# Finite-Horizon Optimal Control by Differential Dynamic Programming with Application to Robotics

Internship report

Mengda Li

June 14, 2019

# Contents

# Introduction

My internship is co-supervised by Justin Carpentier and Nicolas Mansard in the Willow research group at INRIA Paris in France.

Justin Carpentier is a postdoctoral researcher at the interface between Robotics, Machine Learning and Control in Willow. His research is devoted to the embedding of Optimal Control theory inside the formalism of Machine Learning, with Humanoid Robotics as a main target application.

Nicolas Mansard is a permanent researcher in the Gepetto research group at LAAS/CNRS, Toulouse. His research activities are concerned with sensor-based control, and more specifically the integration of sensor-based schemes into humanoid robot applications.

WILLOW is based in the Laboratoire d'Informatique de l'École Normale Superiéure and is a joint research team between INRIA Rocquencourt, École Normale Supérieure de Paris and Centre National de la Recherche Scientifique. Their research is concerned with representational issues in visual object recognition and scene understanding.

GEPETTO is a robotics research team in the Laboratoire d'analyse et d'architecture des systèmes of Centre National de la Recherche Scientifique. They aim at analyzing and generating the motion of anthropomorphic systems.

The goal of my internship is to embed the augmented Lagrangian method to DDP to solve robotics optimal control problem with constraints.

# Chapter 1

# Definitions of the problem

We want to find a path from some point $x_0$ to some point $p$ which doesn't cost too much.

## 1.1 Mathematical formulations

Let $n \in \mathbb{N}$, $m \in \mathbb{N}$, $x_0 \in \mathbb{R}^n$, $p \in \mathbb{R}^n$, $x \in \mathscr{C}^1([0,T] \to \mathbb{R}^n)$, $u \in \mathscr{L}^\infty([0,T) \to \mathbb{R}^m)$, and $f \in \mathscr{C}^2(\mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n)$.

$n$ is the dimension of the state variable and $m$ is the dimension of the control variable. We use $x$ as the state variable, $u$ as the control variable.

$x_0$ is the initial position (or state) and $p$ is the desired final position.

$x$ is an unknown function from $[0,T]$ to $\mathbb{R}^n$. It represents the trajectory.

$u$ is the control function which represents the variation of control over time.

We can control the trajectory $x$ by $u$:

$$\dot{x}(t) = f(x(t), u(t)) \tag{1.1}$$

the derivative of $x$ depends on $x$, $u$ and a function $f$.

**Goal 1: Controllability**

$$\begin{array}{ll} \text{find} & u \\ \text{subject to} & x(0) = x_0,\ x(T) = p, \\ & \dot{x}(t) = f(x(t), u(t)). \end{array} \tag{1.2}$$

**Goal 2: Optimal Control**

In reality, everything has a cost.

Controlling something has a cost, being somewhere has a cost. So we define a cost (or loss) function $l$ on $x$ and $u$, supposing that $l$ is in $\mathscr{C}^2(\mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R})$.

$$\begin{array}{ll} \underset{u}{\text{minimize}} & \displaystyle\int_0^T l(x(t), u(t))dt \\ \text{subject to} & x(0) = x_0,\ x(T) = p, \\ & \dot{x}(t) = f(x(t), u(t)). \end{array} \tag{1.3}$$

### 1.1.1 Transformation of the problem

Satisfying $x(0) = x_0$ is easy while satisfying $x(T) = p$ is not trivial considering $\dot{x}(t) = f(x(t), u(t))$. To ensure that $x(T) = p$, we can define a final position loss function $l_f$ on $x$, $l_f \in \mathscr{C}^2(\mathbb{R}^n \to \mathbb{R})$ and transform the problem to:

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & \int_{[0,T[} l(x(t), u(t))dt + l_f(x(T)) \\ \text{subject to} \quad & x(0) = x_0, \\ & \dot{x}(t) = f(x(t), u(t)). \end{aligned} \tag{1.4}$$

For example, $l_f(x) = \frac{w}{2}\|x(T) - p^*\|$. $w$ is a weight parameter (like $10^6$).

