = 0. \end{aligned} \tag{6.15}$$

To show equivalence, it is thereby sufficient to show that the second line of (6.15) corresponds to:

$$[[Cx^\star - u]_- + z^\star]_+ - z^\star = 0. \tag{6.16}$$

This equivalence is straightforward when (QP(θ)) is feasible, it therefore follows that we only need to handle the infeasible case. When (QP(θ)) is primal infeasible, then $t^\star = Cx^\star - u$ has a set of components $I \sqsubset [1, n_i]$ strictly positive, hence $s^\star_I = [t^\star_I]_+ = t^\star_I > 0$. For these components, a solution $x^\star$ of the closest feasible QP lies on the border $C_I x^\star = u_I + t^\star_I$. The complementarity condition [De Marchi, 2022, Section 2.1] for these components reads:

$$[\underbrace{Cx^\star - u_I - t^\star_I}_{=0} + z^\star_I]_+ - z^\star_I = 0,$$

and we thus have $[z^\star_I]_+ = z^\star_I$. For the other set of components, which we denote by I^c, we have $s^\star_{I^c} = 0$, and hence $x^\star$ follows the complementary conditions as in the feasible case

$$[Cx^\star - u^c_I + z^\star_{I^c}]_+ - z^\star_{I^c} = 0.$$

Therefore, it follows that (6.16) captures the two cases (that is, both feasible and infeasible QPs), which concludes the first part of the proof.

Finally, introducing the slack variable $t^\star = C(\theta)x^\star - u(\theta)$, we have

$$\mathcal{G}(x^\star, z^\star;\ \theta) = 0 \tag{6.17}$$

$$\Leftrightarrow \begin{bmatrix} H(\theta)x^\star + g(\theta) + C(\theta)^\top z^\star \\ [[C(\theta)x^\star - u(\theta)]_- + z^\star]_+ - z^\star \\ C(\theta)^\top [C(\theta)x^\star - u(\theta)]_+ \end{bmatrix} = 0 \tag{6.18}$$

$$\Leftrightarrow \begin{bmatrix} H(\theta)x^\star + g(\theta) + C(\theta)^\top z^\star \\ C(\theta)x^\star - u(\theta) - t^\star \\ [[t^\star]_- + z^\star]_+ - z^\star \\ C(\theta)^\top [t^\star]_+ \end{bmatrix} = 0 \tag{6.19}$$

$$G(x^\star, z^\star, t^\star;\ \theta) = 0. \tag{6.20}$$

Hence $G(x^\star, z^\star, t^\star) = 0$ iff $(x^\star, z^\star, [t^\star]_+)$ solves (QP-H), which concludes the proof.

□

6.4.2 Proof of Lemma 6

Lemma 6. *G is path differentiable w.r.t. $x^\star$, $z^\star$ and $t^\star$. Furthermore, if $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ are differentiable w.r.t. θ, then G is path differentiable w.r.t. θ.*

Proof. We start with the first claim of the lemma. The non-negative projector $[.]_+$ is (component-wise) convex, and hence path differentiable [Bolte and Pauwels, 2020, Proposition 2(i)]. Thus, it remains to show that the third component of G is path differentiable for reaching the desired conclusion.

To do so, we show that the third component is Lipschitz continuous and real semialgebraic [Bolte and Pauwels, 2020, Proposition 2(iv)]. Without loss of generality, we restrict ourselves to the case with 2 components (one for the dual variables, and one for the slack variables) using the following function $h : \mathbb{R}^2 \to \mathbb{R}$, s.t. $h(z, s) \triangleq [[s]_- + z]_+ - z$. Then, the following Lipschitzness argument applies component-wise.

Let $(s_1, z_1) \in \mathbb{R}^2$ and $(s_2, z_2) \in \mathbb{R}^2$:

$$
\begin{aligned}
&|h(z_1, s_1) - h(z_2, s_2)| \leq |[[s_1]_- + z_1]_+ - [[s_2]_- + z_2]_+| + |z_1 - z_2| \\
&\leq |[s_1]_- + z_1 - [s_2]_- - z_2| + |z_1 - z_2| \text{ by monotonicity of } [.]_+ \\
&\leq |[s_1]_- - [s_2]_-| + 2|z_1 - z_2| \\
&= |s_1 - [s_1]_+ - (s_2 - [s_2]_+)| + 2|z_1 - z_2| \\
&\leq |s_1 - s_2| + |[s_1]_+ - [s_2]_+| + 2|z_1 - z_2| \\
&\leq 2|s_1 - s_2| + 2|z_1 - z_2| \text{ by monotonicity of } [.]_+ \\
&\leq 2\sqrt{2} \| \begin{bmatrix} s_1 \\ z_1 \end{bmatrix} - \begin{bmatrix} s_2 \\ z_2 \end{bmatrix} \|_2.
\end{aligned}
$$

Therefore, h is Lipschitz continuous. For showing that the third component G describes a semi-algebraic set, we explicitly formulate the graph of h as a finite union of base semi-algebraic sets, as follows:

$$
\begin{aligned}
\text{gph}(h) =& \{(z, s, y) \in \mathbb{R}^3 | y = [s]_- \text{ and } [s]_- + z > 0\} \\
&\cup \{(z, s, y) \in \mathbb{R}^3 | y = -z \text{ and } [s]_- + z = 0\} \\
&\cup \{(z, s, y) \in \mathbb{R}^3 | y = -z \text{ and } [s]_- + z < 0\} \\
=& \{(z, s, y) \in \mathbb{R}^3 | y = s \text{ and } s + z > 0 \text{ and } s < 0\} \\
&\cup \{(z, s, y) \in \mathbb{R}^3 | y = s \text{ and } s = 0\} \\
&\cup \{(z, s, y) \in \mathbb{R}^3 | y = 0 \text{ and } z > 0 \text{ and } s > 0\} \\
&\cup \{(z, s, y) \in \mathbb{R}^3 | y = -z \text{ and } s + z = 0 \text{ and } s < 0\} \\
&\cup \{(z, s, y) \in \mathbb{R}^3 | y = -z \text{ and } z = 0 \text{ and } s = 0\} \\
&\cup \{(z, s, y) \in \mathbb{R}^3 | y = -z \text{ and } z = 0 \text{ and } s > 0\} \\
&\cup \{(z, s, y) \in \mathbb{R}^3 | y = -z \text{ and } s + z < 0 \text{ and } s < 0\} \\
&\cup \{(z, s, y) \in \mathbb{R}^3 | y = -z \text{ and } z < 0 \text{ and } s = 0\} \\
&\cup \{(z, s, y) \in \mathbb{R}^3 | y = -z \text{ and } z < 0 \text{ and } s > 0\}.
\end{aligned}
$$

Hence, gph(h) is real and semi-algebraic as it is a finite union of sets defined by polynomial equalities and inequalities. Hence h is Lipschitz continuous and real semi-algebraic, thereby reaching the target conclusion for the first part of Lemma 6.

As for the second part of Lemma 6. G is linear, and hence differentiable, w.r.t. $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$. Furthermore, by assumption $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ are differentiable w.r.t. θ. As differentiability implies path-differentiability [Bolte and Pauwels, 2020, Remark 3b], we arrive at the desired claim by the conservativity of the chain rule for path-differentiable functions [Bolte and Pauwels, 2020, Proposition 2]. □

6.4.3 Forward and backward AD for computing ECJs

This section provides technical details for the computation of ECJs in forward and backward modes for both primal feasible and infeasible problems. We further include the proofs of **??** and Lemma 7.

General case

Forward pass. Let $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ be differentiable w.r.t. θ and satisfying Assumption 3, and $x^\star, z^\star, t^\star$ s.t. $G(x^\star, z^\star, t^\star;\ \theta) = 0$.

As G is path-differentiable w.r.t. $v^\star \triangleq (x^\star, z^\star, t^\star)$ (see Lemma 6), we have

$$\begin{bmatrix} H & C^\top & 0 \\ C & 0 & -I \\ 0 & \Pi_1 - I & \Pi_1\Pi_2 \\ 0 & 0 & C^\top\Pi_3 \end{bmatrix} \in \frac{\partial G(x^\star, z^\star, t^\star;\ \theta)}{\partial v^\star},$$

for some $\Pi_1 \in \partial[.]_+([t^\star]_- + z^\star)$, $\Pi_2 \in \partial[.]_-(t^\star)$ and $\Pi_3 \in \partial[.]_+(t^\star)$.

Furthermore, as G is linear w.r.t. $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ and $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ are differentiable w.r.t. θ, the usual chain rule dictates that

$$\frac{\partial G(x^\star, z^\star, t^\star;\ \theta)}{\partial \theta} = \begin{bmatrix} \frac{\partial H}{\partial \theta}x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\partial C}{\partial \theta}x^\star - \frac{\partial u}{\partial \theta} \\ 0 \\ \frac{\partial C}{\partial \theta}^\top [t^\star]_+ \end{bmatrix}.$$

Finally, as $\partial[.]_+(0) = \partial[.]_-(0) = [0, 1]$, we make the following arbitrary choices in zeros

$$\begin{aligned} &\Pi_1 = I \text{ when } [t^\star]_- + z^\star = 0, \\ &\Pi_2 = I \text{ when } t^\star = 0, \\ &\Pi_3 = 0 \text{ when } t^\star = 0, \end{aligned} \tag{6.21}$$

so that $\Pi_3 = I - \Pi_2$. ECJs are thus retrieved as solutions to:

$$\frac{\partial x^\star}{\partial \theta}, \frac{\partial z^\star}{\partial \theta}, \frac{\partial t^\star}{\partial \theta} \in \underset{\frac{\partial x}{\partial \theta}, \frac{\partial z}{\partial \theta}, \frac{\partial t}{\partial \theta}}{\arg\min} \left\| \begin{bmatrix} H & C^\top & 0 \\ C & 0 & -I \\ 0 & \Pi_1 - I & \Pi_1\Pi_2 \\ 0 & 0 & C^\top(I - \Pi_2) \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \theta} \\ \frac{\partial z}{\partial \theta} \\ \frac{\partial t}{\partial \theta} \end{bmatrix} + \begin{bmatrix} \frac{\partial H}{\partial \theta}x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\partial C}{\partial \theta}x^\star - \frac{\partial u}{\partial \theta} \\ 0 \\ \frac{\partial C}{\partial \theta}^\top [t^\star]_+ \end{bmatrix} \right\|_2^2. \tag{6.22}$$

