# LCP Notes

Samuel Pfrommer

June 13, 2019

## 1 Falling block

Consider a block of unit mass falling from an arbitrary height onto a static surface under the influence of a gravitational force with constant $-g$. We then have the following equations of motion:

$$x_{k+1} = x_k + \dot{x}_{k+1}\Delta t$$
$$\dot{x}_{k+1} = \dot{x}_k + (-g + \lambda_{k+1})\Delta t$$

Where $x_n$ represents the separation of the block from the surface at time step $n$, $\lambda_n$ is the nonpenetration contact force, and $\Delta t$ is our discrete time step. Note we are using implicit euler to prevent penetration of the block with the surface. Substituting the second equation into the first we have:

$$x_{k+1} = x_k + (\dot{x}_k + (-g + \lambda_{k+1})\Delta t)\Delta t$$
$$x_{k+1} = x_k + \dot{x}_k\Delta t + -g\Delta t^2 + \lambda_{k+1}\Delta t^2$$
$$\lambda_{k+1} = \frac{1}{\Delta t^2}x_{k+1} - \frac{1}{\Delta t^2}x_k - \frac{1}{\Delta t}\dot{x}_k + g$$
$$\lambda_{k+1} = \frac{1}{\Delta t^2}x_{k+1} - \frac{1}{\Delta t^2}(x_k + \Delta t\dot{x}_k - g\Delta t^2)$$

Observe that we solve an LCP for each time step, and as such $x_k$ and $\dot{x}_k$ are given to us from the previous time step. The problem therefore consists of solving for $x_{k+1}$, $\dot{x}_{k+1}$, and $\lambda_{k+1}$. Since $\lambda_{k+1}$ determines $\dot{x}_{k+1}$ we can ignore $\dot{x}_{k+1}$ and instead consider the following LCP:

$$\lambda_{k+1} \geq 0 \quad \text{only permit separating forces}$$
$$x_{k+1} \geq 0 \quad \text{no penetration}$$
$$\lambda_{k+1}x_{k+1} = 0 \quad \text{must have contact to have force}$$
$$\lambda_{k+1} = Mx_{k+1} + q$$

With $M = \frac{1}{\Delta t^2}$ and $q = -\frac{1}{\Delta t^2}(x_k + \Delta t\dot{x}_k - g\Delta t^2)$ from the equation above.

## 2 Sliding block

### 2.1 Forwards case

Consider a block of unit mass sliding in 1D on a surface under a positive external force $u_n$ with static/dynamic coefficient of friction $\mu$ and friction force $\lambda_n$. We have the following EOM:

$$x_{k+1} = x_k + \dot{x}_{k+1}\Delta t$$
$$\dot{x}_{k+1} = \dot{x}_k + (u_{k+1} - \lambda_{k+1})\Delta t$$

Observe that now the relevant constraint is that the velocity $\dot{x}_{k+1}$ must always be positive. Furthermore, to encode stick/slide transitions, we want:

$$0 \leq \mu g - \lambda_{k+1} \perp \dot{x}_{k+1} \geq 0$$

Such that when the block is moving, $\lambda_{k+1} = \mu g$ (i.e. the maximum friction force is being used to oppose the motion), and when the block is stationary $\lambda_{k+1} < \mu g$ (i.e. some smaller static friction force is being used). To encode this constraint as an LCP, we introduce the new variable $\lambda'_{k+1} = \mu g - \lambda_{k+1}$, where $\lambda'_{k+1}$ intuitively represents the "unused" friction force. Furthermore, since we are interested in bounding velocity we will instead just use the second EOM:

$$\dot{x}_{k+1} = \dot{x}_k + (u_{k+1} - \lambda_{k+1})\Delta t \tag{1}$$
$$\dot{x}_{k+1} = \dot{x}_k + (u_{k+1} + \lambda'_{k+1} - \mu g)\Delta t$$
$$\dot{x}_{k+1} = \Delta t \cdot \lambda'_{k+1} + \dot{x}_k + \Delta t(u_{k+1} - \mu g)$$
$$\lambda'_{k+1} = \frac{1}{\Delta t}\dot{x}_{k+1} - \frac{1}{\Delta t}(\dot{x}_k + \Delta t(u_{k+1} - \mu g))$$
$$\lambda'_{k+1} = \frac{1}{\Delta t}\dot{x}_{k+1} - \frac{1}{\Delta t}\dot{x}_k - u_{k+1} + \mu g$$

We can now write the LCP:

$$\lambda'_{k+1} \geq 0 \quad \text{friction force can't exceed } \mu \text{ times normal force}$$
$$\dot{x}_{k+1} \geq 0 \quad \text{block is sliding to the right}$$
$$\lambda'_{k+1}\dot{x}_{k+1} = 0 \quad \text{either block is stationary or using max friction force}$$
$$\lambda'_{k+1} = M\dot{x}_{k+1} + q$$

With $M = \frac{1}{\Delta t}$ and $q = -\frac{1}{\Delta t}\dot{x}_k - u_{k+1} + \mu g$ from the equation above.

