Trajectory Optimization Methods for Implicit Time-Lag Systems

Marco Ruggia

with help from
Simon Zimmermann and Roi Poranne

May 2021

Contents

1 Introduction

This paper present a handful of discrete trajectory-optimization (TO) methods. These algorithms depend upon two components. First, a model which describes the state-trajectory of a system given an input-trajectory and, second, a function which assigns a cost to every state-input-trajectory pair. Through optimization, these algorithms are used to find the state- and input-trajectories that minimize the cost function while satisfying the system model.

The TO methods discussed here have one commonality. They can be used with systems that do not necessarily have an explicit form of the dynamics equation (e.g. $x_{i+1} = f(x_i, u_i)$), but instead can take an implicit form (e.g. $g(x_{i+1}, x_i, u_i) = 0$). Additionally, systems involving time-lag are addressed. In such systems, the next state not only depends on the current state, but also on previous states and inputs. An example of a system where those two requirements are needed, are systems derived from finite element methods that are discretized in time with some numerical scheme.

Specifically two methods are covered in this paper: *Gauss-Newton* (GN) methods, which iteratively improve the entire input-trajectory in a way that decreases the cost function until a minimum is reached. And *Iterative Linear-Quadratic Regulator* (ILQR) methods, which calculate an input policy at each timestep which tries to minimize the cost of the remaining trajectory.

These two methods are further subdivided into *implicit*, *sparse* and *primal-dual* variants. The *implicit* variant leverages the implicit function theorem to get the derivatives required for an improvement step. The *sparse* variant incorporates the implicit function theorem into the improvement step equations, trading more equations for sparsity. Finally, the *primal-dual* variant, which drops the requirement that the system dynamics always be satisfied and optimizes over both the input and state trajectory simultaneously.

A few of the variants will make use of so called *forward-simulation* (FS) methods. These are numerical methods that solve an implicit system formulations for the next state, given the current state and input (e.g. given x_i, u_i and $g(x_{i+1}, x_i, u_i) = 0$, compute x_{i+1}). A prominent member of these is the Newton method for the solution of nonlinear equations. Forward-simulation methods won't be further discussed in this paper. More information can be found in the hand-written notes in the appendix.

2 Gauss-Newton

The class of discrete trajectory optimization problems that can be solved by the Gauss-Newton methods presented in this paper may be stated as follows:

$$\min_{U,X} \quad \mathcal{O}(X,U) \quad \text{s.t.} \quad \mathcal{G}(X,U) = 0 \tag{1}$$

Where X and U are the input- and the state-trajectory vectors formed by stacking the input- and the state-vectors at each timestep. The scalar function $\mathcal{O}$, assigns a cost to each X, U, and the vector function $\mathcal{G}$ describes a residual that is only zero when the pair X, U satisfies the system dynamics.

2.1 Implicit Gauss-Newton

To derive the implicit version of the Gauss-Newton method, it is assumed that an explicit formulation of the system dynamics exists and has the form $X = \mathcal{F}(U)$. Then the TO problem becomes:

$$\min_{U} \quad \mathcal{O}(\mathcal{F}(U),U) \quad \text{s.t.} \quad \textit{nothing} \tag{2}$$

Taking a second order Taylor approximation of the optimization argument leads to:

$$\mathcal{O}(\mathcal{F}(U+\delta U),U+\delta U) \approx \mathcal{O} + K^T \delta U + \frac{1}{2}\delta U^T H \delta U \tag{3}$$

With the following equations for the gradient vector K, and Hessian matrix H:

$$\mathcal{O} = \mathcal{O}(\mathcal{F}(U),U) \tag{4a}$$

$$K = \frac{d\mathcal{O}}{dU}(\mathcal{F}(U),U)^T = \mathcal{O}_U + {\mathcal{F}_U}^T \mathcal{O}_X \tag{4b}$$

$$H = \frac{d^2\mathcal{O}}{dU^2}(\mathcal{F}(U),U) \approx {\mathcal{F}_U}^T \mathcal{O}_{XX} \mathcal{F}_U + {\mathcal{F}_U}^T \mathcal{O}_{XU} + \mathcal{O}_{UX}\mathcal{F}_U + \mathcal{O}_{UU} \tag{4c}$$

For the equation of the Hessian H, an approximation is made by assuming that the terms containing $\mathcal{F}_{UU}$ are comparatively small and therefore can be dropped. This approximation is what distinguishes the Gauss-Newton method from the Newton method.

The Gauss-Newton step is calculated by taking the derivative of this Taylor approximation and by setting it to zero (first order optimality condition).

$$\frac{d}{d\delta U}\mathcal{O}(X(U+\delta U), U+\delta U) = K + H\delta U = 0 \tag{5}$$

$$\delta U = -H^{-1}K \tag{6}$$

At this point, the unknown $\mathcal{F}_U$ present in K and H is the only term that prevents the step, δU, from being evaluated. This problem is resolved with help of the implicit function theorem (7c), by computing $\mathcal{F}_U$ from the derivatives of $\mathcal{G}$ without ever needing to know $\mathcal{F}$.

$$\mathcal{G}(X, U) = 0 \qquad \left| \frac{d}{dU} \right. \tag{7a}$$

$$\frac{\partial \mathcal{G}}{\partial U} \cdot 1 + \frac{\partial \mathcal{G}}{\partial X} \cdot \frac{dX}{\partial U} = 0 \tag{7b}$$

$$\mathcal{F}_U = \frac{\partial X}{\partial U} = -{\mathcal{G}_X}^{-1}\mathcal{G}_U \tag{7c}$$

Since the existence of $\mathcal{F}$ was just a temporary assumption, it makes sense to rename $\mathcal{F}_U$ to S, the *sensitivity* matrix. To summarize, the implementation of the implicit Gauss-Newton method follows this structure:

Data: Cost $\mathcal{O}$, residual $\mathcal{G}$, initial input/state trajectories U, X
Result: Optimal input- and state-trajectories U, X
while *cost $\mathcal{O}$ has not converged* **do**
  Evaluate $\mathcal{O}_X$, $\mathcal{O}_U$, $\mathcal{O}_{XX}$, $\mathcal{O}_{UX}$, $\mathcal{G}_X$, $\mathcal{G}_U$
  Solve for sensitivity matrix S (Eq. 7c)
  Compute K and H (Eq. 4b, 4c)
  Solve for the improvement step δU (Eq. 6)
  Compute the improved input-trajectory $U^+ = U + \delta U$
  Compute the improved state-trajectory X^+ with forward-simulation
end

Algorithm 1: Implicit Gauss-Newton

2.2 Primal-Dual Gauss-Newton

While the implicit Gauss-Newton method avoids the equality constraint $\mathcal{G} = 0$ by using the implicit function theorem, the primal-dual Gauss-Newton retains it and attempts to solve the equality constrained optimization problem directly. This is accomplished by solving the Karush–Kuhn–Tucker (KKT) conditions, which provide an update step for both the input (δU) and the state (δX). These steps try to jointly minimize the cost, $\mathcal{O}$, and to bring the residual, $\mathcal{G}$, to zero.

The KKT conditions can be interpreted as first order optimality conditions on the Lagrangian function $\mathcal{L}(X, U, \Lambda)$, where Λ are the Lagrange multipliers. The Lagrangian function is given by:

$$\mathcal{L}(X, U, \Lambda) = \mathcal{O}(X, U) + \Lambda^T \mathcal{G}(X, U) \tag{8}$$

There are two optimality conditions in this case. First, the primal feasibility condition, $\nabla_\Lambda \mathcal{L}(X, U, \Lambda) = \mathcal{G}(X, U) = 0$, enforces constraint satisfaction. Second, the stationarity conditions, $\nabla_{X,U} \mathcal{L}(X, U, \Lambda) = \mathcal{O}_{X,U}(X, U) + \Lambda^T \mathcal{G}_{X,U}(X, U) = 0$, require that the gradient of the cost be orthogonal to the gradient of the residual. If both conditions are satisfied, that means that there is no (local) change in X, U that keeps satisfying $\mathcal{G} = 0$ while simultaneously lowering the cost $\mathcal{O}$.

Similar to the implicit GN method, the second order Taylor approximation of $\mathcal{L}$ is computed as:

$$\mathcal{L}(X + \delta X, U + \delta U, \Lambda + \delta\Lambda) \approx \mathcal{L} + \nabla\mathcal{L}^T \begin{bmatrix} \delta X \\ \delta U \\ \delta\Lambda \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta X \\ \delta U \\ \delta\Lambda \end{bmatrix}^T \nabla^2\mathcal{L} \begin{bmatrix} \delta X \\ \delta U \\ \delta\Lambda \end{bmatrix} \tag{9}$$

With the following equations for the gradient $\nabla\mathcal{L}$, and for the Hessian $\nabla^2\mathcal{L}$:

$$\mathcal{L} = \mathcal{L}(X, U, \Lambda) = \mathcal{O}(X, U) + \Lambda^T \mathcal{G}(X, U) \tag{10a}$$

$$\nabla\mathcal{L} = \nabla\mathcal{L}(X, U, \Lambda) = \begin{bmatrix} \mathcal{O}_X + {\mathcal{G}_X}^T \Lambda \\ \mathcal{O}_U + {\mathcal{G}_U}^T \Lambda \\ \mathcal{G} \end{bmatrix} \tag{10b}$$

$$\nabla^2\mathcal{L} = \nabla^2\mathcal{L}(X, U, \Lambda) \approx \begin{bmatrix} \mathcal{O}_{XX} & \mathcal{O}_{XU} & {\mathcal{G}_X}^T \\ \mathcal{O}_{UX} & \mathcal{O}_{UU} & {\mathcal{G}_U}^T \\ \mathcal{G}_X & \mathcal{G}_U & 0 \end{bmatrix} \tag{10c}$$

Again the second order derivatives of the residual, $\mathcal{G}$, are dropped because they are usually comparatively small.

The Gauss-Newton step is then calculated by taking the derivatives of this Taylor approximation and setting them to zero, which corresponds to the linearized KKT conditions, also known as the first order optimality conditions.

$$\nabla_{\delta X, \delta U, \delta\Lambda} \mathcal{L}(X + \delta X, U + \delta U, \Lambda + \delta\Lambda) = \nabla\mathcal{L} + \nabla^2\mathcal{L} \begin{bmatrix} \delta X \\ \delta U \\ \delta\Lambda \end{bmatrix} = 0 \tag{11}$$

The resulting system of linear equations becomes the following:

$$\begin{bmatrix} \mathcal{O}_{XX} & \mathcal{O}_{XU} & {\mathcal{G}_X}^T \\ \mathcal{O}_{UX} & \mathcal{O}_{UU} & {\mathcal{G}_U}^T \\ \mathcal{G}_X & \mathcal{G}_U & 0 \end{bmatrix} \begin{bmatrix} \delta X \\ \delta U \\ \delta\Lambda \end{bmatrix} = - \begin{bmatrix} \mathcal{O}_X + {\mathcal{G}_X}^T \Lambda \\ \mathcal{O}_U + {\mathcal{G}_U}^T \Lambda \\ \mathcal{G} \end{bmatrix} \tag{12}$$

A substitution of $\Lambda + \delta\Lambda$ with Λ^+ can be made, simplifying the equation and revealing that the δX, δU steps do not depend on Λ:

$$\begin{bmatrix} \mathcal{O}_{XX} & \mathcal{O}_{XU} & {\mathcal{G}_X}^T \\ \mathcal{O}_{UX} & \mathcal{O}_{UU} & {\mathcal{G}_U}^T \\ \mathcal{G}_X & \mathcal{G}_U & 0 \end{bmatrix} \begin{bmatrix} \delta X \\ \delta U \\ \Lambda^+ \end{bmatrix} = - \begin{bmatrix} \mathcal{O}_X \\ \mathcal{O}_U \\ \mathcal{G} \end{bmatrix} \tag{13}$$

To summarize, the implementation of the primal-dual Gauss-Newton method follows this structure:

Data: Cost $\mathcal{O}$, residual $\mathcal{G}$, initial input/state trajectories U, X
Result: Optimal input/state trajectories U, X
while *cost $\mathcal{O}$ has not converged* **and** *residual $\mathcal{G}$ not zero* **do**
  Evaluate $\mathcal{O}_X$, $\mathcal{O}_U$, $\mathcal{O}_{XX}$, $\mathcal{O}_{UX}$, $\mathcal{G}_X$, $\mathcal{G}_U$
  Solve for the improvement step δU and δX (Eq. 13)
  Compute the improved input-trajectory $U^+ = U + \delta U$
  Compute the improved state-trajectory $X^+ = X + \delta X$
end

Algorithm 2: Primal-Dual Gauss-Newton

2.3 Sparse Gauss-Newton

The sparse Gauss-Newton method computes the exact same step δU as the implicit Gauss-Newton method, but does so with a different approach. It solves a single large, sparse system of linear equations to directly compute the step, δU.

This method can be derived and interpreted in two ways: One is by reformulating the implicit Gauss-Newton method with a few substitutions, and the other is by slightly modifying the primal-dual Gauss-Newton method. Both approaches are presented here, starting with the implicit route.

2.3.1 Sparse from Implicit Gauss-Newton

Transforming the implicit variant of the Gauss-Newton method to the sparse variant requires the following three substitutions:

$$\delta X = S\delta U \tag{14a}$$
$$\delta\Lambda = -{\mathcal{G}_X}^{-T}(\mathcal{O}_{XX}\delta X + \mathcal{O}_{XU}\delta U) \tag{14b}$$
$$\Lambda^+ = -{\mathcal{G}_X}^{-T}\mathcal{O}_X + \delta\Lambda \tag{14c}$$

Consecutively applying them leads to this series of intermediate equations:

$$H\delta U = -K \tag{15a}$$

$$(S^T \mathcal{O}_{XX} S + \mathcal{O}_{UX} S + S^T \mathcal{O}_{XU} + \mathcal{O}_{UU})\delta U = -K \tag{15b}$$

$$\begin{bmatrix} S^T \mathcal{O}_{XX} + \mathcal{O}_{UX} & S^T \mathcal{O}_{XU} + \mathcal{O}_{UU} \\ \mathcal{G}_X & \mathcal{G}_U \end{bmatrix} \begin{bmatrix} \delta X \\ \delta U \end{bmatrix} = - \begin{bmatrix} K \\ 0 \end{bmatrix} \tag{15c}$$

$$\begin{bmatrix} \mathcal{O}_{XX} & \mathcal{O}_{XU} & {\mathcal{G}_X}^T \\ \mathcal{O}_{UX} & \mathcal{O}_{UU} & {\mathcal{G}_U}^T \\ \mathcal{G}_X & \mathcal{G}_U & 0 \end{bmatrix} \begin{bmatrix} \delta X \\ \delta U \\ \delta \Lambda \end{bmatrix} = - \begin{bmatrix} 0 \\ K \\ 0 \end{bmatrix} \tag{15d}$$

$$\begin{bmatrix} \mathcal{O}_{XX} & \mathcal{O}_{XU} & {\mathcal{G}_X}^T \\ \mathcal{O}_{UX} & \mathcal{O}_{UU} & {\mathcal{G}_U}^T \\ \mathcal{G}_X & \mathcal{G}_U & 0 \end{bmatrix} \begin{bmatrix} \delta X \\ \delta U \\ \Lambda^+ \end{bmatrix} = - \begin{bmatrix} \mathcal{O}_X \\ \mathcal{O}_U \\ 0 \end{bmatrix} \tag{15e}$$

Solving (15e) and extracting δU from its solution will yield the same update step as the implicit GN method (δX and Λ^+ are discarded).

In many problems the derivatives of the cost, $\mathcal{O}$, and of the residual, $\mathcal{G}$, have only few non-zero entries, and so the left-hand side matrix in (15e) is generally sparse. This is in contrast to the left-hand side matrix of the implicit step equation (6) which is smaller, but dense. The need to compute the sensitivity matrix S also disappears with the sparse variant.