### 1.1.2 Discretization of a continuous problem

To be computable, we discretize the problem with an horizon $N \in \mathbb{N}^*$, let $h = \frac{T}{N}$, $i \in \{0, ..., N-1\}$:

$$\begin{aligned} x_{i+1} &= x_i + h\dot{x}_i \\ &= x_i + hf(x_i, u_i) \\ &= F(x_i, u_i) \end{aligned}$$

$$J(U) = \sum_{i=0}^{N-1} L(x_i, u_i) + L_T(x_N) \approx \int_{[0,T[} l(x(t), u(t))dt + l_f(x(T)) \tag{1.5}$$

where $F(x_i, u_i) = x + hf(x_i, u_i)$, $L(x_i, u_i) = h\, l(x_i, u_i)$, $L_T = l_f$.

Our goal is then to minimize the $J$ in $\ell_N^\infty \subset \mathbb{R}^N$ - a finite-dimensional vector space where the control sequence $U = (u_0, ..., u_{N-1})$ lies. Here, we transform a problem of infinite-dimensional control to a finite one.

## 1.2 Dynamic Programming

Notice that:

$$\begin{aligned} \min_U J(U) &= \min_{u_0} \min_{u_1} ... \min_{u_{N-1}} J(U) \\ &= \min_{u_0} L(x_0, u_0) \\ &+ (\min_{u_1} L(x_1, u_1) + ... \min_{u_{N-1}} L(x_{N-1}, u_{N-1}) + L_T(F(x_{N-1}, u_{N-1}))) \end{aligned} \tag{1.6}$$

Optimizing directly $J$ with respect to the whole $U$ may be difficult. Following the principle of dynamic programming, we can optimize only with respect to $u_i$ one by one if all future choices $u_{i+1}, .., u_{N-1}$ are optimal.

We define the optimal value function $V$ and a function $Q$ such that $V(x) = \min_u Q(x, u)$.

$$\begin{aligned} V_0(x_0) &= \min_{u_0} \min_{u_1} ... \min_{u_{N-1}} J(U) \\ &= \min_{u_0} L(x_0, u_0) + \underbrace{\min_{u_1} ... \min_{u_{N-1}} L(x_1, u_1) + ... + L_T(x_N)}_{V_1(x_i)} \end{aligned} \tag{1.7}$$

$$Q_0(x_0, u_0) = L(x_0, u_0) + \underbrace{\min_{u_1} ... \min_{u_{N-1}} L(x_1, u_1) + ... + L_T(x_N)}_{V_1(x_i)} \tag{1.8}$$

For all $i \in \{0, 1, ..., N-1\}$ iteratively:

$$\begin{aligned} V_i(x_i, u_i) &= \min_{u_i} L(x_i, u_i) + V_{i+1}(x_{i+1}) \\ V_N(x_N) &= L_T(x_N) \end{aligned} \tag{1.9}$$

$$\begin{aligned} Q_i(x_i, u_i) &= L(x_i, u_i) + V_{i+1}(x_{i+1}) \\ &= L(x_i, u_i) + V_{i+1}(f(x_i, u_i)) \\ &= L(x_i, u_i) + \min_{u_{i+1}} Q_{i+1}(f(x_i, u_i), u_{i+1}),\ i \leq N-2 \end{aligned} \tag{1.10}$$

where

$$V_i(x_i) = \min_{u_i} Q_i(x_i, u_i) \tag{1.11}$$

• $V_i$ is the optimal value of the cost in the sub-trajectory in the time interval $[t_i, T]$ which depends only on the initial position of the sub-trajectory $x_i$.
• $Q_i$ is the "raw form" of $V_i$ before minimizing with respect to $u_i$ considering that the future controls $u_{i+1}, .., u_{N-1}$ are always optimal.