In practice, solving this problem can be done via an augmented Lagrangian-based solver for the problem:

$$\begin{aligned} \frac{\partial x^\star}{\partial \theta}, \frac{\partial z^\star}{\partial \theta}, \frac{\partial t^\star}{\partial \theta} \in \underset{\frac{\partial x}{\partial \theta}, \frac{\partial z}{\partial \theta}, \frac{\partial t}{\partial \theta}}{\arg\min}\ & 0 \\ \text{s.t.} \begin{bmatrix} H & C^\top & 0 \\ C & 0 & -I \\ 0 & \Pi_1 - I & \Pi_1\Pi_2 \\ 0 & 0 & C^\top(I - \Pi_2) \end{bmatrix} & \begin{bmatrix} \frac{\partial x}{\partial \theta} \\ \frac{\partial z}{\partial \theta} \\ \frac{\partial t}{\partial \theta} \end{bmatrix} = - \begin{bmatrix} \frac{\partial H}{\partial \theta}x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\partial C}{\partial \theta}x^\star - \frac{\partial u}{\partial \theta} \\ 0 \\ \frac{\partial C}{\partial \theta}^\top [t^\star]_+ \end{bmatrix} \end{aligned} \tag{6.23}$$

when this problem is feasible. When it is not feasible, the augmented Lagrangian naturally converges to the solution of the more general (6.22), see [Chiche and Gilbert, 2016, Proposition 4.2]. In comparison to (6.22), a notable advantage of the formulation (6.23) is that it is naturally numerically more stable and sparse–by avoiding square matrix products from the objective.

Backward pass. The following lemma formally details the results from Equation 6.2.4.

Lemma 8. *Let $h : \mathbb{R}^n \times (\mathbb{R}^{n_i})^2 \to \mathbb{R}$ be a differentiable function, and let $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ be differentiable w.r.t. θ and satisfying Assumption 3. Then, denoting $\mathcal{L}(\theta) \triangleq h(x^\star(\theta), z^\star(\theta), s^\star(\theta))$ and under assumptions of **??**, we have that $\frac{\partial \mathcal{L}}{\partial \theta}$ can be derived as follows*

$$\frac{\partial \mathcal{L}}{\partial \theta} = (b_1^\star)^\top \frac{\partial H}{\partial \theta}x^\star + (b_1^\star)^\top \frac{\partial g}{\partial \theta} + (b_2^\star)^\top \frac{\partial C}{\partial \theta}x^\star + (z^\star)^\top \frac{\partial C}{\partial \theta}b_1^\star + (s^\star)^\top \frac{\partial C}{\partial \theta}b_4^\star - (b_2^\star)^\top \frac{\partial u}{\partial \theta},$$

where $b_1^\star$, $b_2^\star$, $b_3^\star$ and $b_4^\star$ are the solutions of the linear system

$$\begin{bmatrix} H & C^\top & 0 & 0 \\ C & 0 & (I-\Pi_1) & 0 \\ 0 & -I & -\Pi_1\Pi_2 & (1-\Pi_2)C \end{bmatrix} \begin{bmatrix} b_1^\star \\ b_2^\star \\ b_3^\star \\ b_4^\star \end{bmatrix} = - \begin{bmatrix} \frac{\delta \mathcal{L}}{\delta x^\star} \\ \frac{\delta \mathcal{L}}{\delta z^\star} \\ \frac{\delta \mathcal{L}}{\delta s^\star} \end{bmatrix}.$$

Proof. Under the assumptions of **??**, it holds that $\begin{bmatrix} \frac{\partial x^\star}{\partial \theta} \\ \frac{\partial z^\star}{\partial \theta} \\ \frac{\partial s^\star}{\partial \theta} \end{bmatrix}$ is a CJ as $\begin{bmatrix} \frac{\partial x^\star}{\partial \theta} \\ \frac{\partial z^\star}{\partial \theta} \\ \frac{\partial t^\star}{\partial \theta} \end{bmatrix}$ and $\Pi_1 \frac{\partial t^\star}{\partial \theta} \in \frac{\partial s^\star}{\partial \theta}$ are CJs ([Bolte and Pauwels, 2020, Proposition 2]). Furthermore, as $\mathcal{L}$ is differentiable w.r.t. $x^\star$, $z^\star$ and $s^\star$, it is path-differentiable [Bolte and Pauwels, 2020, Remark 3b] w.r.t. $x^\star, z^\star, s^\star$, so we can apply chain rule ([Bolte and Pauwels, 2020, Proposition 2]):

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \theta} &= \frac{\partial \mathcal{L}}{\partial x^\star}^\top \frac{\partial x^\star}{\partial \theta} + \frac{\partial \mathcal{L}}{\partial z^\star}^\top \frac{\partial z^\star}{\partial \theta} + \frac{\partial \mathcal{L}}{\partial s^\star}^\top \frac{\partial s^\star}{\partial \theta} \\
&= -(- \begin{bmatrix} \frac{\delta \mathcal{L}}{\delta x^\star} \\ \frac{\delta \mathcal{L}}{\delta z^\star} \\ \frac{\delta \mathcal{L}}{\delta s^\star} \end{bmatrix})^\top \begin{bmatrix} \frac{\partial x^\star}{\partial \theta} \\ \frac{\partial z^\star}{\partial \theta} \\ \frac{\partial s^\star}{\partial \theta} \end{bmatrix} \\
&= -(\begin{bmatrix} H & C^\top & 0 \\ C & 0 & -I \\ 0 & \Pi_1 - I & \Pi_1\Pi_2 \\ 0 & 0 & C^\top(I-\Pi_2) \end{bmatrix} \begin{bmatrix} b_1^\star \\ b_2^\star \\ b_3^\star \\ b_4^\star \end{bmatrix})^\top \begin{bmatrix} \frac{\partial x^\star}{\partial \theta} \\ \frac{\partial z^\star}{\partial \theta} \\ \frac{\partial s^\star}{\partial \theta} \end{bmatrix} \\
&= - \begin{bmatrix} b_1^\star \\ b_2^\star \\ b_3^\star \\ b_4^\star \end{bmatrix}^\top (- \begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta} \\ 0 \\ \frac{\partial C}{\partial \theta}^\top [t^\star]_+ \end{bmatrix}) \\
&= (b_1^\star)^\top (\frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star) \\
&+ (b_2^\star)^\top (\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) \\
&+ (b_4^\star)^\top (\frac{\partial C}{\partial \theta}^\top [t^\star]_+) \\
&= (b_1^\star)^\top \frac{\partial H}{\partial \theta} x^\star + (\frac{\partial C}{\partial \theta} b_1^\star)^\top z^\star + (b_1)^\top \frac{\partial g}{\partial \theta} \\
&+ (b_2^\star)^\top (\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) \\
&+ (\frac{\partial C}{\partial \theta} b_4^\star)^\top [t^\star]_+),
\end{aligned}$$

with $b_1^\star, b_2^\star$, $b_4^\star$ solution of the nonsingular system (see **??**)

$$\begin{bmatrix} H & C^\top & 0 & 0 \\ C & 0 & (I-\Pi_1) & 0 \\ 0 & -I & \Pi_1\Pi_2 & (1-\Pi_2)C \end{bmatrix} \begin{bmatrix} b_1^\star \\ b_2^\star \\ b_3^\star \\ b_4^\star \end{bmatrix} = - \begin{bmatrix} \frac{\delta \mathcal{L}}{\delta x^\star} \\ \frac{\delta \mathcal{L}}{\delta z^\star} \\ \frac{\delta \mathcal{L}}{\delta s^\star} \end{bmatrix},$$

thereby reaching the desired statement with $s^\star = [t^\star]_+$. □

Simplification of when QP is feasible

Simplification of the forward pass When QP(θ) is feasible and $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ are differentiable w.r.t. θ and satisfy Assumption 3, then $[t^\star]_+ = 0$. Further, our choices of

subgradients at zero (see (6.21)) imply that $\Pi_2 = I$ and hence the following simplifications

$$\begin{bmatrix} H & C^\top & 0 \\ C & 0 & -I \\ 0 & \Pi_1 - I & \Pi_1 \\ 0 & 0 & 0 \end{bmatrix} \in \frac{\partial G(x^\star, z^\star, t^\star;\ \theta)}{\partial v^\star},$$
$$\begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta} \\ 0 \\ 0 \end{bmatrix} = \frac{\partial G(x^\star, z^\star, t^\star;\ \theta)}{\partial \theta}.$$

Moreover, the optimality conditions of

$$\frac{\partial x^\star}{\partial \theta}, \frac{\partial z^\star}{\partial \theta}, \frac{\partial t^\star}{\partial \theta} \in \underset{\frac{\partial x}{\partial \theta}, \frac{\partial z}{\partial \theta}, \frac{\partial t}{\partial \theta}}{\arg\min} \left\| \begin{bmatrix} H & C^\top & 0 \\ C & 0 & -I \\ 0 & \Pi_1 - I & \Pi_1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \theta} \\ \frac{\partial z}{\partial \theta} \\ \frac{\partial t}{\partial \theta} \end{bmatrix} + \begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta} \\ 0 \\ 0 \end{bmatrix} \right\|_2^2,$$

write down

$$\begin{aligned} (H^2 + C^\top C)\frac{\partial x^\star}{\partial \theta} + HC^\top \frac{\partial z^\star}{\partial \theta} - C^\top \frac{\partial t^\star}{\partial \theta} + [H(\frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star) + C^\top (\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta})] = 0, \\ CH\frac{\partial x^\star}{\partial \theta} + (C^\top C + I - \Pi_1)\frac{\partial z^\star}{\partial \theta} + C(\frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star) = 0, \\ -C\frac{\partial x^\star}{\partial \theta} + (I + \Pi_1)\frac{\partial t^\star}{\partial \theta} - (\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) = 0. \end{aligned} \tag{6.24}$$

