SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Towards a mass-spring model for human motion capturing using optimal control

av

Da Wang

2012 - No 5

Towards a mass-spring model for human motion capturing using optimal control

Da Wang

Självständigt arbete i tillämpad matematik 30 högskolepoäng, avancerad nivå

Handledare: Mattias Sandberg

2012

Abstract

A reconstruction problem of human body motion can be treated as an optimal control problem. This thesis shows that such a problem can be solved by the Symplectic Pontryagin method. Existence of a solution, characteristic method and regularization are shown. In this work there is convergence of solution for some concrete example.

keywords: Symplectic Pontryagin method, Characteristic method, Hamilton Equation, Regularization

Contents

1 Description

Modeling and simulation of human body dynamics has wide use in research and industry. People realized a simple model is necessary for rapid simulations. The "Contact-aware Nonlinear Control of Dynamic Characters" [3] shows one way of doing this by emulating walking and running motions in rigid-body simulations. We'd like to control similarly as [3] but in another model. And this model will be used to capture a blocked motion.

In case of video recording body motion may be blocked by another object. If those frames of motion is important, people would like to reconstruct them. A numerical method of the body motion capture/reconstruction will be studied in this project.

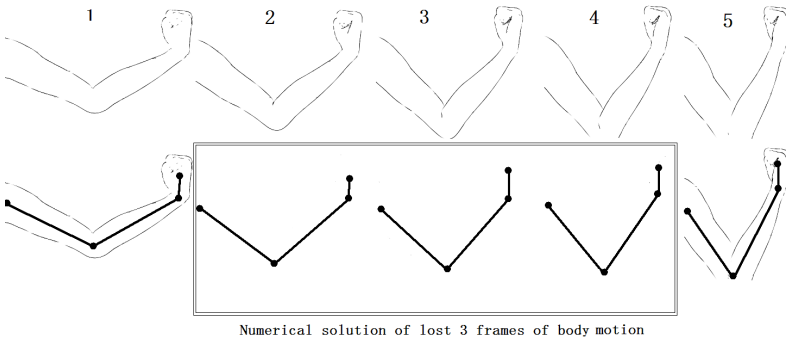


Figure 1: Reconstruction of body motion

This project uses a mass-spring model with control, considering them as a class of optimal control problems. We expect our solutions can track reference body motion in a simple and reasonable behavior. By doing this one can reconstruct missing frames.

From section 2 to section 5 we introduce the basic idea of optimal control, the characteristic method for optimal control problem and the regularization involved in discretization.

In section 6, we provide the detailed analysis of our model. In this section, we shall show that our model is solvable under some reasonable assumption.

Section 7 is a description of the numerical method we used in program, and in Section 8 we give the scheme of program.

In section 9, we show the analysis of solution for a 1-D example and a 2-D arm model.

Once a model is established, people can use this model to show blocked motion.

2 Introduction of optimal control

The following introduction use the idea in Evans [2]

Introduce a dynamic system with control:

$$\begin{cases} \dot{x}(t) &= f(x(t), \alpha(t)) \quad (T_0 < t < T) \\ x(T_0) &= X_0 \end{cases} \tag{1}$$

where $\mathbf{f} : \mathbb{R}^n \times A \longrightarrow \mathbb{R}^n$ is a given bounded, Lipschitz continuous function, and X_0 is given, and A is some given compact subset of $\mathbb{R}^m$. T_0 is initial time and X_0 is initial position. The function α appearing in (1) is a control, that is, some appropriate scheme for adjusting parameters from the set A as time evolves, thereby affecting the dynamic of the system modeled by (1).

Denote the set of admissible controls by $\mathcal{A}$,

$$\mathcal{A} = \{\alpha : [T_0, T] \longrightarrow A \ \mid \ \alpha \ is \ measurable\} \tag{2}$$

Further, we assume

$$\begin{aligned} |f(x,a)| &\leq C, \\ |f(x,a) - f(y,a)| &\leq C|x-y| \end{aligned} \tag{3}$$

for all $x, y \in \mathbb{R}^n, a \in A$, and a constant $C \geq 0$

Our goal is to find a control α^* which optimally steers the system. In order to define what "optimal" means, we introduce a cost criterion. Given $x \in \mathbb{R}^n$ and $0 \leq t \leq T$, for each admissible control $\alpha \in \mathcal{A}$ define the corresponding cost

$$C_{x,t}[\alpha(\cdot)] := \int_t^T h(x(s), \alpha(s))ds + g(x(T)) \tag{4}$$

where $x(\cdot)$ solves the ODE (1) and,

$$h : \mathbb{R}^n \times A \longrightarrow \mathbb{R}, \quad g : \mathbb{R}^n \longrightarrow \mathbb{R}$$

are given functions. We call h the running cost and g the terminal cost, and henceforth assume they are bounded and Liptchitz continuous:

$$\begin{aligned} |h(x,a)| &\leq C \\ |h(x,a) - h(y,a)| &\leq C|x-y| \end{aligned} \tag{5}$$

$$\begin{aligned} |g(x)| &\leq C \\ |g(x) - g(y)| &\leq C|x-y| \end{aligned} \tag{6}$$

for all $x, y \in \mathbb{R}^n, a \in A$, and some constant $C \geq 0$

3 Dynamic programming and the Hamilton-Jacobi-Bellman equation

We denote the infimum of cost along optimal trajectory from (x_0, t) by $u(x_0, t)$.

Definition 1 *A value function u is defined as:*

$$\begin{aligned} u(x,t) &:= \inf_{\alpha(\cdot)\in\mathcal{A}} C_{(x,t)}\Big[\alpha(\cdot)\Big] && (x\in\mathbb{R}^n, 0\le t\le T) \\ &= \inf_{\alpha(\cdot)\in\mathcal{A}} \Big\{ \int_t^T h\big(x(s),\alpha(s)\big)ds + g\big(x(T)\big)\Big\}. \end{aligned} \tag{7}$$

The $u(x,t)$ is the least cost starting at the position x and time t. It is well-defined under assumption (3),(5),(6). That is for any x and t, $u(x,t)$ is unique, but the corresponding optimal control α as well as the optimal path $x(t)$ may not be unique, or may not even exist.

Example 3.1 *Give a dynamic system:*

$$\dot{x} = \{-1,+1\} \quad (0\le t\le 1)$$

Cost functions are defined by: $g(x) = |x(1)|$ *and* $h = 0$. *And control* $\alpha := \dot{x} \in \{-1,+1\}$

If $x(0)\ge 1$ *then one can simply set* $\dot{x} = -1$ *to get the minimum. Similarly for* $x(0)\le -1$ *set* $\dot{x} = 1$.

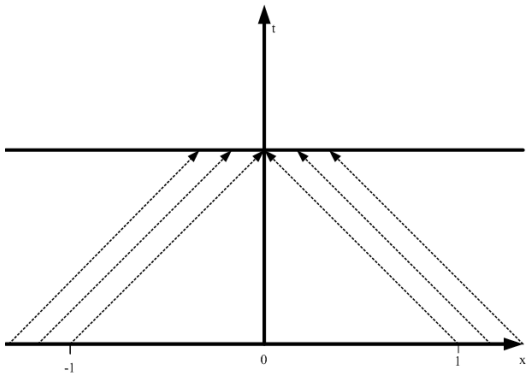


Figure 2: Optimal path for $x(0)\notin(-1,1)$

It is easy to see from the figure 2, for $x(0)\notin(-1,1)$ *the control path is unique.*

However when $x(0)\in(-1,1)$ *the situation becomes quite different. For instance let* $x(0) = 0.5$. *We could find many paths by using different controls, but all of them satisfy* $g(x) = 0$. *We show 4 different paths in figure* **??***:*

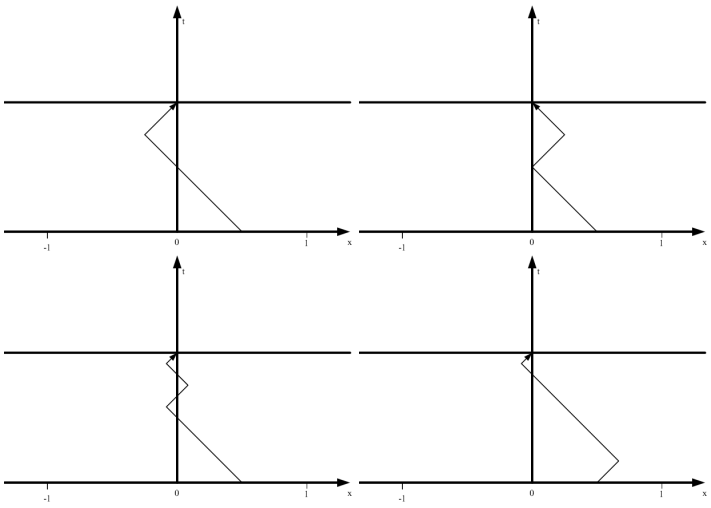


Figure 3: Some optimal paths for $x(0) \in (-1, 1)$

The value function for $T = 1$ can be written as:

$$u(x(0), 0) = \inf_{\alpha(\cdot)\in\mathcal{A}} \Big\{ g\big(x(1)\big) \Big\} = \begin{cases} -x(0) - 1 & x(0) \le -1 \\ x(0) - 1 & x(0) \ge 1 \\ 0 & others \end{cases} \tag{8}$$

We see that for any $x(0)$, the value function has an unique value, but there might be multiple optimal controls.

The solution of the optimal control problem in this project will be calculated by using Hamilton-Jacobi equation.

Definition 2 *The terminal-value problem for the Hamilton-Jacobi equation is*

$$\begin{cases} u_t + \mathbb{H}(Du, x) = 0 \quad in \quad \mathbb{R}^n \times (0,\ T) \\ u = g \quad on \quad \mathbb{R}^n \times (t = T) \end{cases} \tag{9}$$

where Du denote gradient of u.

A Hamiltonian in the optimal control problem is defined as:

$$\mathbb{H}(p, x) := \min_{a\in A} \Big\{ f(x, a) \cdot p + h(x, a) \Big\}, \quad \forall p, x \in \mathbb{R}^n \tag{10}$$

Let a_1 be the minimizer of $\min\limits_{a\in A} \Big\{ f(x, a) \cdot p_1 + h(x, a) \Big\}$ and a_2 be the minimizer of $\min\limits_{a\in A} \Big\{ f(x, a) \cdot p_2 + h(x, a) \Big\}$. That means:

$$\begin{array}{lll} f(x, a_1) \cdot p_1 + h(x, a_1) \le f(x, a) \cdot p_1 + h(x, a) & \forall a, & and \\ f(x, a_2) \cdot p_2 + h(x, a_2) \le f(x, a) \cdot p_2 + h(x, a) & \forall a & \end{array} \tag{11}$$

If $\mathbb{H}(x,p_1) > \mathbb{H}(x,p_2)$, we have:

$$\begin{aligned}
|\mathbb{H}(x,p_1) - \mathbb{H}(x,p_2)| &= \mathbb{H}(x,p_1) - \mathbb{H}(x,p_2) \\
&= \min_{a\in A}\Big\{f(x,a)\cdot p_1 + h(x,a)\Big\} - \min_{a\in A}\Big\{f(x,a)\cdot p_2 + h(x,a)\Big\} \\
(11) \leq\ & f(x,a_2)\cdot p_1 + h(x,a_2) - f(x,a_2)\cdot p_2 - h(x,a_2) \\
&= f(x,a_2)\cdot\big(p_1 - p_2\big) \\
&\leq \big|f(x,a_2)\big|\cdot|p_1 - p_2| \\
(3) \leq\ & C|p_1 - p_2|
\end{aligned}$$

Case $\mathbb{H}(x,p_1) < \mathbb{H}(x,p_2)$ analogous.

Let a_3 be the minimizer of $\min\limits_{a\in A}\Big\{f(x,a)\cdot p + h(x,a)\Big\}$ and a_4 be the minimizer of $\min\limits_{a\in A}\Big\{f(y,a)\cdot p + h(y,a)\Big\}$. That means:

$$\begin{aligned}
|\mathbb{H}(x,p) - \mathbb{H}(y,p)| &= \left|\min_{a\in A}\Big\{f(x,a)\cdot p + h(x,a)\Big\} - \min_{a\in A}\Big\{f(y,a)\cdot p + h(y,a)\Big\}\right| \\
&= \Big|f(x,a_3)\cdot p + h(x,a_3) - f(y,a_4)\cdot p - h(y,a_4)\Big| \\
&\leq \Big|\Big(f(x,a_3) - f(y,a_4)\Big)\cdot p\Big| + \Big|h(x,a_3) - h(y,a_4)\Big| \\
&\leq \Big|f(x,a_3) - f(y,a_4)\Big|\cdot|p| + \Big|h(x,a_3) - h(y,a_4)\Big| \\
(3)(5) \leq\ & C|p|\cdot|x-y| + C|x-y| \\
&= C(1+|p|)\cdot|x-y|
\end{aligned}$$

We conclude:

$$\begin{aligned}
|\mathbb{H}(x,p_1) - \mathbb{H}(x,p_2)| &\leq C|p_1 - p_2| \\
|\mathbb{H}(x,p) - \mathbb{H}(y,p)| &\leq C\big|1+|p|\big|\cdot|x-y|
\end{aligned} \tag{12}$$

for all $x, y, p_1, p_2 \in \mathbb{R}^n$, and some constant $C \geq 0$ (which is differ from the constant in inequalities (3) and (5)). These bounds will be used to prove uniqueness and for the error bound in latter sections.

The concept of viscosity solution is implemented for solving Hamilton-Jacobi-Bellman equation. That is defined as:

Definition 3 *A bounded, uniformly continuous function u is called a viscosity solution of the terminal-value problem (9) for the Hamiltonian-Jacobi equation if:*

(i) $u = g$ *on* $\mathbb{R}^n \times (t = T)$, *and*
(ii) for each $v \in C^\infty(\mathbb{R}^n \times (0, T))$

$$\begin{cases} if\ u - v\ has\ a\ local\ maximum\ at\ a\ point\ (x_0, t_0) \in \mathbb{R}^n \times (0, T) \\ then \\ v_t(x_0, t_0) + \mathbb{H}(Dv(x_0, t_0), x_0) \geq 0, \end{cases} \tag{13}$$

and

$$\begin{cases} if\ u - v\ has\ a\ local\ minimum\ at\ a\ point\ (x_0, t_0) \in \mathbb{R}^n \times (0, T) \\ then \\ v_t(x_0, t_0) + \mathbb{H}(Dv(x_0, t_0), x_0) \leq 0. \end{cases} \tag{14}$$

The value function $u(x, t)$ plays a quite important role for Hamilton-Jacobi equations. The following theorem reveals that is *the unique viscosity solution of a Hamilton-Jacobi equation.*

Theorem 3.1 *(A PDE for the value function) [2, page.557]*
The value function u is the unique viscosity solution of the following terminal-value problem for the Hamilton-Jacobi equation:

$$\begin{cases} u_t + \min\limits_{\alpha \in A} \Big\{ f(x, \alpha) \cdot Du + h(x, \alpha) \Big\} &= 0 \quad in \quad \mathbb{R}^n \times (0, \quad T) \\ u &= g \quad on \quad \mathbb{R}^n \times \{t = T\} \end{cases}$$

where the f is flux function of the dynamic system (1) and h is the running cost and g is the terminal cost.

We see that one character of value function: It is the solution of a Hamilton-Jacobi equation, and again it is unique.

4 The method of characteristics for Hamilton-Jacobi equations

4.1 Motivation of the method of characteristics

The part of motivation mostly follow ideas from Evans [2]

In mathematics, the method of characteristics is a technique for solving partial differential equations. The method is to reduce a partial differential equation to a family of ordinary differential equations along certain curves which is called characteristics. The solution can be integrated along characteristics from some initial data.
Let's start with a general PDE. Suppose that F, g, u are sufficiently smooth functions.

$$F(Du, u, x) = 0 \ \ in \ \ U \ \ (u \in \mathbb{R}, x \in \mathbb{R}^n, Du \in \mathbb{R}^n), \tag{15}$$

$$u = g \ \ on \ \ \Gamma \subseteq \partial U, \tag{16}$$

Assume now u is twice differentiable. Let

$$z(s) := u(x(s)) \quad where \quad x(s) = [x_1(s), x_2(s), \ ... \ , x_n(s)] \tag{17}$$

$$p(s) := Du(x(s)) \quad where \quad p_i(s) := \frac{\partial}{\partial x_i} u(x(s)) =: u_{x_i}(x(s)) \tag{18}$$

Then the partial derivative w.r.t s is:

$$\begin{aligned} \tfrac{d}{ds} p_i &= \tfrac{d}{ds} u_{x_i}(x(s)) \\ &= \sum_{j=1}^{n} \tfrac{\partial u_{x_i}}{\partial x_j} \tfrac{dx_j}{ds} \\ &= \sum_{j=1}^{n} (\tfrac{\partial^2 u}{\partial x_i \partial x_j} \tfrac{dx_j}{ds}), \end{aligned} \tag{19}$$

and

$$\begin{aligned} \tfrac{\partial}{\partial x_i} F \quad &= \quad \sum_{j=1}^{n} (\tfrac{\partial F}{\partial p_j} \tfrac{\partial p_j}{\partial x_i}) + \tfrac{\partial F}{\partial z} \tfrac{\partial z}{\partial x_i} + \tfrac{\partial F}{\partial x_i} = 0 \\ \Rightarrow \quad & \sum_{j=1}^{n} (\tfrac{\partial F}{\partial p_j} \tfrac{\partial^2 u}{\partial x_i \partial x_j}) + \tfrac{\partial F}{\partial z} \tfrac{\partial u}{\partial x_i} + \tfrac{\partial F}{\partial x_i} = 0 \\ \overset{(18)}{\Longrightarrow} \quad & \sum_{j=1}^{n} (\tfrac{\partial F}{\partial p_j} \tfrac{\partial^2 u}{\partial x_i \partial x_j}) + \tfrac{\partial F}{\partial z} p_i(s) + \tfrac{\partial F}{\partial x_i} = 0 \\ \Rightarrow \quad & \sum_{j=1}^{n} (\tfrac{\partial F}{\partial p_j} \tfrac{\partial^2 u}{\partial x_i \partial x_j}) = -\tfrac{\partial F}{\partial z} p_i(s) - \tfrac{\partial F}{\partial x_i} \end{aligned} \tag{20}$$

We wish to avoid the second order derivative term $\frac{\partial^2 u}{\partial x_i \partial x_j}$. Observe that this term exists in both (19) and (20), so we can eliminate $\frac{\partial^2 u}{\partial x_i \partial x_j}$ by letting $x(s)$ satisfy:

$$\frac{dx_j}{ds} = \frac{\partial F}{\partial p_j} \tag{21}$$

By doing this, we can rewrite (19) as:

$$\begin{aligned} \tfrac{dp_i}{ds} \quad &\overset{(19)}{=\!=} \quad \sum_{j=1}^{n} (\tfrac{\partial^2 u}{\partial x_i \partial x_j} \tfrac{dx_j}{ds}) \\ &\overset{(21)}{=\!=} \quad \sum_{j=1}^{n} (\tfrac{\partial^2 u}{\partial x_i \partial x_j} \tfrac{\partial F}{\partial p_j}) \\ &\overset{(20)}{=\!=} \quad -\tfrac{\partial F}{\partial z} p_i - \tfrac{\partial F}{x_i} \end{aligned} \tag{22}$$

Mean while:

$$\begin{array}{rcl} \frac{dz}{ds} & = & \sum_{j=1}^{n} (\frac{\partial u}{\partial x_j} \frac{dx_j}{ds}) \\ & \overset{(21)}{=\!=} & \sum_{j=1}^{n} p_j \frac{\partial F}{p_j} \end{array} \tag{23}$$

That means the solution of (15) satisfies:

$$\begin{cases} \dot{x}(s) & = & D_p F \\ \dot{p}(s) & = & -D_x F - D_z F p \\ \dot{z}(s) & = & D_p F p \end{cases} \tag{24}$$

where $\dot{()} := \frac{d}{ds}$, and D is gradient as it is defined before.

The PDE (15) can be solved using a system of ODE (24). For any initial value x_0 the solution of $s = T$ can be obtained from (24).

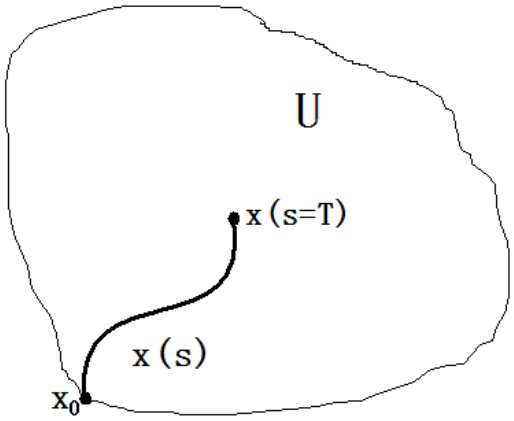


Figure 4: Characteristic path from $x_0 \in \partial U$ to the interier point x_T

Back to the general form Hamilton-Jacobi PDE. Recall that Hamilton-Jacobi PDE is defined as:

$$\begin{cases} u_t + \mathbb{H}(Du, x) & = & 0 \quad in \quad \mathbb{R}^n \times (0, \quad T) \\ u & = & g \quad on \quad \mathbb{R}^n \times (t = 0) \end{cases}$$

To implement the result we proved above, define a new PDE:

$$G(Du, u_t, u, x, t) := u_t + \mathbb{H}(Du, x) = 0 \tag{25}$$

where $Du := D_x u = [u_{x_1}, ..., u_{x_n}]$.

Define $p := D_x u$ and $z := u$ as before, also define $\lambda := [u_{x_1}, u_{x_2}, ..., u_{x_n}, u_t]$, and $y := [x_1, x_2, ..., x_n, t]$. Then we could rewrite equation in such form:

$$G(\lambda, z, y) = \lambda_{n+1} + \mathbb{H}(\lambda_1, ..., \lambda_n, x)$$

together with:

$$D_\lambda G = [D_p \mathbb{H}, 1], D_y G = [D_x \mathbb{H}, 0], D_z G = 0$$

We have the solution:

$$\begin{cases} \dot{x_i}(s) =: \dot{y_i}(s) &= \frac{\partial \mathbb{H}}{\partial p_i} & (i = 1, 2, ..., n), & \dot{y}_{n+1}(s) = 1 \\ \dot{\lambda_i}(s) &= -\frac{\partial \mathbb{H}}{\partial x_i} & (i = 1, 2, ..., n), & \dot{\lambda}_{n+1}(s) = 0 \\ \dot{z}(s) &= D_p \mathbb{H} \cdot p - \mathbb{H} \end{cases}$$

$\dot{y}_{n+1}(s) = dt/ds = 1$ says $t = s + c$ for some constant c, here we choose $t = s$.

In Evans [2] it shows the equation for $z(\cdot)$ is trivial, once solutions of $x(\cdot)$ and $\lambda(\cdot)$ have been found. Therefore we define Hamiltonian system:

$$\begin{cases} \dot{x_i}(t) &= \mathbb{H}_\lambda \\ \dot{\lambda}_i(t) &= -\mathbb{H}_x \end{cases} \tag{26}$$

A special case in this project is when the cost function h is time dependent because of time dependency of reference data. As a consequence the Hamiltonian is then time dependent. In this case the Hamilton-Jacobi PDE becomes:

$$\begin{cases} u_t + \mathbb{H}(Du, x, t) &= 0 \quad in \quad \mathbb{R}^n \times (0, T) \\ u &= g \quad on \quad \mathbb{R}^n \times (t = 0) \end{cases}$$

Use the similar procedure as before, set $p := D_x u$, $z := u$, $\lambda := [u_{x_1}, u_{x_2}, ..., u_{x_n}, u_t]$ and $y := [x_1, x_2, ..., x_n, t]$.

Then we rewrite G in the following form:

$$G(\lambda, z, y) = \lambda_{n+1} + \mathbb{H}(\lambda, y)$$

together with:

$$D_\lambda G = [D_p \mathbb{H}, 1], D_y G = D_y \mathbb{H}, D_z G = 0$$

Then we have quite similar solution:

$$\begin{cases} \dot{y_i}(s) &= \frac{\partial H}{\partial p_i} & (i = 1, 2, ..., n), & \dot{y}_{n+1}(s) = 1 \\ \dot{\lambda_i}(s) &= -\frac{\partial H}{\partial x_i} & (i = 1, 2, ..., n), & \dot{\lambda}_{n+1}(s) = -\frac{\partial H}{\partial t} \\ \dot{z}(s) &= D_p H \cdot p - H \end{cases}$$

Choose $t = s$ as before then one can use (26) to solve time dependent problem. It is work even though u is not differentiable, which shows in theorem (4.1)

4.2 Characteristic method for general case

The result above seems to provide a possible way. But notice that in our derivation, we assumed the value function u is twice differentiable. One problem is to know if u is twice differentiable before compute.

In the next theorem, we can use a weaker condition: For any $R > 0$ there exists C_R such that

$$|h(x,a) - h(y,a)| \leq C_R|x-y| \tag{27}$$

for all $x, y \in B_R \subset \mathbb{R}^n$, B_R is a ball with radius R, and $a \in A$.

Theorem 4.1 *[1] Let f satisfy (3), h satisfy(27), and assume that the control set A is compact. Suppose in addition that f_x, h_x exist and are continuous w.r.t. x, and $\mathbb{H}$ and its first order derivatives are locally Lipschitz continuous in $\mathbb{R}^n \times \mathbb{R}^n$. Given $(x,t) \in \mathbb{R}^n \times [0,T]$, let $\alpha : [t,T] \longrightarrow A$ be an optimal control with initial point (x,t) and let $y(\cdot) = y(\cdot; x, \alpha, t)$ be the corresponding optimal trajectory. For a given $g \in C_1$, let $\lambda : [t,T] \longrightarrow \mathbb{R}^n$ be the solution of the equation Then (y,λ) solves the system*

$$\begin{cases} y'(\tau) &= \mathbb{H}_\lambda(y(\tau),\lambda(\tau)) \\ \lambda'(\tau) &= -\mathbb{H}_x(y(\tau),\lambda(\tau)) \end{cases} \qquad s \in [t,T] \tag{28}$$

As a consequence, y, λ are of class C^1.

Note that it is assumed u is differentiable in the theorem, we don't need an assumption of the value function u.

5 Regularization of the Hamiltonian

In the case when the Hamiltonian is not differentiable. We can't use the Hamilton system (26) directly. Therefore one way to follow idea of the Hamilton system (26) is to construct a regularized Hamiltonian $\mathbb{H}^\delta$ which is a smooth function such that $\|\mathbb{H} - \mathbb{H}^\delta\|_{L^\infty(\mathbb{R}^d \times \mathbb{R}^d)} \leq \delta$ where the d here is dimension of variable space and duel variable space.

The Symplectic Pontryagin scheme should be introduced to solve the perturbed Hamilton system. This method involve the regularized Hamiltonian defined as follows:

Definition 4 *The time interval $[0,T]$ is split into N intervals uniformly with length $\Delta t = T/N$. X_0 is the initial state at $t = 0$. The Symplectic Pontryagin scheme is:*

$$\begin{array}{rcll} x_{n+1} &=& x_n + \Delta t \mathbb{H}^\delta_\lambda(x_n, \lambda_{n+1}), & n = 0,1,...,N-1 \\ x_0 &=& X_0 & \\ \lambda_n &=& \lambda_{n+1} + \Delta t \mathbb{H}^\delta_x(x_n, \lambda_{n+1}), & n = 0,1,...,N-1 \\ \lambda_N &=& Dg(x_N). & \end{array} \tag{29}$$

The $\mathbb{H}^\delta$ is regularized Hamiltonian which is smooth.