2.3.2 Sparse from Primal-Dual Gauss-Newton

It can be noticed that the step equation for the sparse GN method (15e) is very similar to the one for the primal-dual GN method (13). In fact, they become identical when considering that, in sparse GN, the residual $\mathcal{G}$ is always zero.

The difference between primal-dual and sparse GN is that only the input step δU is used and that, after every step, an improved state X^+ is computed from the improved input U^+ by forward-simulation, guaranteeing that $\mathcal{G}$ remains zero.

Data: Cost $\mathcal{O}$, residual $\mathcal{G}$, initial input/state trajectories U, X
Result: Optimal input/state trajectories U, X
while *cost $\mathcal{O}$ has not converged* **do**
  Evaluate $\mathcal{O}_X$, $\mathcal{O}_U$, $\mathcal{O}_{XX}$, $\mathcal{O}_{UX}$, $\mathcal{G}_X$, $\mathcal{G}_U$
  Solve for the improvement step δU (Eq. 15e)
  Compute the improved input-trajectory $U^+ = U + \delta U$
  Compute the improved state-trajectory X^+ with forward-simulation
end

Algorithm 3: Sparse Gauss-Newton

3 ILQR

Before Iterative Linear-Quadratic Regulators (ILQR) can be discussed, its limitations must be stated. The implementations that are described in this paper require that the system and cost function be discrete, causal, and able to be formulated as implicit functions.

The general form of a system that meets these requirements is:

$$g_i(\ x_{i-N}, .., x_{i-1},\ x_i,\ x_{i+1},\ u_{i-M}, .., u_i\ ) = 0 \tag{16}$$

Here g_i is the residual at a single timestep, i, unlike $\mathcal{G}$, which is the residual for a whole trajectory. The variables x_i, u_i are the state/input vectors at a timestep i and N, M represent the counts of old states/inputs needed to express the dynamics.

A layer of abstraction must be added to keep the derivations simple. The extended state, x_i^e is introduced as:

$$x_i^e = [\ x_{i-N}, .., x_{i-1},\ x_i,\ u_{i-M}, .., u_{i-1}\ ] \tag{17}$$

This way the residual function can be reformulated such that the time-lag is removed from the system.:

$$g_i^e(\ x_i^e,\ x_{i+1}^e,\ u_i\ ) = 0 \tag{18}$$

Since the size of g_i^e must still match the size of x_i^e, a number of shift equations are appended to g_i to create g_i^e. These are the relations between the overlapping parts of x_i^e and x_{i+1}^e. More information can be found in the hand-written notes at the end of this document (chapter "Structure").

The same notation can be used to describe a general form of the cost function that assigns a cost to the state and input at a single timestep, i:

$$o_i(\ x_{i-N}, .., x_{i-1},\ x_i,\ u_{i-M}, .., u_i\ ) \tag{19}$$

$$o_i(\ x_i^e,\ u_i\ ) \tag{20}$$

One thing to note is that, at the last timestep of the trajectory no input exists, and so o_i can not be a function of u_i for that timestep. That cost function is usually called the *terminal cost* and all other costs are *stage costs*.

With this notation in hand, the class of trajectory-optimization problems which can be solved with ILQR methods can be stated as:

$$\min_{U,X} \quad o_{T+1}(x^e_{T+1}) + \sum_{i=1}^{T} o_i(x^e_i, u_i) \quad \text{s.t.} \quad g^e_i(x^e_i, x^e_{i+1}, u_i) = 0 \quad \forall i \in [1, T] \tag{21}$$

This form of optimization problem is special, because the dynamic programming principle can be applied. It is done by noting that the optimal inputs from a time i onward only depends on the extended state x^e_i. This means that an optimal input for an additional time at $i-1$ can be calculated without the following optimal inputs changing. This recursive form of the optimization problem can be stated like so:

$$V_i(x^e_i) = \left[\quad \min_{u_i} \quad o_i(x^e_i, u_i) + V_{i+1}(x^e_{i+1}) \quad \text{s.t.} \quad g^e_i(x^e_i, x^e_{i+1}, u_i) = 0 \quad \right] \tag{22}$$

Where $V_i(x^e_i)$ is the optimal cost-to-go, i.e. the optimal cost, to go from state x^e_i, to whatever the terminal state x^e_{T+1} is, when continuously applying the optimal inputs.

An important detail to note is that with the formulation put forward in (22) the optimal input u_i is no longer just a vector that is the solution of an optimization problem like with the GN methods. Instead it becomes a sort of input policy, $u_i = \mu_i(x^e_i)$, which depends not only on the timestep, i, but also on the state, x^e_i the system happens to be at at that time.

Throughout the remainder of this chapter, a lighter notation will be used in which i subscripts are dropped, while the $i+1$ subscripts are replaced with a prime symbol and all superscripts are dropped entirely. Equation (22) may then be more compactly represented as:

$$V(x) = \left[\quad \min_{u} \quad o(x, u) + V'(x') \quad \text{s.t.} \quad g(x, x', u) = 0 \quad \right] \tag{23}$$

3.1 Implicit ILQR

Implicit ILQR is, just like implicit Gauss-Newton, based on the implicit function theorem. It is used to turn the constrained problem (23) into an unconstrained one, which can be solved more easily with standard ILQR.

The approach to derive this method is also similar to implicit GN, in that it is assumed that an explicit form of the system dynamics exist as an equation $x' = f(x, u)$. With this assumption, the optimization problem becomes:

$$V(x) = \quad \min_{u} \quad o(x, u) + V'(f(x, u)) \tag{24}$$

Continuing analogously, the second order Taylor approximation of the optimization parameter is taken.

$$Q(x,u) = o(x,u) + V'(f(x,u)) \tag{25}$$

$$Q(x+\delta x, u+\delta u) \approx Q + \begin{bmatrix} Q_x \\ Q_u \end{bmatrix}^T \begin{bmatrix} \delta x \\ \delta u \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta x \\ \delta u \end{bmatrix}^T \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} \delta x \\ \delta u \end{bmatrix} \tag{26}$$

$$\begin{aligned} \text{with} \quad Q_x &= o_x + {f_x}^T V'_{x'} && \text{(27a)} \\ Q_u &= o_u + {f_u}^T V'_{x'} && \text{(27b)} \\ Q_{xx} &\approx o_{xx} + {f_x}^T V'_{x'x'} f_x && \text{(27c)} \\ Q_{uu} &\approx o_{uu} + {f_u}^T V'_{x'x'} f_u && \text{(27d)} \\ Q_{ux} &\approx o_{ux} + {f_u}^T V'_{x'x'} f_x && \text{(27e)} \end{aligned}$$

With this method also, the second order derivatives of the system dynamics f_{xx}, f_{ux}, f_{uu} are dropped because they are usually comparatively small. In the absence of this approximation, this method is known as Differential Dynamic Programming (DDP).

The ILQR descent policy is then found by taking the derivative of (26) with respect to δu and setting it to zero, resulting in the first order optimality condition.

$$\frac{d}{d\delta u} Q(x+\delta x, u+\delta u) = Q_u + Q_{ux}\delta x + Q_{uu}\delta u = 0 \tag{28}$$

$$\delta u = k + K\delta x \tag{29}$$

$$\begin{aligned} \text{with} \quad k &= -Q_{uu}^{-1} Q_u && \text{(30a)} \\ K &= -Q_{uu}^{-1} Q_{ux} && \text{(30b)} \end{aligned}$$

The input policy in (29) describes a change, δu, of the input at time i, which depends on some constant value, k, and a term $K\delta x$. Here δx can be understood as the distance between the state where the Taylor approximation was taken and the state when the input policy is calculated.

To be able to implement this method, an expression to proliferate the optimal cost-to-go derivatives, V_x and V_{xx} must be derived, as they are needed in

equations (27). This is accomplished by inserting the input policy (29) into the cost-to-go approximation (26) resulting in an optimal cost-to-go approximation. Taking the derivatives of that expression gives values for V_x, V_{xx}, which are the same as $V'_{x'}$, $V'_{x'x'}$ for the preceding timestep.

$$\begin{aligned} V(x+\delta x) &= Q(x+\delta x, u+(k+K\delta x)) \\ &= Q + \begin{bmatrix} Q_x \\ Q_u \end{bmatrix}^T \begin{bmatrix} \delta x \\ k+K\delta x \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta x \\ k+K\delta x \end{bmatrix}^T \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} \delta x \\ k+K\delta x \end{bmatrix} \end{aligned} \tag{31}$$

$$V_x = \frac{d}{d\delta x} V(x+\delta x)\bigg|_{\delta x=0} = Q_x - Q_{xu}k \tag{32a}$$

$$V_{xx} = \frac{d^2}{d\delta x^2} V(x+\delta x)\bigg|_{\delta x=0} = Q_{xx} - Q_{xu}K \tag{32b}$$

Up until this point this derivation was the same as for the standard ILQR method. To get the implicit variant, it is necessary to resolve the initial assumption that an explicit formulation $x' = f(x,u)$ exists and to instead replace it with the implicit formulation $g(x', x, u) = 0$. This is done, again, with the implicit function theorem, resulting in the following expressions for f_x, f_u:

$$f_x = \frac{dx'}{dx} = -{g_{x'}}^{-1} g_x \tag{33a}$$

$$f_u = \frac{dx'}{du} = -{g_{x'}}^{-1} g_u \tag{33b}$$

Finally the steps necessary to perform the implicit ILQR method can be summarized as follows:

Data: Costs o, residuals g, initial input/state trajectories
Result: Optimal input/state trajectories
while *total cost has not converged* **do**
  Initialize $V'_{x'} = o_{T+1,x}$ and $V'_{x'x'} = o_{T+1,xx}$
  for $i \leftarrow T$ *to* 1 **do**
    Evaluate o_x, o_{xx}, o_u, o_{uu}, o_{ux}, $g_{x'}$, g_x, g_u at timestep i
    Solve for f_x, f_u (Eq. 33)
    Compute Q_x, Q_{xx}, Q_u, Q_{uu}, Q_{ux} (Eq. 27)
    Solve for k, K (Eq. 30)
    Compute V_x, V_{xx} (Eq. 32)
    Save k, K as k_i, K_i
    Set $V'_{x'} = V_x$, $V'_{x'x'} = V_{xx}$
  end
  Initialize the first improved state $x_1^{e+} = x_1^e$
  for $i \leftarrow 1$ *to* T **do**
    Compute the improved input $u_i^+ = u_i + k_i + K_i(x_i^{e+} - x_i^e)$
    Compute the improved state x_{i+1}^+ with forward-simulation
  end
end

Algorithm 4: Implicit ILQR

3.2 Primal-Dual ILQR

The analogies between Gauss-Newton and ILQR continue with the primal-dual ILQR method. In the same way that primal-dual GN used the KKT conditions on (1) to directly solve the constrained optimization problem, primal-dual ILQR uses the KKT conditions on (23).

The KKT conditions can be seen as the first order optimality conditions on the Lagrangian $\mathcal{L}$ of (23). It takes the following form where λ represents the Lagrange multipliers:

$$\mathcal{L}(x, x', u, \lambda) = o(x, u) + V'(x') + \lambda^T g(x, x', u) \tag{34}$$

A second order Taylor approximation of the Lagrangian may be computed as:

$$\mathcal{L}(x+\delta x, x'+\delta x', u+\delta u, \lambda+\delta\lambda) \approx \mathcal{L} + \nabla\mathcal{L}^T \begin{bmatrix} \delta x \\ \delta x' \\ \delta u \\ \delta\lambda \end{bmatrix} + \begin{bmatrix} \delta x \\ \delta x' \\ \delta u \\ \delta\lambda \end{bmatrix}^T \nabla^2\mathcal{L} \begin{bmatrix} \delta x \\ \delta x' \\ \delta u \\ \delta\lambda \end{bmatrix} \tag{35}$$

$$\text{with} \quad \mathcal{L} = \mathcal{L}(x, x', u, \lambda) \tag{36a}$$

$$\nabla\mathcal{L} = \begin{bmatrix} o_x + {g_x}^T\lambda \\ V'_{x'} + {g_{x'}}^T\lambda \\ o_u + {g_u}^T\lambda \\ g \end{bmatrix} \tag{36b}$$

$$\nabla^2\mathcal{L} \approx \begin{bmatrix} o_{xx} & 0 & o_{xu} & {g_x}^T \\ 0 & V'_{x'x'} & 0 & {g_{x'}}^T \\ o_{ux} & 0 & o_{uu} & {g_u}^T \\ g_x & g_{x'} & g_u & 0 \end{bmatrix} \tag{36c}$$

Once again, the second order terms of the residual function g, which would appear in the Hessian $\nabla^2\mathcal{L}$, are dropped with the reasoning that they are usually comparatively small.

The first order optimality conditions are then used on this Taylor approximation to produce the following improvement step:

$$\begin{bmatrix} o_{xx} & 0 & o_{xu} & {g_x}^T \\ 0 & V'_{x'x'} & 0 & {g_{x'}}^T \\ o_{ux} & 0 & o_{uu} & {g_u}^T \\ g_x & g_{x'} & g_u & 0 \end{bmatrix} \begin{bmatrix} \delta x \\ \delta x' \\ \delta u \\ \delta\lambda \end{bmatrix} + \begin{bmatrix} o_x + {g_x}^T\lambda \\ V'_{x'} + {g_{x'}}^T\lambda \\ o_u + {g_u}^T\lambda \\ g \end{bmatrix} = 0 \tag{37}$$

A substitution of $\lambda + \delta\lambda$ with λ^+ can be made to further simplify the equation. This also shows that the steps δx, $\delta x'$, δu do not depend on λ.

$$\begin{bmatrix} o_{xx} & 0 & o_{xu} & {g_x}^T \\ 0 & V'_{x'x'} & 0 & {g_{x'}}^T \\ o_{ux} & 0 & o_{uu} & {g_u}^T \\ g_x & g_{x'} & g_u & 0 \end{bmatrix} \begin{bmatrix} \delta x \\ \delta x' \\ \delta u \\ \lambda^+ \end{bmatrix} + \begin{bmatrix} o_x \\ V'_{x'} \\ o_u \\ g \end{bmatrix} = 0 \tag{38}$$

From here, a policy to update δu and $\delta x'$ as a function of δx and the current timestep can be derived:

$$\begin{bmatrix} V'_{x'x'} & 0 & {g_{x'}}^T \\ 0 & o_{uu} & {g_u}^T \\ g_{x'} & g_u & 0 \end{bmatrix} \begin{bmatrix} \delta x' \\ \delta u \\ \lambda^+ \end{bmatrix} = - \begin{bmatrix} V'_{x'} \\ o_u \\ g \end{bmatrix} - \begin{bmatrix} 0 \\ o_{ux} \\ g_x \end{bmatrix} \delta x \tag{39}$$

$$\begin{bmatrix} \delta x' \\ \delta u \\ \lambda^+ \end{bmatrix} = \begin{bmatrix} k_{x'} \\ k_u \\ k_{\lambda^+} \end{bmatrix} + \begin{bmatrix} K_{x'} \\ K_u \\ K_{\lambda^+} \end{bmatrix} \delta x \tag{40}$$

$$\text{with} \quad \begin{bmatrix} k_{x'} \\ k_u \\ k_{\lambda^+} \end{bmatrix} = - \begin{bmatrix} V'_{x'x'} & 0 & {g_{x'}}^T \\ 0 & o_{uu} & {g_u}^T \\ g_{x'} & g_u & 0 \end{bmatrix}^{-1} \begin{bmatrix} V'_{x'} \\ o_u \\ g \end{bmatrix} \tag{41a}$$

$$\begin{bmatrix} K_{x'} \\ K_u \\ K_{\lambda^+} \end{bmatrix} = - \begin{bmatrix} V'_{x'x'} & 0 & {g_{x'}}^T \\ 0 & o_{uu} & {g_u}^T \\ g_{x'} & g_u & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ o_{ux} \\ g_x \end{bmatrix} \tag{41b}$$