Third equation of (6.24) leads to

$$\frac{\partial t^\star}{\partial \theta} = \frac{1}{1 + \Pi_1}(C\frac{\partial x^\star}{\partial \theta} + (\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta})).$$

Hence, optimality conditions without variable $\frac{\partial t^\star}{\partial \theta}$ reduce to

$$\begin{aligned} (H^2 + C^\top \frac{\Pi_1}{1 + \Pi_1} C)\frac{\partial x^\star}{\partial \theta} + HC^\top \frac{\partial z^\star}{\partial \theta} + C^\top \frac{\Pi_1}{1 + \Pi_1}(\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) + H(\frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star) = 0, \\ CH\frac{\partial x^\star}{\partial \theta} + (C^\top C + I - \Pi_1)\frac{\partial z^\star}{\partial \theta} + C(\frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star) = 0. \end{aligned} \tag{6.25}$$

Furthermore, the following problem

$$\min_{\frac{\partial x}{\partial \theta}, \frac{\partial z}{\theta}} \left\| \begin{bmatrix} H & C^\top \\ \frac{\Pi_1}{\sqrt{1+\Pi_1}} C & \Pi_1 - I \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \theta} \\ \frac{\partial z}{\partial \theta} \end{bmatrix} + \begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\Pi_1}{\sqrt{1+\Pi_1}} (\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) \end{bmatrix} \right\|_2^2,$$

have the same KKT conditions as (6.25), thereby allowing to simplify the problem as follows:

$$\begin{aligned} &\min_{\frac{\partial x}{\partial \theta}, \frac{\partial z}{\partial \theta}, \frac{\partial t}{\partial \theta}} \left\| \frac{\partial G(x^\star, z^\star, s^\star;\ \theta)}{\partial \hat{v}^\star} \begin{bmatrix} \frac{\partial x}{\partial \theta} \\ \frac{\partial z}{\partial \theta} \\ \frac{\partial t}{\partial \theta} \end{bmatrix} + \frac{\partial G(x^\star, z^\star, s^\star;\ \theta)}{\partial \theta} \right\|_2^2 \\ &= \min_{\frac{\partial x}{\partial \theta}, \frac{\partial z}{\partial \theta}} \left\| \begin{bmatrix} H & C^\top \\ \frac{\Pi_1}{\sqrt{I+\Pi_1}} C & \Pi_1 - I \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \theta} \\ \frac{\partial z}{\partial \theta} \end{bmatrix} + \begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\Pi_1}{\sqrt{I+\Pi_1}} (\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) \end{bmatrix} \right\|_2^2, \end{aligned} \tag{6.26}$$

Hence

$$\frac{\partial x^\star}{\partial \theta}, \frac{\partial z^\star}{\partial \theta}, \frac{\partial t^\star}{\partial \theta} \in \underset{\frac{\partial x}{\partial \theta}, \frac{\partial z}{\partial \theta}, \frac{\partial t}{\partial \theta}}{\arg\min} \left\| \frac{\partial G(x^\star, z^\star, s^\star;\ \theta)}{\partial \hat{v}^\star} \begin{bmatrix} \frac{\partial x}{\partial \theta} \\ \frac{\partial z}{\partial \theta} \\ \frac{\partial t}{\partial \theta} \end{bmatrix} + \frac{\partial G(x^\star, z^\star, s^\star;\ \theta)}{\partial \theta} \right\|_2^2$$

is equivalent to

$$\begin{aligned} \frac{\partial x^\star}{\partial \theta}, \frac{\partial z^\star}{\partial \theta} &\in \underset{\frac{\partial x}{\partial \theta}, \frac{\partial z}{\partial \theta}}{\arg\min} \left\| \begin{bmatrix} H & C^\top \\ \frac{\Pi_1}{\sqrt{1+\Pi_1}} C & \Pi_1 - I \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \theta} \\ \frac{\partial z}{\partial \theta} \end{bmatrix} + \begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\Pi_1}{\sqrt{1+\Pi_1}} (\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) \end{bmatrix} \right\|_2^2, \\ \frac{\delta t^\star}{\delta \theta} &= (I + \Pi_1)^{-1} (C \frac{\delta x^\star}{\delta \theta} + \frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}). \end{aligned}$$

Simplification of the backward pass. This sections details the results from Lemma 6.2.4.

Lemma 9. *Let $h : \mathbb{R}^n \times (\mathbb{R}^{n_i}) \to \mathbb{R}$ be a differentiable function, and let $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ be differentiable w.r.t. θ and satisfying Assumption 3. Then, denoting $\mathcal{L}(\theta) \triangleq h(x^\star(\theta), z^\star(\theta))$, we have under assumptions of Lemma 7 that $\frac{\partial \mathcal{L}}{\partial \theta}$ can be derived as follows*

$$\frac{\partial \mathcal{L}}{\partial \theta} = (b_x^\star)^\top \frac{\partial H}{\partial \theta} x^\star + (b_x^\star)^\top \frac{\partial g}{\partial \theta} + (\Pi_1 b_z^\star)^\top \frac{\partial C}{\partial \theta} x^\star + (z^\star)^\top \frac{\partial C}{\partial \theta} b_x^\star - (\Pi_1 b_z^\star)^\top \frac{\partial u}{\partial \theta},$$

with $b_x^\star$, $b_z^\star$, the solution of the following linear system

$$\begin{bmatrix} H & C^\top \Pi_1 \\ C & -(I - \Pi_1) \end{bmatrix} \begin{bmatrix} b_x \\ b_z \end{bmatrix} = - \begin{bmatrix} \frac{\delta \mathcal{L}}{\delta x^\star} \\ \frac{\delta \mathcal{L}}{\delta z^\star} \end{bmatrix},$$

Furthermore, this latter linear system can be solved using iterative refinement.

Proof. Under the assumptions of Lemma 7, it holds that $\begin{bmatrix} \frac{\partial x^\star}{\partial \theta} \\ \frac{\partial z^\star}{\partial \theta} \end{bmatrix}$ is a Jacobian. Furthermore, as $\mathcal{L}$ is differentiable, the chain rule implies that:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta} &= \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial x^\star} \\ \frac{\partial \mathcal{L}}{\partial z^\star} \end{bmatrix}^\top \begin{bmatrix} \frac{\partial x^\star}{\partial \theta} \\ \frac{\partial z^\star}{\partial \theta} \end{bmatrix} \\ &= -\left(-\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial x^\star} \\ \frac{\partial \mathcal{L}}{\partial z^\star} \end{bmatrix}\right)^\top \begin{bmatrix} \frac{\partial x^\star}{\partial \theta} \\ \frac{\partial z^\star}{\partial \theta} \end{bmatrix} \\ &= -\left(\begin{bmatrix} H & C^\top \\ \Pi_1 C & \Pi_1 - I \end{bmatrix}^\top \begin{bmatrix} b_x \\ b_z \end{bmatrix}\right)^\top \begin{bmatrix} \frac{\partial x^\star}{\partial \theta} \\ \frac{\partial z^\star}{\partial \theta} \end{bmatrix} \text{ as the matrix is nonsingular (see Lemma 7)} \\ &= -\begin{bmatrix} b_x \\ b_z \end{bmatrix}^\top \left(-\begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \Pi_1 (\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) \end{bmatrix}\right) \\ &= (b_x)^\top (\frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star) \\ &\quad + (b_z)^\top (\Pi(\mathrm{d}C x^\star - \mathrm{d}u)) \\ &= [(b_x)^\top \frac{\partial H}{\partial \theta} x^\star + (\frac{\partial C}{\partial \theta} b_x)^\top z^\star + (b_x)^\top \frac{\partial g}{\partial \theta}] \\ &\quad + (b_z)^\top (\Pi(\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta})), \end{aligned}$$

where (b_x, b_z) is a solution to

$$\begin{bmatrix} H & C^\top \Pi_1 \\ C & -(I - \Pi_1) \end{bmatrix} \begin{bmatrix} b_x \\ b_z \end{bmatrix} = - \begin{bmatrix} \frac{\delta \mathcal{L}}{\delta x^\star} \\ \frac{\delta \mathcal{L}}{\delta z^\star} \end{bmatrix}.$$

As detailed in the proof of Lemma 7 (see details in Section 6.4.4), one can equivalently solve

$$\begin{bmatrix} H & C_J^\top \\ C_J & 0 \end{bmatrix} \begin{bmatrix} b_x \\ b_{z_J} \end{bmatrix} = - \begin{bmatrix} \frac{\delta \mathcal{L}}{\delta x^\star} \\ \frac{\delta \mathcal{L}}{\delta z_J^\star} \end{bmatrix},$$
$$b_{z_J^c} = \frac{\delta \mathcal{L}}{\delta z_{J^c}^\star},$$

with J^c the index set for which the solution is strictly feasible (i.e., $i \in [1, n_i], (\Pi_1)_i = 0$), and J the set of active constraints (i.e., for which $(\Pi_1)_i = 1$). Such linear systems can be solved e.g., via iterative refinement (as the matrix involved is symmetric positive semi-definite [Parikh and Boyd, 2014, Section 4.1.2]). □

6.4.4 Proof of Lemma 7

This section details the proof of Lemma 7, ensuring that, under some regularity assumptions, ECJs reduce to standard Jacobians.