**Remark 1.** *It might seem that the above constraints are insufficient; what would prevent a negative friction force in sticking? Consider the case where $\lambda'_{k+1} > \mu g$; i.e., $\lambda_{k+1} < 0$. If $\lambda_{k+1} < 0$, we can immediately see that the right hand side of (1) must be strictly greater than zero because of our nonnegativity constraints on $\dot{x}_k$ and $u_{k+1}$. That means that $\dot{x}_{k+1} > 0$ and therefore $\lambda'_{k+1} = 0$. This proves that $\lambda_{k+1}$ is always positive even though we never encoded that explicitly as a constraint. Intuitively, one can think of this as "our friction force can never propel the block, since if that happened the velocity would be immediately positive and friction must always oppose motion."*

## 2.2 Bidirectional case (direct formulation)

For the bidirectional case we need to introduce more slack variables: $\dot{x}^+_{k+1}$, $\dot{x}^-_{k+1}$, $\lambda^+_{k+1}$, and $\lambda^-_{k+1}$, which are all positive numbers representing velocity / friction force in the + / - direction. We can then write:

$$0 \leq \mu g - \lambda^-_{k+1} \perp \dot{x}^+_{k+1} \geq 0$$
$$0 \leq \mu g - \lambda^+_{k+1} \perp \dot{x}^-_{k+1} \geq 0$$

We again perform the substitution $\lambda'^+_{k+1} = \mu g - \lambda^+_{k+1}$ and $\lambda'^-_{k+1} = \mu g - \lambda^-_{k+1}$. We can now write the EOM for both $\dot{x}^+_{k+1}$ and $\dot{x}^-_{k+1}$. The first equation looks familiar to the forwards case, with the addition of a term involving $\dot{x}^-_k$. The importance of this term comes when the block changes direction, in this case from a negative velocity to a positive velocity. In order for the positive velocity component to begin increasing, we need to ensure that the negative velocity from the previous timestep has been driven completely to zero. This does not correspond to an actual physical force; it is more of a force threshold that the input $u_{k+1}$ must overcome in order to begin increasing the block's positive velocity.

$$\dot{x}^+_{k+1} = \dot{x}^+_k + (u_{k+1} - \frac{1}{\Delta t}\dot{x}^-_k - \lambda^-_{k+1})\Delta t$$
$$\dot{x}^+_{k+1} = \dot{x}^+_k + (u_{k+1} - \frac{1}{\Delta t}\dot{x}^-_k + \lambda'^-_{k+1} - \mu g)\Delta t$$
$$\lambda'^-_{k+1} = \frac{1}{\Delta t}\dot{x}^+_{k+1} - \frac{1}{\Delta t}\dot{x}^+_k - u_{k+1} + \frac{1}{\Delta t}\dot{x}^-_k + \mu g$$
$$\lambda'^-_{k+1} = \frac{1}{\Delta t}\dot{x}^+_{k+1} + \frac{1}{\Delta t}\dot{x}^-_k - \frac{1}{\Delta t}\dot{x}^+_k - u_{k+1} + \mu g$$

Similarly, we can get:

$$\dot{x}_{k+1}^- = \dot{x}_k^- + (-u_{k+1} - \frac{1}{\Delta t}\dot{x}_k^+ - \lambda_{k+1}^+)\Delta t$$

$$\dot{x}_{k+1}^- = \dot{x}_k^- + (-u_{k+1} - \frac{1}{\Delta t}\dot{x}_k^+ + \lambda_{k+1}'^+ - \mu g)\Delta t$$

$$\lambda_{k+1}'^+ = \frac{1}{\Delta t}\dot{x}_{k+1}^- - \frac{1}{\Delta t}\dot{x}_k^- + u_{k+1} + \frac{1}{\Delta t}\dot{x}_k^+ + \mu g$$

$$\lambda_{k+1}'^+ = \frac{1}{\Delta t}\dot{x}_{k+1}^- + \frac{1}{\Delta t}\dot{x}_k^+ - \frac{1}{\Delta t}\dot{x}_k^- + u_{k+1} + \mu g$$