The optimal cost-to-go derivatives for the current timestep V_x and V_{xx} are calculated by inserting the policies into the Lagrangian and by taking derivatives with respect to δx, in an approach similar to the one taken for implicit ILQR. These are needed because they reappear as $V'_{x'}$ and $V'_{x'x'}$ in the preceding timestep.

$$V(x + \delta x) = \mathcal{L}(x + \delta x, x' + k_{x'} + K_{x'}\delta x, u + k_u + K_u \delta x, k_{\lambda^+} + K_{\lambda^+}\delta x) \tag{42}$$

$$V_x = \begin{bmatrix} I \\ K_{x'} \\ K_u \\ K_{\lambda^+} \end{bmatrix}^T \left(\begin{bmatrix} o_x \\ V'_{x'} \\ o_u \\ g \end{bmatrix} + \begin{bmatrix} o_{xx} & 0 & o_{xu} & {g_x}^T \\ 0 & V'_{x'x'} & 0 & {g_{x'}}^T \\ o_{ux} & 0 & o_{uu} & {g_u}^T \\ g_x & g_{x'} & g_u & 0 \end{bmatrix}^T \begin{bmatrix} 0 \\ k_{x'} \\ k_u \\ k_{\lambda^+} \end{bmatrix} \right) \tag{43a}$$

$$V_{xx} = \begin{bmatrix} I \\ K_{x'} \\ K_u \\ K_{\lambda^+} \end{bmatrix}^T \begin{bmatrix} o_{xx} & 0 & o_{xu} & {g_x}^T \\ 0 & V'_{x'x'} & 0 & {g_{x'}}^T \\ o_{ux} & 0 & o_{uu} & {g_u}^T \\ g_x & g_{x'} & g_u & 0 \end{bmatrix}^T \begin{bmatrix} I \\ K_{x'} \\ K_u \\ K_{\lambda^+} \end{bmatrix} \tag{43b}$$

At this point everything needed to implement the primal-dual ILQR method is present. One thing to note about the update policy $\delta x' = k_{x'} + K_{x'}\delta x$ is that x' is the extended state x^e_{i+1}. This means that it not only contains a policy for the next state x_{i+1}, but also for a few past states and past inputs. This would suggest that for every timestep an entire extended state must be held in memory, but this is not the case. It can be shown that the policies for these old states and inputs perfectly mirror the shift equations, so that only the policy for x_{i+1} has to be used. This also means that these known parts of the policy can be inserted into (41) to make the system of linear equations smaller. The proof of this statement and the resulting policy equations can be found in the hand-written notes at the end of this document.

Putting everything together, the primal-dual ILQR method can be implemented following these steps:

Data: Costs o, residuals g, initial input/state trajectories
Result: Optimal input/state trajectories
while *total cost has not converged* **and** *residuals not zero* **do**
  Initialize $V'_{x'} = o_{T+1,x}$ and $V'_{x'x'} = o_{T+1,xx}$
  for $i \leftarrow T$ *to* 1 **do**
    Evaluate o_x, o_{xx}, o_u, o_{uu}, o_{ux}, $g_{x'}$, g_x, g_u at timestep i
    Solve for the input/state policies (Eq. 41)
    Compute V_x, V_{xx} (Eq. 43)
    Save k_u, K_u as $k_{u,i}$, $K_{u,i}$
    Save the part of $k_{x'}$, $K_{x'}$ concerning x_{i+1} as $k_{x,i}$, $K_{x,i}$
    Set $V'_{x'} = V_x$, $V'_{x'x'} = V_{xx}$
  end
  Initialize the first improved state $x_1^{e+} = x_1^e$
  for $i \leftarrow 1$ *to* T **do**
    Compute the improved input $u_i^+ = u_i + k_{u,i} + K_{u,i}(x_i^{e+} - x_i^e)$
    Compute the improved state $x_{i+1}^+ = x_{i+1} + k_{x,i} + K_{x,i}(x_i^{e+} - x_i^e)$
  end
end

Algorithm 5: Primal-Dual ILQR

3.3 Sparse ILQR

The last method that will be discussed is a sparse variant of ILQR. This variant produces the same policy $\delta u = k + K\delta x$ as implicit ILQR, but does so by solving just one large, sparse system of linear equations instead of two. In this regard, the relation between sparse and implicit ILQR is very similar to the relation between sparse and implicit GN.

Similarly there are two ways to interpret and derive this method. One is by reformulating the implicit ILQR method with a few substitutions, and the other is by slightly modifying the primal-dual ILQR method. Both are presented here, starting with the implicit route.

3.3.1 Sparse from Implicit ILQR

Transforming the implicit variant of the ILQR method to the sparse variant requires these two substitutions:

$$\delta x' = f_u \delta u + f_x \delta x \tag{44a}$$

$$\lambda^+ = {g_u}^{-T}({f_u}^T V'_{x'x'} \delta x' + {f_u}^T V'_{x'}) \tag{44b}$$

Applying them consecutively leads to a series of intermediate equations given by:

$$Q_{uu}\delta u + Q_{ux}\delta x + Q_u = 0 \tag{45a}$$

$$\begin{bmatrix} o_{ux} + {f_u}^T V'_{x'x'} f_x & o_{uu} + {f_u}^T V'_{x'x'} f_u \end{bmatrix} \begin{bmatrix} \delta x \\ \delta u \end{bmatrix} + (o_u + {f_u}^T V'_{x'}) = 0 \tag{45b}$$

$$\begin{bmatrix} o_{ux} & {f_u}^T V'_{x'x'} & o_{uu} \\ g_x & g_{x'} & g_u \end{bmatrix} \begin{bmatrix} \delta x \\ \delta x' \\ \delta u \end{bmatrix} + \begin{bmatrix} o_u + {f_u}^T V'_{x'} \\ 0 \end{bmatrix} = 0 \tag{45c}$$

$$\begin{bmatrix} 0 & V'_{x'x'} & 0 & g_{x'}^T \\ o_{ux} & 0 & o_{uu} & g_u^T \\ g_x & g_{x'} & g_u & 0 \end{bmatrix} \begin{bmatrix} \delta x \\ \delta x' \\ \delta u \\ \lambda^+ \end{bmatrix} + \begin{bmatrix} V'_{x'} \\ o_u \\ 0 \end{bmatrix} = 0 \tag{45d}$$

Rearranging this equation leads to the sparse ILQR policy form:

$$\begin{bmatrix} V'_{x'x'} & 0 & {g_{x'}}^T \\ 0 & o_{uu} & {g_u}^T \\ g_{x'} & g_u & 0 \end{bmatrix} \begin{bmatrix} \delta x' \\ \delta u \\ \lambda^+ \end{bmatrix} = - \begin{bmatrix} V'_{x'} \\ o_u \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ o_{ux} \\ g_x \end{bmatrix} \delta x \tag{46}$$

$$\begin{bmatrix} \delta x' \\ \delta u \\ \lambda^+ \end{bmatrix} = \begin{bmatrix} k_{x'} \\ k \\ k_{\lambda+} \end{bmatrix} + \begin{bmatrix} K_{x'} \\ K \\ K_{\lambda+} \end{bmatrix} \tag{47}$$

$$\text{with} \quad \begin{bmatrix} k_{x'} \\ k \\ k_{\lambda+} \end{bmatrix} = - \begin{bmatrix} V'_{x'x'} & 0 & {g_{x'}}^T \\ 0 & o_{uu} & {g_u}^T \\ g_{x'} & g_u & 0 \end{bmatrix}^{-1} \begin{bmatrix} V'_{x'} \\ o_u \\ 0 \end{bmatrix} \tag{48a}$$

$$\begin{bmatrix} K_{x'} \\ K \\ K_{\lambda+} \end{bmatrix} = - \begin{bmatrix} V'_{x'x'} & 0 & {g_{x'}}^T \\ 0 & o_{uu} & {g_u}^T \\ g_{x'} & g_u & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ o_{ux} \\ g_x \end{bmatrix} \tag{48b}$$

The path to get from the implicit ILQR form (45a) to the sparse form (46) only involved substitutions and rearrangements, meaning that the policy $\delta u = k + K\delta x$ remains the same. The difference is that large parts of the systems of linear equations in (48) are usually sparse and that the sensitivities f_u and f_x no longer appear. The trade-off for this sparsity is a larger system of equations.

3.3.2 Sparse from Primal-Dual ILQR

It can be noticed that the policy equations for the sparse ILQR method (48) are very similar to the ones for the primal-dual ILQR method (41). In fact they become identical when considering that, for sparse ILQR, the residual $\mathcal{G}$ is always zero. A very similar observation was made for sparse GN.

The difference between primal-dual and sparse ILQR is that only the input policy, $u_i^+ = u_i + k_{u,i} + K_{u,i}(x_i^{e+} - x_i^e)$, is used and the state x_{i+1}^+ is instead computed from u_i^+ and x_i^{e+} by forward-simulation, guaranteeing that all g_i stays zero.

Because of this relation it is easily justifiable, that sparse ILQR should use the same equations to proliferate the cost-to-go derivatives V_x, V_{xx} as primal-dual ILQR (43).

Data: Costs o, residuals g, initial input/state trajectories
Result: Optimal input/state trajectories
while *total cost has not converged* **do**
  Initialize $V'_{x'} = o_{T+1,x}$ and $V'_{x'x'} = o_{T+1,xx}$
  for $i \leftarrow T$ *to* 1 **do**
    Evaluate o_x, o_{xx}, o_u, o_{uu}, o_{ux}, $g_{x'}$, g_x, g_u at timestep i
    Solve for the input policy (Eq. 48)
    Compute V_x, V_{xx} (Eq. 43)
    Save k, K as k_i, K_i
    Set $V'_{x'} = V_x$, $V'_{x'x'} = V_{xx}$
  end
  Initialize the first improved state $x_1^{e+} = x_1^e$
  for $i \leftarrow 1$ *to* T **do**
    Compute the improved input $u_i^+ = u_i + k_i + K_i(x_i^{e+} - x_i^e)$
    Compute the improved state x_{i+1}^+ with forward-simulation
  end
end

Algorithm 6: Sparse ILQR

4 Hand-Written Notes

This section contains the authors hand-written notes. They cover the same information that was presented above in a more compact form. Additionally they also cover the following topics:

- Details on strategies for improving convergence rates of the TO methods, particularly line-search and regularization strategies.
- Derivation of the compacted policy equations for sparse and primal-dual ILQR (containing the shift equations).
- Information on forward-simulation methods, in particular on the Newton method.
- Sample models that were used for testing of the various TO methods.
- Graphical explanation on how to compute and assemble the various system and cost derivatives mentioned in the above paper.
- Various open questions written in red.

trajectory optimization

implicit Gauss-Newton (remove equality constraint using implicit function theorem)

$\min_U \mathcal{O}(X,U)$ subj.to: $g(X,U)=0$ $\quad (X=\{x_0,\dots,x_n\}\ U=\{u_0,\dots,u_{n-1}\})$

[Taylor expansion assuming $X=X(U)$: $\mathcal{O}(U+\delta U)\approx \mathcal{O}(U)+d\mathcal{O}/dU\,\delta U+\frac{1}{2}\delta U^T d^2\mathcal{O}/dU^2\,\delta U$
first order optimality condition: $\hookrightarrow d\mathcal{O}/d\delta U = d\mathcal{O}/dU + d^2\mathcal{O}/dU^2\,\delta U = 0 \rightarrow \delta U=-(d^2\mathcal{O}/dU^2)^{-1}d\mathcal{O}/dU$]

$K=\frac{d}{dU}\mathcal{O}^T=\dots=\mathcal{O}_u+S^T\mathcal{O}_x$ $\qquad S=\frac{dX}{dU}=-g_x^{-1}g_u$: sensitivity matrix

$H=\frac{d^2}{dU^2}\mathcal{O}=\dots=S^T\mathcal{O}_{xx}S+S^T\mathcal{O}_{xu}+\mathcal{O}_{ux}S+\mathcal{O}_{uu}$ drop S_x, S_u terms as they are usually small. (Gauss)

$\hookrightarrow \delta U=-H^{-1}\cdot K \rightarrow U^+=U+\delta U \rightarrow X^+$ from solving $g(X^+,U^+)=0$ (forward simulation)

> initialize inputs, e.g. $U=0$
> while $\mathcal{O}$ is decreasing
> | compute X from $g(X,U)=0$
> | compute $K(X,U)$; $H(X,U)$
> | compute $d=-H^{-1}K$
> | update new U to $U+d$

- **implementation details:**

  - **line search**: $U^+=U+\alpha\delta U$ with α such that $\mathcal{O}(X^+,U^+)<\mathcal{O}(X,U)$ (start at $\alpha=1$, decrease iteratively)
    ↳ ensures that each iteration improves guess U by allowing $\mathcal{O}$ to only sink → will find a minimum
    alternative: require $\mathcal{O}(X^+,U^+)<\mathcal{O}(X,U)+c_1\alpha K^T\delta U$, $c_1\in[0,1]$ → expected $\mathcal{O}$ decrease (Armijo)

  - **regularization**: $H\leftarrow H+rI$ with r such that $K^T\delta U<0$ (increase r iteratively)
    ↳ $\mathcal{O}(U+\alpha\delta U)<\mathcal{O}(U)$ for α "small" → $\frac{d}{d\alpha}\mathcal{O}(U+\alpha\delta U)|_{\alpha=0}=K^T\delta U=-K^THK<0$ → requires $H>0$
    ↳ $\|(H+rI)^{-1}K\|<\|H^{-1}K\|\ \forall H>0, r>0$ → makes step shorter if $H>0$
    ↳ limit case r "large" → $H\approx rI$ → $\delta U\approx-\frac{1}{r}K$ → gradient descent with tiny step lenght
    alternative: selectively increase $EW(H)<0$ such that $H>0$ (possibly not worth the effort)

  - **stopping cond.**: stop if $|\mathcal{O}-\mathcal{O}^+|/\mathcal{O}<tol\approx10^{-3}$ (stop when $\mathcal{O}$ decrease was less than 0.1%)
    alternatives: $|U-U^+|/U<tol$ (stop on small input change) or $\|K\|<tol$ (stop on gradient=0)

sparse Gauss-Newton (reformulation of implicit method that avoids the dense sensitivity matrix S)

$$(S^T C_{xx} S + C_{ux} S + S^T C_{xu} + C_{uu})\,\delta u = -K$$

$$\Leftrightarrow \begin{bmatrix} S^T C_{xx} + C_{ux} & S^T C_{xu} + C_{uu} \\ g_x & g_u \end{bmatrix} \begin{bmatrix} \delta x \\ \delta u \end{bmatrix} = \begin{bmatrix} -K \\ 0 \end{bmatrix} \quad \leftarrow \text{ substit: } \delta x = S\delta u \leftrightarrow g_x \delta x + g_u \delta u = 0$$

$$\Leftrightarrow \begin{bmatrix} C_{xx} & C_{xu} & g_x^T \\ C_{ux} & C_{uu} & g_u^T \\ g_x & g_u & 0 \end{bmatrix} \begin{bmatrix} \delta x \\ \delta u \\ \delta \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ -K \\ 0 \end{bmatrix} \quad \leftarrow \text{ substit: } \delta\lambda = -g_x^{-T}(C_{xx}\delta x + C_{xu}\delta u) \leftrightarrow \begin{array}{l} S^T C_{xx}\delta x + S^T C_{xu}\delta u = g_u^T \delta\lambda \\ C_{xx}\delta x + C_{xu}\delta u + g_x^T \delta\lambda = 0 \end{array}$$

$$\Leftrightarrow \begin{bmatrix} C_{xx} & C_{xu} & g_x^T \\ C_{ux} & C_{uu} & g_u^T \\ g_x & g_u & 0 \end{bmatrix} \begin{bmatrix} \delta x \\ \delta u \\ \lambda^+ \end{bmatrix} = \begin{bmatrix} -c_x \\ -c_u \\ 0 \end{bmatrix} \quad \leftarrow \text{ substit: } \lambda^+ = -g_x^{-T} c_x + \delta\lambda \leftrightarrow \delta\lambda = \lambda^+ + g_x^{-T} c_x$$

↳ does not contain S! this system to compute δu is larger, but sparse!