Lemma 7. *If QP(θ) is feasible and $H(\theta)$, $g(\theta)$, $C(\theta)$ and $u(\theta)$ are differentiable w.r.t. θ and satisfy Assumption 3, and if the KKT matrix of active constraints is nonsingular and $x^\star$, $z^\star$ satisfy strict complementarity, then the ECJs matches the standard Jacobian, i.e., $\frac{\partial x^\star(\theta)}{\partial \theta} = \nabla x^\star(\theta)$ and $\frac{\partial z^\star(\theta)}{\partial \theta} = \nabla z^\star(\theta)$.*

Proof. If (QP(θ)) is feasible, then the ECJs of $x^\star$ and $z^\star$ w.r.t. θ are provided by

$$\frac{\partial x^\star}{\partial \theta}, \frac{\partial z^\star}{\partial \theta} \in \underset{\frac{\partial x}{\partial \theta}, \frac{\partial z}{\partial \theta}}{\arg\min} \left\| \underbrace{\begin{bmatrix} H & C^\top \\ \frac{\Pi_1}{\sqrt{1+\Pi_1}} C & \Pi_1 - I \end{bmatrix}}_{\triangleq \Delta} \begin{bmatrix} \frac{\partial x}{\partial \theta} \\ \frac{\partial z}{\partial \theta} \end{bmatrix} + \begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\Pi_1}{\sqrt{1+\Pi_1}} (\frac{\partial C}{\partial \theta} x^\star - \frac{\partial u}{\partial \theta}) \end{bmatrix} \right\|_2^2, \tag{6.27}$$

where Π_1 corresponds to a binary diagonal matrix of the complementarity conditions $Cx^\star - u + z^\star \geq 0$. Denoting by J^c the index set for which the solution is strictly feasible (i.e., $i \in [1, n_i], (\Pi_1)_i = 0$), and by J the index set of active constraints (i.e., for which $(\Pi_1)_i = 1$) then Δ can be reformulated as follows (by strict complementary)

$$\Delta = \begin{bmatrix} H & C_J^\top & C_{J^c}^\top \\ \frac{1}{\sqrt{2}} C_J & 0 & 0 \\ 0 & 0 & -I \end{bmatrix},$$

with I being the identity matrix of appropriate dimension. Furthermore, the right-hand side of the linear system within the ℓ_2 norm becomes

$$\begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{1}{\sqrt{2}} (\frac{\partial C_J}{\partial \theta} x^\star - \frac{\partial u_J}{\partial \theta}) \\ 0 \end{bmatrix}.$$

$\begin{bmatrix} H & C_J^\top \\ C_J & 0 \end{bmatrix}$ corresponds to the KKT matrix of active constraints, and is nonsingular by assumption. As it implies nonsingularity of $\begin{bmatrix} H & C_J^\top \\ \frac{1}{\sqrt{2}} C_J & 0 \end{bmatrix}$, it follows that :

$$\frac{\partial z^\star_{J^c}}{\partial \theta} = 0,$$

and the solution to (6.27) is uniquely determined as the solution of the following linear system (as in [Amos and Kolter, 2017, Appendix A], after multiplying second row block by $\sqrt{2}$):

$$\begin{bmatrix} H & C_J^\top \\ C_J & 0 \end{bmatrix} \begin{bmatrix} \frac{\partial x^\star}{\partial \theta} \\ \frac{\partial z^\star}{\partial \theta} \end{bmatrix} = - \begin{bmatrix} \frac{\partial H}{\partial \theta} x^\star + \frac{\partial g}{\partial \theta} + \frac{\partial C}{\partial \theta}^\top z^\star \\ \frac{\partial C_J}{\partial \theta} x^\star - \frac{\partial u_J}{\partial \theta} \end{bmatrix},$$

Hence, we arrive at the desired conclusion that ECJ coincides with the usual Jacobian in this case.

□

6.4.5 Experimental setting

Description of the cart-pole problem

The cart-pole system [Anderson, 1989] is a classic control problem used for benchmarking control algorithms. The system we consider consists of an extended model with dry friction on the joints of the cart-pole, namely on the prismatic cart joint and the revolute joint of the pole. It makes the dynamics non-smooth. It is described by a set of differential equations relating the position, velocity, acceleration, angle, and angular velocity of the cart and pole plus the additional friction forces. The static friction forces on each joint can be obtained by solving a QP problem, see (6.28) and [Le Lidec et al., 2021].

Task: The initial position of the cart-pole system consists of the pole hanging down vertically. The objective of the task is to move the cart in such a way as to swing the pole up and keep it balanced in the upright position. To swing the pole up, the control inputs may involve moving the cart back and forth in a particular pattern that generates the necessary forces to overcome the friction and accelerate the pole in the desired direction.

The forward dynamics with friction

$$Ma = \tau + \lambda,$$

can be re-written in terms of velocity and impulses with timestep Δt as

$$v = v_f + M^{-1}\lambda \Delta t = v_f + M^{-1}\Lambda,$$

were M is the inertia matrix of the system, $a \in \mathbb{R}^{n_v}$ is the joint acceleration, $\tau \in \mathbb{R}^{n_v}$ the joint torque, v_f the free velocity of the system without friction and $\lambda \in \mathbb{R}^{n_v}$ the dry friction force on every joint. To obtain the friction impulse Λ corresponding to the friction coefficient η, the following quadratic problem can be solved:

$$\begin{aligned} &\min_{\Lambda} \frac{1}{2}\Lambda^T M^{-1}\Lambda + v_f^T \Lambda \\ &\text{s.t. } |\Lambda| \leq \eta. \end{aligned} \tag{6.28}$$

Its Lagrangian L can be written as follows

$$L(\Lambda, y) := \frac{1}{2}\Lambda^T M^{-1}\Lambda + v_f^T \Lambda + y^T(|\Lambda| - \eta),$$

which leads to the KKT system:

$$M^{-1}\Lambda + v_f + \text{diag}(\text{sign}(\Lambda))y = 0, \tag{6.29}$$
$$|\Lambda| \leq \eta, \tag{6.30}$$
$$y \geq 0, \tag{6.31}$$
$$y \odot [|\Lambda| - \eta] = 0, \tag{6.32}$$

where $\odot$ stands for the standard Hadamard product. Considering the case where the friction force is within the friction cone for a specific joint j, i.e., $|\Lambda_j| < \eta_j$, the joint is then not moving. We see from (6.32) that $y_j = 0$ satisfies (6.31) and we get from (6.29)

$$M^{-1}\Lambda_j = -v_{f,j}.$$

Consequently, the friction impulse is acting in the opposite direction than the joint torque τ_j and with a magnitude that is canceling out the free velocity. If $|\Lambda_j| = \eta_j$, the joint will no longer be blocked by the friction forces and will thus start moving. We see from (6.29) that

$$M^{-1}\Lambda_j + v_{f,j} = v_j = -\text{diag}(\text{sign}(\Lambda_j))y_j. \tag{6.33}$$

As $y \geq 0$, (6.33) shows that Λ is opposed to the velocity of the joint.

Optimal control algorithms such as Differential Dynamic Programming (DDP) can be used to compute the optimal trajectory that minimizes a cost function over a finite time horizon, subject to the dynamics of the cart-pole system and control input constraints.

These methods take advantage of the derivatives of the dynamics to efficiently control physical systems. In the presence of non-smooth dynamics, such a class of algorithms is likely to fail due, for instance, to the presence of discontinuities in the dynamics derivatives or because of the non-informative gradient [Lidec et al., 2022]. In the cart-pole benchmark, randomized smoothing, as proposed by [Lidec et al., 2022,Suh et al., 2021] is used to cope with the non-smooth dynamical system. For the optimal swing-up trajectory with 20 timesteps, 5 random samples with a uniform Gaussian noise are generated. The random noise is applied to the input controls, and the dynamics are calculated for each of them and afterward averaged in the forward pass, resulting in informative gradients in the backward pass. The problem (6.28), has to be solved for every random sample in every timestep.

Chapter 7

Conclusion

7.1 Summary of the Thesis

Modern applications of control problems are computationally challenging and require fast and robust numerical calculations. Quadratic programming plays a pivotal role for these domains, since it is used in numerous practical applications [Redon et al., 2002, Carpentier and Wieber, 2021, Escande et al., 2014, Herzog et al., 2016, Kuindersma et al., 2016, Wieber, 2006a, Wieber, 2006b, Wensing et al., 2022, Leineweber et al., 2003, Houska et al., 2011, Tassa et al., 2014, Jallet et al., 2022a]. Recent advances of differentiable optimization have also illustrated that Quadratic Programming layers offer a rich modeling power and can be effective for solving better certain machine learning tasks [Amos and Kolter, 2017, Bounou et al., 2021, Geng et al., 2020, Lee et al., 2005, Le Lidec et al., 2021, Amos et al., 2017, Donti et al., 2017].

In this thesis we have introduced a new algorithm for solving generic QPs, motivated by robotic and control applications. Among others, we notably propose to combine the bound constraint Lagrangian (BCL) globalization strategy [Conn et al., 1991] for automatically scheduling key parameters of a primal-dual proximal augmented Lagrangian algorithm. The intermediary proximal subproblems are solved in quadratic time via a semi-smooth Newton method. Additionally, we have shown that the ProxQP algorithm features a global convergence guarantee, as well as a few other advantageous numerical properties. Furthermore, we have highlighted that the ProxQP algorithm, and more generally primal-dual Proximal augmented Lagrangian methods, actually solves the *closest* primal-feasible QP—in a classical ℓ_2 sense—if the original QP appears to be primally infeasible. As a corollary of its global convergence proof, ProxQP is also guaranteed to find constant penalization parameters in finite time.

The convergence guarantees come together with an efficient software implementation of ProxQP in the open-source ProxSuite library. It contains several practical features for embedded optimization. The performance of ProxQP is illustrated on different standard sparse and dense robotic and control experiments, including a real-world closed-loop controller application. In particular, we have highlighted how solving the *closest* feasible QPs can be efficiently leveraged in the context of closed-loop convex MPC. Finally, through various benchmarks of the optimization literature, we have also shown that ProxQP performs at the level of state-of-the-art solvers on a large set of generic QP problems.

The study of the *closest* feasible QPs has then motivated us to enrich the expressive capabilities of existing QP layers. We have introduced an approach for differentiating both feasible and infeasible convex quadratic programs in a unified fashion. In particular, by leveraging augmented Lagrangian techniques for solving QP layers that are potentially infeasible, we propose an extended conservative Jacobian formulation for differentiating convex QPs, covering both feasible and infeasible problems. For feasible problems, and when the solution is differentiable, this

reduces to standard Jacobians. We then show how driving towards feasibility at test time a new scope of QP layers using our techniques. We illustrate that training these new types of QP layers leads to better-predicting power for solving some tasks.

We further provide an open-source C++ framework, referred to as QPLayer, which efficiently implements the approach. Through extensive benchmarks and experiments, we have also demonstrated how QPLayer is simultaneously faster and numerically more robust than alternative state-of-the-art optimization layers on traditional learning tasks.

7.2 Perspectives

This thesis also raises new research avenues and opportunities concerning using advanced optimization techniques and solvers in many domains.

