**ECE 586 BH: Interplay between Control and Machine Learning** **Fall 2023**

# Lecture 5

**Policy-Based Reinforcement Learning for Control, Part I**

*Lecturer: Bin Hu, Date:09/05/2023*

Reinforcement learning (RL) has shown great promise for continuous control tasks. Policy optimization is a main workhorse for training RL agent to perform control tasks. In this lecture, we will discuss policy optimization and several popular policy-based RL methods.

There are many different types of policy-based RL methods, e.g. REINFORCE, Actor-Critic, natural policy gradient, TRPO, PPO, ACTKR, SAC, SVPG, etc. The basic idea is very simple, i.e. we parameterize the policy and then perform gradient-based optimization on the policy parameters using data. To illustrate this idea, consider the general nonlinear control for the following system:

$$\begin{aligned} x_{t+1} &= f(x_t, u_t, w_t) \\ y_t &= g(x_t, e_t) \end{aligned} \tag{5.1}$$

where $x_t$ is the state, $u_t$ is the control action, $w_t$ is the disturbance, $y_t$ is the output measurement that can be used for control, and $e_t$ characterizes all the external factors that affect the measurement. We want to design a feedback controller that determines control actions based on the output measurement. The key idea in policy optimization is that the nonlinear control design for (5.1) can be formulated as an optimization problem $\min_{K\in\mathcal{K}} \mathcal{C}(K)$, where the decision variable $K$ is determined by the controller parameterization, the cost function $\mathcal{C}(K)$ is a pre-specified control performance measure, and the feasible set $\mathcal{K}$ carries the information of the constraints on the controller. These concepts are briefly reviewed as follows.

- Decision variable $K$: At every time step $t$, the controller $K$ calculates the control action $u_t$ based on the measurement $y_t$ or a stack of past measurements $\{y_t, y_{t-1}, \cdots, y_{t-N}\}$. In the simplest case, we have the state measurement, i.e. $y_t = x_t$, and want to use a linear time-invariant (LTI) state-feedback controller. Then $K$ is parameterized as a static matrix and we have $u_t = -Kx_t$. Then this matrix $K$ becomes the decision variable of our policy optimization problem. For the so-called linear output feedback case, the controller can be a dynamical system that takes the output measurement $y_t$ as the input and the control action $u_t$ as the output, e.g.

$$\begin{aligned} \xi_{t+1} &= A_K \xi_t + B_K y_t \\ u_t &= C_K \xi_t + D_K y_t \end{aligned} \tag{5.2}$$

  Then the controller $K$ is parameterized by four matrices, i.e. $(A_K, B_K, C_K, D_K)$. For nonlinear control, nowadays it is popular to parameterize $K$ as neural networks.

- Objective function $\mathcal{C}(K)$: $\mathcal{C}(K)$ measures the quality of the controller $K$, and how to choose this cost function remains more of an art than a science. It really depends on the control tasks at hand. For example, for tracking tasks, we want to achieve small tracking errors while using small control actions. Then the cost can be a weighted sum of tracking error and control energy (but we need to tune the weights carefully). Sometimes the choice of the cost function is less obvious. For example, for self-driving, it is not that clear what is a good cost function choice that captures "comfortable driving." In general, we have to choose the objective functions in a case-by-case manner, and this requires domain expertise.

- Feasible set $\mathcal{K}$: Constraints on the decision variable $K$ are posed to account for various stability, robustness, or safety concerns. A common example in linear control tasks is the stability constraint, i.e. $K$ is required to stabilize the closed-loop dynamics. There are also other constraints related to robustness or safety concerns in control design. The constraints will naturally confine the policy search to a feasible set. For nonlinear control tasks, global stability is not required in general, and how to enforce local stability is still not that clear at this moment. The current practice in deep RL is just posing norm constraints on the weight matrices of the neural network policy $K$ or even adopting an unconstrained optimization formulation on a finite-horizon problem. The stability requirement is somehow embedded in the cost function (e.g. the cost will be huge for locally unstable system).