- **implementation details:**
  - **line search:** same as non-sparse Gauss-Newton
  - **regularization:** $C_{uu} \leftarrow C_{uu} + rI$ with r such that **linesearch does not fail** (increase r iteratively)
    - ↳ same effect as non-sparse: $S^T C_{xx} S + C_{ux} S + S^T C_{xu} + (C_{uu} + rI) = H + rI$
    - **alternative:** still compute K to use condition $K^T \delta u < 0$ to increase r
  - **stopping cond.:** same as non-sparse Gauss-Newton

primal-dual Gauss-Newton (use KKT conditions to solve for cost and constraint simultaneously)

dual problem: $\underset{\Lambda}{\text{extr}}\ \underset{U,X}{\min}\ \mathcal{O}(X,U)+\Lambda^T g(X,U)$ $\longleftrightarrow$ primal problem: $\underset{U,X}{\min}\ \mathcal{O}(X,U)$ subj.to: $g(X,U)=0$

$\hookrightarrow Z=\begin{bmatrix}X\\U\end{bmatrix} \longrightarrow \underset{\Lambda}{\text{extr}}\ \underset{Z}{\min}\ L(Z,\Lambda)$

$=L(X,U,\Lambda)$ lagrangian function

$=d(\Lambda)$ lagrange dual function

(Λ: lagrange multipliers)

$$\nabla_z L=\mathcal{O}_z+g_z^T\Lambda=\begin{bmatrix}\mathcal{O}_x+g_x^T\Lambda\\ \mathcal{O}_u+g_u^T\Lambda\end{bmatrix} \qquad \nabla_{zz}^2 L=\mathcal{O}_{zz}+g_{zz}:\Lambda=\begin{bmatrix}\mathcal{O}_{xx}+g_{xx}:\Lambda & \mathcal{O}_{xu}+g_{xu}:\Lambda\\ \mathcal{O}_{ux}+g_{ux}:\Lambda & \mathcal{O}_{uu}+g_{uu}:\Lambda\end{bmatrix} \qquad \nabla_{z\Lambda}^2 L=g_z^T=\begin{bmatrix}g_x^T\\ g_u^T\end{bmatrix} \qquad \nabla_\Lambda L=g \qquad \nabla_{\Lambda\Lambda}^2 L=0$$

notation remark: $\nabla_a g_b^T\Lambda=\nabla_a\left(\sum_i \Lambda_i g_{b,i}\right)=\sum_i \Lambda_i g_{ba,i}=g_{ba}:\Lambda$

KKT conditions (solve dual problem!):

1) primal feasibility $\nabla_\Lambda L(Z^*,\Lambda^*)=g(Z^*)=0$

2) stationarity $\nabla_z L(Z^*,\Lambda^*)=\mathcal{O}_z(Z^*)+\Lambda^{*T}g_z(Z^*)=0$

$$\nabla L(Z,\Lambda)=\begin{bmatrix}\nabla_z L\\ \nabla_\Lambda L=g\end{bmatrix}=0 \rightarrow \nabla L(Z+\delta Z,\Lambda+\delta\Lambda)=\begin{bmatrix}\nabla_z L\\ g\end{bmatrix}+\begin{bmatrix}\nabla_{zz}L & g_z^T\\ g_z & 0\end{bmatrix}\begin{bmatrix}\delta Z\\ \delta\Lambda\end{bmatrix}=0 \rightarrow \begin{bmatrix}\nabla_{zz}L & g_z^T\\ g_z & 0\end{bmatrix}\begin{bmatrix}\delta Z\\ \delta\Lambda\end{bmatrix}=-\begin{bmatrix}\nabla_z L\\ g\end{bmatrix}$$

(linearization)

$\rightarrow$ substit. $\Lambda^+=\Lambda+\delta\Lambda$: $\begin{bmatrix}\nabla_{zz}^2 L & g_z^T\\ g_z & 0\end{bmatrix}\begin{bmatrix}\delta Z\\ \Lambda^+\end{bmatrix}=-\begin{bmatrix}\mathcal{O}_z\\ g\end{bmatrix}$ does not contain Λ in right hand side!

$\hookrightarrow$ solve $\begin{bmatrix}\mathcal{O}_{xx}+g_{xx}:\Lambda & \mathcal{O}_{xu}+g_{xu}:\Lambda & g_x^T\\ \mathcal{O}_{ux}+g_{ux}:\Lambda & \mathcal{O}_{uu}+g_{uu}:\Lambda & g_u^T\\ g_x & g_u & 0\end{bmatrix}\begin{bmatrix}\delta X\\ \delta U\\ \Lambda^+\end{bmatrix}=-\begin{bmatrix}\mathcal{O}_x\\ \mathcal{O}_u\\ g\end{bmatrix}$ (Newton method)

(SQP (sequential quadratic programming): $\underset{\delta Z}{\min}\ \frac{1}{2}\delta Z^T\nabla_{zz}^2 L\,\delta Z+\mathcal{O}_z^T\delta Z$ subj.to: $g_z^T\delta Z+g=0$

$\hookrightarrow$ same δZ as in original nonlinear problem when applying KKT on above QP! (Λ^+ as lag.mult.))

- **Gauss approximation** (drop second order of constraint $g_{zz}\approx 0$)

$\hookrightarrow \begin{bmatrix}\mathcal{O}_{xx} & \mathcal{O}_{xu} & g_x^T\\ \mathcal{O}_{ux} & \mathcal{O}_{uu} & g_u^T\\ g_x & g_u & 0\end{bmatrix}\begin{bmatrix}\delta X\\ \delta U\\ \Lambda^+\end{bmatrix}=-\begin{bmatrix}\mathcal{O}_x\\ \mathcal{O}_u\\ g\end{bmatrix}$ $\rightarrow$ does not contain Λ at all! $\rightarrow$ no need to keep track of it.

(Gauss SQP: $\underset{\delta Z}{\min}\ \frac{1}{2}\delta Z^T\mathcal{O}_{zz}\delta Z+\mathcal{O}_z^T\delta Z$ subj.to: $g_z^T\delta Z+g=0$ (Λ^+ as lag.mult.))

- **relation to sparse Gauss-Newton:**

1) get U from last iteration and compute X trough forward simulation $\rightarrow g(X,U)=0$

$\hookrightarrow$ dual Gauss-Newton system: $\begin{bmatrix}\mathcal{O}_{xx} & \mathcal{O}_{xu} & g_x^T\\ \mathcal{O}_{ux} & \mathcal{O}_{uu} & g_u^T\\ g_x & g_u & 0\end{bmatrix}\begin{bmatrix}\delta X\\ \delta U\\ \Lambda^+\end{bmatrix}=-\begin{bmatrix}\mathcal{O}_x\\ \mathcal{O}_u\\ g^{=0}\end{bmatrix}$ $\rightarrow$ same as in sparse Gauss-Newton!

2) use $U^+=U+\delta U$ step, but ignore δX, Λ^+ $\rightarrow$ repeat step ① with new U^+

- **implementation details:**
  - BOX (only limit step size and residuals)
    - *line search:* $X^+ = X + \alpha \delta X$, $U^+ = U + \alpha \delta U$ decrease α until $\alpha \delta U < \delta U_{MAX}$ and $|g(X^+, U^+)| < g_{MAX}$
    - *regularization:* none or same as sparse gauss-newton
    - *stopping cond.:* stop if $|\mathcal{O} - \mathcal{O}^+| / \mathcal{O} < tol \approx 10^{-3}$ and $|g(X^+, U^+)| < tol \approx 10^{-8}$

    (reset if linesearch hits g_{max} twice in a row or if $\mathcal{O}$ tollerance is reached, but not g)
  - GRAD (use gradient ∇L to determine improvement) (could get stuck at wrong extrema!)
    - *line search:* $X^+ = X + \alpha \delta X$, $U^+ = U + \alpha \delta U$, $\Lambda^+ = \Lambda + \alpha \delta \Lambda$ decrease α until $\|\nabla L(X^+, U^+, \Lambda^+)\| < \|\nabla L(X, U, \Lambda)\|$

      $\frac{d}{d\alpha} \|\nabla L(y + \alpha \delta y)\|\big|_{\alpha=0} = \frac{\nabla L(y+\alpha\delta y)^T (\nabla^2 L(y+\alpha\delta y) \cdot \delta y)}{\|\nabla L(y+\alpha\delta y)\|}\Big|_{\alpha=0} = \frac{\nabla L(y)^T (\nabla^2 L(y)\, \delta y)}{\|\nabla L(y)\|} = \frac{-\nabla L(y)^T \nabla L(y)}{\|\nabla L(y)\|} = -\|\nabla L(y)\| \leq 0$

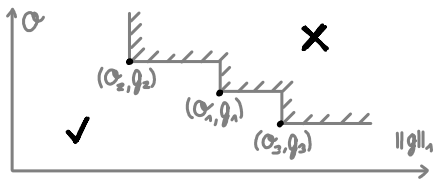
      $\uparrow (y = [X^T\ U^T\ \Lambda^T]^T)$

      alternative: $\|\nabla L^+\| < \|\nabla L\| + c_1 \alpha\, \nabla L^T \nabla^2 L / \|\nabla L\|$, $c_1 \in [0,1)$ → expected $\|\nabla L\|$ decrease (Armijo)
    - *regularization:* not needed. (method performs poorly if $\nabla^2 L$ is close to singular)

      alternative: $\nabla^2 L^{-1} \leftarrow (\nabla^2 L^T \nabla^2 L + rI)^{-1} \nabla^2 L$ (damped inverse, avoids singularity when $r \uparrow$)
    - *stopping cond.:* stop if $\|\nabla L(X^+, U^+, \Lambda^+)\| < tol \approx 10^{-8}$

      alternatives: $|\,\|\nabla L(X,U,\Lambda)\| - \|\nabla L(X^+, U^+, \Lambda^+)\|\,| / \|\nabla L(X,U,\Lambda)\| < tol \approx 10^{-3}$ or same as in BOX
    - *note:* performs badly if Λ is updated to minimize $\|\nabla L\|$ in a reset (makes progress cond. very hard)
  - FILTER
    - *line search:* $X^+ = X + \alpha \delta X$, $U^+ = U + \alpha \delta U$ decrease α until $\mathcal{O}^+ < \mathcal{O}_k$ OR $\|g^+\|_1 < \|g_k\|_1$ $\forall k$ = prev. iter.

      alternatives: use different norm for g. e.g.: $\|g\|_2^2$, $\|g\|_\infty$
    - *regularization / stopping cond.:* same as in BOX

    ($\mathcal{O}_k, g_k$ values at prev. iterations k)
  - PENALTY: (use a penalty function to determine improvement)
    - *line search:* $X^+ = X + \alpha \delta X$, $U^+ = U + \alpha \delta U$ decrease α until $\phi(\mathcal{O}^+; g^+; \mu) \leq \phi(\mathcal{O}; g; \mu)$

      penalty func.: $\phi = \mathcal{O} + \mu \|g\|_1$; $\mu > 0$ (set to 0 at start)

      expected change: $\delta\mathcal{O} = \mathcal{O}_z^T \delta z$; $\delta g = g_z^T \delta z = -g$

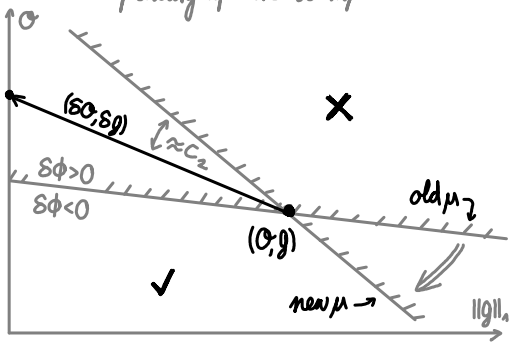
      ↳ $\delta\phi = \mathcal{O}_z^T \delta z - \mu \|g\|_1$

      penalty update: if $\delta\phi > 0$ → $\mu = c_2 \cdot \mathcal{O}_z^T \delta z / \|g\|_1$

      alternatives: $\phi(\mathcal{O}^+; g^+; \mu) \leq \phi(\mathcal{O}; g; \mu) + c_1 \alpha \delta\phi$ (Armijo)

      use different norm for g. e.g.: $\|g\|_2^2$, $\|g\|_\infty$
    - *regularization / stopping cond.:* same as in BOX

    penalty update example:

implicit ILQR (iterative linear-quadratic regulator) (remove equality constraint using implicit function theorem)

$\min_U o_f(x_{T+1}) + \sum_{i=1}^{T} o(x_i, u_i)$ subj. to: $x_{i+1} = f(x_i, u_i)\ \forall i$ trajectory optimization problem

$J_i(x_i, U_i) = o_f(x_n) + \sum_{j=i}^{T} o(x_j, u_j)$ cost to go from time i to $T+1$ with inputs $U_i = \{u_i, \dots, u_T\}$ ←

↳ $J_i(x_i, U_i) = o(x_i, u_i) + J_{i+1}(x_{i+1}, U_{i+1})$ recursive form of cost to go (states from $i+1$ to $T+1$ follow from $x_{i+1} = f(x_i, u_i)$)

$V_i(x_i) = \min_{U_i} J_i(x_i, U_i)$ optimal cost to go from time i to $T+1$

↳ $V_i(x_i) = \min_{u_i} [o(x_i, u_i) + V_{i+1}(f(x_i, u_i))]$ recursive form of optimal cost to go

$V(x) = \min_u [o(x,u) + V'(f(x,u))]$ drop indices for simplicity $x = x_i$, $u = u_i$, $V' = V_{i+1}$

$Q(x,u) = o(x,u) + V'(f(x,u))$ optimization argument

$Q(x+\delta x, u+\delta u) = Q + \begin{pmatrix} Q_x \\ Q_u \end{pmatrix}^T \begin{pmatrix} \delta x \\ \delta u \end{pmatrix} + \frac{1}{2} \begin{pmatrix} \delta x \\ \delta u \end{pmatrix}^T \begin{pmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{pmatrix} \begin{pmatrix} \delta x \\ \delta u \end{pmatrix}$ second order Taylor expansion of $Q(x,u)$

$Q_x = o_x + f_x^T V'_{x'}$; $Q_u = o_u + f_u^T V'_{x'}$; $Q_{xx} = o_{xx} + f_x^T V'_{x'x'} f_x + \cancel{V'_{x'} f_{xx}}$; $Q_{uu} = o_{uu} + f_u^T V'_{x'x'} f_u + \cancel{V'_{x'} f_{uu}}$; $Q_{ux} = o_{ux} + f_u^T V'_{x'x'} f_x + \cancel{V'_{x'} f_{ux}}$

f_{xx}, f_{ux}, f_{xx} are dropped for the same reason as in Gauss-Newton. difference between ILQR and DDP.