For regularized Hamiltonian and Symplectic Pontryagin scheme, define the regularized running cost h^δ and discretized value function:

Definition 5 *Regularized running cost*

$$h^\delta(x,\lambda) \quad = \quad \mathbb{H}^\delta(x,\lambda) - \lambda \cdot \mathbb{H}^\delta_\lambda(x,\lambda) \tag{30}$$

Definition 6 *Discretized value function*

$$\bar{u}(x_0,0) \quad = \quad \min_{\{x_n,\lambda_n\} \ solve \ (29)} \big(\sum_n h^\delta(x_n,\lambda_n)\Delta t + g(x_N)\big) \tag{31}$$

Theorem 5.1 *[4] Let $\mathbb{H}$ be a Hamiltonian which is concave in its second argument, and satisfies $|\mathbb{H}(x,\lambda_1)-\mathbb{H}(x,\lambda_2)| \le C_1|\lambda_1-\lambda_2|$ and $|\mathbb{H}(x,\lambda)-\mathbb{H}(y,\lambda)| \le C_2\big|1+|\lambda|\big|\cdot|x-y|$ for all $x,y,\lambda_1,\lambda_2 \in \mathbb{R}^n$. Let $g:\mathbb{R}^n \to \mathbb{R}$ be differentiable and satisfy $|Dg| \le C_3$, for all $x \in \mathbb{R}^n$. Let $\mathbb{H}^\delta$ be a regularized Hamiltonian which satisfies the same conditions, and which is continuously differentiable, and satisfies*

$$\|\mathbb{H}-\mathbb{H}^\delta\|_{L^\infty(\mathbb{R}^d\times\mathbb{R}^d)} \le \delta.$$

Let u be the solution of

$$\begin{cases} u_t + \mathbb{H}(Du,x) &= 0 \quad in \quad \mathbb{R}^n \times (0,T), \\ u &= g \quad on \quad \mathbb{R}^n \times (t=T), \end{cases},$$

and $\overline{u}$ is defined in (31), then

$$\begin{gathered} -\tfrac{C_1C_2T}{2}((e^{C_2T}-1)\Delta t + \Delta t^2) - \tfrac{C_1C_3}{2}(e^{C_2T}-1)\Delta t - T\delta \\ \le u(x_s,0) - \overline{u}(x_s,0) \le \\ \tfrac{1}{2}C_1C_2(C_3+1)e^{C_2T}T\Delta t + T\delta. \end{gathered}$$

Here C_1, C_2, C_3 are constants given in the assumption. □

This allows us to compute a discretized value function (31) by solving (29), and decrease Δt to make it converge to the solution of (2). One advantage of doing this is avoid involve a non-smooth Hamiltonian. We can use some iterative method.

We would like to introduce a lemma that is useful to check Lipschiz continuity:

Lemma 5.1 *Let $a(x_1,x_2,...,x_n)$ be a real function satisfies:*

$$\begin{gathered} |a(x_1^1,x_2,...,x_n) - a(x_1^2,x_2,...,x_n)| \le C_1|x_1^1-x_1^2| \\ |a(x_1,x_2^1,...,x_n) - a(x_1,x_2^2,...,x_n)| \le C_2|x_2^1-x_2^2| \\ \vdots \\ |a(x_1,x_2,...,x_n^1) - a(x_1,x_2,...,x_n^2)| \le C_n|x_n^1-x_n^2| \end{gathered}$$

Then $|a(x_1^1, x_2^1, ..., x_n^1) - a(x_1^2, x_2^2, ..., x_n^2)| \leq n\mathcal{C}\|\boldsymbol{x}^1 - \boldsymbol{x}^2\|$,
where $\mathcal{C} := \max\{C_1, C_2, ..., C_n\}$ *and* $\boldsymbol{x} := \{x_1, x_2, ..., x_n\}$.

Proof. If $n = 1$, there is nothing to prove.
In case of $n = 2$, then

$$\begin{aligned} |a(x_1^1, x_2) - a(x_1^2, x_2)| &\leq C_1|x_1^1 - x_1^2| \\ |a(x_1, x_2^1) - a(x_1, x_2^2)| &\leq C_2|x_2^1 - x_2^2| \end{aligned}$$

$$\begin{aligned} |a(x_1^1, x_2^1) - a(x_1^2, x_2^2)|^2 &\leq (C_2|x_2^1 - x_2^2| + C_1|x_1^1 - x_1^2|)^2 \\ &\leq (2\max\{C_1|x_1^1 - x_1^2|, C_2|x_2^1 - x_2^2|\})^2 \\ &\leq (2\max\{C_1, C_2\}t\max\{|x_2^1 - x_2^2|, |x_1^1 - x_1^2|\})^2 \\ &= 4\max\{C_1, C_2\}^2\max\{|x_2^1 - x_2^2|, |x_1^1 - x_1^2|\}^2 \\ &\leq 4\max\{C_1, C_2\}^2((x_1^1 - x_1^2)^2 + (x_2^1 - x_2^2)^2) \end{aligned}$$

$$\Longrightarrow |a(x_1^1, x_2^1) - a(x_1^2, x_2^2)| \leq 2\mathcal{C} \cdot \|\mathbf{x}^1 - \mathbf{x}^2\|$$

where $\mathcal{C} = \max\{C_1, C_2\}$

Assume that this apply for $n = k$, k is a positive integer. That is:

$$\begin{aligned} &|a(x_1^1, x_2^1, ..., x_k^1) - a(x_1^2, x_2^2, ..., x_k^2)| \\ &\quad \leq k\max\{C_1, C_2, ..., C_k\}\sqrt{(x_1^1 - x_1^2)^2 + (x_2^1 - x_2^2)^2 + ... + (x_k^1 - x_k^2)^2} \end{aligned}$$

Then for n=k+1,

$$\begin{aligned} &|a(x_1^1, x_2^1, ..., x_k^1, x_{k+1}^1) - a(x_1^2, x_2^2, ..., x_k^2, x_{k+1}^2)| \\ &\quad = |a(x_1^1, x_2^1, ..., x_k^1, x_{k+1}^1) - a(x_1^2, x_2^2, ..., x_k^2, x_{k+1}^1) \\ &\qquad + a(x_1^2, x_2^2, ..., x_k^2, x_{k+1}^1) - a(x_1^2, x_2^2, ..., x_k^2, x_{k+1}^2)| \\ &\quad \leq |a(x_1^1, x_2^1, ..., x_k^1, x_{k+1}^1) - a(x_1^2, x_2^2, ..., x_k^2, x_{k+1}^1)| \\ &\qquad + |a(x_1^2, x_2^2, ..., x_k^2, x_{k+1}^1) - a(x_1^2, x_2^2, ..., x_k^2, x_{k+1}^2)| \\ &\quad \leq k\max\{C_1, ..., C_k\}\sqrt{(x_1^1 - x_1^2)^2 + (x_2^1 - x_2^2)^2 + ... + (x_k^1 - x_k^2)^2} \\ &\qquad + C_{k+1}\sqrt{(x_{k+1}^1 - x_{k+1}^2)^2} \\ &\quad \leq k\max\{C_1, ..., C_k\}\sqrt{(x_1^1 - x_1^2)^2 + ... + (x_{k+1}^1 - x_{k+1}^2)^2} \\ &\qquad + C_{k+1}\sqrt{(x_1^1 - x_1^2)^2 + ... + (x_{k+1}^1 - x_{k+1}^2)^2} \\ &\quad \leq k\max\{C_1, ..., C_{k+1}\}\sqrt{(x_1^1 - x_1^2)^2 + ... + (x_{k+1}^1 - x_{k+1}^2)^2} \\ &\qquad + \max\{C_1, ..., C_{k+1}\}\sqrt{(x_1^1 - x_1^2)^2 + ... + (x_{k+1}^1 - x_{k+1}^2)^2} \\ &\quad = (k+1)\max\{C_1, ..., C_{k+1}\}\sqrt{(x_1^1 - x_1^2)^2 + ... + (x_{k+1}^1 - x_{k+1}^2)^2} \\ &\quad = (k+1)\mathcal{C}\|\mathbf{x}^1 - \mathbf{x}^2\| \end{aligned}$$

where $\mathcal{C} = \max\{C_1, ..., C_{k+1}\}$ □

Here we give an example to show how regularization works.

Example 5.1 *Here is an 1-D example that shows how regularization works. A mass m connected by a spring with spring constant k from the wall. Here the control is the rest length of the spring denoted by L ($L \in [L_-, L_+]$, $L_+ \geq L_- \geq 0$). The running cost is $h = (x_1 - x_r)^2$ where reference position x_r is given, and the terminal cost is $g = 0$. Try to minimize $\mathcal{J} = \int_0^T h ds$.*

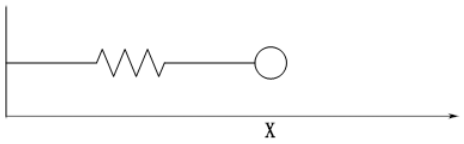


Figure 5: A simple mass-spring example

Denote the displacement of mass by x_1, and the velocity of mass by $\dot{x}_1 =: x_2$. According to Newton's 2nd law: $F = ma$, write down the dynamic system:

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = \begin{pmatrix} x_2 \\ k(L - x_1)/m \end{pmatrix} =: f(\mathbf{x}, L)$$

where $L \in [L_-, L_+] =: A$. Introduce dual variable $\lambda = [\lambda_1, \lambda_2]$, let the Hamiltonian be:

$$\begin{aligned} \mathbb{H} &= \min_{L \in A} \left\{ \lambda_1 x_1 + \lambda_2 \tfrac{k(L - x_1)}{m} + (x_1 - x_r)^2 \right\} \\ &= \min_{L \in A} \left\{ \tfrac{\lambda_2 k L}{m} \right\} + \lambda_1 x_1 - \tfrac{\lambda_2 k x_1}{m} + (x_1 - x_r)^2. \end{aligned}$$

Denote $S := \min_{L \in A} \{ \frac{\lambda_2 k L}{m} \}$ and $\mathcal{C} = \lambda_1 x_2 - \frac{\lambda_2 k x_1}{m} + (x_1 - x_r)^2$ for simplicity, then $\mathbb{H} = S + \mathcal{C}$.

Note that this is a linear control problem. S is not differentiable at $\lambda_2 = 0$, since

$$S = \begin{cases} \frac{\lambda_2 k}{m} L_+, & \lambda_2 < 0, \\ \\ \frac{\lambda_2 k}{m} L_-, & \lambda_2 \geq 0. \end{cases}$$

The Hamiltonian is not differentiable w.r.t. λ_2 at $\lambda_2 = 0$, Therefore we need to regularize the Hamiltonian. Note that we would like to solve the problem by using the Symplectic Pontryagin scheme in which the Newton's method will be used. This requires second order partial derivatives are continuous.

In this example, $\mathcal{C}$ is smooth already. The only thing left to do is trying to find regularized S^δ which is smooth. Because of $|\mathbb{H} - \mathbb{H}^\delta| = |S + \mathcal{C} - S^\delta - \mathcal{C}| = |S - S^\delta|$, construct a S^δ such that $|S - S^\delta| \leq \delta$ leads to $|\mathbb{H} - \mathbb{H}^\delta| \leq \delta$ automatically. Partial derivatives S_{x_1}, S_{x_2}, S_{λ_1} are zero, only S_{λ_2} need to be smoothed.

The expression of S_{λ_2} is:

$$S_{\lambda_2} = \begin{cases} \frac{kL_+}{m}, & \lambda_2 < 0, \\ \\ \frac{kL_-}{m}, & \lambda_2 > 0. \end{cases}$$

We want to construct a smooth function $S^\gamma_{\lambda_2}$. And the corresponding Hamiltonian $\mathbb{H}^\delta$ should satisfy $\|\mathbb{H} - \mathbb{H}^\delta\|_{L^\infty(\mathbb{R}^d \times \mathbb{R}^d)} \leq \delta$. One option is to let $S^\gamma_{\lambda_2} := \frac{k(L_- - L_+)}{2m} tanh(\frac{\lambda_2}{\gamma}) + \frac{k(L_- + L_+)}{2m}$. By this setting, $S^\gamma_{\lambda_2} \longrightarrow S_{\lambda_2}$ as $\gamma \longrightarrow 0$, for $\lambda_2 \neq 0$

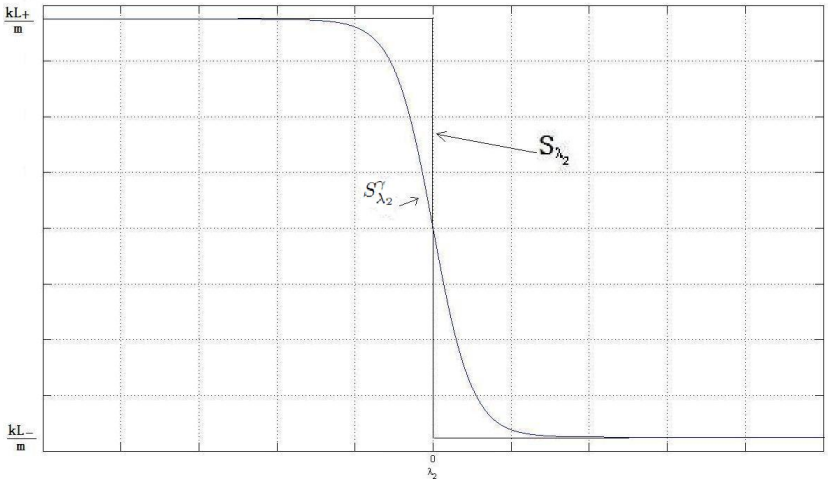


Figure 6: $S^\gamma_{\lambda_2}$ and S_{λ_2}

The primitive S^γ, of $S^\gamma_{\lambda_2}$, can be obtained by integrating:

$$S^\gamma = \frac{\frac{kL_-}{m} - \frac{kL_+}{m}}{2} \cdot \gamma \cdot \ln(\cosh \frac{\lambda_2}{\gamma}) + \frac{\frac{kL_-}{m} + \frac{kL_+}{m}}{2} \lambda_2 + C$$

for some constant C.

We want S^γ and S to coincide at $\lambda_2 = 0$, and therefore choose $C = 0$. Then

$$|S - S^\gamma| = \begin{cases} \left| \frac{kL_+}{m} \lambda_2 - \frac{\frac{kL_-}{m} - \frac{kL_+}{m}}{2} \cdot \gamma \cdot \ln \cosh \frac{\lambda_2}{\gamma} - \frac{\frac{kL_-}{m} + \frac{kL_+}{m}}{2} \lambda_2 \right|, & \lambda_2 < 0 \\ \\ \left| \frac{kL_-}{m} \lambda_2 - \frac{\frac{kL_-}{m} - \frac{kL_+}{m}}{2} \cdot \gamma \cdot \ln \cosh \frac{\lambda_2}{\gamma} - \frac{\frac{kL_-}{m} + \frac{kL_+}{m}}{2} \lambda_2 \right|, & \lambda_2 \geq 0 \end{cases}$$

From figure 6, it is clear that the maximum difference appears at $\lambda_2 = \pm\infty$ since $S^\gamma_\lambda > S_\lambda$ for $\lambda > 0$ and $S^\gamma_\lambda < S_\lambda$ for $\lambda < 0$. If $\lim_{\lambda_2 \to \pm\infty} |S - S^\gamma| \leq C_\gamma$ for some constant C_γ, then $|S - S^\gamma| \leq C_\gamma$ for all λ_2.

For $\lambda \geq 0$:

$$\begin{aligned}
|S - S^\gamma| &= |\tfrac{kL_+}{m}\lambda_2 - \tfrac{\frac{kL_-}{m} - \frac{kL_+}{m}}{2} \cdot \gamma \cdot \ln(\cosh \tfrac{\lambda_2}{\gamma}) - \tfrac{\frac{kL_-}{m} + \frac{kL_+}{m}}{2}\lambda_2| \\
&= |\tfrac{\frac{kL_-}{m} - \frac{kL_+}{m}}{2}\lambda_2 - \tfrac{\frac{kL_-}{m} - \frac{kL_+}{m}}{2} \cdot \gamma \cdot \ln(\cosh \tfrac{\lambda_2}{\gamma})| \\
&= \tfrac{\frac{kL_+}{m} - \frac{kL_-}{m}}{2}|\lambda_2 - \gamma \cdot \ln(\cosh \tfrac{\lambda_2}{\gamma})| \\
&= \tfrac{\frac{kL_+}{m} - \frac{kL_-}{m}}{2} \cdot \gamma \cdot |\ln 2 - \ln(e^{-2\frac{\lambda_2}{\gamma}} + 1)| \\
\lim_{\lambda_2 \to +\infty} |S - S^\gamma| &= \tfrac{\frac{kL_+}{m} - \frac{kL_-}{m}}{2} \cdot \ln 2 \cdot \gamma
\end{aligned}$$

Likewise $\lim_{\lambda_2 \to -\infty} |S - S^\gamma| = \frac{\frac{kL_+}{m} - \frac{kL_-}{m}}{2} \cdot \ln 2 \cdot \gamma$.

By analyzing above, define

$$\delta = \frac{\frac{kL_+}{m} - \frac{kL_-}{m}}{2} \cdot \ln 2 \cdot \gamma \tag{32}$$

We can rewrite S^δ as:

$$S^\delta := \frac{\delta}{\ln 2} \ln\Big(\cosh(\frac{k(L_+ - L_-)\ln 2}{2m\delta}\lambda_2)\Big) + \frac{\frac{kL_-}{m} + \frac{kL_+}{m}}{2}\lambda_2 \tag{33}$$

then $|S - S^\delta| \leq \delta$. Consequently, $|\mathbb{H} - \mathbb{H}^\delta| \leq \delta$.

To implement Theorem 5.1, a check for Lipschiz continuity is needed. Assume that $x_r > 0$, and set $k = m = 1$, $L_- = 1$, $L_+ = 2$ for simplicity, the Hamiltonian becomes:

$$\mathbb{H} = \begin{cases} 2\lambda_2 + \lambda_1 x_2 - \lambda_2 x_1 + (x_1 - x_r)^2, & \lambda_2 < 0 \\ \lambda_2 + \lambda_1 x_2 - \lambda_2 x_1 + (x_1 - x_r)^2, & \lambda_2 \geq 0 \end{cases}$$

We expect every element of the variable x to be bounded. A detailed proof will be provided in the appendix. Here we assume $\max\{|x_1|, |x_2|\} \leq K$.

To check $\mathbb{H}$ satisfies assumptions in the theorem 5.1, we start by computing the Lipschitz constant of each direction. And implement lemma (5.1) prove Lipschitz continuity of $\mathbb{H}$.

(i) x_1 direction:

$\mathbb{H}$ is smooth along the direction of x_1, we can simply compute derivative instead.

$$\begin{aligned}
Lip_{x_1} &= \big|\tfrac{\partial \mathbb{H}}{\partial x_1}\big| = \big| - \lambda_2 + 2(x_1 - x_r)\big| \\
&\leq |\lambda_2| + 2(x_r + K)
\end{aligned}$$

Then we have:

$$\begin{aligned} |\mathbb{H}(x_1^1, x_2, \lambda_1, \lambda_2) - \mathbb{H}(x_1^2, x_2, \lambda_1, \lambda_2)| &\leq |2(x_r + K) + \lambda_2| \cdot |x_1^1 - x_1^2| \\ &\leq |2(x_r + K) + \lambda| \cdot |x_1^1 - x_1^2| \\ &\leq \max\{1, |2(x_r + K)|\}(1 + |\lambda|)|x_1^1 - x_1^2| \end{aligned}$$

(ii) x_2 direction:

Along x_2 direction, find Lipschitz constant as follow:

$$Lip_{x_2} = \left|\frac{\partial \mathbb{H}}{\partial x_2}\right| = \left|\lambda_1\right|$$

Then

$$\begin{aligned} |\mathbb{H}(x_1, x_2^1, \lambda_1, \lambda_2) - \mathbb{H}(x_1, x_2^2, \lambda_1, \lambda_2)| &= \left|\lambda_1(x_2^1 - x_2^2)\right| \\ &= \left|\lambda_1\right| \cdot \left|(x_2^1 - x_2^2)\right| \\ &\leq (1 + |\lambda_1|) \cdot \left|(x_2^1 - x_2^2)\right| \\ &\leq (1 + |\lambda|) \cdot \left|(x_2^1 - x_2^2)\right| \end{aligned}$$

(iii) λ_1 direction:

Next we compute Lipschitz constant along λ_1:

$$Lip_{\lambda_1} = \left|\frac{\partial \mathbb{H}}{\partial \lambda_1}\right| = \left|x_2\right|,$$

and

$$|\mathbb{H}(x_1, x_2, \lambda_1^1, \lambda_2) - \mathbb{H}(x_1, x_2, \lambda_1^2, \lambda_2)| = |x_2| \cdot |\lambda_1^1 - \lambda_1^2| \leq K|\lambda_1^1 - \lambda_1^2|$$

(iv) λ_2 direction:

Along λ_2 direction, we should find Lipschitz from two parts. Firstly derivative is:

$$Lip_{\lambda_2} = \left|\frac{\partial \mathbb{H}}{\partial \lambda_2}\right| = \begin{cases} |2 - x_1| \leq |1 + K|, & \lambda_2 < 0 \\ |1 - x_1| \leq |2 + K|, & \lambda_2 \geq 0 \end{cases}$$

So we conclude that:

$$|\mathbb{H}(x_1, x_2, \lambda_1, \lambda_2^1) - \mathbb{H}(x_1, x_2, \lambda_1, \lambda_2^2)| \leq (2 + K) \cdot |\lambda_2^1 - \lambda_2^2|$$

(v) Finally, implement the lemma 5.1, we have:

$$|\mathbb{H}(x^1, \lambda) - H(x^2, \lambda)| \leq C_1(1 + \|\lambda\|) \cdot \|x^1 - x^2\|$$

$$|\mathbb{H}(x, \lambda^1) - H(x, \lambda^2)| \leq C_2|\lambda^1 - \lambda^2|$$

where $C_1 = 2\max\{1, |2(x_r+K)|\}$ and $C_2 = 4+2K$. If we use (33) and regularize $\mathbb{H}$, then the discrete value function by using $\mathbb{H}^\delta$ satisfies:

$$-T\max\Big\{1, |2(x_r+K)|\Big\}\cdot(4+2K)\Big((e^{(4+2K)T}-1)\Delta t+\Delta t^2\Big)-T\delta$$

$$\leq u(x_0,0)-\overline{u}(x_0,0)\leq$$

$$\max\Big\{1, |2(x_r+K)|\Big\}\cdot(4+2K)e^{(4+2K)T}\cdot T\Delta t+T\delta$$

6 Motion capturing model

6.1 Model description

In this project, we try to build up a spring-mass model to simulate human body motion. The model consists of several nodes and springs. Each node has its own weight, and is connected by at least one spring. Every spring has no mass, and composed by one ideal spring is connected with a damper. The spring constant and damping coefficient are κ and ξ. A spring unit is shown in Fig.7:

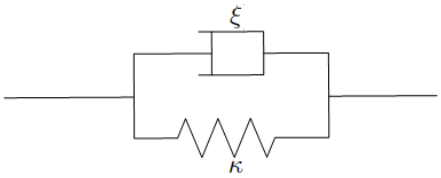


Figure 7: A spring unit

For simplicity, we henceforth in all figures let only the spring part denote the whole "spring + damper" unit.

Consider an arbitrary grid. Only restriction of its structure is: there is no separate parts. That means that every two nodes are connected via some springs and intermediate nodes.

We are interested in if any model can be solved by the method we have introduced. To answer this question, firstly let's study local structure of the grid.

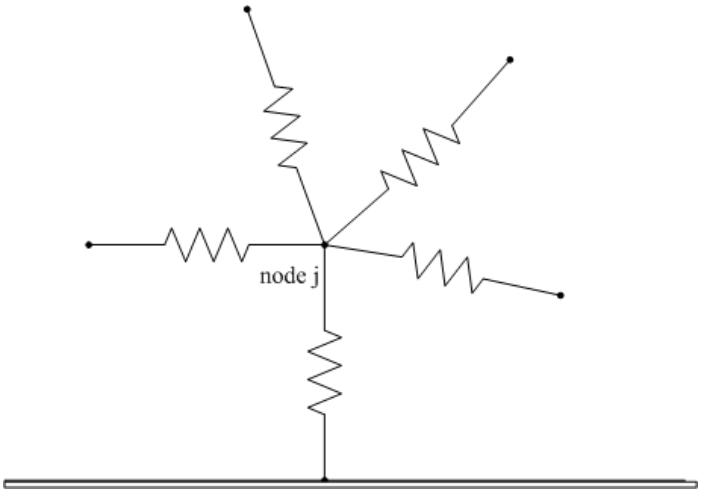


Figure 8: Part of grid around node j

As the Fig.8 shows above, for any node j in the grid, it will be connected by springs with its neighbor nodes. Part of these nodes are connected with other nodes, and the rest part may fixed on certain position. Let us define the set of nodes that are connected with node j:

$$\Omega_j = \{i \mid i \textit{ and } j \textit{ are connected via a spring}\}. \tag{34}$$

Also, define the set of connected pairs:

$$\mathcal{S} = \{(i,j) \mid i \textit{ and } j \textit{ are connected via some springs}\}. \tag{35}$$

For each $(i,j) \in \mathcal{S}$, let $\vec{\mathcal{V}}_{ij}$ denote the unit vector starting from node i and pointing to node j. In this project, we consider Euclidean space of dimension 1 or 2; we denote dimension as d. There are M particles and n springs. By this setting, we define the *position and velocity* vectors and corresponding dual variables:

$$z = \begin{pmatrix} z_1 \\ \vdots \\ z_M \end{pmatrix}, \quad \textit{and} \quad \rho = \begin{pmatrix} \rho_1 \\ \vdots \\ \rho_M \end{pmatrix}, \tag{36}$$

where

$$z_j = \begin{pmatrix} x_{1,j} \\ \vdots \\ x_{d,j} \\ v_{1,j} \\ \vdots \\ v_{d,j} \end{pmatrix}, \quad \rho_j = \begin{pmatrix} \lambda_{1,j} \\ \vdots \\ \lambda_{d,j} \\ \zeta_{1,j} \\ \vdots \\ \zeta_{d,j} \end{pmatrix}.$$

Denote the position of node j along dimension i as $x_{i,j}$, and the velocity of particle j along dimension i as $v_{i,j}$. We would like denote the position dual variable as $\lambda_{i,j}$, and the velocity dual variable as $\zeta_{i,j}$, therefore introduce ρ_j as the collection of all dual variables related to node j.

Let $\{e_1, e_2, ..., e_d\}$ be a standard basis for this Euclidean space. Then the total force applied on node j is:

$$F_j = \sum_{i \in \Omega_j} \vec{\mathcal{V}}_{ij} \Big[\kappa_{ij}(L_{ij} - l_{ij}) - \xi_{ij} \sum_{k=1}^{d} \Big((v_{k,j} - v_{k,i}) \vec{\mathcal{V}}_{ij} \cdot e_k \Big) \Big],$$

where κ_{ij} and ξ_{ij} are spring constant and damping coefficient of the spring connecting node i and j, say, $\mathcal{L}_{ij}$. L_{ij} is the rest length of $\mathcal{L}_{ij}$ which is our control variable. l_{ij} is the current length of $\mathcal{L}_{ij}$ which is equal to: $\sqrt{\sum_{k=1}^{d}(x_{k,j} - x_{k,i})^2}$.

Note that for every (i, j), if node i and node j coincide, then l_{ij} is equal to zero. In that case, vector $\vec{\mathcal{V}}_{ij}$ has no unique direction which is not what we want.

Therefore we assume for all i, j satisfies:

$$l_{ij} = \sum_{k=1}^{d} |x_{k,j} - x_{k,i}| \geq \varepsilon > 0. \tag{37}$$

Decompose total force along each direction $\overline{k}$:

$$F_{\overline{k},j} = \sum_{i \in \Omega_j} \left\{ \Big[\kappa_{ij}(L_{ij} - l_{ij}) - \xi_{ij} \cdot \big(\frac{\sum_{k=1}^{d}(v_{k,j} - v_{k,i})(x_{k,j} - x_{k,i})}{l_{ij}} \big) \Big] \cdot \frac{x_{\overline{k},j} - x_{\overline{k},i}}{l_{ij}} \right\}$$

Remark: l_{ij} can be very small. In this sense $F_{k,j}$ is not well behaved. In this project we try to avoid this by a careful design.

In our project, a quadratic running cost is selected such that

$$h = \sum_{j=1}^{m} \sum_{k=1}^{d} (x_{k,j} - X_{k,j})^2$$

where X is a reference position which is a vector function of time.

It is possible to extend theorem 5.1 to time dependent cases, but in this project we present the theory for time independent cases only, even thogh numerical tests are performed with time dependent Hamiltonians as well.

