Stochastic-Optimization-Based Stochastic Control Problem

Xinyu Li, Ziyu Lu

August 2019

1 Introduction

Our project is to apply optimization methods from machine learning to stochastic control problems, which are traditionally thought to be outside their specialty. A stochastic control problem involves a stochastic dynamical system that can be observed and controlled in limited ways. The control strategy depends on observations from the dynamical system and is to optimize some measure of performance. The optimization methods we apply are variants of stochastic gradient descent that have been developed for machine learning. One advantage of machine learning optimization algorithms is that the control parameters can be adjusted based on only a few stochastic simulations, which are far fewer than what would be needed to get an accurate performance estimate of the system.

Our application is a model of the insulin/glucose metabolism that could hopefully be part of an artificial pancreas device for people with type 1 diabetes. This device would be attached to a patient as an automatic insulin source today. It would measure the glucose level of the patient every few minutes, calculate the time-dependent insulin dose using stochastic control, and inject the patient with the insulin needed. This model is highly non-linear and the actual control cannot be computed by analytical means.

We also test the machine learning stochastic optimization methods on a linear stochastic control problem that can be solved in closed form. We observed that the optimization algorithms are effective in determining the control parameters, but may require many iterations. We also notice that the parameters in the optimization algorithms that are recommended for machine learning do not seem to be the most effective for these control applications.

In this report, we begin by presenting a linear stochastic optimal control problem in section 2. We solve a linear quadratic Gaussian problem with Kalman filter and linear control, where the theoretical optimal control is available. In order to compare the results given by machine learning optimization algorithms with the theoretical solution, we conduct a series of experiments to tune the parameters in the algorithms. In section 3, we move on to the nonlinear insulin control problem where theoretical solution no longer exists. We propose a model of insulin/glucose metabolism that could be used for an artificial pancreas for type 1 diabetes and determine the control parameters by machine learning optimization algorithms. Finally, we close the report with a discussion on our results and a reflection on this project in section 4.

2 Linear quadratic Gaussian control problem

2.1 Model

Consider a spring-mass model described by the 2D Ornstein-Uhlenbeck process:

$$\begin{cases} dx = vdt \\ dv = (-\frac{k}{m}x - \frac{\gamma}{m}v)dt + \sigma dW \end{cases}$$

where x, v are the position and the velocity of the object, k is the spring constant, γ is the friction coefficient, σ is the noise coefficient, and W is a Wiener process.

To prepare for the insulin control problem where the observation is discrete, we discretize the state and observation of the system into a series of time steps: Let $T = N\Delta t$, $X_n = \begin{pmatrix} x(n\Delta t) \\ v(n\Delta t) \end{pmatrix}$, $n = 0, 1, \ldots, N$. Then by spectral decomposition, the dynamics of the system can be captured by

$$X_{n+1} = AX_n + W_n$$

where A is an update matrix depending on Δt,

$$A = A(\Delta t) = \frac{1}{\lambda_2 - \lambda_1} \begin{pmatrix} \lambda_2 e^{\lambda_1 \Delta t} - \lambda_1 e^{\lambda_2 \Delta t} & -e^{\lambda_1 \Delta t} + e^{\lambda_2 \Delta t} \\ \lambda_1 \lambda_2 \left(e^{\lambda_1 \Delta t} - e^{\lambda_2 \Delta t} \right) & -\lambda_1 e^{\lambda_1 \Delta t} + \lambda_2 e^{\lambda_2 \Delta t} \end{pmatrix}$$

and $\lambda_1 = -\frac{\gamma}{2m} + i\sqrt{\frac{4km-\gamma^2}{4m^2}}$, $\lambda_2 = -\frac{\gamma}{2m} - i\sqrt{\frac{4km-\gamma^2}{4m^2}}$. W_n is a Gaussian noise with mean 0 and covariance R, where

$$R = \left(\frac{\sigma}{\lambda_2 - \lambda_1} \right)^2 \begin{pmatrix} \frac{e^{2\lambda_1 \Delta t}-1}{2\lambda_1} + \frac{e^{2\lambda_2 \Delta t}-1}{2\lambda_2} - 2\frac{e^{(\lambda_1+\lambda_2)\Delta t}-1}{(\lambda_1+\lambda_2)} & \frac{e^{2\lambda_1 \Delta t}-1}{2} + \frac{e^{2\lambda_2 \Delta t}-1}{2} - (e^{(\lambda_1+\lambda_2)\Delta t} - 1) \\ \frac{e^{2\lambda_1 \Delta t}-1}{2} + \frac{e^{2\lambda_2 \Delta t}-1}{2} - (e^{(\lambda_1+\lambda_2)\Delta t} - 1) & \lambda_1 \frac{e^{2\lambda_1 \Delta t}-1}{2} + \lambda_2 \frac{e^{2\lambda_2 \Delta t}-1}{2} - 2\lambda_1\lambda_2 \frac{e^{(\lambda_1+\lambda_2)\Delta t}-1}{(\lambda_1+\lambda_2)} \end{pmatrix}$$

With control, the state update becomes

$$X_{n+1} = AX_n + BU_n + W_n$$

where U_n denotes the control at time step n, and $B = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.
The observation Z_{n+1} at time $n+1$ can be expressed as

$$Z_{n+1} = CX_{n+1} + V_{n+1}$$

where $C = \begin{pmatrix} 1 & 0 \end{pmatrix}$, and V_n is a gaussian noise with mean 0 and covariance S, V_n independent of W_n $\forall n$.
Using a Kalman filter, the state estimation update is given by

$$\widehat{X}_{n+1} = A\widehat{X}_n + BU_n + K(Z_{n+1} - C(A\widehat{X}_n + BU_n))$$

where K denotes the Kalman gain, and $\widehat{X}_0 \equiv X_0$.
Define cost rate with weighting parameter r

$$J_n(G) = \mathbb{E}[|X_{n+1}|^2 + r|U_n|^2], \quad \text{for } n = 0, 1, \ldots, N-1$$

and the total cost

$$F(K, G) = \frac{1}{2N} \mathbb{E}[\sum_{n=0}^{N-1} (X_{n+1}^T X_{n+1} + r U_n^T U_n)]$$

Our goal is to find the optimal K and G such that the cost $F(K, G)$ is minimized.

2.2 Theoretical solution

Linear Quadratic Gaussian is mainly composed with two steps: Kalman Filter and Linear Quadratic Regulator (LQR)

2.2.1 Kalman Filter

Let Y_n be the prediction error:

$$Y_n = X_n - \hat{X}_n.$$

Let T_n be the covariance matrix of prediction error Y_n:

$$T_n = \mathbb{E}\left[Y_n Y_n^t \right].$$

Let $\mathcal{Z}_n$ be a filtration of observations from time 0 to time n:

$$\mathcal{Z}_n = \{Z_0, ..., Z_n\}.$$

Given a set of observations $\mathcal{Z}_n$, we want to find a Kalman filter K_n to minimize the mean square error of prediction [1]. Thus,

$$\hat{X}_n = \text{argmin}\mathbb{E}[Y_n^T Y_n | Z_0, ..., Z_n] \tag{1}$$

Since $\hat{X}_n$ depends on $Z_0, ..., Z_n$,

$$\mathbb{E}[\hat{X}_n|\mathcal{Z}_n] = \hat{X}_n,\ \mathbb{E}[\hat{X}_n^T \hat{X}_n|\mathcal{Z}_n] = \hat{X}_n^T \hat{X}_n,$$

Therefore,

$$\begin{aligned}
\mathbb{E}[Y_n^T Y_n|\mathcal{Z}_n] &= \mathbb{E}\left[(X_n - \hat{X}_n)^T (X_n - \hat{X}_n)|\mathcal{Z}_n\right] \\
&= \mathbb{E}\left[X_n^T X_n|\mathcal{Z}_n\right] - \mathbb{E}\left[X_n^T|\mathcal{Z}_n\right]\hat{X}_n - \hat{X}_n^T \mathbb{E}\left[X_n|\mathcal{Z}_n\right] + \mathbb{E}\left[\hat{X}_n^T \hat{X}_n|\mathcal{Z}_n\right] \\
&= \mathbb{E}\left[X_n^T X_n|\mathcal{Z}_n\right] - \mathbb{E}\left[X_n^T|\mathcal{Z}_n\right]\mathbb{E}\left[X_n|\mathcal{Z}_n\right] + (\hat{X}_n - \mathbb{E}\left[X_n|\mathcal{Z}_n\right])^T (\hat{X}_n - \mathbb{E}\left[X_n|\mathcal{Z}_n\right])
\end{aligned}$$

Thus, when $\hat{X}_n = \mathbb{E}\left[X_n|\mathcal{Z}_n\right]$, the above equation is minimized. Also, notice that when $\hat{X}_n = \mathbb{E}\left[X_n|\mathcal{Z}_n\right]$,

$$\mathbb{E}\left[Y_n|\mathcal{Z}_n\right] = \mathbb{E}\left[X_n - \hat{X}_n|\mathcal{Z}_n\right] = \mathbb{E}\left[X_n|\mathcal{Z}_n\right] - \hat{X}_n = 0.$$

Therefore, $\mathbb{E}\left[X_n|\mathcal{Z}_n\right] = \hat{X}_n$ is equivalent to making Y_n independent of $Z_0, Z_1, ..., Z_n$. By deduction, we want Y_{n+1} to be independent of $Z_0, Z_1, ..., Z_{n+1}$. Therefore, for $k = 1, 2, ..., n+1$

$$\begin{aligned}
0 &= \mathbb{E}\left[(Y_{n+1})(Z_k)^T\right] \\
&= \mathbb{E}\left[\left(X_{n+1} - \hat{X}_{n+1}\right) Z_k^T\right] \\
&= \mathbb{E}\left\{AX_n + BU_n + W_n - \left[A\hat{X}_n + BU_n + K_n(Z_{n+1} - C(A\hat{X}_n + BU_n))\right]\right\}\left[Z_k^T\right] \\
&= \mathbb{E}[AY_n + W_n - K_n(CAY_n + CW_n + V_{n+1})][Z_k^T]
\end{aligned} \tag{2}$$

Since for $k = 1, 2, ..., n$, Y_n is independent of Z_k, and V_{n+1} is independent of Z_k, equation 2 always works by induction. Therefore, we only need to consider the scenario when $k = n+1$,

$$\begin{aligned}
0 &= \mathbb{E}\left[AY_n + W_n - K_n(CAY_n + CW_n + V_{n+1})\right]\left[Z_{n+1}^T\right] \\
&= \mathbb{E}\left[AY_n + W_n - K_n(CAY_n + CW_n + V_{n+1})\right]\left[CAY_n + CW_n + V_{n+1}\right]^T \\
&= AT_n A^T C^T + RC^T - K_n CAT_n A^T C^T - K_n CRC^T - K_n S
\end{aligned}$$

Therefore,

$$K_n = \left(AT_n A^t + R\right) C^t \left(CAT_n A^t C^t + CTC^t + S\right)^{-1}\ .$$

The recurrence relation of T_n can be found in the similar way

$$\begin{aligned}
T_{n+1} &= \mathbb{E}[X_{n+1} - \hat{X}_{n+1}][X_{n+1} - \hat{X}_{n+1}]^T \\
&= \left(A - K_n CA\right) T_n \left(A - K_n CA\right)^T + \left(I - K_n C\right) R \left(I - K_n C\right)^T + K_n S K_n^T
\end{aligned}$$

2.2.2 LQR: Approach 1 - Find G in steady state

We used two ways to find optimal control gain matrix G. The first approach only considers G in steady state. In this approach, a non-linear equation $f(G) = 0$ would first be found and implement Quasi-Newton method in Python3 to solve for G.