**Remark.** *In practice, what we use in the DDP is the quadratic approximation of the function $Q$, and that is why we name it $Q$. If $f$ is linear and all cost functions are quadratic, then $Q$ and $V$ will all be quadratic and we can exactly find the $\min_{u_i} Q_i(x_i, u_i)$ for all $i$.*

# Chapter 2

# Differential Dynamic Programming

The computation of differentials (partial derivatives, gradient, and hessian...) is the core of optimization once the function we optimize is differentiable.

Remind that our goal is to optimize $J \in \mathscr{C}^2(\mathbb{R}^N \to \mathbb{R})$.

$$\begin{aligned} J(u_0, ...u_{N-1}) =& L(x_0, u_0) + L(F(x_0, u_0), u_1) + \\ & L(F(F(x_0, u_0), u_1), u_2) + \\ & L_T[F(F(...F(F(F(x_0, u_0), u_1), u_2), ..., u_{N-2}), u_{N-1})] \end{aligned} \tag{2.1}$$

$J$ is differentiable. However, the computation of differentials is not always easy because of the compositions of the "dynamic system" of $F$. Differential dynamic programming can avoid this hardcore work.

## 2.1 Decomposition to sub-problems

Remind our final goal is, for all $x_0$,

**Goal in the general form**

$$\begin{aligned} \text{find} \quad & U^* = (u_0^*, u_1^*, ..., u_{N-1}^*) \\ \text{subject to} \quad & x(0) = x_0, \\ & J(U^*) = \min_{u_0} \min_{u_1} ... \min_{u_{N-1}} J(U) \end{aligned} \tag{2.2}$$

which is equivalent to, for all $i \in \{0, 1, ..., N-1\}$,

**Goals in the DDP form**

$$\begin{aligned} \text{find} \quad & u_i^* \\ \text{subject to} \quad & x(t_i) = x_i, \\ & Q_i(x_i, u_i^*) = \min_{u_i} Q(x_i, u_i) = V_i(x_i) \end{aligned} \tag{2.3}$$

where $t_0 = 0,\ t_{i+1} = t_i + h$

We will first pre-compute what we need to find each "minimum" of $Q_i$ (their gradients and hessians) from $N-1$ to 0 which is called a *Backward Pass*. Once we have these information from differentials, we will compute the "minimum" from 0 to $N$ which is called a *Forward Pass*.

The raison why we first go from future to present is the convenience of computing of differentials to find the minimum. However, we cannot find them immediately in the future: we need to wait the time goes by to compute the minimum from present one by one.

**Remark.** *We don't search the exact global minimum and we don't even get a local minimum by doing backward and forward pass if the functions $Q_i$ are not quadratic or linear in one iteration. However, we will use the principle of* Sequential Quadratic Programming *to make an approximation of the problem as an LQR to compute a descent direction and converge* [1] *to a local minimum.*

## 2.2 Linear Quadratic Regulator (LQR)

If $L, L_T$ are quadratic and $F$ is linear, then the DDP is a *Linear Quadratic Regulator*. All $Q$ and $V$ are quadratic, so all $\approx$ will be $=$ in this section and we can resolve the problem in one iteration.

The LQR is only one method to solve a particular *Quadratic Programming Problem* with *Linear Equality Constraints* which exploits the structure of the problem to solve the problem efficiently:

$$\begin{aligned} \underset{x\in\ell^\infty_{N+1}, u\in\ell^\infty_N}{\text{minimize}} \quad & J(x,u) = \sum_{i=0}^{N-1} L(x_i,u_i) + L_T(x_N) \\ \text{subject to} \quad & x(0) = x_0, \\ & x_{i+1} = F(x_i,u_i)\ \forall i \in [0\,..\,N-1] \end{aligned} \tag{2.4}$$

In this section, we assume that the ongoing loss $L$ with respect to each state is the same. However, LQR works also in the case we have different quadratic $L_i$ with respect to $(x_i, u_i)$.