Let each particle have mass m_i. Every spring has a control L_{ij}. Write dynamic

function as:

$$\dot{z} = f(z, L) =$$

$$\begin{pmatrix} v_{1,1} \\ \vdots \\ v_{d,1} \\ \vdots \\ v_{1,M} \\ \vdots \\ v_{d,M} \\ \sum_{i\in\Omega_1}\{[\kappa_{i1}(L_{i1}-l_{i1}) - \xi_{i1}(\frac{\sum_{k=1}^{d}(v_{k,1}-v_{k,i})(x_{k,1}-x_{k,i})}{l_{i1}})]\frac{x_{1,1}-x_{1,i}}{l_{i1}m_1}\} \\ \vdots \\ \sum_{i\in\Omega_1}\{[\kappa_{i1}(L_{i1}-l_{i1}) - \xi_{i1}(\frac{\sum_{k=1}^{d}(v_{k,1}-v_{k,i})(x_{k,1}-x_{k,i})}{l_{i1}})]\frac{x_{d,1}-x_{d,i}}{l_{i1}m_1}\} \\ \vdots \\ \sum_{i\in\Omega_M}\{[\kappa_{iM}(L_{iM}-l_{iM}) - \xi_{iM}(\frac{\sum_{k=1}^{d}(v_{k,M}-v_{k,i})(x_{k,M}-x_{k,i})}{l_{iM}})]\frac{x_{1,M}-x_{1,i}}{l_{iM}m_M}\} \\ \vdots \\ \sum_{i\in\Omega_M}\{[\kappa_{iM}(L_{iM}-l_{iM}) - \xi_{iM}(\frac{\sum_{k=1}^{d}(v_{k,M}-v_{k,i})(x_{k,M}-x_{k,i})}{l_{iM}})]\frac{x_{d,M}-x_{d,i}}{l_{iM}m_M}\} \end{pmatrix} \tag{38}$$

6.2 Lipschitz constants of the Hamiltonian

In this project we have proved that the variable z and the dual variable ρ are bounded. And we have evaluated Lipschitz constants of the Hamiltonian w.r.t. the primal and dual variables which are defined in theorem 5.1. All these can be found in the appendix A.

Assume that there exists bounds κ_{max}, ξ_{max}, L_{min}, L_{max}, m_{min} and m_{max} such that for every (i,j) $\kappa_{ij} \le \kappa_{max}$, $\xi_{ij} \le D_{max}$, $L_{min} \le L_{ij}^{-} \le L_{ij}^{+} \le L_{max}$ and $m_{min} < m_i < m_{max}$. One can prove:

$$\begin{aligned} C_1 &\le \max\Big\{(M-1)\big(\tfrac{L_{max}\cdot\kappa_{max}}{m_{min}} + \tfrac{2\kappa_{max}}{m_{min}}x_{max} + \tfrac{8d\xi_{max}x_{max}^2 v_{max}}{m_{min}\varepsilon^2}\big), v_{max}\Big\}, \\ C_2 &\le \max\Bigg\{(M-1)\Big(\tfrac{2\kappa_{max}L_{max}}{\varepsilon^3 m_{min}} + \tfrac{2\kappa_{max}L_{max}d}{\varepsilon m_{min}} + \tfrac{2\kappa_{max}}{m_{min}} \\ &\quad + \tfrac{8d\xi_{max}x_{max}}{m_{min}}\big(\tfrac{v_{max}}{\varepsilon^2} + \tfrac{8dx_{max}^2 v_{max}}{\varepsilon^4}\big) + \tfrac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2}\Big), \\ &\quad 2x_{max}, (m-1)\tfrac{8d\xi_{max}x_{max}^2}{m_{min}\varepsilon^2} + 1\Bigg\}. \end{aligned} \tag{39}$$

The two parameters in dominator are critical for convergence. During the simulation, when two particles are close to each other, that is ε is very small then C_1 and C_2 are huge, the convergence became hard to archived. The similar effect exist when mass also.

Note that this Lipschitz continuity holds in the set $\Big\{(z,\rho) \quad | \quad \varepsilon \leq \|z\| \leq z_{max}, \quad \|\rho\| \leq \rho_{max}\Big\}$. Hence it is not possible to use these bounds in theorem 5.1 directly. However, it could indicate the effect of different parameters in our control problem on the overall error.

6.3 Regularization of project model

The last part left to show is how to correctly regularize the Hamiltonian in this project. The control in this model is the rest length of springs. For instance, a spring connecting node i and j has the rest length L_{ij} which is bounded by L_{ij}^- and L_{ij}^+, that is $L_{ij} \in [L_{ij}^-, L_{ij}^+] =: \mathbb{A}$. For each spring, L_{ij}^* is the minimizer of a problem. We further define the domain of rest length all n springs as $\mathbb{A}^n$, and vector of all rest length L, where $L \in \mathbb{A}^n$. Because the definition of the Hamiltonian in this project, L_{ij} is chosen as either L_{ij}^- or L_{ij}^+ as optimum control depending on the sign of the coefficient, when the Hamiltonian is computed.

With definitions (10), (36) and (38), the Hamiltonian of a general spring-damper system can be written as:

$$\mathbb{H} = \min_{L\in\mathbb{A}^n} \{\sum_{j=1}^{M}(\sum_{k=1}^{d}(\lambda_{k,j}v_{k,j}) + \sum_{k=1}^{d}(\zeta_{k,j}F_{k,j})) + h\}$$

Expand the formula and gather terms related to controls of springs, we have the control related part of the Hamiltonian:

$$\begin{aligned}(I) \quad &:= \quad \sum_{k=1}^{d}\sum_{j=1}^{M}\sum_{i\in\Omega_j} \frac{\zeta_{k,j}\kappa_{ij}L_{ij}^*(x_{k,j}-x_{k,i})}{m_j\sqrt{\sum_{s=1}^{d}(x_{s,j}-x_{s,i})^2}} \\ &= \quad \sum_{\substack{(i,j)\in\mathcal{S}\\ i<j}}\sum_{k=1}^{d}\left(\frac{\left(\frac{\zeta_{k,j}}{m_j}-\frac{\zeta_{k,i}}{m_i}\right)\kappa_{ij}L_{ij}^*(x_{k,j}-x_{k,i})}{\sqrt{\sum_{s=1}^{d}(x_{s,j}-x_{s,i})^2}}\right)\end{aligned}$$

That means we need only regularize this part when we regularize the Hamiltonian $\mathbb{H}$.

Define the scalar function

$$P_{ij} := \frac{\sum_{k=1}^{d}(\frac{\zeta_{k,j}}{m_j}-\frac{\zeta_{k,i}}{m_i})(x_{k,i}-x_{k,j})\kappa_{ij}}{\sqrt{\sum_{k=1}^{d}(x_{k,i}-x_{k,j})^2}},$$

and

$$S_{ij} := S_{ij}(P_{ij}) = \min_{L_{ij}}\{P_{ij}L_{ij}\}$$

for all $(i,j) \in \mathcal{S}$.

Then

$$(I) = \sum_{\substack{(i,j)\in\mathcal{S}\\ i<j}} S_{ij}$$

where S_{ij} is a piecewise linear function:

$$S_{ij} = \begin{cases} P_{ij}L_{ij}^{+}, & P_{ij} < 0 \\ P_{ij}L_{ij}^{-}, & P_{ij} \geq 0 \end{cases}$$

The function P_{ij} is already smooth for all variables, we only need to consider how to regularize S_{ij} as a function of P_{ij}. Because we will use Newton's method in which second order derivatives are needed, we need to regularize the derivative of S_{ij} w.r.t. P_{ij}, which is:

$$\frac{dS_{ij}}{dP_{ij}} = \begin{cases} L_{ij}^{+}, & P_{ij} \leq 0 \\ L_{ij}^{-}, & P_{ij} > 0 \end{cases}$$

Mimic what we did in section 5, define:

$$\frac{dS_{ij}^{\gamma}}{dP_{ij}} = \frac{L_{ij}^{-} - L_{ij}^{+}}{2} tanh(\frac{P_{ij}}{\gamma}) + \frac{L_{ij}^{-} + L_{ij}^{+}}{2}$$

for all i, j.

We obtain primitive S_{ij}^{γ} by integration:

$$S_{ij}^{\gamma} = \frac{L_{ij}^{-} - L_{ij}^{+}}{2}\gamma \ln\left(cosh\Big(\frac{P_{ij}}{\gamma}\Big)\right) + \frac{L_{ij}^{-} + L_{ij}^{+}}{2}P_{ij}$$

. We choose integral constant is equal to zero here.

For $P_{ij} > 0$,

$$\begin{aligned} \frac{dS_{ij}^{\gamma}}{dP_{ij}} - \frac{dS_{ij}}{dP_{ij}} &= \frac{L_{ij}^{-} - L_{ij}^{+}}{2} tanh(\frac{P_{ij}}{\gamma}) + \frac{L_{ij}^{-} + L_{ij}^{+}}{2} - L_{ij}^{-} \\ &= \Big(1 - tanh(\frac{P_{ij}}{\gamma})\Big)\frac{L_{ij}^{+} - L_{ij}^{-}}{2} \\ &> 0 \end{aligned}$$

That means S_{ij}^{γ} grows faster that S_{ij}. By knowledge of $S_{ij}^{\gamma} = S_{ij} = 0$ at $P_{ij} = 0$, we conclude that $S_{ij}^{\gamma} > S_{ij}$ for all $P_{ij} > 0$.

Similarly we obtain $S_{ij}^{\gamma} < S_{ij}$ for $P_{ij} < 0$. We conclude that the maximum of $|S_{ij} - S_{ij}^{\gamma}|$ appears at $P_{ij} = \pm\infty$.

Fig.9 shows the regularized S^{δ} converges to S as $\delta \to 0$.

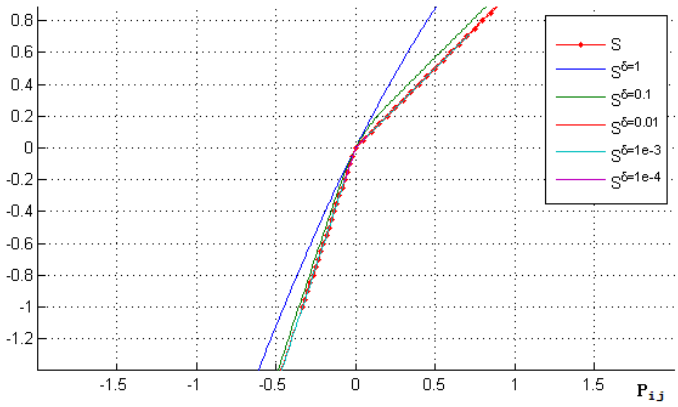


Figure 9: S^δ converge to S when $\delta \longrightarrow 0$

Follow the same idea as the example 5.1, if $P_{ij} > 0$:

$$\lim_{P_{ij} \longrightarrow +\infty} \left| S_{ij} - S_{ij}^\gamma \right| = \frac{L_{ij}^+ - L_{ij}^-}{2} \cdot \ln 2 \cdot \gamma.$$

Similarly for $P_{ij} \leq 0$:

$$\lim_{P_{ij} \longrightarrow +\infty} \left| S_{ij} - S_{ij}^\gamma \right| = \frac{L_{ij}^+ - L_{ij}^-}{2} \cdot \ln 2 \cdot \gamma.$$

We conclude: $\lim_{P_{ij} \longrightarrow +\infty} \left| S_{ij} - S_{ij}^\gamma \right| \leq \frac{L_{ij}^+ - L_{ij}^-}{2} \cdot \ln 2 \cdot \gamma$. That means: $\left| S_{ij} - S_{ij}^\gamma \right| \leq \frac{L_{ij}^+ - L_{ij}^-}{2} \cdot \ln 2 \cdot \gamma$.

As a consequence:

$$\begin{aligned} \left| H - H^\gamma \right| &= \Big| \sum_{\substack{(i,j) \in \mathcal{S} \\ i<j}} (S_{ij} - S_{ij}^\gamma) \Big| \\ &\leq \sum_{\substack{(i,j) \in \mathcal{S} \\ i<j}} \left| S_{ij} - S_{ij}^\gamma \right| \\ &\leq \sum_{\substack{(i,j) \in \mathcal{S} \\ i<j}} \frac{L_{ij}^+ - L_{ij}^-}{2} \cdot \ln 2 \cdot \gamma \\ &\leq \sum_{\substack{(i,j) \in \mathcal{S} \\ i<j}} \frac{L_{max} - L_{min}}{2} \cdot \ln 2 \cdot \gamma \\ &= \frac{1}{2} \ln 2 \cdot n \cdot (L_{max} - L_{min}) \cdot \gamma \end{aligned}$$

Let's denote $\delta := \frac{1}{2} \ln 2 \cdot n \cdot (L_{max} - L_{min}) \cdot \gamma$.

6.4 Analysis of discrete solution

Once a model is built up with reference to some external data, the terminal time, masses, coefficients of springs and topology of grid are fixed. What can be modified are Δt and δ. In the scheme of this project, we change the time interval when we try to find a good initial guess.

We must be ware that even though a unique solution can be obtained, it is "unique" in sense of value function $u(x_0, 0)$. The corresponding optimal solution and optimal control may not be unique.

7 Solution of model

There are at least two possible ways to solve discretized Hamilton system with Symplectic Pontryagin scheme. The one is use fixed-point like iteration, the other way is using Newton's method.

7.1 Fixed-point like method

To implemented the Symplectic Pontryagin scheme defined in (29) with the fixed-point like iteration method start with an initial guess:

$$\overline{x} = \{x_0 = X_0, x_1, ..., x_N\}$$

.

---

1) Guess an initial guess $\overline{x} = \{x_0, x_1, ..., x_N\}$
2) Compute $\lambda_N = g'(x_N)$
3) Compute $\lambda_n = \lambda_{n+1} + \Delta t \ \mathbb{H}_x^\delta(x_n, \lambda_{n+1})$ backward from $n = N-1$ to $n = 0$
4) Compute $x_{n+1} = x_n + \Delta t \ \mathbb{H}_\lambda^\delta(x_n, \lambda_{n+1})$ forward from $n = 0$ to $n = N-1$
5) If the norm of difference between old $\overline{x}$ and newly updated $\overline{x}$ is larger than a predefined tolerance, then go to 2).

The Fig.10 demonstrates how it works.

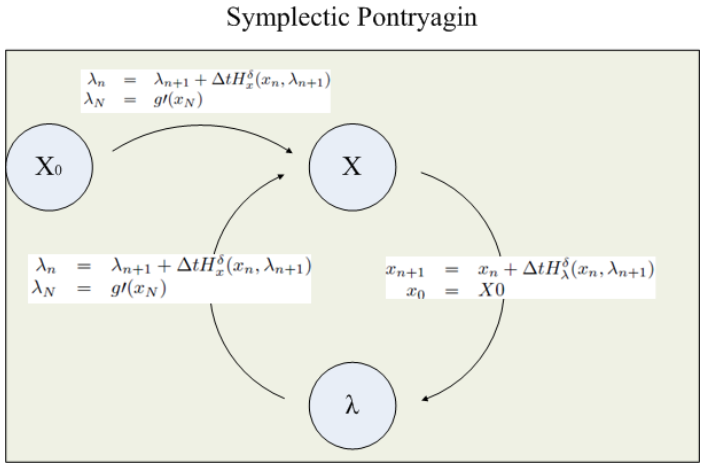


Figure 10: The fixed-point like iteration method

This method is easy to implement. We introduce damped viscosity to get to converge easier.

7.2 Newton's method

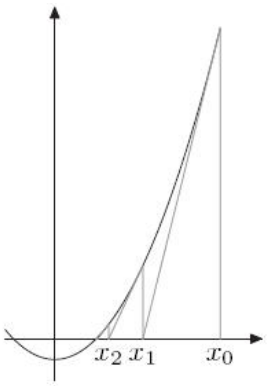
Newton's method is used to solve equations of the form $F(x) = 0$. The idea of the method is as follows: one starts with an initial guess which is reasonably close to the true root, then the function is approximated by its tangent plane (in the 1-D case its tangent line), and one computes the x-intercept of this tangent plane (which is easily done with elementary algebra). This x-intercept will typically be a better approximation to the function's root than the original guess, and the method can be iterated.

In 1-D, denote the initial guess by x_0, the new x-intercept by x_1, we have:

$$(x_0 - x_1)F'(x_0) = F(x_0) - 0$$

If $F'(x_0) \neq 0$ we can write:

$$x_1 = x_0 - \frac{F(x_0)}{F'(x_0)}.$$



For higher dimensional cases it is still true. We would like write this into iteration mode:

$$\mathbf{x}_n = \mathbf{x}_{n-1} - J_{n-1}^{-1} F(\mathbf{x}_{n-1}) \qquad (40)$$

where J_{n-1} is Jacobian matrix of $F(\mathbf{x})$ at point $\mathbf{x}_{n-1}$.

It can be proved that under mild conditions Newton's method does converge if the initial point is closed enough to the solution. And Newton's method has quadratic convergence, which is a big advantage compared to the fixed-point like iterative method.

The disadvantage of Newton's method in this project is that an initial guess is not only needed for the variable z but also for the dual variable ρ. To find a good initial guess for z is not big problem, but for ρ it is.

Combining the two methods, we use the fixed-point method to compute ρ from initial guess of z, after that use $y := \{z, \rho\}$ as the variable of in Newton's method. The whole process could be shown as follows:

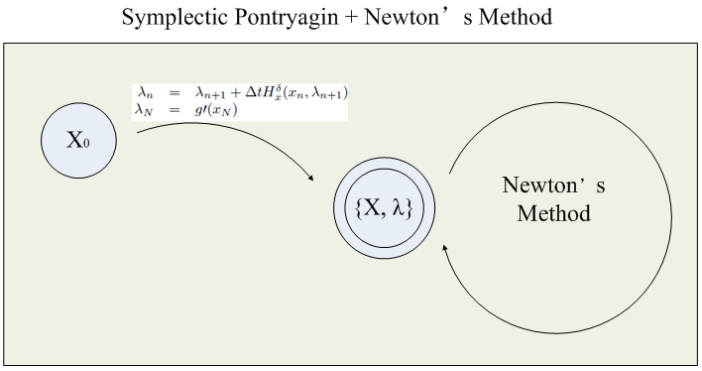


Figure 11: Symplectic Pontryagin method

7.3 Implementation of Newton's method

The discrete system of equations (29) can be written in this form:

$$F(z,\rho) := \begin{pmatrix} z_1^1 - z_1^0 - \Delta t \frac{\partial \mathbb{H}^\delta}{\partial \rho_1}(z_1^0, \rho_1^1) \\ \vdots \\ z_1^N - z_1^{N-1} - \Delta t \frac{\partial \mathbb{H}^\delta}{\partial \rho_1}(z_1^{N-1}, \rho_1^N) \\ \vdots \\ z_{2Md}^1 - z_{2Md}^0 - \Delta t \frac{\partial \mathbb{H}^\delta}{\partial \rho_{2Md}}(z_{2Md}^0, \rho_{2Md}^1) \\ \vdots \\ z_{2Md}^N - z_{2Md}^{N-1} - \Delta t \frac{\partial \mathbb{H}^\delta}{\partial \rho_{2Md}}(z_{2Md}^{N-1}, \rho_{2Md}^N) \\ \vdots \\ \rho_1^0 - \rho_1^1 - \Delta t \frac{\partial \mathbb{H}^\delta}{\partial z_1}(z_1^0, \rho_1^1) \\ \vdots \\ \rho_1^{N-1} - \rho_1^N - \Delta t \frac{\partial \mathbb{H}^\delta}{z_1}(\partial z_1^{N-1}, \rho_1^N) \\ \vdots \\ \rho_{2Md}^0 - \rho_{2Md}^1 - \Delta t \frac{\partial \mathbb{H}^\delta}{\partial z_{2Md}}(z_{2Md}^0, \rho_{2Md}^1) \\ \vdots \\ \rho_{2Md}^{N-1} - \rho_{2Md}^N - \Delta t \frac{\partial \mathbb{H}^\delta}{\partial z_{2Md}}(z_{2Md}^{N-1}, \rho_{2Md}^N) \end{pmatrix} = 0. \tag{41}$$

Let

$$y := \begin{pmatrix} z_1^1 \\ \vdots \\ z_1^N \\ \vdots \\ z_{2Md}^1 \\ \vdots \\ z_{2Md}^N \\ \\ \rho_1^0 \\ \vdots \\ \rho_1^{N-1} \\ \vdots \\ \rho_{2Md}^0 \\ \vdots \\ \rho_{2Md}^{N-1} \end{pmatrix}, \tag{42}$$

and we want to solve equation F(y)=0.

By this setting, the jacobian of $F(y)$ w.r.t. y is:

$$J := \begin{pmatrix} \frac{\partial F_1}{\partial y_1} & \cdots & \frac{\partial F_1}{\partial y_{2MdN}} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_{2MdN}}{\partial y_1} & \cdots & \frac{\partial F_{2MdN}}{\partial y_{2MdN}} \end{pmatrix}. \tag{43}$$

Remark: When calculating F and J, the value of ρ^N is a function of z and ρ. It should not be added to the vector of unknowns, y.

8 Scheme of program

8.1 Parameter approaching

In this project the rest length of springs, spring constants, damping coefficients and the function of reference data are assumed to be known. We know from the analysis in section 6 that lowering the ratio of κ_{max}/m_{min} and ξ_{max}/m_{min} will help solution convergence. We also found that long distance of reference route is also difficult for convergence.

In many cases large constants and long paths have to be involved. Our strategy is to start with small constants and short enough paths which is easier for handle. If a convergent solution is obtained, we can prolong the path or enlarge constants and use Newton's method again.

Consider a general constant κ in the model. A simulation starts with a first guess κ_0 which is rather small compared to κ. Once the solution converges, run another simulation with κ_1 which is larger than κ_0. If we fail to get a convergent solution, then try a smaller one (but still larger than κ_0). In this way, we will get κ_0, κ_1, κ_2 $\cdots$. If for each step, there will be a convergent solution, then we will get a solution for κ. The Fig.12 illustrates how the constant increasing progress works.

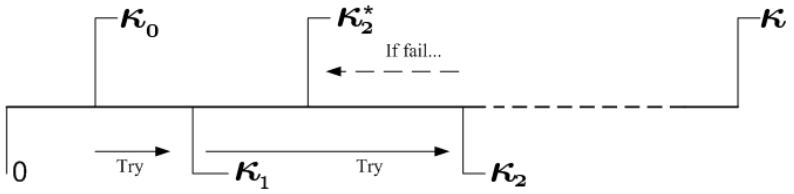


Figure 12: A approaching process of κ

For multiple constants, we can either increase all constants in the same time and try to get convergence, or we can increase the constants one by one.

The route approaching is quite similar. The computing is started with a short route from the start point. If we get a convergent solution, then we try longer route. We show an example of route approaching for one node:

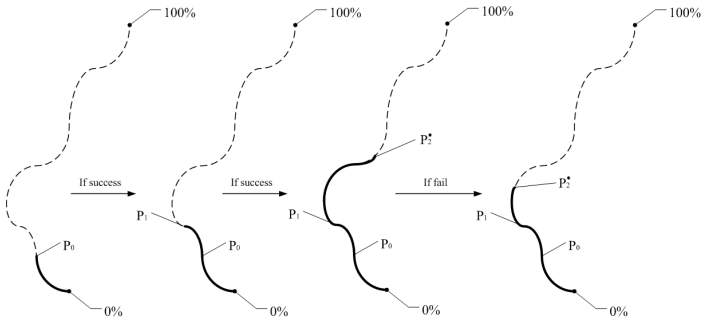


Figure 13: Route approaching process of node

We do route approaching for all nodes in the same time. A failure from any node will force us to try smaller routes for all nodes. This seems unnecessary, but to try route approaching for one node per iteration may bring unexpected instability.

8.2 Scheme

Let the spring constant vector be $\widetilde{\kappa}$, damping coefficient vector $\widetilde{\xi}$, and minimum regularization coefficient δ equals $\widetilde{\delta}$. By implemented coefficient approaching, our main process scheme of our program can be described as:

Main process scheme

---

1) Set initial value of δ and corresponding decrease coefficient $C^\delta \in (0,1)$.
2) Set initial spring constants κ, damping coefficients ξ and corresponding increase coefficient $C^{\kappa\xi} \in (0,1)$.
3) Set initial percentage of movement p and corresponding increase coefficient $C^p \in (0,1)$.
4) Try to use $p = \min\{p(1+C^p), 1\}$ to implement Newton's method.
   If solution convergent, and $p < 1$ then repeat 4).
   If solution convergent, and $p = 1$ then go to 5).
   If solution doesn't converge, $C^p = C^p/2$ and repeat 4).
5) Try to use $\kappa = \min\{\kappa(1+C^{\kappa\xi}), \widetilde{\kappa}\}$ and $\xi = \min\{\xi(1+C^{\kappa\xi}), \widetilde{\xi}\}$ to implement Newton's method.
   If solution convergent, and $\kappa_i < \widetilde{\kappa}_i$ or $\xi_i < \widetilde{\xi}_i$ for all i then repeat 5).
   If solution convergent, and $\kappa = \widetilde{\kappa}$ and $\xi = \widetilde{\xi}$ then go to 6).
   If solution doesn't convergent, $C^{\kappa\xi} = C^{\kappa\xi}/2$ and repeat 5).
6) Try to use $\delta = \max\{\delta C^\delta, \widetilde{\delta}\}$ to implement Newton's method.
   If solution convergent, and $\delta > \widetilde{\delta}$ then repeat 6).
   If solution convergent, and $\delta = \widetilde{\delta}$ then we have the solution.
   If solution doesn't convergent, $C^p = \sqrt{C^p}$ and repeat 6).

Remarks:
1)The main process scheme does not include all details. In some critical situations, increasing coefficient may become rather small after several non-convergence. We let the program terminate when it happens. In that case we need to count the number of iterations, if Newton's method used too many iteration, then program terminates.

2)Always be aware that Newton's method is a *local* method. Although we have the opportunity to get a global solution for some early setting, that is for small spring constants, damping coefficients and percentage of movement compared to the ordinary problem setting. Then after we prolong our reference or enlarge spring constants and damping coefficients, there is no guarantee we will have a global solution for this "new" problem.

9 Analysis of numerical solution

9.1 Analysis of the simple case

In section 5 we introduced a simple 1-dimension example. With the ordinary running cost: $h = (x_1 - x_r)^2$, we found that the solution does not convergent fast at the end of the trajectory. Because this running cost allows overshot near x_r. To get a better solution, we change the configuration that is we allow a reference of displacement and velocity.
Let spring length at $t = 0$ is 1, that is the mass start from $x = 1$. We expect it

finally stop at $x = 1.01$. The reference velocity is shown in Fig.14:

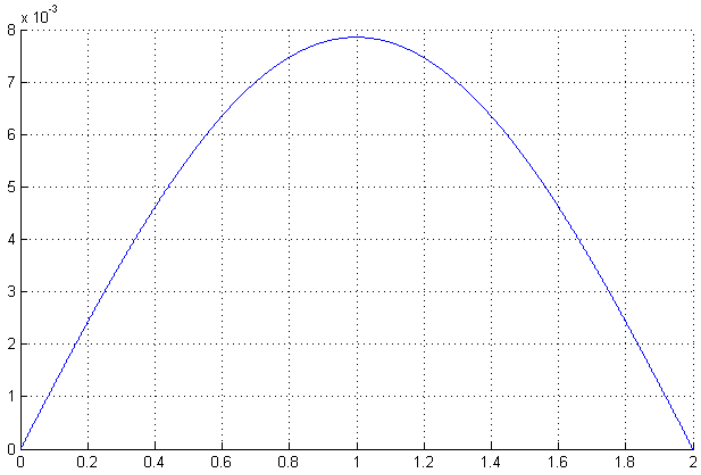


Figure 14: Reference velocity

And the reference displacement is shown in Fig.15:

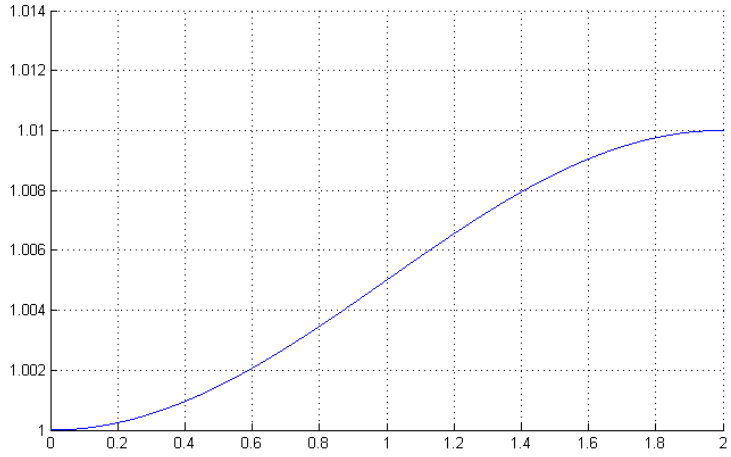


Figure 15: Reference displacement

Choose coefficients as follow:

$$\begin{cases} L_{min} &= 0.5 \\ L_{max} &= 1.515 \\ \kappa_{max} &= 10 \\ \xi_{max} &= 20 \\ m_{min} &= 100 \\ \gamma &= 0.0001 \\ d &= 1 \\ n &= 1 \\ M &= 2 \\ T &= 2 \\ \varepsilon &= 1 \end{cases}$$

In general it is difficult to find an analytical solution for optimal control problem. To get limit of displacement and velocity in this case we treat the solution solved for $N = 4000$ as exact solution. The maximum displacement and maximum velocity from solution are:

$$\begin{cases} x_{max} &= 1.00035 \\ v_{max} &= 0.00029451 \end{cases}$$

After choosing of coefficients, we can compute upper bound and lower bound in the theorem (5.1) from the solution when $N = 4000$.