Find S in the steady state Let S_n be the covariance matrix of X_n, R be the constant covariance matrix of noise W_n, then

$$S_{n+1} = (A + BG)S_n(A + BG)^T + R$$

Let denote M to be $A + BG$. In the steady state, let S be the covariance matrix of X, then we will have

$$S = (A + BG)S(A + BG)^T + R = MSM^T + R$$

As S is a symmetric matrix, we can rewrite the above equation as

$$\begin{pmatrix} S_{11} \\ S_{12} \\ S_{22} \end{pmatrix} = \begin{pmatrix} M_{11}^2 & 2M_{11}M_{12} & M_{12}^2 \\ M_{11}M_{21} & M_{11}M_{22} + M_{12}M_{21} & M_{22}M_{12} \\ M_{21}^2 & 2M_{21}M_{22} & M_{22}^2 \end{pmatrix} \begin{pmatrix} S_{11} \\ S_{12} \\ S_{22} \end{pmatrix} + \begin{pmatrix} R_{11} \\ R_{12} \\ R_{22} \end{pmatrix}$$

Let $\xi_S = \begin{pmatrix} S_{11} \\ S_{12} \\ S_{22} \end{pmatrix}$, $D = \begin{pmatrix} M_{11}^2 & 2M_{11}M_{12} & M_{12}^2 \\ M_{11}M_{21} & M_{11}M_{22} + M_{12}M_{21} & M_{22}M_{12} \\ M_{21}^2 & 2M_{21}M_{22} & M_{22}^2 \end{pmatrix}$, $\xi_R = \begin{pmatrix} R_{11} \\ R_{12} \\ R_{22} \end{pmatrix}$, $E = (I - D)^{-1}$, then

$$\xi_S = D\xi_S + \xi_R, \tag{3}$$

$$\xi_S = E\xi_R. \tag{4}$$

Differentiate S w.r.t G If differentiating S with respect to parameter θ in the steady state, $\theta \in \{G_1, G_2\}$, according to chain rule we will get

$$\dot{S} = M\dot{S}M^T + \dot{M}SM^T + MS\dot{M}^T.$$

If differentiating ξ_S with respect to parameter θ , we will get

$$\xi_{\dot{S}} = \dot{D}\xi_S + D\xi_{\dot{S}},$$

Then plugging in equation 4, we can find a formula of $\xi_{\dot{S}}$,

$$\xi_{\dot{S}} = E\dot{D}\xi_S = E\dot{D}E\xi_R \tag{5}$$

Use S to represent the cost rate Notice that $\mathbb{E}[X^TX] = \text{tr}(S)$; According to the cyclic property of trace, we also have

$$\mathbb{E}[U^TrU] = \text{tr}(X^TG^TrGX) = \text{tr}(G^TrGXX^T) = \text{tr}(G^TrGS) = \text{tr}(rGSG^T)$$

Then the cost rate can be represented as

$$J_n(G) = \text{tr}(S) + \text{tr}(rGSG^T). \tag{6}$$

Differentiating J_n w.r.t θ gives

$$\begin{aligned} 0 &= \frac{\partial J_n}{\partial \theta} \\ &= \text{tr}(\dot{S}) + \text{tr}(r(\dot{G}SG^T + G\dot{S}G^T + GS\dot{G}^T)) \\ &= \text{tr}(\dot{S}) + \text{tr}(r(2GS\dot{G}^T + G\dot{S}G^T)) \\ &= \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \xi_{\dot{S}} + r(2GS\dot{G}^T + G\dot{S}G^T) \end{aligned} \tag{7}$$

where θ is G_1 or G_2. Define $q := \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}$, $H := \begin{pmatrix} G_1^2 & 2G_1G_2 & G_2^2 \end{pmatrix}$. Then

$$G\dot{S}G^T = \begin{pmatrix} G_1^2 & 2G_1G_2 & G_2^2 \end{pmatrix} \xi_{\dot{S}} = H\xi_{\dot{S}}.$$

Let $\xi_G = \begin{pmatrix} G_1 & G_2 & 0 \\ 0 & G_1 & G_2 \end{pmatrix}$, then $S\dot{G}^T = \xi_{\dot{G}}\xi_S$. Plugging them and equation 4 and 5 into equation 7 finally yields

$$0 = [qE\dot{D} + r(HE\dot{D} + 2G\xi_{\dot{G}})]E\xi_R \tag{8}$$

where

$$\frac{\partial D}{\partial G_1} = \begin{pmatrix} 0 & 0 & 0 \\ a_{11} & a_{12} & 0 \\ 2(a_{21}+G_1) & 2(a_{22}+G_2) & 0 \end{pmatrix}, \frac{\partial D}{\partial G_2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & a_{11} & a_{12} \\ 0 & 2(a_{21}+G_1) & 2(a_{22}+G_2) \end{pmatrix},$$

$$\frac{\partial \xi_G}{\partial G_1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \frac{\partial \xi_G}{\partial G_2} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Equation 8 is only about G_1 and G_2, in the form of $f(G_1, G_2) = 0$, so we can use deterministic optimization method such as BFGS and Nelder-Mead to find out the value of G_1 and G_2.

2.2.3 LQR: Approach 2 - Dynamic Programming

In the first approach, though we find the optimal gain matrix G in the steady state, G may not be optimal in the transitional state. Therefore, we would review a classic Dynamic Programming approach [2] to solve for optimal transitional G_n for all time n.

Deterministic LQR First, let's consider the deterministic model without noise

$$X_{n+1} = AX_n + BU_n$$

Consider the cost-to-go function V_t for $t = 0, 1, 2...N$,

$$V_t(z) = \min_{U_t,...,U_{N-1}} \left[\sum_{n=t}^{N-1} (X_n^T X_n + U_n^T r U_n) + X_N^T X_N \right]$$

subject to $X_t = z$, and $X_{n+1} = AX_n + BU_n$, for $n = t, t+1, ...N$. Since V_t is a quadratic function, we can write

$$V_t(z) = z^T P_t z$$

where $P_t = P_t^T$. Now suppose we know $V_{t+1}(z)$, we want to find optimal U_t. We know that if the cost at time t subject to $U_t = w$, then

$$\begin{aligned} V_t(z) &= \min_w \left[z^T z + w^T r w + V_{t+1}(Az + Bw) \right] \\ &= \min_w \left[z^T z + w^T r w + (Az + Bw)^T P_{t+1}(Az + Bw) \right] \end{aligned}$$

We differentiate V_t w.r.t w to solve for optimal w^*:

$$\begin{aligned} 0 &= \frac{\partial V_t}{\partial w} \\ &= 2w^T r + 2(Az + Bw)^T P_{t+1} \end{aligned} \tag{9}$$

Therefore, the optimal w^* is

$$w^* = -(r + B^T P_{t+1} B)^{-1} B^T P_{t+1} Az \tag{10}$$

Now, let plug w^* back into V_t:

$$\begin{aligned} V_t(z) &= z^T z + w^{*T} r w^* + (Az + Bw^*)^T P_{t+1}(Az + Bw^*) \\ &= z^T z + w^{*T} r w^* + z^T A^T P_{t+1} Az + z^T A^T P_{t+1} Bw^* + w^{*T} B^T P_{t+1} Az + w^{*T} B^T P_{t+1} Bw^* \\ &= z^T (I + A^T P_{t+1} A) z + w^{*T} (r + B^T PB) w^* - 2z^T A^T P_{t+1} B (r + B^T P_{t+1} B)^{-1} B^T P_{t+1} Az \\ &= z^T (I + A^T P_{t+1} A) z + z^T A^T P_{t+1} B (r + B^T P_{t+1} B)^{-1} (r + B^T P_{t+1} B)(r + B^T P_{t+1} B)^{-1} B^T P_{t+1} Az \\ &- 2z^T A^T P_{t+1} B (r + B^T P_{t+1} B)^{-1} B^T P_{t+1} Az \\ &= z^T \left[I + A^T P_{t+1} A - A^T P_{t+1} B (r + B^T P_{t+1} B)^{-1} B^T P_{t+1} A \right] z \end{aligned}$$

Therefore, we find find the backward relation of P_t:

$$P_t = I + A^T \left[P_{t+1} - P_{t+1}B(r + B^T P_{t+1}B)^{-1}B^T P_{t+1}\right] A \tag{11}$$

Stochastic LQR The update function of stochastic model also includes a vector of Gaussian noise W_n,

$$X_{n+1} = AX_n + BU_n + W_n$$

Now consider the cost-to-go function V_t for $t = 0, 1, 2...N$ has the form

$$V_t(z) = z^T S_t z + q_t$$

subject to $X_t = z$,, $X_{n+1} = AX_n + BU_n + W_n$, for $n = t, t+1, ...N$, and $S_t = S_t^T$. Now suppose we know $V_{t+1}(z)$, we want to find optimal U_t. We know that if the cost at time t subject to $U_t = w$, then

$$\begin{aligned} V_t(z) &= \min_w \left[z^T z + w^T r w + V_{t+1}(Az + Bw + W_t)\right] \\ &= \min_w \left[z^T z + w^T r w + \mathbb{E}(Az + Bw + W_t)^T S_{t+1}(Az + Bw + W_t) + q_{t+1}\right] \\ &= z^T z + \text{tr}(RS_{t+1}) + q_{t+1} + \min_w \left[w^T r w + (Az + Bw)^T S_{t+1}(Az + Bw)\right] \end{aligned}$$

Then,

$$\frac{\partial V_t}{\partial w} = 2w^T r + 2(Az + Bw)^T S_{t+1}$$

which is same to equation 9 in the deterministic case. So the formulas of optimal w^* (equation 10) and S_n (equation 11) in deterministic case can also be applied to the stochastic LQR problem.

2.3 Optimization with machine learning optimization algorithms

Stochastic optimization is an old subject given new life by application to machine learning. Among the numerous variants of the classical gradient descent algorithm, the Adagrad algorithm[3], the Adam algorithm[4], and the RMSprop algorithm[5] are particularly well-known. They have been proven to be more effective than the vanilla gradient descent algorithm especially with deep neural networks, where the optimization problem is usually nonlinear and high-dimensional. While these algorithms have been widely applied in machine learning, little effort has been made to adapt them to stochastic optimal control problems beyond this field.

2.3.1 Review of RMSprop algorithm

The RMSprop algorithm is an adaptive learning rate method proposed by Geoff Hinton in Lecture 6e of his Coursera Class [5]. The basic idea of RMSprop algorithm is summarized in algorithm 1.

Algorithm 1 RMSprop

Input: Cost function f, total number of gradient descent iterations n, learning rate α, smoothing constant β, initial value of the parameter(s) to be optimized θ_0, initial $\phi_0 = 0$, term added to denominator to avoid divide-by-zero error ϵ.

for $i = 1, 2, \ldots, n$ **do**

Compute the cost with the parameter(s) at the current iteration $f(\theta_i)$

Compute the gradient $g_i = \frac{\partial f(\theta_i)}{\partial \theta_i}$

Update ϕ_i: $\phi_i = \beta\phi_{i-1} + (1 - \beta)g_i^2$ (element-wise square)

Update θ_i: $\theta_i = \theta_{i-1} - \alpha\frac{g_i}{\sqrt{\phi_i + \epsilon}}$

2.3.2 Gradient calculation

At each iteration, the gradient g_i is calculated using a backward recurrence relation:
Assume $\widehat{X}_0 = X_0 = x_0$. Define the total cost

$$F(K,G) = \sum_{n=0}^{N-1} (X_n^T X_n + r U_n^T U_n) + X_N^T X_N$$

In practice, we store the value of U_N as 0, so we can simply compute $F(K,G) = \sum_{n=0}^{N} (X_n^T X_n + r U_n^2)$. This is a random variable that depends on K, G and the random noise W_n and V_n. For K and G fixed, the expectation is

$$V_0(x_0, x_0, K, G) \triangleq \mathrm{E}\,[F]$$

The random cost starting at time j with $\widehat{X}_j = \widehat{x}$, $X_j = x$ is

$$F_j(x, \widehat{x}, K, G, V_{[j+1,\ldots,N]}, X_{[j+1,\ldots,N]}) = \sum_{n=j}^{N} (X_n^T X_n + r U_n^T U_n)$$

In this formula, $\widehat{X}_j = \widehat{x}$, $X_j = x$, so the $n = j$ term is $x^T x + r U_j^2$. Assume the "initial" condition at time $n = j$ is given by x and $\widehat{x}$, then the cost-to-go function is

$$V_j(x, \widehat{x}, K, G) = \mathrm{E}[\, F_j(x, \widehat{x}, K, G \ldots)] \ .$$

The optimal filtering and control problem, therefore, is to choose K and G to minimize $V_0(x_0, x_0, K, G)$, and we need to compute $\nabla_K V_0(x_0, x_0, K, G)$ and $\nabla_G V_0(x_0, x_0, K, G)$.
Define

$$Q_j \triangleq \nabla_K F_j(x, \widehat{x}, K, G \ldots), \quad H_j \triangleq \nabla_G F_j(x, \widehat{x}, K, G \ldots)$$

then

$$\nabla_K V_j(x, \widehat{x}, K, G) = \mathbb{E}[Q_j], \quad \nabla_G V_j(x, \widehat{x}, K, G) = \mathbb{E}[H_j]$$

Other important quantities are the gradients of the random cost and the cost-to-go functions with respect to x and $\widehat{x}$.
Define

$$P_j(x, \widehat{x}, K, G \ldots) \triangleq \nabla_{\widehat{x}} F_j(x, \widehat{x}, K, G \ldots)$$
$$T_j(x, \widehat{x}, K, G \ldots) \triangleq \nabla_x F_j(x, \widehat{x}, K, G \ldots)$$

then

$$\nabla_{\widehat{x}} V_j(x, \widehat{x}, K, G \ldots) = \mathrm{E}[\, P_j(x, \widehat{x}, K, G \ldots)]$$
$$\nabla_x V_j(x, \widehat{x}, K, G \ldots) = \mathrm{E}[\, R_j(x, \widehat{x}, K, G \ldots)]$$

An algorithm for computing Q_j, H_j uses P_j and T_j. The P_j and T_j are computed using a backward recurrence.
Start with

$$F_N = X_N^T X_N + r U_N^T U_N$$

The derivative with respect to K is zero, and the derivative with respect to G is 0 (since $U_N = 0$). The derivative with respect to $\widehat{x}$ (in numerator-layout) is

$$P_N = 2 r U_N^T \frac{\partial U_N}{\partial \widehat{x}_N} = 2 r U_N^T G = 0$$

The derivative with respect to x (in numerator-layout) is

$$\begin{aligned} T_N &= 2 X_N^T + 2 r U_N^T \frac{\partial U_N}{\partial \widehat{x}_N} \frac{\partial \widehat{x}_N}{\partial x_N} \\ &= 2 X_N^T + 2 r U_N^T G K C = 2 X_N^T \end{aligned}$$

From this start, the rest of the derivatives are calculated using back-propagation. The backward recurrence relation is

$$F_j(x, \widehat{x}, K, G \ldots) = x^T x + rU_j^2 + F_{j+1}(X_{j+1}, \widehat{X}_{j+1}, K, G \ldots)$$

In this formula, X_{j+1} is a function of x, $\widehat{x}$, G, and $\widehat{X}_{j+1}$ is a function of x, $\widehat{x}$, K and G.