We can now put these in matrix form as:

$$\begin{bmatrix} \lambda_{k+1}'^- \\ \lambda_{k+1}'^+ \end{bmatrix} = \begin{bmatrix} \frac{1}{\Delta t} & 0 \\ 0 & \frac{1}{\Delta t} \end{bmatrix} \begin{bmatrix} \dot{x}_{k+1}^+ \\ \dot{x}_{k+1}^- \end{bmatrix} + \begin{bmatrix} \frac{1}{\Delta t}\dot{x}_k^- - \frac{1}{\Delta t}\dot{x}_k^+ - u_{k+1} + \mu g \\ \frac{1}{\Delta t}\dot{x}_k^+ - \frac{1}{\Delta t}\dot{x}_k^- + u_{k+1} + \mu g \end{bmatrix}$$

This equation gives us the form of our $M$ matrix and $q$ vector. We also have the below constraints:

$$\lambda_{k+1}'^+ \geq 0$$
$$\lambda_{k+1}'^- \geq 0$$
$$\dot{x}_{k+1}^+ \geq 0$$
$$\dot{x}_{k+1}^- \geq 0$$
$$\lambda_{k+1}'^- \dot{x}_{k+1}^+ = 0 \quad \text{block sliding to the right means all neg friction force used}$$
$$\lambda_{k+1}'^+ \dot{x}_{k+1}^- = 0 \quad \text{block sliding to the left means all pos friction force used}$$

Finally, we can just update the positions by:

$$x_{k+1} = x_k + (\dot{x}_{k+1}^+ - \dot{x}_{k+1}^-)\Delta t$$

## 2.3 Sanity checks

### 2.3.1 Velocity characteristics

**Lemma 2.1.** *The positive and negative velocities at any time step are complementary; i.e., $\dot{x}_k^+ \cdot \dot{x}_k^- = 0$ for all time steps $k$.*

*Proof.* Since $\dot{x}_k^+$ and $\dot{x}_k^-$ are both optimization variables they are not immediately constrained to be complimentary; however, this ends up being the case. Empirically, this constraint was never violated for about 30,000 time steps with random forces using the above formulation. It can also formally be proven via induction (assuming $\Delta t = 1$ for notational cleanliness):

Assume $\dot{x}_k^+ \perp \dot{x}_k^-$; we want to show that $\dot{x}_{k+1}^+ \perp \dot{x}_{k+1}^-$. Observe that the base case is trivial (assign all of $\dot{x}_0$ to the appropriate direction and set the other direction to zero). Now assume WLOG that from our induction hypothesis we have $\dot{x}_k^+ = 0$. Also assume for sake of contradiction that both $\dot{x}_{k+1}^+$ and $\dot{x}_{k+1}^-$ are strictly positive. We can then write:

$$\begin{aligned} \dot{x}_{k+1}^+ &= \dot{x}_k^+ + u_{k+1} - \dot{x}_k^- - \lambda_{k+1}^- \\ &= u_{k+1} - \dot{x}_k^- - (\mu g - \lambda_{k+1}'^-) \\ &= u_{k+1} - \dot{x}_k^- - \mu g \quad \text{by complimentarity} \\ &\leq u_{k+1} - \dot{x}_k^- \\ \dot{x}_{k+1}^- &= \dot{x}_k^- - u_{k+1} - \dot{x}_k^+ - \lambda_{k+1}^+ \\ &= -u_{k+1} + \dot{x}_k^- - (\mu g - \lambda_{k+1}'^+) \\ &= -u_{k+1} + \dot{x}_k^- - \mu g \quad \text{by complimentarity} \\ &\leq -(u_{k+1} - \dot{x}_k^-) \end{aligned}$$

It is now easy to see that one of $\dot{x}_{k+1}^+$ or $\dot{x}_{k+1}^-$ must be nonpositive–which is a contradiction because we assumed them to be strictly positive. Therefore we have proven the induction step. ∎

**Remark 2.** *It seems that if $u_{k+1} - \dot{x}_k^- \neq 0$ the above two inequalities would force one of the velocities to be strictly negative, violating the nonnegativity constraints. However, note that this only occurs because of our contradictory assumption that both $\dot{x}_{k+1}^+$ and $\dot{x}_{k+1}^-$ are strictly positive. In reality, one would need to be zero, which would free up the complementary $\lambda'$ to be arbitrarily positive.*

### 2.3.2 Summed velocity

Let's double check what happens when we add our velocities to get the overall velocity $\dot{x}_{k+1}$.

$$\dot{x}_{k+1}^+ = \dot{x}_k^+ + (u_{k+1} - \frac{1}{\Delta t}\dot{x}_k^- - \lambda_{k+1}^-)\Delta t$$

$$-\left(\dot{x}_{k+1}^- = \dot{x}_k^- + (-u_{k+1} - \frac{1}{\Delta t}\dot{x}_k^+ - \lambda_{k+1}^+)\Delta t\right)$$

$$\dot{x}_{k+1}^+ - \dot{x}_{k+1}^- = 2(\dot{x}_k^+ - \dot{x}_k^-) + 2u_{k+1}\Delta t + (\lambda_{k+1}^+ - \lambda_{k+1}^-)\Delta t \tag{2}$$

At first glance, the factors of 2 don't look quite right (why is velocity doubling every timestep?). However, this is actually not an issue.

