Numerical optimization packages for optimal control

QPIPM and NLPSQP

Kaysfeld, Morten Wahlgreen; Jørgensen, John Bagterp

Publication date:
2023

Document Version
Publisher's PDF, also known as Version of record

Technical University of Denmark

Morten Wahlgreen Kaysfeld

Numerical optimization packages for optimal control: QPIPM and NLPSQP

Technical Report, July 25, 2023

DTU Compute
Department of Applied Mathematics and Computer Science

MORTEN WAHLGREEN KAYSFELD

Numerical optimization packages for optimal control: QPIPM and NLPSQP

Technical Report, July 25, 2023

Supervisors:

John Bagterp Jørgensen

DTU - Technical University of Denmark, Kgs. Lyngby - 2023

Numerical optimization packages for optimal control: QPIPM and NLPSQP

This report was prepared by:
Morten Wahlgreen Kaysfeld

Advisors:
John Bagterp Jørgensen

DTU Compute - Department of Applied Mathematics and Computer Science
Section for Scientific Computing
Technical University of Denmark
Matematiktorvet, Building 303B
2800 Kgs. Lyngby
Denmark

morwa@dtu.dk

Field: Numerical optimization, quadratic programming, nonlinear programming

Class: Report publicly available - Software is closed-source

Remarks: This technical report is prepared to document the numerical optimization software packages: QPIPM (quadratic-programming-interior-point-method) and NLPSQP (nonlinear-programming-sequential-quadratic-programming)

Table of Contents

Part I

QPIPM

CHAPTER 1

Introduction

In this part, we introduce the Riccati based primal-dual interior-point software, QPIPM (quadratic-programming-interior-point-method), for solution of quadratic programming problems (QPs). QPIPM can solve QPs with 1) equality constraints, 2) box constraints, and 3) soft constraints. We have implemented QPIPM in both a Matlab version and a C version. Due to time constraints, currently only the Matlab version of QPIPM supports QPs with soft constraints. The Matlab version provides a non-optimized and simple implementation that can be useful in a development phase. The C version is implemented thread-safe with the intention to solve multiple optimal control problems (OCPs) in parallel. The thread-safety is achieved by QPIPM having no internal memory allocations. The main purpose of QPIPM is to be included in the sequential quadratic programming algorithm, NLPSQP, introduced in the next part of this report and the integration of QPIPM and NLPSQP in a previously implemented toolbox for parallel Monte Carlo simulation of closed-loop systems (Wahlgreen et al. 2021). We also point out that the current version of QPIPM is work in progress and that the implementation can be optimized for better computational performance.

In this report, we introduce the mathematical details in the QPIPM implementation and introduce the interfaces of QPIPM in both Matlab and C. QPIPM is stored in a private gitlab-repository `QPIPM` and is part of the project `SCProject`, which is implemented in C and contains a number of other gitlab-repositories. For the C version, we introduce the other dependencies in `SCProject` and explain how to allocate the required memory prior to calling QPIPM.

We point out that the implementation of QPIPM is highly inspired by previous work on the topic (Rao et al. 1998, Jørgensen 2004, Wächter and Biegler 2006, Frison and Jørgensen 2013, Jørgensen et al. 2012, Wahlgreen and Jørgensen 2022).

CHAPTER 2

Mathematical details

We introduce the mathematical details of the QPIPM implementation. The mathematical details of the Matlab and C implementation are identical. However, the C version does not include the option to apply soft constraints in the current version. QPIPM is a primal-dual interior-point algorithm, which can both apply an LDL-factorization and a Riccati based method to solve the system of linear equations for the Newton search direction. The Riccati based method requires a structured QP, which, e.g., occurs in optimal control applications.

2.1 Primal-dual interior-point algorithm

In this section, we introduce the mathematical details of the primal-dual interior-point algorithm applied in QPIPM. The algorithm solves the first order Karush–Kuhn–Tucker (KKT) conditions with Newtons' method (Karush 1939, Kuhn and Tucker 1951, Kjeldsen 2000). As such, the algorithm is iterative and in each iteration, l, a system of linear equations is solved for the Newton search direction. We apply Mehrotra's predictor-corrector method, as such QPIPM computes both a predictor and corrector step with the same factorization of the search direction matrix (Mehrotra 1992).

We design QPIPM to solve QPs with bound constraints and general soft constraints. The general soft constraints include a lower and upper soft bound with slack variables, and the slack variables are penalized with both a linear and quadratic term in the objective. As such, the QP is in the form

$$\min_{x,\epsilon_l,\epsilon_u} \quad \frac{1}{2}x^\top Hx + g^\top x + \frac{1}{2}\epsilon_l^\top Q_l \epsilon_l + q_l^\top \epsilon_l + \frac{1}{2}\epsilon_u^\top Q_u \epsilon_u + q_u^\top \epsilon_u, \tag{2.1a}$$

$$s.t. \quad A^\top x = b, \tag{2.1b}$$

$$l \leq x \leq u, \tag{2.1c}$$

$$l_s - \epsilon_l \leq S^\top x \leq u_s + \epsilon_u, \tag{2.1d}$$

$$\epsilon_l, \epsilon_u \geq 0. \tag{2.1e}$$

$H \in \mathbb{R}^{n\times n}$, $g \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times m_e}$, $b \in \mathbb{R}^{m_e}$, $l \in \mathbb{R}^n$, $u \in \mathbb{R}^n$, $S \in \mathbb{R}^{n\times m_s}$, $l_s \in \mathbb{R}^{m_s}$, $u_s \in \mathbb{R}^{m_s}$, and $x \in \mathbb{R}^n$ are the decision variables. $\epsilon_l \in \mathcal{R}^{m_s}$ are lower soft bound slack variables and $\epsilon_u \in \mathcal{R}^{m_s}$ are upper soft bound slack variables. $Q_l \in \mathbb{R}^{m_s\times m_s}$ and $Q_u \in \mathbb{R}^{m_s\times m_s}$ are (assumed) diagonal penalty matrices, and $q_l \in \mathbb{R}^{m_s}$ and $q_u \in \mathbb{R}^{m_s}$ are penalty vectors. As such, n is the number of decision variables, m_e is the number of equality constraints, and m_s is the number of upper and lower soft constraints.

2.1.1 Search direction

QPIPM computes a search direction in each iteration. First, we consider the Lagrangian function, $\mathcal{L} = \mathcal{L}(x, \epsilon_l, \epsilon_u, y, v_l, v_u, z_{s_l}, z_{s_u}, v_{\epsilon_l}, v_{\epsilon_u})$, of (2.1), which is

$$\begin{aligned}\mathcal{L} = &\frac{1}{2}x^\top Hx + g^\top x + \frac{1}{2}\epsilon_l^\top Q_l \epsilon_l + q_l^\top \epsilon_l + \frac{1}{2}\epsilon_u^\top Q_u \epsilon + q_u^\top \epsilon_u \\ &- y^\top (A^\top x - b) - v_l^\top (x - l) - v_u^\top (u - x) - v_{\epsilon_l}^\top \epsilon_l - v_{\epsilon_u}^\top \epsilon_u \\ &- z_{s_l}^\top (S^\top x - l_s + \epsilon_l) - z_{s_u}^\top (-S^\top x + u_s + \epsilon_u).\end{aligned} \tag{2.2}$$

y are equality constraint (2.1b) Lagrange multipliers, v_l and v_u are bound constraint (2.1c) Lagrange multipliers, z_{s_l} and z_{s_u} are soft constraint (2.1d) Lagrange multipliers, and v_{ϵ_l} and v_{ϵ_u} are ϵ-bound constraint (2.1e) Lagrange multipliers. As such, we write up the corresponding first order KKT-conditions,

$$\nabla_x \mathcal{L} = Hx + g - Ay - v_l + v_u - Sz_{s_l} + Sz_{s_u} = 0, \tag{2.3a}$$
$$\nabla_{\epsilon_l} \mathcal{L} = Q_l \epsilon_l + q_l - z_{s_l} - v_{\epsilon_l} = 0, \tag{2.3b}$$
$$\nabla_{\epsilon_u} \mathcal{L} = Q_u \epsilon_u + q_u - z_{s_u} - v_{\epsilon_u} = 0, \tag{2.3c}$$
$$b - A^\top x = 0, \tag{2.3d}$$
$$t_l + l - x = 0, \qquad t_u + x - u = 0, \tag{2.3e}$$
$$t_{\epsilon_l} - \epsilon_l = 0, \qquad t_{\epsilon_u} - \epsilon_u = 0, \tag{2.3f}$$
$$s_{s_l} - S^\top x + l_s - \epsilon_l = 0, \qquad s_{s_u} + S^\top x - u_s - \epsilon_u = 0, \tag{2.3g}$$
$$t_{l,i} v_{l,i} = 0, \qquad t_{u,i} v_{u,i} = 0, \tag{2.3h}$$
$$t_{\epsilon_l,i} v_{\epsilon_l,i} = 0, \qquad t_{\epsilon_u,i} v_{\epsilon_u,i} = 0, \tag{2.3i}$$
$$s_{s_l,i} z_{s_l,i} = 0, \qquad s_{s_u,i} z_{s_u,i} = 0, \tag{2.3j}$$
$$(v_l, v_u, z_{s_l}, z_{s_u}, v_{\epsilon_l}, v_{\epsilon_u}) \geq 0, \qquad (t_l, t_u, s_{s_l}, s_{s_u}, t_{\epsilon_l}, t_{\epsilon_u}) \geq 0, \tag{2.3k}$$

where t_l and t_u are bound constraint (2.1c) slack variables, s_{s_l} and s_{s_u} are soft constraint (2.1d) slack variables, and t_{ϵ_l} and t_{ϵ_u} are ϵ-bound constraint (2.1e) slack variables. The slack variables are defined as

$$s_{s_l} = S^\top x - l_s + \epsilon_l, \qquad s_{s_u} = -S^\top x + u_s + \epsilon_u, \tag{2.4a}$$
$$t_l = x - l, \qquad t_u = u - x, \tag{2.4b}$$
$$t_{\epsilon_l} = \epsilon_l, \qquad t_{\epsilon_u} = \epsilon_u. \tag{2.4c}$$

We write the KKT-conditions, (2.3), as a system of nonlinear equations in the form

$$\begin{bmatrix} r_L \\ r_{\epsilon_l} \\ r_{\epsilon_u} \\ r_A \\ r_{S_l} \\ r_{S_u} \\ r_{B_l} \\ r_{B_u} \\ r_{B_{\epsilon_l}} \\ r_{B_{\epsilon_u}} \\ r_{SZ_{s_l}} \\ r_{SZ_{s_u}} \\ r_{TV_l} \\ r_{TV_u} \\ r_{TV_{\epsilon_l}} \\ r_{TV_{\epsilon_u}} \end{bmatrix} = \begin{bmatrix} Hx + g - Ay - v_l + v_u - Sz_{s_l} + Sz_{s_u} \\ Q_l\epsilon_l + q_l - z_{s_l} - v_{\epsilon_l} \\ Q_u\epsilon_u + q_u - z_{s_u} - v_{\epsilon_u} \\ b - A^\top x \\ s_{s_l} - S^\top x + l_s - \epsilon_l \\ s_{s_u} + S^\top x - u_s - \epsilon_u \\ t_l + l - x \\ t_u + x - u \\ t_{\epsilon_l} - \epsilon_l \\ t_{\epsilon_u} - \epsilon_u \\ S_{s_l} Z_{s_l} e \\ S_{s_u} Z_{s_u} e \\ T_l V_l e \\ T_u V_u e \\ T_{\epsilon_l} V_{\epsilon_l} e \\ T_{\epsilon_u} V_{\epsilon_u} e \end{bmatrix} = 0, \tag{2.5a}$$

$$(v_l, v_u, v_{\epsilon_l}, v_{\epsilon_u}, z_{s_l}, z_{z_u}, t_l, t_u, t_{\epsilon_l}, t_{\epsilon_u}, s_{s_l}, s_{s_u}) \geq 0. \tag{2.5b}$$

$V_l = \text{diag}(v_l)$, $V_u = \text{diag}(v_u)$, $V_{\epsilon_l} = \text{diag}(v_{\epsilon_l})$, $V_{\epsilon_u} = \text{diag}(v_{\epsilon_u})$, $Z_{s_l} = \text{diag}(z_{s_l})$, $Z_{s_u} = \text{diag}(z_{s_u})$, $T_l = \text{diag}(t_l)$, $T_u = \text{diag}(t_u)$, $T_{\epsilon_l} = \text{diag}(t_{\epsilon_l})$, $T_{\epsilon_u} = \text{diag}(t_{\epsilon_u})$, $S_{s_l} = \text{diag}(s_l)$, $S_{s_u} = \text{diag}(s_u)$, and e is a vector of ones of proper dimension. We apply Newtons' method to solve the nonlinear system of equations, (2.5), which results in the following linear system of equations for the Newton search direction,

$$\left[\begin{array}{ccc|c|cccccc|cccccc} H & 0 & 0 & -A & -S & S & -I & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & Q_l & 0 & 0 & -I & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & Q_u & 0 & 0 & -I & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline -A^\top & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline -S^\top & -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \\ S^\top & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 \\ -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 \\ I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 \\ 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I \\ \hline 0 & 0 & 0 & 0 & S_{s_l} & 0 & 0 & 0 & 0 & 0 & Z_{s_l} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & S_{s_u} & 0 & 0 & 0 & 0 & 0 & Z_{s_u} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & T_l & 0 & 0 & 0 & 0 & 0 & V_l & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & T_u & 0 & 0 & 0 & 0 & 0 & V_u & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & T_{\epsilon_l} & 0 & 0 & 0 & 0 & 0 & V_{\epsilon_l} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & T_{\epsilon_u} & 0 & 0 & 0 & 0 & 0 & V_{\epsilon_u} \end{array}\right] \begin{bmatrix} \Delta x \\ \Delta\epsilon_l \\ \Delta\epsilon_u \\ \hline \Delta y \\ \hline \Delta z_{s_l} \\ \Delta z_{s_u} \\ \Delta v_l \\ \Delta v_u \\ \Delta v_{\epsilon_l} \\ \Delta v_{\epsilon_u} \\ \hline \Delta s_{s_l} \\ \Delta s_{s_u} \\ \Delta t_l \\ \Delta t_u \\ \Delta t_{\epsilon_l} \\ \Delta t_{\epsilon_u} \end{bmatrix} = - \begin{bmatrix} r_L \\ r_{\epsilon_l} \\ r_{\epsilon_u} \\ \hline r_A \\ \hline r_{S_l} \\ r_{S_u} \\ r_{B_l} \\ r_{B_u} \\ r_{B_{\epsilon_l}} \\ r_{B_{\epsilon_u}} \\ \hline r_{SZ_{s_l}} \\ r_{SZ_{s_u}} \\ r_{TV_l} \\ r_{TV_u} \\ r_{TV_{\epsilon_l}} \\ r_{TV_{\epsilon_u}} \end{bmatrix}. \tag{2.6}$$

The solution,

$$(\Delta x, \Delta\epsilon_l, \Delta\epsilon_u, \Delta y, \Delta z_{s_l}, \Delta z_{s_u}, \Delta v_l, \Delta v_u, \Delta v_{\epsilon_l}, \Delta v_{\epsilon_u}, \Delta s_{s_l}, \Delta s_{s_u}, \Delta t_l, \Delta t_u, \Delta t_{\epsilon_l}, \Delta t_{\epsilon_u}), \tag{2.7}$$

is the search direction applied in QPIPM. We point out that the system of equations, (2.6), can be compactly written as

$$\begin{bmatrix} \hat{H} & -\hat{A} & -\hat{C} & 0 \\ -\hat{A}^\top & 0 & 0 & 0 \\ -\hat{C}^\top & 0 & 0 & I \\ 0 & 0 & \hat{S} & \hat{Z} \end{bmatrix} \begin{bmatrix} \Delta\hat{x} \\ \Delta\hat{y} \\ \Delta\hat{z} \\ \Delta\hat{s} \end{bmatrix} = - \begin{bmatrix} \hat{r}_L \\ \hat{r}_A \\ \hat{r}_C \\ \hat{r}_{SZ} \end{bmatrix}, \tag{2.8}$$

where

$$\hat{x} = \begin{bmatrix} x \\ \epsilon_l \\ \epsilon_u \end{bmatrix}, \quad \hat{y} = y, \quad \hat{z} = \begin{bmatrix} z_{s_l} \\ z_{s_u} \\ v_l \\ v_u \\ v_{\epsilon_l} \\ v_{\epsilon_u} \end{bmatrix}, \quad \hat{s} = \begin{bmatrix} s_{s_l} \\ s_{s_u} \\ t_l \\ t_u \\ t_{\epsilon_l} \\ t_{\epsilon_u} \end{bmatrix}, \tag{2.9a}$$

$$\hat{r}_L = \begin{bmatrix} r_L \\ r_{\epsilon_l} \\ r_{\epsilon_u} \end{bmatrix}, \quad \hat{r}_A = r_A, \quad \hat{r}_C = \begin{bmatrix} r_{S_l} \\ r_{S_u} \\ r_{B_l} \\ r_{B_u} \\ r_{B_{\epsilon_l}} \\ r_{B_{\epsilon_u}} \end{bmatrix}, \quad \hat{r}_{SZ} = \begin{bmatrix} r_{SZ_{s_l}} \\ r_{SZ_{s_u}} \\ r_{TV_l} \\ r_{TV_u} \\ r_{TV_{\epsilon_l}} \\ r_{TV_{\epsilon_u}} \end{bmatrix}, \tag{2.9b}$$

$$\hat{H} = \begin{bmatrix} H & & \\ & Q_l & \\ & & Q_u \end{bmatrix}, \quad \hat{A} = \begin{bmatrix} A \\ 0 \\ 0 \end{bmatrix}, \quad \hat{C} = \begin{bmatrix} S & -S & I & -I & 0 & 0 \\ I & 0 & 0 & 0 & I & 0 \\ 0 & I & 0 & 0 & 0 & I \end{bmatrix}, \tag{2.9c}$$

$$\hat{Z} = \begin{bmatrix} Z_{s_l} & & & & & & \\ & Z_{s_u} & & & & & \\ & & V_l & & & & \\ & & & V_u & & & \\ & & & & V_{\epsilon_l} & & \\ & & & & & V_{\epsilon_u} \end{bmatrix}, \quad \hat{S} = \begin{bmatrix} S_{s_l} & & & & & & \\ & S_{s_u} & & & & & \\ & & T_l & & & & \\ & & & T_u & & & \\ & & & & T_{\epsilon_l} & & \\ & & & & & T_{\epsilon_u} \end{bmatrix}. \tag{2.9d}$$

However, we exploit the structure of the matrices in (2.9), and elimination of Lagrange multipliers and slack variables, to reduce the size of the system (2.6) in the following section.

