Reinforcement Learning as optimal control for Shear Flows

Hosseinkhan Boucher, Rémy - Laboratoire Interdisciplinaire des Sciences du Numérique

Joint work with:

Semeraro, Onofrio
Mathelin, Lionel

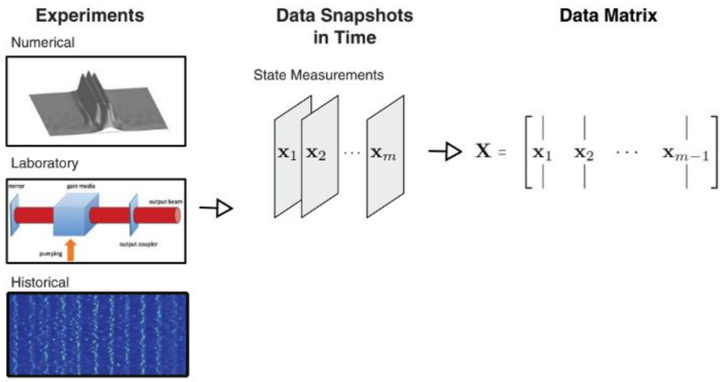
Model-based vs. Model-free Control

Model-based: An **explicit representation** of the environment, system, is provided to design a control policy.

Example (Navier-Stokes equation for fluid models): $\frac{\partial u}{\partial t} + (u \cdot \nabla)u - \nu \nabla^2 u = -\frac{1}{\rho}\nabla p + g$

Example (Geometric Brownian Motion for stock prices): $dS_t = \mu S_t dt + \sigma S_t dW_t$

Model-free: An implicit representation of the system is derived using **statistics** and **data sets**.



Data can be collected from a number of differents sources. Dynamic Mode Decomposition: Data Driven Modeling Of Complex Systems, J. N. Kutz et al. (2016).

Reinforcement Learning (RL): Data-driven control policy.

Data is collected by **interacting with the environment**.

Objective: Improve state-of-the-art RL algorithms applied to fluids-mechanics dynamical systems. Find **data sampling and processing strategies** to obtain better control learning.

Kuramoto-Sivashinsky Equation

Definition (Controlled Kuramoto-Sivashinsky): $\frac{\partial v}{\partial t}(x,t) + v(x,t)\frac{\partial v}{\partial x}(x,t) = -\frac{\partial^2 v}{\partial x^2}(x,t) - \frac{\partial^4 v}{\partial x^4}(x,t) + \phi(u(x),t)$

with periodic condition $v(x+L,t) = v(x,t)$ where $(x,t) \in [0,L] \times [0,T]$.

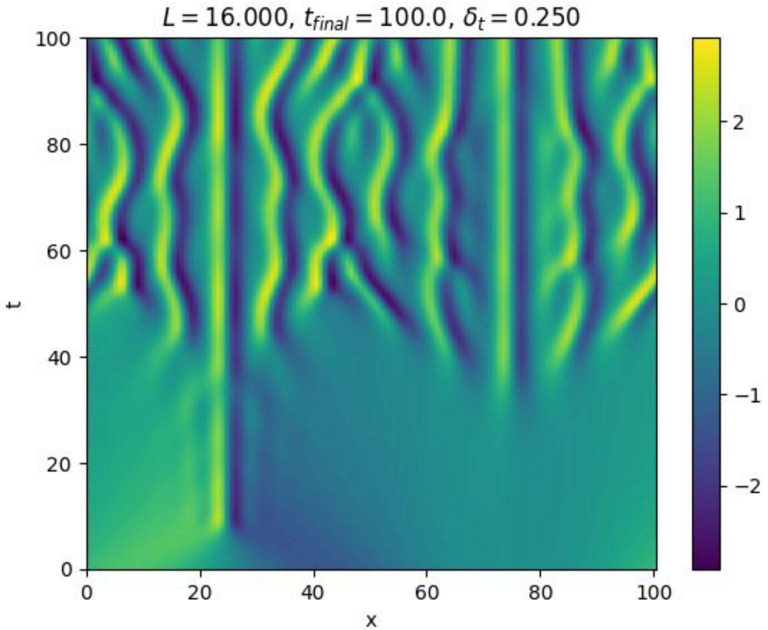
ϕ is called *control*, u is called *action*.