$$X_{j+1} = Ax + BG\widehat{x} + W_j$$

$$\begin{aligned}
\widehat{X}_{j+1} &= (A + BG)\widehat{x} + K(Z_{j+1} - C(A + BG)\widehat{x}) \\
&= (I - KC)(A + BG)\widehat{x} + KZ_{j+1} \\
&= (I - KC)(A + BG)\widehat{x} + K(CX_{j+1} + V_{j+1}) \\
&= (I - KC)(A + BG)\widehat{x} + K(CAx + CBG\widehat{x} + CW_j + V_{j+1}) \\
&= [(I - KC)A + BG]\widehat{x} + KCAx + KCW_j + KV_{j+1}
\end{aligned}$$

The recurrence relations for P_j and T_j use the chain rule and differentiates the above update formulas for X and $\widehat{X}$

$$\nabla_{\widehat{x}} X_{j+1} = BG, \quad \nabla_x X_{j+1} = A$$

$$\nabla_{\widehat{x}} \widehat{X}_{j+1} = [(I - KC)A + BG], \quad \nabla_x X_{j+1} = KCA$$

The chain rule gives (in numerator-layout)

$$\begin{aligned}
P_j = \nabla_{\widehat{x}} F_j &= \nabla_{\widehat{x}}(x^T x + rU_j^T U_j) + \nabla_{\widehat{x}} F_{j+1}(X_{j+1}, \widehat{X}_{j+1}, K, G \ldots) \\
&= 2rU_j^T G + \nabla_{\widehat{X}_{j+1}} F_{j+1} \nabla_{\widehat{x}} \widehat{X}_{j+1} + \nabla_{X_{j+1}} F_{j+1} \nabla_{\widehat{x}} X_{j+1} \\
&= 2rU_j^T G + P_{j+1}[(I - KC)A + BG] + T_{j+1}BG
\end{aligned}$$

$$\begin{aligned}
T_j = \nabla_x F_j &= \nabla_x(x^T x + rU_j^T U_j) + \nabla_x F_{j+1}(X_{j+1}, \widehat{X}_{j+1}, K, G \ldots) \\
&= 2x^T + \nabla_{X_{j+1}} F_{j+1} \nabla_x X_{j+1} + \nabla_{\widehat{X}_{j+1}} F_{j+1} \nabla_x \widehat{X}_{j+1} \\
&= 2x^T + T_{j+1}A + P_{j+1}KCA
\end{aligned}$$

Since

$$\widehat{X}_{n+1} = A\widehat{X}_n + BU_n + K(Z_{n+1} - C(A\widehat{X}_n + BU_n))$$

we have

$$\nabla_K \widehat{X}_{j+1} = \text{diag}\{(Z_{j+1} - C(A\widehat{x} + BU_j))\}$$

$$\nabla_G \widehat{X}_{j+1} = (I - KC)B\widehat{x}^T$$

so

$$\begin{aligned}
Q_j &= \nabla_K(x^T x + rU_j^T U_j) + \nabla_K F_{j+1} + \nabla_{\widehat{X}_{j+1}} F_{j+1} \nabla_K \widehat{X}_{j+1} + \nabla_{X_{j+1}} F_{j+1} \nabla_K X_{j+1} \\
&= Q_{j+1} + P_{j+1} \cdot \text{diag}\{(Z_{j+1} - C(A\widehat{x} + BU_j))\}
\end{aligned}$$

Since

$$X_{n+1} = AX_n + BU_n + W_n$$

we have

$$\nabla_G X_{j+1} = B\widehat{x}^T$$

So

$$\begin{aligned}
H_j &= \nabla_G(x^T x + rU_j^T U_j) + \nabla_G F_{j+1} + \nabla_{\widehat{X}_{j+1}} F_{j+1} \nabla_G \widehat{X}_{j+1} + \nabla_{X_{j+1}} F_{j+1} \nabla_G X_{j+1} \\
&= 2rU_j^T \widehat{x}^T + H_{j+1} + P_{j+1}(I - KC)B\widehat{x}^T + T_{j+1}B\widehat{x}^T
\end{aligned}$$

2.3.3 Hyper-parameter settings

According our experiments, the hyper-parameters that have dominant influence on the performance of the algorithm are: total number of gradient descent iterations n, smoothing constant β, and mini-batch size M. To investigate the impacts of these three factors, we fix other hyper-parameters as constants and list them here.

- mass of the object $m = 1$
- spring constant $k = 1$
- friction coefficient $\gamma = 0.1$
- noise coefficient in SDE $\sigma = 0.1$
- covariance of observation noise $S = 0.3$
- initial displacement $x_0 = 1$
- initial velocity $v_0 = 0$
- initial state $X_0 = [x_0, v_0]^T = [1.0, 0.0]^T$
- total time in one simulation $t_1 = 30$
- step size in one simulation $dt = 1.5$
- total time steps in one simulation $N = \frac{t_1}{dt} + 1 = 21$
- learning rate $\alpha = 0.1$
- initial Kalman gain $K_0 = [0.5, 0.5]^T$
- initial control gain $G_0 = [-1.0, -0.1]$
- random seed $s = 1$

2.3.4 Theoretical result

In the steady state, $K = [1.12 \times 10^{-1}, 2.44 \times 10^{-3}]^T$, $G = [6.49 \times 10^{-1}, -4.89 \times 10^{-2}]$. The cost simulating with K, G in the steady state is 5.37×10^{-2}.
Evolution of displacement x, velocity v, Kalman gain K, and control gain G w.r.t. time are plotted in figures 1 to 4.

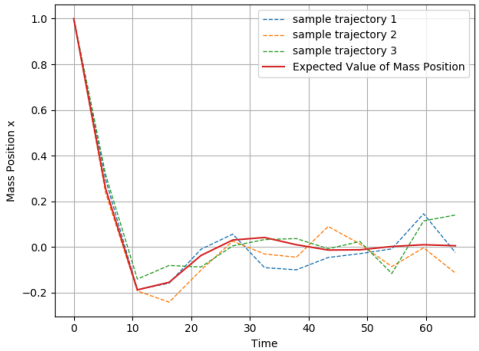

Figure 1: x w.r.t t: sample trajectories and expected value

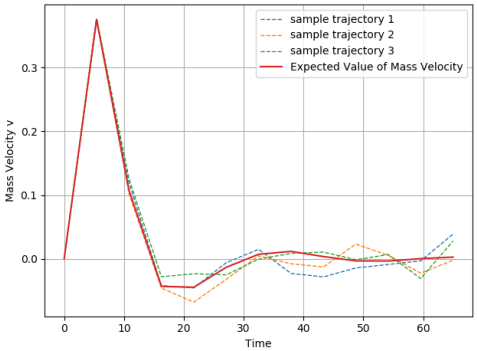

Figure 2: v w.r.t t: sample trajectories and expected value

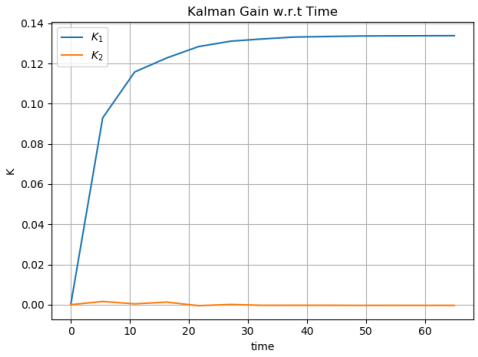

Figure 3: Kalman gain K w.r.t time

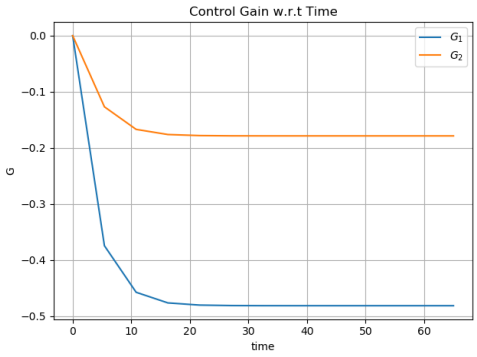

Figure 4: Control gain G w.r.t time

2.3.5 Numerical results

Experiments with n With $n = 2000$, $\beta = 0.99$ (default value in PyTorch), $M = 8$, the ultimate K is $[9.17 \times 10^{-2}, 1.70 \times 10^{-2}]^T$, and the ultimate G is $[6.08 \times 10^{-1}, -6.67 \times 10^{-2}]$. The difference (in L2 norm) between the ultimate K and the theoretical steady state K is 2.49×10^{-2}, and the difference (in L2 norm) between the ultimate G and the theoretical steady state G is 4.47×10^{-2}. The cost simulating with the ultimate K, G is 7.79×10^{-2}, and testing the ultimate K, G on 1000 simulations gives an expected cost of 6.94×10^{-2}.
Figures 5 to 7 show the cost decay, the decay in the difference between K, G and theoretical steady state K, G in L2 norm, and the decay in the element-wise difference between K, G and theoretical steady state K, G, respectively.

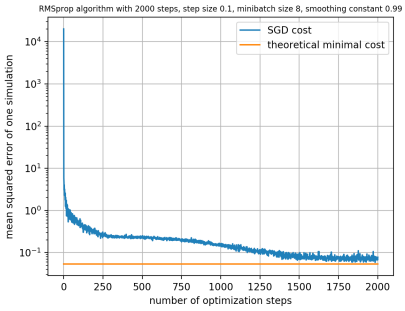

Figure 5: $n = 2000$: cost w.r.t n

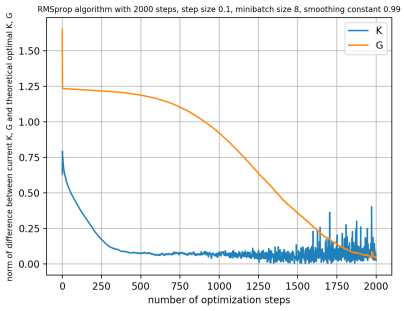

Figure 6: $n = 2000$: difference between K, G and theoretical steady state K, G w.r.t n

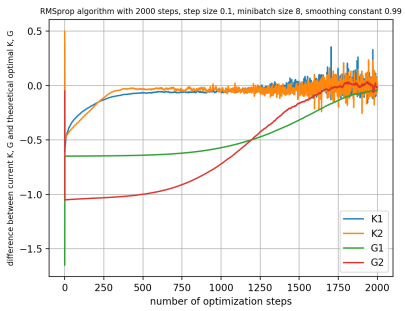

Figure 7: $n = 2000$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

From the graphs, it seems that overall the numerical result converges to the theoretical steady result. However, there are noticeable fluctuations in the difference between the current K and the theoretical steady state K as we approaches 2000 iterations. The cause for these fluctuations may be that as we approach the optimum, the learning rate is too large so the result oscillates around the optimum. To further narrow down the difference between the numerical result and the theoretical result, we increase the number of iterations and use a learning rate scheduler to decay the learning rate manually.
In particular, if we run 4000 iterations and use a learning rate scheduler to set $\alpha = 0.1\alpha$ after 2000 iterations and 3000 iterations, then the ultimate K is $[8.41 \times 10^{-2}, 1.70 \times 10^{-3}]^T$, and the ultimate G is $[6.45 \times 10^{-1}, -5.35 \times 10^{-2}]$. The differences between the ultimate K, G and the theoretical steady state K, G are 2.79×10^{-2} and 6.14×10^{-3}, respectively. The cost simulating with the ultimate K, G is 6.67×10^{-2}, and testing the ultimate K, G on 1000 simulations gives an expected cost of 6.97×10^{-2}.

Figures 8 to 10 show the cost decay, the decay in the difference between K, G and theoretical steady state K, G in L2 norm, and the decay in the element-wise difference between K, G and theoretical steady state K, G, respectively.

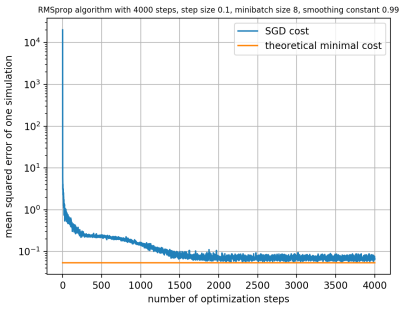

Figure 8: $n = 4000$: cost w.r.t n

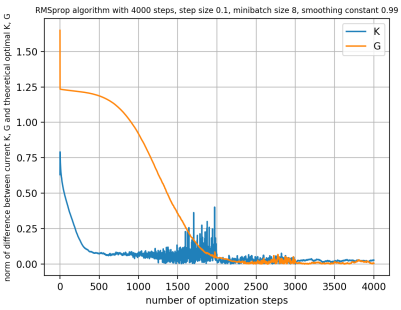

Figure 9: $n = 4000$: difference between K, G and theoretical steady state K, G w.r.t n

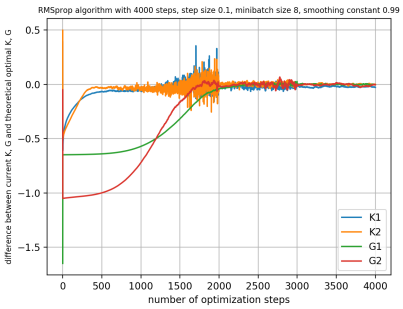

Figure 10: $n = 4000$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

Experiments with β While the above experiments show that with a sufficiently large number of iterations and a sufficiently small learning rate, the result of RMSprop can eventually converge to the theoretical solution, we can't help wondering if there is a way to accelerate this convergence. It turns out that the smoothing constant β plays a crucial role here. We observe that by decreasing β, we are able to achieve convergence within fewer iterations. Table 1 compares the number of iterations that takes to converge for different βs when $\alpha = 0.1$ and $M = 8$. The first row in the table shows the theoretical result. Figures 11 to 31 plot the decay in the cost, the decay in the difference between K, G and theoretical steady state K, G in L2 norm, and the decay in the element-wise difference between K, G and theoretical steady state K, G w.r.t. n for different βs. Figure 32 offers a more intuitive comparison of the performances of different βs. From the results it seems that smaller β can achieve faster convergence. This is especially surprising because when β equals to 0, the algorithm is simply clipping the gradient to 1. However, we also note that although the convergence is fast when $\beta = 0$, the final cost and testing cost in this case is relatively large, which may be a drawback of doing simple gradient clipping. On the other hand, the poor performance of $\beta = 0.99$ indicates that it is probably not a good idea to rely too heavily on the past gradients.