2.1.2 System reduction

The linear system (2.6) can be reduced in size by elimination of the inequality Lagrange multipliers and corresponding slack variables (i.e., for the lower and upper bound constraint, the soft constraints, and the ϵ-bound constraints). We define six diagonal matrices from the Lagrange multipliers and corresponding slack variables,

$$D_{s_l} = \text{diag}(z_{s_l}/s_{s_l}), \qquad D_{s_u} = \text{diag}(z_{s_u}/s_{s_u}), \tag{2.10a}$$

$$D_l = \text{diag}(v_l/t_l), \qquad D_u = \text{diag}(v_u/t_u), \tag{2.10b}$$

$$D_{\epsilon_l} = \text{diag}(v_{\epsilon_l}/t_{\epsilon_l}), \qquad D_{\epsilon_u} = \text{diag}(v_{\epsilon_u}/t_{\epsilon_u}). \tag{2.10c}$$

By elimination of the six Lagrange multipliers and slack variables, we arrive at the following reduced system

$$\begin{bmatrix} \bar{H} & E & F & -A \\ E^\top & \bar{Q}_l & & \\ F^\top & & \bar{Q}_u & \\ -A^\top & & & \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \epsilon_l \\ \Delta \epsilon_u \\ \Delta y \end{bmatrix} = \begin{bmatrix} \bar{r}_L \\ \bar{r}_{\epsilon_l} \\ \bar{r}_{\epsilon_u} \\ \bar{r}_A \end{bmatrix} \tag{2.11}$$

where

$$\bar{H} = H + D_l + D_u + ES^\top - FS^\top, \tag{2.12a}$$

$$E = SD_{s_l}, \qquad F = -SD_{s_u}, \tag{2.12b}$$

$$\bar{Q}_l = Q_l + D_{\epsilon_l} + D_{s_l}, \qquad \bar{Q}_u = Q_u + D_{\epsilon_u} + D_{s_u}, \tag{2.12c}$$

and

$$\begin{aligned} \bar{r}_L = & -r_L + S(S_{s_l}^{-1} Z_{s_l}(r_{S_l} - Z_{s_l}^{-1} r_{SZ_{s_l}})) - S(S_{s_u}^{-1} Z_{s_u}(r_{S_u} - Z_{s_u}^{-1} r_{SZ_{s_u}})) \\ & + T_l^{-1} V_l (r_{B_l} - V_l^{-1} r_{TV_l}) - T_u^{-1} V_u (r_{B_u} - V_u^{-1} r_{TV_u}), \end{aligned} \tag{2.13a}$$

$$\bar{r}_{\epsilon_l} = -r_{\epsilon_l} + T_{\epsilon_l}^{-1} V_{\epsilon_l} (r_{B_{\epsilon_l}} - V_{\epsilon_l}^{-1} r_{TV_{\epsilon_l}}) + S_{s_l}^{-1} Z_{s_l} (r_{S_l} - Z_{s_l}^{-1} r_{SZ_{s_l}}), \tag{2.13b}$$

$$\bar{r}_{\epsilon_u} = -r_{\epsilon_u} + T_{\epsilon_u}^{-1} V_{\epsilon_u} (r_{B_{\epsilon_u}} - V_{\epsilon_u}^{-1} r_{TV_{\epsilon_u}}) + S_{s_u}^{-1} Z_{s_u} (r_{S_u} - Z_{s_u}^{-1} r_{SZ_{s_u}}), \tag{2.13c}$$

$$\bar{r}_A = -r_A. \tag{2.13d}$$

The eliminated Lagrange multipliers and slack variables are

$$\Delta v_l = T_l^{-1} V_l (r_{B_l} - V_l^{-1} r_{TV_l}) - T_l^{-1} V_l \Delta x, \tag{2.14a}$$

$$\Delta v_u = T_u^{-1} V_u (r_{B_u} - V_u^{-1} r_{TV_u}) + T_u^{-1} V_u \Delta x, \tag{2.14b}$$

$$\Delta v_{\epsilon_l} = T_{\epsilon_l}^{-1} V_{\epsilon_l} (r_{B_{\epsilon_l}} - V_{\epsilon_l}^{-1} r_{TV_{\epsilon_l}}) - T_{\epsilon_l}^{-1} V_{\epsilon_l} \Delta \epsilon_l, \tag{2.14c}$$

$$\Delta v_{\epsilon_u} = T_{\epsilon_u}^{-1} V_{\epsilon_u} (r_{B_{\epsilon_u}} - V_{\epsilon_u}^{-1} r_{TV_{\epsilon_u}}) - T_{\epsilon_u}^{-1} V_{\epsilon_u} \Delta \epsilon_u, \tag{2.14d}$$

$$\Delta z_{s_l} = S_{s_l}^{-1} Z_{s_l} (r_{S_l} - Z_{s_l}^{-1} r_{SZ_{s_l}}) - S_{s_l}^{-1} Z_{s_l} (S^\top \Delta x + \Delta \epsilon_l), \tag{2.14e}$$

$$\Delta z_{s_u} = S_{s_u}^{-1} Z_{s_u} (r_{S_u} - Z_{s_u}^{-1} r_{SZ_{s_u}}) - S_{s_u}^{-1} Z_{s_u} (-S^\top \Delta x + \Delta \epsilon_u), \tag{2.14f}$$

$$\Delta t_l = -V_l^{-1} r_{TV_l} - V_l^{-1} T_l \Delta v_l, \tag{2.14g}$$

$$\Delta t_u = -V_u^{-1} r_{TV_u} - V_u^{-1} T_u \Delta v_u, \tag{2.14h}$$

$$\Delta t_{\epsilon_l} = -V_{\epsilon_l}^{-1} r_{TV_{\epsilon_l}} - V_{\epsilon_l}^{-1} T_{\epsilon_l} \Delta v_{\epsilon_l}, \tag{2.14i}$$

$$\Delta t_{\epsilon_u} = -V_{\epsilon_u}^{-1} r_{TV_{\epsilon_u}} - V_{\epsilon_u}^{-1} T_{\epsilon_u} \Delta v_{\epsilon_u}, \tag{2.14j}$$

$$\Delta s_{s_l} = -Z_{s_l}^{-1} r_{SZ_{s_l}} - Z_{s_l}^{-1} S_{s_l} \Delta z_{s_l}, \tag{2.14k}$$

$$\Delta s_{s_u} = -Z_{s_u}^{-1} r_{SZ_{s_u}} - Z_{s_u}^{-1} S_{s_u} \Delta z_{s_u}. \tag{2.14l}$$

In addition, we eliminate the soft constraint slack variables, ϵ_l and ϵ_u, from the system (2.11) to further reduce the size. The resulting system of linear equations is

$$\begin{bmatrix} \tilde{H} & -A \\ -A^\top & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \tilde{r}_L \\ \tilde{r}_A \end{bmatrix}, \tag{2.15}$$

where

$$\tilde{H} = \bar{H} - E\bar{Q}_l^{-1} E^\top - F\bar{Q}_u^{-1} F^\top, \tag{2.16a}$$

$$\tilde{r}_L = \bar{r}_L - E\bar{Q}_l^{-1} \bar{r}_{\epsilon_l} - F\bar{Q}_u^{-1} \bar{r}_{\epsilon_u}, \tag{2.16b}$$

$$\tilde{r}_A = \bar{r}_A. \tag{2.16c}$$

The eliminated slack variables are given as

$$\Delta\epsilon_l = \bar{Q}_l^{-1}(\bar{r}_{\epsilon_l} - E^\top \Delta x), \tag{2.17a}$$

$$\Delta\epsilon_u = \bar{Q}_u^{-1}(\bar{r}_{\epsilon_u} - F^\top \Delta x). \tag{2.17b}$$

The search direction (2.7) can be obtained by solution of the system of linear equations, (2.15), to obtain $(\Delta x, \Delta y)$ and computing first the soft constraint slack variables from (2.17) and finally the remaining Lagrange multipliers and slack variables from (2.14). QPIPM solves (2.15) with an LDL-factorization and back substitution.

Applying the compact notation in (2.9a), we define the QPIPM step as

$$(\hat{x}, \hat{y}, \hat{z}, \hat{s}) = (\hat{x}, \hat{y}, \hat{z}, \hat{s}) + \eta\alpha(\Delta\hat{x}, \Delta\hat{y}, \Delta\hat{z}, \Delta\hat{s}), \tag{2.18}$$

where $\eta = 0.995$ and the step-size, α, ensures $(\hat{z}, \hat{s}) \geq 0$.

2.1.3 Fraction-to-the-boundary

QPIPM applies a fraction-to-the-boundary rule to avoid the QPIPM step zeroing the Lagrange multipliers or slack variables (Wahlgreen and Jørgensen 2022). The rule is

$$\begin{bmatrix} \hat{z} \\ \hat{s} \end{bmatrix} + \alpha \begin{bmatrix} \Delta\hat{z} \\ \Delta\hat{s} \end{bmatrix} \geq \kappa \begin{bmatrix} \hat{z} \\ \hat{s} \end{bmatrix}, \tag{2.19}$$

where $0 \leq \kappa \ll 1$ and $\kappa \to 0$ as the iteration number of QPIPM, l, increases. The rule (2.19) implements a proportional step-back from the zero-boundary. In the predictor phase, QPIPM uses $\kappa = 0$ to compute α^{aff}, and in the corrector phase QPIPM uses $\kappa = \min(1 - \eta, \mu^{aff})$ to compute α. The rule (2.19) is similar to the rule applied in IPOPT (Wächter and Biegler 2006).

2.1.4 Predictor-corrector algorithm

QPIPM applies Mehrotra's predictor-corrector algorithm (Mehrotra 1992), i.e., QPIPM applies the factorization of (2.15) twice: 1) in the predictor step and 2) in the corrector step. In the predictor phase, we solve

$$\begin{bmatrix} \tilde{H} & -A \\ -A^\top & 0 \end{bmatrix} \begin{bmatrix} \Delta x^{aff} \\ \Delta y^{aff} \end{bmatrix} = \begin{bmatrix} \tilde{r}_L \\ \tilde{r}_A \end{bmatrix}, \tag{2.20}$$

and compute the remaining part of the affine search direction from (2.17) and (2.14). From the affine search direction, we compute the duality gap, μ, and the centering parameter, σ as

$$\mu^{aff} = \frac{(\hat{z} + \alpha^{aff}\Delta\hat{z}^{aff})^\top(\hat{s} + \alpha^{aff}\Delta\hat{s}^{aff})}{\bar{m}}, \qquad \mu = \frac{\hat{s}^\top\hat{z}}{\bar{m}}, \qquad \sigma = \left(\frac{\mu^{aff}}{\mu}\right)^3, \tag{2.21}$$

where we apply the notation in (2.9a) for simplicity and $\bar{m}$ is the total number of inequality constraints (bound constraints, soft constraints, and ϵ-bound constraints). In the corrector step, we adapt the right hand side of (2.15) and consider the system

$$\begin{bmatrix} \tilde{H} & -A \\ -A^\top & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \tilde{\tilde{r}}_L \\ \tilde{r}_A \end{bmatrix}, \tag{2.22}$$

where $\tilde{\tilde{r}}_L$ is computed according to (2.13a) and (2.16b), with the terms, r_{TV_l}, r_{TV_u}, $r_{TV_{\epsilon_l}}$, $r_{TV_{\epsilon_u}}$, $r_{SZ_{s_l}}$, and $r_{SZ_{s_u}}$ being defined as

$$r_{TV_l} \leftarrow r_{TV_l} + \Delta T_l^{aff} \Delta V_l^{aff} - \sigma\mu e, \qquad r_{TV_u} \leftarrow r_{TV_u} + \Delta T_u^{aff} \Delta V_u^{aff} - \sigma\mu e, \tag{2.23a}$$

$$r_{TV_{\epsilon_l}} \leftarrow r_{TV_{\epsilon_l}} + \Delta T_{\epsilon_l}^{aff} \Delta V_{\epsilon_l}^{aff} - \sigma\mu e, \qquad r_{TV_{\epsilon_u}} \leftarrow r_{TV_{\epsilon_u}} + \Delta T_{\epsilon_u}^{aff} \Delta V_{\epsilon_u}^{aff} - \sigma\mu e, \tag{2.23b}$$

$$r_{SZ_{s_l}} \leftarrow r_{SZ_{s_l}} + \Delta S_{s_l}^{aff} \Delta Z_{s_l}^{aff} - \sigma\mu e, \qquad r_{SZ_{s_u}} \leftarrow r_{SZ_{s_u}} + \Delta S_{s_u}^{aff} \Delta Z_{s_u}^{aff} - \sigma\mu e. \tag{2.23c}$$

Then the QPIPM search direction is the solution to (2.22) with the remaining part being computed from (2.17) and (2.14).

We point out that system matrix in the predictor and corrector phase is identical. Therefore, QPIPM reuses the factorization from the predictor phase in the corrector phase.

2.1.5 Convergence criterion

QPIPM converges once the KKT-conditions (2.3) are satisfied. In practice, we consider a scaled violation, ξ, and define convergence as $\xi < \epsilon$, where $\epsilon > 0$ is a user-selected convergence tolerance. The scaled violation is

$$\begin{aligned}\xi = \max\Big(&s_H||r_L, r_{\epsilon_l}, r_{\epsilon_u}||_\infty, s_A||r_A||_\infty, s_S||r_{S_l}, r_{S_u}||_\infty, s_B||r_{B_l}, r_{B_u}||_\infty, ||r_{B_{\epsilon_l}}, r_{B_{\epsilon_l}}||_\infty, \\ &||r_{SZ_{s_l}}, r_{SZ_{s_u}}, r_{TV_l}, r_{TV_u}, r_{TV_{\epsilon_l}}, r_{TV_{\epsilon_u}}||_\infty\Big),\end{aligned} \tag{2.24}$$

where

$$s_H = \max(1, ||H||_\infty, ||\, g||_\infty, ||A\,||_\infty, ||Q_l||_\infty, ||Q_u||_\infty, ||q_l, q_u||_\infty, ||S_l||_\infty, ||S_u||_\infty)^{-1}, \tag{2.25a}$$

$$s_A = \max(1, ||A^\top||_\infty, ||b||_\infty)^{-1}, \tag{2.25b}$$

$$s_S = \max(1, ||S_l^\top||_\infty, ||S_u^\top||_\infty, ||l_s||_\infty, ||u_s||_\infty)^{-1}, \tag{2.25c}$$

$$s_B = \max(1, ||l||_\infty, ||u||_\infty)^{-1}. \tag{2.25d}$$

QPIPM computes ξ after taking the step (2.18) in the end of the corrector phase.

2.1.6 Infinity bound constraints

QPIPM eliminates all infinity bounds, i.e., bounds set to $-\infty$ or ∞, before starting the loop. As such, columns of S are not accessed if both l_s and u_s are infinity.

2.1.7 Algorithm

Algorithm 1 presents a detailed implementation guide for QPIPM.

Algorithm 1: QPIPM pseudo code

Input: Initial guess, x_0, and soft constrained QP,

$$\begin{aligned}\min_{x,\epsilon_l,\epsilon_u} \quad & \frac{1}{2}x^\top Hx + g^\top x + \frac{1}{2}\epsilon_l^\top Q_l \epsilon_l + q_l^\top \epsilon_l + \frac{1}{2}\epsilon_u^\top Q_u \epsilon + q_u^\top \epsilon_u,\\ s.t. \quad & A^\top x = b,\\ & l \le x \le u,\\ & l_s - \epsilon_l \le S^\top x \le u_s + \epsilon_u,\\ & \epsilon_l, \epsilon_u \ge 0,\end{aligned}$$

i.e. the matrices and vectors: H, Q_l, Q_u, g, g_l, g_u, A, b, l, u, S, l_s, and l_u.