Towards Non-linear and Non-Convex Programming: A first area of exploration consists of extending the primal-dual proximal augmented Lagrangian framework developed for non-linear programs, both at the constraint and cost levels, as well as for constrained trajectory optimization problems. This extension builds upon recent endeavors in this direction [Jallet et al., 2022a, Jallet et al., 2022b].

Extending QPLayer to Other Conic Constraints: Another captivating research trajectory involves expanding the QPLayer approach to encompass various conic problems. This line of study poses two fundamental and practical inquiries:

- Firstly, when considering a convex conic problem, what assumptions govern the convergence of PMM (and other splitting operators) towards the "closest" feasible problem? Are there inherent limitations that delineate the scope of this property?
- Secondly, in cases where a conic problem solution lacks differentiability in any form, does a least-squares estimate, serving as an alternative "gradient," possess theoretical guarantees for offering an "informative" direction for minimizing certain losses? More generally, which alternative gradient would be the best fit?

Exploring Practical Applicability of Closest-Feasible Solutions: The initial series of experiments employing the closest feasible QP solutions has yielded promising results within this thesis—whether it pertains to expanding the range of learnable QPs or enables dealing with infeasible random perturbations in control scenarios. Delving further into alternative experiments to practically assess the use of this feature in other contexts is also an interesting subject.

Software Development and Outreach: Regarding software development, I am enthusiastic about providing long-term support to facilitate the dissemination of this approach across research, industrial, and commercial domains. Moreover, I would be happy to extend the capabilities of the ProxSuite library to encompass a broader spectrum of optimization problems.

Bibliography

[Abdelmalek, 1983] Abdelmalek, N. N. (1983). Restoration of images with missing high-frequency components using quadratic programming. *Applied optics*, 22(14):2182–2188. 2, 8

[Adams et al., 1994] Adams, J. W., Sullivan, J. L., Gleeson, D. R., Chang, P., and Hashemi, R. (1994). Application of quadratic programming to fir digital filter design problems. In *Proceedings of 1994 28th Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 314–318. IEEE. 2, 8

[Agrawal et al., 2019] Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. (2019). Differentiable convex optimization layers. *Advances in neural information processing systems*, 32. 79, 83, 87, 91, 93, 95

[Amos and Kolter, 2017] Amos, B. and Kolter, J. Z. (2017). Optnet: Differentiable optimization as a layer in neural networks. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR. 3, 9, 79, 80, 83, 84, 85, 90, 91, 93, 94, 95, 102, 106, 111, 120, 123

[Amos et al., 2018] Amos, B., Rodriguez, I. D. J., Sacks, J., Boots, B., and Kolter, J. Z. (2018). Differentiable MPC for end-to-end planning and control. In *NeurIPS*, pages 8299–8310. 3, 8, 80, 90, 109

[Amos et al., 2017] Amos, B., Xu, L., and Kolter, J. Z. (2017). Input convex neural networks. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 146–155. PMLR. 81, 123

[Andersen et al., 2013] Andersen, M. S., Dahl, J., and Vandenberghe, L. (2013). Cvxopt: A python package for convex optimization. *version 1.1.6*. 29, 60

[Anderson, 1989] Anderson, C. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9(3):31–37. 120

[Andersson et al., 2019] Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M. (2019). CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36. 4, 9

[Arnström et al., 2022] Arnström, D., Bemporad, A., and Axehill, D. (2022). A dual active-set solver for embedded quadratic programming using recursive LDL^T updates. *IEEE Transactions on Automatic Control*, 67(8):4362–4369. 21, 59

[Audren et al., 2014] Audren, H., Vaillant, J., Kheddar, A., Escande, A., Kaneko, K., and Yoshida, E. (2014). Model preview control in multi-contact motion-application to a humanoid robot. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4030–4035. IEEE. 70

[Bambade et al., 2022] Bambade, A., El-Kazdadi, S., Taylor, A., and Carpentier, J. (2022). PROX-QP: Yet another Quadratic Programming Solver for Robotics and beyond. In *RSS 2022 - Robotics: Science and Systems*, New York, United States. 38, 49, 71, 72, 74

[Banjac et al., 2019] Banjac, G., Goulart, P., Stellato, B., and Boyd, S. (2019). Infeasibility detection in the alternating direction method of multipliers for convex optimization. *Journal of Optimization Theory and Applications*, 183:490–519. 16, 31

[Belanger and McCallum, 2016] Belanger, D. and McCallum, A. (2016). Structured prediction energy networks. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 983–992. JMLR.org. 81

[Belanger et al., 2017] Belanger, D., Yang, B., and McCallum, A. (2017). End-to-end learning for structured prediction energy networks. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 429–439. PMLR. 81

[Bemporad, 2015] Bemporad, A. (2015). A quadratic programming algorithm based on nonnegative least squares with applications to embedded model predictive control. *IEEE Transactions on Automatic Control*, 61(4):1111–1116. 21

[Bengio, 2000] Bengio, Y. (2000). Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900. 83

[Berthet et al., 2020] Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., and Bach, F. (2020). Learning with differentiable pertubed optimizers. *Advances in neural information processing systems*, 33:9508–9519. 79, 87

[Bertrand et al., 2020] Bertrand, Q., Klopfenstein, Q., Blondel, M., Vaiter, S., Gramfort, A., and Salmon, J. (2020). Implicit differentiation of lasso-type models for hyperparameter optimization. In *International Conference on Machine Learning*, pages 810–821. PMLR. 83

[Bertsekas, 1982] Bertsekas, D. (1982). Projected newton methods for optimization problems with simple constraints. *SIAM Journal on Control and Optimization*, 20(2):762–767. 37

[Best, 1996] Best, M. J. (1996). *An algorithm for the solution of the parametric quadratic programming problem*. Springer. 25

[Birgin and Martínez, 2014] Birgin, E. G. and Martínez, J. M. (2014). *Practical Augmented Lagrangian Methods for Constrained Optimization*. SIAM. 37, 39, 41

[Bitmead et al., 1986] Bitmead, R., Tsoi, A., and Parker, P. (1986). A kalman filtering approach to short-time fourier analysis. *IEEE transactions on acoustics, speech, and signal processing*, 34(6):1493–1501. 2, 8

[Blondel et al., 2022] Blondel, M., Berthet, Q., Cuturi, M., Frostig, R., Hoyer, S., Llinares-López, F., Pedregosa, F., and Vert, J. (2022). Efficient and modular implicit differentiation. In *NeurIPS*. 79, 83, 87, 93, 95

[Blondel et al., 2020a] Blondel, M., Martins, A. F., and Niculae, V. (2020a). Learning with fenchel-young losses. *Journal of Machine Learning Research*, 21(35):1–69. 88

[Blondel et al., 2020b] Blondel, M., Teboul, O., Berthet, Q., and Djolonga, J. (2020b). Fast differentiable sorting and ranking. In *International Conference on Machine Learning*, pages 950–959. PMLR. 87

[Bolte et al., 2021] Bolte, J., Le, T., Pauwels, E., and Silveti-Falls, A. (2021). Nonsmooth implicit differentiation for machine-learning and optimization. In *NeurIPS*, pages 13537–13549. 81, 85, 91, 95

[Bolte and Pauwels, 2020] Bolte, J. and Pauwels, E. (2020). Conservative set valued fields, automatic differentiation, stochastic gradient method and deep learning. *Mathematical Programming*, 188(19-51). 85, 91, 93, 113, 114, 116

[Bounou et al., 2021] Bounou, O., Ponce, J., and Carpentier, J. (2021). Online learning and control of complex dynamical systems from sensory input. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*. 3, 8, 80, 90, 123

[Boyd et al., 2017] Boyd, S., Busseti, E., Diamond, S., Kahn, R. N., Koh, K., Nystrup, P., Speth, J., et al. (2017). Multi-period trading via convex optimization. *Foundations and Trends® in Optimization*, 3(1):1–76. 2, 8

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press. 2, 8, 13, 14, 15, 16, 28

[Candes et al., 2008] Candes, E. J., Wakin, M. B., and Boyd, S. P. (2008). Enhancing sparsity by reweighted l1 minimization. *Journal of Fourier analysis and applications*, 14:877–905. 2, 8

[Caron et al., 2019] Caron, S., Kheddar, A., and Tempier, O. (2019). Stair climbing stabilization of the hrp-4 humanoid robot using whole-body admittance control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 277–283. 70

[Carpentier et al., 2021] Carpentier, J., Budhiraja, R., and Mansard, N. (2021). Proximal and sparse resolution of constrained dynamic equations. In *Robotics: Science and Systems 2021*. 2, 8

[Carpentier et al., 2019] Carpentier, J., Saurel, G., Buondonno, G., Mirabel, J., Lamiraux, F., Stasse, O., and Mansard, N. (2019). The pinocchio c++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *2019 IEEE/SICE International Symposium on System Integration (SII)*, pages 614–619. IEEE. 68

[Carpentier and Wieber, 2021] Carpentier, J. and Wieber, P.-B. (2021). Recent progress in legged robots locomotion control. *Current Robotics Reports*, 2(3):231–238. 2, 8, 123

[Chapelle et al., 2002] Chapelle, O., Vapnik, V., Bousquet, O., and Mukherjee, S. (2002). Choosing multiple parameters for support vector machines. *Machine learning*, 46:131–159. 83

[Chiche and Gilbert, 2016] Chiche, A. and Gilbert, J. C. (2016). How the augmented Lagrangian algorithm can deal with an infeasible convex quadratic optimization problem. *Journal of Convex Analysis*, 23(2). 36, 38, 91, 92, 94, 112, 115

[Conn et al., 1992a] Conn, A., Gould, N., and Toint, P. L. (1992a). A fortran package for large-scale nonlinear optimization (release a). *Springer Ser. Comput. Math*, 17. 37

[Conn et al., 1992b] Conn, A. R., Gould, N. I., Toint, P. L., and Conn, A. R. (1992b). *On the number of inner iterations per outer iteration of a globally convergent algorithm for optimization with general nonlinear inequality constraints and simple bounds*. Rutherford Appleton Laboratory. 37

[Conn et al., 1991] Conn, A. R., Gould, N. I. M., and Toint, P. (1991). A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572. 36, 37, 41, 46, 47, 48, 51, 123