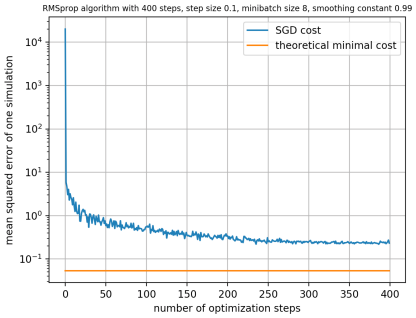

Figure 11: $\beta = 0.99$: cost w.r.t n

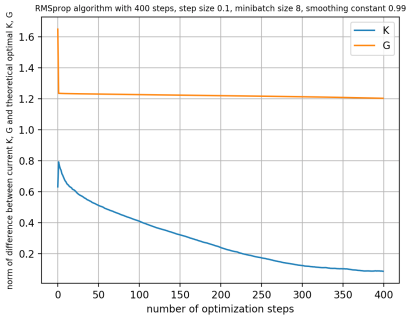

Figure 12: $\beta = 0.99$: difference between K, G and theoretical steady state K, G w.r.t n

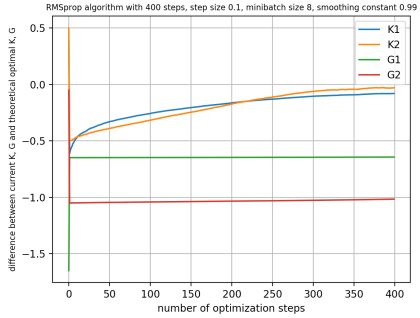

Figure 13: $\beta = 0.99$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

β	n	ultimate K	ultimate G^T	final K difference	final G difference	final cost	testing cost
-	-	$\begin{bmatrix} 1.12 \times 10^{-1} \\ 2.44 \times 10^{-3} \end{bmatrix}$	$\begin{bmatrix} 6.49 \times 10^{-1} \\ -4.89 \times 10^{-2} \end{bmatrix}$	–	–	–	5.37×10^{-2}
0.99	400	$\begin{bmatrix} 3.18 \times 10^{-2} \\ -2.89 \times 10^{-2} \end{bmatrix}$	$\begin{bmatrix} 5.04 \times 10^{-3} \\ -1.07 \times 10^{0} \end{bmatrix}$	8.60×10^{-2}	1.20×10^{0}	2.36×10^{-1}	2.35×10^{-1}
0.9	400	$\begin{bmatrix} 1.13 \times 10^{-1} \\ 1.91 \times 10^{-2} \end{bmatrix}$	$\begin{bmatrix} 6.11 \times 10^{-1} \\ -1.28 \times 10^{-1} \end{bmatrix}$	1.67×10^{-2}	8.81×10^{-2}	7.45×10^{-2}	7.01×10^{-2}
0.85	300	$\begin{bmatrix} 8.57 \times 10^{-2} \\ 5.60 \times 10^{-2} \end{bmatrix}$	$\begin{bmatrix} 6.30 \times 10^{-1} \\ -1.00 \times 10^{-1} \end{bmatrix}$	5.97×10^{-2}	5.45×10^{-2}	7.52×10^{-2}	6.95×10^{-2}
0.75	300	$\begin{bmatrix} 8.88 \times 10^{-2} \\ 6.72 \times 10^{-2} \end{bmatrix}$	$\begin{bmatrix} 6.40 \times 10^{-1} \\ -9.95 \times 10^{-2} \end{bmatrix}$	6.88×10^{-2}	5.13×10^{-2}	7.51×10^{-2}	6.99×10^{-2}
0.5	200	$\begin{bmatrix} 9.04 \times 10^{-2} \\ -1.33 \times 10^{-2} \end{bmatrix}$	$\begin{bmatrix} 6.37 \times 10^{-1} \\ -1.70 \times 10^{-2} \end{bmatrix}$	2.67×10^{-2}	3.40×10^{-2}	7.51×10^{-2}	6.96×10^{-2}
0.25	200	$\begin{bmatrix} 8.16 \times 10^{-2} \\ -1.30 \times 10^{-2} \end{bmatrix}$	$\begin{bmatrix} 6.19 \times 10^{-1} \\ 1.45 \times 10^{-2} \end{bmatrix}$	3.40×10^{-2}	6.99×10^{-2}	7.53×10^{-2}	6.97×10^{-2}
0.00	200	$\begin{bmatrix} 1.00 \times 10^{-1} \\ 1.00 \times 10^{-1} \end{bmatrix}$	$\begin{bmatrix} 6.00 \times 10^{-1} \\ -1.00 \times 10^{-1} \end{bmatrix}$	9.83×10^{-2}	7.05×10^{-2}	7.63×10^{-2}	7.26×10^{-2}

Table 1: Comparison of different β

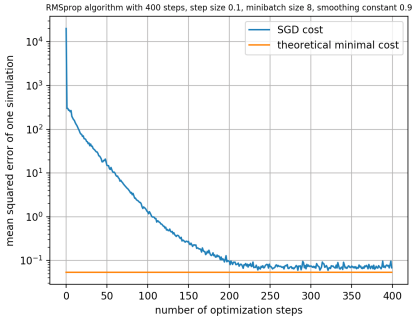

Figure 14: $\beta = 0.9$: cost w.r.t n

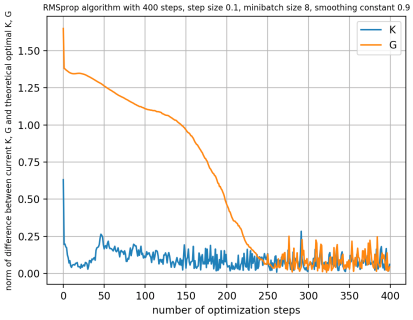

Figure 15: $\beta = 0.9$: difference between K, G and theoretical steady state K, G w.r.t n

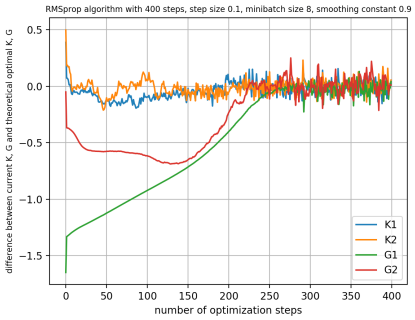

Figure 16: $\beta = 0.9$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

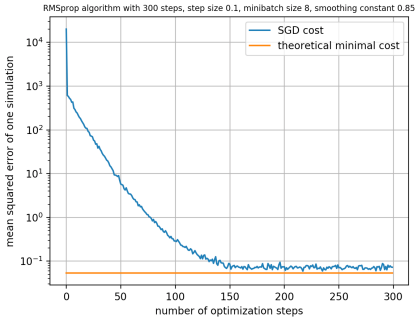

Figure 17: $\beta = 0.85$: cost w.r.t n

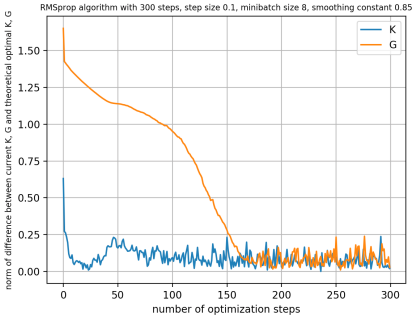

Figure 18: $\beta = 0.85$: difference between K, G and theoretical steady state K, G w.r.t n

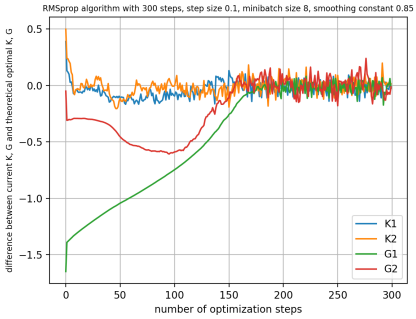

Figure 19: $\beta = 0.85$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

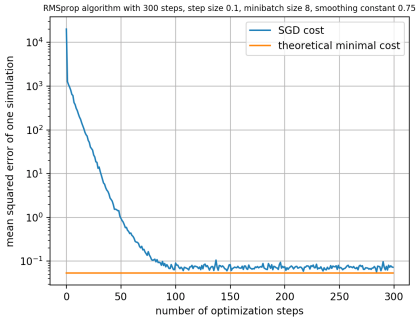

Figure 20: $\beta = 0.75$: cost w.r.t n

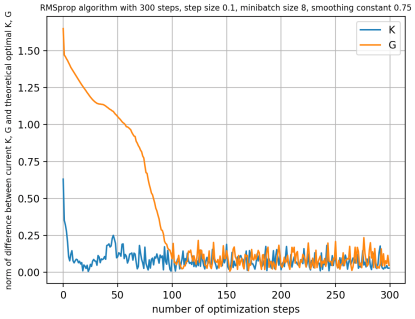

Figure 21: $\beta = 0.75$: difference between K, G and theoretical steady state K, G w.r.t n

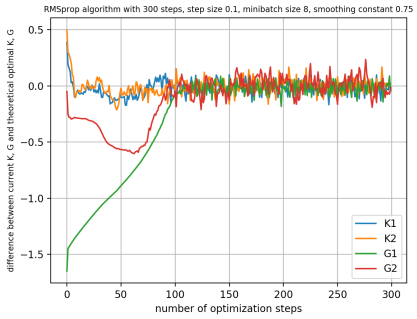

Figure 22: $\beta = 0.75$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

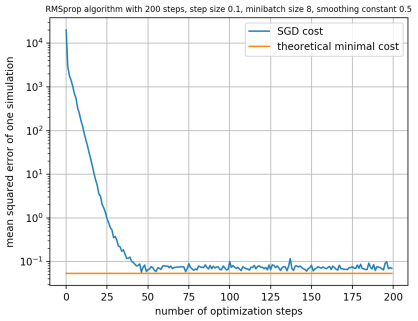

Figure 23: $\beta = 0.5$: cost w.r.t n

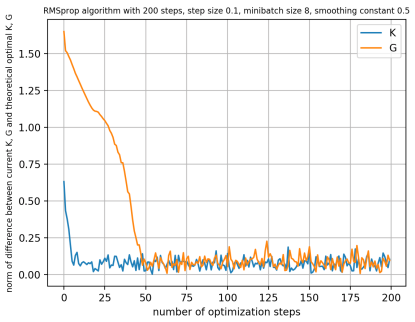

Figure 24: $\beta = 0.5$: difference between K, G and theoretical steady state K, G w.r.t n

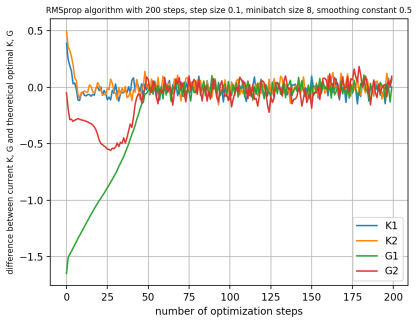

Figure 25: $\beta = 0.5$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

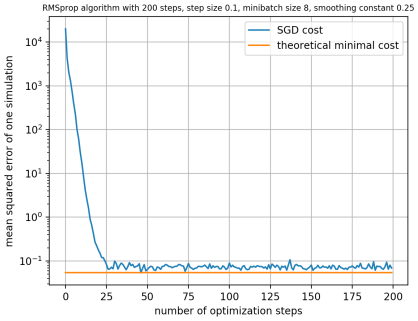

Figure 26: $\beta = 0.25$: cost w.r.t n

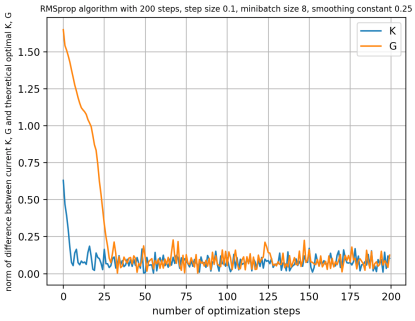

Figure 27: $\beta = 0.25$: difference between K, G and theoretical steady state K, G w.r.t n

Figure 28: $\beta = 0.25$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

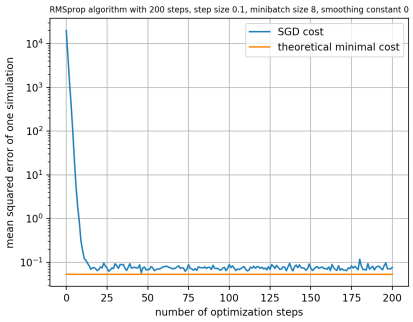

Figure 29: $\beta = 0.00$: cost w.r.t n

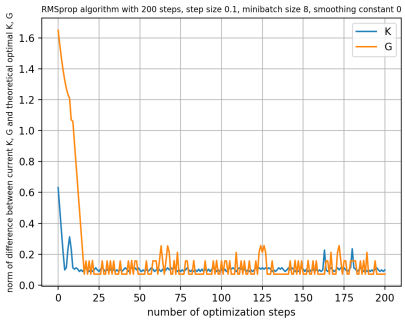

Figure 30: $\beta = 0.00$: difference between K, G and theoretical steady state K, G w.r.t n

Figure 31: $\beta = 0.00$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

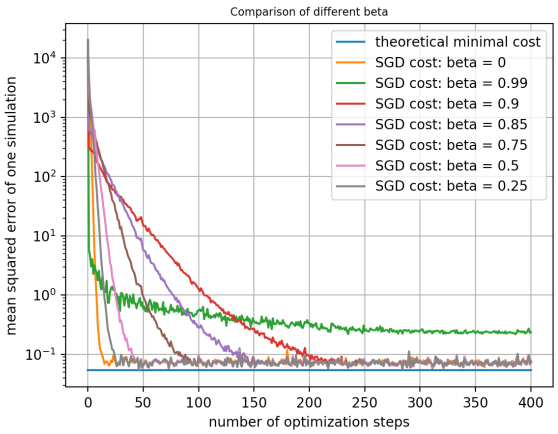

Figure 32: Comparison of different β

Experiments with M Bearing a similar idea to the Sample Average Approximation algorithm, in the presence of the noise, mini-batch gradient descent is generally preferred over purely stochastic gradient descent in machine learning, as it is less prone to the noisiness of the samples. In our problem, we also investigate the influence of mini-batch sizes on the performance of the RMSprop algorithm through a series of experiments.
In the following experiments, $n = 400$ and $\beta = 0.9$. A comparison of the results is demonstrated in table 2. The first row in the table shows the theoretical result. Figures 33 to 56 plot the decay in the cost, the decay in the difference between K, G and theoretical steady state K, G in L2 norm, and the decay in the element-wise difference between K, G and theoretical steady state K, G w.r.t. n for different Ms. Figure 57 offers a more intuitive comparison of the performances of different Ms. From the results we can see that larger mini-batches can effectively reduce the noise in the gradient, as they produce smoother learning curves. However, whether increasing mini-batch size can improve numerical accuracy is not very clear. By comparing the results, it seems that $M = 8$ is able to achieve both relatively good accuracy as well as data efficiency.