- Initialize:

$$x = x_0, \qquad \epsilon_l = \epsilon_u = 0, \qquad y = 0, \qquad \hat{z} = 1, \qquad \hat{s} = 1.$$

- Compute scaling factors,

$$\begin{aligned}&\tilde{r}_L = \max(1, ||H, Q_l, Q_u||_\infty, ||g, g_l, g_u||_\infty, ||A||_\infty, ||S||_\infty)^{-1}, \quad \tilde{r}_A = \max(1, ||A^\top||_\infty, ||b||_\infty)^{-1},\\ &\tilde{r}_B = \max(1, ||l||_\infty, ||u||_\infty)^{-1}, \qquad \tilde{r}_S = \max(1, ||S^\top||_\infty, ||l_s||_\infty, ||l_u||_\infty)^{-1}\end{aligned}$$

- Compute scaled KKT-violation, ξ,

$$\begin{aligned}\xi = \max(&\tilde{r}_L||r_L, r_{\epsilon_l}, r_{\epsilon_u}||_\infty, \tilde{r}_A||r_A||_\infty, \tilde{r}_S||r_{S_l}, r_{S_u}||_\infty, \tilde{r}_B||r_{B_l}, r_{B_u}||_\infty, ||r_{\epsilon_l}, r_{\epsilon_u}||_\infty,\\ &||r_{SZ_{s_l}}, r_{SZ_{s_u}}, r_{TV_l}, r_{TV_u}, r_{TV_{\epsilon_l}}, r_{TV_{\epsilon_u}}||_\infty)\end{aligned}$$

while $\xi > \epsilon$ **do**

1. **Predictor phase:**

   i. Setup augmented system,

$$\underbrace{\begin{bmatrix}\tilde{H} & -A\\ -A^\top & 0\end{bmatrix}}_{M}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = \begin{bmatrix}\tilde{r}_L\\ \tilde{r}_A\end{bmatrix} \tag{2.26}$$

   ii. LDL factorize: $[L, D] = ldl(M)$

   iii. Solve the system (2.26) to get the affine direction, $\Delta x = \Delta x^{aff}$ and $\Delta y = \Delta y^{aff}$

   iv. Compute $\Delta\epsilon_l$ and $\Delta\epsilon_u$,

$$\Delta\epsilon_l = \bar{Q}_l^{-1}(\bar{r}_{\epsilon_l} - E^\top \Delta x), \qquad \Delta\epsilon_u = \bar{Q}_u^{-1}(\bar{r}_{\epsilon_u} - F^\top \Delta x)$$

   v. Compute Δz_{s_l}, Δz_{s_u}, Δv_l, Δv_u, $\Delta\epsilon_l$, $\Delta\epsilon_u$, Δs_{s_l}, Δs_{s_u}, Δt_l, Δt_u, Δt_{ϵ_l}, and Δt_{ϵ_u},

$$\begin{aligned}
\Delta z_{s_l} &= S_{s_l}^{-1} Z_{s_l}(r_{S_l} - Z_{s_l}^{-1} r_{SZ_{s_l}}) - S_{s_l}^{-1} Z_{s_l}(S^\top \Delta x + \Delta\epsilon_l), & \Delta s_{s_l} &= -Z_{s_l}^{-1} r_{SZ_{s_l}} - Z_{s_l}^{-1} S_{s_l} \Delta z_{s_l},\\
\Delta z_{s_u} &= S_{s_u}^{-1} Z_{s_u}(r_{S_u} - Z_{s_u}^{-1} r_{SZ_{s_u}}) - S_{s_u}^{-1} Z_{s_u}(-S^\top \Delta x + \Delta\epsilon_u), & \Delta s_{s_u} &= -Z_{s_u}^{-1} r_{SZ_{s_u}} - Z_{s_u}^{-1} S_{s_u} \Delta z_{s_u},\\
\Delta v_l &= T_l^{-1} V_l(r_{B_l} - V_l^{-1} r_{TV_l}) - T_l^{-1} V_l \Delta x, & \Delta t_l &= -V_l^{-1} r_{TV_l} - V_l^{-1} T_l \Delta v_l,\\
\Delta v_u &= T_u^{-1} V_u(r_{B_u} - V_u^{-1} r_{TV_u}) + T_l^{-1} V_l \Delta x, & \Delta t_u &= -V_u^{-1} r_{TV_u} - V_u^{-1} T_u \Delta v_u,\\
\Delta v_{\epsilon_l} &= T_{\epsilon_l}^{-1} V_{\epsilon_l}(r_{B_{\epsilon_l}} - V_{\epsilon_l}^{-1} r_{TV_{\epsilon_l}}) - T_{\epsilon_l}^{-1} V_{\epsilon_l} \Delta\epsilon_l, & \Delta t_{\epsilon_l} &= -V_{\epsilon_l}^{-1} r_{TV_{\epsilon_l}} - V_{\epsilon_l}^{-1} T_{\epsilon_l} \Delta v_{\epsilon_l},\\
\Delta v_{\epsilon_u} &= T_{\epsilon_u}^{-1} V_{\epsilon_u}(r_{B_{\epsilon_u}} - V_{\epsilon_u}^{-1} r_{TV_{\epsilon_u}}) - T_{\epsilon_u}^{-1} V_{\epsilon_u} \Delta\epsilon_u, & \Delta t_{\epsilon_u} &= -V_{\epsilon_u}^{-1} r_{TV_{\epsilon_u}} - V_{\epsilon_u}^{-1} T_{\epsilon_u} \Delta v_{\epsilon_u},
\end{aligned}$$

   vi. Find α^{aff} such that $(\hat{z}, \hat{s}) + \alpha^{aff}\Delta(\hat{z}, \hat{s}) \ge 0$, where $\hat{z} = (z_{s_l}, z_{s_u}, v_l, v_u, v_{\epsilon_l}, v_{\epsilon_u})$ and $\hat{s} = (s_{s_l}, s_{s_u}, t_l, t_u, t_{\epsilon_l}, t_{\epsilon_u})$

   vii. Compute the duality gap, μ, and the centering parameter, σ (with $\bar{m}$ being the total number of inequality constraints including soft constraints)

$$\mu^{aff} = \frac{(\hat{z} + \alpha^{aff}\Delta\hat{z})^\top(\hat{s} + \alpha^{aff}\Delta\hat{s})}{\bar{m}} \qquad \mu = \frac{\hat{s}^\top \hat{z}}{\bar{m}}, \qquad \sigma = \left(\frac{\mu^{aff}}{\mu}\right)^3$$

2. **Corrector phase:**

   i. Recompute $\tilde{r}_L$ with the following definitions

$$\begin{aligned}
r_{SZ_{s_l}} &\leftarrow r_{SZ_{s_l}} + \Delta S_{s_l}^{aff}\Delta Z_{s_l}^{aff} - \sigma\mu e, & r_{SZ_{s_u}} &\leftarrow r_{SZ_{s_u}} + \Delta S_{s_u}^{aff}\Delta Z_{s_u}^{aff} - \sigma\mu e,\\
r_{TV_l} &\leftarrow r_{TV_l} + \Delta T_l^{aff}\Delta V_l^{aff} - \sigma\mu e, & r_{TV_u} &\leftarrow r_{TV_u} + \Delta T_u^{aff}\Delta V_u^{aff} - \sigma\mu e,\\
r_{TV_{\epsilon_l}} &\leftarrow r_{TV_{\epsilon_l}} + \Delta T_{\epsilon_l}^{aff}\Delta V_{\epsilon_l}^{aff} - \sigma\mu e, & r_{TV_{\epsilon_u}} &\leftarrow r_{TV_{\epsilon_u}} + \Delta T_{\epsilon_u}^{aff}\Delta V_{\epsilon_u}^{aff} - \sigma\mu e.
\end{aligned}$$

   ii. Repeat step 1ii-1v from the predictor phase (reapply LDL factorization from predictor phase)

   iii. Compute the step size, α, such that $(\hat{z}, \hat{s}) + \alpha^{aff}\Delta(\hat{z}, \hat{s}) \ge \kappa(\hat{z}, \hat{s})$, for $\kappa = \min(1 - \eta, \mu^{aff})$

3. Take step: $\chi = \hat{\chi} + \eta\alpha\Delta\hat{\chi}$, where $\chi = (x, \epsilon_l, \epsilon_u, y, z_{s_l}, z_{s_u}, v_l, v_u, v_{\epsilon_l}, v_{\epsilon_u}, s_{s_l}, s_{s_u}, t_l, t_u, t_{\epsilon_l}, t_{\epsilon_u})$ and $\eta = 0.995$

4. Compute scaled KKT-violation,

$$\begin{aligned}\xi = \max(&\tilde{r}_L||r_L, r_{\epsilon_l}, r_{\epsilon_u}||_\infty, \tilde{r}_A||r_A||_\infty, \tilde{r}_S||r_{S_l}, r_{S_u}||_\infty, \tilde{r}_B||r_{B_l}, r_{B_u}||_\infty, ||r_{\epsilon_l}, r_{\epsilon_u}||_\infty,\\ &||r_{SZ_{s_l}}, r_{SZ_{s_u}}, r_{TV_l}, r_{TV_u}, r_{TV_{\epsilon_l}}, r_{TV_{\epsilon_u}}||_\infty)\end{aligned}$$

Return: $\hat{\chi} = (x, \epsilon_l, \epsilon_u, y, z_{s_l}, z_{s_u}, v_l, v_u, v_{\epsilon_l}, v_{\epsilon_u}, s_{s_l}, s_{s_u}, t_l, t_u, t_{\epsilon_l}, t_{\epsilon_u})$

2.2 Riccati based factorization for optimal control problems

In this section, we introduce QPIPM's Riccati option to solve structured QPs. QPIPM is intended to solve QPs arising in OCPs in the form

$$\min_{\{u_k, x_{k+1}, \epsilon_{l,k+1}, \epsilon_{u,k+1}\}_{k=0}^{N-1}} \quad \phi = l_0(u_0) + \sum_{k=1}^{N-1} l_k(x_k, u_k) + l_N(x_N) + \sum_{k=1}^{N} l_{s,k}(\epsilon_{l,k}, \epsilon_{u,k}), \tag{2.27a}$$

$$s.t. \quad x_{k+1} = A_k^\top x_k + B_k^\top u_k + b_k, \qquad k = 0, 1, ..., N-1, \tag{2.27b}$$

$$u_{\min,k} \le u_k \le u_{\max,k}, \qquad k = 0, 1, ..., N-1, \tag{2.27c}$$

$$x_{\min,k} - \epsilon_{l,k} \le S_k^\top x_k \le x_{\max,k} + \epsilon_{u,k}, \qquad k = 1, 2, ..., N, \tag{2.27d}$$

$$(\epsilon_{l,k}, \epsilon_{u,k}) \ge 0, \qquad k = 1, 2, ..., N, \tag{2.27e}$$

where $x_0 = \hat{x}_0$ is a parameter and

$$l_0(u_0) = \frac{1}{2} u_0^\top R_0 u_0 + r_0^\top u_0 + \rho_0, \tag{2.28a}$$

$$l_k(x_k, u_k) = \frac{1}{2} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^\top \begin{bmatrix} Q_k & M_k \\ M_k^\top & R_k \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} + \begin{bmatrix} q_k \\ r_k \end{bmatrix}^\top \begin{bmatrix} x_k \\ u_k \end{bmatrix} + \rho_k, \qquad k = 1, 2, ..., N-1, \tag{2.28b}$$

$$l_N(x_N) = \frac{1}{2} x_N^\top Q_N x_N + q_N^\top x_N + \rho_N, \tag{2.28c}$$

$$l_{s,k}(\epsilon_{l,k}, \epsilon_{u,k}) = \frac{1}{2} \begin{bmatrix} \epsilon_{l,k} \\ \epsilon_{u,k} \end{bmatrix}^\top \begin{bmatrix} Q_{\epsilon_l,k} & \\ & Q_{\epsilon_u,k} \end{bmatrix} \begin{bmatrix} \epsilon_{l,k} \\ \epsilon_{u,k} \end{bmatrix} + \begin{bmatrix} q_{\epsilon_l,k} \\ q_{\epsilon_u,k} \end{bmatrix}^\top \begin{bmatrix} \epsilon_{l,k} \\ \epsilon_{u,k} \end{bmatrix}, \quad k = 1, 2, ..., N. \tag{2.28d}$$

The OCP (2.27) can be written as the general soft constrained QP in the form (2.1) with

$$x = \begin{bmatrix} u_0 & x_1 & u_1 & x_2 & \cdots & u_{N-1} & x_N \end{bmatrix}^\top, \tag{2.29a}$$

$$\epsilon_l = \begin{bmatrix} \epsilon_{l,1} & \epsilon_{l,2} & \cdots & \epsilon_{l,N} \end{bmatrix}^\top, \tag{2.29b}$$

$$\epsilon_u = \begin{bmatrix} \epsilon_{u,1} & \epsilon_{u,2} & \cdots & \epsilon_{u,N} \end{bmatrix}^\top, \tag{2.29c}$$

$$H = \begin{bmatrix} R_0 & & & & & & \\ & Q_1 & M_1 & & & & \\ & M_1^\top & R_1 & & & & \\ & & & \ddots & & & \\ & & & & Q_{N-1} & M_{N-1} & \\ & & & & M_{N-1}^\top & R_{N-1} & \\ & & & & & & Q_N \end{bmatrix}, \tag{2.29d}$$

$$Q_l = \begin{bmatrix} 0 & & & & & & \\ & Q_{\epsilon_l,1} & & & & & \\ & & 0 & & & & \\ & & & Q_{\epsilon_l,2} & & & \\ & & & & \ddots & & \\ & & & & & 0 & \\ & & & & & & Q_{\epsilon_{l,N}} \end{bmatrix}, \tag{2.29e}$$

$$Q_u = \begin{bmatrix} 0 & & & & & & \\ & Q_{\epsilon_u,1} & & & & & \\ & & 0 & & & & \\ & & & Q_{\epsilon_u,2} & & & \\ & & & & \ddots & & \\ & & & & & 0 & \\ & & & & & & Q_{\epsilon_u,N} \end{bmatrix}, \tag{2.29f}$$

$$g = \begin{bmatrix} r_0 & q_1 & r_1 & \cdots & q_{N-1} & r_{N-1} & q_N \end{bmatrix}^\top, \tag{2.29g}$$

$$q_l = \begin{bmatrix} 0 & q_{\epsilon_l,1} & 0 & q_{\epsilon_l,2} & \cdots & 0 & q_{\epsilon_l,N} \end{bmatrix}^\top, \tag{2.29h}$$

$$q_u = \begin{bmatrix} 0 & q_{\epsilon_u,1} & 0 & q_{\epsilon_u,2} & \cdots & 0 & q_{\epsilon_u,N} \end{bmatrix}^\top, \tag{2.29i}$$

$$A = \begin{bmatrix} -B_0^\top & I & & & & \\ & -A_1^\top & -B_1^\top & I & & \\ & & \ddots & \ddots & \ddots & \\ & & & -A_{N-1}^\top & -B_{N-1}^\top & I \end{bmatrix}^\top, \tag{2.29j}$$

$$b = \begin{bmatrix} \tilde{b}_0 & b_1 & \cdots & b_{N-1} \end{bmatrix}^\top, \tag{2.29k}$$

$$l = \begin{bmatrix} u_{\min,0} & -\infty & u_{\min,1} & -\infty & \cdots & u_{\min,N-1} & -\infty \end{bmatrix}^\top, \tag{2.29l}$$

$$u = \begin{bmatrix} u_{\max,0} & \infty & u_{\max,1} & \infty \cdots & u_{\max,N-1} & \infty \end{bmatrix}^\top, \tag{2.29m}$$

$$S = \begin{bmatrix} 0 & & & & & & \\ & S_1 & & & & & \\ & & 0 & & & & \\ & & & S_2 & & & \\ & & & & \ddots & & \\ & & & & & 0 & \\ & & & & & & S_N \end{bmatrix}, \tag{2.29n}$$

$$l_s = \begin{bmatrix} -\infty & x_{\min,1} & -\infty & x_{\min,2} & \cdots & -\infty & x_{\min,N} \end{bmatrix}^\top, \tag{2.29o}$$

$$u_s = \begin{bmatrix} \infty & x_{\max,1} & \infty & x_{\max,2} & \cdots & \infty & x_{\max,N} \end{bmatrix}^\top, \tag{2.29p}$$

where $\tilde{b}_0 = b_0 + A_0^\top x_0$. We point out that QPIPM can solve the OCP (2.27) by applying the definitions (2.29). However, the Riccati based version utilizes the structure, which will result in better computational performance.

In the Riccati version, QPIPM utilizes the structure of the QP (2.27) to compute the search direction. As such, QPIPM does not apply a standard LDL factorization to solve (2.6), but rather a dedicated structure-utilizing Riccati algorithm.