$0 = \frac{d}{d\delta u} Q(x+\delta x, u+\delta u) = Q_u + Q_{ux}\delta x + Q_{uu}\delta u$ set derivate to 0, to get an expression that lowers $V(x+\delta x)$

↳ $\delta u = -Q_{uu}^{-1} Q_u - Q_{uu}^{-1} Q_{ux} \delta x = k + K\delta x$ { rule on how to update $u^+ = u + \delta u$ to lower $V(x+\delta x)$, where δx is the offset to x, where Q was approximated

($k = -Q_{uu}^{-1} Q_u$; $K = -Q_{uu}^{-1} Q_{ux}$)

$V(x+\delta x) = Q(x+\delta x, u + (k + K\delta x))$ assume above strategy minimizes Q

$= Q + \begin{pmatrix} Q_x \\ Q_u \end{pmatrix}^T \begin{pmatrix} \delta x \\ k + K\delta x \end{pmatrix} + \frac{1}{2} \begin{pmatrix} \delta x \\ k + K\delta x \end{pmatrix}^T \begin{pmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{pmatrix} \begin{pmatrix} \delta x \\ k + K\delta x \end{pmatrix}$

$V_x(x) = \frac{d}{d\delta x} V(x+\delta x)|_{\delta x = 0} = Q_x + K^T Q_u + Q_{xu} k + K^T Q_{uu} k = Q_x - Q_{xu} Q_{uu}^{-1} Q_u$

$V_{xx}(x) = \frac{d^2}{d\delta x^2} V(x+\delta x)|_{\delta x = 0} = Q_{xx} + Q_{xu} K + K^T Q_{ux} + K^T Q_{uu} K = Q_{xx} - Q_{xu} Q_{uu}^{-1} Q_{ux}$

(formulas for V_x, V_{xx} that can be used as $V'_{x'}, V'_{x'x'}$ in a next step)

- **extended ILQR** (allow for past states/inputs)

replace states x_i with $x_i^e = [x_{i-N+1}, \dots, x_i, u_{i-N+1}, \dots, u_{i-1}]$

→ add equations to f resp. g to pass along unchanged parts of x_i^e, to x_{i+1}^e

$\min_U o_f(x_{T+1}^e) + \sum_{i=1}^{T} o(x_i^e, u_i)$ subj. to: $x_{i+1}^e = f(x_i^e, u_i)$

- **ILQR with implicit constraint**

adapt extended DDP/ILQR to implicit form:

$\min_U o_f(x_{T+1}^e) + \sum_{i=1}^{T} o(x_i^e, u_i)$ subj. to: $g(x_{i+1}^e, x_i^e, u_i) = 0$

requires f_x, f_u to work, which can be expr. by g:

↳ $f_x = \partial x_{i+1}^e / \partial x_i^e = -\left(\partial g / \partial x_{i+1}^e\right)^{-1} \cdot \partial g / \partial x_i^e$

↳ $f_u = \partial x_{i+1}^e / \partial u_i = -\left(\partial g / \partial x_{i+1}^e\right)^{-1} \cdot \partial g / \partial u_i$

initialize inputs, e.g. $U = 0$
compute X with $g(x_{i+1}^e, x_i^e, u_i) = 0\ \forall i$
while O is decreasing
  set $V_{T+1} = o_f(x_{T+1}^e)$; compute V_x V_{xx}
  for $i = T$ to 1 (backward pass)
    compute $Q_{i,x}$ $Q_{i,xx}$ $Q_{i,u}$ $Q_{i,uu}$ $Q_{i,ux}$
    compute k_i and K_i
    compute derivatives of V_i
  for $i = 1$ to T (forward pass)
    compute new $u_i^+ = u_i + k_i + K_i(x_i^{e+} - x_i^e)$
    simulate new x_{i+1}^{e+} from u_i^+, x_i^{e+}
  swap U for U^+ and X for X^+
  compute O

- implementation details:
  - line search: $u_i^+ = u_i + \alpha k_i + K_i(x_i^{e+} - x_i^e)$ with α such that $O(X^+, U^+) < O(X, U)$ (start at $\alpha = 1$, decr. iteratively)
    - ↳ ensures that each iteration improves guess U by allowing O to only sink → will find a minimum
    - alternative: possible to recover K, so that Armijo cond. can be applied?
  - regularization: $Q_{uu} \leftarrow Q_{uu} + rI$ with r such that linesearch does not fail (increase r iteratively)
    - ↳ analogous approach to Gauss-Newton. guarantees on lowering cost? ← can do better?
  - stopping cond.: same as Gauss-Newton

sparse ILQR (reformulation of implicit method that avoids the dense sensitivity matrices f_x, f_u)

$$Q_{uu}\delta u + Q_{ux}\delta x + Q_u = 0$$
$$(o_{uu} + f_u^T V'_{x'x'} f_u)\delta u + (o_{ux} + f_u^T V'_{x'x'} f_x)\delta x + (o_u + f_u^T V'_{x'}) = 0$$
$$o_{uu}\delta u + o_{ux}\delta x + f_u^T V'_{x'x'}\delta x' + o_u + f_u^T V'_{x'} = 0 \quad \leftarrow \text{substit: } \delta x' = f_u\delta u + f_x\delta x \quad \leftrightarrow g_{x'}\delta x' + g_x\delta x + g_u\delta u = 0$$
$$o_{uu}\delta u + o_{ux}\delta x + g_u^T\lambda^+ + o_u = 0 \quad \leftarrow \text{substit: } g_u^T\lambda^+ = f_u^T V'_{x'x'}\delta x' + f_u^T V'_{x'} \leftrightarrow V'_{x'x'}\delta x' + V'_{x'} + g_{x'}^T\lambda^+ = 0$$

$$\hookrightarrow \begin{bmatrix} V'_{x'x'} & 0 & g_{x'}^T \\ 0 & o_{uu} & g_u^T \\ g_{x'} & g_u & 0 \end{bmatrix}\begin{bmatrix}\delta x' \\ \delta u \\ \lambda^+\end{bmatrix} = -\begin{bmatrix} V'_{x'} \\ o_u \\ 0\end{bmatrix} - \begin{bmatrix} 0 \\ o_{ux} \\ g_x\end{bmatrix}\delta x \rightarrow \begin{bmatrix}\delta x' \\ \delta u \\ \lambda^+\end{bmatrix} = \begin{bmatrix} k_{x'} \\ k \\ k_{\lambda^+}\end{bmatrix} + \begin{bmatrix} K_{x'} \\ K \\ K_{\lambda^+}\end{bmatrix}\delta x$$
$$A_{sp} \quad x_{sp} \quad b_{sp} \quad c_{sp} \qquad\qquad x_{sp} \quad k_{sp} \quad K_{sp}$$

with k_{sp}, K_{sp} from solving $A_{sp}k_{sp} = -b_{sp}$; $A_{sp}K_{sp} = -c_{sp}$; get k, K from inside k_{sp}, K_{sp}

$$\hookrightarrow V_x = Q_x + K^T Q_u + Q_{xu}k + K^T Q_{uu}k = \dots$$
$$= \begin{bmatrix} I \\ K_{x'} \\ K \\ K_{\lambda^+}\end{bmatrix}^T \left(\begin{bmatrix} o_x \\ V'_{x'} \\ o_u \\ 0\end{bmatrix} + \begin{bmatrix} o_{xx} & 0 & o_{xu} & g_x^T \\ 0 & V'_{x'x'} & 0 & g_{x'}^T \\ o_{ux} & 0 & o_{uu} & g_u^T \\ g_x & g_{x'} & g_u & 0\end{bmatrix}^T \begin{bmatrix} 0 \\ k_{x'} \\ k \\ k_{\lambda^+}\end{bmatrix}\right)$$

$$V_{xx} = Q_{xx} + Q_{xu}K + K^T Q_{ux} + K^T Q_{uu}K = \dots$$
$$= \begin{bmatrix} I \\ K_{x'} \\ K \\ K_{\lambda^+}\end{bmatrix}^T \begin{bmatrix} o_{xx} & 0 & o_{xu} & g_x^T \\ 0 & V'_{x'x'} & 0 & g_{x'}^T \\ o_{ux} & 0 & o_{uu} & g_u^T \\ g_x & g_{x'} & g_u & 0\end{bmatrix}\begin{bmatrix} I \\ K_{x'} \\ K \\ K_{\lambda^+}\end{bmatrix}$$

(see dual ILQR for derivation of V_x, V_{xx} equations as the update policy is equal for $g=0$)

↳ does not contain f_u, f_x! this system to compute k, K is larger, but sparse (except for $V'_{x'x'}$)!

↳ since sparse ILQR is very similar to primal-dual ILQR, the same shift eq. policy simplification can be used to make the lin.eq.system smaller ($\lambda^+ = \lambda + \delta\lambda$, $k_{\lambda^+} = k_\lambda + \lambda$, $K_{\lambda^+} = K_\lambda$, $0 = g$)

- **implementation details:**
  - **line search:** same as ILQR
  - **regularization:** $o_{uu} \leftarrow o_{uu} + rI$ with r such that linesearch does not fail (increase r iteratively)
    - ↳ same effect as non-sparse: $(o_{uu} + rI) + f_u^T V'_{x'x'} f_u = Q_{uu} + rI$ same problem like implicit ILQR
  - **stopping cond.:** same as Gauss-Newton

primal-dual iLQR (use KKT conditions to solve for cost and constraint simultaneously)

primal problem: $V(x) = \min_u \; o(x,u) + V'(f(x,u)) \;\rightarrow\; V(x) = \min_{x',u} \; o(x,u) + V'(x') \;\text{ subj.to: }\; g(x,x',u)=0$

↳ dual problem: $V(x) = \operatorname{extr}_\lambda \min_{x',u} L(x,x',u,\lambda) \;\text{ with: }\; L(x,x',u,\lambda) = o(x,u) + V'(x') + \lambda^T g(x,x',u)$

second order Taylor expansion of L:

$$L(x+\delta x, x'+\delta x', u+\delta u, \lambda+\delta\lambda) = L(x,x',u,\lambda) + \nabla L^T \begin{bmatrix}\delta x\\ \delta x'\\ \delta u\\ \delta\lambda\end{bmatrix} + \tfrac{1}{2}\begin{bmatrix}\delta x\\ \delta x'\\ \delta u\\ \delta\lambda\end{bmatrix}^T \nabla^2 L \begin{bmatrix}\delta x\\ \delta x'\\ \delta u\\ \delta\lambda\end{bmatrix}$$

with:

$$\nabla L = \begin{bmatrix} o_x + g_x^T\lambda \\ V'_{x'} + g_{x'}^T\lambda \\ o_u + g_u^T\lambda \\ g \end{bmatrix} \quad ; \quad \nabla^2 L = \begin{bmatrix} o_{xx}+g_{xx}:\lambda & g_{xx'}:\lambda & o_{xu}+g_{xu}:\lambda & g_x^T \\ g_{x'x}:\lambda & V'_{x'x'}+g_{x'x'}:\lambda & g_{x'u}:\lambda & g_{x'}^T \\ o_{ux}+g_{ux}:\lambda & g_{ux'}:\lambda & o_{uu}+g_{uu}:\lambda & g_u^T \\ g_x & g_{x'} & g_u & 0 \end{bmatrix}$$

Gauss: drop $g_{\square\square}$ →

$$= \begin{bmatrix} o_{xx} & 0 & o_{xu} & g_x^T \\ 0 & V'_{x'x'} & 0 & g_{x'}^T \\ o_{ux} & 0 & o_{uu} & g_u^T \\ g_x & g_{x'} & g_u & 0 \end{bmatrix}$$

KKT conditions:

$$\begin{bmatrix}\partial L/\partial\delta x'\\ \partial L/\partial\delta u\\ \partial L/\partial\delta\lambda\end{bmatrix} = \begin{bmatrix} V'_{x'} + g_{x'}^T\lambda \\ o_u + g_u^T\lambda \\ g \end{bmatrix} + \begin{bmatrix} 0 & V'_{x'x'} & 0 & g_{x'}^T \\ o_{ux} & 0 & o_{uu} & g_u^T \\ g_x & g_{x'} & g_u & 0 \end{bmatrix} \begin{bmatrix}\delta x\\ \delta x'\\ \delta u\\ \delta\lambda\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix}$$

$$\rightarrow \begin{bmatrix} V'_{x'x'} & 0 & g_{x'}^T \\ 0 & o_{uu} & g_u^T \\ g_{x'} & g_u & 0 \end{bmatrix} \begin{bmatrix}\delta x'\\ \delta u\\ \delta\lambda\end{bmatrix} = - \begin{bmatrix} V'_{x'} + g_{x'}^T\lambda \\ o_u + g_u^T\lambda \\ g \end{bmatrix} - \begin{bmatrix} 0 \\ o_{ux} \\ g_x \end{bmatrix} \delta x \;\rightarrow\; \begin{bmatrix}\delta x'\\ \delta u\\ \delta\lambda\end{bmatrix} = \begin{bmatrix} k_{x'} \\ k_u \\ k_\lambda \end{bmatrix} + \begin{bmatrix} K_{x'} \\ K_u \\ K_\lambda \end{bmatrix} \delta x$$

$$\begin{bmatrix} V'_{x'x'} & 0 & g_{x'}^T \\ 0 & o_{uu} & g_u^T \\ g_{x'} & g_u & 0 \end{bmatrix} \begin{bmatrix} k_{x'} \\ k_u \\ k_\lambda + \lambda \end{bmatrix} = - \begin{bmatrix} V'_{x'} \\ o_u \\ g \end{bmatrix} \qquad \begin{bmatrix} V'_{x'x'} & 0 & g_{x'}^T \\ 0 & o_{uu} & g_u^T \\ g_{x'} & g_u & 0 \end{bmatrix} \begin{bmatrix} K_{x'} \\ K_u \\ K_\lambda \end{bmatrix} = - \begin{bmatrix} 0 \\ o_{ux} \\ g_x \end{bmatrix}$$

← x', u policy does not depend on λ!

with: $\tilde{k} = \begin{bmatrix} 0 \\ k_{x'} \\ k_u \\ k_\lambda \end{bmatrix} \quad \tilde{K} = \begin{bmatrix} I \\ K_{x'} \\ K_u \\ K_\lambda \end{bmatrix}$

propagation of cost-to-go derivatives V_x, V_{xx}:

$$V(x+\delta x) = L(x+\delta x, x'+\delta x', u+\delta u, \lambda+\delta\lambda)\Big|_{\substack{\delta x' = k_{x'} + K_{x'}\delta x \\ \delta u = k_u + K_u \delta x \\ \delta\lambda = k_\lambda + K_\lambda \delta x}} = L + \nabla L^T \tilde{k} + \nabla L^T \tilde{K}\delta x + \tfrac{1}{2}\tilde{k}^T\nabla^2 L\,\tilde{k} + \tilde{k}^T\nabla^2 L\,\tilde{K}\delta x + \tfrac{1}{2}\delta x^T \tilde{K}^T \nabla^2 L\,\tilde{K}\delta x$$

$$V_x = \partial V(x+\delta x)/\partial \delta x\,\big|_{\delta x=0} = \tilde{K}^T\nabla L + \tilde{K}^T\nabla^2 L^T \tilde{k} = \ldots =$$

$$= \begin{bmatrix} I \\ K_{x'} \\ K_u \\ K_\lambda \end{bmatrix}^T \left(\begin{bmatrix} o_x \\ V'_{x'} \\ o_u \\ g \end{bmatrix} + \begin{bmatrix} o_{xx} & 0 & o_{xu} & g_x^T \\ 0 & V'_{x'x'} & 0 & g_{x'}^T \\ o_{ux} & 0 & o_{uu} & g_u^T \\ g_x & g_{x'} & g_u & 0 \end{bmatrix}^T \begin{bmatrix} 0 \\ k_{x'} \\ k_u \\ k_\lambda + \lambda \end{bmatrix} \right)$$

$$V_{xx} = \partial^2 V(x+\delta x)/\partial \delta x^2\,\big|_{\delta x=0} = \tilde{K}^T\nabla^2 L\,\tilde{K} =$$

$$= \begin{bmatrix} I \\ K_{x'} \\ K_u \\ K_\lambda \end{bmatrix}^T \begin{bmatrix} o_{xx} & 0 & o_{xu} & g_x^T \\ 0 & V'_{x'x'} & 0 & g_{x'}^T \\ o_{ux} & 0 & o_{uu} & g_u^T \\ g_x & g_{x'} & g_u & 0 \end{bmatrix} \begin{bmatrix} I \\ K_{x'} \\ K_u \\ K_\lambda \end{bmatrix}$$

when working with extended systems it would be convenient if the policy $\delta x' = k_{x'} + K_{x'} \delta x$ matched the shift equations, as then these parts of the policy wouldn't need to be computed and the extended states wouldn't need to be tracked. This can be proven by inserting the shift eq. in the policy eq.:

extended state: $x = \begin{bmatrix} x_{old}^T & x_{new}^T & u_{old}^T & u_{last}^T \end{bmatrix}^T$ → x' from x, u: $x' = \begin{bmatrix} x'_{old} \\ x'_{new} \\ u'_{old} \\ u'_{last} \end{bmatrix} \begin{matrix} \text{drop} \rightarrow \begin{bmatrix} x_{old} \end{bmatrix} \\ \leftarrow \\ \ldots\ldots x_{new} \\ \leftarrow \text{calculated!} \\ \text{drop} \rightarrow \begin{bmatrix} u_{old} \end{bmatrix} \\ \leftarrow \\ \ldots\ldots u_{last} \\ \leftarrow u \end{matrix}$ } shift eq. } shift eq.