M	ultimate K	ultimate G	final K difference	final G difference	final cost	testing cost
-	$\begin{bmatrix} 1.12 \times 10^{-1} \\ 2.44 \times 10^{-3} \end{bmatrix}$	$[6.49 \times 10^{-1}, -4.89 \times 10^{-2}]$	–	–	–	5.37×10^{-2}
1	$\begin{bmatrix} 2.57 \times 10^{-1} \\ -1.34 \times 10^{-1} \end{bmatrix}$	$[6.01 \times 10^{-1}, 1.69 \times 10^{-3}]$	1.99×10^{-1}	6.94×10^{-2}	6.59×10^{-2}	8.12×10^{-2}
8	$\begin{bmatrix} 1.13 \times 10^{-1} \\ 1.91 \times 10^{-2} \end{bmatrix}$	$[6.11 \times 10^{-1}, -1.28 \times 10^{-1}]$	1.67×10^{-2}	8.81×10^{-2}	7.45×10^{-2}	7.01×10^{-2}
16	$\begin{bmatrix} 8.80 \times 10^{-2} \\ 2.72 \times 10^{-2} \end{bmatrix}$	$[6.58 \times 10^{-1}, 8.83 \times 10^{-3}]$	3.44×10^{-2}	5.84×10^{-2}	6.80×10^{-2}	7.15×10^{-2}
32	$\begin{bmatrix} 9.23 \times 10^{-2} \\ 2.69 \times 10^{-2} \end{bmatrix}$	$[6.19 \times 10^{-1}, 2.01 \times 10^{-2}]$	3.14×10^{-2}	7.52×10^{-2}	6.68×10^{-2}	7.05×10^{-2}
64	$\begin{bmatrix} 9.92 \times 10^{-2} \\ 4.59 \times 10^{-2} \end{bmatrix}$	$[6.55 \times 10^{-1}, -7.56 \times 10^{-2}]$	4.53×10^{-2}	2.75×10^{-2}	6.80×10^{-2}	6.98×10^{-2}
128	$\begin{bmatrix} 3.48 \times 10^{-2} \\ 3.71 \times 10^{-3} \end{bmatrix}$	$[5.20 \times 10^{-1}, -9.11 \times 10^{-2}]$	7.72×10^{-2}	1.36×10^{-1}	7.39×10^{-2}	7.30×10^{-2}
256	$\begin{bmatrix} 2.10 \times 10^{-1} \\ -4.35 \times 10^{-3} \end{bmatrix}$	$[7.19 \times 10^{-1}, 3.44 \times 10^{-2}]$	9.83×10^{-2}	1.09×10^{-1}	7.73×10^{-2}	7.71×10^{-2}
512	$\begin{bmatrix} 2.74 \times 10^{-2} \\ -1.80 \times 10^{-2} \end{bmatrix}$	$[5.73 \times 10^{-1}, -6.26 \times 10^{-2}]$	8.70×10^{-2}	7.66×10^{-2}	7.33×10^{-2}	7.26×10^{-2}

Table 2: Comparison of different mini-batch sizes

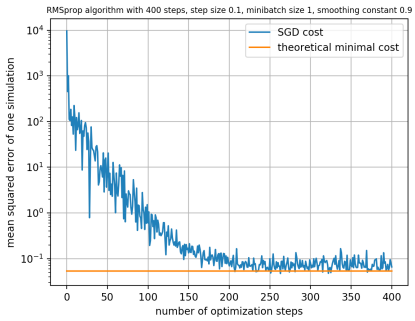

Figure 33: $M = 1$: cost w.r.t n

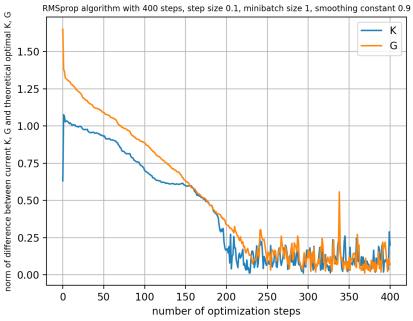

Figure 34: $M = 1$: difference between K, G and theoretical steady state K, G w.r.t n

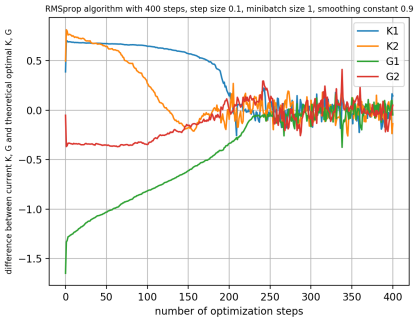

Figure 35: $M = 1$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

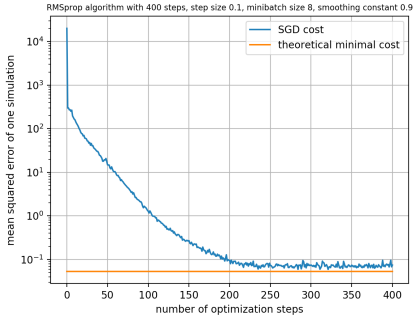

Figure 36: $M = 8$: cost w.r.t n

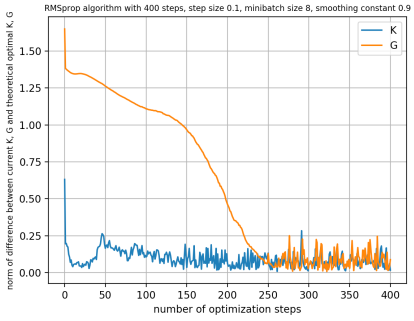

Figure 37: $M = 8$: difference between K, G and theoretical steady state K, G w.r.t n

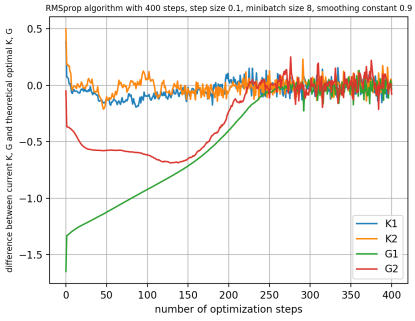

Figure 38: $M = 8$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

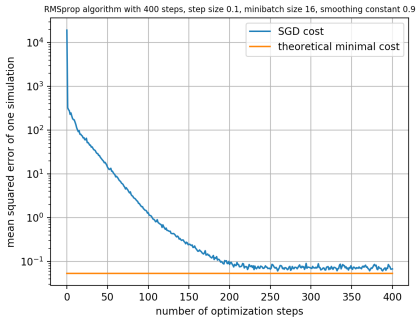

Figure 39: $M = 16$: cost w.r.t n

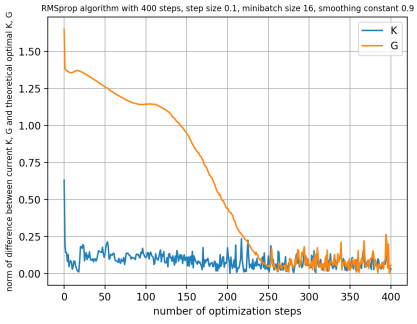

Figure 40: $M = 16$: difference between K, G and theoretical steady state K, G w.r.t n

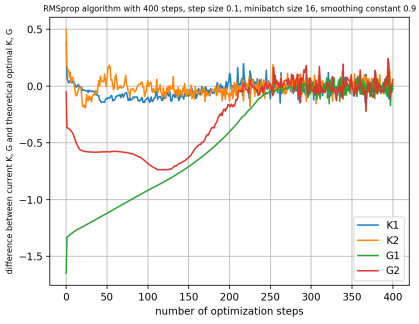

Figure 41: $M = 16$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

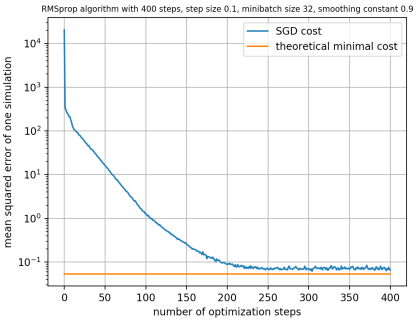

Figure 42: $M = 32$: cost w.r.t n

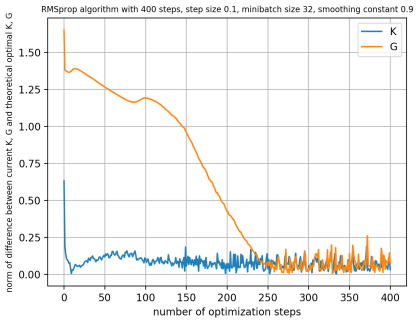

Figure 43: $M = 32$: difference between K, G and theoretical steady state K, G w.r.t n

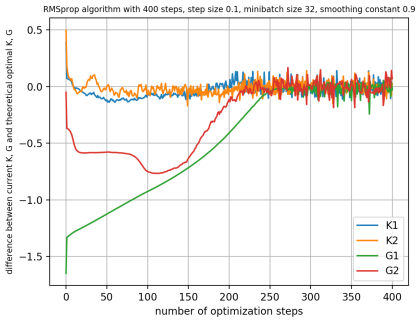

Figure 44: $M = 32$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

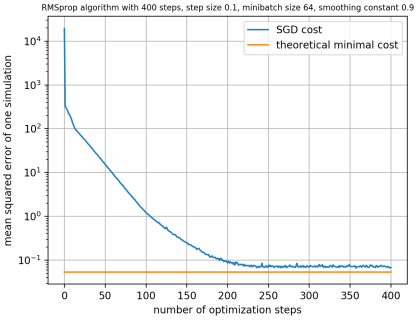

Figure 45: $M = 64$: cost w.r.t n

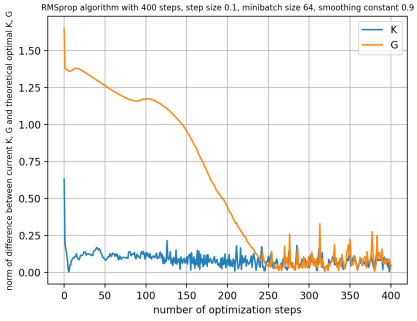

Figure 46: $M = 64$: difference between K, G and theoretical steady state K, G w.r.t n

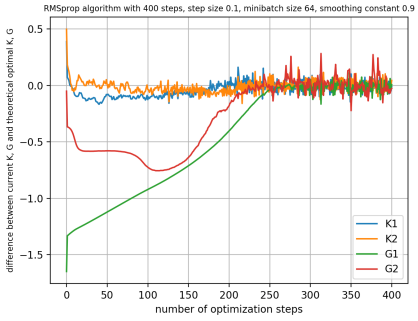

Figure 47: $M = 64$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

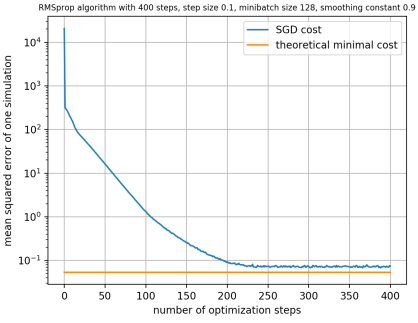

Figure 48: $M = 128$: cost w.r.t n

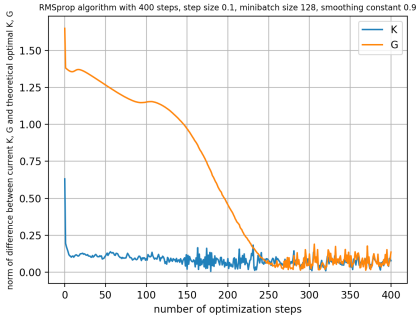

Figure 49: $M = 128$: difference between K, G and theoretical steady state K, G w.r.t n