### 2.2.1 Backward pass: Optimal step, Q and its derivatives

For some fixed $x_i, u_i$

$$V_i(x_i) = \min_u Q_i(x_i,u) = Q_i(x_i,u_i) + \min_{\delta u}[Q_i(x_i,u_i+\delta u) - Q_i(x_i,u_i)] \tag{2.5}$$

We approximate $Q_i$ quadratically in practice, because *doing global optimization of a function without special properties is hopeless*[2], and approximating the function in higher orders Taylor series is computationally expensive:

$$Q_i(x_i+\delta_x, u_i+\delta_u) - Q_i(x_i,u_i) \approx DQ_i(x_i,u_i)(\delta_x,\delta_u) + \frac{1}{2}D^2Q_i(x_i,u_i)(\delta_x,\delta_u)(\delta_x,\delta_u) \tag{2.6}$$

What we need is then to find the optimal $\delta_u^*$ minimizing $Q_i(x_i+\delta_x, u_i+\delta_u)$. **For all** $x_i, u_i, \delta_x$

$$\begin{aligned} \delta_u^*(i) &:= \underset{\delta_u(i)}{\arg\min}\, Q_i(x_i+\delta_x(i), u_i+\delta_u(i)) \\ \delta_u^*(i) &= -(D^2_{uu}Q_i(x_i,u_i))^{-1}(\nabla_u Q_i(x_i,u_i) + D^2_{ux}Q_i(x_i,u_i)\delta_x) \end{aligned} \tag{2.7}$$

[1] there is some theoretical subtleties in the roll-out (DDP line search) phase. But there are theoretical convergence guarantees on classical line search. DDP line search strategies work in practice better and respect the equality constraint of dynamics

[2] by Francis Bach, 2019

By abuse of notation, we note

$$\bullet \quad \delta_u^* = -Q_{uu}^{-1}(Q_u + Q_{ux}\delta_x) \tag{2.8}$$

with the open-loop term:

$$k = -Q_{uu}^{-1}Q_u \tag{2.9}$$

and the feedback gain term:

$$K = -Q_{uu}^{-1}Q_{ux} \tag{2.10}$$

So

$$\delta_u^* = k + K\delta_x \tag{2.11}$$

To compute the best variation of control variable $\delta_u^*$, we need to know the differential of $Q$:

$$DQ_i(x_i, u_i) = DL(x_i, u_i) + \underline{DV_{i+1}}(x_{i+1}) \circ DF(x, u) \tag{2.12}$$

where the unknown is underlined. And the hessian which can be ignored in iLQR but not in DDP [3]

$$\begin{aligned} D^2Q_i(x_i, u_i)(e_k)(e_l) = {} & D^2L(x_i, u_i)(e_k)(e_l) + \\ & \underline{DV_{i+1}}(x_{i+1})\big(D^2F(x_i, u_i)(e_k)(e_l)\big) + \\ & \underline{D^2V_{i+1}}(x_{i+1})\big(DF(x_i, u_i)(e_k)\big)\big(DF(x_i, u_i)(e_l)\big) \end{aligned} \tag{2.13}$$

[4] where $e_k, e_l$ are vectors of canonical basis. We need use them as canonical variations to compute the Hessian matrix. In a less rigorous form:

$$\begin{aligned} Q_u &= L_u + \underline{V_x'}\, F_u \quad {}^{5} \\ Q_{uu} &= L_{uu} + \underline{V_x'} \cdot F_{uu} + F_u^T \underline{V_{xx}'} F_u \\ Q_{ux} &= L_{ux} + \underline{V_x'} \cdot F_{ux} + F_u^T \underline{V_{xx}'} F_x \end{aligned} \tag{2.14}$$

where the $\cdot$ denote the tensor dot product. [6]

As the underline information is from the future, we need the backward pass.

### 2.2.2 Forward pass: V and its derivatives

In each backward pass, we compute the first and second order differentials (≈gradient and hessian) of each $V_i$ and we use this information to find the quadratically approximate minimum of each $V_i$ (thus $J$).