Once the control design problem is formulated as the policy optimization problem, one can just apply gradient-based methods such as gradient descent or quasi-Newton methods to optimize over the policy space. For example, if the cost function is differentiable, then it is straightforward to apply the gradient descent method:

$$K_{l+1} = K_l - \alpha \nabla \mathcal{C}(K_l) \tag{5.3}$$

In general, $\mathcal{C}$ is non-convex, and one may need to use multiple initial points to find a reasonably good solution. One may also use quasi-Newton updates to accelerate the optimization process.

**Model-based Policy Update vs. Model-Free Policy-Based RL.** When the system dynamics (5.1) are known and relatively easy, one can typically calculate $\nabla \mathcal{C}(K)$ directly. However, if the system is unknown or too complex, then one has to estimate $\nabla \mathcal{C}(K)$ from data. Importantly, policy-based RL methods can be viewed as model-free versions of the gradient-based scheme (5.3), where various learning tools such as policy gradient theorem or stochastic finite difference are used to estimate $\nabla \mathcal{C}(K)$ from sampled trajectories of (5.1). Such model-free methods can be more flexible and easy-to-implement for the following two main user-cases:

- Case 1 (Complex hybrid dynamics): The dynamic model can be very complex for robotic tasks such as robotic hand manipulation due to complicated object shapes and contact forces. No matter how complicated the dynamics are, model-free policy optimization can be implemented using sampled trajectories in a unified manner.

- Case 2 (High-dimensional rich observation): There are many modern autonomous systems that rely on high-dimensional rich sensing modalities such as cameras and LIDARs. In this case, $g$ maps the system state to high-dimensional measurements such as images and is significantly affected by external environmental factors such as lighting and weather. Model-free policy optimization provides an efficient framework to learn the controller as a mapping from the high-dimensional sensory data $y_t$ (such as images) to the control action $u_t$ in an end-to-end manner.

Next, we present a few examples to illustrate how model-based policy update and model-free policy-based RL work.

### 5.0.1 Model-based Policy Update: LQR as an Example

For illustrative purposes, let's look at the linear quadratic regulator (LQR) problem first. For simplicity, consider the following linear dynamical system

$$x_{t+1} = Ax_t + Bu_t \tag{5.4}$$

with the quadratic cost:

$$\mathcal{C} = \mathbb{E}_{x_0 \sim \mathcal{D}} \sum_{t=0}^{\infty} (x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t) \tag{5.5}$$

where $Q$ and $R$ are positive definite matrices. Let $\Sigma_0$ be the covariance matrix of $x_0$. The design goal for LQR is to choose control actions to minimize (5.5).

To formulate LQR as policy optimization, we first need to parameterize the policy/controller. For LQR, it is known that a state-feedback linear controller can achieve the optimal performance. Hence we can just confine the policy search to linear state-feedback policies, $u_t = -Kx_t$. Now the cost becomes a function of $K$. We have

$$\mathcal{C}(K) = \mathbb{E}_{x_0 \sim \mathcal{D}} \sum_{t=0}^{\infty} x_t^\mathsf{T} (Q + K^\mathsf{T} R K) x_t \tag{5.6}$$

In principle, we can just think the above function as an objective function and $K$ as decision variables. This allows us to formulate the LQR problem as an optimization problem $\min_K \mathcal{C}(K)$. Suppose we are using the gradient descent method (5.3) to solve this optimization problem. If we know $(A, B, Q, R)$, then we can calculate both $\mathcal{C}(K)$ and $\nabla \mathcal{C}(K)$ in closed-form.