[Conn et al., 1992c] Conn, A. R., Gould, N. I. M., and Toint, P. L. (1992c). *Lancelot a Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer Berlin Heidelberg, Berlin, Heidelberg. 46

[Cornuejols and Tütüncü, 2006] Cornuejols, G. and Tütüncü, R. (2006). *Optimization methods in finance*, volume 5. Cambridge University Press. 2, 8

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20:273–297. 2, 8

[Cui et al., 2016] Cui, Y., Sun, D., and Toh, K.-C. (2016). On the asymptotic superlinear convergence of the augmented lagrangian method for semidefinite programming with multiple solutions. *arXiv preprint arXiv:1610.00875*. 37

[Dadush and Huiberts, 2018] Dadush, D. and Huiberts, S. (2018). A friendly smoothed analysis of the simplex method. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 390–403. 18

[Dantzig, 1990] Dantzig, G. B. (1990). Origins of the simplex method. In *A history of scientific computing*, pages 141–151. 17

[de Avila Belbute-Peres et al., 2018] de Avila Belbute-Peres, F., Smith, K. A., Allen, K. R., Tenenbaum, J., and Kolter, J. Z. (2018). End-to-end differentiable physics for learning and control. In *NeurIPS*, pages 7178–7189. 3, 8, 80, 90

[De Marchi, 2022] De Marchi, A. (2022). On a primal-dual Newton proximal method for convex quadratic programs. *Computational Optimization and Applications*, 81(2):369–395. 112, 113

[Deb, 2012] Deb, K. (2012). *Optimization for engineering design: Algorithms and examples*. PHI Learning Pvt. Ltd. 1, 7

[Del Prete et al., 2016] Del Prete, A., Mansard, N., Ramos, O., Stasse, O., and Nori, F. (2016). Implementing torque control with high-ratio gear boxes and without joint-torque sensors. In *Int. Journal of Humanoid Robotics*, page 1550044. 4, 9

[Diamond and Boyd, 2016] Diamond, S. and Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5. 4, 9

[Domahidi et al., 2013] Domahidi, A., Chu, E., and Boyd, S. P. (2013). ECOS: an SOCP solver for embedded systems. In *ECC*, pages 3071–3076. IEEE. 29, 60

[Domke, 2012] Domke, J. (2012). Generic methods for optimization-based modeling. In *AISTATS*, volume 22 of *JMLR Proceedings*, pages 318–326. JMLR.org. 81

[Dontchev et al., 2009a] Dontchev, A. L., Rockafellar, R. T., and Rockafellar, R. T. (2009a). *Implicit functions and solution mappings: A view from variational analysis*, volume 616. Springer. 52, 58

[Dontchev et al., 2009b] Dontchev, A. L., Rockafellar, R. T., and Rockafellar, R. T. (2009b). *Implicit functions and solution mappings: A view from variational analysis*, volume 616. Springer. 79, 81, 83, 91, 93

[Donti et al., 2017] Donti, P. L., Kolter, J. Z., and Amos, B. (2017). Task-based end-to-end model learning in stochastic optimization. In *NIPS*, pages 5484–5494. 3, 8, 80, 90, 123

[d'Aspremont et al., 2021] d'Aspremont, A., Scieur, D., and Taylor, A. (2021). Acceleration methods. *Foundations and Trends® in Optimization*, 5(1-2):1–245. 42

[Eckstein, 1989] Eckstein, J. (1989). *Splitting methods for monotone operators with applications to parallel optimization*. PhD thesis, Massachusetts Institute of Technology. 30

[El Ghaoui, 2012] El Ghaoui, L. (2012). *Convex Optimization and Applications*. 15

[El Kazdadi et al., 2021] El Kazdadi, S., Carpentier, J., and Ponce, J. (2021). Equality Constrained Differential Dynamic Programming. In *2021 International Conference on Robotics and Automation (ICRA)*. 46

[Escande et al., 2014] Escande, A., Mansard, N., and Wieber, P.-B. (2014). Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, 33(7):1006–1028. 2, 8, 123

[Facchinei and Pang, 2003] Facchinei, F. and Pang, J.-S. (2003). *Finite-dimensional variational inequalities and complementarity problems*. Springer. 49

[Fan and Zhang, 1998] Fan, J.-Y. and Zhang, L. (1998). Real-time economic dispatch with line flow and emission constraints using quadratic programming. *IEEE Transactions on Power Systems*, 13(2):320–325. 2, 8

[Ferreau, 2006] Ferreau, H. J. (2006). An online active set strategy for fast solution of parametric quadratic programs with applications to predictive engine control. *University of Heidelberg*. 25

[Ferreau et al., 2008] Ferreau, H. J., Bock, H. G., and Diehl, M. (2008). An online active set strategy to overcome the limitations of explicit mpc. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 18(8):816–830. 25

[Ferreau et al., 2014] Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., and Diehl, M. (2014). qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363. 13, 25, 26, 59, 61, 69

[Fiacco and McCormick, 1968] Fiacco, A. V. and McCormick, G. P. (1968). *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley & Sons, New York, NY, USA. Reprinted by SIAM Publications in 1990. 83

[Fortin and Glowinski, 2000] Fortin, M. and Glowinski, R. (2000). *Augmented Lagrangian methods: applications to the numerical solution of boundary-value problems*. Elsevier. 30

[Frison and Diehl, 2020] Frison, G. and Diehl, M. (2020). Hpipm: a high-performance quadratic programming framework for model predictive control. *IFAC-PapersOnLine*, 53(2):6563–6569. 29, 60

[Gabay, 1983] Gabay, D. (1983). Chapter ix applications of the method of multipliers to variational inequalities. In *Studies in mathematics and its applications*, volume 15, pages 299–331. Elsevier. 30

[Gabay and Mercier, 1976] Gabay, D. and Mercier, B. (1976). A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & mathematics with applications*, 2(1):17–40. 30

[Garstka et al., 2021] Garstka, M., Cannon, M., and Goulart, P. (2021). COSMO: A conic operator splitting method for convex conic problems. *Journal of Optimization Theory and Applications*, 190(3):779–810. 30

[Geng et al., 2020] Geng, Z., Johnson, D., and Fedkiw, R. (2020). Coercing machine learning to output physically accurate results. *J. Comput. Phys.*, 406:109099. 3, 8, 80, 90, 123

[Gertz and Wright, 2003] Gertz, E. M. and Wright, S. J. (2003). Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software (TOMS)*, 29(1):58–81. 29, 60

[Gilbert, 2021] Gilbert, J. C. (2021). Fragments d'Optimisation Différentiable - Théories et Algorithmes. Lecture. 83

[Glowinski and Marroco, 1975] Glowinski, R. and Marroco, A. (1975). Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique*, 9(R2):41–76. 30

[Goldfarb and Idnani, 1983] Goldfarb, D. and Idnani, A. (1983). A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27:1–33. 13, 20, 21, 22, 23, 59

[Gould and Toint, 2000] Gould, N. I. and Toint, P. L. (2000). A quadratic programming bibliography. *Numerical Analysis Group Internal Report*, 1:32. 1, 7

[Gould et al., 2017] Gould, N. I. M., Orban, D., and Toint, P. L. (2017). GALAHAD user documentation. *GALAHAD*. 19, 59

[Gould et al., 2016] Gould, S., Fernando, B., Cherian, A., Anderson, P., Cruz, R. S., and Guo, E. (2016). On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv preprint arXiv:1607.05447*. 83

[Gould et al., 2021] Gould, S., Hartley, R., and Campbell, D. (2021). Deep declarative networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):3988–4004. 79, 87

[Guennebaud et al., 2010] Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. 60

[Güler, 1991] Güler, O. (1991). On the convergence of the proximal point algorithm for convex minimization. *SIAM Journal on Control and Optimization*, 29(2):403–419. 95

[Hassanien et al., 2008] Hassanien, A., Vorobyov, S. A., and Wong, K. M. (2008). Robust adaptive beamforming using sequential quadratic programming: An iterative solution to the mismatch problem. *IEEE Signal Processing Letters*, 15:733–736. 2, 8

[Hermans et al., 2019] Hermans, B., Themelis, A., and Patrinos, P. (2019). QPALM: A Newton-type Proximal Augmented Lagrangian Method for Quadratic Programs. *2019 IEEE 58th Conference on Decision and Control (CDC)*. 37, 39

[Hermans et al., 2021] Hermans, B., Themelis, A., and Patrinos, P. (2021). QPALM: A Proximal Augmented Lagrangian Method for Nonconvex Quadratic Programs. 16, 38, 39, 41, 49, 50, 60, 61, 62, 71, 72

[Herzog et al., 2016] Herzog, A., Rotella, N., Mason, S., Grimminger, F., Schaal, S., and Righetti, L. (2016). Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid. *Autonomous Robots*, 40(3):473–491. 2, 8, 123

[Hestenes, 1969] Hestenes, M. R. (1969). Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320. 35, 36

[Houska et al., 2011] Houska, B., Ferreau, H. J., and Diehl, M. (2011). Acado toolkit—an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312. 2, 8, 123

[Huber, 1992] Huber, P. J. (1992). Robust estimation of a location parameter. In *Breakthroughs in statistics: Methodology and distribution*, pages 492–518. Springer. 2, 8

[Huber, 2004] Huber, P. J. (2004). *Robust statistics*, volume 523. John Wiley & Sons. 2, 8

[Jallet et al., 2022a] Jallet, W., Bambade, A., Mansard, N., and Carpentier, J. (2022a). Constrained differential dynamic programming: A primal-dual augmented lagrangian approach. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13371–13378. IEEE. 2, 8, 46, 123, 124

[Jallet et al., 2022b] Jallet, W., Bambade, A., Mansard, N., and Carpentier, J. (2022b). Proxnlp: a primal-dual augmented lagrangian solver for nonlinear programming in robotics and beyond. *arXiv preprint arXiv:2210.02109*. 124

[Jallet et al., 2021] Jallet, W., Mansard, N., and Carpentier, J. (2021). Implicit differential dynamic programming. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 46