If $i = N$, $V_N = L_T$. If $i \in [0\,..\,N-1]$, for all $\underline{\text{fixed } u_i}$, by 2.5,

$$V_i(x_i) = \min_u Q_i(x_i, u) \approx Q_i(x_i, u_i) - \frac{1}{2} D_u Q_i(x_i, u_i)(D_{uu}^2 Q_i(x_i, u_i))^{-1} \nabla_u Q_i(x_i, u_i) \tag{2.15}$$

In fact:

$$V_i(x_i) = \min_u Q_i(x_i, u) = Q_i(x_i, u_i + \delta_u^*(i)) \tag{2.16}$$

[3] [?]

[4] To prove this, use the theorem in the reminder of Differential calculus

[4] In the community, we use the prime notation to mean the "next"

[6] It is not a real tensor dot product and $F_{uu}, F_{ux}$ are neither real tensors defined in Wikipedia. It is a 3-dimensional array, but we don't need this term because $F$ is supposed to be linear: $D^2F = 0$.

Notice that here $\delta_u^*(i) = k(i) = -Q_{uu}^{-1}Q_u$. So

$$\begin{aligned} V &\approx Q + Q_u^T\delta_u^* + \frac{1}{2}{\delta_u^*}^T Q_{uu}\delta_u^* \\ &= Q - \frac{1}{2}Q_u^T Q_{uu}^{-1}Q_u \end{aligned} \tag{2.17}$$

Remark that $V = Q + \frac{1}{2}Q_u^T\delta_u^*$ which is like a gradient descent.

$$DV_i(x_i) = D_xQ_i(x_i,u_i) - D_uQ_i(x_i,u_i) \circ [D_{uu}^2Q_i(x_i,u_i)]^{-1} \circ D_{ux}^2Q_i(x_i,u_i) \tag{2.18}$$

$$D^2V_i(x_i) = D_{xx}^2Q_i(x_i,u_i) - D_{xu}^2Q_i(x_i,u_i) \circ [D_{uu}^2Q_i(x_i,u_i)]^{-1} \circ D_{ux}^2Q_i(x_i,u_i) \tag{2.19}$$

which are

$$\begin{aligned} V_x &= Q_x - Q_uQ_{uu}^{-1}Q_{ux} &&= Q_x + Q_uK \\ V_{xx} &= Q_{xx} - Q_{xu}Q_{uu}^{-1}Q_{ux} &&= Q_{xx} + Q_{xu}K \end{aligned} \tag{2.20}$$

Also

$$V_x = Q_x + kQ_{ux} \tag{2.21}$$

**Remark.** *As we do a quadratic approximation, the higher order ($> 2$) differentials are supposed to be null. So $Q_{uu}^{-1}$ is constant.*

### 2.2.3 Roll-out

Once we have the $\delta_u^* = (\delta_u(i)^*)_{i\in[0..N-1]}$, we "integrate" $\delta_u^*$ to obtain $u^* := u + \delta_u^*$. Then we compute $x^*$ recursively by doing the equation in Euler Schema:

$$\begin{aligned} x_0^* &= x_0, \\ x_{i+1}^* &= F(x_i^*, u_i^*)\ \forall i \in [0\,..\,N-1] \end{aligned} \tag{2.22}$$

## 2.3 Feasible and non-feasible trajectories

It can happen that there are some "gaps" between the next predicted state node $x_{i+1}^- = F(x_i,u_i)$ and the real $x_{i+1}$ such that $x_{i+1} = F(x_i,u_i) + \epsilon_{i+1}\ \forall i \in [0\,..\,N-1]$. The sequence of states will be

$$\begin{array}{ccc} x_0 & \xrightarrow{u_0} & x_1^- \\ & & \downarrow \epsilon_1 \\ & & x_1 \xrightarrow{u_1} x_2^- ... \end{array}$$

When all gaps $\epsilon$ are 0, the trajectory is *feasible*.