**Cost formula.** If the model is known, the cost $\mathcal{C}(K)$ can be calculated as trace$(P_K\Sigma_0)$ by solving the following Bellman equation (Lyapunov equation)

$$P_K = Q + K^\mathsf{T} R K + (A - BK)^\mathsf{T} P_K (A - BK). \tag{5.7}$$

To see this, we just rewrite (5.6) as

$$\mathcal{C}(K) = \mathbb{E}_{x_0\sim\mathcal{D}} \quad x_0^\mathsf{T}\left(\sum_{t=0}^{\infty}((A-BK)^\mathsf{T})^t(Q+K^\mathsf{T}RK)(A-BK)^t\right)x_0 \tag{5.8}$$

When $\rho(A - BK) \geq 1$, the above cost blows up to infinity. It makes sense to restrict the policy search within the class of stabilizing $K$. So we have $\mathcal{K} = \{K : \rho(A - BK) < 1\}$. When $\rho(A-BK) < 1$, we know $\sum_{t=0}^{\infty}((A-BK)^\mathsf{T})^t(Q+K^\mathsf{T}RK)(A-BK)^t$ will converge to a fixed constant matrix. We denote this matrix by $P_K$. The Bellman equation can be derived as follows.

$$\begin{aligned}
x_0^\mathsf{T} P_K x_0 &= \sum_{t=0}^{\infty} x_t^\mathsf{T}(Q+K^\mathsf{T}RK)x_t \\
&= x_0^\mathsf{T}(Q+K^\mathsf{T}RK)x_0 + \sum_{t=1}^{\infty} x_t^\mathsf{T}(Q+K^\mathsf{T}RK)x_t \\
&= x_0^\mathsf{T}(Q+K^\mathsf{T}RK)x_0 + x_1^\mathsf{T} P_K x_1 \\
&= x_0^\mathsf{T}(Q+K^\mathsf{T}RK)x_0 + x_0^\mathsf{T}(A-BK)^\mathsf{T} P_K (A-BK)x_0 \\
&= x_0^\mathsf{T}\left(Q+K^\mathsf{T}RK + (A-BK)^\mathsf{T} P_K (A-BK)\right)x_0
\end{aligned}$$

Therefore, the Bellman equation takes the form (5.7). For any fixed $K$, the Bellman equation is a linear equation of $P_K$. Hence the existence and uniqueness of the solution to the Bellman equation can be established using linear equation theory. To obtain a closed-form solution for $P_K$, we introduce the Kronecker product and the vectorization operation. The Kronecker product of two matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$ is denoted by $A \otimes B$ and given by:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}.$$

where $a_{ij}$ is the $(i, j)$-th entry of $A$. Let vec denote the standard vectorization operation that stacks the columns of a matrix into a vector. For example, we have

$$\text{vec}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 3 \\ 5 \\ 2 \\ 4 \\ 6 \end{bmatrix}.$$

Since $\mathrm{vec}(AXB) = (B^\intercal \otimes A)\,\mathrm{vec}(X)$, we must have

$$\mathrm{vec}\left((A-BK)^\intercal P_K(A-BK)\right) = \left((A-BK)^\intercal \otimes (A-BK)^\intercal\right)\mathrm{vec}(P_K)$$

Then we can vectorize both sides of the Bellman equation (5.7) to obtain

$$\mathrm{vec}(P_K) = \mathrm{vec}(Q + K^\top RK) + \left((A-BK)^\intercal \otimes (A-BK)^\intercal\right)\mathrm{vec}(P_K)$$

which can be easily solved for $P_K$:

$$\mathrm{vec}(P_K) = \left(I - (A-BK)^\intercal \otimes (A-BK)^\intercal\right)^{-1}\mathrm{vec}(Q + K^\intercal RK)$$

Now we have a closed-form solution for $P_K$. Using properties of the Kronecker product, one can show $\left(I - (A-BK)^\intercal \otimes (A-BK)^\intercal\right)$ is nonsingular under the assumption $\rho(A-BK) < 1$. The key message here is that for any stabilizing $K$, we can solve (5.7) to obtain $P_K$ and then the cost value is given by $\mathcal{C}(K) = \mathbb{E}_{x_0\sim\mathcal{D}} x_0^\intercal P_K x_0 = \mathrm{trace}(P_K\Sigma_0)$.