2.2.1 Search direction

In the Riccati mode, the Newton search direction is on the form (2.6) with the provided matrices in (2.29). Due to space restrictions, we do not write out the full system matrix. The right hand side of the linear

system is

$$r_L = \begin{bmatrix} r_{L,u_0} & r_{L,x_1} & r_{L,u_1} & r_{L,x_2} & \cdots & r_{L,u_{N-1}} & r_{L,x_N} \end{bmatrix}^\top, \tag{2.30a}$$

$$r_{\epsilon_l} = \begin{bmatrix} r_{\epsilon_l,1} & r_{\epsilon_l,2} & \cdots & r_{\epsilon_l,N} \end{bmatrix}^\top, \tag{2.30b}$$

$$r_{\epsilon_u} = \begin{bmatrix} r_{\epsilon_u,1} & r_{\epsilon_u,2} & \cdots & r_{\epsilon_u,N} \end{bmatrix}^\top, \tag{2.30c}$$

$$r_A = \begin{bmatrix} r_{A,0} & r_{A,1} & \cdots & r_{A,N-1} \end{bmatrix}^\top, \tag{2.30d}$$

$$r_{S_l} = \begin{bmatrix} r_{S_l,1} & r_{S_l,2} & \cdots & r_{S_l,N} \end{bmatrix}^\top, \tag{2.30e}$$

$$r_{S_u} = \begin{bmatrix} r_{S_u,1} & r_{S_u,2} & \cdots & r_{S_u,N} \end{bmatrix}^\top, \tag{2.30f}$$

$$r_{B_l} = \begin{bmatrix} r_{B_l,0} & r_{B_l,1} & \cdots & r_{B_l,N-1} \end{bmatrix}^\top, \tag{2.30g}$$

$$r_{B_u} = \begin{bmatrix} r_{B_u,0} & r_{B_u,1} & \cdots & r_{B_u,N-1} \end{bmatrix}^\top, \tag{2.30h}$$

$$r_{\epsilon_l} = \begin{bmatrix} r_{\epsilon_l,1} & r_{\epsilon_l,2} & \cdots & r_{\epsilon_l,N} \end{bmatrix}^\top, \tag{2.30i}$$

$$r_{\epsilon_u} = \begin{bmatrix} r_{\epsilon_u,1} & r_{\epsilon_u,2} & \cdots & r_{\epsilon_u,N} \end{bmatrix}^\top, \tag{2.30j}$$

$$r_{SZ_{s_l}} = \begin{bmatrix} r_{SZ_{s_l},1} & r_{SZ_{s_l},2} & \cdots & r_{SZ_{s_l},N} \end{bmatrix}^\top, \tag{2.30k}$$

$$r_{SZ_{s_u}} = \begin{bmatrix} r_{SZ_{s_u},1} & r_{SZ_{s_u},2} & \cdots & r_{SZ_{s_u},N} \end{bmatrix}^\top, \tag{2.30l}$$

$$r_{TV_l} = \begin{bmatrix} r_{TV_l,0} & r_{TV_l,1} & \cdots & r_{TV_l,N-1} \end{bmatrix}^\top, \tag{2.30m}$$

$$r_{TV_u} = \begin{bmatrix} r_{TV_u,0} & r_{TV_u,1} & \cdots & r_{TV_u,N-1} \end{bmatrix}^\top, \tag{2.30n}$$

$$r_{TV_{\epsilon_l}} = \begin{bmatrix} r_{TV_{\epsilon_l},1} & r_{TV_{\epsilon_l},2} & \cdots & r_{TV_{\epsilon_l},N} \end{bmatrix}^\top, \tag{2.30o}$$

$$r_{TV_{\epsilon_u}} = \begin{bmatrix} r_{TV_{\epsilon_u},1} & r_{TV_{\epsilon_u},2} & \cdots & r_{TV_{\epsilon_u},N} \end{bmatrix}^\top, \tag{2.30p}$$

Currently, QPIPM computes the right hand side (2.30) directly from (2.5). However, the algorithm can be improved further by exploiting the structure of the problem and compute individual elements separately.

2.2.2 System reduction

Similarly as in the general case, we eliminate Lagrange multipliers and slack variables. Diagonal matrices are defined as in (2.10) and submatrices are defined with k as subscript. By elimination of Lagrange multipliers and slack variables for inequality constraints and rearranging decision variables, we arrive at the

KKT system (for $N = 3$)

$$
\left[\begin{array}{cccccccccccc|ccc}
\bar{R}_0 & & & & & & & & & & & & B_1 & & \\
 & \bar{Q}_1 & E_1 & F_1 & M_1 & & & & & & & & -I & A_1 & \\
 & E_1^\top & \bar{Q}_{\epsilon_l,1} & & & & & & & & & & & & \\
 & F_1^\top & & \bar{Q}_{\epsilon_u,1} & & & & & & & & & & & \\
 & M_1^\top & & & \bar{R}_1 & & & & & & & & B_1 & & \\
 & & & & & \bar{Q}_2 & E_2 & F_2 & M_2 & & & & -I & A_2 & \\
 & & & & & E_2^\top & \bar{Q}_{\epsilon_l,2} & & & & & & & & \\
 & & & & & F_2^\top & & \bar{Q}_{\epsilon_u,2} & & & & & & & \\
 & & & & & M_2^\top & & & \bar{R}_2 & & & & B_2 & & \\
 & & & & & & & & & \bar{Q}_3 & E_3 & F_3 & -I & & \\
 & & & & & & & & & E_3^\top & \bar{Q}_{\epsilon_l,3} & & & & \\
 & & & & & & & & & F_3^\top & & \bar{Q}_{\epsilon_u,3} & & & \\
\hline
B_0^\top & -I & & & & & & & & & & & & & \\
 & A_1^\top & & & B_1^\top & -I & & & & & & & & & \\
 & & & & & A_2^\top & & & B_2^\top & -I & & & & &
\end{array}\right]
\left[\begin{array}{c}
\Delta u_0 \\ \Delta x_1 \\ \Delta \epsilon_{l,1} \\ \Delta \epsilon_{u,1} \\ \Delta u_1 \\ \Delta x_2 \\ \Delta \epsilon_{l,2} \\ \Delta \epsilon_{u,2} \\ \Delta u_2 \\ \Delta x_3 \\ \Delta \epsilon_{l,3} \\ \Delta \epsilon_{u,3} \\ \hline \Delta y_0 \\ \Delta y_1 \\ \Delta y_2
\end{array}\right]
=
\left[\begin{array}{c}
\bar{r}_{L,u_0} \\ \bar{r}_{L,x_1} \\ \bar{r}_{\epsilon_l,1} \\ \bar{r}_{\epsilon_u,1} \\ \bar{r}_{L,u_1} \\ \bar{r}_{L,x_2} \\ \bar{r}_{\epsilon_l,2} \\ \bar{r}_{\epsilon_u,2} \\ \bar{r}_{L,u_2} \\ \bar{r}_{L,x_3} \\ \bar{r}_{\epsilon_l,3} \\ \bar{r}_{\epsilon_u,3} \\ \hline \bar{r}_{A,0} \\ \bar{r}_{A,1} \\ \bar{r}_{A,2}
\end{array}\right], \tag{2.31}
$$

where

$$
\begin{aligned}
E_k &= S_k D_{s_l,k}, && k = 1, ..., N, && \text{(2.32a)}\\
F_k &= -S_k D_{s_u,k}, && k = 1, ..., N, && \text{(2.32b)}\\
\bar{Q}_k &= Q_k + E_k S_k^\top - F_k S_k^\top, && k = 1, ..., N, && \text{(2.32c)}\\
\bar{Q}_{\epsilon_l,k} &= Q_{\epsilon_l,k} + D_{\epsilon_l,k} + D_{s_l,k}, && k = 1, ..., N, && \text{(2.32d)}\\
\bar{Q}_{\epsilon_u,k} &= Q_{\epsilon_u,k} + D_{\epsilon_u,k} + D_{s_u,k}, && k = 1, ..., N, && \text{(2.32e)}\\
\bar{R}_k &= R_k + D_{l,k} + D_{u,k}, && k = 0, ..., N-1. && \text{(2.32f)}
\end{aligned}
$$

and

$$
\begin{aligned}
\bar{r}_{L,x_k} = & -r_{L,x_k} + S_k(S_{s_l,k}^{-1} Z_{s_l,k}(r_{S_l,k} - Z_{s_l,k}^{-1} r_{SZ_{s_l,k}})) \\
& - S_k(S_{s_u,k}^{-1} Z_{s_u,k}(r_{S_u,k} - Z_{s_u,k}^{-1} r_{SZ_{s_u,k}})), && k = 1, ..., N, && \text{(2.33a)}\\
\bar{r}_{\epsilon_l,k} = & -r_{\epsilon_l,k} + T_{\epsilon_l,k}^{-1} V_{\epsilon_l,k}(r_{B_{\epsilon_l},k} - V_{\epsilon_l,k}^{-1} r_{TV_{\epsilon_l,k}}) \\
& + S_{s_l,k}^{-1} Z_{s_l,k}(r_{S_l,k} - Z_{l,k}^{-1} r_{SZ_{s_l,k}}), && k = 1, ..., N, && \text{(2.33b)}\\
\bar{r}_{\epsilon_u,k} = & -r_{\epsilon_u,k} + T_{\epsilon_u,k}^{-1} V_{\epsilon_u,k}(r_{B_{\epsilon_u},k} - V_{\epsilon_u,k}^{-1} r_{TV_{\epsilon_u,k}}) \\
& + S_{s_u,k}^{-1} Z_{s_u,k}(r_{S_u,k} - Z_{u,k}^{-1} r_{SZ_{s_u,k}}), && k = 1, ..., N, && \text{(2.33c)}\\
\bar{r}_{L,u_k} = & -r_{L,u_k} + T_{l,k} V_{l,k}(r_{B_l,k} - V_{l,k}^{-1} r_{TV_{l,k}}) \\
& - T_{u,k}^{-1} V_{u,k}(r_{B_u,k} - V_{u,k}^{-1} r_{TV_{u,k}}), && k = 0, ..., N-1, && \text{(2.33d)}\\
\bar{r}_{A,k} = & -r_{A,k}, && k = 0, ..., N-1. && \text{(2.33e)}
\end{aligned}
$$

The eliminated Lagrange multipliers and slack variables are

$$\Delta v_{l,k} = T_{l,k}^{-1} V_{l,k} (r_{B_l,k} - V_{l,k}^{-1} r_{TV_l,k}) - T_{l,k}^{-1} V_{l,k} \Delta x_k, \qquad k = 0, ..., N-1, \tag{2.34a}$$

$$\Delta v_{u,k} = T_{u,k}^{-1} V_{u,k} (r_{B_u,k} - V_{u,k}^{-1} r_{TV_u,k}) + T_{u,k}^{-1} V_{u,k} \Delta x_k, \qquad k = 0, ..., N-1, \tag{2.34b}$$

$$\Delta v_{\epsilon_l,k} = T_{\epsilon_l,k}^{-1} V_{\epsilon_l,k} (r_{B_{\epsilon_l,k}} - V_{\epsilon_l,k}^{-1} r_{TV_{\epsilon_l,k}}) - T_{\epsilon_l,k}^{-1} V_{\epsilon_l,k} \Delta \epsilon_{l,k}, \qquad k = 1, ..., N, \tag{2.34c}$$

$$\Delta v_{\epsilon_u,k} = T_{\epsilon_u,k}^{-1} V_{\epsilon_u,k} (r_{B_{\epsilon_u,k}} - V_{\epsilon_u,k}^{-1} r_{TV_{\epsilon_u,k}}) - T_{\epsilon_u,k}^{-1} V_{\epsilon_u,k} \Delta \epsilon_{u,k}, \qquad k = 1, ..., N, \tag{2.34d}$$

$$\begin{aligned} \Delta z_{s_l,k} = {} & S_{s_l,k}^{-1} Z_{s_l,k} (r_{S_l,k} - Z_{s_l,k}^{-1} r_{SZ_{s_l,k}}) \\ & - S_{s_l,k}^{-1} Z_{s_l,k} (S_k^\top \Delta x_k + \Delta \epsilon_{l,k}), \end{aligned} \qquad k = 1, ..., N, \tag{2.34e}$$

$$\begin{aligned} \Delta z_{s_u,k} = {} & S_{s_u,k}^{-1} Z_{s_u,k} (r_{S_u,k} - Z_{s_u,k}^{-1} r_{SZ_{s_u,k}}) \\ & - S_{s_u,k}^{-1} Z_{s_u,k} (-S_k^\top \Delta x_k + \Delta \epsilon_{u,k}), \end{aligned} \qquad k = 1, ..., N, \tag{2.34f}$$

$$\Delta t_{l,k} = -V_{l,k}^{-1} r_{TV_l,k} - V_{l,k}^{-1} T_{l,k} \Delta v_{l,k}, \qquad k = 0, ..., N-1, \tag{2.34g}$$

$$\Delta t_{u,k} = -V_{u,k}^{-1} r_{TV_u,k} - V_{u,k}^{-1} T_{u,k} \Delta v_{u,k}, \qquad k = 0, ..., N-1, \tag{2.34h}$$

$$\Delta t_{\epsilon_l,k} = -V_{\epsilon_l,k}^{-1} r_{TV_{\epsilon_l,k}} - V_{\epsilon_l,k}^{-1} T_{\epsilon_l,k} \Delta v_{\epsilon_l,k}, \qquad k = 1, ..., N, \tag{2.34i}$$

$$\Delta t_{\epsilon_u,k} = -V_{\epsilon_u,k}^{-1} r_{TV_{\epsilon_u,k}} - V_{\epsilon_u,k}^{-1} T_{\epsilon_u,k} \Delta v_{\epsilon_u,k}, \qquad k = 1, ..., N, \tag{2.34j}$$

$$\Delta s_{s_l,k} = -Z_{s_l,k}^{-1} r_{SZ_{s_l,k}} - Z_{s_l,k}^{-1} S_{s_l,k} \Delta z_{s_l,k}, \qquad k = 1, ..., N, \tag{2.34k}$$

$$\Delta s_{s_u,k} = -Z_{s_u,k}^{-1} r_{SZ_{s_u,k}} - Z_{s_u,k}^{-1} S_{s_u,k} \Delta z_{s_u,k}, \qquad k = 1, ..., N. \tag{2.34l}$$

We eliminate the soft constraint slack variables. The resulting system is (for $N = 3$)

$$\left[\begin{array}{ccccccc|ccc} \tilde{R}_0 & & & & & & & B_0 & & \\ & \tilde{Q}_1 & M_1 & & & & & -I & A_1 & \\ & M_1^\top & \tilde{R}_1 & & & & & & B_1 & \\ & & & \tilde{Q}_2 & M_2 & & & & -I & A_2 \\ & & & M_2^\top & \tilde{R}_2 & & & & & B_2 \\ & & & & & \tilde{Q}_3 & & & & -I \\ \hline B_0^\top & -I & & & & & & & & \\ & A_1^\top & B_1^\top & -I & & & & & & \\ & & & A_2^\top & B_2^\top & -I & & & & \end{array}\right] \left[\begin{array}{c} \Delta u_0 \\ \Delta x_1 \\ \Delta u_1 \\ \Delta x_2 \\ \Delta u_2 \\ \Delta x_3 \\ \hline \Delta y_0 \\ \Delta y_1 \\ \Delta y_2 \end{array}\right] = \left[\begin{array}{c} \tilde{r}_{L,u_0} \\ \tilde{r}_{L,x_1} \\ \tilde{r}_{L,u_1} \\ \tilde{r}_{L,x_2} \\ \tilde{r}_{L,u_2} \\ \tilde{r}_{L,x_3} \\ \hline \tilde{r}_{A,0} \\ \tilde{r}_{A,1} \\ \tilde{r}_{A,2} \end{array}\right], \tag{2.35}$$

where

$$\tilde{Q}_k = \bar{Q}_k - E_k \bar{Q}_{\epsilon_l,k}^{-1} E_k^\top - F_k \bar{Q}_{\epsilon_u,k}^{-1} F_k^\top, \qquad k = 1, ..., N, \tag{2.36a}$$

$$\tilde{R}_k = \bar{R}_k, \qquad k = 0, ..., N-1, \tag{2.36b}$$

$$\tilde{r}_{L,x_k} = \bar{r}_{L,x_k} + E_k \bar{Q}_{\epsilon_l,k}^{-1} \bar{r}_{\epsilon_{l,k}} + F_k \bar{Q}_{\epsilon_u,k}^{-1} \bar{r}_{\epsilon_{u,k}}, \qquad k = 1, ..., N, \tag{2.36c}$$

$$\tilde{r}_{L,u_k} = \bar{r}_{L,u_k}, \qquad k = 0, ..., N-1, \tag{2.36d}$$

$$\tilde{r}_A = \bar{r}_A, \qquad k = 0, ..., N-1. \tag{2.36e}$$

and the eliminated slack variables are

$$\Delta \epsilon_{l,k} = \bar{Q}_{\epsilon_l,k}^{-1} \left(\bar{r}_{\epsilon_{l,k}} - E_k^\top \Delta x_k \right), \tag{2.37a}$$

$$\Delta \epsilon_{u,k} = \bar{Q}_{\epsilon_u,k}^{-1} \left(\bar{r}_{\epsilon_{u,k}} - F_k^\top \Delta x_k \right). \tag{2.37b}$$

The KKT-system (2.35) can be solved with Riccati recursion, and finally the remaining part of the search direction can be compute from (2.37) and (2.34).