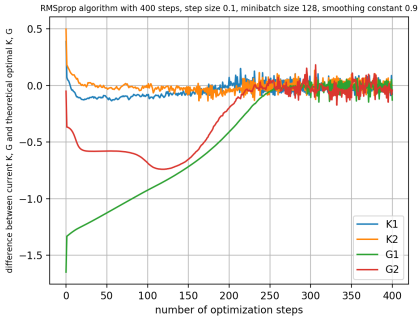

Figure 50: $M = 128$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

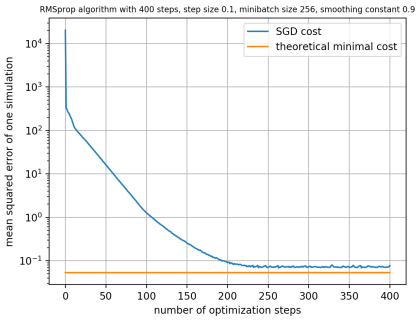

Figure 51: $M = 256$: cost w.r.t n

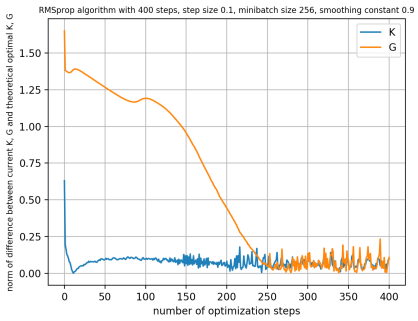

Figure 52: $M = 256$: difference between K, G and theoretical steady state K, G w.r.t n

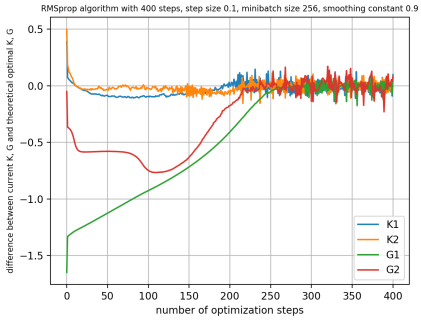

Figure 53: $M = 256$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

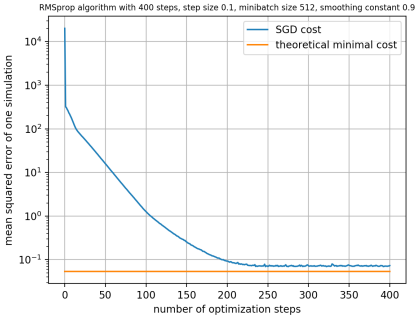

Figure 54: $M = 512$: cost w.r.t n

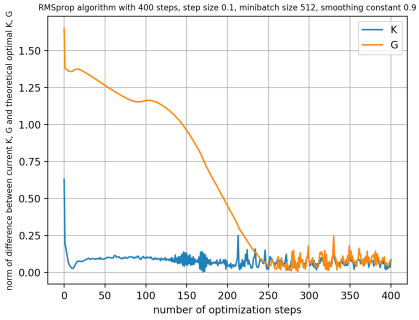

Figure 55: $M = 512$: difference between K, G and theoretical steady state K, G w.r.t n

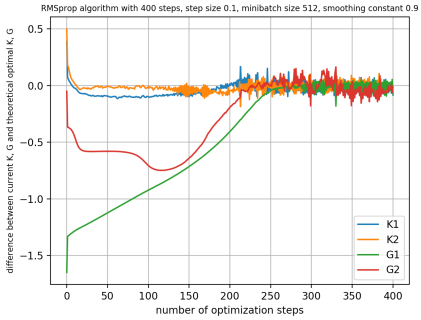

Figure 56: $M = 512$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

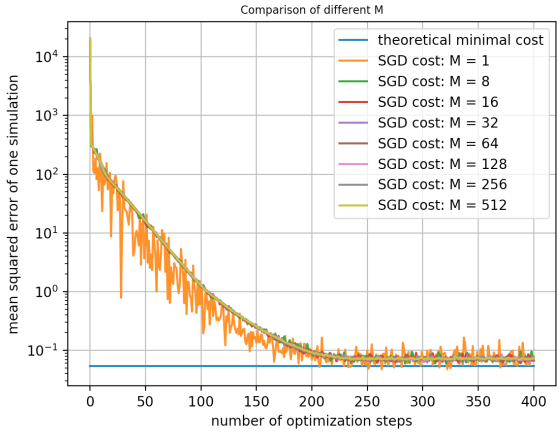

Figure 57: Comparison of different M

Experiments with Polyak averaging As mentioned in section 5.1, because of the presence of noise, instead of really converging to the optimum, it is more likely that after getting close to the optimum, the result of stochastic gradient descent oscillates around it. Inspired by the idea proposed in [6], instead of making the values of K and G at the last iteration as our final result, we keep track of the values of K, G in the last n_0 iterations, and define the final K as $\frac{1}{n_0}\sum_{i=n-n_0+1}^{n} K_i$, and the final G as $\frac{1}{n_0}\sum_{i=n-n_0+1}^{n} G_i$, where n_0 is an additional hyper-parameter to choose. Intuitively, this can help reduce the effect of noise and make the final result closer to the optimum. Table 3 compares the results of different n and n_0 with $\beta = 0.9$, $\alpha = 0.1$, and $M = 8$. The first row in the table shows the theoretical result. We can see that by averaging, the difference between the ultimate K, G from stochastic gradient descent and the theoretical solution is effectively narrowed. A closer look at the oscillation in last 200 iterations of gradient descent when n is 400, 500, and 1000 is presented in figures 58 to 66.

n	n_0	ultimate K	ultimate G^T	final K difference	final G difference	final cost	testing cost
-	-	$\begin{bmatrix} 1.12 \times 10^{-1} \\ 2.44 \times 10^{-3} \end{bmatrix}$	$\begin{bmatrix} 6.49 \times 10^{-1} \\ -4.89 \times 10^{-2} \end{bmatrix}$	–	–	–	5.37×10^{-2}
400	1	$\begin{bmatrix} 1.13 \times 10^{-1} \\ 1.91 \times 10^{-2} \end{bmatrix}$	$\begin{bmatrix} 6.11 \times 10^{-1} \\ -1.28 \times 10^{-1} \end{bmatrix}$	1.67×10^{-2}	8.81×10^{-2}	7.45×10^{-2}	7.01×10^{-2}
400	10	$\begin{bmatrix} 1.08 \times 10^{-1} \\ 1.65 \times 10^{-2} \end{bmatrix}$	$\begin{bmatrix} 6.39 \times 10^{-1} \\ -2.93 \times 10^{-2} \end{bmatrix}$	1.47×10^{-2}	2.18×10^{-2}	7.63×10^{-2}	7.03×10^{-2}
400	100	$\begin{bmatrix} 1.05 \times 10^{-1} \\ 8.66 \times 10^{-3} \end{bmatrix}$	$\begin{bmatrix} 6.22 \times 10^{-1} \\ -3.89 \times 10^{-2} \end{bmatrix}$	9.17×10^{-3}	2.83×10^{-2}	7.64×10^{-2}	7.03×10^{-2}
500	1	$\begin{bmatrix} 1.35 \times 10^{-1} \\ 1.40 \times 10^{-2} \end{bmatrix}$	$\begin{bmatrix} 6.12 \times 10^{-1} \\ -1.07 \times 10^{-1} \end{bmatrix}$	2.59×10^{-2}	6.87×10^{-2}	5.86×10^{-2}	7.12×10^{-2}
500	100	$\begin{bmatrix} 1.06 \times 10^{-1} \\ 8.00 \times 10^{-3} \end{bmatrix}$	$\begin{bmatrix} 6.31 \times 10^{-1} \\ -5.54 \times 10^{-2} \end{bmatrix}$	7.80×10^{-3}	1.87×10^{-2}	5.71×10^{-2}	7.05×10^{-2}
1000	1	$\begin{bmatrix} 6.14 \times 10^{-2} \\ 4.40 \times 10^{-2} \end{bmatrix}$	$\begin{bmatrix} 6.00 \times 10^{-1} \\ 4.85 \times 10^{-2} \end{bmatrix}$	6.55×10^{-2}	1.09×10^{-1}	7.07×10^{-2}	7.10×10^{-2}
1000	200	$\begin{bmatrix} 1.06 \times 10^{-1} \\ 3.45 \times 10^{-3} \end{bmatrix}$	$\begin{bmatrix} 6.27 \times 10^{-1} \\ -5.26 \times 10^{-2} \end{bmatrix}$	6.50×10^{-3}	2.16×10^{-2}	6.93×10^{-2}	7.00×10^{-2}

Table 3: Comparison of different n and n_0

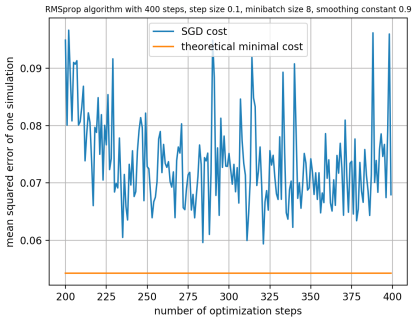

Figure 58: $n = 400$: cost w.r.t n

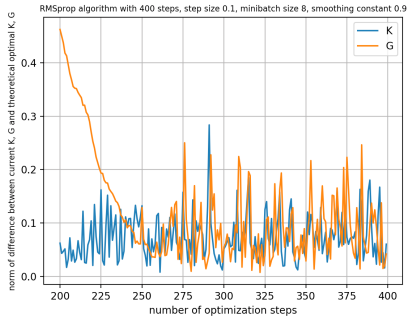

Figure 59: $n = 400$: difference between K, G and theoretical steady state K, G w.r.t n

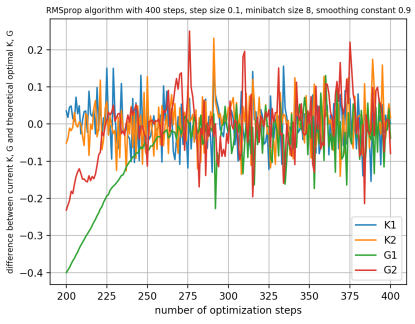

Figure 60: $n = 400$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

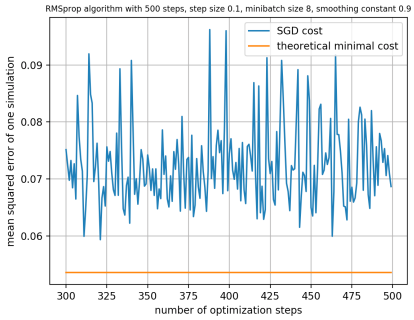

Figure 61: $n = 500$: cost w.r.t n

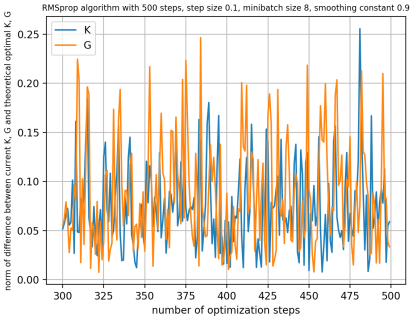

Figure 62: $n = 500$: difference between K, G and theoretical steady state K, G w.r.t n

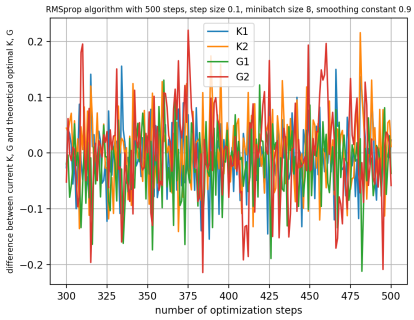

Figure 63: $n = 500$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

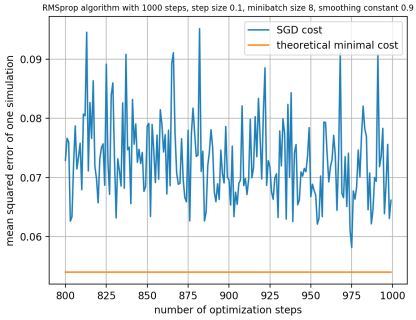

Figure 64: $n = 1000$: cost w.r.t n

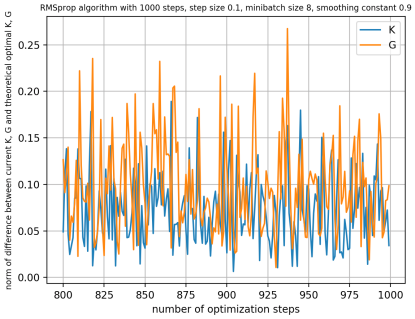

Figure 65: $n = 1000$: difference between K, G and theoretical steady state K, G w.r.t n

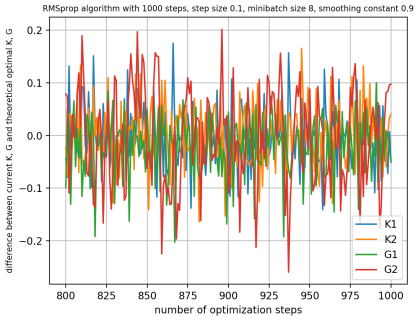

Figure 66: $n = 1000$: difference between K, G and theoretical steady state K, G w.r.t n (element-wise)

3 Glucose minimal model and insulin control

To design an artificial pancreas for type I diabetes, we employ the well-known minimal model (see, for instance, [7] for reference) to model the glucose kinetics of Type 1 Diabetes Mellitus (T1DM) patients. In our research project, we would apply the minimal model to in-silico subjects. Without any knowledge on meal intake, we estimate the meal intake amount by observing the glucose level, and use a filter to minimize the estimation error between minimal model and glucose measurements. The control of the glucose is through plasma insulin injection. The proper control would be determined by stochastic gradient descent with respect to a cost function. The cost function contains the control of glucose level into normal region, the cost the insulin, and the mean square error between true state value of the in-silico subject and the model estimation.