From (39), we compute C_1 and C_2:

$$\begin{aligned} C_1 &= \max\left\{(M-1)\big(\tfrac{L_{max}\cdot\kappa_{max}}{m_{min}} + \tfrac{2\kappa_{max}}{m_{min}}x_{max} + \tfrac{8d\xi_{max}x_{max}^2 v_{max}}{m_{min}\varepsilon^2}\big), v_{max}\right\} \\ &= \max\left\{\big(\tfrac{1.515\cdot 10}{100} + \tfrac{2\cdot 10}{100}\cdot 1.00035 + \tfrac{8\cdot 20\cdot 1.00035^2\cdot 0.00029451}{100}\big), 0.00029451\right\} \\ &= 0.352 \end{aligned}$$

$$
\begin{aligned}
C_2 &= \max\Bigg\{ \quad (M-1)\Big(\tfrac{2\kappa_{max}}{\varepsilon^3 m_{min}} + \tfrac{2\kappa_{max}L_{max}d}{\varepsilon m_{min}} + \tfrac{2\kappa_{max}}{m_{min}} \\
&\qquad + \tfrac{8d\xi_{max}x_{max}}{m_{min}}\big(\tfrac{v_{max}}{\varepsilon^2} + \tfrac{8dx_{max}^2 v_{max}}{\varepsilon^4}\big) + \tfrac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2}\Big) \\
&\qquad , 2x_{max}, (m-1)\tfrac{8d\xi_{max}x_{max}^2}{m_{min}\varepsilon^2} + 1 \qquad\qquad \Bigg\} \\
&= \max\Bigg\{ \quad \Big(\tfrac{2\cdot 10}{100} + \tfrac{2\cdot 10\cdot 1.515}{100} + \tfrac{2\cdot 10}{100} \\
&\qquad + \tfrac{8\cdot 20\cdot 1.00035}{100}(0.00029451 + 8\cdot 1.00035^2\cdot 0.00029451) \\
&\qquad + \tfrac{8\cdot 20\cdot 1.00035\cdot 0.00029451}{100}\Big) \\
&\qquad , 2\cdot 1.00035, \tfrac{8\cdot 20\cdot 1.00035^2}{100} + 1 \qquad\qquad \Bigg\} \\
&= 2.00069
\end{aligned}
$$

According assumption of the theorem 5.1, $C_3 = 0.$ and δ is:

$$
\begin{aligned}
\delta &= \tfrac{1}{2}\ln 2 \cdot n \cdot (L_{max} - L_{min}) \cdot \gamma \\
&= \tfrac{1}{2}\ln 2(1.515 - 0.5) \cdot 0.0001 \\
&= 0.0000351772
\end{aligned}
$$

Formally we have the upper bound and lower bound.

$$
-\tfrac{0.352\cdot 2.00069\cdot 2}{2}\big((e^{2.00069\cdot 2} - 1)\Delta t + \Delta t^2\big) - 2\cdot 0.0000351772
$$

$$
\leq u(x_s, 0) - \overline{u}(x_s, 0) \leq
$$

$$
0.5 \cdot 0.352 \cdot 2.00069 \cdot e^{2.00069\cdot 2} \cdot 2\Delta t + 2 \cdot 0.0000351772
$$

However, since $\mathbb{H}$ is time dependent, the theorem 5.1 can not be used directly. But those bounds actually applies for this case, which is shown by an empirical convergence.

Together with value of value-function $u(x, 0)$ for N=500,1000,1500,2000, 2500,3000,3500,4000, we have figure:

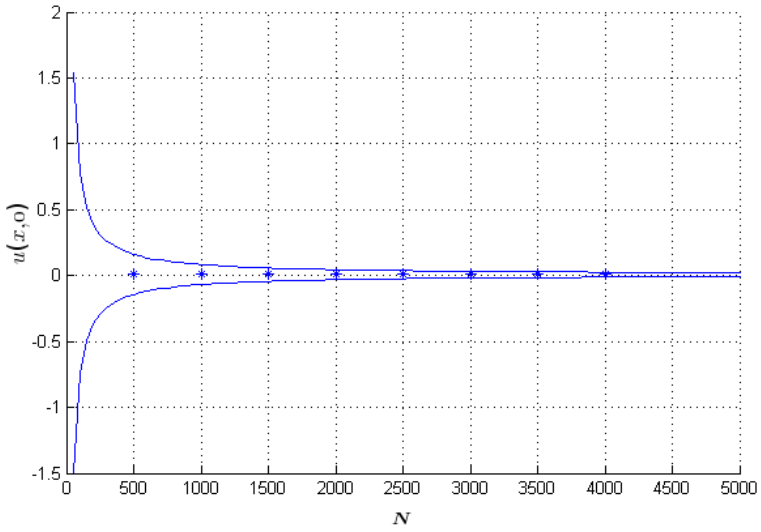


Figure 16: Value functions

In the Fig.16, the upper curve is upper bound computed from theorem 5.1 and lower curve is lower bound computed from theorem 5.1. Star marks shows value-function convergent towards the value of $N = 4000$ when N increases.

It shows theorem 5.1 is valid. And also we found that the evaluation of coefficients C_1 and C_2 is coarse. That may not a problem when model is simple enough, and displacement and velocity of model is not very large. However, in general a model of human body consist many nodes. Using bounds to evaluate solution seems not possible in those cases.

The convergence of solution of motion $z(t)$ is not guaranteed by the theorem. We could only expect it is convergent. To see whether it is convergent, we divide $[0, T]$ into 500 partitions, take value at borders of these portions, then take maximum norm of difference between values for some N and N=4000, that is $|x_N - x_{4000}|_{max}$. Luckily in this example, we actually have convergence:

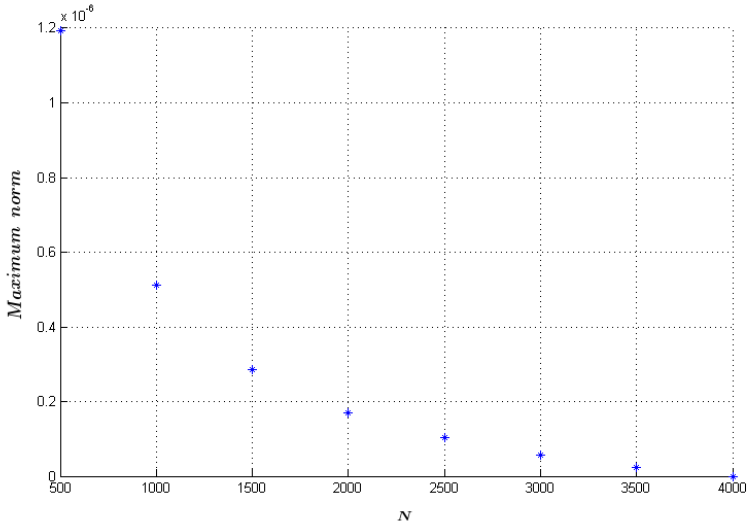


Figure 17: Maximum norm of solution

The Fig.17 shows the convergence of maximum norm: $|x_N - x_{4000}|_{max}$ as N increases.

9.2 Analyze of arm model

In this project we focus on the arm motion shown in Fig.1. To simplify the problem, a 2D model is established. We assume the model has 4 virtual bones, and consists of 4 free moving nodes and 2 fixed nodes. There are 8 springs connect all those 6 nodes in a preset manner shown in Fig.18:

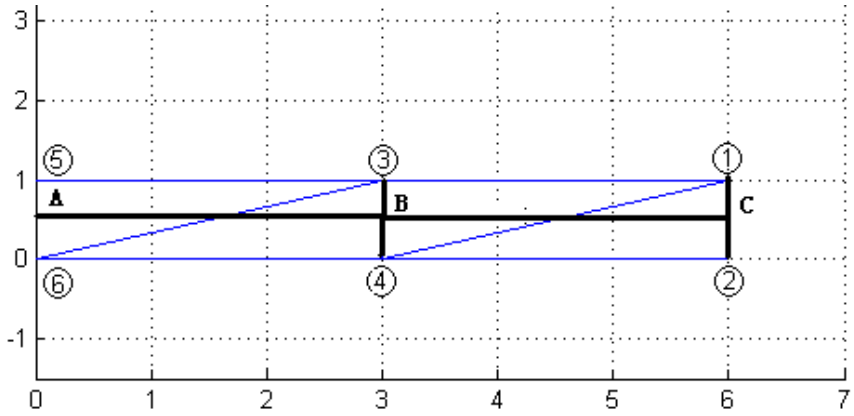


Figure 18: Black lines are "virtual bones" of "arm". Blue lines and black lines between nodes 1,2 and 3,4 are springs.

Bone A-B fixed to the wall at A, bone B-C connect to A-B at B. Bone 3-4 rotate around B and bone 1-2 rotate around A. We know distance between 3 and 4 and distance between 1 and 4 are not constant, here for simplicity, we assume distance of 3-4 and 1-2 are fixed in reference data. Therefore it makes possible to compute L_{max} and L_{min}, and moreover the manner of movement.

The position of the initial state is described in Fig.18. And all nodes at the initial has zero velocity which is same as maximum displacement state. At the maximum displacement state the 4 virtual bones are rest with such angles:

$$\begin{aligned} A-B &: 15^\circ \\ B-C &: 45^\circ \\ 3-4 &: 15^\circ \\ 1-2 &: 20^\circ \end{aligned}$$

The reference movement is like this: The arm start from the initial state, and move to the maximum displacement state, and then move back to the initial state which is also the final state. Each bone has its own angle to move. We set its angle as a sin function between 0 to its maximum angle. This is shown in Fig.(19). And its angle speed as a cosine function which is derivative of angle function. That is shown in Fig.(20). We let arm moves one period.

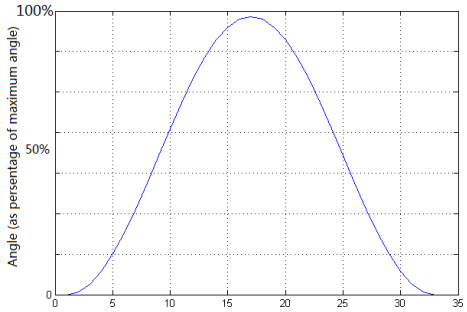


Figure 19: Angle of a bone

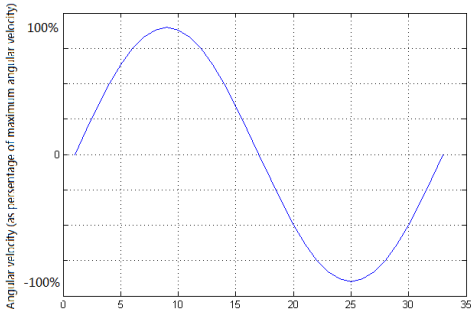


Figure 20: Angle velocity of a bone

Reference traces of free moving nodes: ①,②,③,④ are shown in Fig.21:

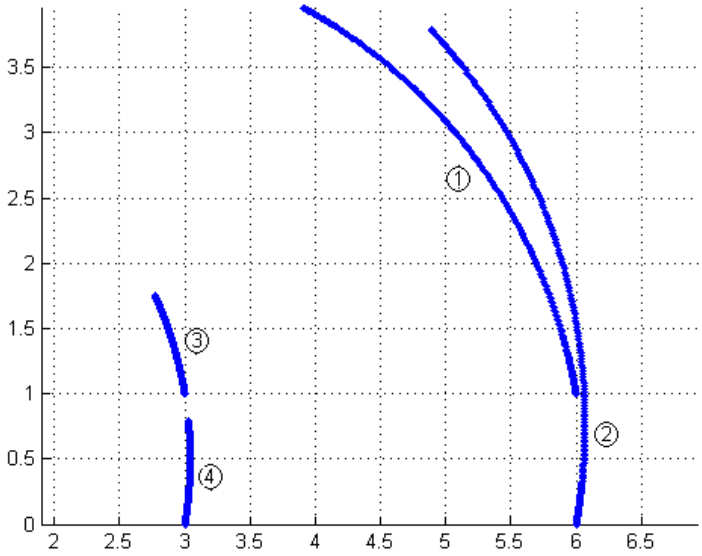


Figure 21: Reference traces

Denote reference traces as: $X_1(t)$, $X_2(t)$, $X_3(t)$, $X_4(t)$. Our target is minimize the value function $u = h + g$ where the running cost is $h = \sum_{n=0}^{N} \sum_{i=1}^{4} (x_i(n\Delta t) - X_i(n\Delta t))$, and the terminal cost is $g = 0$. N is number of time intervals.

During implement the algorithm described in section 8.2, we found that a bad setting of parameters leads the algorithm to be hard to get convergence. This is explained in section 6.2. Therefore we choose coefficients as follow:

$$\begin{cases} L_{min} &= 0.5 \\ L_{max} &= 1.59 \\ \kappa_{max} &= 20 \\ \xi_{max} &= 40 \\ m_{min} &= 3 \\ \gamma &= 0.0001 \\ d &= 2 \\ n &= 8 \\ M &= 6 \\ T &= 30 \\ \varepsilon &= 0.98 \end{cases}$$

We can evaluate:

$$\begin{cases} x_{max} &= 6.0736 \\ v_{max} &= 0.35 \end{cases}$$

This is a complex system, the better way to evaluate the convergence is observe convergence of the value function for different N and observe convergence of maximum norm of differences between two successive displacement solutions.

From Fig.22 one can observe that the value function is convergent (at least locally) as N increasing.

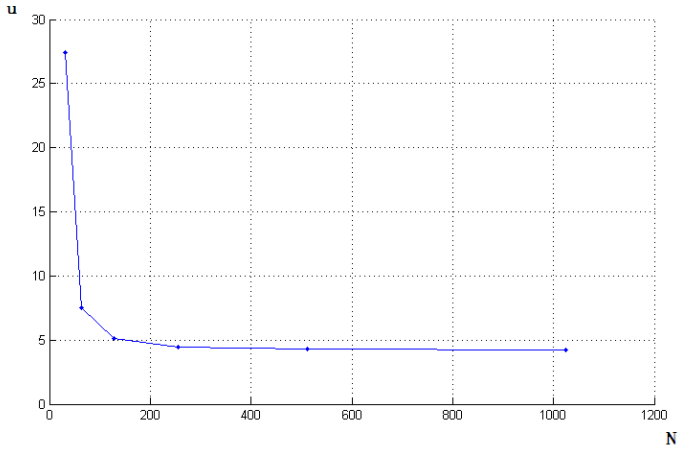


Figure 22: The value function for different N

Also we interested in if displacement solution is convergent. Because of our chosen of N, we use all displacement when $N = 32$, compare to every displacement data at same time nodes while $N = 64$, then take maximum difference. And use same way to compare data at $N = 64$ to $N = 128$, at $N = 128$ to $N = 256$, and so on. Finally maximum norm of difference of x vs. time intervals N is

obtained as:

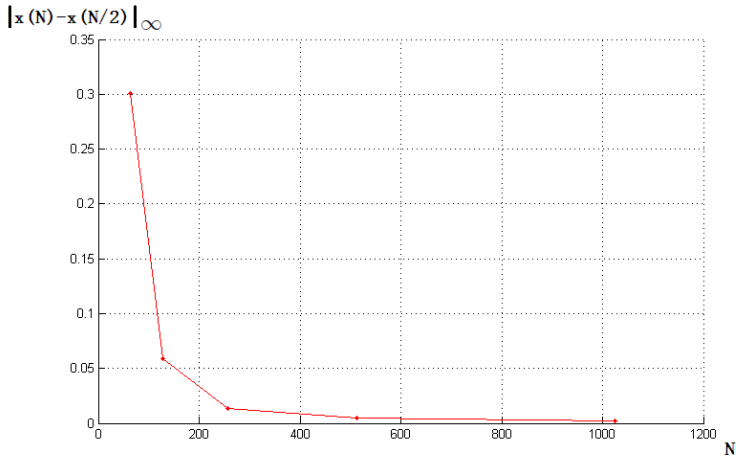


Figure 23: maximum norm of differences between two successive displacement

It is clear we got a convergent solution, (at least locally). [4]

A Appendix A: Proof of bounds of x and λ and Lipschitz constants of Hamiltonian

A.1 Bounds of x and λ

To implement theorem 5.1 we need to check if assumptions $|\mathbb{H}(x,\lambda_1)-\mathbb{H}(x,\lambda_2)| \leq C_1|\lambda_1-\lambda_2|$ and $|\mathbb{H}(x,\lambda)-\mathbb{H}(y,\lambda)| \leq C_2\big|1+|\lambda|\big|\cdot|x-y|$ are hold when assumptions in the section 6.2 holds. One way of doing this is firstly show the variable z and the dual variable ρ are bounded, then compute the Lipschitz constants.

Recall that there are positive constants κ_{max}, ξ_{max}, L_{min}, L_{max}, m_{min} and m_{max}, such that for every spring connects two nodes i and j, its spring constant κ_{ij}, damping coefficient ξ_{ij}, minimum rest length L_{ij}^- and maximum rest length L_{ij}^+ satisfy $\kappa_{ij} \leq \kappa_{max}$, $\xi_{ij} \leq D_{max}$, $L_{min} \leq L_{ij}^- \leq L_{ij}^+ \leq L_{max}$; for every node its mass m_i satisfies $m_{min} < m_i < m_{max}$.

Proof of z is bounded:

The first step is proof of z is bounded.

The norm of z here is computed as: $\|z\| := \sqrt{\sum_{j=1}^{M}\sum_{k=1}^{d}(x_{k,j}^2+v_{k,j}^2)}$.

So we have:

$$\|z\|^2 = \sum_{j=1}^{M}\sum_{k=1}^{d}(x_{k,j}^2+v_{k,j}^2),$$

and

$$\begin{aligned}
\|z\|\tfrac{d\|z\|}{dt} &= \sum_{j=1}^{M}\sum_{k=1}^{d}(x_{k,j}\tfrac{dx_{k,j}}{dt}| + v_{k,j}\tfrac{dv_{k,j}}{dt}|) \\
&= \sum_{j=1}^{M}\sum_{k=1}^{d}\Bigg(x_{k,j}v_{k,j} \\
&\quad +v_{k,j}\sum_{i\in\Omega_j}\Big\{\big[\kappa_{ij}(L_{ij}-l_{ij}) \\
&\quad -\xi_{ij}(\tfrac{\sum_{s=1}^{d}(v_{s,j}-v_{s,i})(x_{s,j}-x_{s,i})}{l_{ij}})\big]\tfrac{x_{k,j}-x_{k,i}}{l_{ij}m_j}\Big\}\Bigg)
\end{aligned}$$

$$
\begin{aligned}
\therefore \frac{d\|z\|}{dt} &= \sum_{j=1}^{M}\sum_{k=1}^{d}\Bigg(\frac{|x_{k,j}||v_{k,j}|}{\|z\|} \\
&\quad + \frac{|v_{k,j}|\Big|\sum\limits_{i\in\Omega_j}\Big\{[\kappa_{ij}(L_{ij}-l_{ij})-\xi_{ij}(\frac{\sum_{s=1}^{d}(v_{s,j}-v_{s,i})(x_{s,j}-x_{s,i})}{l_{ij}})]\frac{x_{k,j}-x_{k,i}}{l_{ij}m_j}\Big\}\Big|}{\|z\|}\Bigg) \\
&\leq \sum_{j=1}^{M}\sum_{k=1}^{d}\frac{|x_{k,j}||v_{k,j}|}{\|z\|} \qquad ① \\
&\quad + \sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_j}\frac{\frac{\kappa_{ij}L_{ij}|v_{k,j}||x_{k,j}-x_{k,i}|}{\sqrt{\sum_{s=1}^{d}(x_{s,j}-x_{s,i})^2}}}{\|z\|m_j} \qquad ② \\
&\quad + \sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_j}\frac{\kappa_{ij}|v_{k,j}||x_{k,j}-x_{k,i}|}{\|z\|m_j} \qquad ③ \\
&\quad + \sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_j}\sum_{s=1}^{d}\frac{\xi_{ij}|v_{k,j}|\cdot|x_{k,j}-x_{k,i}|\left|\frac{(v_{s,j}-v_{s,i})(x_{s,j}-x_{s,i})}{\sum_{l=1}^{d}(x_{l,j}-x_{l,i})^2}\right|}{\|z\|m_j} \qquad ④
\end{aligned}
$$

Analyze each term. Assume that there are n nodes in a grid.

$$
\begin{aligned}
① &\leq \sum_{j=1}^{M}\sum_{k=1}^{d}|x_{k,j}| \\
&\leq \sum_{j=1}^{M}\sum_{k=1}^{d}\|z\| \\
&= Md\|z\|
\end{aligned}
$$

$$
\begin{aligned}
② &\leq \sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_j}\frac{\kappa_{ij}L_{ij}|v_{k,j}|}{\|z\|m_j} \\
&\leq \sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_j}\frac{\kappa_{ij}L_{ij}}{m_j} \\
&= \sum_{j=1}^{M}\sum_{i\in\Omega_j}d\frac{\kappa_{ij}L_{ij}}{m_j} \\
&\leq 2nd\frac{\kappa_{max}L_{max}}{m_{min}}
\end{aligned}
$$

$$
\begin{aligned}
③ &\leq \sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_j}\frac{\kappa_{ij}|x_{k,j}-x_{k,i}|}{m_j} \\
&\leq \sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_j}\frac{\kappa_{ij}(|x_{k,j}|+|x_{k,i}|)}{m_j} \\
&\leq \sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_j}2\frac{\kappa_{ij}\|z\|}{m_j} \\
&= \sum_{j=1}^{M}\sum_{i\in\Omega_j}2\frac{d\kappa_{ij}\|z\|}{m_j} \\
&\leq 4nd\kappa_{max}\|z\|/m_{min}
\end{aligned}
$$

$$
\begin{aligned}
④ &\leq \sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_1}\sum_{s=1}^{d}\left(\frac{\xi_{ij}|v_{k,j}||x_{k,j}-x_{k,i}|\frac{|v_{s,j}-v_{s,i}|}{\sqrt{\sum_{l=1}^{d}(x_{l,j}-x_{l,i})^2}}}{\|z\|m_j}\right) \\
&= \sum_{j=1}^{M}\sum_{i\in\Omega_1}\left(\frac{\xi_{ij}\sum_{s=1}^{d}|v_{s,j}-v_{s,i}|\frac{\sum_{k=1}^{d}|x_{k,j}-x_{k,i}|\cdot|v_{k,j}|}{\sqrt{\sum_{l=1}^{d}(x_{l,j}-x_{l,i})^2}}}{\|z\|m_j}\right) \\
&\leq \sum_{j=1}^{M}\sum_{i\in\Omega_1}\Big(\sum_{k=1}^{d}\frac{|v_{k,j}|}{\|z\|}\xi_{ij}\sum_{s=1}^{d}|v_{s,j}-v_{s,i}|\Big)/m_j \\
&\leq \sum_{j=1}^{M}\sum_{i\in\Omega_1}\sum_{k=1}^{d}\sum_{s=1}^{d}\left(\frac{\xi_{ij}|v_{s,j}-v_{s,i}|}{m_j}\right) \\
&\leq \sum_{j=1}^{M}\sum_{i\in\Omega_1}\sum_{k=1}^{d}\sum_{s=1}^{d}\left(\frac{\xi_{ij}(|v_{s,j}|+|v_{s,i}|)}{m_j}\right) \\
&\leq \sum_{j=1}^{M}\sum_{i\in\Omega_1}\sum_{k=1}^{d}\sum_{s=1}^{d}2\xi_{ij}\|z\|/m_j \\
&\leq \sum_{j=1}^{M}\sum_{i\in\Omega_1}2d^2\xi_{ij}\|z\|/m_j \\
&= 4nd^2\xi_{max}\|z\|/m_{min}
\end{aligned}
$$

Then we have the conclusion:

$$\frac{d\|z\|}{dt} \leq (Md + \frac{4nd\kappa_{max}+4nd^2\xi_{max}}{m_{min}})\|z\| + 2nd\frac{\kappa_{max}L_{max}}{m_{min}}$$

Denote $A := (Md + \frac{4nd\kappa_{max}+4nd^2\xi_{max}}{m_{min}})$ and $B := 2nd\frac{\kappa_{max}L_{max}}{m_{min}}$, then:

$$\begin{aligned}
&\frac{d\|z\|}{dt} \leq A\|z\| + B \\
\implies &\frac{d\|z\|}{dt} - A\|z\| \leq B \\
\implies &e^{-At}(\frac{d\|z\|}{dt} - A\|z\|) \leq e^{-At}B \\
\implies &(e^{-At}\|z\|)' \leq e^{-At}B \\
\implies &\int_0^t (e^{-As}\|z\|)'ds \leq \int_0^t e^{-As}Bds \\
\implies &\|z(t)\| - \|z(0)\| \leq \int_0^t e^{-As}Bds \\
\implies &\|z(t)\| \leq \|z(0)\| + \int_0^t e^{-As}Bds
\end{aligned}$$

We have implemented Gronwall's Lemma here, one can read more at [**?**].
Since all constants in A and B are positive, then A and B are positive, then it is clear that $\|z\|$ is bounded. As a consequence, all position $x_{k,j}$ and velocity $v_{k,j}$ are bounded. We say that there are x_{max} and v_{max}, such that $|x_{k,j}| \leq x_{max}$ and $|v_{k,j}| \leq v_{max}$ for all k and j.

Proof of ρ is bounded:

After our proof for z, here we will prove ρ is also bounded use a similar idea.

We want to prove $\|\rho\|$ is bounded and therefore $|\lambda_{k,j}| \leq \lambda_{max}$ and $|\zeta_{k,j}| \leq \zeta_{max}$ for all k, j. This can be done by using the Gronwall's Lemma again which is similar procedure as we did for x.
In this model rest length of springs are control of this system. We let $L_{ij} \in [L_{ij}^-, L_{ij}^+]$ a compact set, $L_{ij}^+ \geq L_{ij}^- \geq 0$. Denote $\mathbb{A} := [L_{ij}^-, L_{ij}^+]$ and $\mathbb{A}^n$ is collection of all control sets.

The Hamiltonian is minimized when, $L_{ij} = L_{ij}^-$ or $L_{ij} = L_{ij}^+$. Let $\{L_{ij}^*\}$ denote the minimizer set of then Hamiltonian. Recall that the Hamiltonian H defined as $\mathbb{H}(p,x) := \min_{L\in\mathbb{A}^n}\Big\{f(z,L)\cdot\rho + h(z,L)\Big\}, \quad \forall\rho, z \in \mathbb{R}^n$ where f, z and ρ is defined in (38) and (36). We express the Hamiltonian in following way:

$$
\begin{aligned}
\mathbb{H} = & \min_{L\in\mathbb{A}^n}\Big\{f(z,L)\cdot\rho+h(z,L)\Big\} \\
= & \min_{L\in\mathbb{A}^n}\{\sum_{j=1}^{M}(\sum_{k=1}^{d}(\lambda_{k,j}v_{k,j})+\sum_{k=1}^{d}(\zeta_{k,j}F_{k,j}))+h\} \\
& +\sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_j}\zeta_{k,j}\tfrac{-\kappa_{ij}(x_{k,j}-x_{k,i})}{m_j} && \cdots \quad (II) \\
& +\sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_j}\zeta_{k,j}(-\xi_{ij}(x_{k,j}-x_{k,i})(\tfrac{\sum_{s=1}^{d}(v_{s,j}-v_{s,i})(x_{s,j}-x_{s,i})}{l_{ij}^2 m_j})) && \cdots \quad (III) \\
& +\sum_{j=1}^{M}\sum_{k=1}^{d}(\lambda_{k,j}v_{k,j}) && \cdots \quad (IV) \\
& +\sum_{j=1}^{M}\sum_{k=1}^{d}(x_{k,j}-X_{k,j})^2 && \cdots \quad (V)
\end{aligned}
$$

Observe that these 5 parts are piecewise smooth w.r.t all $x_{k,j}$, $v_{k,j}$, $\lambda_{k,j}$ and $\zeta_{k,j}$. We can find Lipschitz constants from a combination of partial derivatives. Therefore we would like to write partial derivatives for each variable before we continue.