2.2.3 Riccati recursion algorithm

We apply a Riccati recursion based algorithm to solve structured systems of linear equations in the form (2.35). For simplicity of notation, we write the system (2.35) as

$$\left[\begin{array}{ccccccc|ccc} R_0 & & & & & & & B_0 & & \\ & Q_1 & M_1 & & & & & -I & A_1 & \\ & M_1^\top & R_1 & & & & & & B_1 & \\ & & & Q_2 & M_2 & & & & -I & A_2 \\ & & & M_2^\top & R_2 & & & & & B_2 \\ & & & & & P_3 & & & & -I \\ \hline B_0^\top & -I & & & & & & & & \\ & A_1^\top & B_1^\top & -I & & & & & & \\ & & & A_2^\top & B_2^\top & -I & & & & \end{array}\right] \left[\begin{array}{c} \Delta u_0 \\ \Delta x_1 \\ \Delta u_1 \\ \Delta x_2 \\ \Delta u_2 \\ \Delta x_3 \\ \hline \Delta y_0 \\ \Delta y_1 \\ \Delta y_2 \end{array}\right] = - \left[\begin{array}{c} r_0 \\ q_1 \\ r_1 \\ q_2 \\ r_2 \\ p_3 \\ \hline b_0 \\ b_1 \\ b_2 \end{array}\right], \tag{2.38}$$

We point out that the data in (2.38) should not be confused with variables with similar names previously introduced. Algorithm 2 and 3 introduce the factorization and solution phase of the Riccati recursion algorithm (Jørgensen 2004, Wahlgreen and Jørgensen 2022). We point out that the Algorithm 2 returns the cholesky factorization of $R_{e,k}$, which QPIPM applies in Algorithm 3 to solve the linear systems involving $R_{e,k}$.

Note the negation on the right hand side in (2.38). Before calling the Riccati algorithm to solve (2.35), QPIPM negates the right hand side of the system (2.35) such that it is in the form (2.38).

2.2.4 Algorithm

In Riccati mode, QPIPM follows the steps in Algorithm 1, where the LDL-factorization step and LDL-solve step are replaced with the the Riccati factorization and Riccati solve algorithms in Algorithm 2 and 3.

2.2.5 A note on bounds

The Riccati recursion part of QPIPM does allow for hard output constraints, i.e., box constraints on x_k (we have not provided these equations here, but they are easily included based on the input, u_k, box constraints). Therefore, if elements corresponding to x_k in l and/or u are not set to infinity, QPIPM does include the bound. On the other hand, QPIPM does not support soft input constraints in Riccati mode. Therefore, elements corresponding to the inputs, u_k, in S will never be accessed even if the corresponding values of l_s and/or l_u are not set to infinity. QPIPM does however require the entries in S corresponding to u_k to be set. We have implemented QPIPM in this way such that one can turn Riccati mode on and off without changing the provided QP formulation.

Algorithm 2: Riccati factorization

Input: $\{R_k, Q_k, M_k, A_k, B_k\}_{k=0}^{N-1}$, P_N.

1. Compute,

$$\begin{aligned} R_{e,k} &= R_k + B_k P_{k+1} B_k^\top, \\ K_k &= -R_{e,k}^{-1}(M_k^\top + B_k P_{k+1} A_k^\top), \\ P_k &= Q_k + A_k P_{k+1} A_k^\top - K_k^\top R_{e,k} K_k, \end{aligned}$$

for $k = N-1, N-2, ..., 1$ and

$$R_{e,0} = R_0 + B_0 P_1 B_0^\top.$$

Return: $\{R_{e,k}, chol(R_{e,k}), P_{k+1}\}_{k=0}^{N-1}$, $\{K_k\}_{k=1}^{N-1}$.

Algorithm 3: Riccati solution

Input: $\{Q_k, M_k, A_k, B_k, R_{e,k}, chol(R_{e,k}), P_{k+1}\}_{k=0}^{N-1}$, $\{K_k\}_{k=1}^{N-1}$.

1. Compute,

$$\begin{aligned} a_k &= -R_{e,k}^{-1}(r_k + B_k(P_{k+1} b_k + p_{k+1})), \\ p_k &= q_k + A_k(P_{k+1} b_k + p_{k+1}) + K_k^\top (r_k + B_k(P_{k+1} b_k + p_{k+1})), \end{aligned}$$

for $k = N-1, N-2, ..., 1$ and

$$a_0 = -R_{e,0}^{-1}(r_0 + B_0(P_1 \tilde{b}_0 + p_1)).$$

2. Compute the solution, $\{\Delta u_k, \Delta x_{k+1}\}_{k=0}^{N-1}$,

$$\begin{aligned} \Delta u_0 &= a_0, \\ \Delta x_1 &= B_0^\top \Delta u_0 + \tilde{b}_0, \end{aligned}$$

and

$$\begin{aligned} \Delta u_k &= K_k \Delta x_k + a_k, \\ \Delta x_{k+1} &= A_k^\top \Delta x_k + B_k^\top \Delta u_k + b_k, \end{aligned}$$

for $k = 1, 2, ..., N-1$.

3. Compute the Lagrange multipliers, $\{\Delta y_k\}_{k=0}^{N-1}$,

$$\begin{aligned} \Delta y_{N-1} &= P_N \Delta x_N + p_N, \\ \Delta y_{k-1} &= A_k \Delta y_k + Q_k \Delta x_k + M_k \Delta u_k + q_k, \end{aligned}$$

for $k = N-1, N-2, ..., 1$.

Return: $\{\Delta u_k, \Delta x_{k+1}, \Delta y_k\}_{k=0}^{N-1}$.

CHAPTER 3

Implementation of QPIPM in Matlab and C

In this chapter, we introduce how QPIPM can be called in both Matlab and in C. Both versions are part of a gitlab-repository, which can be cloned with the command line command

```
git clone https://gitlab.gbar.dtu.dk/SCGroup/QPIPM.git
```

3.1 Matlab

QPIPM in Matlab has the following interface:

```
1 function [x, stat] = QPIPM(H, g, A, b, C, d, l, u, options, ls, S, us, Ql, Qu, ql, qu)
```

Inputs:

The inputs `H`, `g`, `A`, `b`, `l`, `u`, `ls`, `us`, `S`, `Ql`, `Qu`, `ql`, and `qu` are as in (2.1). The inputs `C` and `d` implements general inequality constraints in the form

$$C^\top x \geq d. \tag{3.1}$$

The general inequality constraints (3.1) have only been included in QPIPM for testing purposes and are ignored in Riccati mode. The inputs `ls` – `qu` can be left empty in which case QPIPM solves a problem without soft constraints. The `options` input is a structure with the following fields

`print`	0 or 1 to print iteration information	Default: 1
`tol`	Convergence tolerance	Default: 10^{-8}
`maxit`	Maximum iterations	Default: 100
`riccati`	0 or 1 to turn Riccati mode off/on	Default: 0
`N`	Horizon. Required if `riccati=1`	Default: NaN

We point out that QPIPM takes the same inputs and have the same outputs when Riccati mode is off and on. When applying Riccati mode, QPIPM assumes that the provided matrices are structured as described in section 2.2. QPIPM will not check that this is the case. Therefore, Riccati mode can be applied for a non-structured QP, but the result will likely be wrong.

Outputs:

The output `x` is the solution at convergence or after maximum iterations are reached. QPIPM prints a warning message in the case that maximum iterations are reached. The `stat` output is a structure with the

following fields

`obj`	Objective value at solution
`conv`	0 (not converged) or 1 (converged)
`iter`	Number of iterations
`lamEq`	Lagrange multipliers for equality constraints
`lamIneq`	Lagrange multipliers for inequality constraints
`lamBn`	Lagrange multipliers for bound constraints
`lamZs`	Lagrange multipliers for soft constraints
`lamEpsBn`	Lagrange multipliers for ϵ-bound constraints
`eps`	ϵ-slack variables

3.2 C

As previously mentioned, the C version of QPIPM does currently not have the option to include soft constraints. QPIPM in C has the following interface:

```
void QPIPM(
    // Inputs
    struct mat  *H          ,
    struct vec  *g          ,
    struct mat  *A          ,
    struct vec  *b          ,
    struct mat  *C          ,
    struct vec  *d          ,
    struct vec  *l          ,
    struct vec  *u          ,
    void        *optionsIn  ,
    mem         *memory     ,

    // Outputs
    struct vec  *x          ,
    void        *statIn
)
```

The structures `vec`, `mat`, and `mem` are vector, matrix, and memory structures, respectively. Theses structures are defined in the dependency `SCInterface`, which is shortly introduced in section 3.2.2. In the following, we introduce the inputs and outputs of the C version.

Inputs:

The inputs `H`, `g`, `A`, `b`, `C`, `d`, `l`, and `u` are as in the Matlab version. The `optionsIn` input is a options structure of type `optionsQPIPM_t`, which has the fields

`print`	0 or 1 to print iteration information	Default: 1
`tol`	Convergence tolerance	Default: 10^{-8}
`maxit`	Maximum iterations	Default: 100
`riccati`	0 or 1 to turn Riccati mode off/on	Default: 0
`N`	Horizon. Required if `riccati=1`	Default: NaN
`bigN`	Numbers above treated as infinity	Default: 10^{20}

The input `memory` is a structure of type `mem`, which contains sufficient integer and double memory for QPIPM (see section 3.2.1).

Outputs:

The output `x` is the solution at convergence or after maximum iterations are reached. Similarly to the Matlab version, QPIPM in C prints a warning message if the maximum number of iterations are reached. The `stat` structure is of type `statQPIPM_t` and has the following fields

`obj`	Objective value at solution
`conv`	0 (not converged) or 1 (converged)
`iter`	Number of iterations
`lamEq`	Lagrange multipliers for equality constraints
`lamIneq`	Lagrange multipliers for inequality constraints
`lamBn`	Lagrange multipliers for bound constraints

3.2.1 Memory allocation

QPIPM requires both integer and double workspace, which should be allocated in the input memory structure. QPIPM features the function

```
1    void workspaceQPIPM( int n, int me, int mi, int *iwork, int *dwork )
```

which given the dimensions of the QP, `n` (decision variables), `me` (equality constraints), and `mi` (inequality constraints), computes the required workspace for QPIPM. Then the amount of integer workspace, `iwork`, and double workspace, `dwork`, can be use to initialize the `memory` input with sufficient memory. Additionally, the `stat` structure for the output is required to be initialized, which can be done with the function

```
1    void createStatQPIPM( const int n, const int me, const int mi, statQPIPM_t
         *const stat )
```

`createStatQPIPM` allocates the required memory for the output `stat` structure. Note that when finished using the `stat` structure, the memory can be freed with the function

```
1    void destroyStatQPIPM( statQPIPM_t *stat )
```

3.2.2 Dependencies

QPIPM is a part of the private gitlib-repository `SCProject`, which is a project containing a series of git repositories. QPIPM is dependent on the following two repositories in `SCProject`

`SCInterface`	A set of structure and function definitions
`linalg`	A set of vector and matrix linear algebra functions

Additionally, `linalg` is BLAS dependent and requires linking to a BLAS installation on the system.

3.2.3 Gitlab

The private Gitlab group `SCGroup` grants access to all projects contained in `SCProject`. Therefore, the three projects, `QPIPM`, `SCInterface`, and `linalg` are also included in `SCGroup`. When access is granted to `SCGroup`, one can clone the whole `SCProject` or parts of it. To apply QPIPM, one has to clone `QPIPM`, `SCInterface`, and `linalg` (and install a version of BLAS). The C version of QPIPM includes a `settings.mk` file where the dependency paths can be set. The three git repositories can be cloned with the following command line commands (accompanied with a username and password):

```
git clone https://gitlab.gbar.dtu.dk/SCGroup/SCInterface.git
git clone https://gitlab.gbar.dtu.dk/SCGroup/linalg.git
git clone https://gitlab.gbar.dtu.dk/SCGroup/QPIPM.git
```

3.2.4 Doxygen documentation

The C version of QPIPM is documented with Doxygen. The Doxygen documentation is available in `QPIPM/C/docs`, which can be compiled by typing `doxygen` in the command line. Afterwards, the documentation is available in `QPIPM/C/docs/results/html/index.html`, which will open in a browser. The documentation includes descriptions of all QPIPM functions and their inputs and outputs. Note, this requires an installation of Doxygen on the system.

3.3 Examples

Both the Matlab and C version of QPIPM has a few test examples. The Matlab version has a driver to test the implementation on a linearized four tank system. The C version includes a simple test example and a few examples showing that the algorithm can be called in parallel to solve multiple QPs. The examples can be found in the `examples` folder in the Matlab and C version of QPIPM.

Note: The C version of QPIPM is thread-safe such that it can be called in parallel to solve multiple QPs. This feature requires linking to a thread-safe BLAS library, e.g., BLASFEO (Frison et al. 2018, 2020).

CHAPTER 4

Conclusion

In this part, we introduced the Riccati based primal-dual interior-point software, QPIPM, to solve structured quadratic programming problems (QPs). QPIPM is a software package that is stored in a private gitlab-repository `QPIPM`, which is part of the project `SCProject`. QPIPM has a Matlab version and a C version, where the Matlab version is intended for testing purposes and have not been implemented for computational speed. The C version is thread-safe due to internal distribution of memory allocated prior to calling QPIPM. QPIPM can solve QPs with equality constraints, box constraints, and soft constraints. However, currently only the Matlab version supports soft constraints. We have provided the mathematical details of QPIPM and introduced the implementation of QPIPM in both Matlab and C. We have provided the interfaces of the implementations and described the inputs and outputs. In the C version, we have elaborated on how to allocate the needed memory and how to link to the introduced dependencies.

Part II

NLPSQP

CHAPTER 5

Introduction

We introduce the sequential quadratic programming (SQP) software, NLPSQP (nonlinear-programming-sequential-quadratic-programming), for solution of nonlinear programming problems (NLPs). NLPSQP applies an iterative sequential quadratic programming (SQP) algorithm. In each iteration, NLPSQP performs three major steps, 1) solve a quadratic programming problem (QP) subproblem with QPIPM, 2) apply a line-search algorithm to ensure sufficient decrease in a merit function, and 3) perform a Broyden–Fletcher–Goldfarb–Shanno (BFGS) update to avoid the need of evaluating second order derivatives. NLPSQP supports a Riccati mode for solution of structured problems arising in optimal control problems (OCPs). NLPSQP is intended for use in nonlinear model predictive control (NMPC) and economic NMPC (ENMPC) applications. We have implemented NLPSQP in both a Matlab version and a C version. The Matlab version is intended for testing purposes, while the C version is intended for uncertainty quantification studies of closed-loop systems with Monte Carlo simulation. To that end, the C version of NLPSQP is implemented thread-safe to enable parallel scaling in Monte Carlo simulations, i.e., NLPSQP can be called in parallel to solve different NLPs. The thread-safety of NLPSQP is ensured by internally distributing memory allocated prior to calling NLPSQP. Similarly to QPIPM, the Matlab version supports soft constraints, while the C version lacks this feature due to time constraints. The current implementation of NLPSQP is still work in progress and can likely be optimized for better performance. However, the most computational work is done in QPIPM.

In this report, we introduce the mathematical details in the NLPSQP implementation and introduce the interfaces of NLPSQP in both Matlab and C. NLPSQP is stored in a private gitlab-repository `NLPSQP` and is part of the project `SCProject`, which is implemented in C and contains a number of other gitlab-repositories. For the C version, we introduce the other dependencies in `SCProject` and explain how to allocate the required memory prior to calling NLPSQP.

We point out that the implementation of NLPSQP is highly inspired by previous work (Wächter and Biegler 2006, Kaysfeld et al. 2023).

CHAPTER 6

Mathematical details

We have developed NLPSQP to solve NLPs with equality constraints, box constraints, and soft constraints in the form,

$$\min_{x,\epsilon_l,\epsilon_u} \quad f(x) + Q(\epsilon_l, \epsilon_u), \tag{6.1a}$$

$$s.t. \quad g(x) = 0, \tag{6.1b}$$

$$l \leq x \leq u, \tag{6.1c}$$

$$l_s - \epsilon_l \leq s(x) \leq u_s + \epsilon_u, \tag{6.1d}$$

$$\epsilon_l, \epsilon_u \geq 0, \tag{6.1e}$$

where $x \in \mathbb{R}^n$, $\epsilon_l \in \mathbb{R}^{m_s}$, $\epsilon_u \in \mathbb{R}^{m_s}$, $f : \mathbb{R}^n \to \mathbb{R}$, $Q : \mathbb{R}^{2m_s} \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^{m_e}$, $l \in \mathbb{R}^n$, $u \in \mathbb{R}^n$, $s : \mathbb{R}^n \to \mathbb{R}^{m_s}$, $l_s \in \mathbb{R}^{m_s}$, and $u_s \in \mathbb{R}^{m_s}$. We point out that l and u can have elements set to $-\infty$ and ∞ in which case NLPSQP eliminates these constraints in a pre-computing phase. We let m_l and m_u denote the actual number of lower bounds and upper bounds after elimination of ∞-bounds, respectively.