(see chap. "structure" for more info)

split policy with fixed shift eq.: $k_{x'} = \begin{bmatrix} k_{x'(x0)} \\ k_{x'(xn)} \\ k_{x'(u0)} \\ k_{x'(ul)} \end{bmatrix} = \begin{bmatrix} 0 \\ calc. \\ 0 \\ k_u \end{bmatrix}$; $K_{x'} = \begin{bmatrix} K_{x'(x0)} \\ K_{x'(xn)} \\ K_{x'(u0)} \\ K_{x'(ul)} \end{bmatrix} = \begin{bmatrix} \overset{x0}{0} & \overset{xn}{I} & \overset{u0}{0} & \overset{ul}{0} \\ & calc. & & \\ 0 & 0 & 0 & I \\ & K_u & & \end{bmatrix}$

split cost-to-go derivatives: $V'_{x'x'} = \begin{bmatrix} \overset{x0}{\overset{\downarrow}{V'_{x'x'(x0,x0)}}} & \overset{xn}{\overset{\downarrow}{V'_{x'x'(x0,xn)}}} & \overset{u0}{\overset{\downarrow}{V'_{x'x'(x0,u0)}}} & \overset{ul}{\overset{\downarrow}{V'_{x'x'(x0,ul)}}} \\ V'_{x'x'(xn,x0)} & V'_{x'x'(xn,xn)} & V'_{x'x'(xn,u0)} & V'_{x'x'(xn,ul)} \\ V'_{x'x'(u0,x0)} & V'_{x'x'(u0,xn)} & V'_{x'x'(u0,u0)} & V'_{x'x'(u0,ul)} \\ V'_{x'x'(xl,x0)} & V'_{x'x'(xl,xn)} & V'_{x'x'(xl,u0)} & V'_{x'x'(xl,ul)} \end{bmatrix} \begin{matrix} x0 \\ xn \\ u0 \\ ul \end{matrix}$; $V'_{x'} = \begin{bmatrix} V'_{x'(x0)} \\ V'_{x'(xn)} \\ V'_{x'(u0)} \\ V'_{x'(xl)} \end{bmatrix} \begin{matrix} x0 \\ xn \\ u0 \\ ul \end{matrix}$

split system dynamics:

$g_{x'} = \begin{bmatrix} \overset{x0}{I} & \overset{xn}{0} & \overset{u0}{0} & \overset{ul}{0} \\ 0 & g_{x'(xn,xn)} & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \begin{matrix} x0 \\ xn \\ u0 \\ ul \end{matrix}$; $g_u = \begin{bmatrix} \overset{u}{0} \\ g_{u(xn)} \\ 0 \\ -I \end{bmatrix} \begin{matrix} x0 \\ xn \\ u0 \\ ul \end{matrix}$; $g_x = \begin{bmatrix} \overset{x0}{0} & \overset{xn}{-I} & \overset{u0}{0} & \overset{ul}{0} \\ & g_{x(xn,all)} & & \\ 0 & 0 & 0 & -I \\ & 0 & & \end{bmatrix} \begin{matrix} x0 \\ xn \\ u0 \\ ul \end{matrix}$; $g = \begin{bmatrix} 0 \\ g_{(xn)} \\ 0 \\ 0 \end{bmatrix} \begin{matrix} x0 \\ xn \\ u0 \\ ul \end{matrix}$ ← from policy

reshape coefficient matrix of policy computation:

$$\begin{bmatrix} V'_{x'x'} & 0 & g_{x'}^T \\ 0 & o_{uu} & g_u^T \\ g_{x'} & g_u & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta u \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} V'_{x'x'(x0,x0)} & V'_{x'x'(x0,xn)} & V'_{x'x'(x0,u0)} & V'_{x'x'(x0,ul)} & 0 & I & 0 & 0 & 0 \\ V'_{x'x'(xn,x0)} & V'_{x'x'(xn,xn)} & V'_{x'x'(xn,u0)} & V'_{x'x'(xn,ul)} & 0 & 0 & g_{x'(xn,xn)}^T & 0 & 0 \\ V'_{x'x'(u0,x0)} & V'_{x'x'(u0,xn)} & V'_{x'x'(u0,u0)} & V'_{x'x'(u0,ul)} & 0 & 0 & 0 & I & 0 \\ V'_{x'x'(xl,x0)} & V'_{x'x'(xl,xn)} & V'_{x'x'(xl,u0)} & V'_{x'x'(xl,ul)} & 0 & 0 & 0 & 0 & I \\ 0 & 0 & 0 & 0 & o_{uu} & 0 & g_{u(xn)}^T & 0 & -I \\ I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & g_{x'(xn,xn)} & 0 & 0 & g_{u(xn)} & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & -I & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{(x0)} \\ \Delta x_{(xn)} \\ \Delta x_{(u0)} \\ \Delta x_{(xl)} \\ \Delta u \\ \Delta \lambda_{(x0)} \\ \Delta \lambda_{(xn)} \\ \Delta \lambda_{(u0)} \\ \Delta \lambda_{(xl)} \end{bmatrix}$$

($\Delta x, \Delta u, \Delta \lambda$ are placeholders!)

$$\begin{bmatrix} V'_{x'x'(x0,xn)} & V'_{x'x'(x0,ul)} & I & 0 & 0 & 0 \\ V'_{x'x'(xn,xn)} & V'_{x'x'(xn,ul)} & 0 & g_{x'(xn,xn)}^{T} & 0 & 0 \\ V'_{x'x'(u0,xn)} & V'_{x'x'(u0,ul)} & 0 & 0 & I & 0 \\ V'_{x'x'(xl,xn)} & V'_{x'x'(xl,ul)} & 0 & 0 & 0 & I \\ 0 & o_{uu} & 0 & g_{u(xn)}^{T} & 0 & -I \\ 0 & 0 & 0 & 0 & 0 & 0 \\ g_{x'(xn,xn)} & g_{u(xn)} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{(xn)} \\ \Delta u \\ \Delta\lambda_{(x0)} \\ \Delta\lambda_{(xn)} \\ \Delta\lambda_{(u0)} \\ \Delta\lambda_{(xl)} \end{bmatrix} + \begin{bmatrix} V'_{x'x'(x0,x0)} & V'_{x'x'(x0,u0)} \\ V'_{x'x'(xn,x0)} & V'_{x'x'(xn,u0)} \\ V'_{x'x'(u0,x0)} & V'_{x'x'(u0,u0)} \\ V'_{x'x'(xl,x0)} & V'_{x'x'(xl,u0)} \\ 0 & 0 \\ I & 0 \\ 0 & 0 \\ 0 & I \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{(x0)} \\ \Delta x_{(u0)} \end{bmatrix}$$

system to compute k of policy:

$$\begin{bmatrix} V'_{x'x'(x0,xn)} & V'_{x'x'(x0,ul)} & I & 0 & 0 & 0 \\ V'_{x'x'(xn,xn)} & V'_{x'x'(xn,ul)} & 0 & g_{x'(xn,xn)}^{T} & 0 & 0 \\ V'_{x'x'(u0,xn)} & V'_{x'x'(u0,ul)} & 0 & 0 & I & 0 \\ V'_{x'x'(xl,xn)} & V'_{x'x'(xl,ul)} & 0 & 0 & 0 & I \\ 0 & o_{uu} & 0 & g_{u(xn)}^{T} & 0 & -I \\ 0 & 0 & 0 & 0 & 0 & 0 \\ g_{x'(xn,xn)} & g_{u(xn)} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} k_{x'(xn)} \\ k_{u} \\ k_{\lambda(x0)}+\lambda_{(x0)} \\ k_{\lambda(xn)}+\lambda_{(xn)} \\ k_{\lambda(u0)}+\lambda_{(u0)} \\ k_{\lambda(xl)}+\lambda_{(xl)} \end{bmatrix} = - \begin{bmatrix} V'_{x'(x0)} \\ V'_{x'(xn)} \\ V'_{x'(u0)} \\ V'_{x'(xl)} \\ o_{u} \\ 0 \\ g_{(xn)} \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} V'_{x'x'(x0,x0)} & V'_{x'x'(x0,u0)} \\ V'_{x'x'(xn,x0)} & V'_{x'x'(xn,u0)} \\ V'_{x'x'(u0,x0)} & V'_{x'x'(u0,u0)} \\ V'_{x'x'(xl,x0)} & V'_{x'x'(xl,u0)} \\ 0 & 0 \\ I & 0 \\ 0 & 0 \\ 0 & I \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} V'_{x'x'(x0,xn)} & V'_{x'x'(x0,ul)} & I & 0 & 0 & 0 \\ V'_{x'x'(xn,xn)} & V'_{x'x'(xn,ul)} & 0 & g_{x'(xn,xn)}^{T} & 0 & 0 \\ V'_{x'x'(u0,xn)} & V'_{x'x'(u0,ul)} & 0 & 0 & I & 0 \\ V'_{x'x'(xl,xn)} & V'_{x'x'(xl,ul)} & 0 & 0 & 0 & I \\ 0 & o_{uu} & 0 & g_{u(xn)}^{T} & 0 & -I \\ g_{x'(xn,xn)} & g_{u(xn)} & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} k_{x'(xn)} \\ k_{u} \\ k_{\lambda(x0)}+\lambda_{(x0)} \\ k_{\lambda(xn)}+\lambda_{(xn)} \\ k_{\lambda(u0)}+\lambda_{(u0)} \\ k_{\lambda(xl)}+\lambda_{(xl)} \end{bmatrix} = - \begin{bmatrix} V'_{x'(x0)} \\ V'_{x'(xn)} \\ V'_{x'(u0)} \\ V'_{x'(xl)} \\ o_{u} \\ g_{(xn)} \end{bmatrix}$$

system to compute K of policy:

$$\begin{bmatrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{bmatrix} \begin{bmatrix} K_{x'(xn)} \\ K_{u} \\ K_{\lambda(x0)} \\ K_{\lambda(xn)} \\ K_{\lambda(u0)} \\ K_{\lambda(xl)} \end{bmatrix} = - \begin{bmatrix} \\ \\ 0 \\ \\ \\ o_{ux} \\ \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} -I \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} \\ g_{x(xn,all)} \\ \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} -I \end{bmatrix} \\ 0 \\ \\ \end{bmatrix} - \begin{bmatrix} V'_{x'x'(x0,x0)} & V'_{x'x'(x0,u0)} \\ V'_{x'x'(xn,x0)} & V'_{x'x'(xn,u0)} \\ V'_{x'x'(u0,x0)} & V'_{x'x'(u0,u0)} \\ V'_{x'x'(xl,x0)} & V'_{x'x'(xl,u0)} \\ 0 & 0 \\ I & 0 \\ 0 & 0 \\ 0 & I \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} I \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} \\ \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} I \end{bmatrix} & \end{bmatrix}$$

$$\begin{bmatrix} V'_{x'x'(x0,xn)} & V'_{x'x'(x0,ul)} & I & 0 & 0 & 0 \\ V'_{x'x'(xn,xn)} & V'_{x'x'(xn,ul)} & 0 & g_{x'(xn,xn)}^\top & 0 & 0 \\ V'_{x'x'(u0,xn)} & V'_{x'x'(u0,ul)} & 0 & 0 & I & 0 \\ V'_{x'x'(xl,xn)} & V'_{x'x'(xl,ul)} & 0 & 0 & 0 & I \\ 0 & 0_{uu} & 0 & g_{u(xn)}^\top & 0 & -I \\ g_{x'(xn,xn)} & g_{u(xn)} & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} K_{x'(xn)} \\ K_u \\ K_{\lambda(x0)} \\ K_{\lambda(xn)} \\ K_{\lambda(u0)} \\ K_{\lambda(xl)} \end{bmatrix} = - \begin{bmatrix} \begin{bmatrix} V'_{x'x'(x0,x0)} & V'_{x'x'(x0,u0)} \\ V'_{x'x'(xn,x0)} & V'_{x'x'(xn,u0)} \\ V'_{x'x'(u0,x0)} & V'_{x'x'(u0,u0)} \\ V'_{x'x'(xl,x0)} & V'_{x'x'(xl,u0)} \end{bmatrix} \begin{bmatrix} 0 & I & 0 & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \\ 0_{ux} \\ g_{x(xn,all)} \end{bmatrix}$$

both the systems to solve for k and K are not overdetermined after inserting the shift policy, meaning that the shift is actually part of the general solution. it is still worthwile to do this substitution as the new systems are smaller and thus easier to solve.

- **relation to sparse ILQR:**

  the relation between primal-dual and sparse ILQR is the same as for gauss-newton, as the step computation systems are equal if g=0 !