[Kajita et al., 2001] Kajita, S., Kanehiro, F., Kaneko, K., Yokoi, K., and Hirukawa, H. (2001). The 3d linear inverted pendulum mode: a simple modeling for a biped walking pattern generation. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, volume 1, pages 239–246 vol.1. 70

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980. 106

[Klee and Minty, 1972] Klee, V. and Minty, G. J. (1972). How good is the simplex algorithm. *Inequalities*, 3(3):159–175. 17

[Krantz and Parks, 2002] Krantz, S. G. and Parks, H. R. (2002). *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media. 79, 81, 87

[Kuindersma et al., 2016] Kuindersma, S., Deits, R., Fallon, M., Valenzuela, A., Dai, H., Permenter, F., Koolen, T., Marion, P., and Tedrake, R. (2016). Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous robots*, 40(3):429–455. 2, 8, 123

[Le Lidec et al., 2021] Le Lidec, Q., Kalevatykh, I., Laptev, I., Schmid, C., and Carpentier, J. (2021). Differentiable simulation for physical system identification. *IEEE Robotics and Automation Letters*, 6(2):3413–3420. 3, 8, 80, 87, 90, 120, 123

[Lee et al., 2005] Lee, G. M., Tam, N. N., Yen, N. D., et al. (2005). *Quadratic programming and affine variational inequalities: a qualitative study*. Springer. 123

[Lee et al., 2019] Lee, K., Maji, S., Ravichandran, A., and Soatto, S. (2019). Meta-learning with differentiable convex optimization. *CoRR*, abs/1904.03758. 3, 8, 80, 90

[Leineweber et al., 2003] Leineweber, D. B., Bauer, I., Bock, H. G., and Schlöder, J. P. (2003). An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization. part 1: theoretical aspects. *Computers & Chemical Engineering*, 27(2):157–166. 2, 8, 123

[Lidec et al., 2022] Lidec, Q. L., Montaut, L., Schmid, C., Laptev, I., and Carpentier, J. (2022). Leveraging randomized smoothing for optimal control of nonsmooth dynamical systems. *CoRR*, abs/2203.03986. 121

[Lions and Mercier, 1979] Lions, P.-L. and Mercier, B. (1979). Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979. 30

[Liu and Zhang, 2016] Liu, M. and Zhang, D. (2016). A dynamic logistics model for medical resources allocation in an epidemic control with demand forecast updating. *Journal of the Operational Research Society*, 67(6):841–852. 2, 8

[Lorraine et al., 2020] Lorraine, J., Vicol, P., and Duvenaud, D. (2020). Optimizing millions of hyperparameters by implicit differentiation. In *International conference on artificial intelligence and statistics*, pages 1540–1552. PMLR. 83

[Luque, 1984] Luque, F. J. (1984). Asymptotic convergence analysis of the proximal point algorithm. *SIAM Journal on Control and Optimization*, 22(2):277–293. 38, 51, 59

[Mandi and Guns, 2010] Mandi, J. and Guns, T. (2010). Interior point solving for lp-based prediction+ optimisation, 2020. *URL http://arxiv. org/abs*. 79, 88

[Marchi., 2021] Marchi., D. (2021). On a primal-dual newton proximal method for convex quadratic programs. *Computational Optimization and Applications*, 76:451—-467. 13, 39, 40, 41, 49, 50, 53

[Markowitz, 1952] Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, 7(1):77–91. 2, 8

[Maros and Mészaros, 1999] Maros, I. and Mészaros, C. (1999). A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11(1-4):671–681. 62, 71

[Marques et al., 2010] Marques, G. F., Lund, J. R., and Howitt, R. E. (2010). Modeling conjunctive use operations and farm decisions with two-stage stochastic quadratic programming. *Journal of Water Resources Planning and Management*, 136(3):386–394. 2, 8

[Martins and Astudillo, 2016] Martins, A. F. T. and Astudillo, R. F. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. 87

[Matskul et al., 2021] Matskul, V., Kovalyov, A., and Saiensus, M. (2021). Optimization of the cold supply chain logistics network with an environmental dimension. In *IOP Conference Series: Earth and Environmental Science*, volume 628, page 012018. IOP Publishing. 2, 8

[Mattingley and Boyd, 2010] Mattingley, J. and Boyd, S. (2010). Real-time convex optimization in signal processing. *IEEE Signal processing magazine*, 27(3):50–61. 2, 8

[McCarl et al., 1977] McCarl, B. A., Moskowitz, H., and Furtan, H. (1977). Quadratic programming applications. *Omega*, 5(1):43–55. 2, 8

[McFarquhar, 1961] McFarquhar, A. (1961). Rational decision making and risk in farm planning–an application of quadratic programming in british arable farming. *Journal of Agricultural Economics*, 14(4):552–563. 2, 8

[Mehrotra, 1992] Mehrotra, S. (1992). On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4):575–601. 13, 28

[Mensch and Blondel, 2018] Mensch, A. and Blondel, M. (2018). Differentiable dynamic programming for structured prediction and attention. In *International Conference on Machine Learning*, pages 3462–3471. PMLR. 87

[Mészáros, 1999] Mészáros, C. (1999). The bpmpd interior point solver for convex quadratic problems. *Optimization Methods and Software*, 11(1-4):431–449. 29, 60

[Metz et al., 2019] Metz, L., Maheswaranathan, N., Nixon, J., Freeman, C. D., and Sohl-Dickstein, J. (2019). Understanding and correcting pathologies in the training of learned optimizers. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 4556–4565. PMLR. 81

[Mifflin, 1977] Mifflin, R. (1977). Semismooth and semiconvex functions in constrained optimization. *Siam Journal on Control*, 15:957–972. 39

[Momoh et al., 1999] Momoh, J. A., Adapa, R., and El-Hawary, M. (1999). A review of selected optimal power flow literature to 1993. i. nonlinear and quadratic programming approaches. *IEEE transactions on power systems*, 14(1):96–104. 2, 8

[Monga et al., 2021] Monga, V., Li, Y., and Eldar, Y. C. (2021). Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Process. Mag.*, 38(2):18–44. 81

[Montaut et al., 2023] Montaut, L., Lidec, Q. L., Bambade, A., Petrik, V., Sivic, J., and Carpentier, J. (2023). Differentiable collision detection: a randomized smoothing approach. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3240–3246. 87

[Monteiro, 1994] Monteiro, R. D. (1994). A globally convergent primal—dual interior point algorithm for convex programming. *Mathematical Programming*, 64(1-3):123–147. 28

[Mosek, 2022] Mosek (2022). MOSEK optserver documentation. *Mosek*. 29, 60

[Nanda et al., 1989] Nanda, J., Kothari, D. P., and Srivastava, S. C. (1989). New optimal power-dispatch algorithm using fletcher's quadratic programming method. In *IEE Proceedings C (Generation, Transmission and Distribution)*, volume 136, pages 153–161. IET. 2, 8

[Niculae and Blondel, 2017] Niculae, V. and Blondel, M. (2017). A regularized framework for sparse and structured neural attention. *Advances in neural information processing systems*, 30. 83

[Niculae et al., 2018] Niculae, V., Martins, A., Blondel, M., and Cardie, C. (2018). Sparsemap: Differentiable sparse structured inference. In *International Conference on Machine Learning*, pages 3799–3808. PMLR. 83

[Nishihara et al., 2015] Nishihara, R., Lessard, L., Recht, B., Packard, A., and Jordan, M. I. (2015). A general analysis of the convergence of admm. 31, 34

[Nocedal and Wright, 2006] Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer series in operations research and financial engineering. Springer, 2nd edition. 3, 9, 13, 17, 18, 19, 20, 21, 25, 28, 29, 32, 35, 36, 37, 40, 46, 50, 87

[Nordebo et al., 1994] Nordebo, S., Claesson, I., and Nordholm, S. (1994). Weighted chebyshev approximation for the design of broadband beamformers using quadratic programming. *IEEE Signal Processing Letters*, 1(7):103–105. 2, 8

[O'Donoghue et al., 2016] O'Donoghue, B., Chu, E., Parikh, N., and Boyd, S. (2016). Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068. 13, 15, 30, 31, 33, 34, 35, 60, 61

[Optimization, 2020] Optimization, G. (2020). GUROBI optimizer reference manual. *Gurobi Optimization*. 29, 60

[Paige and Saunders, 1982] Paige, C. C. and Saunders, M. A. (1982). Lsqr: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software (TOMS)*, 8(1):43–71. 87

[Pandala et al., 2019] Pandala, A. G., Ding, Y., and Park, H.-W. (2019). qpSWIFT: A Real-Time Sparse Quadratic Program Solver for Robotic Applications. *IEEE Robotics and Automation Letters*, 4(4):3355–3362. 29, 60

[Parikh and Boyd, 2014] Parikh, N. and Boyd, S. (2014). Proximal algorithms. *Found. Trends Optim.*, 1(3):127–239. 30, 36, 87, 95, 119

[Plancher et al., 2017] Plancher, B., Manchester, Z., and Kuindersma, S. (2017). Constrained unscented dynamic programming. In *2017 International Conference on Intelligent Robots and Systems (IROS)*, pages 5674–5680. 46

[Potra and Wright, 2000] Potra, F. A. and Wright, S. J. (2000). Interior-point methods. *Journal of computational and applied mathematics*, 124(1-2):281–302. 28

[Powell, 1969] Powell, M. J. D. (1969). A method for nonlinear constraints in minimization problems. *Optimization*, pages 283–298. 35, 36

[Rao, 1961] Rao, S. R. A. (1961). Quadratic programming in economics. *Indian Economic Review*, 5(4):385–387. 2, 8

[Redon et al., 2002] Redon, S., Kheddar, A., and Coquillart, S. (2002). Gauss least constraints principle and rigid body simulations. In *2002 International Conference on Robotics and Automation (ICRA)*, volume 1, pages 517–522. IEEE. 2, 8, 123

[Reid and Hasdorff, 1973] Reid, G. F. and Hasdorff, L. (1973). Economic dispatch using quadratic programming. *IEEE Transactions on Power Apparatus and Systems*, PAS-92:2015–2023. 2, 8

[Robinson, 1980] Robinson, S. M. (1980). Strongly regular generalized equations. *Math. Oper. Res.*, 5(1):43–62. 83