### 2.3.1 Non-feasible start in LQR

The chain of $Q, V$ will be changed: $\forall i \in [0\,..\,N-1]$

$$Q_i(x_i,u_i) = L_i(x_i,u_i) + V_{i+1}(\underbrace{f(x_i,u_i)}_{x_{i+1}^-}) \tag{2.23}$$

The schema in the interior loop will be

$$\begin{array}{c} ... \uparrow [V_{N-1}(x_{N-1})] \xleftarrow{2.20} [Q_{N-1}(x_{N-1}, u_{N-1})] \xleftarrow{2.14} [V_N(x_N^-)] \\ \uparrow 2.25 \\ [V_N(x_N)] \end{array}$$

[...] represents the first and second order differentials of ..., not the function value. For all $i \in [1\,..\,N]$,

$$\begin{aligned} V^i(\overbrace{x_i^- + \epsilon_i}^{x_i}) &= V^i(x_i^-) + V_x^i(x_i^-) \cdot \epsilon_i + \frac{1}{2}\epsilon_i^T V_{xx}^i \epsilon_i \\ &= \text{constant}(x_i^-) + V_x^i(x_i^-) \cdot x_i + \frac{1}{2}x_i^T V_{xx}^i x_i - x_i^T V_{xx}^i x_i^- \end{aligned} \tag{2.24}$$

$$\begin{aligned} V_x^i(x_i) &= V_x^i(x_i^-) + x_i^T V_{xx}^i - {x_i^-}^T V_{xx}^i \\ &= V_x^i(x_i^-) + \epsilon_i^T V_{xx}^i \\ V_x^i(x_i^-) &= V_x^i(x_i) - \epsilon_i^T V_{xx}^i \end{aligned} \tag{2.25}$$

Once all $Q_u, Q_{uu}, Q_{ux}$ are computed, we have $\delta_u$ i.e. $k, K$. Then either we can compute a new feasible trajectory with the classical full-step LQR roll-out, or we can perform a partial step to have another non feasible trajectory with a smaller gaps[7] *with respect to* the LQR linear dynamics model.

## 2.4 Algorithms

In this section, we consider the DDP as a *Non-Linear Optimization Problem* with respect to the state sequence $x \in \ell_{N+1}^\infty, u \in \ell_N^\infty$:

$$\begin{aligned} \underset{x \in \ell_{N+1}^\infty, u \in \ell_N^\infty}{\text{minimize}} \quad & J(x,u) = \sum_{i=0}^{N-1} L(x_i, u_i) + L_T(x_N) \\ \text{subject to} \quad & x(0) = x_0, \\ & x_{i+1} = F(x_i, u_i)\ \forall i \in [0\,..\,N-1] \end{aligned} \tag{2.26}$$

### 2.4.1 Sequential LQR Programming

Let $\bar{L}, \bar{L}_T$ be some quadratic approximation functions of $L, L_T$, and $\bar{F}$ be some linear approximation function of $F$. We repeat the following procedure until the criteria of convergence is satisfied.

**Approximative LQR problem**

$$\begin{aligned} \underset{x \in \ell_{N+1}^\infty, u \in \ell_N^\infty}{\text{minimize}} \quad & J(x,u) = \sum_{i=0}^{N-1} \bar{L}_i(x_i, u_i) + \bar{L}_T(x_N) \\ \text{subject to} \quad & x(0) = x_0, \\ & x_{i+1} = \bar{F}_i(x_i, u_i)\ \forall i \in [0\,..\,N-1] \end{aligned} \tag{2.27}$$

[7] the gaps could be bigger in a non LQR model. For example, in the each sequential LQR phase of DDP, the gaps of trajectory could be bigger with respect to the real non linear dynamics constraint but reduced in the LQR approximation model.