Recall that $\mathcal{S}$ is the set of connected pairs defined in (35), and ε is the lower bound for all l_{ij} given in (37).

We have:

$$
\begin{aligned}
(I) \quad = & \sum_{k=1}^{d}\sum_{j=1}^{M}\sum_{i\in\Omega_j}\frac{\zeta_{k,j}\kappa_{ij}L_{ij}^*(x_{k,j}-x_{k,i})}{m_j\sqrt{\sum_{s=1}^{d}(x_{s,j}-x_{s,i})^2}} \\
= & \sum_{k=1}^{d}\sum_{\substack{(i,j)\in\mathcal{S}\\ i<j}}\left(\frac{\zeta_{k,j}\kappa_{ij}L_{ij}^*(x_{k,j}-x_{k,i})}{m_j\sqrt{\sum_{s=1}^{d}(x_{s,j}-x_{s,i})^2}}+\frac{\zeta_{k,i}\kappa_{ij}L_{ij}^*(x_{k,i}-x_{k,j})}{m_i\sqrt{\sum_{s=1}^{d}(x_{s,i}-x_{s,j})^2}}\right) \\
= & \sum_{\substack{(i,j)\in\mathcal{S}\\ i<j}}\sum_{k=1}^{d}\left(\frac{\left(\frac{\zeta_{k,j}}{m_j}-\frac{\zeta_{k,i}}{m_i}\right)\kappa_{ij}L_{ij}^*(x_{k,j}-x_{k,i})}{\sqrt{\sum_{s=1}^{d}(x_{s,j}-x_{s,i})^2}}\right)
\end{aligned}
$$

Then

$$\begin{aligned}
\frac{d(I)}{dx_{\overline{k},\overline{j}}} &= \sum_{i\in\Omega_{\overline{j}}}\Big(\kappa_{i\overline{j}}L^*_{i\overline{j}}\Big(-\sum_{\substack{k=1\\k\neq\overline{k}}}^{d}\big(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}}-\frac{\zeta_{k,i}}{m_i}\big)(x_{k,\overline{j}}-x_{k,i})\frac{x_{\overline{k},\overline{j}}-x_{\overline{k},i}}{\Big(\sum\limits_{s=1}^{d}(x_{s,\overline{j}}-x_{s,i})^2\Big)^{3/2}}\\
&\quad+\big(\frac{\zeta_{\overline{k},\overline{j}}}{m_{\overline{j}}}-\frac{\zeta_{\overline{k},i}}{m_i}\big)\frac{1-(x_{\overline{k},j}-x_{\overline{k},i})^2}{\Big(\sum\limits_{s=1}^{d}(x_{s,\overline{j}}-x_{s,i})^2\Big)^{3/2}}\Big)\Big)\\
&= \sum_{i\in\Omega_{\overline{j}}}\Big(\kappa_{i\overline{j}}L^*_{i\overline{j}}\frac{(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}}-\frac{\zeta_{k,i}}{m_i})-(x_{\overline{k},\overline{j}}-x_{\overline{k},i})\sum\limits_{k=1}^{d}\big(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}}-\frac{\zeta_{k,i}}{m_i}\big)(x_{k,\overline{j}}-x_{k,i})}{\Big(\sum\limits_{s=1}^{d}(x_{s,\overline{j}}-x_{s,i})^2\Big)^{3/2}}\Big)\\
\frac{d(I)}{d\zeta_{\overline{k},\overline{j}}} &= \sum_{i\in\Omega_{\overline{j}}}\frac{\kappa_{i\overline{j}}L^*_{i\overline{j}}(x_{\overline{k},\overline{j}}-x_{\overline{k},i})}{m_{\overline{j}}\sqrt{\sum\limits_{s=1}^{d}(x_{s,\overline{j}}-x_{s,i})^2}}\\
|\frac{d(I)}{dx_{\overline{k},\overline{j}}}| &\leq \sum_{i\in\Omega_{\overline{j}}}\Big(\kappa_{i\overline{j}}L^*_{i\overline{j}}|\frac{(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}}-\frac{\zeta_{k,i}}{m_i})-(x_{\overline{k},j}-x_{\overline{k},i})\sum\limits_{k=1}^{d}\big(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}}-\frac{\zeta_{k,i}}{m_i}\big)(x_{k,\overline{j}}-x_{k,i})}{\Big(\sum\limits_{s=1}^{d}(x_{s,\overline{j}}-x_{s,i})^2\Big)^{3/2}}|\Big)\\
&\leq \sum_{i\in\Omega_{\overline{j}}}\Big(\kappa_{i\overline{j}}L^*_{i\overline{j}}|\frac{\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}}-\frac{\zeta_{k,i}}{m_i}}{\Big(\sum\limits_{s=1}^{d}(x_{s,\overline{j}}-x_{s,i})^2\Big)^{3/2}}|\\
&\quad+\kappa_{i\overline{j}}L^*_{i\overline{j}}|\frac{(x_{\overline{k},j}-x_{\overline{k},i})\sum\limits_{k=1}^{d}\big(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}}-\frac{\zeta_{k,i}}{m_i}\big)(x_{k,\overline{j}}-x_{k,i})}{\Big(\sum\limits_{s=1}^{d}(x_{s,\overline{j}}-x_{s,i})^2\Big)^{3/2}}|\Big)\\
&\leq \sum_{i\in\Omega_{\overline{j}}}\Big(\kappa_{i\overline{j}}L^*_{i\overline{j}}|\frac{\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}}-\frac{\zeta_{k,i}}{m_i}}{\varepsilon^3}|+\kappa_{i\overline{j}}L^*_{i\overline{j}}|\frac{\sum\limits_{k=1}^{d}\big(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}}-\frac{\zeta_{k,i}}{m_i}\big)}{\sqrt{\sum\limits_{s=1}^{d}(x_{s,\overline{j}}-x_{s,i})^2}}|\Big)\\
&\leq \sum_{i\in\Omega_{\overline{j}}}\Big(\kappa_{i\overline{j}}L^*_{i\overline{j}}|\frac{2\|\rho\|/m_{min}}{\varepsilon^3}|+\kappa_{i\overline{j}}L^*_{i\overline{j}}|\frac{\frac{1}{m_{min}}\sum\limits_{k=1}^{d}\big(|\zeta_{k,\overline{j}}|+|\zeta_{k,i}|\big)}{\varepsilon}|\Big)\\
&\leq 2card(\Omega_{\overline{j}})\kappa_{max}L_{max}\|\rho\|/\varepsilon^3 m_{min}+2card(\Omega_{\overline{j}})\kappa_{max}L_{max}d\|\rho\|/\varepsilon m_{min}
\end{aligned}$$

$$(II)=\sum_{k=1}^{d}\sum_{\substack{(i,j)\in\mathcal{S}\\i<j}}-\frac{\kappa_{ij}}{m_j}(\zeta_{k,j}-\zeta_{k,i})(x_{k,j}-x_{k,i})$$

$$
\begin{aligned}
\frac{d(II)}{dx_{\overline{k},\overline{j}}} &= \sum_{i\in\Omega_{\overline{j}}} (-\frac{\kappa_{i\overline{j}}}{m_{\overline{j}}})(\zeta_{\overline{k},\overline{j}} - \zeta_{\overline{k},i}) \\
\frac{d(II)}{d\zeta_{\overline{k},\overline{j}}} &= \sum_{i\in\Omega_{\overline{j}}} (-\frac{\kappa_{i\overline{j}}}{m_{\overline{j}}})(x_{\overline{k},\overline{j}} - x_{\overline{k},i}) \\
\left|\frac{d(II)}{dx_{\overline{k},\overline{j}}}\right| &\leq \sum_{i\in\Omega_{\overline{j}}} |-\frac{\kappa_{i\overline{j}}}{m_{\overline{j}}}| \cdot |\zeta_{\overline{k},\overline{j}} - \zeta_{\overline{k},i}| \\
&\leq \sum_{i\in\Omega_{\overline{j}}} |\frac{\kappa_{i\overline{j}}}{m_{\overline{j}}}| \cdot (|\zeta_{\overline{k},\overline{j}}| + |\zeta_{\overline{k},i}|) \\
&\leq card(\Omega_{\overline{j}}) \frac{2\kappa_{max}}{m_{min}} \|\rho\|
\end{aligned}
$$

$$
\begin{aligned}
(III) &= \sum_{k=1}^{d} \sum_{\substack{(i,j)\in\mathcal{S} \\ i<j}} \left(-\xi_{ij}\left(\frac{\zeta_{k,j}}{m_j} - \frac{\zeta_{k,i}}{m_i}\right)(x_{k,j} - x_{k,i}) \frac{\sum_{s=1}^{d}(v_{s,j}-v_{s,i})(x_{s,j}-x_{s,i})}{\sum_{l=1}^{d}(x_{l,j}-x_{l,i})^2} \right) \\
\\
\frac{d(III)}{dx_{\overline{k},\overline{j}}} &= \sum_{i\in\Omega_{\overline{j}}} \Bigg\{ -\xi_{i\overline{j}} \sum_{\substack{k=1 \\ k\neq\overline{k}}}^{d} \left(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}} - \frac{\zeta_{k,i}}{m_i}\right)(x_{k,\overline{j}} - x_{k,i}) \cdot \frac{d}{dx_{\overline{k},\overline{j}}} \\
&\quad \left(\frac{\sum_{s=1}^{d}(v_{s,j}-v_{s,i})(x_{s,j}-x_{s,i})}{\sum_{l=1}^{d}(x_{l,j}-x_{l,i})^2} \right) \\
&\quad -\xi_{i\overline{j}}\left(\frac{\zeta_{\overline{k},\overline{j}}}{m_{\overline{j}}} - \frac{\zeta_{\overline{k},i}}{m_i}\right) \cdot \frac{d}{dx_{\overline{k},\overline{j}}} \left((x_{\overline{k},\overline{j}} - x_{\overline{k},i}) \frac{\sum_{s=1}^{d}(v_{s,j}-v_{s,i})(x_{s,j}-x_{s,i})}{\sum_{l=1}^{d}(x_{l,j}-x_{l,i})^2} \right) \Bigg\} \\
\\
&= \sum_{i\in\Omega_{\overline{j}}} \Bigg\{ -\xi_{i\overline{j}} \sum_{\substack{k=1 \\ k\neq\overline{k}}}^{d} \left(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}} - \frac{\zeta_{k,i}}{m_i}\right)(x_{k,\overline{j}} - x_{k,i}) \left(\frac{(v_{\overline{k},\overline{j}} - v_{\overline{k},i})}{\sum_{l=1}^{d}(x_{l,\overline{j}}-x_{l,i})^2} \right. \\
&\quad \left. - \frac{2(x_{\overline{k},\overline{j}} - x_{\overline{k},i})(\sum_{s=1}^{d}(v_{s,\overline{j}}-v_{s,i})(x_{s,\overline{j}}-x_{s,i}))}{(\sum_{l=1}^{d}(x_{l,\overline{j}}-x_{l,i})^2)^2} \right) \\
&\quad -\xi_{i\overline{j}}\left(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}} - \frac{\zeta_{k,i}}{m_i}\right)(x_{k,\overline{j}} - x_{k,i}) \frac{\sum_{s=1}^{d}(v_{s,j}-v_{s,i})(x_{s,j}-x_{s,i})}{\sum_{l=1}^{d}(x_{l,j}-x_{l,i})^2} \\
&\quad -\xi_{i\overline{j}}\left(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}} - \frac{\zeta_{k,i}}{m_i}\right)(x_{k,\overline{j}} - x_{k,i}) \left(\frac{(v_{\overline{k},\overline{j}} - v_{\overline{k},i})}{\sum_{l=1}^{d}(x_{l,\overline{j}}-x_{l,i})^2} \right. \\
&\quad \left. - \frac{2(x_{\overline{k},\overline{j}} - x_{\overline{k},i})(\sum_{s=1}^{d}(v_{s,\overline{j}}-v_{s,i})(x_{s,\overline{j}}-x_{s,i}))}{(\sum_{l=1}^{d}(x_{l,\overline{j}}-x_{l,i})^2)^2} \right) \Bigg\} \\
\\
&= \sum_{i\in\Omega_{\overline{j}}} \Bigg\{ -\xi_{i\overline{j}} \sum_{k=1}^{d} \left(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}} - \frac{\zeta_{k,i}}{m_i}\right)(x_{k,\overline{j}} - x_{k,i}) \\
&\quad \left(\frac{(v_{\overline{k},\overline{j}} - v_{\overline{k},i})}{\sum_{l=1}^{d}(x_{l,\overline{j}}-x_{l,i})^2} - \frac{2(x_{\overline{k},\overline{j}} - x_{\overline{k},i})(\sum_{s=1}^{d}(v_{s,\overline{j}}-v_{s,i})(x_{s,\overline{j}}-x_{s,i}))}{(\sum_{l=1}^{d}(x_{l,\overline{j}}-x_{l,i})^2)^2} \right) \\
&\quad -\xi_{i\overline{j}}\left(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}} - \frac{\zeta_{k,i}}{m_i}\right) \frac{\sum_{s=1}^{d}(v_{s,\overline{j}}-v_{s,i})(x_{s,\overline{j}}-x_{s,i})}{\sum_{l=1}^{d}(x_{l,\overline{j}}-x_{l,i})^2} \Bigg\} \\
\\
\frac{d(III)}{dv_{\overline{k},\overline{j}}} &= \sum_{i\in\Omega_{\overline{j}}} \left\{ -\xi_{i\overline{j}} \sum_{k=1}^{d} \left(\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}} - \frac{\zeta_{k,i}}{m_i}\right)(x_{k,\overline{j}} - x_{k,i}) \frac{x_{\overline{k},\overline{j}} - x_{\overline{k},i}}{\sum_{l=1}^{d}(x_{l,\overline{j}}-x_{l,i})^2} \right\}
\end{aligned}
$$

$$
\begin{aligned}
\frac{d(III)}{d\zeta_{\overline{k},\overline{j}}} &= \sum_{i\in\Omega_{\overline{j}}}\sum_{k=1}^{d}\left(-\frac{\xi_{ij}}{m_{\overline{j}}}(x_{k,j}-x_{k,i})\frac{\sum_{s=1}^{d}(v_{s,j}-v_{s,i})(x_{s,j}-x_{s,i})}{\sum_{l=1}^{d}(x_{l,j}-x_{l,i})^2}\right)\\
\\
\left|\frac{d(III)}{dx_{\overline{k},\overline{j}}}\right| &\le \sum_{i\in\Omega_{\overline{j}}}\left\{\xi_{i\overline{j}}\sum_{k=1}^{d}(|\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}}|+|\frac{\zeta_{k,i}}{m_i}|)\cdot(|x_{k,\overline{j}}|+|x_{k,i}|)\right.\\
&\quad\left(\left|\frac{(|v_{\overline{k},\overline{j}}|+|v_{\overline{k},i}|)}{\sum_{l=1}^{d}(x_{l,\overline{j}}-x_{l,i})^2}\right|+\left|\frac{2|x_{\overline{k},\overline{j}}-x_{\overline{k},i}|(\sum_{s=1}^{d}(|v_{s,\overline{j}}|+|v_{s,i}|)(|x_{s,\overline{j}}|+|x_{s,i}|))}{(\sum_{l=1}^{d}(x_{l,\overline{j}}-x_{l,i})^2)^2}\right|\right)\\
&\quad\left.+\xi_{i\overline{j}}(|\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}}|+|\frac{\zeta_{k,i}}{m_i}|)\frac{\sum_{s=1}^{d}(|v_{s,\overline{j}}|+|v_{s,i}|)(|x_{s,\overline{j}}|+|x_{s,i}|)}{\sum_{l=1}^{d}(x_{l,\overline{j}}-x_{l,i})^2}\right\}\\
\\
&\le \sum_{i\in\Omega_{\overline{j}}}\left\{\xi_{max}\sum_{k=1}^{d}\frac{1}{m_{min}}\cdot 2\|\rho\|\cdot 2x_{max}(\frac{2v_{max}}{\varepsilon^2}+\frac{4x_{max}\cdot d\cdot 2v_{max}\cdot 2x_{max}}{\varepsilon^4})\right.\\
&\quad\left.+\xi_{max}\frac{1}{m_{min}}\cdot 2\|\rho\|\frac{d\cdot 2v_{max}\cdot 2x_{max}}{\varepsilon^2}\right\}\\
\\
&= card(\Omega_{\overline{j}})\cdot\left(\frac{8d\xi_{max}x_{max}}{m_{min}}(\frac{v_{max}}{\varepsilon^2}+\frac{8dx_{max}^2v_{max}}{\varepsilon^4})+\frac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2}\right)\cdot\|\rho\|\\
\left|\frac{d(III)}{dv_{\overline{k},\overline{j}}}\right| &\le \sum_{i\in\Omega_{\overline{j}}}\left\{\xi_{i\overline{j}}\sum_{k=1}^{d}(|\frac{\zeta_{k,\overline{j}}}{m_{\overline{j}}}|+|\frac{\zeta_{k,i}}{m_i}|)(|x_{k,\overline{j}}|+|x_{k,i}|)\frac{|x_{\overline{k},\overline{j}}|+|x_{\overline{k},i}|}{\sum_{l=1}^{d}(x_{l,\overline{j}}-x_{l,i})^2}\right\}\\
\\
&\le \sum_{i\in\Omega_{\overline{j}}}\left\{\xi_{max}\sum_{k=1}^{d}(\frac{2}{m_{min}}\|\rho\|)(2x_{max})\frac{2x_{max}}{\varepsilon^2}\right\}\\
&= card(\Omega_{\overline{j}})\frac{8d\xi_{max}x_{max}^2}{m_{min}\varepsilon^2}\|\rho\|
\end{aligned}
$$

$$
(IV)=\sum_{j=1}^{M}\sum_{k=1}^{d}\lambda_{k,j}v_{k,j}
$$

$$
\begin{aligned}
\frac{d(IV)}{dv_{\overline{k},\overline{j}}} &= \lambda_{\overline{k},\overline{j}}\\
\frac{d(IV)}{d\lambda_{\overline{k},\overline{j}}} &= v_{\overline{k},\overline{j}}\\
|\frac{d(IV)}{dv_{\overline{k},\overline{j}}}| &= |\lambda_{\overline{k},\overline{j}}|\\
&\le \|\rho\|
\end{aligned}
$$

$$
(V)=\sum_{j=1}^{M}\sum_{k=1}^{d}(x_{k,j}-X_{k,j})
$$

$$
\begin{aligned}
\frac{d(V)}{dx_{\overline{k},\overline{j}}} &= 2(x_{\overline{k},\overline{j}}-X_{\overline{k},\overline{j}})\\
|\frac{d(V)}{dx_{\overline{k},\overline{j}}}| &= 2|x_{\overline{k},\overline{j}}-X_{\overline{k},\overline{j}}|\\
&\le 2x_{max}
\end{aligned}
$$

We want to show the dual variable ρ is bounded by implement the Gronwall's Lemma in which $|-\frac{d\lambda_{\overline{k},\overline{j}}}{dt}|$ and $|-\frac{d\zeta_{\overline{k},\overline{j}}}{dt}|$ are used. From previous works we have

all necessary partial derivatives to compute these two.

For any $\overline{k}$ and $\overline{j}$,

$$
\begin{aligned}
|-\tfrac{d\lambda_{\overline{k},\overline{j}}}{dt}| &= |\tfrac{dH}{dx_{\overline{k},\overline{j}}}| = |\tfrac{d(I)}{dx_{\overline{k},\overline{j}}} + \tfrac{d(II)}{dx_{\overline{k},\overline{j}}} + \tfrac{d(III)}{dx_{\overline{k},\overline{j}}} + \tfrac{d(IV)}{dx_{\overline{k},\overline{j}}} + \tfrac{d(V)}{dx_{\overline{k},\overline{j}}}| \\
&\leq |\tfrac{d(I)}{dx_{\overline{k},\overline{j}}}| + |\tfrac{d(II)}{dx_{\overline{k},\overline{j}}}| + |\tfrac{d(III)}{dx_{\overline{k},\overline{j}}}| + |\tfrac{d(IV)}{dx_{\overline{k},\overline{j}}}| + |\tfrac{d(V)}{dx_{\overline{k},\overline{j}}}| \\
&\leq 2card(\Omega_{\overline{j}})\kappa_{max}L_{max}\|\rho\|/\varepsilon^3 m_{min} + 2card(\Omega_{\overline{j}})\kappa_{max}L_{max}d\|\rho\|/\varepsilon m_{min} \\
&\quad + card(\Omega_{\overline{j}})\tfrac{2\kappa_{max}}{m_{min}}\|\rho\| \\
&\quad + card(\Omega_{\overline{j}}) \cdot \Big(\tfrac{8d\xi_{max}x_{max}}{m_{min}} \big(\tfrac{v_{max}}{\varepsilon^2} + \tfrac{8dx_{max}^2 v_{max}}{\varepsilon^4} \big) + \tfrac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2} \Big) \cdot \|\rho\| \\
&\quad + 0 + 2x_{max} \\
&= card(\Omega_{\overline{j}}) \Big(\tfrac{2\kappa_{max}L_{max}}{\varepsilon^3 m_{min}} + \tfrac{2\kappa_{max}L_{max}d}{\varepsilon m_{min}} + \tfrac{2\kappa_{max}}{m_{min}} + \big(\tfrac{8d\xi_{max}x_{max}}{m_{min}} \big(\tfrac{v_{max}}{\varepsilon^2} + \tfrac{8dx_{max}^2 v_{max}}{\varepsilon^4} \big) \\
&\quad + \tfrac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2} \Big) \Big) \cdot \|\rho\| + 2x_{max}
\end{aligned}
$$

$$
\begin{aligned}
|-\tfrac{d\zeta_{\overline{k},\overline{j}}}{dt}| &= |\tfrac{dH}{dv_{\overline{k},\overline{j}}}| = |\tfrac{d(I)}{dv_{\overline{k},\overline{j}}} + \tfrac{d(II)}{dv_{\overline{k},\overline{j}}} + \tfrac{d(III)}{dv_{\overline{k},\overline{j}}} + \tfrac{d(IV)}{dv_{\overline{k},\overline{j}}} + \tfrac{d(V)}{dv_{\overline{k},\overline{j}}}| \\
&\leq |\tfrac{d(I)}{dv_{\overline{k},\overline{j}}}| + |\tfrac{d(II)}{dv_{\overline{k},\overline{j}}}| + |\tfrac{d(III)}{dv_{\overline{k},\overline{j}}}| + |\tfrac{d(IV)}{dv_{\overline{k},\overline{j}}}| + |\tfrac{d(V)}{dv_{\overline{k},\overline{j}}}| \\
&\leq 0 + 0 + card(\Omega_{\overline{j}})\tfrac{8d\xi_{max}x_{max}^2}{m_{min}\varepsilon^2}\|\rho\| + \|\rho\| + 0 \\
&= \Big(card(\Omega_{\overline{j}})\tfrac{8d\xi_{max}x_{max}^2}{m_{min}\varepsilon^2} + 1 \Big)\|\rho\|
\end{aligned}
$$

So we have:

$$
\begin{aligned}
\tfrac{d}{dt}\|\rho\| &= \tfrac{d}{dt}\sqrt{\lambda_{1,1}^2+\cdots+\lambda_{d,M}^2+\zeta_{1,1}^2+\cdots+\zeta_{d,M}^2} \\
&= \tfrac{\lambda_{1,1}\cdot\dot{\lambda_{1,1}}+\cdots+\lambda_{d,M}\cdot\dot{\lambda_{d,M}}+\zeta_{1,1}\cdot\dot{\zeta_{1,1}}+\cdots+\zeta_{d,M}\cdot\dot{\zeta_{d,M}}}{\sqrt{\lambda_{1,1}^2+\cdots+\lambda_{d,M}^2+\zeta_{1,1}^2+\cdots+\zeta_{d,M}^2}} \\
&\le |\dot{\lambda_{1,1}}+\cdots+\dot{\lambda_{d,m}}+\dot{\zeta_{1,1}}+\cdots+\dot{\zeta_{d,m}}| \\
&\le |\tfrac{\lambda_{1,1}}{dt}|+\cdots+|\tfrac{\lambda_{d,M}}{dt}|+|\tfrac{\zeta_{1,1}}{dt}|+\cdots+|\tfrac{\zeta_{d,M}}{dt}| \\
&\le d\sum_{\bar{j}}^{M}\Bigg(card(\Omega_{\bar{j}})\Big(\tfrac{2\kappa_{max}L_{max}}{\varepsilon^3 m_{min}}+\tfrac{2\kappa_{max}L_{max}d}{\varepsilon m_{min}}+\tfrac{2\kappa_{max}}{m_{min}} \\
&\quad +\Big(\tfrac{8d\xi_{max}x_{max}}{m_{min}}\big(\tfrac{v_{max}}{\varepsilon^2}+\tfrac{8dx_{max}^2 v_{max}}{\varepsilon^4}\big) \\
&\quad +\tfrac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2}\Big)\Big)\cdot\|\rho\|+2x_{max}\Bigg) \\
&\quad +d\sum_{\bar{j}}^{M}\Big(card(\Omega_{\bar{j}})\tfrac{8d\xi_{max}x_{max}^2}{m_{min}\varepsilon^2}+1\Big)\cdot\|\rho\| \\
&\xrightarrow{n=\sum_{\bar{j}}^{M}card(\Omega_{\bar{j}})} dn\Big(\tfrac{2\kappa_{max}L_{max}}{\varepsilon^3 m_{min}}+\tfrac{2\kappa_{max}L_{max}d}{\varepsilon m_{min}}+\tfrac{2\kappa_{max}}{m_{min}} \\
&+ \quad \big(\tfrac{8d\xi_{max}x_{max}}{m_{min}}\big(\tfrac{v_{max}}{\varepsilon^2}+\tfrac{8dx_{max}^2 v_{max}}{\varepsilon^4}\big) \\
&\quad +\tfrac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2}\big)+\tfrac{8\xi_{max}x_{max}^2}{m_{min}\varepsilon^2}+1\Big)\cdot\|\rho\|+2dMx_{max}
\end{aligned}
$$

Then Gronwall's Lemma says ρ is also bounded.

Now we have proved both z and ρ are bounded. We will bring these two bound in to the next stage.

A.2 Lipschitz constants of Hamiltonian

Before marching to constants, we'd like to introduce two useful lemmas.