**Policy gradient formula.** A useful result from control theory states that the LQR policy gradient $\nabla\mathcal{C}(K)$ can be calculated as

$$\nabla\mathcal{C}(K) = 2((R + B^\intercal P_K B)K - B^\intercal P_K A)\Sigma_K \tag{5.9}$$

where $\Sigma_K = \sum_{t=0}^\infty \mathbb{E}(x_t x_t^\intercal) = \sum_{t=0}^\infty (A-BK)^t\Sigma_0((A-BK)^\intercal)^t$. There are several proofs for this result. We present one proof here. We can take the total derivative of both sides of the Bellman equation to get

$$dP_K = d(K^\intercal RK) + d\left((A-BK)^\intercal P_K(A-BK)\right)$$

By the chain rule, we have

$$\begin{aligned}&dP_K\\ =&dK^\intercal RK + K^\intercal RdK + (A-BK)^\intercal dP_K(A-BK) - dK^\intercal B^\intercal P_K(A-BK) - (A-BK)^\intercal P_K BdK\\ =&dK^\intercal\left((R+B^\intercal P_K B)K - B^\intercal P_K A\right) + \left(K^\intercal(R+B^\intercal P_K B) - A^\intercal P_K B\right)dK + (A-BK)^\intercal dP_K(A-BK)\end{aligned}$$

If we view $dP_K$ as the variable, the above is a Bellman equation which can be solved as

$$dP_K = \sum_{t=0}^\infty ((A-BK)^\intercal)^t(dK^\intercal E_K + E_K^\intercal dK)(A-BK)^t$$

where $E_K = (R + B^\intercal P_K B)K - B^\intercal P_K A$. By definition, we have $d\mathcal{C}(K) = \sum_{i,j}\frac{\partial\mathcal{C}}{\partial K_{ij}}dK_{ij} = \mathrm{trace}(\nabla\mathcal{C}(K)dK^\intercal)$. Since $\mathcal{C}(K) = \mathrm{trace}(P_K\Sigma_0)$, it is also straightforward to show (you should verify this step by yourself)

$$d\mathcal{C}(K) = \mathrm{trace}(dP_K\Sigma_0) = \mathrm{trace}(2E_K\Sigma_K dK^\intercal)$$

Therefore, we have $\nabla\mathcal{C}(K) = 2E_k\Sigma_K$.

**Algorithms beyond the gradient descent method.** Based on the gradient formula, two other algorithms beyond the gradient method $K_{l+1} = K_l - \alpha \nabla \mathcal{C}(K_l)$ can be used.

- Natural policy gradient: $K_{l+1} = K_l - \alpha \nabla \mathcal{C}(K_l)\Sigma_{K_l}^{-1} = K_l - 2\alpha((R + B^\mathsf{T} P_{K_l} B)K_l - B^\mathsf{T} P_{K_l} A)$. This algorithm can be viewed as a quasi-Newton variant of the gradient method where $\Sigma_K$ is used to improve the condition of the update direction.
- Policy iteration: $K_{l+1} = K_l - (R + B^\mathsf{T} P_{K_l} B)^{-1} \nabla \mathcal{C}(K_l)\Sigma_{K_l}^{-1} = K_l - 2\alpha(K_l - (R + B^\mathsf{T} P_{K_l} B)^{-1} B^\mathsf{T} P_{K_l} A)$. This method provides even better conditioning on update direction than the natural policy gradient method. When $\alpha = \frac{1}{2}$, this method exactly becomes the policy iteration method. Interestingly, the policy iteration method can also be implemented in a model-free manner for LQR. For example, Least Square Policy Iteration (LSPI) provides one such model-free implementation.