The penalty function, $Q(\cdot)$, is a combination of quadratic and linear terms similarly to the penalty term the QP solved in QPIPM,

$$Q(\epsilon_l, \epsilon_u) = \frac{1}{2}\epsilon_l^\top Q_l \epsilon_l + q_l^\top \epsilon_l + \frac{1}{2}\epsilon_u^\top Q_u \epsilon_u + q_u^\top \epsilon_u, \tag{6.2}$$

where we assume that $Q_l \in \mathbb{R}^{m_s \times m_s}$ and $Q_u \in \mathbb{R}^{m_s \times m_s}$ are diagonal matrices.

NLPSQP is an iterative algorithm that in each iteration goes through the following three major steps,

- Compute the search-direction by solving a QP-subproblem,
- A line-search algorithm to ensure sufficient decrease in a merit function,
- A BFGS update for Lagrangian Hessian approximation.

We let the superscript $[l]$ denote the l'th iteration of NLPSQP. In each iteration NLPSQP takes the step

$$x^{[l+1]} = x^{[l]} + \alpha^{[l]} \Delta x^{[l]}, \tag{6.3}$$

where $\alpha^{[l]}$ is a step-size computed by the line-search algorithm to ensure sufficient decrease in a merit function and $\Delta x^{[l]}$ is the search direction computed as the solution to the QP-subproblem.

NLPSQP features a version to solve the general NLP (6.1) and a Riccati recursion based version to solve structured NLPs arising in OCPs. The non-Riccati version is not optimized and primarily implemented for testing purposes.

6.1 Sequential quadratic programming algorithm

In this section, we introduce the SQP algorithm implemented in NLPSQP to solve soft constrained NLPs in the form (6.1).

6.1.1 Optimality conditions

We define the Lagrangian function for (6.1), which is

$$\begin{aligned}\mathcal{L}(x,\lambda,\pi_l,\pi_u,\pi_{\epsilon_l},\pi_{\epsilon_u},\pi_{l_s},\pi_{u_s}) = f(x) + Q(\epsilon_l,\epsilon_u) - \lambda^\top g(x) - \pi_l^\top(x-l) - \pi_u^\top(u-x) \\ - \pi_{\epsilon_l}^\top\epsilon_l - \pi_{\epsilon_u}^\top\epsilon_u - \pi_{l_s}^\top(s(x)-l_s+\epsilon_l) - \pi_{u_s}^\top(u_s+\epsilon_u-s(x)).\end{aligned} \tag{6.4}$$

$\lambda \in \mathbb{R}^m$ are equality constraint Lagrange multipliers, $\pi_l \in \mathbb{R}^n$ are lower bound Lagrange multipliers, $\pi_u \in \mathbb{R}^n$ are upper bound Lagrange multipliers, $\pi_{\epsilon_l} \in \mathbb{R}^{m_s}$ are ϵ_l-non-negativity Lagrange multipliers, $\pi_{\epsilon_u} \in \mathbb{R}^{m_s}$ are ϵ_u-non-negativity Lagrange multipliers, $\pi_{l_s} \in \mathbb{R}^{m_s}$ are lower soft constraint Lagrange multipliers, and $\pi_{u_s} \in \mathbb{R}^{m_s}$ are upper soft constraint Lagrange multipliers. The Lagrangian gradient with respect to the decision variables, x, and the soft constraint Lagrange multipliers, ϵ_l and ϵ_u, is then

$$\nabla_x\mathcal{L} = \nabla f(x) - \nabla g(x)\lambda - \pi_l + \pi_u - \nabla s(x)\pi_{l_s} + \nabla s(x)\pi_{u_s}, \tag{6.5a}$$

$$\nabla_{\epsilon_l}\mathcal{L} = \nabla_{\epsilon_l} Q(\epsilon_l,\epsilon_u) - \pi_{\epsilon_l} - \pi_{l_s}, \tag{6.5b}$$

$$\nabla_{\epsilon_u}\mathcal{L} = \nabla_{\epsilon_u} Q(\epsilon_u,\epsilon_u) - \pi_{\epsilon_u} - \pi_{u_s}, \tag{6.5c}$$

where $\mathcal{L}(x,\lambda,\pi_l,\pi_u,\pi_{\epsilon_l},\pi_{\epsilon_u},\pi_{l_s},\pi_{u_s})$. The first order KKT-conditions for (6.1) are given as

$$\nabla_x\mathcal{L} = 0, \tag{6.6a}$$

$$\nabla_{\epsilon_l}\mathcal{L} = 0, \tag{6.6b}$$

$$\nabla_{\epsilon_u}\mathcal{L} = 0, \tag{6.6c}$$

$$g(x) = 0, \tag{6.6d}$$

$$x - l \geq 0, \qquad u - x \geq 0, \tag{6.6e}$$

$$s(x) - l_s + \epsilon_l \geq 0, \qquad u_s + \epsilon_u - s(x) \geq 0, \tag{6.6f}$$

$$\epsilon_l \geq 0, \qquad \epsilon_u \geq 0. \tag{6.6g}$$

6.1.2 Quadratic programming subproblem

NLPSQP solves a QP-subproblem in each iteration to get the search direction. For simplicity of notation, we disregard the iteration superscript $[l]$ in this section ($x = x^{[l]}$, $\Delta x = \Delta x^{[l]}$, $\epsilon_l = \epsilon_l^{[l]}$, $\epsilon_u = \epsilon_u^{[l]}$, $W = W^{[l]}$). The QP-subproblem solved in NLPSQP is

$$\min_{\Delta x,\epsilon_l,\epsilon_u} \quad \frac{1}{2}\Delta x^\top W \Delta x + \nabla f(x)^\top \Delta x + Q(\epsilon_l,\epsilon_u), \tag{6.7a}$$

$$s.t. \quad \nabla g(x)^\top \Delta x = -g(x), \tag{6.7b}$$

$$l - x \leq \Delta x \leq u - x, \tag{6.7c}$$

$$l_s - s(x) - \epsilon_l \leq \nabla s(x)^\top \Delta x \leq u_s - s(x) + \epsilon_u, \tag{6.7d}$$

$$\epsilon_l, \epsilon_u \geq 0. \tag{6.7e}$$

W is a BFGS approximation of the second order derivative of the Lagrangian. We denote the Lagrange multipliers of the QP-subproblem (6.7) as: μ for equality constrain, τ_l and τ_u for bound constraints, and τ_{l_s} and τ_{u_s} for soft constraints. We point out that the slack variables ϵ_l and ϵ_u in the QP-subproblem (6.7)

are identical to those in the original NLP (6.1). Therefore, the Lagrange multipliers for the ϵ-bounds in the QP-subproblem are exactly π_{ϵ_l} and π_{ϵ_u}, i.e., the Lagrange multipliers from the original NLP (6.1).

The Lagrange multipliers of the QP-subproblem (6.7) are related to the Lagrange multipliers of the original NLP (6.1) as

$$\mu = \lambda + \Delta\lambda, \tag{6.8a}$$

$$\tau_l = \pi_l + \Delta\pi_l, \qquad \tau_u = \pi_u + \Delta\pi_u, \tag{6.8b}$$

$$\tau_{l_s} = \pi_{l_s} + \Delta\pi_{l_s}, \qquad \tau_{u_s} = \pi_{u_s} + \Delta\pi_{u_s}. \tag{6.8c}$$

Using the relation (6.8), we can compute the search direction for the Lagrange multipliers,

$$(\Delta\lambda, \Delta\pi_l, \Delta\pi_u, \Delta\pi_{l_s}, \Delta\pi_{u_s}). \tag{6.9}$$

We point out that the solution to the QP-subproblem (6.7) ensures to satisfy the linear constraints in the original NLP (6.1), i.e., the bound constraints (6.1c) and the ϵ-non-negativity constraints (6.1e). Note also that the QP-subproblem (6.7) is in the form (2.1) and can be solved with QPIPM.

6.1.3 Line-search

NLPSQP applies a backtracking line-search algorithm to compute the step-size, α, that ensures sufficient degrees in Powell's l_1-merit function (Powell 1978, Jørgensen 2004). We have adapted the merit function to include soft constraint

$$\begin{aligned} P(x) = f(x) + \sigma^\top |g(x)| + \kappa_l^\top |\min(0, s(x) - l_s + \epsilon_l)| \\ + \kappa_u^\top |\max(0, s(x) - u_s - \epsilon_u)|. \end{aligned} \tag{6.10}$$

The j'th element of the vectors, σ, κ_l, and κ_u, are defined as

$$\sigma_j = \max\left(|\mu_j|, \frac{1}{2}(\sigma_j + |\mu_j|)\right), \qquad j = 1, ..., m, \tag{6.11a}$$

$$\kappa_{l,j} = \max\left(|\tau_{l_s,j}|, \frac{1}{2}(\kappa_{l,j} + |\tau_{l_s,j}|)\right), \qquad j = 1, ..., m_s, \tag{6.11b}$$

$$\kappa_{u,j} = \max\left(|\tau_{u_s,j}|, \frac{1}{2}(\kappa_{u,j} + |\tau_{u_s,j}|)\right), \qquad j = 1, ..., m_s, \tag{6.11c}$$

where $\sigma_j = |\mu_j|$, $\kappa_{l,j} = |\tau_{l_s,j}|$, and $\kappa_{u,j} = |\tau_{u_s,j}|$ in the first iteration ($l = 0$). Note, linear constraints are not included in the merit function since these are satisfied by construction of the QP-subproblem (6.7). Also, the penalty function, $Q(\epsilon_l, \epsilon_u)$, is not included in the merit function since ϵ_l and ϵ_u are not affected by changes in the step-size, α. We define the following function

$$T(\alpha) = P(x^{[l+1]}) = P(x^{[l]} + \alpha\Delta x^{[l]}). \tag{6.12}$$

We define sufficient decrease with the Armijo condition as

$$T(\alpha) \leq T(0) + c_1\alpha \mathrm{D}_{\Delta x} T(0), \tag{6.13}$$

where

$$\begin{aligned} T(\alpha) = f(x^{[l]} + \alpha\Delta x^{[l]}) &+ \sigma^\top |g(x^{[l]} + \alpha\Delta x^{[l]})| \\ &+ \kappa_l^\top |\min(0, s(x^{[l]} + \alpha\Delta x^{[l]}) - l_s + \epsilon_l^{[l]})| \\ &+ \kappa_u^\top |\max(0, s(x^{[l]} + \alpha\Delta x^{[l]}) - u_s - \epsilon_u^{[l]})| \end{aligned} \tag{6.14a}$$

$$\begin{aligned} T(0) = f(x^{[l]}) &+ \sigma^\top |g(x^{[l]})| \\ &+ \kappa_l^\top |\min(0, s(x^{[l]}) - l_s + \epsilon_l^{[l]})| \\ &+ \kappa_u^\top |\max(0, s(x^{[l]}) - u_s - \epsilon_u^{[l]})| \end{aligned} \tag{6.14b}$$

$$\begin{aligned} \mathrm{D}_{\Delta x} T(0) = \nabla f(x^{[l]})^\top \Delta x^{[l]} &- \sigma^\top |g(x^{[l]})| \\ &- \kappa_l^\top |\min(0, s(x^{[l]}) - l_s + \epsilon_l^{[l]})| \\ &- \kappa_u^\top |\max(0, s(x^{[l]}) - u_s - \epsilon_u^{[l]})| \end{aligned} \tag{6.14c}$$

The backtracking line-search algorithm is (Kaysfeld et al. 2023)

1. Set $\alpha = 1$
2. Check the Armijo condition (6.13) and if satisfied **break** with $\alpha^{[l]} = \alpha$ as output
3. Reduce step $\alpha \leftarrow \beta\alpha$
4. Go to 2.

We apply $c_1 = 10^{-4}$ and $\beta = 0.5$, which are similar values as chosen in IPOPT (Wächter and Biegler 2006).

Once the step-size, $\alpha^{[l]}$, is computed by the line-search algorithm, NLPSQP performs the step

$$x^{[l+1]} = x^{[l]} + \alpha^{[l]}\Delta x^{[l]}, \qquad \lambda^{[l+1]} = \lambda^{[l]} + \alpha^{[l]}\Delta\lambda^{[l]}, \tag{6.15a}$$

$$\pi_l^{[l+1]} = \pi_l^{[l]} + \alpha^{[l]}\Delta\pi_l^{[l]}, \qquad \pi_u^{[l+1]} = \pi_u^{[l]} + \alpha^{[l]}\Delta\pi_u^{[l]}, \tag{6.15b}$$

$$\pi_{l_s}^{[l+1]} = \pi_{l_s}^{[l]} + \alpha^{[l]}\Delta\pi_{l_s}^{[l]}, \qquad \pi_{u_s}^{[l+1]} = \pi_{u_s}^{[l]} + \alpha^{[l]}\Delta\pi_{u_s}^{[l]}. \tag{6.15c}$$

6.1.4 BFGS update

NLPSQP requires only gradient information. Thus, no second order derivatives are required to apply NLPSQP. In NLPSQP, we apply a BFGS update for the Lagrange Hessian. Specifically, we apply a damped version of the BFGS update to ensure positive definiteness of the update (Powell 1978). In the remainder of this section, we apply the following definitions to ease notation: $W = W^{[l]}$ and $\bar{W} = W^{[l+1]}$.

We define the following two vectors

$$s = x^{[l+1]} - x^{[l]}, \tag{6.16a}$$

$$y = \nabla_x \mathcal{L}_+ - \nabla_x \mathcal{L}_-, \tag{6.16b}$$

where

$$\begin{aligned} \nabla_x \mathcal{L}_- &= \nabla_x \mathcal{L}(x^{[l]}, \lambda^{[l+1]}, \pi_l^{[l+1]}, \pi_u^{[l+1]}, \pi_{\epsilon_l}^{[l+1]}, \pi_{\epsilon_u}^{[l+1]}, \pi_{l_s}^{[l+1]}, \pi_{u_s}^{[l+1]}) \\ &= \nabla f(x^{[l]}) - \nabla g(x^{[l]})\lambda^{[l+1]} - \pi_l^{[l+1]} + \pi_u^{[l+1]} - \nabla s(x^{[l]})\pi_{l_s}^{[l+1]} + \nabla s(x^{[l]})\pi_{u_s}^{[l+1]}, \end{aligned} \tag{6.17a}$$

$$\begin{aligned} \nabla_x \mathcal{L}_+ &= \nabla_x \mathcal{L}(x^{[l+1]}, \lambda^{[l+1]}, \pi_l^{[l+1]}, \pi_u^{[l+1]}, \pi_{\epsilon_l}^{[l+1]}, \pi_{\epsilon_u}^{[l+1]}, \pi_{l_s}^{[l+1]}, \pi_{u_s}^{[l+1]}) \\ &= \nabla f(x^{[l+1]}) - \nabla g(x^{[l+1]})\lambda^{[l+1]} - \pi_l^{[l+1]} + \pi_u^{[l+1]} - \nabla s(x^{[l+1]})\pi_{l_s}^{[l+1]} + \nabla s(x^{[l+1]})\pi_{u_s}^{[l+1]}. \end{aligned} \tag{6.17b}$$

We point out that the π_l and π_u contributions in $\mathcal{L}_-$ and $\mathcal{L}_+$ can be ignored as these are eliminated in (6.16b). Now let

$$r = \theta y + (1 - \theta) W s, \tag{6.18}$$

where

$$\theta = \begin{cases} 1 & s^\top y \geq 0.2 s^\top W s \\ \dfrac{0.8 s^\top W s}{s^\top W s - s^\top y} & s^\top y < 0.2 s^\top W s \end{cases} \tag{6.19}$$

The damped BFGS update is then

$$\bar{W} = W - \frac{(Ws)(Ws)^\top}{s^\top (Ws)} + \frac{r r^\top}{s^\top r}. \tag{6.20}$$

NLPSQP applies $W^{[0]} = I$, where I is an identity matrix of proper dimensions.

6.1.5 Initialization

NLPSQP requires an initial guess on the decision variables $x^{[0]}$, which the user has to provide. The soft constraint slack variables, ϵ_l and ϵ_u, and all Lagrange multipliers are initialized by NLPSQP to 0.