Lemma A.1 *Let $F(u_1,u_2,...,u_n)$ be a function of $\{u_1,...,u_n\}$, and x a variable of some u_k. Denote $\Omega := \{k \mid x \text{ is the variable of } u_k\}$. Assume following inequality satisfies:*

$$
\begin{aligned}
|F(u_1,...,u_i^{(1)},...,u_n)-F(u_1,...,u_i^{(2)},...,u_n)| &\le A_i \mid u_i^{(1)}-u_i^{(2)}| \quad for \quad i\in\Omega \\
|u_j(x^{(1)})-u_j(x^{(2)})| &\le B_j \mid x^{(1)}-x^{(2)}| \quad for \quad j\in\Omega
\end{aligned}
$$

for A_i and B_j nonnegative. Then the Lipschitz constant of F w.r.t x is computed as:

$$C=\sum_{j\in\Omega}A_jB_j$$

That is $|F(x^{(1)})-F(x^{(2)})|\le\sum_{j\in\Omega}A_jB_j|x^{(1)}-x^{(2)}|$

Proof: Let $\{\overline{u_j}\}$ where $j = 1, 2, ..., p$ denotes the collection of functions which depend on the variable x, then:

$$
\begin{aligned}
|F(x^{(1)}) - F(x^{(2)})| &= |F(\overline{u_1}(x^{(1)}), ..., \overline{u_p}(x^{(1)})) - F(\overline{u_1}(x^{(2)}), ..., \overline{u_p}(x^{(2)}))| \\
&= \Big| F(\overline{u_1}(x^{(1)}), \overline{u_2}(x^{(1)}), ..., \overline{u_p}(x^{(1)})) \\
&\quad - F(\overline{u_1}(x^{(2)}), \overline{u_2}(x^{(1)}), ..., \overline{u_p}(x^{(1)})) \\
&\quad + F(\overline{u_1}(x^{(2)}), \overline{u_2}(x^{(1)}), ..., \overline{u_p}(x^{(1)})) \\
&\quad - F(\overline{u_1}(x^{(2)}), \overline{u_2}(x^{(2)}), ..., \overline{u_p}(x^{(1)})) \\
&\quad + F(..., \overline{u_2}(x^{(2)}), \overline{u_3}(x^{(1)}), ..., \overline{u_p}(x^{(1)})) \\
&\quad - F(..., \overline{u_2}(x^{(2)}), \overline{u_3}(x^{(2)}), ..., \overline{u_p}(x^{(1)})) \\
&\quad \vdots \\
&\quad + F(\overline{u_1}(x^{(2)}), ..., \overline{u_p}(x^{(1)})) - F(\overline{u_1}(x^{(2)}), ..., \overline{u_p}(x^{(2)})) \Big| \\
&\leq \Big| F(\overline{u_1}(x^{(1)}), \overline{u_2}(x^{(1)}), ..., \overline{u_p}(x^{(1)})) \\
&\quad - F(\overline{u_1}(x^{(2)}), \overline{u_2}(x^{(1)}), ..., \overline{u_p}(x^{(1)})) \Big| \\
&\quad + \Big| F(\overline{u_1}(x^{(2)}), \overline{u_2}(x^{(1)}), ..., \overline{u_p}(x^{(1)})) \\
&\quad - F(\overline{u_1}(x^{(2)}), \overline{u_2}(x^{(2)}), ..., \overline{u_p}(x^{(1)})) \Big| \\
&\quad \vdots \\
&\quad + \Big| F(\overline{u_1}(x^{(2)}), ..., \overline{u_p}(x^{(1)})) - F(\overline{u_1}(x^{(2)}), ..., \overline{u_p}(x^{(2)})) \Big| \\
&\leq A_1 |\overline{u_1}(x^{(1)}) - \overline{u_1}(x^{(2)})| + \cdots + A_p |\overline{u_p}(x^{(1)}) - \overline{u_p}(x^{(2)})| \\
&\leq A_1 B_1 |x^{(1)} - x^{(2)}| + A_2 B_2 |x^{(1)} - x^{(2)}| \\
&\quad + \cdots + A_p B_p |x^{(1)} - x^{(2)}| \\
&= \textstyle\sum_{j \in \Omega} A_j B_j |x^{(1)} - x^{(2)}|
\end{aligned}
$$

□

Lemma A.2 *Let f and g satisfy:*

$$\begin{array}{rcl} |f(x)| & \leq & M_1 \\ |g(x)| & \leq & M_2 \\ |f(x_1) - f(x_2)| & \leq & C_1|x_1 - x_2| \\ |g(x_1) - g(x_2)| & \leq & C_2|x_1 - x_2| \end{array}$$

where M_1, M_2, C_1 and C_2 are nonnegative constants. Then the function defined as $h := f \cdot g$ satisfies:

$$|h(x_1) - h(x_2)| \leq K|x_1 - x_2|$$

where $K = M_1C_1 + M_2C_2$.

Proof:

$$\begin{array}{rcl} |h(x_1) - h(x_2)| & = & |f(x_1) \cdot g(x_1) - f(x_2) \cdot g(x_2)| \\ & = & |f(x_1) \cdot g(x_1) - f(x_1) \cdot g(x_2) + f(x_1) \cdot g(x_2) - f(x_2) \cdot g(x_2)| \\ & \leq & \Big|f(x_1)\big(g(x_1) - g(x_2)\big)\Big| + \Big|g(x_2)\big(f(x_1) - f(x_2)\big)\Big| \\ & \leq & |f(x_1)| \cdot |g(x_1) - g(x_2)| + |g(x_2)| \cdot |f(x_1) - f(x_2)| \\ & \leq & M_1C_1|x_1 - x_2| + M_2C_2|x_1 - x_2| \\ & = & (M_1C_1 + M_2C_2)|x_1 - x_2| \end{array}$$

□

The Hamiltonian is minimized when, $L_{ij} = L_{ij}^-$ or $L_{ij} = L_{ij}^+$. Let $\{L_{ij}^*\}$ denote the minimizer set of then Hamiltonian. Recall that the Hamiltonian H defined as $\mathbb{H}(p, x) := \min_{L \in \mathbb{A}^n} \Big\{f(z, L) \cdot \rho + h(z, L)\Big\}, \quad \forall \rho, z \in \mathbb{R}^n$ where f, z and ρ is defined in (38) and (36). We express the Hamiltonian in following way:

$$
\begin{aligned}
\mathbb{H} = & \min_{L\in\mathbb{A}^n}\Big\{f(z,L)\cdot\rho+h(z,L)\Big\} \\
= & \min_{L\in\mathbb{A}^n}\{\sum_{j=1}^{M}(\sum_{k=1}^{d}(\lambda_{k,j}v_{k,j})+\sum_{k=1}^{d}(\zeta_{k,j}F_{k,j}))+h\} \\
& +\sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_j}\zeta_{k,j}\tfrac{-\kappa_{ij}(x_{k,j}-x_{k,i})}{m_j} \qquad \cdots \quad (II) \\
& +\sum_{j=1}^{M}\sum_{k=1}^{d}\sum_{i\in\Omega_j}\zeta_{k,j}(-\xi_{ij}(x_{k,j}-x_{k,i})(\tfrac{\sum_{s=1}^{d}(v_{s,j}-v_{s,i})(x_{s,j}-x_{s,i})}{l_{ij}^2 m_j})) \qquad \cdots \quad (III) \\
& +\sum_{j=1}^{M}\sum_{k=1}^{d}(\lambda_{k,j}v_{k,j}) \qquad \cdots \quad (IV) \\
& +\sum_{j=1}^{M}\sum_{k=1}^{d}(x_{k,j}-X_{k,j})^2 \qquad \cdots \quad (V)
\end{aligned}
$$

Lipschitz constants along different directions can be obtained as follows:

Part (I): Notice that forces on one spring between two connected particles have same magnitude but opposite direction, thus we can count forces not for each node but every springs, such that

$$
(I)=\left\{\min_{L\in\mathbb{A}}\sum_{\substack{(i,j)\in\mathcal{S}\\ i<j}}\sum_{k=1}^{d}\left(\frac{(\frac{\zeta_{k,j}}{m_j}-\frac{\zeta_{k,i}}{m_i})\kappa_{ij}(x_{k,j}-x_{k,i})}{\sqrt{\sum_{s=1}^{d}(x_{s,j}-x_{s,i})^2}}\cdot L_{ij}\right)\right\}
$$

Define scalar function

$$
P_{ij}:=\frac{\sum_{k=1}^{d}(\frac{\zeta_{k,j}}{m_j}-\frac{\zeta_{k,i}}{m_i})(x_{k,i}-x_{k,j})\cdot\kappa_{ij}}{\sqrt{\sum_{k=1}^{d}(x_{k,i}-x_{k,j})^2}},
$$

and

$$
S_{ij}:=S_{ij}(P_{ij})=\min_{L_{ij}}\{P_{ij}\cdot L_{ij}\}
$$

for all $(i,j)\in\mathbb{S}$.
S_{ij} is a linear function, so we can write S_{ij} as:

$$
S_{ij}=\begin{cases}P_{ij}\cdot L_{ij}^{+}, & P_{ij}<0\\ P_{ij}\cdot L_{ij}^{-}, & P_{ij}\geq 0\end{cases}
$$

And

$$
(I)=\sum_{\substack{(i,j)\in\mathcal{S}\\ i<j}}S_{ij}
$$

The Lipschitz constant of (I) w.r.t $\zeta_{\overline{k},\overline{j}}$ for some $\overline{k},\overline{j}$ is:

$$\begin{aligned} Lip_{\zeta_{\overline{k},\overline{j}}}(I) &\leq \sum_{i\in\Omega_{\overline{j}}} Lip_{P_{i\overline{j}}}(S_{i\overline{j}}) \cdot Lip_{\zeta_{\overline{k},i}}(P_{i\overline{j}}) \\ &\leq \sum_{i\in\Omega_{\overline{j}}} L^{+}_{i\overline{j}} \cdot \frac{|x_{\overline{k},i} - x_{\overline{k},j}|}{m_{\overline{j}}\sqrt{\sum_{s=1}^{d}(x_{s,i} - x_{s,\overline{j}})^2}} \kappa_{i\overline{j}} \\ &\leq \sum_{i\in\Omega_{\overline{j}}} \frac{L^{+}_{i\overline{j}} \cdot \kappa_{i\overline{j}}}{m_{\overline{j}}} \\ &= card(\Omega_{\overline{j}}) \frac{L_{max} \cdot \kappa_{max}}{m_{min}} \end{aligned}$$

And the Lipschitz constant of (I) w.r.t $x_{\overline{k},\overline{j}}$ for some $\overline{j},\overline{k}$ is:

$$\begin{aligned} Lip_{x_{\overline{k},\overline{j}}}(I) &\leq \sum_{i\in\Omega_{\overline{j}}} Lip_{P_{i\overline{j}}}(S_{i\overline{j}}) \cdot Lip_{x_{k,\overline{j}}}(P_{i\overline{j}}) \\ &\leq \sum_{i\in\Omega_{\overline{j}}} L^{+}_{i\overline{j}} \Big(\kappa_{i\overline{j}} L^{*}_{i\overline{j}} | \tfrac{1}{\varepsilon^3} | + \kappa_{i\overline{j}} L^{*}_{i\overline{j}} | \frac{\frac{1}{m_{min}} \sum_{k=1}^{d} \left(|\zeta_{k,\overline{j}}| + |\zeta_{k,i}| \right)}{\sqrt{\sum_{s=1}^{d}(x_{s,\overline{j}} - x_{s,i})^2}} | \Big) \\ &\leq card(\Omega_{\overline{j}}) \kappa_{max} L_{max} / \varepsilon^3 + 2 card(\Omega_{\overline{j}}) \kappa_{max} L_{max} d \|\rho\| / \varepsilon m_{min} \end{aligned}$$

The Lipschitz constant of (II) w.r.t $\zeta_{\overline{k},\overline{j}}$ for some $\overline{k},\overline{j}$ is:

$$\begin{aligned} Lip_{\zeta_{\overline{k},\overline{j}}}(II) &= \sum_{i\in\Omega_{\overline{j}}} (-\frac{\kappa_{i\overline{j}}}{m_{\overline{j}}})(x_{\overline{k},\overline{j}} - x_{\overline{k},i}) \\ &\leq card(\Omega_{\overline{j}}) \frac{2\kappa_{max}}{m_{min}} x_{max} \end{aligned}$$

And the Lipschitz constant of (II) w.r.t $x_{\overline{k},\overline{j}}$ for some $\overline{j},\overline{k}$ is:

$$\begin{aligned} Lip_{x_{\overline{k},\overline{j}}}(II) &= \sum_{i\in\Omega_{\overline{j}}} (-\frac{\kappa_{i\overline{j}}}{m_{\overline{j}}})(\zeta_{\overline{k},\overline{j}} - \zeta_{\overline{k},i}) \\ &\leq card(\Omega_{\overline{j}}) \frac{2\kappa_{max}}{m_{min}} \|\rho\| \end{aligned}$$

The Lipschitz constant of (III) w.r.t $\zeta_{\overline{k},\overline{j}}$ for some $\overline{k},\overline{j}$ is:

$$\begin{aligned} Lip_{\zeta_{\overline{k},\overline{j}}}(III) &= \sum_{i\in\Omega_{\overline{j}}} \sum_{k=1}^{d} \left(-\frac{\xi_{ij}}{m_{\overline{j}}} (x_{k,j} - x_{k,i}) \frac{\sum_{s=1}^{d} (v_{s,j} - v_{s,i})(x_{s,j} - x_{s,i})}{\sum_{l=1}^{d} (x_{l,j} - x_{l,i})^2} \right) \\ &\leq \sum_{i\in\Omega_{\overline{j}}} \sum_{k=1}^{d} \left(\frac{D_{ij}}{m_{\overline{j}}} (|x_{k,j}| + |x_{k,i}|) \frac{\sum_{s=1}^{d} (|v_{s,j}| + |v_{s,i}|)(|x_{s,j}| + |x_{s,i}|)}{\sum_{l=1}^{d} (x_{l,j} - x_{l,i})^2} \right) \\ &\leq card(\Omega_{\overline{j}}) \frac{8 d \xi_{max} x^2_{max} v_{max}}{m_{min} \varepsilon^2} \end{aligned}$$

And the Lipschitz constant of (III) w.r.t $x_{\overline{k},\overline{j}}$ for some $\overline{j},\overline{k}$ is:

$$\begin{aligned} Lip_{x_{\overline{k},\overline{j}}}(III) \leq\ & card(\Omega_{\overline{j}}) \cdot \Big(\frac{8 d \xi_{max} x_{max}}{m_{min}} \big(\frac{v_{max}}{\varepsilon^2} + \frac{8 d x^2_{max} v_{max}}{\varepsilon^4} \big) \\ & + \frac{8 d \xi_{max} x_{max} v_{max}}{m_{min} \varepsilon^2} \Big) \cdot \|\rho\| \end{aligned}$$

And the Lipschitz constant of (III) w.r.t $v_{\overline{k},\overline{j}}$ for some $\overline{j},\overline{k}$ is:

$$Lip_{v_{\overline{k},\overline{j}}}(III) \leq card(\Omega_{\overline{j}})\frac{8d\xi_{max}x_{max}^2}{m_{min}\varepsilon^2}\|\rho\|$$

The Lipschitz constant of (IV) w.r.t $\zeta_{\overline{k},\overline{j}}$ for some $\overline{k},\overline{j}$ is:

$$\begin{aligned} Lip_{\lambda_{\overline{k},\overline{j}}}(IV) &= v_{\overline{k},\overline{j}} \\ &\leq v_{max} \end{aligned}$$

And the Lipschitz constant of (IV) w.r.t $v_{\overline{k},\overline{j}}$ for some $\overline{j},\overline{k}$ is:

$$\begin{aligned} Lip_{v_{\overline{k},\overline{j}}}(IV) &= |\lambda_{\overline{k},\overline{j}}| \\ &\leq \|\rho\| \end{aligned}$$

And the Lipschitz constant of (V) w.r.t $x_{\overline{k},\overline{j}}$ for some $\overline{j},\overline{k}$ is:

$$\begin{aligned} Lip_{x_{\overline{k},\overline{j}}}(V) &= 2|x_{\overline{k},\overline{j}} - X_{\overline{k},\overline{j}}| \\ &\leq 2x_{max} \end{aligned}$$

We conclude that for Hamiltonian $\mathbb{H}$ has following properties in this project:

$$\begin{aligned} Lip_{\zeta_{\overline{k},\overline{j}}}\mathbb{H} &\leq card(\Omega_{\overline{j}})\tfrac{L_{max}\cdot\kappa_{max}}{m_{min}} + card(\Omega_{\overline{j}})\tfrac{2\kappa_{max}}{m_{min}}x_{max} \\ &\quad + card(\Omega_{\overline{j}})\tfrac{8d\xi_{max}x_{max}^2 v_{max}}{m_{min}\varepsilon^2} \\ Lip_{\lambda_{\overline{k},\overline{j}}}\mathbb{H} &\leq v_{max} \\ Lip_{x_{\overline{k},\overline{j}}}\mathbb{H} &\leq 2card(\Omega_{\overline{j}})\kappa_{max}L_{max}\|\rho\|/\varepsilon^3 m_{min} \\ &\quad + 2card(\Omega_{\overline{j}})\kappa_{max}L_{max}d\|\rho\|/\varepsilon m_{min} \\ &\quad + card(\Omega_{\overline{j}})\tfrac{2\kappa_{max}}{m_{min}}\|\rho\| \\ &\quad + card(\Omega_{\overline{j}})\cdot\Big(\tfrac{8d\xi_{max}x_{max}}{m_{min}}(\tfrac{v_{max}}{\varepsilon^2} + \tfrac{8dx_{max}^2 v_{max}}{\varepsilon^4}) \\ &\quad + \tfrac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2}\Big)\cdot\|\rho\| + 0 + 2x_{max} \\ &= card(\Omega_{\overline{j}})\Big(\tfrac{2\kappa_{max}L_{max}}{\varepsilon^3 m_{min}} + \tfrac{2\kappa_{max}L_{max}d}{\varepsilon m_{min}} + \tfrac{2\kappa_{max}}{m_{min}} \\ &\quad + \big(\tfrac{8d\xi_{max}x_{max}}{m_{min}}(\tfrac{v_{max}}{\varepsilon^2} + \tfrac{8dx_{max}^2 v_{max}}{\varepsilon^4}) \\ &\quad + \tfrac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2}\big)\Big)\cdot\|\rho\| + 2x_{max} \\ &\leq \max\Big\{card(\Omega_{\overline{j}})\Big(\tfrac{2\kappa_{max}L_{max}}{\varepsilon^3 m_{min}} + \tfrac{2\kappa_{max}L_{max}d}{\varepsilon m_{min}} + \tfrac{2\kappa_{max}}{m_{min}} \\ &\quad + \tfrac{8d\xi_{max}x_{max}}{m_{min}}(\tfrac{v_{max}}{\varepsilon^2} + \tfrac{8dx_{max}^2 v_{max}}{\varepsilon^4}) + \tfrac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2}\Big) \\ &\quad , 2x_{max}\Big\}(\|\rho\| + 1) \\ Lip_{v_{\overline{k},\overline{j}}}\mathbb{H} &\leq card(\Omega_{\overline{j}})\tfrac{8d\xi_{max}x_{max}^2}{m_{min}\varepsilon^2}\|\rho\| + \|\rho\| \\ &\leq \big(card(\Omega_{\overline{j}})\tfrac{8d\xi_{max}x_{max}^2}{m_{min}\varepsilon^2} + 1\big)(\|\rho\| + 1) \end{aligned}$$

By using Lemma 1 and Lemma 2 Hamiltonian $\mathbb{H}$ satisfies:

$$\begin{aligned} |\mathbb{H}(x,\lambda_1)-\mathbb{H}(x,\lambda_2)| &\leq C_1|\lambda_1-\lambda_2|, \\ |\mathbb{H}(x,\lambda)-\mathbb{H}(y,\lambda)| &\leq C_2(1+|\lambda|)|x-y| \end{aligned}$$

for

$$\begin{aligned} C_1 &= \max\Big\{\max\{card(\Omega_{\bar{j}})\}\big(\tfrac{L_{max}\cdot\kappa_{max}}{m_{min}}+\tfrac{2\kappa_{max}}{m_{min}}x_{max}+\tfrac{8d\xi_{max}x_{max}^2v_{max}}{m_{min}\varepsilon^2}\big) \\ &\quad ,v_{max}\Big\} \\ &\leq \max\Big\{(M-1)\big(\tfrac{L_{max}\cdot\kappa_{max}}{m_{min}}+\tfrac{2\kappa_{max}}{m_{min}}x_{max}+\tfrac{8d\xi_{max}x_{max}^2v_{max}}{m_{min}\varepsilon^2}\big),v_{max}\Big\} \\ C_2 &= \max\Bigg\{\max\Bigg\{\max\{card(\Omega_{\bar{j}})\}\Big(\tfrac{2\kappa_{max}L_{max}}{\varepsilon^3m_{min}}+\tfrac{2\kappa_{max}L_{max}d}{\varepsilon m_{min}}+\tfrac{2\kappa_{max}}{m_{min}} \\ &\quad +\tfrac{8d\xi_{max}x_{max}}{m_{min}}\big(\tfrac{v_{max}}{\varepsilon^2}+\tfrac{8dx_{max}^2v_{max}}{\varepsilon^4}\big) \\ &\quad +\tfrac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2}\Big),2x_{max}\Bigg\},\big(card(\Omega_{\bar{j}})\tfrac{8d\xi_{max}x_{max}^2}{m_{min}\varepsilon^2}+1\big)\Bigg\} \\ &= \max\Bigg\{\max\{card(\Omega_{\bar{j}})\}\Big(\tfrac{2\kappa_{max}L_{max}}{\varepsilon^3m_{min}}+\tfrac{2\kappa_{max}L_{max}d}{\varepsilon m_{min}}+\tfrac{2\kappa_{max}}{m_{min}} \\ &\quad +\tfrac{8d\xi_{max}x_{max}}{m_{min}}\big(\tfrac{v_{max}}{\varepsilon^2}+\tfrac{8dx_{max}^2v_{max}}{\varepsilon^4}\big)+\tfrac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2}\Big) \\ &\quad ,2x_{max},\max\{card(\Omega_{\bar{j}})\}\tfrac{8d\xi_{max}x_{max}^2}{m_{min}\varepsilon^2}+1\Bigg\} \\ &\leq \max\Bigg\{(M-1)\Big(\tfrac{2\kappa_{max}L_{max}}{\varepsilon^3m_{min}}+\tfrac{2\kappa_{max}L_{max}d}{\varepsilon m_{min}}+\tfrac{2\kappa_{max}}{m_{min}} \\ &\quad +\tfrac{8d\xi_{max}x_{max}}{m_{min}}\big(\tfrac{v_{max}}{\varepsilon^2}+\tfrac{8dx_{max}^2v_{max}}{\varepsilon^4}\big) \\ &\quad +\tfrac{8d\xi_{max}x_{max}v_{max}}{m_{min}\varepsilon^2}\Big),2x_{max},(m-1)\tfrac{8d\xi_{max}x_{max}^2}{m_{min}\varepsilon^2}+1\Bigg\} \end{aligned}$$