The reason why is pretty unintuitive and has to do with the fact that our friction forces are being encoded as the slack variables $w = Mz + q$. The best way to see this is with a simple example. Assume that the block is moving with a constant rightwards velocity $\dot{x}_k^+ = \dot{x}_{k+1}^+ = 1$, and frictional / input forces are zero. Then by the above lemma, we have $\dot{x}_k^- = \dot{x}_{k+1}^- = 0$; the corresponding variable $\lambda_{k+1}'^+$ is now free to vary. We can now satisfy equation (2) by letting $\lambda_{k+1}'^+ = 1$ (this is what the LCP outputs as well). Observe that since $\mu g = 0$, this actual creates a fictitious force $\lambda_{k+1}^+ = -1$ which prevents the block from increasing in velocity and keeps it moving in a straight line.

Essentially, we structured the problem in a way that the slack variable $w = Mz + q$ had rough physical meaning. Unless the block is at rest with no external force, there is always a physical positive friction force in one direction, and a fictitious negative friction force in the opposite direction. The nice thing about this formulation is that $M$ is 2x2 and a multiple of the identity and therefore positive definite, so we're guaranteed a unique LCP solution. We also managed to use 2 complimentarity constraints instead of 3.

## 2.4 Generalization

Do the convenient above results generalize? Let's consider the case where $u$ does not necessarily align with $\dot{x}$; so, we're looking at an $xy$ plane. You can't just solve an LCP for $x$ and an LCP for $y$ since it's unclear what you would set your max friction force to. That's coupled between $x$ and $y$ and must be in the direction of $v$ from the next time step. I think you can probably do some kind of a polyhedral approximation similar to the classic Stewart paper, but I haven't worked it out yet.

**Note 1.** *What if we aligned bases with $u$ and $v$?*

## 2.5 Traditional formulation

The traditional formulation of the above bidirectional sliding block problem involves a third variables, $\gamma$, which is effectively an upper bound on the absolute value of the sliding velocity. We still decompose $\lambda$ into $\lambda_+$ and $\lambda_-$, but these are now $z$ variables instead of $w$ variables in the LCP. Velocity isn't a decision variable in the formulation. Observe that we can write (assuming $\Delta t = 1$):

$$v_{k+1} = v_k + \lambda_{k+1}^+ - \lambda_{k+1}^- + u_{k+1}$$

And now have the following complimentarity constraints:

$$0 \le \lambda_{k+1}^+ \perp \gamma_{k+1} + v_{k+1} \ge 0 \quad \text{use positive friction force if block is moving to the left}$$
$$0 \le \lambda_{k+1}^- \perp \gamma_{k+1} - v_{k+1} \ge 0 \quad \text{use negative friction force if block is moving to the right}$$
$$0 \le \gamma_{k+1} \perp mg\mu - \lambda_{k+1}^+ - \lambda_{k+1}^- \ge 0 \quad \text{use max friction force if block is moving}$$

We can now substitute in for $v_{k+1}$ to only include our decision variables and constants:

$$0 \le \lambda_{k+1}^+ \perp \gamma_{k+1} + v_k + \lambda_{k+1}^+ - \lambda_{k+1}^- + u_{k+1} \ge 0$$
$$0 \le \lambda_{k+1}^- \perp \gamma_{k+1} - v_k - \lambda_{k+1}^+ + \lambda_{k+1}^- - u_{k+1} \ge 0$$
$$0 \le \gamma_{k+1} \perp mg\mu - \lambda_{k+1}^+ - \lambda_{k+1}^- \ge 0$$

Now this can be written in matrix form as:

$$0 \le \begin{bmatrix} \lambda_{k+1}^+ \\ \lambda_{k+1}^- \\ \gamma_{k+1} \end{bmatrix} \perp \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & 1 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_{k+1}^+ \\ \lambda_{k+1}^- \\ \gamma_{k+1} \end{bmatrix} + \begin{bmatrix} v_k + u_{k+1} \\ -v_k - u_{k+1} \\ mg\mu \end{bmatrix} \ge 0$$

In this case I believe $M$ is a P-matrix and Lemke's algorithm is guaranteed to have a unique solution.