- **implementation details:**

  - BOX (*only limit step size and residuals*)

    - *line search:* $u_i^+ = u_i + \alpha k_{u_i} + K_{u_i}(X_i^{ext+} - X_i^{ext})$, $x_{i+1}^+ = x_{i+1} + \alpha k_{x'(xn)i} + K_{x'(xn)i}(X_i^{ext+} - X_i^{ext})$
      decrease α *until* $\alpha k_{u_i} + K_{u_i}(X_i^{ext+} - X_i^{ext}) < \delta U_{MAX}$ *and* $g(X_i^{ext+}, u_i^+, x_{i+1}^+) < g_{MAX}$
    - *regularization:* none
    - *stopping cond.:* stop if $|\mathcal{O} - \mathcal{O}^+|/\mathcal{O} < tol \approx 10^{-3}$ and $g(X^+, U^+) < tol \approx 10^{-8}$

  (*reset if linesearch hits* g_{max} *twice in a row or if* $\mathcal{O}$ *tollerance is reached, but not g*)

forward simulation

newton method

find x_i such that: $g(x_i, x_{i-1}, x_{i-2}, \dots, u_i) = 0$ $\quad \forall i$: timesteps in trajectory

↳ $g(x_i + \delta x_i, x_{i-1}, x_{i-2}, \dots, u_i) \approx g + \partial g/\partial x_i \cdot \delta x_i = 0 \rightarrow \delta x_i = -(\partial g/\partial x_i^T)^{-1} g \rightarrow$ improved guess: $x_i^+ = x_i + \delta x_i$

- implementation details:

  - line search: $x_i^+ = x_i + \alpha \delta x_i$ with α such that $\|g(x_i^+, \dots)\| < \|g(x_i, \dots)\|$ (start at $\alpha = 1$, decrease iteratively)

    $$\frac{d}{d\alpha} \|g(x_i + \alpha \delta x_i, \dots)\|\Big|_{\alpha=0} = \frac{g(x_i + \alpha \delta x_i, \dots)^T (\partial g/\partial x_i (x_i + \alpha \delta x_i, \dots) \cdot \delta x_i)}{\|g(x_i + \alpha \delta x_i, \dots)\|}\Big|_{\alpha=0} = \frac{g^T (\partial g/\partial x_i \cdot \delta x_i)}{\|g\|} = \frac{-g^T g}{\|g\|} = -\|g\| \leq 0$$

    alternative: $\|g(x_i^+, \dots)\| < \|g(x_i, \dots)\| + c_1 \alpha \, g(x_i, \dots)^T \frac{\partial g}{\partial x_i}(x_i, \dots) / \|g(x_i, \dots)\|$ → expected $\|g\|$ decrease (Armijo)

    alternative: if available use an energy $e(x_i, \dots)$ that is minimal iff $g = 0$ → $e^+ < e$ (can also add Armijo)

  - regularization: usually not needed. (method performs poorly if $\partial g/\partial x_i$ is close to singular)

    alternative: $\delta x_i = -\left(\frac{\partial g}{\partial x_i} \frac{\partial g}{\partial x_i}^T + rI\right)^{-1} \frac{\partial g}{\partial x_i} g$ (damped inverse, avoids singularity when $r \uparrow$)

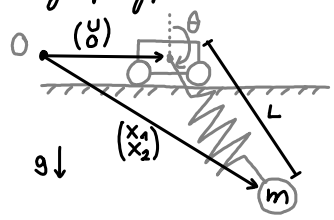
  - stopping cond.: $\|g(u_i^+, \dots)\| < tol \approx 10^{-8}$ (stop when $g \approx 0$)

Models

discretization:
- BDF1: $\dot{x}_i = \frac{1}{h}(x_i - x_{i-1})$; $\ddot{x}_i = \frac{1}{h^2}(x_i - 2x_{i-1} + x_{i-2})$
- BDF2: $\dot{x}_i = \frac{1}{h}(1.5x_i - 2x_{i-1} + 0.5x_{i-2})$; $\ddot{x}_i = \frac{1}{h^2}(2x_i - 5x_{i-1} + 4x_{i-2} - x_{i-3})$

(x_i: x at time step i, h: timestep lenght)

- single spring pendulum:

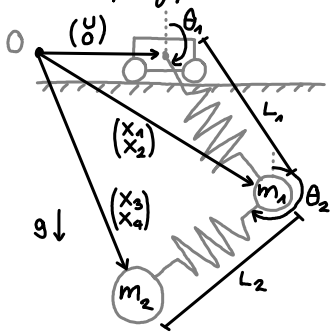


$L = \sqrt{(x_1 - u)^2 + (x_2)^2}$; $\theta = \operatorname{atan2}(x_1 - u, x_2)$; $F = K \cdot (L - L_0)$

$m\ddot{x}_1 = -\sin(\theta) \cdot F$; $m\ddot{x}_2 = -\cos(\theta) \cdot F - m \cdot g$

$$\hookrightarrow \begin{pmatrix} m\ddot{x}_1 + \sin(\theta)F \\ m\ddot{x}_2 + \cos(\theta)F + mg \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \rightarrow g(x, \ddot{x}, u) = 0 \quad \text{with } x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

- double spring pendulum:



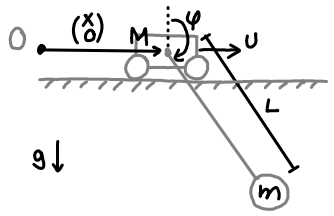
$L_1 = \sqrt{(x_1 - u)^2 + (x_2)^2}$; $\theta_1 = \operatorname{atan2}(x_1 - u, x_2)$; $F_1 = K_1 \cdot (L_1 - L_{1,0})$

$L_2 = \sqrt{(x_3 - x_1)^2 + (x_4 - x_2)^2}$; $\theta_2 = \operatorname{atan2}(x_3 - x_1, x_4 - x_2)$; $F_2 = K_2 \cdot (L_2 - L_{2,0})$

$m_1\ddot{x}_1 = -\sin(\theta_1)F_1 + \sin(\theta_2)F_2$; $m_1\ddot{x}_2 = -\cos(\theta_1)F_1 + \cos(\theta_2)F_2 - m_1 g$

$m_2\ddot{x}_3 = -\sin(\theta_2)F_2$; $m_2\ddot{x}_4 = -\cos(\theta_2)F_2 - m_2 g$

- single rigid pendulum:



$v_M = \dot{x}$; $v_m = \sqrt{(\dot{x} + \dot{\varphi}L\cos\varphi)^2 + (\dot{\varphi}L\sin\varphi)^2} = \sqrt{\dot{x}^2 + 2\dot{x}\dot{\varphi}L\cos\varphi + \dot{\varphi}^2L^2}$

$T_M = \frac{1}{2}Mv_M^2 = \frac{1}{2}M\dot{x}^2$; $T_m = \frac{1}{2}mv_m^2 = \frac{1}{2}m(\dot{x}^2 + 2\dot{x}\dot{\varphi}L\cos\varphi + \dot{\varphi}^2L^2)$

$U_M = 0$; $U_m = mg\cos\varphi$

$\hookrightarrow L = T_M + T_m - U_M - U_m = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}m(\dot{x}^2 + 2\dot{x}\dot{\varphi}L\cos\varphi + \dot{\varphi}^2L^2) - mg\cos\varphi$

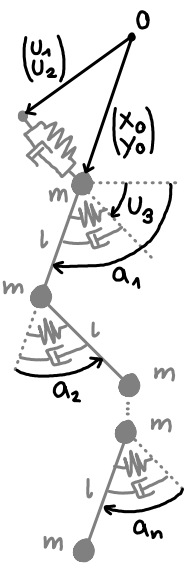
$\partial L/\partial x = 0$; $\partial L/\partial \dot{x} = M\dot{x} + m\dot{x} + m\dot{\varphi}L\cos\varphi$

$\partial L/\partial \varphi = -m\dot{x}\dot{\varphi}L\sin\varphi + mg\sin\varphi$; $\partial L/\partial \dot{\varphi} = m\dot{x}L\cos\varphi + m\dot{\varphi}L^2$

$$\frac{d}{dt}(\partial L/\partial \dot{x}) - \partial L/\partial x = (M+m)\ddot{x} + m\ddot{\varphi}L\cos\varphi - m\dot{\varphi}^2L\sin\varphi = u$$

$$\frac{d}{dt}(\partial L/\partial \dot{\varphi}) - \partial L/\partial \varphi = m\ddot{x}L\cos\varphi + m\ddot{\varphi}L^2 - mg\sin\varphi = 0$$

• rigid whip:



n: element count m: node mass l: element lenght

k_i: node spring const. d_i: node damper const.

$q = [x_0, y_0, a_1, a_2, \dots, a_n]^T$: states $u = [u_1, u_2, u_3]$: inputs

- i=0: strong linear spr./dmp.
- i=1: strong angular spr./dmp.
- i>1: weak angular spr./dmp.

incremental homog. transform matrices:

$$T_{inc,1} = \begin{bmatrix} \cos a_1 & -\sin a_1 & x_0 \\ \sin a_1 & \cos a_1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} ; \; T_{inc,i} = \begin{bmatrix} \cos a_i & -\sin a_i & l \\ \sin a_i & \cos a_i & 0 \\ 0 & 0 & 1 \end{bmatrix} ; \; T_{inc,n+1} = \begin{bmatrix} 1 & 0 & l \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

joint positions: $\bar{r}_i = \left(\prod_{j=0}^{i} T_{inc,j}\right) \cdot (0,0,1)^T$

geometric jacobian: $J_i = \begin{bmatrix} 1 & 0 & -(r_{1,y} - r_{i,y}) & \cdots & -(r_{(i-1),y} - r_{i,y}) & 0 & \cdots & 0 \\ 0 & 1 & (r_{1,x} - r_{i,x}) & \cdots & (r_{(i-1),x} - r_{i,x}) & 0 & \cdots & 0 \end{bmatrix}$ ($\rightarrow \dot{r}_i = J_i \dot{q}$)

energies: $T = \sum_{i=1}^{n+1} \frac{1}{2} m \dot{q}^T J_i^T J_i \dot{q}$ (kinetic E) ; $R = \frac{\wedge}{2} \dot{q}^T \mathrm{diag}(d) \dot{q}$ (viscous E)

$U = \frac{1}{2} (q - [u,0,\dots])^T \mathrm{diag}(k) (q - [u,0,\dots]) + \sum_{i=1}^{n+1} mg \begin{bmatrix} 0 \\ -1 \end{bmatrix}^T r_i$ (potential E)

lagrange method: $\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}}\right) - \frac{\partial T}{\partial q} + \frac{\partial R}{\partial \dot{q}} + \frac{\partial U}{\partial q} = 0$

↳ EoM: $\underbrace{\sum_{i=1}^{n+1} (m J_i^T J_i) \ddot{q}}_{\text{inertia}} - \underbrace{\sum_{i=1}^{n+1} (m J_i^T \dot{J}_i) \dot{q}}_{\text{centrifugal f.}} + \underbrace{\mathrm{diag}(d) \dot{q}}_{\text{dampers}} + \underbrace{\sum_{i=1}^{n+1} (mg J_i^T \begin{bmatrix} 0 \\ -1 \end{bmatrix}) q}_{\text{gravity}} + \underbrace{\mathrm{diag}(k)(q - [u,0,\dots])}_{\text{springs}} = 0$

Structure

n : dimension of current state
m : dimension of current input
N : prev. states included in extended state
M : prev. inputs included in extended state
T: trajectory lenght

notation changes in this chapter: $x \to x^c$; $x^e \to x^e$; $X \to x^t$; $u \to u^c$; $U \to u^t$
$g \to g^c$; $g^e \to g^e$; $g \to g^t$; $o \to o^c$; $o^e \to o^e$; $O \to o^t$

state/input descriptions

current:

$$x^c(t) = \begin{bmatrix} x_1^c(t) \\ x_2^c(t) \\ \vdots \\ x_n^c(t) \end{bmatrix} \in \mathbb{R}^n \qquad u^c(t) = \begin{bmatrix} u_1^c(t) \\ u_2^c(t) \\ \vdots \\ u_m^c(t) \end{bmatrix} \in \mathbb{R}^m$$

trajectory:

$$x^t = \begin{bmatrix} x^c(2) \\ x^c(3) \\ \vdots \\ x^c(T+1) \end{bmatrix} \in \mathbb{R}^{nT} \qquad u^t = \begin{bmatrix} u^c(1) \\ u^c(2) \\ \vdots \\ u(T) \end{bmatrix} \in \mathbb{R}^{mT}$$

wide:

$$x^w(t) = \begin{bmatrix} x^c(t-N+1) \\ \vdots \\ x^c(t) \\ x^c(t+1) \end{bmatrix} = \begin{bmatrix} x_1^w(t) \\ \vdots \\ x_N^w(t) \\ x_{N+1}^w(t) \end{bmatrix} \in \mathbb{R}^{n(N+1)}$$

$$u^w(t) = \begin{bmatrix} u^c(t-M+1) \\ \vdots \\ u^c(t) \end{bmatrix} = \begin{bmatrix} u_1^w(t) \\ \vdots \\ u_M^w(t) \end{bmatrix} \in \mathbb{R}^{mM}$$

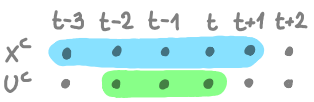
extended:

$$x^e(t) = \begin{bmatrix} x^c(t-N+1) \\ \vdots \\ x^c(t) \\ u^c(t-M+1) \\ \vdots \\ u^c(t-1) \end{bmatrix} = \begin{bmatrix} x_2^w(t) \\ \vdots \\ x_N^w(t) \\ u_1^w(t) \\ \vdots \\ u_{M-1}^w(t) \end{bmatrix} = \begin{bmatrix} x_1^e(t) \\ \vdots \\ x_N^e(t) \\ x_{N+1}^e(t) \\ \vdots \\ x_{N+M-1}^e(t) \end{bmatrix} \in \mathbb{R}^{nN+m(M-1)}$$

implicit dynamics

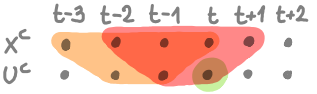
$$g^w(t) = g^w(x^w(t), u^w(t)) = 0 \in \mathbb{R}^n$$

e.g. $N=4, M=3$:



directly expresses system dynamics

$$g^e(t) = g^e(x^e(t), u^c(t), x^e(t+1)) = 0 \quad \in \mathbb{R}^{nN+m(M-1)}$$



add equations to force overlapping x^e states to match: $x_2^e(t) - x_1^e(t+1) = 0, \dots$

$$g^c(t) = g^c(x^c(t+1)) = 0 \quad \in \mathbb{R}^n$$

assume all prev. state/input to be const. (usefull for forward simulation)

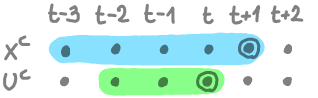
$$g^t = g^t(x^t, u^t, x^c(1)) = 0 \quad \in \mathbb{R}^{n \cdot T}$$

collection of g^w for all times

---

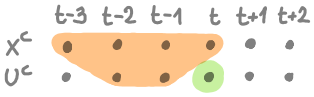
cost functions (o: stage cost, o_f: terminal cost)

$$o^w(t) = o^w(x^w(t), u^w(t)) \quad \in \mathbb{R}$$
$$o_f^w(t) = o_f^w(x^w(t), u^w(t)) \quad \in \mathbb{R}$$

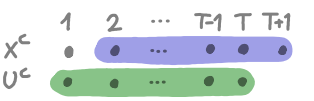


$o^w(t)$ must not use $x^c(t+1)$ and $o_f^w(t)$ must not use $x^c(t+1)$ nor $u^c(t)$! see DDP for reason.

$$o^e(t) = o^e(x^e(t), u^c(t)) \quad \in \mathbb{R}$$
$$o_f^e(t) = o_f^e(x^e(t)) \quad \in \mathbb{R}$$



$$o^t = o^t(x^t, u^t, x^c(1)) \quad \in \mathbb{R}$$



total trajectory cost: sum of all stage costs + terminal cost.

relation between implicit dynamics descriptions

$$g^w(t) = g^w(x^w(t), u^w(t)) = \begin{bmatrix} g_1^w(x^w(t), u^w(t)) \\ \vdots \\ g_n^w(x^w(t), u^w(t)) \end{bmatrix} \in \mathbb{R}^n$$

$$g^c(t) = g^c(x^c(t+1)) = g^w(x^c(t-N+1), \dots, x^c(t+1), u^c(t-M+1), \dots, u^c(t)) \in \mathbb{R}^n$$

($x^c(t-N+1), \dots, x^c(t+1)$: treat as known const.)

$$g^t = g^t(x^t, u^t, x^c(1)) = [g^w(x^w(1), u^w(1))^T, \dots, g^w(x^w(T), u^w(T))^T]^T \in \mathbb{R}^{nT}$$

($x^w(1), u^w(1)$: extracted from x^t, u^t)

$$g^e(t) = g^e(x^e(t), u^c(t), x^e(t+1)) = \begin{bmatrix} x_1^e(t+1) - x_2^e(t) \\ \vdots \\ x_{N-1}^e(t+1) - x_N^e(t) \end{bmatrix} \in \mathbb{R}^{nN+m(M-1)}$$

(shift states) $\in \mathbb{R}^{n(N-1)}$

$$g^w(x_1^e(t), \dots, x_N^e(t), x_N^e(t+1)) = x^w(t), \quad x_{N+1}^e(t), \dots, x_{N+M-1}^e(t), u^c(t)) = u^w(t)$$

$$\begin{bmatrix} x_{N+1}^e(t+1) - x_{N+2}^e(t) \\ \vdots \\ x_{N+M-2}^e(t+1) - x_{N+M-1}^e(t) \end{bmatrix}$$