B Appendix B: Matlab code

preSymbolic.m

```
function preSymbolic()
    clear
    close all
    clc
    global currentDirectory globalVariables

    dim = 2;
    Spring_limit = 1/2;
    N = 512;
    mu1 = 100;
    mu2 = 500;
    mu3 = 1;
```

```
13      currentDirectory = 'D:\workwithmath\Thesis\dynamic';
14
15      delete([currentDirectory '\J\*.*']);
16
17      syms Δ ΔT
18
19      globalVariables = [ΔT, Δ];
20
21      MASS = [
22          3;
23          3;
24          3;
25          3;
26          0;
27          0;
28          ];
29
30      SPRINGS = [
31          1, 2, 20, 40;
32          1, 2, 20, 40;
33          1, 2, 20, 40;
34          1, 2, 20, 40;
35          1, 2, 20, 40;
36          1, 2, 20, 40;
37          1, 2, 20, 40;
38          1, 2, 20, 40;
39          ];
40
41      CONNECTION = [
42          +1, 0,-1, 0, 0, 0;%3-1
43          +1, 0, 0,-1, 0, 0;%4-1
44          +1,-1, 0, 0, 0, 0;%2-1
45           0,+1, 0,-1, 0, 0;%4-2
46           0, 0,+1, 0,-1, 0;%5-3
47           0, 0,+1, 0, 0,-1;%6-3
48           0, 0,+1,-1, 0, 0;%4-3
49           0, 0, 0,+1, 0,-1;%6-4
50          ];
51
52      POSITION = zeros(dim*2, size(MASS,1));
53      POSITION(:,5) = columnVector([0,0,0,0]);
54      POSITION(:,6) = columnVector([0,1,0,0]);
55
56
57
58      if(size(CONNECTION,2)≠size(MASS,1) || ...
            size(CONNECTION,1)≠size(SPRINGS,1))
59          BB(2,0.2);
60          disp('Dimension fetal error!');
61          return;
62      end
63
64      Param = [
65          java.lang.String('N');
66          java.lang.String('MassNr');
67          java.lang.String('dim');
68          ];
69
70      counter = 0;
71      for i=1:size(MASS,1)
72          if(MASS(i)≠0)
73              counter = counter + 1;
```

```
74          end
75      end
76      ParamVal = [
77          N;
78          counter;
79          dim;
80          ];
81      generateInitialization(Param, ParamVal);
82
83      strBufferZtotal = '';
84      strBufferLamdatotal = '';
85      strBufferXRef = '';
86      strBufferXRefN = '';
87      for i=1:size(MASS,1)
88          if(MASS(i)≠0)
89
90              part1 = '';
91              part2 = '';
92              for j=1:dim
93                  part1 = [part1, 'sym(''X_',num2str(i),...
94                          '_',num2str(j),'''); '];
95                  part2 = [part2, 'sym(''X_',num2str(i),...
96                          '_',num2str(dim+j),'''); '];
97              end
98              strBufferZ = ['X_',num2str(i),' = [',part1,part2,'];'];
99
100             part1 = '';
101             part2 = '';
102             for j=1:dim
103                 part1 = [part1, 'sym(''Lamda_',num2str(i),...
104                         '_',num2str(j),'''); '];
105                 part2 = [part2, 'sym(''Lamda_',num2str(i),...
106                         '_',num2str(dim+j),'''); '];
107             end
108             strBufferLamda = ['Lamda_',num2str(i),' = ...
                    [',part1,part2,'];'];
109
110             part1 = '';
111             part2 = '';
112             for j=1:dim
113                 part1 = [part1, 'sym(''XRef_',num2str(i),...
114                         '_',num2str(j),'''); '];
115                 part2 = [part2, 'sym(''XRef_',num2str(i),...
116                         '_',num2str(dim+j),'''); '];
117             end
118             strBufferRef = ['XRef_',num2str(i),' = ...
                    [',part1,part2,'];'];
119
120             part1 = '';
121             part2 = '';
122             for j=1:dim
123                 part1 = [part1, 'sym(''XRefN_',num2str(i),...
124                         '_',num2str(j),'''); '];
125                 part2 = [part2, 'sym(''XRefN_',num2str(i),...
126                         '_',num2str(dim+j),'''); '];
127             end
128             strBufferRefN = ['XRefN_',num2str(i),' = ...
                    [',part1,part2,'];'];
129
130             part1 = '';
131             part2 = '';
132             for j=1:dim
```

```
133                 part1 = [part1, 'sym(''XInit_',num2str(i),...
134                         '_',num2str(j),'''); '];
135                 part2 = [part2, 'sym(''XInit_',num2str(i),...
136                         '_',num2str(dim+j),'''); '];
137             end
138             strBufferInit = ['XInit_',num2str(i),' = ...
                    [',part1,part2,'];'];
139
140             eval([strBufferZ, strBufferLamda, strBufferRef, ...
                    strBufferRefN, strBufferInit]);
141
142             strBufferZtotal = [strBufferZtotal,'X_',num2str(i),';'];
143             strBufferLamdatotal = ...
                    [strBufferLamdatotal,'Lamda_',num2str(i),';'];
144             strBufferXRef = [strBufferXRef,'XRef_',num2str(i),';'];
145             strBufferXRefN = ...
                    [strBufferXRefN,'XRefN_',num2str(i),';'];
146         else
147 %             commandBuffer
148             eval(['X_', num2str(i), '=', ...
                    'POSITION(:,',num2str(i),');']);
149             eval(['XInit_', num2str(i), '=', ...
                    'POSITION(:,',num2str(i),');']);
150             eval(['XRef_', num2str(i), '=', ...
                    'POSITION(:,',num2str(i),');']);
151             eval(['XRefN_', num2str(i), '=', ...
                    'POSITION(:,',num2str(i),');']);
152         end
153
154     end
155     eval(['X = [',strBufferZtotal,'];']);
156     eval(['Lamda = [',strBufferLamdatotal,'];']);
157     eval(['XRef = [',strBufferXRef,'];']);
158     eval(['XRefN = [',strBufferXRefN,'];']);
159
160     variableBuffer = '';
161
162     for i=1:size(SPRINGS,1)
163         variableBuffer = [variableBuffer,...
164             'H',num2str(i),' ', 'D',num2str(i),' ', ...
                    'L',num2str(i),' '...
165             ];
166         globalVariables = [globalVariables, ['H',num2str(i)],' ...
                ', ['D',num2str(i)],' '];
167     end
168     eval(['syms ',variableBuffer]);
169
170     SPRINGS_UPP_LOW = sym(SPRINGS);
171     for i=1:size(SPRINGS,1)
172         [s,t] = findStartEndPoints(CONNECTION(i,:));
173 %         X1_Init = eval(['XInit_', num2str(s)]);
174 %         X2_Init = eval(['XInit_', num2str(t)]);
175         X1_Ref = eval(['XRefN_', num2str(s),'']);
176         X2_Ref = eval(['XRefN_', num2str(t)]);
177
178         L_UPP = ...
                ((X2_Ref(1)-X1_Ref(1))^2+((X2_Ref(2)-X1_Ref(2))^2))^.5;
179         L_LOW = (1-Spring_limit)*L_UPP;
180         L_UPP = (1+Spring_limit)*L_UPP;
181
182         SPRINGS_UPP_LOW(i,1) = L_UPP;
183         SPRINGS_UPP_LOW(i,2) = L_LOW;
```

```
184     end
185     clear L_UPP L_LOW X1_Ref X2_Ref
186
187     SPRINGS_VECTOR = sym(zeros(size(SPRINGS,1), dim*2+1));
188     for i=1:size(CONNECTION,1)
189         [s,t] = findStartEndPoints(CONNECTION(i,:));
190         X_START = eval(['X_',num2str(s)]);
191         X_TERMINAL = eval(['X_',num2str(t)]);
192
193         part1 = '';
194         part2 = '';
195         for j=1:dim
196             part1 = [part1, 'X_TERMINAL(',num2str(j),'), '];
197             part2 = [part2, 'X_START(',num2str(j),'), '];
198         end
199
200         Vector = eval(['[',part1,'] - [',part2,']']);
201
202         Vector_Norm = (Vector(2)^2 + Vector(1)^2)^.5;
203         UnitVector = Vector/Vector_Norm;
204
205         SPRINGS_VECTOR(i, 1) = Vector_Norm;
206         SPRINGS_VECTOR(i, 2) = Vector(1);
207         SPRINGS_VECTOR(i, 3) = Vector(2);
208         SPRINGS_VECTOR(i, 4) = UnitVector(1);
209         SPRINGS_VECTOR(i, 5) = UnitVector(2);
210
211     end
212
213     SPRINGS_FORCE = sym(zeros(size(SPRINGS,1), dim));
214     for i=1:size(SPRINGS,1)
215         H = eval(['H',num2str(i)]);
216         D = eval(['D',num2str(i)]);
217         L = eval(['L',num2str(i)]);
218
219         Vector_Norm = SPRINGS_VECTOR(i,1);
220         %Vector = [SPRINGS_VECTOR(i,2); SPRINGS_VECTOR(i,3)];
221
222         Vector_Unit = [SPRINGS_VECTOR(i,4); SPRINGS_VECTOR(i,5)];
223
224         Force_Hook = H*(L-Vector_Norm);
225
226         [s,t] = findStartEndPoints(CONNECTION(i,:));
227
228         part1 = '';
229         part2 = '';
230         for j=1:dim
231             part1 = [part1, 'X_',num2str(t),'(', ...
                        num2str(dim+j),'), '];
232             part2 = [part2, 'X_',num2str(s),'(', ...
                        num2str(dim+j),'), '];
233         end
234
235         V = columnVector(eval(['[', part1, '] - [', part2,']']));
236
237         Force_Damper = 0;
238         for k=1:dim
239             %Force_Damper = D*(V.'*Vector_Unit);
240             Force_Damper = Force_Damper + D*(V(k)*Vector_Unit(k));
241         end
242
243         Force_NORM = Force_Hook - Force_Damper;
```

```
244             Force = Force_NORM * Vector_Unit;
245             for j=1:dim
246                 Vec = zeros(dim,1);
247                 Vec(j) = 1;
248                 SPRINGS_FORCE(i,j) = Force.'*Vec;
249             end
250         end
251
252         MASS_FORCE = sym(zeros(size(MASS,1),dim));
253         for i=1:size(MASS,1)
254             if(MASS(i)≠0)
255                 SpringForce = zeros(1,dim);
256                 for j = 1:size(CONNECTION,1)
257                     if(CONNECTION(j,i)==1 || CONNECTION(j,i)==-1)
258                         SpringForce = SpringForce + ...
                                CONNECTION(j,i)*SPRINGS_FORCE(j,:);
259                     end
260                 end
261                 MASS_FORCE(i,:) = SpringForce;
262             end
263         end
264
265         f = [];
266         for i=1:size(MASS,1)
267             if(MASS(i)≠0)
268                 for j=1:dim
269                     f = [f; eval(['X_', num2str(i), ...
                            '(',num2str(j+dim),')'])];
270                 end
271                 for j=1:dim
272                     f = [f; MASS_FORCE(i,j)/MASS(i)];
273                 end
274             end
275         end
276
277         %% Running cost h
278         h = 0;
279         for i=1:size(MASS,1)
280             if(MASS(i)≠0)
281                 for j=1:dim
282                     h = h + eval(['mu1*(X_', num2str(i), ...
                            '(',num2str(j),') - XRef_', num2str(i), ...
                            '(',num2str(j),'))^2']);
283                 end
284             end
285         end
286
287         g = 0;
288
289         Dg = jacobian(g, X).';
290
291         Htemp = Lamda.'*f + h;
292
293         strBuffer = '';
294         for i=1:size(MASS,1)
295             if(MASS(i)≠0)
296                 XReftemp = eval(['XRef_',num2str(i)]);
297                 for j=1:size(XReftemp,1)
298                     strBuffer = [strBuffer, ...
                            'XRef_',num2str(i),'_',num2str(j),' '];
299                     strBuffer = [strBuffer, ...
                            'XRefN_',num2str(i),'_',num2str(j),' '];
```

```
300                 strBuffer = [strBuffer, ...
                        'X_',num2str(i),'_',num2str(j),' '];
301                 strBuffer = [strBuffer, ...
                        'Lamda_',num2str(i),'_',num2str(j),' '];
302             end
303         end
304     end
305     eval(['syms ', strBuffer]);
306
307     PANDdSdP = sym(zeros(size(SPRINGS,1),2));
308     for i=1:size(SPRINGS,1)
309         PANDdSdP(i,1) = eval(['diff(Htemp,L',num2str(i),')']);
310         PANDdSdP(i,2) = ...
                eval(['(',class2str(SPRINGS_UPP_LOW(i,2)),'-',...
311                 class2str(SPRINGS_UPP_LOW(i,1)),')/2*tanh(...
312                 (',class2str(PANDdSdP(i,1)),')/Δ)...+(',...
313             class2str(SPRINGS_UPP_LOW(i,2)),'+'...
314             ,class2str(SPRINGS_UPP_LOW(i,1)),')/2']);
315     end
316
317     part1 = '';
318     part2 = '';
319     for i=1:size(SPRINGS,1)
320         part1 = [part1, 'L', num2str(i), ', '];
321         part2 = [part2, '0, '];
322     end
323     C = eval(['subs(Htemp,{',part1,'},{',part2,'});']);
324
325     dHdZ = jacobian(C, X).';
326     dHdLamda = jacobian(C, Lamda).';
327     for i=1:size(SPRINGS,1)
328         dHdZ = dHdZ + PANDdSdP(i,2).*jacobian(PANDdSdP(i,1), X).';
329         dHdLamda = dHdLamda + ...
                PANDdSdP(i,2).*jacobian(PANDdSdP(i,1), Lamda).';
330     end
331
332     % generateJ3(Z, Lamda, dHdZ, dHdLamda, Dg)
333     [X_m, Lamda_m, XRef_m, XRefN_m, dHdZ_m, dHdLamda_m, Dg_m] = ...
            mappingForwardXLamda(X, Lamda, XRef, XRefN, dHdZ, ...
            dHdLamda, Dg);
334     for i=1:size(XRef_m)
335         globalVariables = [globalVariables, XRef_m(i), ' ', ...
                XRefN_m(i), ' '];
336     end
337
338     generateJ3(X_m, Lamda_m, dHdZ_m, dHdLamda_m, Dg_m, N)
339     generateXRefDefine(X);
340     generatePARAMETER_VECTORS(SPRINGS);
341
342     generateFJ(X_m, Lamda_m, XRef_m, dHdZ_m, dHdLamda_m, Dg_m, N)
343     generateEulerZtoLamdaJ(X_m, Lamda_m, XRef_m, dHdZ_m, Dg_m, N)
344     generateLamdaNPlus1FromZNPlus1J(X_m, Dg_m, N)
345     generateGLOBALVARIABLE();
346
347     BB(4,0.05);
348     disp(['preSymbolic finish!', char(globalVariables)]);
349 return;
350
351 function str = class2str(obj)
352     cla = class(obj);
353     if(strcmp(cla,'sym'))
354         str = char(obj);
```

```
355     elseif(strcmp(cla,'double'))
356         str = num2str(obj);
357     else
358         str = obj;
359     end
360 return;
361
362 function [s,t] = findStartEndPoints(V)
363     V = columnVector(V);
364     [minVal, minPos] = min(V);
365     [maxVal, maxPos] = max(V);
366     if(minVal == -1 && maxVal == 1)
367         s = minPos;
368         t = maxPos;
369     end
370 return;
371
372 function [X_m, Lamda_m, XRef_m, XRefN_m, dHdZ_m, dHdLamda_m, ...
        Dg_m] = mappingForwardXLamda(X, Lamda, XRef, XRefN, dHdZ, ...
        dHdLamda, Dg)
373
374     X_m = X;
375     for i=1:size(X,1)
376         X_m(i) = sym(['x', num2str(i)]);
377     end
378     Lamda_m = Lamda;
379     for i=1:size(Lamda,1)
380         Lamda_m(i) = sym(['lamda', num2str(i)]);
381     end
382     XRef_m = XRef;
383     for i=1:size(XRef,1)
384         XRef_m(i) = sym(['Xref', num2str(i)]);
385     end
386
387     XRefN_m = XRefN;
388     for i=1:size(XRefN,1)
389         XRefN_m(i) = sym(['XrefN', num2str(i)]);
390     end
391
392     dHdZ_m = dHdZ;
393     for i=1:size(dHdZ,1)
394         dHdZ_m(i) = subs(dHdZ(i),{X, Lamda, XRef, XRefN},{X_m, ...
            Lamda_m, XRef_m, XRefN_m});
395     end
396     dHdLamda_m = dHdLamda;
397     for i=1:size(dHdLamda,1)
398         dHdLamda_m(i) = subs(dHdLamda(i),{X, Lamda, XRef, ...
            XRefN},{X_m, Lamda_m, XRef_m, XRefN_m});
399     end
400     Dg_m = Dg;
401     for i=1:size(Dg,1)
402         Dg_m(i) = subs(Dg(i),{X, Lamda, XRef, XRefN},{X_m, ...
            Lamda_m, XRef_m, XRefN_m});
403     end
404 return;
```

main.m

```
1
2 clc
3 clear
```

```
close all

GLOBALVARIABLE

currentDirectory = 'D:\workwithmath\Thesis\dynamic';

addpath([currentDirectory,'\J'],[currentDirectory]);


safeTurn = 9;


delete([currentDirectory '\DATA\*.*']);

ParamInit;
T = 30;
ΔT = T/N;

PARAMETER_VECTORS;

if(length(PARAMETER_NAME)≠length(PARAMETER_VALUE) ||...
        length(PARAMETER_MAX)≠length(PARAMETER_VALUE) ||...
        length(PARAMETER_NAME)≠length(PARAMETER_MAX))
    BB(2,0.1);
    disp('Input parameters'' error, HDforward halt.');
end
for i=1:length(PARAMETER_NAME)
    eval([char(PARAMETER_NAME(i)), '=', ...
        num2str(PARAMETER_VALUE(i)), ';']);
    eval([char(PARAMETER_NAME(i)), '_max =', ...
        num2str(PARAMETER_MAX(i)), ';']);
end

Δ = 1;

Δ_max = 0.0001;

%Manually input estimate start and terminate state, only for ...
    setting initial step.
Global_X_init = columnVector([6,1,0,0,6,0,0,0,3,1,0,0,3,0,0,0,]);
Global_X_ref = columnVector([5.72513,1.4676,0,0,5.98164,...
        0.762831,0,0,2.80915,1.03093,0,0,3.08726,0.174978,0,0,]);

IRDifference = 0;
for i=1:MassNr
    if(norm(Global_X_ref([1:dim*2]+(i-1)*dim*2) - ...
        Global_X_init([1:dim*2]+(i-1)*dim*2))>IRDifference)
        IRDifference = norm(Global_X_ref([1:dim*2]+(i-1)*dim*2) ...
            - Global_X_init([1:dim*2]+(i-1)*dim*2));
    end
end
InitStep = 1e-4;
StepPercentage = InitStep/IRDifference;

Z= [];
Lamda = [];
resultSequence = 0;
StepPercentage_TEMP = StepPercentage;
coefficient = 0.25;
Z_LAST_SUCCESS = [];

Theta_M_S = 0*pi/180;
```

```
61  Theta_M_E = 5*pi/180;
62
63  Phi_M_S = 0*pi/180;
64  Phi_M_E = 5*pi/180;
65
66  Theta_N_S = 0*pi/180;
67  Theta_N_E = 15*pi/180;
68
69  Phi_N_S = 0*pi/180;
70  Phi_N_E = 20*pi/180;
71
72  while(StepPercentage<1)
73      if(isempty(Z) || isempty(Lamda))
74          disp(sprintf('Try percentage: %g%%', StepPercentage*100));
75
76          thetaM = StepPercentage*Theta_M_E + ...
                (1-StepPercentage)*Theta_M_S;
77          phiM = StepPercentage*Phi_M_E + (1-StepPercentage)*Phi_M_S;
78
79          thetaN = StepPercentage*Theta_N_E + ...
                (1-StepPercentage)*Theta_N_S;
80          phiN = StepPercentage*Phi_N_E + (1-StepPercentage)*Phi_N_S;
81
82          Ref = computeRef2DVer1(0, T, 1, [0,0.5,0,0], [Theta_M_S, ...
                thetaM], [Phi_M_S, phiM], 3, 0.5, [Theta_N_S, ...
                thetaN], [Phi_N_S, phiN], 3, 0.5, N);
83          X_init = columnVector(Ref(1,:));
84          X_refN = columnVector(Ref(N/2+1,:));
85
86          XRefNDefine
87
88          Z = Ref;
89
90          Z_LAST_SUCCESS = Z;
91          Xref = Z;
92
93          Lamda = EulerZtoLamda(Z, Xref);
94      else
95          StepPercentage = StepPercentage * (1+coefficient);
96          if(StepPercentage>1)
97              StepPercentage = 1;
98          end
99          disp(sprintf('Percentage: %g%%', StepPercentage*100));
100
101         thetaM = StepPercentage*Theta_M_E + ...
                (1-StepPercentage)*Theta_M_S;
102         phiM = StepPercentage*Phi_M_E + (1-StepPercentage)*Phi_M_S;
103
104         thetaN = StepPercentage*Theta_N_E + ...
                (1-StepPercentage)*Theta_N_S;
105         phiN = StepPercentage*Phi_N_E + (1-StepPercentage)*Phi_N_S;
106
107         Ref = computeRef2DVer1(0, T, 1, [0,0.5,0,0], [Theta_M_S, ...
                thetaM], [phiM, phiM], 3, 0.5, [Theta_N_S, thetaN], ...
                [Phi_N_S, phiN], 3, 0.5, N);
108         X_init = columnVector(Ref(1,:));
109         X_refN = columnVector(Ref(N/2+1,:));
110
111         XRefNDefine
112         Xref = Ref;
113
114     end
```

```
115
116
117     [Z_TEMP, Lamda_TEMP, converged] = NewtonMethod(Z, Lamda, ...
            Xref, 1e-5, safeTurn);
118
119     if(converged==1)
120         saveData;
121         resultSequence = resultSequence + 1;
122         if(~isempty(Z_TEMP) && ~isempty(Lamda_TEMP))
123             Z = Z_TEMP;
124             Lamda = Lamda_TEMP;
125         end
126         Z_LAST_SUCCESS = Z;
127         StepPercentage_TEMP = StepPercentage;
128     else
129         ratioLimit = 1e-8;
130         StepPercentage = StepPercentage_TEMP;
131         coefficient = coefficient * 0.5;
132         BB(2,0.05);
133         if(coefficient<ratioLimit)
134             disp(sprintf('%g, after %d, when ...
                    percetage:%g%%',coefficient, resultSequence, ...
                    StepPercentage*100));
135             BB(100,0.01)
136             return;
137         end
138     end
139 end
140 HDforwardNewTest
141 Deltaforward;
```

BB.m

```
1 %% Beep-Beep
2 function BB(n, t)
3     for i=1:n
4         beep;
5         pause(t);
6     end
7 return;
```

columnVector.m

```
1 function V = columnVector(V)
2     if(size(V,2)>size(V,1))
3         V = V.';
4     end
5 return;
```

computeRef2DVer1.m

```
1 % computeRef2DVer1(0, 10, 1, [0,0,0,0], [0,pi/6], [0,pi/24], 3, ...
      0.5, [0,pi/3], [0,pi/12], 3,0.5, 10)
2 function Ref = computeRef2DVer1(StartTime, TerminalTime, ...
      PeriodNumber, X, theta1, phi1, L1, l1, theta2, phi2, L2, l2, N)
3
4     [vel, dis] = getSineVelocity(StartTime, TerminalTime, ...
          (theta1(2)-theta1(1))/2, PeriodNumber, 0, N);
5     Theta1 = dis;
```

```
    %
    Pos_M = [L1 * cos(Theta1) + X(1), L1 * sin(Theta1) + X(2)];
    Vel_M = [L1 .* vel .* cos(Theta1+pi/2) + X(3), L1 .* vel .* ...
        sin(Theta1+pi/2) + X(4)];


    [vel, dis] = getSineVelocity(StartTime, TerminalTime, ...
        (phi1(2)-phi1(1))/2, PeriodNumber, 0, N);
    Phi1 = dis;
    %
    Pos_M_U = [l1 * cos(pi/2+Theta1+Phi1) + Pos_M(:,1), l1 * ...
        sin(pi/2+Theta1+Phi1) + Pos_M(:,2)];
    Vel_M_U = [l1 .* vel .* cos(Theta1+Phi1+pi) + Vel_M(:,1), l1 ...
        .* vel .* sin(Theta1+Phi1+pi) + Vel_M(:,2)];
    %
    Pos_M_D = [l1 * cos(-pi/2+Theta1+Phi1) + Pos_M(:,1), l1 * ...
        sin(-pi/2+Theta1+Phi1) + Pos_M(:,2)];
    Vel_M_D = [l1 .* vel .* cos(Theta1+Phi1) + Vel_M(:,1), l1 .* ...
        vel .* sin(Theta1+Phi1) + Vel_M(:,2)];

    [vel, dis] = getSineVelocity(StartTime, TerminalTime, ...
        (theta2(2)-theta2(1))/2, PeriodNumber, 0, N);
    Theta2 = dis;
    %
    Pos_N = [L2 * cos(Theta2) + Pos_M(:,1), L2 * sin(Theta2) + ...
        Pos_M(:,2)];
    Vel_N = [L2 .* vel .* cos(Theta2+pi/2) + Vel_M(:,1), L2 .* ...
        vel .* sin(Theta2+pi/2) + Vel_M(:,2)];

    [vel, dis] = getSineVelocity(StartTime, TerminalTime, ...
        (phi2(2)-phi2(1))/2, PeriodNumber, 0, N);
    Phi2 = dis;
    %
    Pos_N_U = [l2 * cos(pi/2+Theta2+Phi2) + Pos_N(:,1), l2 * ...
        sin(pi/2+Theta2+Phi2) + Pos_N(:,2)];
    Vel_N_U = [l2 .* vel .* cos(Theta2+Phi2+pi) + Vel_N(:,1), l2 ...
        .* vel .* sin(Theta2+Phi2+pi) + Vel_N(:,2)];
    %
    Pos_N_D = [l2 * cos(-pi/2+Theta2+Phi2) + Pos_N(:,1), l2 * ...
        sin(-pi/2+Theta2+Phi2) + Pos_N(:,2)];
    Vel_N_D = [l2 .* vel .* cos(Theta2+Phi2) + Vel_N(:,1), l2 .* ...
        vel .* sin(Theta2+Phi2) + Vel_N(:,2)];

    Ref = [Pos_M_U, Vel_M_U, Pos_M_D, Vel_M_D, Pos_N_U, Vel_N_U, ...
        Pos_N_D, Vel_N_D];
%     Anim2D(Ref, 1)
return;
```

Deltaforward.m

```
stopΔ = 0;

Round = 1;
coeStandard = 0.6;
Δ_TEMP = Δ;
while(stopΔ == 0)
    if(stopΔ==0)
        coe = coeStandard;
        converged = 0;
        while(converged==0 || converged==-1)
            if(coe<1e-4)
```

```
                beep;
                disp('Failed at coe');
                return;
            end
            [Z_TEMP, Lamda_TEMP, converged] = NewtonMethod(Z, ...
                Lamda, Xref, 1e-3, 16);
            if(converged==0 || converged==-1)
                Δ = Δ_TEMP;
                coe = coe^.5;
                disp(sprintf('Try coe = %d', coe));
            else
                Z = Z_TEMP;
                Lamda = Lamda_TEMP;
                Δ_TEMP = Δ;
                saveData;
                resultSequence = resultSequence + 1;
            end
            [Δ, stopΔ] = ParameterStepForward(Δ, Δ_max, coe);
        end
    end

    disp(sprintf('Now: Δ = %10.10f',Δ));
end


clear Δ_TEMP %Z_TEMP Lamda_TEMP coeStandard
```

generateEulerZtoLamdaJ.m

```
function generateEulerZtoLamdaJ(Z, Lamda, XRef, dHdZ, Dg, N)
    global currentDirectory globalVariables
    %                ł
    globalVariableNames = [''];
    for i=1:length(globalVariables)
        globalVariableNames = [globalVariableNames, ...
            char(globalVariables(i)), ' '];
    end

    %   d H d Z
    row = length(dHdZ);
    %       c e l l       d H d Z        ł              ł
    dhdz = cell(row,1);

    for i=1:row
        %           d H d Z  ł
        dhdz(i) = cellstr(char(dHdZ(i)));
        %      Z     ł   ł               ł
        for j=length(Z):-1:1
            dhdz(i) = cellstr(strrep(char(dhdz(i)), char(Z(j)), ...
                ['Z(n,' num2str(j) ')']));
        end
        %     L a m d a     ł   ł              ł
        for j=length(Lamda):-1:1
            dhdz(i) = cellstr(strrep(char(dhdz(i)), ...
                char(Lamda(j)), ['Lamda(n+1,' num2str(j) ')']));
        end
        %     X r e f     ł   ł              ł
        for j=length(XRef):-1:1
            dhdz(i) = cellstr(strrep(char(dhdz(i)), ...
                char(XRef(j)), ['Xref(n,' num2str(j) ')']));
        end
```

```
29          %
30          dhdz(i) = cellstr(strrep(char(dhdz(i)), '*', '.*'));
31          dhdz(i) = cellstr(strrep(char(dhdz(i)), '/', './'));
32          dhdz(i) = cellstr(strrep(char(dhdz(i)), '^', '.^'));
33      end
34
35      %   g r a d i e n G
36      row = length(Dg);
37      %        c e l l        D g         ł                ł
38      gradienG = cell(row,1);
39
40      for i=1:row
41          %           d H d Z  ł
42          gradienG(i) = cellstr(char(Dg(i)));
43          %       Z      ł    ł                ł
44          for j=length(Z):-1:1
45              gradienG(i) = cellstr(strrep(char(gradienG(i)), ...
                    char(Z(j)), ['Z(' num2str(N+1) ',' num2str(j) ...
                    ')']));
46          end
47          %
48          gradienG(i) = cellstr(strrep(char(gradienG(i)), '*', '.*'));
49          gradienG(i) = cellstr(strrep(char(gradienG(i)), '/', './'));
50          gradienG(i) = cellstr(strrep(char(gradienG(i)), '^', '.^'));
51      end
52
53      gG = ['['];
54      for i=1:row
55          gG = [gG, char(gradienG(i)), ','];
56      end
57      gG = [gG, ']'];
58
59      file = fopen([currentDirectory, '\EulerZtoLamda.m'],'w+');
60
61      fprintf(file,'%s\r\n','function Lamda = EulerZtoLamda(Z, ...
            Xref)');
62      fprintf(file,'\t%s\r\n',['global ', globalVariableNames]);
63      fprintf(file,'\t%s;\r\n','Lamda = 0*Z');
64      fprintf(file,'\t%s;\r\n',['Lamda(' num2str(N+1) ',:) = ' gG]);
65      fprintf(file,'\t%s;\r\n',['for n=' num2str(N) ':-1:1']);
66      fprintf(file,'\t\t%s\r\n','dHdZ = [');
67      row = length(dhdz);
68      for i=1:row
69  %         fprintf(file,'\t\t\t%s;\r\n',char(dhdz(i)));
70          writeToFile(file, char(dhdz(i)), 100);
71      end
72
73      fprintf(file,'\t\t%s;\r\n',']');
74      fprintf(file,'\t\t%s;\r\n','Lamda(n,:) = Lamda(n+1,:) + Δ...
            T*dHdZ.''');
75      fprintf(file,'\t%s;\r\n','end');
76      fprintf(file,'%s;\r\n','return');
77      fclose(file);
78  return;
```

generateFJ.m

```
1   function generateFJ(Z, Lamda, XRef, dHdZ, dHdLamda, Dg, N)
2       global currentDirectory globalVariables
3
4       globalVariableNames = [''];
```

```
5       for i=1:length(globalVariables)
6           globalVariableNames = [globalVariableNames, ...
                char(globalVariables(i)), ' '];
7       end
8
9       row = length(dHdZ);
10
11      dhdz = cell(row,1);
12
13      for i=1:row
14
15          dhdz(i) = cellstr(char(dHdZ(i)));
16
17          for j=length(Z):-1:1
18              dhdz(i) = cellstr(strrep(char(dhdz(i)), char(Z(j)), ...
                    ['Z(1:' num2str(N) ',' num2str(j) ')']));
19          end
20
21          for j=length(Lamda):-1:1
22              dhdz(i) = cellstr(strrep(char(dhdz(i)), ...
                    char(Lamda(j)), ['Lamda(2:' num2str(N+1) ',' ...
                    num2str(j) ')']));
23          end
24
25          for j=length(XRef):-1:1
26              dhdz(i) = cellstr(strrep(char(dhdz(i)), ...
                    char(XRef(j)), ['Xref(1:' num2str(N) ',' ...
                    num2str(j) ')']));
27          end
28
29          dhdz(i) = cellstr(strrep(char(dhdz(i)), '*', '.*'));
30          dhdz(i) = cellstr(strrep(char(dhdz(i)), '/', './'));
31          dhdz(i) = cellstr(strrep(char(dhdz(i)), '^', '.^'));
32      end
33
34      row = length(dHdLamda);
35      dhdlamda = cell(row,1);
36
37      for i=1:row
38          dhdlamda(i) = cellstr(char(dHdLamda(i)));
39          for j=length(Z):-1:1
40              dhdlamda(i) = cellstr(strrep(char(dhdlamda(i)), ...
                    char(Z(j)), ['Z(1:' num2str(N) ',' num2str(j) ...
                    ')']));
41          end
42          for j=length(Lamda):-1:1
43              dhdlamda(i) = cellstr(strrep(char(dhdlamda(i)), ...
                    char(Lamda(j)), ['Lamda(2:' num2str(N+1) ',' ...
                    num2str(j) ')']));
44          end
45
46          for j=length(XRef):-1:1
47              dhdz(i) = cellstr(strrep(char(dhdz(i)), ...
                    char(XRef(j)), ['Xref(1:' num2str(N) ',' ...
                    num2str(j) ')']));
48          end
49
50          dhdlamda(i) = cellstr(strrep(char(dhdlamda(i)), '*', '.*'));
51          dhdlamda(i) = cellstr(strrep(char(dhdlamda(i)), '/', './'));
52          dhdlamda(i) = cellstr(strrep(char(dhdlamda(i)), '^', '.^'));
53      end
54
```

```
55
56      row = length(Dg);
57
58      gradienG = cell(row,1);
59
60      for i=1:row
61
62          gradienG(i) = cellstr(char(Dg(i)));
63
64          for j=length(Z):-1:1
65              gradienG(i) = cellstr(strrep(char(gradienG(i)), ...
                    char(Z(j)), ['Z(' num2str(N+1) ',' num2str(j) ...
                    ')']));
66          end
67
68          gradienG(i) = cellstr(strrep(char(gradienG(i)), '*', '.*'));
69          gradienG(i) = cellstr(strrep(char(gradienG(i)), '/', './'));
70          gradienG(i) = cellstr(strrep(char(gradienG(i)), '^', '.^'));
71      end
72
73      gG = ['['];
74      for i=1:row
75          gG = [gG, char(gradienG(i)), ','];
76      end
77      gG = [gG, ']'];
78
79
80      file = fopen([currentDirectory, '\getF.m'],'w+');
81
82      fprintf(file,'%s\r\n','function F = getF(Z, Lamda, Xref)');
83      fprintf(file,'\t%s\r\n',['global ', globalVariableNames]);
84      fprintf(file,'\t%s;\r\n',['Lamda(' num2str(N+1) ',:) = ' gG]);
85      fprintf(file,'\t%s\r\n','F = [');
86      row = length(dHdLamda);
87      for i=1:row
88
89          writeToFile(file, ['Z(2:' num2str(N+1) ',' num2str(i) ') ...
                - ...
90              Z(1:' num2str(N) ',' num2str(i) ') - ΔT.*(', ...
                    char(dhdlamda(i)), ')'], 100);
91      end
92
93      row = length(dHdZ);
94      for i=1:row
95          writeToFile(file, ['Lamda(1:' num2str(N) ',' num2str(i) ...
                ') - ...
96              Lamda(2:' num2str(N+1) ',' num2str(i) ') - Δ...
                    T.*(', char(dhdz(i)), ')'], 100);
97      end
98      fprintf(file,'\t%s;\r\n',']');
99      fprintf(file,'%s;\r\n','return');
100     fclose(file);
101 return;
```