## 5.0.2 Model-Free Policy Optimization

There is a famous result: policy gradient theorem. This theorem can be used to estimate the policy gradient from sampled trajectories directly. Notice that policy gradient theorem requires the policy to be stochastic so that we can explore the space. Denote the policy parameter in our controller $K$ as a big vector $\theta$. Now we discuss several versions of the policy gradient theorem.

**Warm-up: Policy Gradient Theorem V0.** To make things simple, we consider the additive structure for the cost (which is standard for RL) and use a large $N$ to approximate the infinite horizon. Therefore, we consider

$$\mathcal{C}(\theta) = \mathbb{E} \sum_{t=0}^{N} \gamma^t c(x_t, u_t) \tag{5.10}$$

Clearly the joint density of $(x_0, u_0, x_1, u_1, x_2, u_2, \ldots, x_N, u_N)$ depends on $\theta$ and we denote this density as $f_\theta$. By definition, we have

$$\mathbb{E} c(x_t, u_t) = \int c(x_t, u_t) f_\theta(x_0, u_0, x_1, u_1, x_2, u_2, \ldots, x_N, u_N) dx_0 du_0 dx_1 du_1 \ldots dx_N du_N$$

Hence we can take gradient on both sides to show

$$\begin{aligned}
\nabla_\theta \mathbb{E} c(x_t, u_t) &= \int c(x_t, u_t) \nabla_\theta f_\theta(\cdot) dx_0 du_0 dx_1 du_1 \ldots dx_N du_N \\
&= \int c(x_t, u_t) \frac{\nabla_\theta f_\theta(\cdot)}{f_\theta(\cdot)} f_\theta(x_0, u_0, x_1, u_1, \ldots, x_N, u_N) dx_0 du_0 dx_1 du_1 \ldots dx_N du_N \\
&= \int c(x_t, u_t) \nabla_\theta \log f_\theta(\cdot) f_\theta(x_0, u_0, x_1, u_1, \ldots, x_N, u_N) dx_0 du_0 dx_1 du_1 \ldots dx_N du_N \\
&= \mathbb{E}\left[c(x_t, u_t) \nabla_\theta \log f_\theta(\cdot)\right]
\end{aligned}$$

In addition, we have

$$\begin{aligned}&\log f_\theta(x_0,u_0,\ldots,x_N,u_N)\\=&\log \pi_\theta(u_N|x_N)+\log p(x_N|x_{N-1},u_{N-1})+\log \pi_\theta(u_{N-1}|x_{N-1})+\log p(x_{N-1}|x_{N-2},u_{N-2})+\ldots\end{aligned}$$

After taking gradients, we have

$$\nabla_\theta \log f_\theta(x_0,u_0,\ldots,x_N,u_N)=\sum_{t=0}^{N}\nabla_\theta \log \pi_\theta(u_t|x_t)$$

Putting things together, we have

$$\begin{aligned}\nabla \mathcal{C}(\theta)&=\sum_{t=0}^{N}\gamma^t\mathbb{E}\left[c(x_t,u_t)\sum_{k=0}^{N}\nabla_\theta\log\pi(u_k|x_k)\right]\\&=\mathbb{E}\left[\big(\sum_{t=0}^{N}\gamma^t c(x_t,u_t)\big)\big(\sum_{k=0}^{N}\nabla_\theta\log\pi(u_k|x_k)\big)\right]\end{aligned}$$

The above is the most naive version of the policy gradient theorem. It states that we can estimate the policy gradient by sampling the cost sequence and calculating $\nabla_\theta \log\pi(u_t|x_t)$.