6.1.6 Convergence

NLPSQP converges when the KKT-conditions (6.6) are satisfied, i.e., a local optimum is located. In practice, NLPSQP evaluates a scaled convergence criterion based on a user-specified convergence tolerance $\epsilon > 0$

$$||\nabla_x \mathcal{L} / s_d||_\infty \leq \epsilon, \tag{6.21a}$$
$$||\nabla_{\epsilon_l} \mathcal{L}||_\infty \leq \epsilon, \tag{6.21b}$$
$$||\nabla_{\epsilon_u} \mathcal{L}||_\infty \leq \epsilon, \tag{6.21c}$$
$$||g(x)||_\infty \leq \epsilon, \tag{6.21d}$$
$$||\min(0, s(x) - l_s + \epsilon_l)||_\infty \leq \epsilon, \tag{6.21e}$$
$$||\max(0, s(x) - u_s - \epsilon_u)||_\infty \leq \epsilon, \tag{6.21f}$$

where

$$s_d = \max\left(s_{\max}, \frac{||\lambda||_1 + ||\pi_l||_1 + ||\pi_u||_1}{m + m_l + m_u}\right) / s_{\max}. \tag{6.22}$$

We apply $s_{\max} = 100$ similarly to IPOPT (Wächter and Biegler 2006). NLPSQP evaluates the criterion (6.21) after the step (6.15) is computed.

6.1.7 Algorithm

Algorithm 4 presents a detailed implementation guide for NLPSQP.

6.2 Riccati version for optimal control problems

In this section, we introduce the Riccati recursion option for NLPSQP. In this mode, NLPSQP assumes that the NLP has a specific structure, where the QP-subproblem is in the form (2.27) such that QPIPM can apply Riccati mode. Therefore, the following is required of the NLP for NLPSQP to apply Riccati mode,

Algorithm 4: NLPSQP pseudo code

Input: Initial guess, x_0, and soft constrained NLP,

$$\begin{aligned} \min_{x,\epsilon_l,\epsilon_u} \quad & f(x) + Q(\epsilon_l, \epsilon_u), \\ s.t. \quad & g(x) = 0, \\ & l \leq x \leq u, \\ & l_s - \epsilon_l \leq s(x) \leq u_s + \epsilon_u, \\ & \epsilon_l, \epsilon_u \geq 0, \end{aligned}$$

i.e. the functions: $f(x)$, $g(x)$, $s(x)$ and the matrices and vectors: Q_l, Q_u, g_l, g_u, l, u, l_s, l_u.

- Initialize ($l = 0$):

$$x^{[0]} = x_0, \quad \epsilon_l^{[0]} = \epsilon_u^{[0]} = 0, \quad \lambda^{[0]} = \pi_l^{[0]} = \pi_u^{[0]} = \pi_{\epsilon_l}^{[0]} = \pi_{\epsilon_u}^{[0]} = \pi_{l_s}^{[0]} = \pi_{u_s}^{[0]} = 0, \quad W^{[0]} = I.$$

- Check convergence (6.21).

while *not converged* **do**

1. Update iteration counter: $l \leftarrow l + 1$.
2. Apply QPIPM to solve the QP-subproblem for $\Delta x^{[l]} = \Delta x$, $\epsilon_l^{[l]} = \epsilon_l$, and $\epsilon_u^{[l]} = \epsilon_u$,

$$\begin{aligned} \min_{\Delta x,\epsilon_l,\epsilon_u} \quad & \frac{1}{2}\Delta x^\top W \Delta x + \nabla f(x)^\top \Delta x + Q(\epsilon_l, \epsilon_u), \\ s.t. \quad & \nabla g(x)^\top \Delta x = -g(x), \\ & l - x \leq \Delta x \leq u - x, \\ & l_s - s(x) - \epsilon_l \leq \nabla s(x)^\top \Delta x \leq u_s - s(x) + \epsilon_u, \\ & \epsilon_l, \epsilon_u \geq 0, \end{aligned}$$

   where $W = W^{[l]}$ and $x = x^{[l]}$. Note, the Lagrange multipliers for the QP-subproblem are

$$\begin{aligned} \mu &= \lambda^{[l]} + \Delta\lambda^{[l]}, \\ \tau_l &= \pi_l^{[l]} + \Delta\pi_l^{[l]}, & \tau_u &= \pi_u^{[l]} + \Delta\pi_u^{[l]}, \\ \tau_{l_s} &= \pi_{l_s}^{[l]} + \Delta\pi_{l_s}^{[l]}, & \tau_{u_s} &= \pi_{u_s}^{[l]} + \Delta\pi_{u_s}^{[l]}, \\ \pi_{\epsilon_l} &= \pi_{\epsilon_l}^{[l]}, & \pi_{\epsilon_u} &= \pi_{\epsilon_u}^{[l]}. \end{aligned}$$

3. Compute the step-size, $\alpha^{[l]}$, with the line-search algorithm as descriped in section 6.1.3.
4. Compute the step

$$\begin{aligned} x^{[l+1]} &= x^{[l]} + \alpha^{[l]}\Delta x^{[l]}, & \lambda^{[l+1]} &= \lambda^{[l]} + \alpha^{[l]}\Delta\lambda^{[l]}, \\ \pi_l^{[l+1]} &= \pi_l^{[l]} + \alpha^{[l]}\Delta\pi_l^{[l]}, & \pi_u^{[l+1]} &= \pi_u^{[l]} + \alpha^{[l]}\Delta\pi_u^{[l]}, \\ \pi_{l_s}^{[l+1]} &= \pi_{l_s}^{[l]} + \alpha^{[l]}\Delta\pi_{l_s}^{[l]}, & \pi_{u_s}^{[l+1]} &= \pi_{u_s}^{[l]} + \alpha^{[l]}\Delta\pi_{u_s}^{[l]}. \end{aligned}$$

5. Check convergence (6.21) - **break** if criterion is satisfied.
6. Apply the BFGS update as described in section 6.2.1

$$W^{[l+1]} = W^{[l]} - \frac{(W^{[l]}s)(W^{[l]}s)^\top}{s^\top(W^{[l]}s)} + \frac{rr^\top}{s^\top r}.$$

Return: x, y, π_l, π_u, π_{ϵ_l}, π_{ϵ_u}, π_{l_s}, and π_{u_s}.

1. The decision variable vector contains two variables, x_k and u_k, and are structured as follows

$$\xi = \begin{bmatrix} u_0 & x_1 & u_1 & \cdots & x_{N-1} & u_{N-1} & x_N \end{bmatrix}^\top. \tag{6.23}$$

2. The equality constraints, $g(\xi) = 0$, are structured such that the gradient has the structure

$$\nabla g(\xi) = \begin{bmatrix} -B_0 & & & & & & \\ I & -A_1 & & & & & \\ & -B_1 & & & & & \\ & I & -A_2 & & & & \\ & & -B_2 & & & & \\ & & & \ddots & \ddots & -A_{N-1} & \\ & & & & & -B_{N-1} & \\ & & & & & I & \end{bmatrix}. \tag{6.24}$$

3. The soft constraint function, $s(\xi)$, contain no soft constraints on the inputs and its gradient has a block diagonal structure in the form

$$\nabla s(\xi) = \begin{bmatrix} 0 & & & & & & \\ & S_1 & & & & & \\ & & 0 & & & & \\ & & & S_2 & & & \\ & & & & \ddots & & \\ & & & & & 0 & \\ & & & & & & S_N \end{bmatrix}. \tag{6.25}$$

4. In addition, NLPSQP in Riccati mode applies a block BFGS update to ensure the block diagonal Hessian structure for the QP-subproblem. For the update to be a good approximation of the true Lagrange Hessian, we suggest that the NLP has the following Lagrange Hessian structure,

$$\nabla^2_{\xi\xi}\mathcal{L} = \begin{bmatrix} W_0 & & & & \\ & W_1 & & & \\ & & \ddots & & \\ & & & W_{N-1} & \\ & & & & W_N \end{bmatrix}, \tag{6.26}$$

with

$$W_0 = R_0, \tag{6.27a}$$

$$W_k = \begin{bmatrix} Q_k & M_k \\ M_k^\top & R_k \end{bmatrix}, \qquad k = 1, ..., N-1, \tag{6.27b}$$

$$W_N = P_N. \tag{6.27c}$$

Under the above assumptions, NLPSQP can apply QPIPM in Riccati mode to efficiently solve the QP-subproblem. In the following section, we introduce the applied block BFGS update to maintain the required block diagonal Hessian structure in the QP-subproblem. We point out that the line-search algorithm, convergence criterion, and algorithm initialization is identical to the non-Riccati mode version of NLPSQP.

6.2.1 Block BFGS update

In Riccati mode, NLPSQP applies a block BFGS update to maintain a block diagonal Hessian structure for the QP-subproblem. A usual BFGS update would result in a dense matrix and would therefore not produce the structure required to apply Riccati recursion in the QP-subproblem. In the remainder of this section, we apply $W_k = W_k^{[l]}$ and $\bar{W}_k = W_k^{[l+1]}$ for simplicity of notation.

We define the vectors, s and y, similar to (6.16),

$$s = \xi^{[l+1]} - \xi^{[l]}, \tag{6.28a}$$

$$y = \nabla_\xi \mathcal{L}_+ - \nabla_\xi \mathcal{L}_-, \tag{6.28b}$$

where $\mathcal{L}_-$ and $\mathcal{L}_+$ is defined as in (6.17). We let s_k and y_k be sub-vectors in s and y corresponding to the diagonal block matrices, W_k, in (6.26). Similarly to the normal damped BFGS update, we define

$$r_k = \theta_k y_k + (1 - \theta_k) W_k s_k, \tag{6.29}$$

where

$$\theta_k = \begin{cases} 1 & s_k^\top y_k \geq 0.2 s_k^\top W_k s_k \\ \dfrac{0.8 s_k^\top W_k s_k}{s_k^\top W_k s_k - s_k^\top y_k} & \text{else} \end{cases} \tag{6.30}$$

Finally, the BFGS update of each block is

$$\bar{W}_k = \begin{cases} W_k - \dfrac{(W_k s_k)(W_k s_k)^\top}{s_k^\top (W_k s_k)} + \dfrac{r_k r_k^\top}{s_k^\top r_k} & \kappa > \epsilon_m \\ W_k & \text{else} \end{cases} \tag{6.31}$$

ϵ_m is the machine precision of the computer and $\kappa = \min(\kappa_1, \kappa_2)$ with $\kappa_1 = s_k^\top W_k s_k$ and $\kappa_2 = s_k^\top r_k$. These update safeguards are implemented to avoid zero-division if some blocks converge faster than others. NLPSQP initializes the full block diagonal structured Hessian approximation as $W^{[0]} = I$, where I is an identity matrix of proper dimensions. Numerical tests have shown that numerical errors might cause indefinite BFGS block updates. NLPSQP applies the simple strategy to reset the entire Hessian approximation to identity if an indefinite update is detected.

6.2.2 Application to solve OCPs

In this section, we introduce an OCP and demonstrate that direct multiple shooting discretization transcribes the OCP to an NLP in the form required for NLPSQP to apply Riccati mode.

We consider continuous OCPs in the form

$$\min_{[x(t);u(t)]_{t_0}^{t_f}} \quad \phi = \int_{t_0}^{t_f} l(t, x(t), u(t), p) dt + \hat{l}(x(t_f), p), \tag{6.32a}$$

$$\text{s.t.} \quad x(t_0) = x_0, \tag{6.32b}$$

$$\dot{x}(t) = f(t, x(t), u(t), d(t), p), \qquad t_0 \leq t \leq t_f, \tag{6.32c}$$

$$u_{\min}(t) \leq u(t) \leq u_{\max}(t), \qquad t_0 \leq t \leq t_f. \tag{6.32d}$$

By direct multiple shooting discretization, we transcribe the continuous OCP (6.32) to the following NLP,

$$\min_{\{u_k, x_{k+1}\}_{k=0}^{N-1}} \quad \phi = \hat{\Phi}\left(\{u_k, x_{k+1}\}_{k=0}^{N-1}\right), \tag{6.33a}$$

$$\text{s.t.} \quad R_k = x_{k+1} - F(t_k, x_k, u_k, d_k, p) = 0, \qquad k = 0, ..., N-1, \tag{6.33b}$$

$$u_{\min,k} \leq u_k \leq u_{\max,k}, \qquad k = 0, ..., N-1, \tag{6.33c}$$

where $F(\cdot)$ is a numerical integration scheme and

$$\hat{\Phi}\left(\{u_k, x_{k+1}\}_{k=0}^{N-1}\right) = \Bigg\{ \sum_{k=0}^{N-1} \int_{t_k}^{t_{k+1}} l(x_k(t), u_k, d_k, p)dt + \hat{l}(x_N, p) : \begin{aligned} & x_0(t_0) = x_0, \\ & x_k(t_k) = x_k, && k = 1, ..., N-1, \\ & \dot{x}_k(t) = f(t, x_k(t), u_k, d_k, p), && t_k \le t \le t_{k+1} \end{aligned} \Bigg\}. \tag{6.34}$$

We add soft constraints to the discretized OCP and get a soft constrained OCP in the form

$$\min_{\{u_k, x_{k+1}, \epsilon_{l,k+1}, \epsilon_{u,k+1}\}_{k=0}^{N-1}} \quad \phi = \hat{\Phi}\left(\{u_k, x_{k+1}\}_{k=0}^{N-1}\right) + Q(\epsilon_l, \epsilon_u), \tag{6.35a}$$

$$\text{s.t.} \quad R_k = x_{k+1} - F(t_k, x_k, u_k, d_k, p) = 0, \qquad k = 0, ..., N-1, \tag{6.35b}$$

$$u_{\min,k} \le u_k \le u_{\max,k}, \qquad k = 0, ..., N-1, \tag{6.35c}$$

$$x_{\min,k} - \epsilon_{l,k} \le s_k(x_k) \le x_{\max,k} + \epsilon_{u,k}, \qquad k = 1, ..., N, \tag{6.35d}$$

where $\epsilon_l = \begin{bmatrix} \epsilon_{l,1} & \epsilon_{l,2} & \cdots & \epsilon_{l,N} \end{bmatrix}^\top$, $\epsilon_u = \begin{bmatrix} \epsilon_{u,1} & \epsilon_{u,2} & \cdots & \epsilon_{u,N} \end{bmatrix}^\top$, and $Q(\epsilon_l, \epsilon_u)$ is as in (6.2). In the following, we demonstrate that the NLP (6.35) satisfies the requirements for NLPSQP to be called in Riccati mode.

Equality constraints

We write the equality constraints (6.35b) as

$$g(\xi) = \begin{bmatrix} R_0 & R_1 & \cdots & R_{N-1} \end{bmatrix}^\top, \tag{6.36}$$

and observe that the gradient has the required form (6.24),

$$\nabla g(\xi) = \begin{bmatrix} -\nabla_{u_0} F & & & & & \\ I & -\nabla_{x_1} F & & & & \\ & -\nabla_{u_1} F & & & & \\ & I & -\nabla_{x_2} F & & & \\ & & -\nabla_{u_2} F & & & \\ & & I & \ddots & -\nabla_{x_{N-1}} F & \\ & & & & -\nabla_{u_{N-1}} F & \\ & & & & I & \end{bmatrix}, \tag{6.37}$$

where we define

$$A_k = \nabla_{x_k} F = \nabla_{x_k} F(t_k, x_k, u_k, d_k, p), \qquad k = 1, ..., N-1, \tag{6.38}$$

$$B_k = \nabla_{u_k} F = \nabla_{u_k} F(t_k, x_k, u_k, d_k, p), \qquad k = 0, ..., N-1. \tag{6.39}$$

Soft constraints

Similarly, we write the soft constraints (6.35d) as

$$s(\xi) = \begin{bmatrix} s_1(x_1) & s_2(x_2) & \cdots & s_N(x_N) \end{bmatrix}^\top \tag{6.40}$$

and observe that the gradient has the required form (6.25),

$$\nabla s(\xi) = \begin{bmatrix} 0 & & & & & & \\ & \nabla_{x_1} s_1 & & & & & \\ & & 0 & & & & \\ & & & \nabla_{x_2} s_2 & & & \\ & & & & \ddots & & \\ & & & & & 0 & \\ & & & & & & \nabla_{x_N} s_N \end{bmatrix}, \tag{6.41}$$

where we define

$$S_k = \nabla_{x_k} s_k = \nabla_{x_k} s_k(x_k), \qquad k = 1, ..., N. \tag{6.42}$$

Lagrangian Hessian block diagonal structure

We notice that the soft constrained NLP (6.35) has a partially separable (in the states, x, and inputs, u) objective function and constraints. Therefore, the Lagrangian is also partial separable as

$$\mathcal{L}(\xi) = \mathcal{L}_0(u_0) + \sum_{k=1}^{N-1} \mathcal{L}_k(x_k, u_k) + \mathcal{L}_N(x_N), \tag{6.43}$$

where we leave out the Lagrange multiplier dependencies in $\mathcal{L}$ for simplicity. As a result of the partial separability, the Lagrangian Hessian is given as

$$\nabla^2_{\xi\xi}\mathcal{L} = \begin{bmatrix} \nabla^2_{u_0,u_0}\mathcal{L}_0 & & & & \\ & \begin{matrix} \nabla^2_{x_1,x_1}\mathcal{L}_1 & \nabla^2_{x_1,u_1}\mathcal{L}_1 \\ \nabla^2_{u_1,x_1}\mathcal{L}_1 & \nabla^2_{u_1,u_1}\mathcal{L}_1 \end{matrix} & & & \\ & & \ddots & & \\ & & & \begin{matrix} \nabla^2_{x_{N-1},x_{N-1}}\mathcal{L}_{N-1} & \nabla^2_{x_{N-1},u_{N-1}}\mathcal{L}_{N-1} \\ \nabla^2_{u_{N-1},x_{N-1}}\mathcal{L}_{N-1} & \nabla^2_{u_{N-1},u_{N-1}}\mathcal{L}_{N-1} \end{matrix} & \\ & & & & \nabla^2_{x_N,x_N}\mathcal{N}_N \end{bmatrix}. \tag{6.44}$$

We notice that the Lagrangian Hessian has the required block diagonal structure (6.26), where

$$R_0 = \nabla^2_{u_0,u_0}\mathcal{L}_0, \tag{6.45a}$$

$$\begin{bmatrix} Q_k & M_k \\ M_k^\top & R_k \end{bmatrix} = \begin{bmatrix} \nabla^2_{x_k,x_k}\mathcal{L}_k & \nabla^2_{x_k,u_k}\mathcal{L}_k \\ \nabla^2_{u_k,x_k}\mathcal{L}_k & \nabla^2_{u_k,u_k}\mathcal{L}_k \end{bmatrix}, \tag{6.45b}$$

$$P_N = \nabla^2_{x_N,x_N}\mathcal{L}_N. \tag{6.45c}$$

We point out that the matrices Q_k, R_k, M_k, and P_N are not required to be evaluated as NLPSQP apply the block BFGS update described in section 6.2.1.