**Note 2.** *Matt says P and Q matrices are useless. Didn't quite understand why but apparently we can't really use them effectively.*

### 2.5.1 As Mixed LCP

Usually LCP formulations don't go through the trouble of subbing $v_{k+1}$ back in. It then looks something like the following:

$$\begin{bmatrix} 1 & 1 & -1 & 0 \\ 1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ 0 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} v_{k+1} \\ \lambda_{k+1}^+ \\ \lambda_{k+1}^- \\ \gamma_{k+1} \end{bmatrix} + \begin{bmatrix} v_k + u_{k+1} \\ 0 \\ 0 \\ mg\mu \end{bmatrix} = \begin{bmatrix} 0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$$0 \le \begin{bmatrix} \lambda_{k+1}^+ \\ \lambda_{k+1}^- \\ \gamma_{k+1} \end{bmatrix}, \quad 0 \le \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}, \quad \begin{bmatrix} \lambda_{k+1}^+ \\ \lambda_{k+1}^- \\ \gamma_{k+1} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = 0$$

# 3 Learning

We want to ultimately learn the dynamics for a full Kuka arm interacting with an object, but we start with just the sliding block example. The most obvious idea is to learn the unknown friction force $\lambda = f(\dot{x}, u)$ by fitting a neural network or some other function approximator to the raw data. However, this approach won't be able to properly capture sticking behavior. We need some way of embedding the complementary nature of the problem into our learning algorithm.

## 3.1 Known frictional forces

### 3.1.1 Direct formulation

Assume that we know the friction forces vector $\boldsymbol{\lambda} = \begin{bmatrix} \lambda^+ & \lambda^- \end{bmatrix}$, which contains the complementary positive/negative friction components. We want to predict the velocity vector $\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x}^+ & \dot{x}^- \end{bmatrix}$; we will denote the scalar velocity as $\dot{x}$ without boldface. Let's claim that $\dot{\boldsymbol{x}}$ is affine in $\boldsymbol{\lambda}$, as is the case from the underlying dynamics. Then we can learn two functions $\boldsymbol{f}$ (2x1 vector) and $G$ (2x2 matrix) such that:

$$\dot{\hat{\boldsymbol{x}}}_{k+1} = \boldsymbol{f}(\dot{x}_k, u_{k+1}) + G(\dot{x}_k, u_{k+1})\boldsymbol{\lambda}_{k+1}$$

Where $\dot{\hat{\boldsymbol{x}}}$ denotes learning algorithm predictions. We also need to consider our complementarity constraints:

$$0 \le \boldsymbol{\lambda}_{k+1} \perp \dot{\hat{\boldsymbol{x}}}_{k+1} \ge 0$$

Let's now look at the true system we're trying to learn:

$$\dot{x}_{k+1}^+ = \dot{x}_k^+ + \Delta t \cdot u_{k+1} - \dot{x}_k^- - \Delta t \cdot \lambda_{k+1}^-$$
$$\dot{x}_{k+1}^- = \dot{x}_k^- - \Delta t \cdot u_{k+1} - \dot{x}_k^+ - \Delta t \cdot \lambda_{k+1}^+$$

This can be written in matrix form as:

$$\dot{\boldsymbol{x}}_{k+1} = \begin{bmatrix} \dot{x}_{k+1}^+ \\ \dot{x}_{k+1}^- \end{bmatrix} = \begin{bmatrix} \dot{x}_k + \Delta t \cdot u_{k+1} \\ -\dot{x}_k - \Delta t \cdot u_{k+1} \end{bmatrix} + \begin{bmatrix} 0 & -\Delta t \\ -\Delta t & 0 \end{bmatrix} \begin{bmatrix} \lambda_{k+1}^+ \\ \lambda_{k+1}^- \end{bmatrix}$$

Therefore we know to learn this that $\boldsymbol{f}$ and $G$ only need to be affine in their inputs in order to capture the true dynamics. We now just need to define our cost function:

$$c = \alpha_1 \left\| \dot{\hat{\boldsymbol{x}}}_{k+1} - \dot{\boldsymbol{x}}_{k+1} \right\|_2 + \alpha_2 \left\| \boldsymbol{\lambda}_{k+1} \odot \dot{\hat{\boldsymbol{x}}}_{k+1} \right\|_1 + \alpha_3 \left\| \max\left(0, -\dot{\hat{\boldsymbol{x}}}_{k+1}\right) \right\|_1$$

Where max denotes elementwise maximum and $\odot$ denotes elementwise multiplication. The first term encodes the prediction error, the second the complementarity violation, and the third the nonnegativity violation.