[Robinson, 1991] Robinson, S. M. (1991). An implicit-function theorem for a class of nonsmooth functions. *Mathematics of operations research*, 16(2):292–309. 81

[Rockafellar, 1970] Rockafellar, R. T. (1970). *Convex analysis*. Princeton Mathematical Series. Princeton University Press, Princeton, N. J. 52

[Rockafellar, 1976a] Rockafellar, R. T. (1976a). Augmented Lagrangians and Applications of the Proximal Point Algorithm in Convex Programming. *Mathematics of Operations Research*, 1(2):97–116. 13, 35, 36, 37, 38, 41, 48, 94

[Rockafellar, 1976b] Rockafellar, R. T. (1976b). Monotone operators and the proximal point algorithm. *J. Control Optim.*, page 877–898. 51, 52

[Ruiz, 2001] Ruiz, D. (2001). A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical report, Rutherford Appleton Laboratory. 61

[Ryu and Boyd, 2016] Ryu, E. K. and Boyd, S. (2016). Primer on monotone operator methods. *Appl. comput. math*, 15(1):3–43. 51

[Scianca et al., 2020] Scianca, N., De Simone, D., Lanari, L., and Oriolo, G. (2020). Mpc for humanoid gait generation: Stability and feasibility. *IEEE Transactions on Robotics*, 36(4):1171–1188. 70

[Scieur et al., 2022] Scieur, D., Gidel, G., Bertrand, Q., and Pedregosa, F. (2022). The curse of unrolling: Rate of differentiating through optimization. In *NeurIPS*. 83

[Shi et al., 2016] Shi, R., Liu, L., Long, T., and Liu, J. (2016). An efficient ensemble of radial basis functions method based on quadratic programming. *Engineering Optimization*, 48(7):1202–1225. 1, 7

[SHIM, 1983] SHIM, J. K. (1983). A survey of quadratic programming applications to business and economics. *International Journal of Systems Science*, 14(1):105–115. 2, 8

[Simon and Simon, 2006] Simon, D. and Simon, D. L. (2006). Kalman filtering with inequality constraints for turbofan engine health estimation. *IEE Proceedings-Control Theory and Applications*, 153(3):371–378. 2, 8

[Spielman and Teng, 2004] Spielman, D. A. and Teng, S.-H. (2004). Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)*, 51(3):385–463. 18

[Stellato et al., 2020] Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S. (2020). OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672. 13, 15, 30, 32, 60, 61, 62, 63, 64, 71, 72, 74, 76

[Suh et al., 2022] Suh, H. J., Simchowitz, M., Zhang, K., and Tedrake, R. (2022). Do differentiable simulators give better policy gradients? In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 20668–20696. PMLR. 87

[Suh et al., 2021] Suh, H. J. T., Pang, T., and Tedrake, R. (2021). Bundled gradients through contact via randomized smoothing. *CoRR*, abs/2109.05143. 121

[Sun and Qi, 1999] Sun, D. and Qi, L. (1999). On ncp-functions. *Comput. Optim. Appl.*, 13(1-3):201–220. 112

[Sun et al., 2022] Sun, H., Shi, Y., Wang, J., Tuan, H. D., Poor, H. V., and Tao, D. (2022). Alternating differentiation for optimization layers. *CoRR*, abs/2210.01802. 79, 85, 86, 104

[Sun, 1997] Sun, J. (1997). On piecewise quadratic Newton and trust region problems. *Mathematical Programming*, 76:451–467. 36, 39, 49

[Tassa et al., 2014] Tassa, Y., Mansard, N., and Todorov, E. (2014). Control-limited differential dynamic programming. In *2014 International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. 2, 8, 123

[tasts-robots, 2023] tasts-robots (2023). upkie_description. https://github.com/tasts-robots/upkie_description. 63, 70, 71

[Tibshirani, 1996] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288. 2, 8

[Van Loan and Golub, 1996] Van Loan, C. F. and Golub, G. (1996). Matrix computations (johns hopkins studies in mathematical sciences). *Matrix Computations*, 5. 32

[Vlastelica et al., 2019] Vlastelica, M., Paulus, A., Musil, V., Martius, G., and Rolínek, M. (2019). Differentiation of blackbox combinatorial solvers. *arXiv preprint arXiv:1912.02175*. 87

[Wang and Boyd, 2009] Wang, Y. and Boyd, S. (2009). Fast model predictive control using online optimization. *IEEE Transactions on control systems technology*, 18(2):267–278. 63, 66

[Wensing et al., 2022] Wensing, P. M., Posa, M., Hu, Y., Escande, A., Mansard, N., and Del Prete, A. (2022). Optimization-based control for dynamic legged robots. *arXiv preprint arXiv:2211.11644*. 2, 8, 123

[Wieber, 2006a] Wieber, P.-B. (2006a). Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *2006 International Conference on Humanoid Robots*, pages 137–142. IEEE. 2, 8, 70, 123

[Wieber, 2006b] Wieber, P.-b. (2006b). Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*, pages 137–142. 123

[Wiens, 1976] Wiens, T. B. (1976). Peasant risk aversion and allocative behavior: a quadratic programming experiment. *American Journal of Agricultural Economics*, 58(4_Part_1):629–635. 2, 8

[Wilder et al., 2019] Wilder, B., Dilkina, B., and Tambe, M. (2019). Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1658–1665. 79, 87

[Wright, 1997] Wright, S. J. (1997). Primal-dual interior-point methods. *SIAM*. 29

[Zhang et al., 2018] Zhang, C., Chen, H., Shi, K., Liang, Z., Mo, W., and Hua, D. (2018). A multi-time reactive power optimization under interval uncertainty of renewable power generation by an interval sequential quadratic programming method. *IEEE Transactions on Sustainable Energy*, 10(3):1086–1097. 2, 8

[Zhou et al., 2012] Zhou, J., Cheng, S., and Li, M. (2012). Sequential quadratic programming for robust optimization with interval uncertainty. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 45028, pages 1087–1100. American Society of Mechanical Engineers. 1, 7

RÉSUMÉ

Les applications modernes de problèmes de contrôle posent des défis computationnels et requièrent des calculs numériques rapides et robustes. La programmation quadratique joue un rôle central pour ces domaines, étant utilisée dans de nombreuses applications pratiques. Les avancées récentes de l'optimisation différentiable ont également montré que les couches de programmation quadratique offrent une puissance de modélisation riche et peuvent être efficaces pour résoudre certaines tâches d'apprentissage automatique de manière améliorée. Dans cette thèse, nous avons introduit un nouvel algorithme, PROXQP, pour résoudre des problèmes génériques de programmation quadratique (QP). Entre autres, nous proposons notamment de combiner la stratégie de globalisation de contraintes de borne Lagrangienne (BCL) pour calibrer automatiquement les paramètres clés d'un algorithme Lagrangien dual proximal augmenté. Nous avons également montré que PROXQP peut résoudre le QP faisable *le plus proche* si le QP original est infaisable. Ces améliorations théoriques sont accompagnées d'une implémentation logicielle efficace de PROXQP dans la bibliothèque open-source PROXSUITE. Nous avons évalué les performances de PROXQP sur diverses expériences standard de robotique et de contrôle, y compris une dans le contexte d'une application sur rorbot réel. En particulier, nous avons mis en évidence que la résolution des QP faisables *les plus proches* peut être efficacement exploitée dans le contexte de la commande prédictive. Enfin, à travers diverses évaluations issues de la littérature en optimisation, nous avons également démontré que PROXQP se situe au niveau des solveurs de pointe pour une large gamme de problèmes QP génériques. L'étude des QP faisables *les plus proches* nous a ensuite poussée à enrichir les capacités expressives des couches de QP existantes. En particulier, nous proposons une formulation de Jacobienne conservatrice étendue pour différencier les QPs convexes, couvrant à la fois les problèmes faisables et infaisables. Pour les problèmes faisables, et lorsque la solution est différentiable, cela se réduit aux Jacobiens standards. Nous illustrons notamment que l'entraînement de ces nouvelles types de couches de QPs conduit à une meilleure capacité de prédiction pour résoudre certaines tâches. Enfin, nous fournissons un bibliothèque C++ open-source, appelé QPLAYER, qui implémente efficacement l'approche.

MOTS CLÉS

programmation quadratique; optimisation différentiable; optimisation numérique; contrôle optimal; robotique.

ABSTRACT

Modern applications of control problems are computationally challenging and require fast and robust numerical calculations. Quadratic programming plays for these domains a pivotal role since it is used in numerous practical applications. Recent advances of differentiable optimization have also illustrated that Quadratic Programming layers offer a rich modeling power and can be effective for solving certain machine learning tasks. In this thesis, we have introduced a new algorithm, PROXQP, for solving generic QPs. Among others, we notably propose to combine the bound constraint Lagrangian (BCL) globalization strategy for automatically scheduling key parameters of a primal-dual proximal augmented Lagrangian algorithm. Additionally, we show that the PROXQP algorithm features a global convergence guarantee, as well as a few other advantageous numerical properties. Furthermore, we highlight that the PROXQP algorithm actually solves the *closest* primal-feasible QP if the original QP appears to be primally infeasible. These convergence guarantees come together with an efficient software implementation of PROXQP in the open-source PROXSUITE library. The performance of PROXQP is evaluated on different robotic and control experiments, including a real-world closed-loop controller application. In particular, we have highlighted how solving the *closest* feasible QPs can be efficiently leveraged in the context of closed-loop convex MPC. Finally, through various benchmarks of the optimization literature, we have also shown that PROXQP performs at the level of state-of-the-art solvers on a large set of generic QP problems. The study of the *closest* feasible QPs has then motivated us to enrich the expressive capabilities of existing QP layers. In particular, we propose an extended conservative Jacobian formulation for differentiating convex QPs, covering both feasible and infeasible problems. For feasible problems, and when the solution is differentiable, this reduces to standard Jacobians. We then show how driving towards feasibility at test time a new scope of QP layers using our techniques. We illustrate that training these new types of QP layers leads to better predicting power for solving some tasks. We further provide an open-source C++ framework, referred to as QPLAYER, which efficiently implements the approach.

KEYWORDS

quadratic programming; differentiable optimization; numerical optimization; optimal control; robotics.