**How to calculate $\nabla_\theta \log \pi_\theta(u_t|x_t)$?** When applying the policy gradient theorem, a stochastic policy has to be used. The noise is injected into the gradient for exploration purpose. For example, consider a LQR problem where one can confine the policy search to linear policies, i.e. $u_t=-Kx_t$. However, to apply the policy gradient theorem, noise has to be injected into the policy, and hence one typically use a Gaussian policy, i.e. $u_t\sim\mathcal{N}(-Kx,\sigma I)$. Basically we just add zero-mean Gaussian noise to the control input $-Kx_t$. How can we calculate $\nabla_\theta\log\pi_\theta(u_t|x_t)$ for Gaussian policies? Suppose $\sigma$ is fixed. Then log will cancel the exponential part in the Gaussian density, and we can calculate $\nabla_\theta\log\pi_\theta(u_t|x_t)$ easily. For complicated tasks, we typically use neural network to parameterize the policy, and $\nabla_\theta\log\pi_\theta(u_t|x_t)$ can be calculated using standard back propagation method. Such operations are available in PyTorch or TensorFlow.

**REINFORCE: Policy Gradient Theorem V1.** The gradient estimator introduced above is noisy and has high variance. Now we discuss an improved version of the policy gradient theorem. Let us directly address the infinite horizon problem, i.e.

$$\mathcal{C}(\theta)=\mathbb{E}\sum_{t=0}^{\infty}\gamma^t c(x_t,u_t)$$

Notice the process is causal, i.e. $(x_t,u_t)$ do not depend on the states/actions in the future. To calculate $\mathbb{E}c(x_t,u_t)$, we only need the joint density of $(x_0,u_0,x_1,u_1,\ldots,x_t,u_t)$. Hence we have

$$\mathbb{E}c(x_t,u_t)=\int c(x_t,u_t)f_\theta(x_0,u_0,x_1,u_1,x_2,u_2,\ldots,x_t,u_t)dx_0du_0dx_1du_1\ldots dx_tdu_t$$

After applying similar tricks, we can show

$$\nabla_\theta \mathbb{E} c(x_t, u_t) = \mathbb{E}\left[ c(x_t, u_t) \sum_{k=0}^{t} \nabla_\theta \log \pi_\theta(u_k|x_k) \right]$$

To see what happens when we sum all terms, we write out the first few term explicitly:

$$\begin{aligned}
\nabla_\theta \mathbb{E} c(x_0, u_0) &= \mathbb{E}\left[c(x_0, u_0)\nabla_\theta \log \pi_\theta(u_0|x_0)\right] \\
\gamma \nabla_\theta \mathbb{E} c(x_1, u_1) &= \mathbb{E}\left[\gamma c(x_1, u_1)\nabla_\theta \log \pi_\theta(u_0|x_0)\right] + \mathbb{E}\left[\gamma c(x_1, u_1)\nabla_\theta \log \pi_\theta(u_1|x_1)\right] \\
\gamma^2 \nabla_\theta \mathbb{E} c(x_2, u_2) &= \mathbb{E}\left[\gamma^2 c(x_2, u_2)\nabla_\theta \log \pi_\theta(u_0|x_0)\right] + \mathbb{E}\left[\gamma^2 c(x_2, u_2)\nabla_\theta \log \pi_\theta(u_1|x_1)\right] \\
&\quad + \mathbb{E}\left[\gamma^2 c(x_2, u_2)\nabla_\theta \log \pi_\theta(u_2|x_2)\right]
\end{aligned}$$

Based on the above pattern, we can get

$$\begin{aligned}
\nabla \mathcal{C}(\theta) &= \mathbb{E}\sum_{t=0}^{\infty}\left[ \Big(\sum_{k=t}^{\infty} \gamma^k c(x_k, u_k)\Big) \nabla_\theta \log \pi_\theta(u_t|x_t) \right] \\
&= \mathbb{E}\sum_{t=0}^{\infty}\left[ \gamma^t \Big(\sum_{k=t}^{\infty} \gamma^{k-t} c(x_k, u_k)\Big) \nabla_\theta \log \pi_\theta(u_t|x_t) \right]
\end{aligned}$$

which gives an improved version of the policy gradient theorem. The algorithm REINFORCE is actually based on the above gradient formula.