### 3.1.2 With slack friction

Now assume that we know the slack friction forces vector $\boldsymbol{\lambda}' = \begin{bmatrix} \lambda'^- & \lambda'^+ \end{bmatrix}$ (pos/neg order flipped to maintain complimentarity). Everything from above holds, replacing $\boldsymbol{\lambda}$ with $\boldsymbol{\lambda}'$, except the actual system is:

$$\begin{bmatrix} \dot{x}_{k+1}^+ \\ \dot{x}_{k+1}^- \end{bmatrix} = - \begin{bmatrix} -\dot{x}_k - u_{k+1}\Delta t + \mu g \Delta t \\ \dot{x}_k + u_{k+1}\Delta t + \mu g \Delta t \end{bmatrix} + \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} \lambda'^-_{k+1} \\ \lambda'^+_{k+1} \end{bmatrix}$$

**Remark 3.** *Let's compare the two approaches with actual/slack friction. The only real difference comes from what kind of LCP conditions we are penalizing in the cost function. With actual friction, we are penalizing occurrences where friction occurs in the same direction as motion; in the second case, we are enforcing the rule that friction force must max out in the direction opposing motion. Neither of these say anything about what should occur when the box is standing still, which is what we're interested in since we want to enforce sticking. Also, in practice I imagine measuring slack friction is a lot harder since you need to determine both actual friction and theoretical maximum friction.*

**Remark 4.** *I think going about this by penalizing LCP violations is not the right path forward. Instead, we should be using the fact that velocities are decomposed to help enforce sticking. More specifically, we should probably relu the output to allow totally zero velocities to be produced. Update: this works! But unless we can formulate velocity in this complementary framework doesn't really generalize well.*

**Remark 5.** *When considering $\boldsymbol{f}$ and $G$, you need to let them be linear maps with biases because there are bias terms in the actual system.*

### 3.1.3 Traditional formulation

Assume we know the decision variables of the LCP:

$$\boldsymbol{\lambda}_{k+1} = \begin{bmatrix} \lambda_{k+1}^+ \\ \lambda_{k+1}^- \\ \gamma_{k+1} \end{bmatrix}$$

We want to find an $\boldsymbol{f}$ and $G$ such that the following equation is always satisfied:

$$0 \leq \boldsymbol{\lambda}_{k+1} \perp \boldsymbol{f}(\dot{x}_k, u_{k+1}) + G(\dot{x}_k, u_{k+1})\boldsymbol{\lambda}_{k+1} \geq 0$$

We can penalize violations as follows:

$$c = \alpha_1 \|\boldsymbol{\lambda}_{k+1} \odot (\boldsymbol{f}(\dot{x}_k, u_{k+1}) + G(\dot{x}_k, u_{k+1})\boldsymbol{\lambda}_{k+1})\| + \alpha_2 \|\max(0, -\boldsymbol{f}(\dot{x}_k, u_{k+1}) - G(\dot{x}_k, u_{k+1})\boldsymbol{\lambda}_{k+1})\|_1$$

Observe that the trivial solution is $\boldsymbol{f} = 0$ and $G = 0$. Therefore we know that it's not enough just to penalize complementarity/nonnegativity constraints; we need to impose more structure. Note that $\boldsymbol{f}$ and $G$ are invariant to scaling; we can therefore just fix the first element of $G$ equal to one, for example. We could also experiment with imposing the following structure on $M$:

$$M = \sum_i \phi_i \phi_i^T + \sum_i \alpha_i \Phi_i^+ + \sum_i \beta_i \Phi_i^{asym}$$

Finally, we should train an additional model for $\dot{x}_{k+1} = h(\dot{x}_k, u_{k+1})$.

*Results:* I tried two things: just learning $\boldsymbol{f}$ and $G$ directly (each element is affine in $\dot{x}_k$ and $u_{k+1}$) and learning with a basis. In the first case, things didn't converge unless I had a ton of samples (10-30k ish). To get the correct scaling, I constrained one element of $G$ from each row. Furthermore, I needed to make $G$ not a function of $\dot{x}_k$ and $u_{k+1}$, otherwise it just converged with some weird degenerate solutions. Then the correct matrix was recovered.