6.2.3 Algorithm

In Riccati mode, NLPSQP applies the steps in Algorithm 4. However, NLPSQP calls QPIPM in Riccati mode to solve the QP-subproblem and applies the block BFGS update described in section 6.2.1 to ensure the required structure of the QP-subproblem.

6.2.4 A note on bounds

Even though we have not included hard output constraints in the problem (6.32), NLPSQP does have the option to include these in the OCP.

CHAPTER 7

Implementation of NLPSQP in Matlab and C

In this chapter, we introduce how to call NLPSQP in both Matlab and in C. Both versions are part of a private gitlab-repository, which can be cloned with the command line command

```
git clone https://gitlab.gbar.dtu.dk/SCGroup/NLPSQP.git
```

7.1 Matlab

NLPSQP in Matlab has the following interface:

```
1  function [x, stat] = NLPSQP(ffun, x0, gfun, hfun, l, u, options, varargin)
```

Inputs:

The inputs are as follows: `ffun` is the objective function, $f(x)$, `x0` is the user-provided initial condition, x_0, `gfun` is the equality constraint function, $g(x)$, `l` is the lower bound vector, `u` is the upper bound vector, `options` is an options structure, and `varargin` contains a set of variable input arguments for $f(x)$, $g(x)$, and $h(x)$.

The input `hfun` is for general inequality constraints,

$$h(x) \geq 0, \tag{7.1}$$

which is only implemented for testing purposes and not optimized in any way. In Riccati mode, general inequality constraints are not supported and `hfun` has to be left empty.

The input `options` contains a number of options and the possibility to enable soft constraints. The options structure has the following fields

`print`	0 or 1 to print iteration information	Default: 1
`tol`	Convergence tolerance	Default: 10^{-8}
`tolStep`	Minimum allowed step-size	Default: 10^{-8}
`maxit`	Maximum iterations	Default: 100
`riccati`	0 or 1 to turn Riccati mode off/on	Default: 0
`N`	Horizon. Required if `riccati=1`	Default: NaN
`printQP`	0 or 1 to print QPIPM iteration information	Default: 0
`tolQP`	QPIPM convergence tolerance	Default: $10^{-2} \cdot$ `tol`
`maxitQP`	QPIPM maximum iterations	Default: 100

`subWarn`	0 or 1 to suppress Matlab warnings	Default: 0
`softLin`	0 or 1 to specify linear soft constraints	Default: 0
`softNonlin`	0 or 1 to specify nonlinear soft constraints	Default: 0
`softProblemLin`	Structure with linear soft constraints	Default: empty
`softProblemNonlin`	Structure with nonlinear soft constraints	Default: empty

The two soft constrained problem structures `softProblemLin` and `softProblemNonlin` are required to be set if `softLin=1` and `softNonlin=1` respectively. Note also that `softLin` and `softNonlin` cannot be set to 1 at the same time. The fields of `softProblemLin` and `softProblemNonlin` are

`ls`	Lower soft bound
`S/sfun`	Linear case: Soft constraint matrix - Nonlinear case: Soft constraint function
`us`	Upper soft bound
`Ql`	Quadratic penalty maitrx for lower soft bound - assumed diagonal
`Qu`	Quadratic penalty matrix for upper soft bound - assumed diagonal
`ql`	Linear penalty vector for lower soft bound
`qu`	Linear penalty vector for upper soft bound

We point out that NLPSQP takes the same inputs and have the same outputs when Riccati mode is off and on. When applying Riccati mode, NLPSQP assumes that the provided NLP has the required structure as described in section 6.2. NLPSQP does not check that this is the case. Riccati mode can be applied for a non-structured NLP, but NLPSQP makes assumptions about the structure in the QP-subproblem, which likely leads to poor search directions. This can cause bad convergence properties of NLPSQP and might ultimately prevent convergence.

Outputs:

The output `x` is the solution vector after 1) convergence, i.e., `x` is a local optimum, 2) the maximum number of iterations are reached in which case NLPSQP prints a warning, and 3) the computed step-size is smaller than `tolStep` in which case NLPSQP prints a warning. The `stat` output is a structure with the following fields

`obj`	Objective value at solution
`conv`	0 (not converged) or 1 (converged)
`iter`	Number of iterations
`lamEq`	Lagrange multipliers for equality constraints
`lamIneq`	Lagrange multipliers for inequality constraints
`lamBn`	Lagrange multipliers for bound constraints
`eps`	ϵ-slack variables

7.2 C

The C version of NLPSQP does not include soft constraints as previously mentioned. The interface for NLPSQP in C is

```
void NLPSQP(
    // Inputs
    objectiveFunctionNLPSQP_t               *ffun       ,
    struct vec                              *x0         ,
    void                                    *varargin   ,
    equalityConstraintFunctionNLPSQP_t      *gfun       ,
    inequalityConstraintFunctionNLPSQP_t    *hfun       ,
    struct vec                              *l          ,
    struct vec                              *u          ,
    optionsNLPSQP_t                         *options    ,
    mem                                     *memory     ,

    // Outputs
    struct vec                              *x          ,
    statNLPSQP_t                            *stat
)
```

Inputs:

NLPSQP takes three function inputs `ffun`, `gfun`, and `hfun` similarly to the Matlab version. The `varargin` input is a set of variable input arguments required by the three input functions. The vectors `l` and `u` are the lower and upper bounds, respectively. The `memory` input contains both integer and double workspace required by NLPSQP (see section 7.2.1). The `options` inputs is a structure of type `optionsNLPSQP_t` which has the following fields

`print`	0 or 1 to print iteration information	Default: 1
`tol`	Convergence tolerance	Default: 10^{-8}
`tolStep`	Minimum allowed step-size	Default: 10^{-8}
`maxit`	Maximum iterations	Default: 100
`riccati`	0 or 1 to turn Riccati mode off/on	Default: 0
`N`	Horizon. Required if `riccati=1`	Default: `idummy`
`printQP`	0 or 1 to print QPIPM iteration information	Default: 0
`tolQP`	QPIPM convergence tolerance	Default: $10^{-2} \cdot$ `tol`
`maxitQP`	QPIPM maximum iterations	Default: 100
`bigN`	Numbers above treated as infinity	Default: 10^{20}

where `idummy = -11111` is an integer dummy variable defined in NLPSQP. The function types of `ffun`, `gfun`, and `hfun` are

```
typedef void objectiveFunctionNLPSQP_t(
    // Inputs
    struct vec  *x          ,
    void        *varargin   ,
    int          nargout    ,

    // Outputs
```

```
    double      *f          ,
    struct vec  *df
);
```

```
typedef void equalityConstraintFunctionNLPSQP_t(
    // Inputs
    struct vec  *x          ,
    void        *varargin   ,
    int          nargout    ,

    // Outputs
    struct vec  *g          ,
    struct mat  *dg
);
```

```
typedef void inequalityConstraintFunctionNLPSQP_t(
    // Inputs
    struct vec  *x          ,
    void        *varargin   ,
    int          nargout    ,

    // Outputs
    struct vec  *h          ,
    struct mat  *dh
);
```

The three function types have the same inputs, which are

`x`	Decision variables
`varargin`	A set of variable input arguments
`nargout`	Number of outputs to evaluate

and their outputs are

`f`	Objective value
`df`	Gradient for objective function
`g`	Equality constraints vector
`dg`	Gradient for equality constraints
`h`	Inequality constraints vector
`dh`	Gradient for inequality constraints

Outputs:

The output `x` is the solution vector after 1) convergence, i.e., `x` is a local optimum, 2) the maximum number of iterations are reached in which case NLPSQP prints a warning, and 3) the computed step-size is smaller than `tolStep` in which case NLPSQP prints a warning. The `stat` output is a structure of type

statNLPSQP_t with the following fields

obj	Objective value at solution
conv	0 (not converged) or 1 (converged)
iter	Number of iterations
lamBn	Lagrange multipliers for bound constraints
lamEq	Lagrange multipliers for equality constraints
lamIneq	Lagrange multipliers for inequality constraints

7.2.1 Memory allocation

NLPSQP requires both integer and double workspace, which should be allocated in the input memory structure. NLPSQP features the function

```
1    void workspaceNLPSQP( int n, int me, int mi, int *iwork, int *dwork )
```

which given the dimensions of the NLP, n (decision variables), me (equality constraints), and mi (inequality constraints), computes the required workspace for NLPSQP. Then the amount of integer workspace, iwork, and double workspace, dwork, can be use to initialize the memory input with sufficient memory. Additionally, the stat structure for the output is required to be initialized, which can be done with the function

```
1    void createStatNLPSQP( const int n, const int me, const int mi,
         statNLPSQP_t *const stat )
```

createStatNLPSQP allocates the required memory for the output stat structure. Note that when finished using the stat structure, the memory can be freed with the function

```
1    void destroyStatNLPSQP( statNLPSQP_t *stat )
```

7.2.2 Dependencies

NLPSQP is a part of the private gitlib-repository SCProject, which is a project containing a series of git repositories. NLPSQP is dependent on the following three repositories in SCProject:

SCInterface	A set of structure and function definitions
linalg	A set of vector and matrix linear algebra functions
util	A set of utility functions
QPIPM	A primal-dual interior-point software to solve QPs

Additionally, linalg is BLAS dependent and requires linking to a BLAS installation on the system.

7.2.3 Gitlab

The private Gitlab group SCGroup grants access to all projects contained in SCProject. Therefore, the four projects, NLPSQP, QPIPM, SCInterface, and linalg are also included in SCGroup. When access is granted to SCGroup, one can clone the whole SCProject or parts of it. To apply NLPSQP, one has to clone NLPSQP, QPIPM, SCInterface, util, and linalg (and install a version of BLAS). The C version of NLPSQP includes a settings.mk file, where the dependency paths can be set. The five git

repositories can be cloned with the following command line commands (accompanied with a username and password):

```
git clone https://gitlab.gbar.dtu.dk/SCGroup/SCInterface.git
git clone https://gitlab.gbar.dtu.dk/SCGroup/linalg.git
git clone https://gitlab.gbar.dtu.dk/SCGroup/util.git
git clone https://gitlab.gbar.dtu.dk/SCGroup/QPIPM.git
git clone https://gitlab.gbar.dtu.dk/SCGroup/NLPSQP.git
```

7.2.4 Doxygen documentation

The C version of NLPSQP is documented with Doxygen. The Doxygen documentation is available in `NLPSQP/C/docs`, which can be compiled by typing `doxygen` in the command line. Afterwards, the documentation is available in `NLPSQP/C/docs/results/html/index.html`, which opens in a browser. The documentation includes descriptions of all NLPSQP functions and their inputs and outputs. Note, this requires an installation of Doxygen on the system.

7.3 Examples

Both the Matlab and C version of NLPSQP has a few test examples. The Matlab version includes a driver to test NLPSQP on a simple NLP and a few drivers to apply NLPSQP to solve OCPs for both a four tank system model and a continuous stirred tank reactor (CSTR) model. The drivers show how to apply NLPSQP and demonstrate NLPSQP with/without Riccati mode and with/without soft constraints.

The C version also includes a test simple test NLP. Additionally, the C version includes test examples that demonstrate that NLPSQP can solve an OCP for the CSTR model similarly to the Matlab version. Finally, the C version includes an example that demonstrates that NLPSQP can be called to solve multiple OCPs in parallel using `openMP`. This example requires linking to a thread-safe BLAS installation, e.g., BLASFEO (Frison et al. 2018, 2020).

Furthermore, we refer to previous work, where we have integrated NLPSQP in an NMPC. The NMPC was applied in large-scale closed-loop Monte Carlo simulations to quantify uncertainties in the closed-loop system (Kaysfeld et al. 2023).

CHAPTER 8

Conclusion

In this part, we introduced the sequential quadratic programming (SQP) software, NLPSQP, to solve structured nonlinear programming problems (NLPs). NLPSQP is a software package that is stored in a private gitlab-repository `NLPSQP`, which is part of the project `SCProject`. NLPSQP has a Matlab version and a C version. In the current version only NLPSQP in Matlab supports soft constraints. We have provided the mathematical details of NLPSQP and introduced the implementation of NLPSQP in both Matlab and C. We showed interfaces of the implementations and described the inputs and outputs. In the C version, we elaborated on how to allocate the needed memory for NLPSQP and how to link to the introduced the dependencies.

The C version of NLPSQP is intended for application in parallel Monte Carlo simulation of closed-loop systems containing nonlinear model predictive control (NMPC) algorithms. Due to the thread-safety of the implementation, NLPSQP can be applied to solve multiple OCPs in parallel with almost linear scaling and is therefore well-suited for the purpose.

Bibliography

Frison, G. and Jørgensen, J. B.: 2013, Efficient implementation of the riccati recursion for solving linear-quadratic control problems, *IEEE International Conference on Control Applications (CCA), Hyderabad, India* pp. 1117–1122.

Frison, G., Kouzoupis, D., Sartor, T., Zanelli, A. and Diehl, M.: 2018, BLASFEO: Basic linear algebra subroutines for embedded optimization, *ACM Transactions on Mathematical Software* **4**4(4).

Frison, G., Sartor, T., Zanelli, A. and Diehl, M.: 2020, The BLAS API of BLASFEO: Optimizing performance for small matrices, *ACM Transactions on Mathematical Software* **4**6(2).

Jørgensen, J. B.: 2004, *Moving Horizon Estimation and Control*, PhD thesis, Technical University of Denmark.

Jørgensen, J. B., Frison, G., Gade-Nielsen, N. F. and Damman, B.: 2012, Numerical methods for solution of the extended linear-quadratic control problem, *IFAC Proceedings Volumes* **4**5(17), 187–193.

Karush, W.: 1939, *Minima of functions of several variables with inequalities as side constraints*, M.sc. thesis, University of Chicago, Chicago, Illinois.

Kaysfeld, M. W., Zanon, M. and Jørgensen, J. B.: 2023, Performance quantification of a nonlinear model predictive controller by parallel Monte Carlo simulations of a closed-loop system, *Proceedings of the 21st European Control Conference (ECC), Bucharest, Romania, 2023, accepted* .

Kjeldsen, T. H.: 2000, A contextualized historical analysis of the Kuhn–Tucker theorem in nonlinear programming: The impact of world war II, *Historia Mathematica* **2**7(4), 331–361.

Kuhn, H. W. and Tucker, A. W.: 1951, Nonlinear programming, *University of California Press* pp. 481–492.

Mehrotra, S.: 1992, On the implementation of a primal-dual interior point method, *SIAM Journal on Optimization* **2**(4), 575–601.

Powell, M. J. D.: 1978, A fast algorithm for nonlinearly constrained optimization calculations, *Proceedings of the Biennial Conference on Numerical Analysis* pp. 144–157.

Rao, C. V., Wright, S. J. and Rawlings, J. B.: 1998, Application of interior-point methods to model predictive control, *Journal of Optimization Theory and Applications* **9**9(3), 723–757.

Wahlgreen, M. R. and Jørgensen, J. B.: 2022, On the implementation of a preconditioned riccati recursion based primal-dual interior-point algorithm for input constrained optimal control problems, *IFAC-PapersOnLine* **5**5(7), 346–351. 13th IFAC Symposium on Dynamics and Control of Process Systems, including Biosystems (DYCOPS) 2022.

Wahlgreen, M. R., Reenberg, A. T., Nielsen, M. K., Rydahl, A., Ritschel, T. K. S., Dammann, B. and Jørgensen, J. B.: 2021, A high-performance monte carlo simulation toolbox for uncertainty quantification of closed-loop systems, *Proceedings of the 60th IEEE Conference on Decision and Control (CDC)* pp. 6755–6761.

Wächter, A. and Biegler, L. T.: 2006, On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, *Mathematical Programming* **1**06(1), 25–57.