3.1 Parameter and Variable Definition

First, we define the variables used in our Glucose Minimal Model with meal digestion:

- $G :=$ glucose level, unit: mg/dl
- $X :=$ remote insulin level, unit: mU/l
- $I :=$ insulin level, unit: mU/l
- $Ra :=$ glucose appearance rate, unit: mg/kg/min

- $x := [G, X, I, Ra]$ state variables of model estimation
- $\tilde{x} := [\tilde{G}, \tilde{X}, \tilde{I}, \tilde{Ra}]$ true state value of in-silico person
- $D :=$ meal disturbance function, unit: mg/kg/min
- $v :=$ insulin control injection function, unit: mU/l
- $t_k :=$ meal intake time, unit: min
- $q_k :=$ meal intake appearance rate, unit: mg/min
- $T :=$ total simulation time, unit:min
- $t_0 :=$ a control period, unit: min
- $Z_n :=$ glucose measurement at time t_n, unit: mg/dl
- $K :=$ filter exerted on state variables
- $H :=$ control gain matrix exerted on the whole system

The biology parameters and initial conditions of state variables used in the model are introduced here:

Parameter	Value	Dimension	Description
p_1	3.6	min^{-1}	Parameter in Glucose Minimal Model
p_2	0.022	min^{-1}	Parameter in Glucose Minimal Model
p_3	1.7e-5	$\text{min}^{-1}\text{mU}/l$	Parameter in Glucose Minimal Model
c_1	0.25	min^{-1}	Parameter in Insulin Model
c_2	0.2	min^{-1}	Parameter in Insulin Model
G_b	125	mg/dl	Basal glucose level
I_b	0	mU/l	Basal insulin level
G_0	130	mg/dl	Initial glucose level
X_0	0	mU/l	Initial remote insulin level
I_0	0.38	mU/l	Initial Insulin level
Ra_0	5	mg/kg/min	Initial glucose appearance rate

Table 4: Minimal Glucose Dynamics Model Parameters

3.2 Model Description

During each control period, let's denote the starting time of this period as time t_n, denote the state variable x at time t_n as x_n. Our model estimation of the state variable x_{n+1} has two parts: ODE model evolution and filtering.

3.2.1 ODE model evolution

Let x_n^- be an a prior estimate for x_n. We first apply our ODE model to get a-prior estimate x_n^-, which joins glucose minimal model and meal digestion. The minimal model of glucose kinetics has the form of these two differential equations:

$$\frac{dG(t)}{dt} = -(p_1 + X(t))G(t) + p_1 G_b + Ra(t) \tag{12}$$

$$\frac{dX(t)}{dt} = -p_2 X(t) + p_3(I(t) - I_b) \tag{13}$$

Glucose rate of appearance Ra (mg/kg/min) given the mean disturbance D (mg/kg/min) introduced by Patek et al.(2007) in [8] is:

$$\frac{dRa(t)}{dt} = -\frac{1}{\tau}(Ra(t) - D(t)) \tag{14}$$

$$D(t) = \sum_{k=1}^{M} q_k \delta(t - t_k) \tag{15}$$

where $\delta(t)$ is an approximation of a Dirac Delta function and q_k is the peak of the signal. $t_1, t_2, ..., t_M$ is the time when food intake happens. τ is a parameter associated with digestion efficiency. With continuous intravenous (IV) insulin infusion, Chervoneva et al use the one-compartment model to describe the kinetics of insulin [9]

$$\frac{dI(t)}{dt} = -c_1 I(t) + c_2 v(t), \tag{16}$$

where c_1 [min^{-1}] is the insulin first order disappearance rate, c_2 [l^{-1}] is the reciprocal of the volumn of the insulin distribution space, v_n [mU/min] is the IV insulin infusion rate. The control $v(t)$ is designed to be uniform during a control period t_0, namely $v(t) = v_n$ for $t \in [t_n, t_{n+1})$. And the constant insulin control v_n during time $[t_n, t_{n+1})$ given a constant control gain matrix $H = [h_1, h_2, h_3, h_4]$ is defined as follows:

$$v_n = h_1(G_n - G_b) + h_2 X_n + h_3(I_n - I_b) + h_4 Ra_n.$$

To summarize, if we let F be the update of the evolution of ODE system from time t_n to t_{n+1}, then we have

$$x_{n+1}^- = F(x_n).$$

3.2.2 Filtering

The next step is to add a filter to the prior estimation based on the glucose measurement at time t_n.
Let's denote the filter K as $[K_1, K_2, K_3, K_4]$. The update rule is:

$$x_{n+1}(i) = \begin{cases} x_{n+1}^-(i) + K(i)(Z_n - G_n), & \text{if } x_{n+1}^-(i) + K(i)(Z_n - G_n) \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

where $x_n(i), K(i)$ refers to the ith term in vector x_n, K, $i \in \{1, 2, 3, 4\}$. Z_n are noisy measurements of in-silico glucose level given by

$$Z_n = \tilde{G}_n + w_n,$$

where w_n is a Gaussian random variable with mean zero and variance σ^2_{obs}. $\tilde{G}_n$ is the real glucose level of the in-silico subject. The filter K is applied to minimize the error between the measurement and the model estimation.

3.3 In-silico Subject Design

We use the ODE evolution model for in-silico simulation. However, as the model cannot capture the full dynamics of the system for sure, we assume that the true dynamic is a random process. A Gaussian process noise would be added after each ODE evolution to "correct" the model. Denote true state variable of the in-silico subject at time t_n as $\tilde{x}_n$. With same ODE evolution rule, the prior state variable $\tilde{x}_{n+1}^-$ can be determined in the same way as the model a prior state variable x_{n+1}^-, given the true state variable $\tilde{x}_n$ at time n of the in-silico subject,

$$\tilde{x}_{n+1}^- = F(\tilde{x}_n) \tag{17}$$

Then, we consider the true state variable $\tilde{x}_{n+1}$ to be

$$\tilde{x}_{n+1} = \tilde{x}_{n+1}^- + W_{n+1} \tag{18}$$

where W_{n+1} is a vector of Gaussian random variable with mean zero. Since for each element in vector x, the process noises on $G^-_{n+1}, X^-_{n+1}, I^-_{n+1}, Ra^-_{n+1}$ should be independent. Therefore,

$$W_{n+1} \sim N(0, \Sigma),$$

where Σ is a diagonal matrix such that $\Sigma = \text{diag}(\sigma_1^2, \sigma_2^2, \sigma_3^2, \sigma_4^2)$.

3.4 Finding the optimal control parameters

3.4.1 Cost function

The essential part of the cost function is the penalty for the glucose being outside the euglycemic zone $[l, u]$. Since hypoglycemia is known to be more dangerous than hyperglycemia, a larger penalty is given when the glucose level becomes lower than the lower bound l. Mathematically, we define this part of the cost function as

$$J_1 = \sum_n \hat{G}_n$$

where

$$\hat{G}_n = \begin{cases} G_n - u, & \text{if } G_n > u \\ 0, & \text{if } l \leq G_n \leq u \\ l - G_n, & \text{if } G_n < l \end{cases}$$

In our experiments, we adopt the euglycemic zone used in [10], which is (80 mg/dl-140 mg/dl).
Since the accuracy of the model estimation of the states can also influence the effectiveness of our control, an estimation cost is added to our cost function. This is defined as

$$J_2 = \sum_n (x_n - \tilde{x}_n)^T \text{diag}(\lambda_1, \lambda_2, \lambda_3, \lambda_4)(x - \tilde{x}_n)$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are scaling constants. We observe that the magnitude of I is about 1/10 the magnitude of other state variables, so we set $\lambda_1 = \lambda_2 = \lambda_4 = 1$, $\lambda_3 = 10$.
As too much insulin can also be hazardous to humans, we added a control cost to our cost function, which is defined as

$$J_3 = \sum_n v_n^2$$

Combining J_1, J_2, and J_3, we define the total cost as

$$J = \mu_1 J_1 + \mu_2 J_2 + \mu_3 J_3 \tag{19}$$

where μ_1, μ_2, μ_3 are the weights for each part. To mark clearly that the primary objective of our model is to control the glucose level, we can set $\mu_1 >> \mu_2, \mu_3$.

3.4.2 Experimental results

We then turn to machine learning optimization algorithms to determine the optimal filter and control parameters $(K_1, \ldots, K_4, H_1, \ldots, H4)$ w.r.t. cost function (19). Since analytical expressions for the gradients are very hard to obtain, we approximate the gradients using first-order finite difference method. Algorithm 2 provides an overview of our whole program.

Algorithm 2 Glucose minimal model and insulin control

Initialize K, H, $\phi_0^K = \phi_0^H = 0$
for $i = 1, 2, \ldots$ **do**
  Initialize $\tilde{x}_0$, x_0
  for $n = 0, 1, 2, \ldots, N$ **do**
    Simulate in-silico subject:
      $\tilde{x}_{n+1}^- = F_H(\tilde{x}_n)$
      $\tilde{x}_{n+1} = \tilde{x}_{n+1}^- + W_{n+1}$
    Observe $Z_{n+1} = \tilde{G}_{n+1} + w_{n+1}$
    Model estimation:
      $x_{n+1}^- = F_H(x_n)$
      $x_{n+1} = x_{n+1}^- + K(Z_{n+1} - G_{n+1}^-)$
  Compute the cost at the current iteration $f(K, H)$
  Compute the gradients $g_K = \frac{\partial f(K,H)}{\partial K}$, $g_H = \frac{\partial f(K,H)}{\partial H}$
  Update ϕ_i^K, ϕ_i^H: $\phi_i^K = \beta\phi_{i-1}^K + (1-\beta)g_K^2$, $\phi_i^H = \beta\phi_{i-1}^H + (1-\beta)g_H^2$ (element-wise square)
  Update K,H: $K = K - \alpha\frac{g_K}{\sqrt{\phi_i^K + \epsilon}}$, $H = H - \alpha\frac{g_H}{\sqrt{\phi_i^H + \epsilon}}$

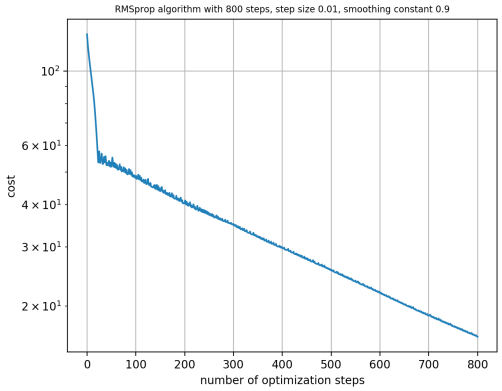

Figure 67: Stage 1: cost w.r.t n

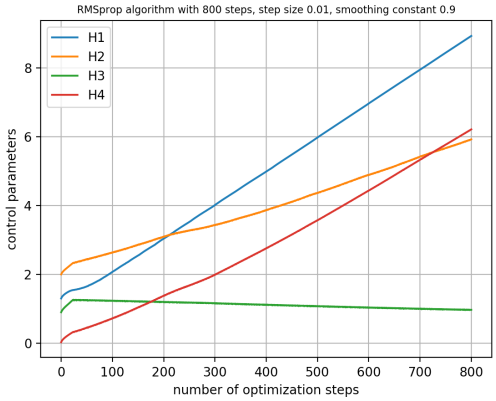

Figure 68: Stage 1: Control parameters w.r.t n

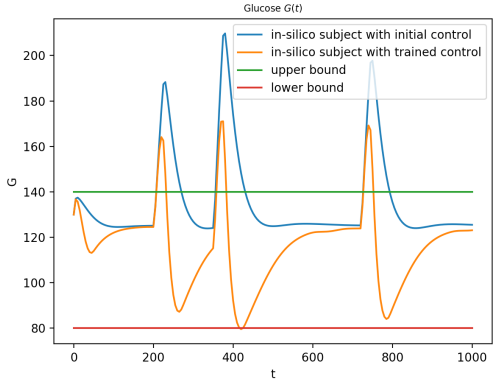

Figure 69: Stage 1: Glucose level with initial and trained controls

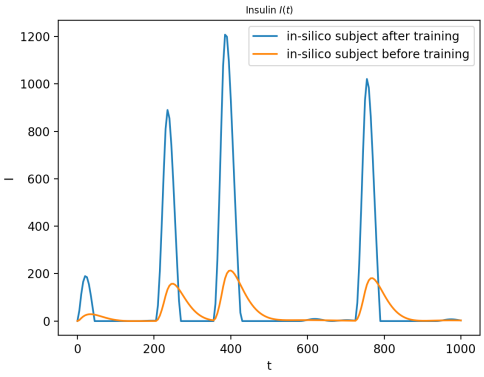

Figure 70: Stage 1: Insulin level with initial and trained controls

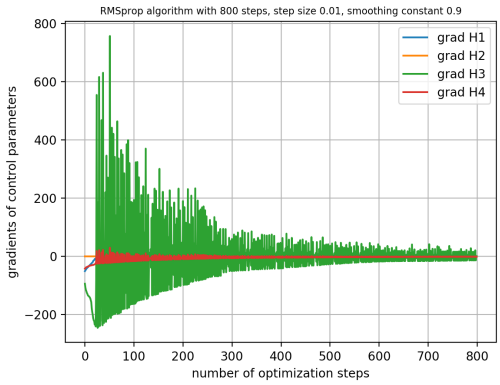

Figure 71: Stage 1: Gradients of control w.r.t n

Stage 1 We first test our model on the simplest case with $\sigma_1 = \sigma_2 = \sigma_3 = \sigma_4 = \sigma_{obs} = 0$, $\mu_1 = 1$, $\mu_2 = \mu_3 = 0$. We also make the information on meal intake available to the model. So the filter is ineffective, and there is no limit on the amount of insulin used. We used RMSprop algorithm with learning rate 0.01 and smoothing constant 0.9. The results after 800 gradient descent iterations are shown in figures 67 to 71.