Given any *fixed* initial guess $x = (x_i)_{i\in[0..N]}, u = (u_i)_{i\in[0..N-1]}$, we define: for any $i \in [0\,..\,N-1]$,

$$\bar{L}_i(x,u) := L(x_i,u_i) + DL(x_i,u_i)(x-x_i,u-u_i) + \frac{1}{2}D^2L(x_i,u_i)(x-x_i,u-u_i)(x-x_i,u-u_i)$$

$$\bar{L}_i(x_i+\delta_x^i, u_i+\delta_u^i) = L(x_i,u_i) + \nabla L(x_i,u_i)(\delta_x^i,\delta_u^i) + \frac{1}{2}(\delta_x^i,\delta_u^i)\nabla^2 L(x_i,u_i)\begin{pmatrix}\delta_x^i\\ \delta_u^i\end{pmatrix} \tag{2.28}$$

I note $\delta_x(i), \delta_u(i)$ as $\delta_x^i, \delta_u^i$ to save space. For terminal loss:

$$\bar{L}_T(x) := L_T(x_N) + DL_T(x_N)(x-x_N) + \frac{1}{2}D^2L_T(x_N)(x-x_N)(x-x_N)$$

$$\bar{L}_T(x_N+\delta_x^N) = L_T(x_N) + \nabla L_T(x_N)\cdot\delta_x^N + \frac{1}{2}(\delta_x^N)^T\nabla^2 L_T(x_N)(\delta_x^N) \tag{2.29}$$

Note that $\sum_{i=0}^{N-1} L(x_i,u_i) + L_T(x_N)$ is constant, so what we optimize is

$$\sum_{i=0}^{N-1}[\nabla L(x_i,u_i)\cdot\begin{pmatrix}\delta_x^i\\ \delta_u^i\end{pmatrix} + \frac{1}{2}\begin{pmatrix}\delta_x^i\\ \delta_u^i\end{pmatrix}^T\nabla^2 L(x_i,u_i)\begin{pmatrix}\delta_x^i\\ \delta_u^i\end{pmatrix}] + \nabla L_T(x_N)\cdot\delta_x^N + \frac{1}{2}(\delta_x^N)^T\nabla^2 L_T(x_N)(\delta_x^N)$$

with respect to $\delta_x = (\delta_x^i)_{i\in[0..N]}$ and $\delta_u = (\delta_u^i)_{i\in[0..N-1]}$.

The linearization of $F$ is defined as:

$$\begin{aligned}\forall i \in [0\,..\,N-1],\ \bar{F}_i(x,u) &= F(x_i,u_i) + DF(x_i,u_i)(x-x_i,u-u_i)\\ \bar{F}_i(x_i+\delta_x^i,u_i+\delta_u^i) &= \underbrace{F(x_i,u_i)}_{x_{i+1}} + \nabla F(x_i,u_i)(\delta_x^i,\delta_u^i)\\ \bar{F}_i(\delta_x^i,\delta_u^i) &= \nabla F(x_i,u_i)(\delta_x^i,\delta_u^i)\end{aligned} \tag{2.30}$$

We expect $x_{i+1} + \delta_x^{i+1} = \bar{F}_i(x_i+\delta_x^i, u_i+\delta_u^i) + \epsilon_{i+1}$, so the equality constraint becomes

$$\begin{aligned}\delta_x^0 &= 0\\ \delta_x^{i+1} &= \bar{F}_i(\delta_x^i,\delta_u^i)\ \forall i \in [0\,..\,N-1]\end{aligned} \tag{2.31}$$

If the trajectory $(x,u)$ does not respect the linear dynamics constraint, there are gaps to be computed:

$$\forall i \in [1\,..\,N],\ \epsilon_i := x_i - \bar{F}_{i-1}(x_{i-1},u_{i-1}) \tag{2.32}$$

The gaps will not be changed if the optimization variable is $(delta_x, delta_u)$ instead of $(x,u)$. By **??**,