### 3.1.4 Linear program formulation

Just using a normal gradient based method for the above problem doesn't really converge reliably. However, we can impose a linear structure and turn it into a linear program. Let's turn the nonnegativity condition into a constraint ($M\boldsymbol{\lambda} + \boldsymbol{q} \geq \boldsymbol{0}$) and the complimentarity condition into the objective (minimize $\boldsymbol{\lambda}^T(M\boldsymbol{\lambda} + \boldsymbol{q})$). Making both into inequality constraints would probably mess up the solver, since they often rely on finding an initial feasible point. Let's now try and vectorize these, starting with the nonnegativity constraint:

We want to write $M\boldsymbol{\lambda} + \boldsymbol{q} \geq 0$. Note that we are assuming that $M$ is not dependant on $\dot{x}$ or $u$ and $\boldsymbol{q}(\dot{x}, u) = W \begin{bmatrix} \dot{x} & u & 1 \end{bmatrix}^T$, where W is a 3x3 matrix. This corresponds to each element of $q$ being affine in $\dot{x}$ and $u$; we will denote $\boldsymbol{d}^i = \begin{bmatrix} \dot{x} & u & 1 \end{bmatrix}$ for the $i$th data point. Denote the $i$th lcp solution data point as $\boldsymbol{\lambda}^i$, and $\boldsymbol{0} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$. We can then write $M\boldsymbol{\lambda} + \boldsymbol{q} \geq 0$ as:

$$
\begin{bmatrix}
\boldsymbol{\lambda}^1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^1 & \boldsymbol{0} & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{\lambda}^1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^1 & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\lambda}^1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^1 \\
& & \cdots & \cdots & & \\
\boldsymbol{\lambda}^n & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^n & \boldsymbol{0} & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{\lambda}^n & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^n & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\lambda}^n & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^n
\end{bmatrix}
\begin{bmatrix}
M_{11} \\ M_{12} \\ M_{13} \\ \cdots \\ M_{31} \\ M_{32} \\ M_{33} \\ W_{11} \\ W_{12} \\ W_{13} \\ \cdots \\ W_{31} \\ W_{32} \\ W_{33}
\end{bmatrix}
\geq 0
$$

Now we want to express the cost function $\boldsymbol{\lambda}^T(M\boldsymbol{\lambda} + \boldsymbol{q})$:

$$
\begin{bmatrix} \boldsymbol{\lambda}^1 & \cdots & \boldsymbol{\lambda}^n \end{bmatrix}
\begin{bmatrix}
\boldsymbol{\lambda}^1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^1 & \boldsymbol{0} & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{\lambda}^1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^1 & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\lambda}^1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^1 \\
& & \cdots & \cdots & & \\
\boldsymbol{\lambda}^n & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^n & \boldsymbol{0} & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{\lambda}^n & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^n & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\lambda}^n & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^n
\end{bmatrix}
\begin{bmatrix}
M_{11} \\ M_{12} \\ M_{13} \\ \cdots \\ M_{31} \\ M_{32} \\ M_{33} \\ W_{11} \\ W_{12} \\ W_{13} \\ \cdots \\ W_{31} \\ W_{32} \\ W_{33}
\end{bmatrix}
$$

This is now a linear program! Ideally, we'd solve it and the cost would go to zero. However, we also need to take into account the trivial solution. In order to prevent just driving $M$ and $\boldsymbol{q}$ to zero, we can instead have $M\boldsymbol{\lambda} + q \geq \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is some small positive constant vector.

Another thing that could help drive things to zero that should be zero is $L1$ regularization. We do that by adding a vector $\boldsymbol{s}$ of length equal to the sum of $M$ and $W$'s sizes. These would then be decision variables, and we would have the following constraint matrix:

$$
\begin{bmatrix}
\boldsymbol{\lambda}^1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^1 & \boldsymbol{0} & \boldsymbol{0} & 0 & \cdots & 0 \\
\boldsymbol{0} & \boldsymbol{\lambda}^1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^1 & \boldsymbol{0} & 0 & \cdots & 0 \\
\boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\lambda}^1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^1 & 0 & \cdots & 0 \\
& & \cdots & \cdots & & & & & \\
\boldsymbol{\lambda}^n & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^n & \boldsymbol{0} & \boldsymbol{0} & 0 & \cdots & 0 \\
\boldsymbol{0} & \boldsymbol{\lambda}^n & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^n & \boldsymbol{0} & 0 & \cdots & 0 \\
\boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\lambda}^n & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}^n & 0 & \cdots & 0 \\
1 & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 0 & \cdots \\
0 & 1 & \cdots & \cdots & \cdots & 0 & 0 & 1 & \cdots \\
\cdots & & & & & & & & \\
-1 & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 0 & \cdots \\
0 & -1 & \cdots & \cdots & \cdots & 0 & 0 & 1 & \cdots
\end{bmatrix}
\begin{bmatrix}
M_{11} \\ M_{12} \\ M_{13} \\ \cdots \\ M_{31} \\ M_{32} \\ M_{33} \\ W_{11} \\ W_{12} \\ W_{13} \\ \cdots \\ W_{31} \\ W_{32} \\ W_{33} \\ s_1 \\ \cdots \\ s_{18}
\end{bmatrix}
\geq 0
$$

The new cost vector would just append some constants $\mu$ for length $\boldsymbol{s}$.