**Generalizations.** There are many variants of the policy gradient theorem which can be used to improve the policy gradient estimation. Specifically, we can estimate the policy gradient as

$$\nabla \mathcal{C}(\theta) = \mathbb{E}\sum_{t=0}^{\infty} [\gamma^t \Psi_t \nabla_\theta \log \pi_\theta(u_t|x_t)] \tag{5.11}$$

where $\Psi_t$ can be chosen as:

- Monte Carlo estimation: $\sum_{t'=t}^{\infty} \gamma^{t'-t} c_{t'}$
- Baselined versions of Monte Carlo estimation: $\sum_{t'=t}^{\infty} (\gamma^{t'-t} c_{t'} - b(x_t))$
- State-action value function: $Q^\pi(x_t, u_t)$
- Advantage function (Actor-critic method) : $A^\pi(x_t, u_t)$
- TD residual: $c_t + \gamma V^\pi(x_{t+1}) - V^\pi(x_t)$
- Generalized advantage estimation

We will skip the detailed derivations of these variants. There are also various improvements of the vanilla policy gradient method. Examples include the natural policy gradient method, the trust-region policy optimization (TRPO) method, the proximal policy optimization (PPO), and the soft actor-critic (SAC) method. There exist efficient implementations of TRPO, PPO, and SAC which can be found online. it is worth emphasizing that all these methods require the cost function to have an additive structure.

**Zeroth-order optimization.** When using the policy gradient theorem, we inject noise into the control actions for exploration purposes. Suppose now we want to directly learn the gradient of a deterministic policy. Then we can use evolution strategies (or zeroth-order optimization) which does not require stochastic policy. The exploration in zeroth-order optimization is done by perturbing the policy randomly. Specifically, consider the following estimation of the policy gradient:

$$\nabla \mathcal{C}(K) \approx \frac{\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,\sigma^2 I)}\mathcal{C}(K+\varepsilon)\varepsilon}{\sigma^2}$$

To understand this update rule, we analyze a shifted variant of the above update:

$$g = \frac{\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,\sigma^2 I)}(\mathcal{C}(K+\varepsilon)-\mathcal{C}(K))\varepsilon}{\sigma^2} = \frac{\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}(\mathcal{C}(K+\sigma\varepsilon)-\mathcal{C}(K))\varepsilon}{\sigma}$$

Roughly speaking, the above estimation shifts the original zeroth-order gradient estimate with a zero mean vector and should not change the mean of the gradient estimator. Due to the fact $\lim_{\sigma\to 0}\frac{\mathcal{C}(K+\sigma\varepsilon)-\mathcal{C}(K)}{\sigma} = (\nabla\mathcal{C}(K))^{\mathsf{T}}\varepsilon$, we can show:

$$\begin{aligned}
\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}\left(\lim_{\sigma\to 0}\frac{\mathcal{C}(K+\sigma\varepsilon)-\mathcal{C}(K)}{\sigma}\right)\varepsilon &= \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}(\varepsilon^{\mathsf{T}}\nabla\mathcal{C}(K))\varepsilon \\
&= \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}\varepsilon(\varepsilon^{\mathsf{T}}\nabla\mathcal{C}(K)) \\
&= \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}(\varepsilon\varepsilon^{\mathsf{T}})\nabla\mathcal{C}(K) \\
&= \nabla C(K)
\end{aligned}$$

Therefore, the zeroth-order optimization can be viewed as a stochastic version of the finite difference method. Notice that zeroth-order optimization is general since it can address any cost function. The drawback is that this approach is too general and does not exploit the problem structure of MDPs. Hence the sample efficiency of zeroth-order optimization may not be as good as policy gradient theorem when applied to standard Markov decision process (MDP) problems with additive cost structures. Nevertheless, variants of the above zeroth-order optimization methods have achieved competitive results on many model-free control tasks.