### 2.4.2 Interior loop (LQR)

If $Q, V \in \mathscr{C}^2$, then we can proceed the inner DDP loop as a *Linear Quadratic Regulator*: we linearize the dynamics function $f$ so that all $Q$ are quadratic to find minimum (Of course, all cost functions need to be quadratic which is usually the case). So all $\approx$ will be $=$ in this section. In a linear or quadratic case (for $Q_i$), practical results show that by doing the procedure described below only for one time, the trajectory will converge. It is comprehensible because each $\delta_u^*(i)$ is indeed the best

step of control variable and we indeed have a real minimum on each $Q_i(x_i, u_i)$. The approximative equality in 2.6 and 2.15 is nor more approximative but exact. The theory guarantees that we have a global minimum of $J$ by doing only one time backward and forward pass. In a non linear and non quadratic case, things are much more complicate: we need iterate this procedure many times until we have a satisfied convergence i.e. the trajectory of control is becoming to satisfy the hypothesis of dynamic programming and the $Q_i$ is becoming "locally quadratic".

First, we compute the $DV_N(x_N) = DL_{final}(x_N)$ which is directly computable. Once we have $DV_N(x_N)$, we have $DQ_{N-1}(x_{N-1}, u_{N-1})$. Then we can find $DV_{N-1}(x_{N-1})$, and $DQ_{N-2}(x_{N-2}, u_{N-2})$ by the same method. We iterate the process until we have all $DQ_i(x_i, u_i)$ from $N-1$ to 0.

Second, we compute the "best step of control variable" $\delta_u^* = (\delta_u^*(i))_{i\in\{0,...,N-1\}}$. If we are in a linear or quadratic case, we update the new control trajectory simply by $u = u+\delta_u$ and then compute the new and converging state trajectory by Euler method. In a non linear and non quadratic case, we need to use line search methods to find a best step in the direction of $\delta_u^*$ and regularization methods if the hessian matrices are not "regular".

### 2.4.3 Line search methods

### 2.4.4 Regularization methods

### Back to our simple exemple

Initially, we generate a original sequence of $x, u$: $x_i = 0,\ u_i = 0\ \forall i \in \{1, ..., N\}$. Restricting in the plan i.e. setting $x, u, p^* \in \mathbb{R}^2$, we have a better graphic representation. Our goal is to make our robot $x$ find a path to go to some place $p^*$ at the end of time $T$.

In the first iteration, the final on-going loss is clair:

$$V_N^*(x) = V_N(x) = \frac{w}{2}\|x - p^*\|^8$$

$$DV_N^*(x) = w(x^T - {p^*}^T) \tag{2.33}$$

$$D^2V_N^*(x) = wId_{2\times 2} \tag{2.34}$$

In our case,

$$l(x, u) = u^T u$$

$$Dl(x, u) = \begin{pmatrix} 0 & 0 & 2u_1 & 2u_2 \end{pmatrix} \tag{2.35}$$

$$D_x l(x, u) = 0,\ D_u l(x, u) = 2u \tag{2.36}$$

$$D^2 l(x, u) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.37}$$

$$D^2_{uu} l = 2Id_{2\times 2},\ D^2_{xx} l(x, u) = 0,\ D^2_{xu} l(x, u) = 0 \tag{2.38}$$

And

[8] if not mentioned, the norm is euclidean

$$Df(x,u) = \begin{pmatrix} 1 & 0 & d & 0 \\ 0 & 1 & 0 & d \end{pmatrix} \tag{2.39}$$

$$D^2 f(x,u) = 0 \tag{2.40}$$

## Reminder of Differential calculus

**Theorem 2.4.1** (Second order differential of composition of functions)**.**

$$D^2(f\circ g)(x)(h_1)(h_2) = Df(g(x))\big(D^2g(x)(h_1)(h_2)\big) + D^2f(g(x))\big(Dg(x)(h_1)\big)\big(Dg(x)(h_2)\big)$$

*Proof.* By the lemma below. □

For $f : X \to L(V,W),\ g : X \to L(U,V)$

**Definition 2.4.1** (generalized product of functions)**.**

$$(f\cdot g)(x) := f(x)\circ g(x)$$

**Lemma 2.4.2.**

$$D(f\cdot g)(x)(h) = f(x)\circ Dg(x)(h) + Df(x)(h)\circ g(x)$$

**Remark.** *Books remain silent about this theorem. Either not mentioned or in a false expression.*