## 3.2 Unknown frictional forces

What if we don't know the friction forces? Then we definitely need to solve an LCP on every training iteration for both the friction and velocities. Under the traditional framework, we would learn $\boldsymbol{f}$ and $G$, use those to solve for the friction forces, then calculate $\dot{x}_{k+1}$ from there. The issue here is, can we predict sticking and sliding if there is noise on $\boldsymbol{f}$ and $G$? I looked at this in *lcp_study.py*. It turns out if only single elements of $M$ and $\boldsymbol{q}$ are perturbed, the solution is usually stable (sticking is observed). However, as soon as we add random noise, this breaks down. When $u = 0$ we sometimes coincidentally get the right answer if all the $\boldsymbol{q}$ elems are strictly positive (gives zero friction forces as trivial solution). However, as soon as $u \neq 0$, we need a precise nonzero friction force equal to $-u$ to induce sticking. Random noise makes this impossible. Let's look at one particular case more analytically:

### 3.2.1 Small $\boldsymbol{q}$ error

Let's start from our original traditional formulation equations, assuming $v_k = 0$ and $u_{k+1} = 0$, and including a small error $\epsilon > 0$ in $\boldsymbol{q}$:

$$0 \leq \lambda_{k+1}^+ \perp \gamma_{k+1} + v_{k+1} - \epsilon \geq 0$$
$$0 \leq \lambda_{k+1}^- \perp \gamma_{k+1} - v_{k+1} \geq 0$$
$$0 \leq \gamma_{k+1} \perp mg\mu - \lambda_{k+1}^+ - \lambda_{k+1}^- \geq 0$$

For the first complimentarity statement, we have two cases: either $\gamma_{k+1} + v_{k+1} - \epsilon \geq 0$, or $\gamma_{k+1} + v_{k+1} - \epsilon = 0$. Assume the second case. Then we can substitute into the RHS inequality of the second complimentarity condition to get $\gamma_{k+1} \geq \frac{\epsilon}{2}$. Let's again assume that this is a hard equality; i.e. $\gamma_{k+1} = \frac{\epsilon}{2}$. We now want to show that the remaining inequalities can be satisfied. Since $\gamma_{k+1} > 0$, we know $mg\mu = \lambda_{k+1}^+ + \lambda_{k+1}^-$. Setting $mg\mu = \lambda_{k+1}^+$ and $\lambda_{k+1}^- = 0$ satisfies the LCP. Therefore, we have shown that a small error $-\epsilon$ in the first element of $\boldsymbol{q}$ produces a valid solution with $v_{k+1} = \gamma_{k+1} = \frac{\epsilon}{2}$.

### 3.2.2 Traditional formulation

Following the traditional formulation of the LCP we can let our decision variables be:

$$\boldsymbol{\lambda}_{k+1} = \begin{bmatrix} \lambda_{k+1}^+ \\ \lambda_{k+1}^- \\ \gamma_{k+1} \end{bmatrix}$$

Then on each iteration of the learning algorithm we would solve an LCP of the form:

$$0 \leq \boldsymbol{\lambda}_{k+1} \perp \boldsymbol{f}(\dot{x}_k, u_{k+1}) + G(\dot{x}_k, u_{k+1})\boldsymbol{\lambda}_{k+1} \geq 0$$

This would give us the friction forces, from which we could calculate $\dot{x}_{k+1}$ and then evaluate the same above cost function. This approach has a few issues. First of all, the LCP might not even be solvable. You could try to fix this by imposing some structure on $G$, but this is not that easy. If we forced $G$ to be positive semidefinite, for example by letting $G = \sum_i \phi_i \phi_i^T + \epsilon I$, we wouldn't be able to learn the correct $M$ matrix in the traditional formulation example. The smaller issue is that training time would be dramatically slower because you'd have to solve an LCP on every iteration.

Furthermore, I don't think the traditional way is capable of encoding sticking. If you keep your velocity as a real number, there's no meaningful way of constraining it to be zero in the next time step under sticking. However, if you decompose velocity into positive and negative components, you could potentially RELU the two output velocities to help give you that nonlinear sticking behavior.

# 4 Random ideas

- Markov random field / Markov chain for state transitions. Captures multimodality. However, continuous and multidimensional state space is an issue. Could potentially use stick/slip/detach as states instead.

- Generalizing direct formulation to non-full-rank jacobian scenarios