(shift inputs) $\in \mathbb{R}^{m(M-2)}$

$$x_{N+M-1}^e(t+1) - u^c(t)$$

(copy curr. input) $\in \mathbb{R}^m$

derivatives of implicit dynamics

$$\partial g^w / \partial x^w = \begin{bmatrix} \partial g_1^w / \partial x_1^w & \dots & \partial g_1^w / \partial x_{N+1}^w \\ \vdots & & \vdots \\ \partial g_n^w / \partial x_1^w & \dots & \partial g_n^w / \partial x_{N+1}^w \end{bmatrix} \in \mathbb{R}^{n \times n(N+1)}$$

$$\partial g^w / \partial u^w = \begin{bmatrix} \partial g_1^w / \partial u_1^w & \dots & \partial g_1^w / \partial u_M^w \\ \vdots & & \vdots \\ \partial g_n^w / \partial u_1^w & \dots & \partial g_n^w / \partial u_M^w \end{bmatrix} \in \mathbb{R}^{n \times mM}$$

$$\partial g^e(t) / \partial x^e(t+1) = \begin{bmatrix} \mathbb{I}^{n(N-1) \times n(N-1)} & & \\ & \partial g^w(t) / \partial x_{N+1}^w(t) & \\ & & \mathbb{I}^{m(M-2) \times m(M-2)} \\ & & & \mathbb{I}^{mxm} \end{bmatrix} \in \mathbb{R}^{nN+m(M-1) \times nN+m(M-1)}$$

$$\partial g^e(t) / \partial u^c(t) = \begin{bmatrix} 0^{n(N-1) \times m} \\ \partial g^w(t) / \partial u_M^w(t) \\ 0^{m(M-2) \times m} \\ -\mathbb{I}^{mxm} \end{bmatrix} \in \mathbb{R}^{nN+m(M-1) \times m}$$

$$\partial g^e(t) / \partial x^e(t) = \begin{bmatrix} \begin{bmatrix} 0^{n(N-1) \times n} & -\mathbb{I}^{n(N-1) \times n(N-1)} \end{bmatrix} & 0^{n(N-1) \times m(M-1)} \\ \begin{bmatrix} \partial g^w(t) / \partial x_1^w(t) & \dots & \partial g^w(t) / \partial x_N^w(t) \end{bmatrix} & \begin{bmatrix} \partial g^w(t) / \partial u_1^w(t) & \dots & \partial g^w(t) / \partial u_{M-1}^w(t) \end{bmatrix} \\ 0^{m(M-2) \times nN} & \begin{bmatrix} 0^{m(M-2) \times m} & -\mathbb{I}^{m(M-2) \times m(M-2)} \end{bmatrix} \\ 0^{m \times nN} & 0^{m \times m(M-1)} \end{bmatrix} \in \mathbb{R}^{nN+m(M-1) \times nN+m(M-1)}$$

$$\partial g^c(t) / \partial x^c(t+1) = \partial g^w(t) / \partial x_{N+1}^w(t) \in \mathbb{R}^{n \times n}$$

$$\partial g^t / \partial x^t = \begin{bmatrix} \ddots & & & & \\ & 0 & \partial g^w(t) / \partial x^w(t) & 0 & \\ & & & \ddots & \\ & 0 & \partial g^w(t+1) / \partial x^w(t+1) & 0 & \\ & & & & \ddots \end{bmatrix} \begin{matrix} \\ g^w(1) \\ g^w(t) \\ g^w(t+1) \\ g^w(T) \end{matrix} \quad \begin{matrix} x^c(2), \dots, x^c(t-N+1), \dots, x^c(t+1), \dots, x^c(T+1) \end{matrix} \in \mathbb{R}^{nT \times nT}$$

$$\partial g^t / \partial u^t = \begin{bmatrix} \ddots & & & & \\ & 0 & \partial g^w(t) / \partial u^w(t) & 0 & \\ & & & \ddots & \\ & 0 & \partial g^w(t+1) / \partial u^w(t+1) & 0 & \\ & & & & \ddots \end{bmatrix} \begin{matrix} \\ g^w(1) \\ g^w(t) \\ g^w(t+1) \\ g^w(T) \end{matrix} \quad \begin{matrix} u^c(1), \dots, u^c(t-M+1), \dots, u^c(t), \dots, u^c(T) \end{matrix} \in \mathbb{R}^{nT \times mT}$$

relation between cost descriptions

$o^e(t) = o^e(x^e(t), u^c(t)) = o^w(t) = o^w(x^w(t), u^w(t)) \in \mathbb{R}$

$o_p^e(t) = o_f^e(x^e(t)) = o_f^w(t) = o_f^w(x^w(t), u^w(t)) \in \mathbb{R}$

$o^t = o^t(x^t, u^t, x^c(1)) = o_f^w(x^w(T+1), u^w(T+1)) + \sum_{t=1}^{T} o^w(x^w(t), u^w(t)) \in \mathbb{R}$

derivatives of cost

$\partial o^w / \partial x^w = \begin{bmatrix} \partial o^w / \partial x_1^w \\ \vdots \\ \partial o^w / \partial x_{N+1}^w \end{bmatrix} \in \mathbb{R}^{n(N+1)\times 1}$ $\quad \partial o^w / \partial u^w = \begin{bmatrix} \partial o^w / \partial u_1^w \\ \vdots \\ \partial o^w / \partial u_M^w \end{bmatrix} \in \mathbb{R}^{mM\times 1}$ $\quad \partial^2 o^w / \partial u^{w2} = \begin{bmatrix} \partial^2 o^w / \partial u_1^w \partial u_1^w & \cdots & \partial^2 o^w / \partial u_1^w \partial u_M^w \\ \vdots & & \vdots \\ \partial^2 o^w / \partial u_M^w \partial u_1^w & \cdots & \partial^2 o^w / \partial u_M^w \partial u_M^w \end{bmatrix} \in \mathbb{R}^{mM\times mM}$

$\partial^2 o^w / \partial x^{w2} = \begin{bmatrix} \partial^2 o^w / \partial x_1^w \partial x_1^w & \cdots & \partial^2 o^w / \partial x_1^w \partial x_{N+1}^w \\ \vdots & & \vdots \\ \partial^2 o^w / \partial x_{N+1}^w \partial x_1^w & \cdots & \partial^2 o^w / \partial x_{N+1}^w \partial x_{N+1}^w \end{bmatrix} \in \mathbb{R}^{n(N+1)\times n(N+1)}$ $\quad \partial o^w / \partial u^w \partial x^w = \begin{bmatrix} \partial^2 o^w / \partial u_1^w \partial x_1^w & \cdots & \partial^2 o^w / \partial u_1^w \partial x_{N+1}^w \\ \vdots & & \vdots \\ \partial^2 o^w / \partial u_M^w \partial x_1^w & \cdots & \partial^2 o^w / \partial u_M^w \partial x_{N+1}^w \end{bmatrix} \in \mathbb{R}^{mM\times n(N+1)}$

(wide terminal cost o_f^w is analogous to wide stage cost o^w)

$\partial o^e(t) / \partial x^e(t) = \begin{bmatrix} \partial o^w / \partial x_1^w \\ \vdots \\ \partial o^w / \partial x_N^w \\ \partial o^w / \partial u_1^w \\ \vdots \\ \partial o^w / \partial u_{M-1}^w \end{bmatrix} \in \mathbb{R}^{nN+m(M-1)\times 1}$

$\partial^2 o^e(t) / \partial x^e(t)^2 = \begin{bmatrix} \partial^2 o^w / \partial x_1^w \partial x_1^w & \cdots & \partial^2 o^w / \partial x_1^w \partial x_N^w & \partial^2 o^w / \partial x_1^w \partial u_1^w & \cdots & \partial^2 o^w / \partial x_1^w \partial u_{M-1}^w \\ \vdots & & \vdots & \vdots & & \vdots \\ \partial^2 o^w / \partial x_N^w \partial x_1^w & \cdots & \partial^2 o^w / \partial x_N^w \partial x_N^w & \partial^2 o^w / \partial x_N^w \partial u_1^w & \cdots & \partial^2 o^w / \partial x_N^w \partial u_{M-1}^w \\ \partial^2 o^w / \partial u_1^w \partial x_1^w & \cdots & \partial^2 o^w / \partial u_1^w \partial x_N^w & \partial^2 o^w / \partial u_1^w \partial u_1^w & \cdots & \partial^2 o^w / \partial u_1^w \partial u_{M-1}^w \\ \vdots & & \vdots & \vdots & & \vdots \\ \partial^2 o^w / \partial u_{M-1}^w \partial x_1^w & \cdots & \partial^2 o^w / \partial u_{M-1}^w \partial x_N^w & \partial^2 o^w / \partial u_{M-1}^w \partial u_1^w & \cdots & \partial^2 o^w / \partial u_{M-1}^w \partial u_{M-1}^w \end{bmatrix} \in \mathbb{R}^{nN+m(M-1)\times nN+m(M-1)}$

$\partial o^e(t) / \partial u^c(t) = \partial o^w / \partial u_M^w \in \mathbb{R}^{m\times 1}$ $\quad \partial^2 o^e(t) / \partial u^c(t)^2 = \partial^2 o^w / \partial u_M^w \partial u_M^w \in \mathbb{R}^{m\times m}$

$\partial o^e(t) / \partial u^c(t) \partial x^e(t) = \begin{bmatrix} \partial o^w / \partial u_M^w \partial x_1^w & \cdots & \partial o^w / \partial u_M^w \partial x_N^w & \partial o^w / \partial u_M^w \partial u_1^w & \cdots & \partial o^w / \partial u_M^w \partial u_{M-1}^w \end{bmatrix} \in \mathbb{R}^{m\times nN+m(M-1)}$

($\partial o_f^e(t) / \partial x^e(t)$ and $\partial^2 o_f^e(t) / \partial x^e(t)^2$ are analogous to $\partial o^e(t) / \partial x^e(t)$ and $\partial^2 o^e(t) / \partial x^e(t)^2$)

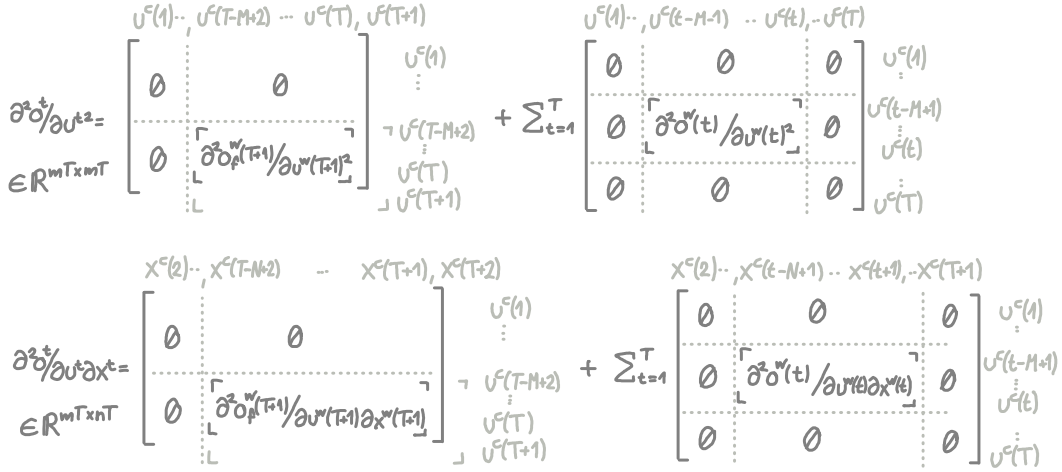
$\partial o^t / \partial x^t = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hline T \\ \partial o_f^w(T+1) / \partial x^w(T+1) \\ \hline \perp \end{bmatrix} \begin{matrix} x^c(2) \\ \vdots \\ \\ x^c(T-N+2) \\ \vdots \\ x^c(T+1) \\ x^c(T+2) \end{matrix} + \sum_{t=1}^{T} \begin{bmatrix} 0 \\ \hline T \\ \partial o^w(t) / \partial x^w(t) \\ \hline \perp \\ 0 \end{bmatrix} \begin{matrix} x^c(2) \\ \vdots \\ x^c(t-N+1) \\ \vdots \\ x^c(t+1) \\ \vdots \\ x^c(T+1) \end{matrix}$ $\quad \in \mathbb{R}^{nT\times 1}$

$\partial o^t / \partial u^t = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hline T \\ \partial o_f^w(T+1) / \partial u^w(T+1) \\ \hline \perp \end{bmatrix} \begin{matrix} u^c(1) \\ \vdots \\ \\ u^c(T-M+2) \\ \vdots \\ u^c(T) \\ u^c(T+1) \end{matrix} + \sum_{t=1}^{T} \begin{bmatrix} 0 \\ \hline T \\ \partial o^w(t) / \partial u^w(t) \\ \hline \perp \\ 0 \end{bmatrix} \begin{matrix} u^c(1) \\ \vdots \\ u^c(t-M+1) \\ \vdots \\ u^c(t) \\ \vdots \\ u^c(T) \end{matrix}$ $\quad \in \mathbb{R}^{mT\times 1}$

$\partial^2 o^t / \partial x^{t2} = \begin{matrix} x^c(2) \cdots, x^c(T-N+2) \cdots x^c(T+1), x^c(T+2) \end{matrix}$

$\begin{bmatrix} 0 & & 0 \\ \hline & \ulcorner\, \partial^2 o_f^w(T+1) / \partial x^w(T+1)^2 \,\urcorner & \\ 0 & \llcorner\, \llcorner & \end{bmatrix} \begin{matrix} x^c(2) \\ \vdots \\ \urcorner\, x^c(T-N+2) \\ \vdots \\ x^c(T+1) \\ \llcorner\, x^c(T+2) \end{matrix} + \sum_{t=1}^{T} \begin{bmatrix} 0 & 0 & 0 \\ 0 & \ulcorner\, \partial^2 o^w(t) / \partial x^w(t)^2 \,\urcorner & 0 \\ 0 & \llcorner\, \llcorner & 0 \end{bmatrix} \begin{matrix} x^c(2) \\ \vdots \\ x^c(t-N+1) \\ \vdots \\ x^c(t+1) \\ \vdots \\ x^c(T+1) \end{matrix}$ $\quad \in \mathbb{R}^{nT\times nT}$

$x^c(2) \cdots, x^c(t-N+1) \cdots x^c(t+1), \cdots x^c(T+1)$



5 Further Information

The information presented here is meant to provide the necessary knowledge to continue the authors research into this topic.

Reading material:

- *Dynamic manipulation of deformable objects with implicit integration*
  S. Zimmermann, R. Poranne, S.Coros, 2021
  CRL paper on the implicit Gauss-Newton and ILQR methods applied to FE mesh models.
- *Sparse Gauss-Newton for Accelerated Sensitivity Analysis*
  Anonymous ??, 2020
  CRL paper on the sparse Gauss-Newton method.
- *A dual Newton strategy for the efficient solution of sparse quadratic programs arising in SQP-based nonlinear MPC, J. Frasch et al., 2013*
  Interesting paper. Might not be relevant?
- *Numerical Optimization*
  J. Nocedal, S.Wright, 2006
  Textbook on basics of numerical optimization

Repositories:

- *https://gitlab.ethz.ch/simonzi/whipping*
  CRL C++ app (trajOpt) containing implementations of all TO methods. Implemented models are: a spring pendulum and a FE whip model (cost missing). In general this code is still work in progress.
- *https://gitlab.ethz.ch/mruggia/implicit-ddp-semesterproject*
  Contains MATLAB implementations of all TO methods with various pendulums and 2D whips as models. This implementation is more complete in its features, but is not built for performance and does not contain any large systems.

Future work:

- A major performance issue is present in the C++ code in the SysMat class for sparse matrices. In the current implementation it is necessary to loop trough an entire extended matrix to extract a single block.
- Look into ways to prioritize lowering of cost vs. residual in primal-dual methods.
- Try Armijo conditions on all line-searches and try filter/penalty approaches for the primal-dual methods (see hand-written notes)
- Possibly find a way to reconstruct the gradient matrix in the sparse methods in order to apply the Armijo condition.
- Perform performance comparison of the TO methods on a large system (like the FE whip in the C++ code) !!