As there is no noise in the process and observation, the fluctuation in the cost appears to be very suspicious. So we analyzed the gradients of the control parameters. After examining the gradients near the point where the fluctuation begins, we think there is a very steep valley with the (local) minimum sitting at the bottom of it, so our approach to the minimum is zigzag unless each step is taken to be very small. But since the cost is decreasing and the fluctuation is diminishing, we are still on the right track of converging to the minimum, so there is no need to reduce learning rate and slow down learning. Since it is very likely that there is more than one such valley and each of them will possibly cause fluctuations, we expect optimization in this geography to be hard.

Stage 2 We add a small noise and take the estimation cost into account. The meal intake information, the intake time the amount is unknown to the model. Therefore, filter is effective in stage to improve the model estimation based on the difference between the model glucose level and glucose measurements from the in-silico subject. We set $\sigma_1 = \frac{1}{200}G_0, \sigma_2 = \frac{1}{200}X_0, \sigma_3 = \frac{1}{200}I_0, \sigma_4 = \frac{1}{200}Ra_0, \sigma_{obs} = \frac{1}{200}G_0$, and $\mu_1 = 1000, \mu_2 = 1, \mu_3 = 0$. Using RMSprop algorithm with learning rate 0.01, smoothing constant 0.9, after 7000 iterations, we can effectively control the glucose level of the in-silico subject and make an accurate estimation. The results are shown in figures 72 to 78.

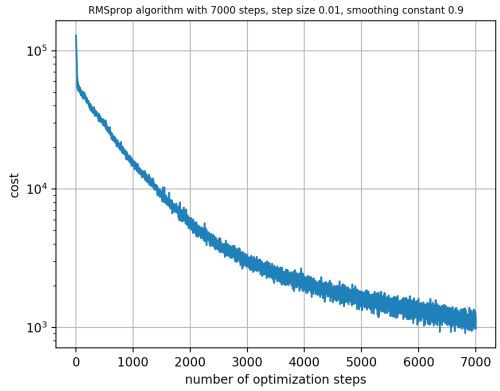

Figure 72: Stage 2: cost w.r.t n

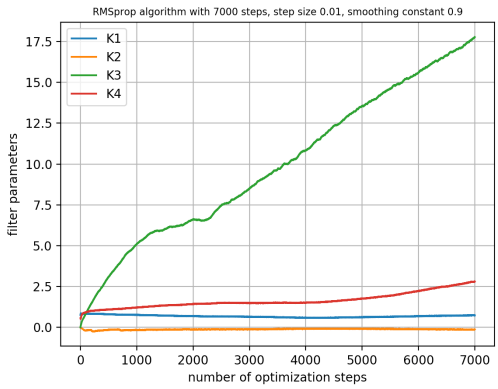

Figure 73: Stage 2: Filter parameters w.r.t n

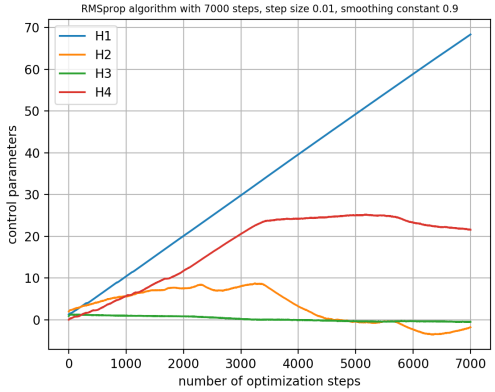

Figure 74: Stage 2: Control parameters w.r.t n

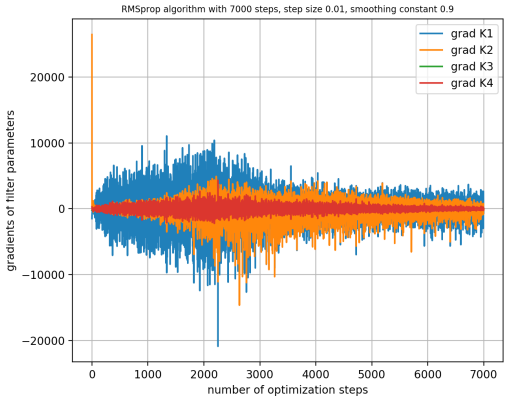

Figure 75: Stage 2: Gradients of filter w.r.t n

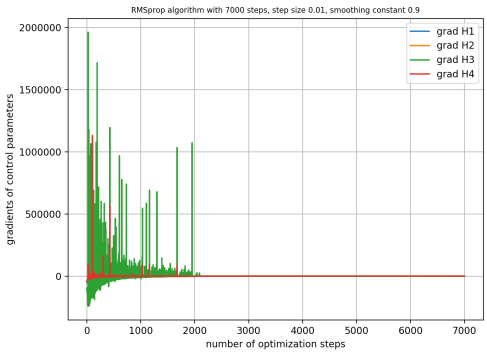

Figure 76: Stage 2: Gradients of control w.r.t n

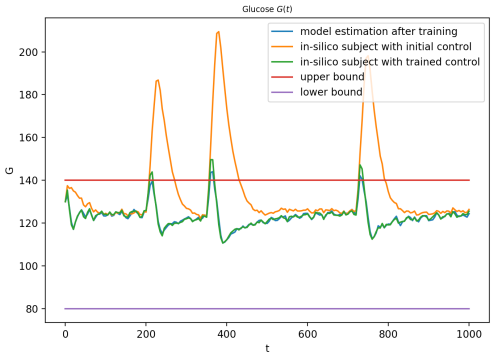

Figure 77: Glucose level with initial and trained controls

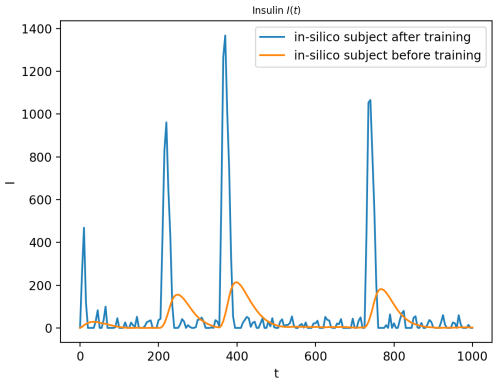

Figure 78: Insulin level with initial and trained controls

Stage 3 Even though the model estimation of glucose seems to be accurate and the control of glucose seems to be successful, we immediately notice that the amount of insulin it takes to control the glucose level is too large to be realistic. Therefore, it is necessary to consider the control cost. We set $\mu_3 = 1$ and keep $\sigma_1, \sigma_2, \sigma_3, \sigma_4, \mu_1, \mu_2$ the same as before. Using RMSprop algorithm with learning rate 0.01, smoothing constant 0.9, after 6000 iterations, we

can roughly control the glucose level with a reasonable amount of insulin. The results are shown in figures 79 to 83.

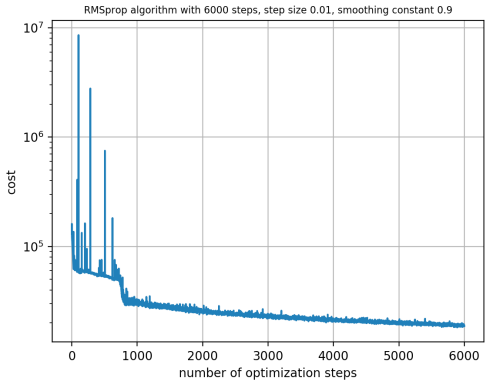

Figure 79: Stage 3: cost w.r.t n

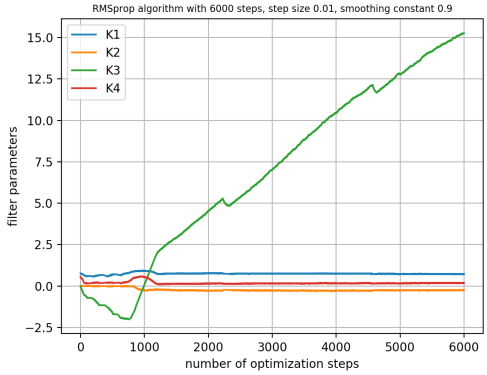

Figure 80: Stage 3: Filter parameters w.r.t n

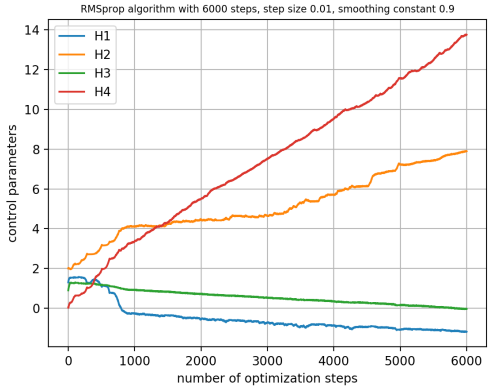

Figure 81: Stage 3: Control parameters w.r.t n

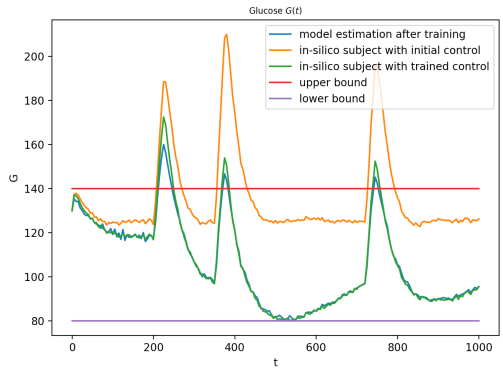

Figure 82: Glucose level with initial and trained controls

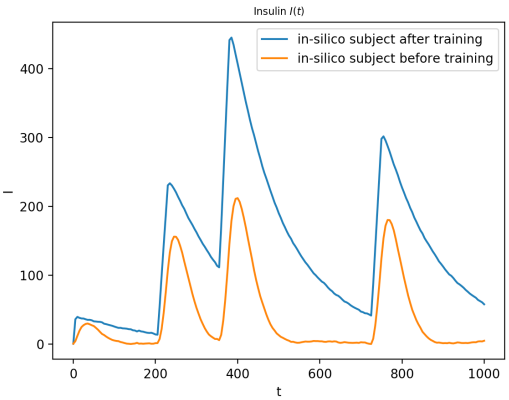

Figure 83: Insulin level with initial and trained controls

4 Discussion

From this project, we learned basic knowledge about control theory, including the Kalman filter and the linear-quadratic-Gaussian control. We applied a widely used optimization algorithm from machine learning, RMSprop, to a 2-dimensional linear quadratic gaussian problem. We experimented with different numbers of gradient descent iterations, different smoothing constants, different mini-batch sizes, as well as Polyak averaging. We derived the theoretical solution for this problem and observed good agreements between our numerical results and the theoretical results. We showed from this simple example that it is possible to find the optimal filter and control parameters from the RMSprop algorithm. Then we proposed a model for an artificial pancreas for type 1 diabetes. The model consists of an ODE system describing the glucose dynamics of the patient, a filter for estimating the glucose level, and a linear control to regulate the glucose level. The filter and control parameters are determined by minimizing a cost function using the RMSprop algorithm. Due to time limit, we have not yet achieved complete convergence of the filter and control parameters, but we can already confirm the feasibility of this model by our current results. A variety of possibilities are left for us to explore in the future. For example, we can improve the accuracy of gradient computation by using higher order finite difference approximations, or try to derive formulas to calculate the gradients. Also, the mini-batch method could be very helpful as we deal with noises with larger variances. A different combination of the learning rate and the smoothing constant in the algorithm may also help improve the result. For now, we only considered glucose control and state variable filter. The parameters used in the model is same as those of in-silico subject. However, it is not the case in general. Different patients should have different parameters. How to "customize" those parameters is also an interesting topic for further investigation.

5 Acknowledgements

We sincerely thank Professor Jonathan Goodman for his guidance and supervision. We thank NYU Courant Summer Undergraduate Research Program for supporting our project.

References

[1] Maria Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. Feb 2004.

[2] Stephen Boyd.

[3] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[6] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[7] Tomas Ludwig and Ivan Ottinger. Identification of t1dm minimal model using non-consistent data from ivgtt. *Journal of Electrical Systems and Information Technology*, 1(2):144–149, 2014.

[8] Stephen D. Patek, Marc D. Breton, Yuanda Chen, Chad Solomon, and Boris Kovatchev. Linear quadratic gaussian-based closed-loop control of type 1 diabetes. *Journal of Diabetes Science and Technology*, 1(6):834–841, 2007.

[9] Inna Chervoneva, Boris Freydin, Brian Hipszer, Tatiyana V. Apanasovich, and Jeffrey I. Joseph. Estimation of nonlinear differential equation model for glucose–insulin dynamics in type i diabetic patients using generalized smoothing. *The Annals of Applied Statistics*, 8(2):886–904, 2014.

[10] K. Van Heusden, E. Dassau, H. C. Zisser, D. E. Seborg, and F. J. Doyle. Control-relevant models for glucose control using a priori patient characteristics. *IEEE Transactions on Biomedical Engineering*, 59(7):1839–1849, 2012.