generateGLOBALVARIABLE.m

```
1  function generateGLOBALVARIABLE()
2      global currentDirectory globalVariables
3
4      %                ł
5      globalVariableNames = ['currentDirectory '];
```

```
    for i=1:length(globalVariables)
        globalVariableNames = [globalVariableNames, ...
            char(globalVariables(i)), ' '];
    end

    file = fopen([currentDirectory, '\GLOBALVARIABLE.m'],'w+');
    fprintf(file,'%s\r\n',['global ', globalVariableNames]);
return;
```

generateInitialization.m

```
function generateInitialization(Param, ParamVal)
    global currentDirectory
    file = fopen([currentDirectory, '\ParamInit.m'],'w+');
%     fprintf(file,'%s\r\n','function ParamInit()');
    row = size(Param,1);
    for i=1:row
        fprintf(file,'\t%s\r\n',['global ' char(Param(i))]);
        fprintf(file,'\t%s;\r\n',[char(Param(i))...
             ' = ' , num2str(ParamVal(i))]);
    end
    fclose(file);
return;
```

generateJ3.m

```
function generateJ3(Z, Lamda, dHdZ, dHdLamda, Dg, N)

    global currentDirectory globalVariables
    global Len
    Len = N;

    fileName = 'k_block';
    delete([currentDirectory '\J\', fileName, '*.*'])

    for i=1:length(globalVariables)
        eval(['syms ', char(globalVariables(i)), ';']);
    end

    globalVariableNames = [''];
    for i=1:length(globalVariables)
        globalVariableNames = [globalVariableNames, ...
            char(globalVariables(i)), ' '];
    end

    lengthOfZ = length(Z);
    lengthOfLamda = length(Z);
    lengthOfVariable = lengthOfZ + lengthOfLamda;

    buffer = ['['];
    for i=1:lengthOfZ
        eval(['syms xN',num2str(i),';']);
        buffer = [buffer, 'xN',num2str(i),';'];
        Dg = subs(Dg,{sym(['x' num2str(i)])},{sym(['xN' ...
            num2str(i)])});
    end
    buffer = [buffer, ']'];
    eval(['ZN = ', buffer, ';']);
    tableFX = [];
    tableFLamda = [];
```

```
33      for Fi = 1:lengthOfZ
34          tableFX = [tableFX; jacobian((- dHdLamda(Fi)), [Z; Lamda])];
35          tableFLamda = [tableFLamda; jacobian((- dHdZ(Fi)), [Z; ...
                Lamda])];
36      end
37
38      tableFX0 = [];
39      tableFLamda0 = [];
40      for Fi = 1:lengthOfZ
41          tableFX0 = [tableFX0; jacobian((- dHdLamda(Fi)), [Lamda])];
42          tableFLamda0 = [tableFLamda0; jacobian((- dHdZ(Fi)), ...
                [Lamda])];
43      end
44
45      FX_N_1 = [];
46      FLamda_N_1 = [];
47      for Fi = 1:lengthOfZ
48          FX_N_1 = [FX_N_1; ZN(Fi) - Z(Fi) - ΔT*subs(dHdLamda(Fi), ...
                {Lamda}, {Dg})];
49          FLamda_N_1 = [FLamda_N_1; Lamda(Fi) - Dg(Fi) - Δ...
                T*subs(dHdZ(Fi), {Lamda}, {Dg})];
50      end
51
52      tableFX_N_1 = [];
53      tableFLamda_N_1 = [];
54
55      for Fi = 1:lengthOfZ
56          tableFX_N_1 = [tableFX_N_1; jacobian(FX_N_1(Fi), [Z; ZN; ...
                Lamda])];
57          tableFLamda_N_1 = [tableFLamda_N_1; ...
                jacobian(FLamda_N_1(Fi), [Z; ZN; Lamda])];
58      end
59
60
61      fileNamePrefix = [currentDirectory, '\J\', fileName];
62      fileNameCounter = 0;
63      for i=1:lengthOfZ
64          for j = 1:lengthOfZ
65              if(i==j)
66                  MainDiagonal = ['[1; ', symboReplacement(1, Z, ...
                        Lamda, 1, N-2), '; ', ...
67                          symboReplacement(tableFX_N_1(i, ...
                                lengthOfZ+j), Z, Lamda, N-1, N-1), ']'];
68                  subDiagonal = ['[ Δ...
                        T*(',symboReplacement(tableFX(i, j), Z, ...
                        Lamda, 1, N-2),...
69                              ') - ', symboReplacement(1, Z, Lamda, ...
                                1, N-2), ';', ...
                                symboReplacement(tableFX_N_1(i, j), ...
                                Z, Lamda, N-1, N-1), ']'];
70                  str = ['jblock_', num2str(i), '_',  num2str(j), ...
                        ' = sparse(diag(', MainDiagonal,', 0)...
71                          + diag(', subDiagonal, ', -1))'];
72              else
73                  MainDiagonal = ['[', symboReplacement(0, Z, ...
                        Lamda, 0, N-2), '; ', ...
74                          symboReplacement(tableFX_N_1(i, ...
                                lengthOfZ+j), Z, Lamda, N-1, N-1), ']'];
75                  subDiagonal = ['[ Δ...
                        T*(',symboReplacement(tableFX(i, j), Z, ...
                        Lamda, 1, N-2), '); '...
```

```
76                          , symboReplacement(tableFX_N_1(i, j), Z, ...
                                Lamda, N-1, N-1), ']'];
77                  str = ['jblock_', num2str(i), '_',  num2str(j), ...
                        ' = sparse(diag(', MainDiagonal,', 0) ...
78                          + diag(', subDiagonal, ', -1))'];
79              end
80              fileNameCounter = writeToFileSafe(str, ...
                    fileNamePrefix, fileNameCounter, 100, 2000);
81          end
82
83          for j = 1:lengthOfLamda
84              subDiagonal = ['[ ΔT*(', ...
                    symboReplacement(tableFX0(i, j), Z, Lamda, 0, ...
                    0), ');  ΔT*(',...
85                      symboReplacement(tableFX(i, lengthOfZ + j), ...
                            Z, Lamda, 1, N-2), ')]'];
86              str = ['jblock_', num2str(i), '_',  ...
                    num2str(lengthOfZ + j), ' = sparse(diag(', ...
                    subDiagonal, ', +1))'];
87              fileNameCounter = writeToFileSafe(str, ...
                    fileNamePrefix, fileNameCounter, 100, 2000);
88          end
89      end
90
91      for i = 1:lengthOfLamda
92          for j = 1:lengthOfZ
93
94              MainDiagonal = ['[', symboReplacement(0, Z, Lamda, ...
                    0, N-2), '; ', ...
95                      symboReplacement(tableFLamda_N_1(i, ...
                            lengthOfZ+j), Z, Lamda, N-1, N-1), ']'];
96              subDiagonal = ['[ ΔT*(', ...
                    symboReplacement(tableFLamda(i, j), Z, ...
97                      Lamda, 1, N-2), '); ', ...
                            symboReplacement(tableFLamda_N_1(i, j), ...
                            Z, Lamda, N-1, N-1),']'];
98              str = ['jblock_', num2str(lengthOfZ + i), '_',  ...
                    num2str(j), ' = sparse(diag(', MainDiagonal,', ...
                    0) + diag(', subDiagonal, ', -1))'];
99              fileNameCounter = writeToFileSafe(str, ...
                    fileNamePrefix, fileNameCounter, 100, 2000);
100         end
101
102         for j = 1:lengthOfLamda
103             if(i==j)
104                 MainDiagonal = ['[1; ', symboReplacement(1, Z, ...
                        Lamda, 1, N-2), '; ', ...
105                         symboReplacement(tableFLamda_N_1(i, ...
                                lengthOfZ*2 + j), Z, Lamda, N-2, ...
                                N-2), ...
                                ']'];%≤=   ł  L a m d a _ n
106                 subDiagonal = ['[ ΔT*(', ...
                        symboReplacement(tableFLamda0(i, j), Z, ...
                        Lamda, 0, 0), ')...
107                          - ',symboReplacement(1, Z, Lamda, 0, ...
                                0), '; ΔT*(', ...
                                symboReplacement(tableFLamda(i, ...
                                lengthOfZ + j), Z, Lamda, 1, N-2), ...
                                ') - ', symboReplacement(1, Z, ...
                                Lamda, 1, N-2), ']'];
108                 str = ['jblock_', num2str(lengthOfZ + i), '_',  ...
                        num2str(lengthOfZ + j), ' = sparse(diag(', ...
```

```
                    MainDiagonal,', 0) + diag(', subDiagonal, ', ...
                    +1))'];
109                 fileNameCounter = writeToFileSafe(str, ...
                        fileNamePrefix, fileNameCounter, 100, 2000);
110             else
111                 subDiagonal = ['[ ΔT*(', ...
                        symboReplacement(tableFLamda0(i, j), Z, ...
                        Lamda, 0, 0), '); ΔT*(',...
112                         symboReplacement(tableFLamda(i, ...
                                lengthOfZ + j), Z, Lamda, 1, N-2), ...
                                ')]'];
113                 str = ['jblock_', num2str(lengthOfZ + i), '_',  ...
                        num2str(lengthOfZ + j), ' = sparse(diag(', ...
                        subDiagonal, ', +1))'];
114                 fileNameCounter = writeToFileSafe(str, ...
                        fileNamePrefix, fileNameCounter, 100, 2000);
115             end
116         end
117     end
118
119     file = fopen([currentDirectory, '\getJTri.m'],'w+');
120
121     fprintf(file,'%s\r\n','function J = getJTri(Z, Lamda)');
122     fprintf(file,'\t%s\r\n',['global ', globalVariableNames]);
123     for i=0:fileNameCounter
124         fprintf(file,'\t%s\r\n',[fileName, '_', num2str(i)]);
125     end
126     fprintf(file,'\t%s\r\n','J = [');
127     for i=1:lengthOfVariable
128         buffer = '';
129         for j = 1:lengthOfVariable
130             buffer = [buffer, ' jblock_', num2str(i), '_',  ...
                    num2str(j), ', '];
131         end
132         writeToFile(file, buffer, 100);
133     end
134     fprintf(file,'\t%s\r\n','];');
135     fprintf(file,'%s;\r\n','return');
136     fclose(file);
137     clear Len
138 return;
139
140 function expression = symboReplacement(expression, Z, Lamda, ...
        start, terminal)
141     global Len
142     cla = class(expression);
143     try
144         expression = double(expression);
145         expression = ['ones(', num2str(terminal - start + 1), ', ...
                1)*', num2str(expression)];
146     catch e
147         if(start == terminal)
148             Z_str = [num2str(terminal+1)];
149             Lamda_str = [num2str(terminal+2)];
150         else
151             Z_str = [num2str(start+1), ':', num2str(terminal+1)];
152             Lamda_str = [num2str(start+2), ':', ...
                    num2str(terminal+2)];
153         end
154         if(strcmp(cla,'sym'))
155             for j=length(Z):-1:1
```

```
156                 expression = strrep(char(expression), ...
                        char(Z(j)), ['Z(', Z_str, ',' num2str(j) ')']);
157             end
158             for j=length(Lamda):-1:1
159                 expression = strrep(char(expression), ...
                        char(Lamda(j)), ['Lamda(', Lamda_str, ',' ...
                        num2str(j) ')']);
160             end
161         elseif(ischar(expression))
162             for j=length(Z):-1:1
163                 expression = strrep(expression, char(Z(j)), ...
                        ['Z(', Z_str ',' num2str(j) ')']);
164             end
165             for j=length(Lamda):-1:1
166                 expression = strrep(expression, char(Lamda(j)), ...
                        ['Lamda(', Lamda_str, ',' num2str(j) ')']);
167             end
168         elseif(strcmp(cla,'double'))
169             expression = ['ones(', num2str(terminal - start + ...
                    1), ', 1).*', num2str(expression)];
170         end
171     end
172     for i=length(Z):-1:1
173         expression = strrep(expression, ['xN', num2str(i)], ...
                ['Z(', ...
174         num2str(Len+1), ',' num2str(i) ')']);
175     end
176     expression = strrep(expression, '*', '.*');
177     expression = strrep(expression, '/', './');
178     expression = strrep(expression, '^', '.^');
179 return;
```

generateLamdaNPlus1FromZNPlus1J.m

```
1  function generateLamdaNPlus1FromZNPlus1J(Z, Dg, N)
2      global currentDirectory globalVariables
3
4      %               ł
5      globalVariableNames = [''];
6      for i=1:length(globalVariables)
7          globalVariableNames = [globalVariableNames, ...
               char(globalVariables(i)), ' '];
8      end
9
10     %   g r a d i e n G
11     row = length(Dg);
12     %        c e l l        D g         ł                ł
13     gradienG = cell(row,1);
14
15     for i=1:row
16         %           d H d Z  ł
17         gradienG(i) = cellstr(char(Dg(i)));
18         %       Z      ł    ł                 ł
19         for j=length(Z):-1:1
20             gradienG(i) = cellstr(strrep(char(gradienG(i)), ...
                   char(Z(j)), ['Z(' num2str(N+1) ',' num2str(j) ...
                   ')']));
21         end
22         %
23         gradienG(i) = cellstr(strrep(char(gradienG(i)), '*', '.*'));
24         gradienG(i) = cellstr(strrep(char(gradienG(i)), '/', './'));
```

```
25          gradienG(i) = cellstr(strrep(char(gradienG(i)), '^', '.^'));
26      end
27
28      gG = ['['];
29      for i=1:row
30          gG = [gG, char(gradienG(i)), ','];
31      end
32      gG = [gG, ']'];
33
34      file = fopen([currentDirectory, ...
            '\getLamdaNPlus1FromZNPlus1.m'],'w+');
35
36      fprintf(file,'%s\r\n','function LamdaNPlus1 = ...
            getLamdaNPlus1FromZNPlus1(Z)');
37      fprintf(file,'\t%s\r\n',['global ', globalVariableNames]);
38      fprintf(file,'\t%s;\r\n',['LamdaNPlus1 = ' gG]);
39      fprintf(file,'%s;\r\n','return');
40      fclose(file);
41  return;
```

generatePARAMETER_VECTORS.m

```
1   function [PARAMETER_NAME, PARAMETER_VALUE, PARAMETER_MAX] = ...
        generatePARAMETER_VECTORS(SPRINGS)
2       global currentDirectory
3       SpringsNr = size(SPRINGS,1);
4       PARAMETER_NAME = [];
5       PARAMETER_VALUE = [];
6       PARAMETER_MAX = [];
7       for i=1:SpringsNr
8           PARAMETER_NAME = [PARAMETER_NAME, ...
9               'java.lang.String(''H',num2str(i),''''),...
10           ', 'java.lang.String(''D',num2str(i),''''), '];
11          PARAMETER_VALUE = [PARAMETER_VALUE, ...
                num2str(SPRINGS(i,1)),...
12           ', ', num2str(SPRINGS(i,2)), ', '];
13          PARAMETER_MAX = [PARAMETER_MAX, num2str(SPRINGS(i,3)),...
14           ', ', num2str(SPRINGS(i,4)), ', '];
15      end
16
17      file = fopen([currentDirectory, '\PARAMETER_VECTORS.m'],'w+');
18      fprintf(file,'%s\r\n',['PARAMETER_NAME = [', PARAMETER_NAME, ...
            '];']);
19      fprintf(file,'%s\r\n',['PARAMETER_VALUE = [', ...
            PARAMETER_VALUE, '];']);
20      fprintf(file,'%s\r\n',['PARAMETER_MAX = [', PARAMETER_MAX, ...
            '];']);
21      fclose(file);
22  return;
```

generateXRefDefine.m

```
1   function generateXRefDefine(X)
2       global currentDirectory
3       file = fopen([currentDirectory, '\XRefNDefine.m'],'w+')
4
5       for i=1:length(X)
6           fprintf(file,'%s\r\n',['XrefN', num2str(i),...
7            ' = X_refN(', num2str(i), ');']);
8       end
```

```
    fclose(file);
return;
```

getSineVelocity.m

```
function [velocity, displacement] = getSineVelocity(StartTime, ...
    TerminalTime, AmplitudeOfDisplacement, PeriodNumber, ...
    startPosition, N)
    ΔT = (TerminalTime-StartTime)/N;
    f = 2*pi*PeriodNumber/(TerminalTime-StartTime);
    Amplitude = AmplitudeOfDisplacement * f;
    velocity = Amplitude*sin(f*[StartTime:ΔT:TerminalTime].');
    C = startPosition + Amplitude/f*cos(f*StartTime);
    displacement = ...
        -Amplitude/f*cos(f*[StartTime:ΔT:TerminalTime].') + C;
return;
```

HDforwardNewTest.m

```

coefficientStandard = 3e-1;

for index = 1:length(PARAMETER_NAME)
    eval([char(PARAMETER_NAME(index)), '=', ...
        num2str(PARAMETER_VALUE(index)), ';']);
end


PARAMETER_STOP = 0*PARAMETER_VALUE;

while(prod(PARAMETER_STOP)==0)
    coe = coefficientStandard;
    TEXT_OUTPUT = '';
    for i=1:length(PARAMETER_NAME)
        TEXT_OUTPUT = [TEXT_OUTPUT, char(PARAMETER_NAME(i)),'='...
                , num2str(PARAMETER_VALUE(i)),'; '];
    end
    disp(['From: ',TEXT_OUTPUT]);
    for i = 1:length(PARAMETER_NAME)
        eval([char(PARAMETER_NAME(i)), '_TEMP = ', ...
            num2str(PARAMETER_VALUE(i)), ';']);
        [PARAMETER_VALUE(i), PARAMETER_STOP(i)] = ...
            ParameterStepForward(PARAMETER_VALUE(i), ...
            PARAMETER_MAX(i), 1+coe);
        disp(['Try: ', [char(PARAMETER_NAME(i)), '=', ...
            num2str(PARAMETER_VALUE(i))]]);
        eval([char(PARAMETER_NAME(i)), '=', ...
            num2str(PARAMETER_VALUE(i)), ';']);
    end
        coe = coefficientStandard;
        converged = 0;
        while(converged==0 || converged==-1)
            if(coe<1e-8)
                beep;
                disp('Forward coefficient coe is too small !');
                return;
            end
            [Z_TEMP, Lamda_TEMP, converged] = NewtonMethod(Z, ...
                Lamda, Xref, 1e-3, safeTurn);
            if(converged==0 || converged==-1)
```

```
35                  eval([char(PARAMETER_NAME(i)), ' = ', ...
                        char(PARAMETER_NAME(i)), '_TEMP;']);
36                  coe = coe*0.50;
37                  disp(sprintf('Now trying coe = %d', coe));
38              else
39                  Z = Z_TEMP;
40                  Lamda = Lamda_TEMP;
41                  eval([char(PARAMETER_NAME(i)), '_TEMP = ', ...
                        char(PARAMETER_NAME(i)), ';']);
42                  saveData;
43                  resultSequence = resultSequence + 1;
44              end
45          end
46
47      disp('============= Done ================')
48  end
49  clear TEXT_OUTPUT coefficientStandard
```

newGuess.m

```
1   function Z = newGuess(S, T, V0, T0, T1, N)
2
3   columnVector(S);
4   columnVector(T);
5   columnVector(V0);
6
7   P = T - S;
8
9   A = (-(T0^3)/3+T0^2*(T0+T1)/2-T0^2*T1);
10  B = (-(T1^3)/3+T1^2*(T0+T1)/2-T0*T1^2);
11
12  Result = [A,1;B,1]\[0;M];
13  a = Result(1);
14  C = Result(2);
15  b = a*(T0+T1);
16  c = -a*T0*T1;
17
18  ΔT = (T1-T0)/N;
19  t = [T0:ΔT:T1].';
20
21  V = -a*t.^2 + b*t + c;
22  X = -a*(t.^3)/3 + b*(t.^2)/2 + c*t + C;
23
24  dimension = length(S);
25  Q = zeros(dimension, 1);
26  Z = zeros(N+1, dimension*2);
27  Pu = P/M;
28  for i=1:dimension
29      Qt = Q;
30      Qt(i) = 1;
31      Z(:,i) = X*Pu.'*Qt+S(i);
32      Z(:,i+dimension) = V*Pu.'*Qt+V0(i);
33  end
34  return;
```

NewtonMethod.m

```
1   function [Z, Lamda, converged] = NewtonMethod(Z, Lamda, Xref, ...
        tolerant, safeNumber)
2   global J
```

```
3       counter = 1;
4       Norm = 1;
5       converged = 1;
6       while(Norm>tolerant)
7           if(counter>safeNumber)
8
9               if(Norm>1e3);
10                  converged = 0;
11              else
12                  converged = -1;
13              end
14              return;
15          end
16          tic
17          F = getF(Z, Lamda, Xref);
18          J = getJTri(Z, Lamda);
19          Y = compositeY(Z, Lamda);
20          toc
21          tempY = Y;
22          try
23              Y = Y - J\F;
24          catch ME
25              converged = 0;
26              return;
27          end
28          Norm = norm(Y-tempY);
29          disp(sprintf('Norm = %f', Norm))
30          if(isnan(Norm) || Norm>1e9 || condest(J)>1e30)
31              converged = 0;
32              return;
33          end
34          [Z, Lamda] = decompositeY(Y, Z, Lamda);
35          counter = counter+1;
36      end
37
38  return;
39
40  function Y = compositeY(Z, Lamda)
41      Y = [];
42      [row, column] = size(Z);
43      N = row - 1;
44      for i=1:column
45          Y = [Y; Z(2:N+1,i)];
46      end
47      [row, column] = size(Lamda);
48      N = row - 1;
49      for i=1:column
50          Y = [Y; Lamda(1:N,i)];
51      end
52  return;
53
54  function [Z, Lamda] = decompositeY(Y, OldZ, OldLamda)
55  %     Length = length(Y);
56      [rowZ, columnZ] = size(OldZ);
57      N = rowZ - 1;
58      for i=1:columnZ
59          OldZ(2:rowZ,i) = Y([1:N]+N*(i-1));
60      end
61      Z = OldZ;
62
63      [rowLamda, columnLamda] = size(OldLamda);
64      N = rowLamda - 1;
```

```
65      for i=1:columnLamda
66          OldLamda(1:N,i) = Y([1:N]+N*(i-1)+columnZ*N);
67      end
68      Lamda = OldLamda;
69      Lamda(N+1,:) = getLamdaNPlus1FromZNPlus1(Z);
70  return;
```

ParameterStepForward.m

```
1   function [param, stop] = ParameterStepForward(param, paramMax, coe)
2   %     coe = 1.3;
3       stop = 0;
4       if(param == paramMax)
5           stop=1;
6       elseif(abs(paramMax)>1 && abs(param)≤abs(paramMax))
7           if (param==0 && paramMax≠0)
8               param =sign(paramMax)*eps;
9           elseif(paramMax==0)
10              param=0;
11              stop = 1;
12          elseif(abs(param)^coe≥abs(paramMax))
13              param = paramMax;
14              stop = 1;
15          else
16              if(abs(param)<0.8)
17                  param = sign(paramMax)*abs(param)^.3;
18              elseif(abs(param)>0.8 && abs(param)<1)
19                  param = sign(paramMax);
20              elseif(abs(param)==1 && abs(param)≤abs(paramMax))
21                  param = sign(paramMax)*abs(param)*(coe);
22              else
23                  param = sign(paramMax)*abs(param)*coe;
24              end
25          end
26      elseif(abs(paramMax)≤1 && abs(param)≤abs(paramMax))
27          if (param==0 && paramMax≠0)
28              param =sign(paramMax)*eps;
29          elseif(paramMax==0)
30              param=0;
31              stop = 1;
32          elseif(abs(param)*coe≥abs(paramMax))
33              param = paramMax;
34              stop = 1;
35          else
36  %             if(abs(param)<0.95)
37                    param = sign(paramMax)*abs(param)*coe;
38  %             elseif(abs(param)>0.95 && abs(param)<1)
39  %                 param = sign(paramMax);
40  %             end
41          end
42      else
43          if (param==0 && paramMax≠0)
44              param =sign(paramMax)*eps;
45          elseif(paramMax==0)
46              param=0;
47              stop = 1;
48          elseif(abs(param)*coe≤abs(paramMax))
49              param = paramMax;
50              stop = 1;
51          else
52              param = sign(paramMax)*abs(param)*coe;
```

```
        end
    end
return;
```

readMatrixData.m

```
function matrix = readMatrixData(filePath)

file = fopen(filePath,'r');
matrix=[];
while 1
    stringLine=fgetl(file);
    if ¬ischar(stringLine)
        break;
    end
    matrix = [
            matrix;
            str2num(stringLine)
    ];
end
fclose(file);
```

saveData.m

```
global currentDirectory

saveMatrixData([PARAMETER_VALUE, Δ], [currentDirectory ...
    '\DATA\param' num2str(resultSequence) '.txt'], 'w+');
saveMatrixData(Z, [currentDirectory '\DATA\Z'...
     num2str(resultSequence) '.txt'], 'w+');
saveMatrixData(Xref, [currentDirectory '\DATA\Xref'...
     num2str(resultSequence) '.txt'], 'w+');
saveMatrixData(Lamda, [currentDirectory '\DATA\Lamda'...
     num2str(resultSequence) '.txt'],'w+');
```

saveMatrixData.m

```
function saveMatrixData(matrix, filePath, permission)
    N = size(matrix,1);
    file = fopen(filePath,permission);
    for i=1:N
        string = num2str(matrix(i,:),'%10.16f, ');
        fprintf(file,'%s\r\n',string);
    end
    fclose(file);
return;
```

writeToFile.m

```
function totalLines = writeToFile(file, str, TextsPerRow)
    totalLines = 1;
    if(isempty(TextsPerRow))
        TextsPerRow = 100;
    end

    Length = length(str);
    counter = 1;
    pos = 76;
```

```
10      lastpos = 1;
11      finish = 0;
12      if(Length>TextsPerRow)
13          PosionPlus = strfind(str,'+');
14          if(isempty(PosionPlus))
15              PosionPlus = strfind(str,',');
16          end
17          while(finish==0)
18              while(PosionPlus(counter)-pos<0)
19                  counter = counter+1;
20                  if(counter>length(PosionPlus))
21                      finish = 1;
22                      break;
23                  end
24              end
25              if(finish==1)
26                  fprintf(file,'\t\t%s;\r\n',str(lastpos:Length));
27              else
28                  fprintf(file,'\t\t%s  ...
                        ...\r\n',str(lastpos:PosionPlus(counter)));
29                  lastpos = PosionPlus(counter)+1;
30                  pos = 75 + PosionPlus(counter);
31              end
32              totalLines = totalLines + 1;
33          end
34
35      else
36          fprintf(file,'\t\t%s;\r\n',str);
37      end
38  return;
```

writeToFileSafe.m

```
1   function [fileNameCounter] = writeToFileSafe(str, ...
        fileNamePrefix, fileNameCounter, columnLimit, rowLimit)
2   % ...
          f t e l l                 ł             1 0 0
3   %
4       file = fopen([fileNamePrefix, '-', num2str(fileNameCounter), ...
            '.m'], 'a+');
5       fileCounter = ftell(file);
6       if(isempty(columnLimit))
7           columnLimit = 100;
8       end
9       if(isempty(rowLimit))
10          rowLimit = 1000;
11      end
12      writeToFile(file, str, columnLimit);
13      if(fileCounter ≥ rowLimit*columnLimit)
14          fileNameCounter = fileNameCounter + 1;
15      end
16      fclose(file);
17  return;
```

References

[1] S. Cannarsa, *Semiconcave Functions, Hamilton-Jacobi Equations, and Optimal Control*, Birkhaeuser, Boston, 2004.

[2] L. C.Evans, *Partial Differential Equations*, vol. 19 of Graduate Studies in Mathematics, American Mathematical Society, Rhode Island, 1998.

[3] U. Muico, Y. Lee, J. Popović, and Z. Popović, *Contact-aware nonlinear control of dynamic characters*, ACM Transactions on Graphics, 28 (2009).

[4] M. Sandberg, *Extened applicability of the symplectic pontryagin method*. http://arxiv.org/PS_cache/arxiv/pdf/0901/.

http://grail.cs.washington.edu/projects/nqr/s2009/