History:

- Derived in 1977-78
- Flame propagation (flame front)
- Reaction-Diffusion systems

Properties:

- Spatio-temporal **chaos**
- 4$^{\text{th}}$ order non-linear PDE
- Stiffness
- Equilibria and relative equilibria
- Symmetries of solutions



Time evolution of the Kuramoto Sivashinsky equation with L = 16.

[1] Diffusion-Induced Chaos in Reaction Systems, Y. Kuramoto (1978).

[2] Nonlinear analysis of hydrodynamic instability in laminar flames—I. Derivation of basic equations, G.I. Sivashinsky (1977).

Markov Decision Processes

Given a dynamical system $x_{t+1} = G(x_t, u_t)$ defined by $G : \mathcal{X} \to \mathcal{X}$ with $u_t \in \mathcal{A}$, $x_t \in \mathcal{X}$.

Note: $\mathcal{A}$ is called *action space* and $\mathcal{X}$ is called state space

Modelisation Hypothesis (our Dynamical System is a MDP):

$$X : \Omega \to \mathcal{X}^{\mathbb{N}} \qquad U : \Omega \to \mathcal{A}^{\mathbb{N}}$$

$$H_t := X_0,\, U_1,\, X_1,\, U_1, \ldots,\, X_{t-1},\, U_{t-1},\, X_t$$

Transition Probability:

$P((x, u),\, dx\,)$ is a distribution over $\mathcal{X}$

Policy (Markovian stationary policy):

$\pi(x, da)$ is a distribution over $\mathcal{A}$

Process Distribution

$$P^\pi\left(dx_0, du_0, dx_1, du_1 \ldots, dx_t\right) = \nu\left(dx_0\right)\pi\left(x_0,\, du_0\right)P\left(dx_2 \mid x_1,\, u_1\right)\pi\left(x_2,\, du_2\right)\cdots$$
$$\pi\left(x_{t-1},\, du_{t-1}\right)P\left(dx_t \mid x_{t-1},\, u_{t-1}\right)$$

Remark (Dynamical System case):

Transition probability is $\delta_{\{G(x_t,\, u_t)\}}(dx_{x+1})$ for deterministic system (when model is given).

Markov Decision Processes

Policy (Markovian stationary policy):

$\pi(x, da)$ is a distribution over $\mathcal{A}$

Example (Policy):

$$\pi(x, da) \sim \mathcal{N}(\mu_x, \sigma_x)$$

Criterion (Cost function):

$$J_x^{\pi^*} := \mathbb{E}_x^{\pi^*}\left[\sum_{t=1}^{\infty} \gamma^t c\left(X_t, U_t\right)\right]$$

Cost-per-stage:

$$c : \mathcal{X} \times \mathcal{A} \to \mathbb{R}_+$$

Example (Cost-per-stage):

$$c\left(x, a\right) = \|x\| + \|u\|$$

Optimal policy:

$$\pi^* := \arg\min_{\pi} J^{\pi}$$

State-Action function (Q-function):

Statistics: Markov Decision Process estimation

Hypothesis (Parametric Statistics): Policy is *parametrised* by some vector $\theta \in \Theta$ i.e. $\pi := \pi_\theta$

Consequently, criterion becomes parametrised: $J^\pi := J^{\pi_\theta} = J(\theta)$

Optimal *control policy* is then given by π_{θ^*} where $\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}}\, J_\theta(x)$

Trajectories are sampled to estimate the process distribution:

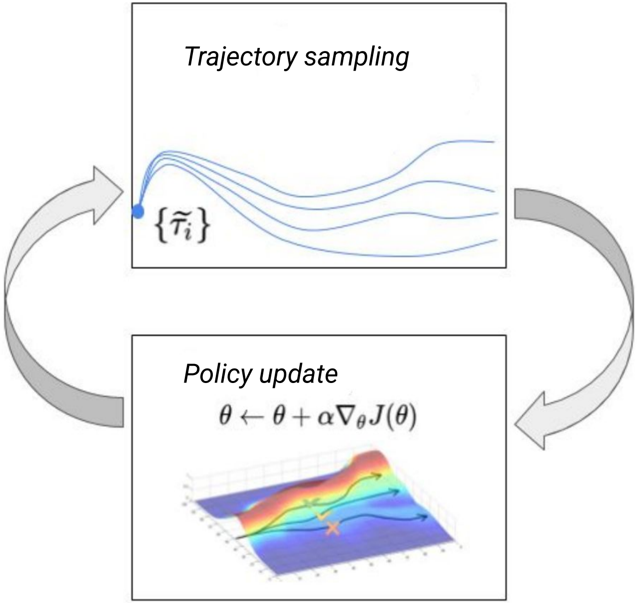
Monte Carlo method (Estimation)

$$h^i = (x_1^i, u_1^i, x_2^i, \ldots, x_{t-1}^i, u_{t-1}^i, x_t^i)$$

$$J_x(\theta) = E_x^\pi \left[\sum_{t=0}^{\infty} \alpha^t c\left(x_t, a_t\right)\right] \simeq \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{t=0}^{\infty} \alpha^t c\left(x_t^i, a_t^i\right)\right]$$

Optimisation (Gradient Descent)

$$\theta_{t+1} = \theta_t + \eta \nabla J(\theta_t)$$



Deep Deterministic Policy Gradient

In practice, π_θ is a neural network *perturbed by random noise* and a more sophisticated algorithm based on this concept is applied (DDPG[1]).

Parametrisation of Q:

Let Q_β be a functional approximator parametrised by β.

DDPG Policy:

$$\pi_\theta(x) := f_\theta(x) + \mathcal{N}(\mu, \sigma)$$

State-action update (Critic)

Update Q by minimising:

$$L_1(\beta) = \mathbb{E}\left[(c(X,U) + \gamma Q_\beta(X, \pi_\theta(X) - Q_\beta(X,U)))^2\right]$$

Policy update (Actor)

Update policy by minimising

$$L_2(\theta) = \mathbb{E}[J_X^{\pi_\theta}] = \mathbb{E}[Q_\beta(X, \pi_\theta(X))]$$

Bellman Equation:

$$J_x^{\pi^*} := E_x^{\pi^*}\left[\sum_{t=1}^{\infty} \gamma^t c\,(X_t, U_t)\right]$$

$$J_x^{\pi^*} = \min_{u \in \mathcal{A}} \left[c(x,u) + \gamma\, \mathbb{E}_{P_{x,u}}^{\pi^*}\left[J_X^{\pi^*}\right]\right]$$

$$\min_{u \in \mathcal{A}} Q(x,u) = \min_{u \in \mathcal{A}} \left[c(x,u) + \gamma\, \mathbb{E}_{P_{x,u}}^{\pi^*}\left[Q^{\pi^*}(X,u)\right]\right]$$

$$Q(x,u) := c(x,u) + \gamma\, \mathbb{E}_{P_{x,u}}^{\pi^*}\left[J_X^{\pi^*}\right]$$

[1] Continuous control with deep reinforcement learning, T. P. Lillicrap (2015).

Reinforcement Learning: Kuramoto-Sivashinsky environment

In the context of the Kuramoto-Sivashinsky equation:

State and action spaces:

$$\mathcal{X} = L^2([0, L]) \simeq \mathbb{R}^d$$

$$\mathcal{A} = L^2([-a, a]) \simeq [-a,\, a]^b$$

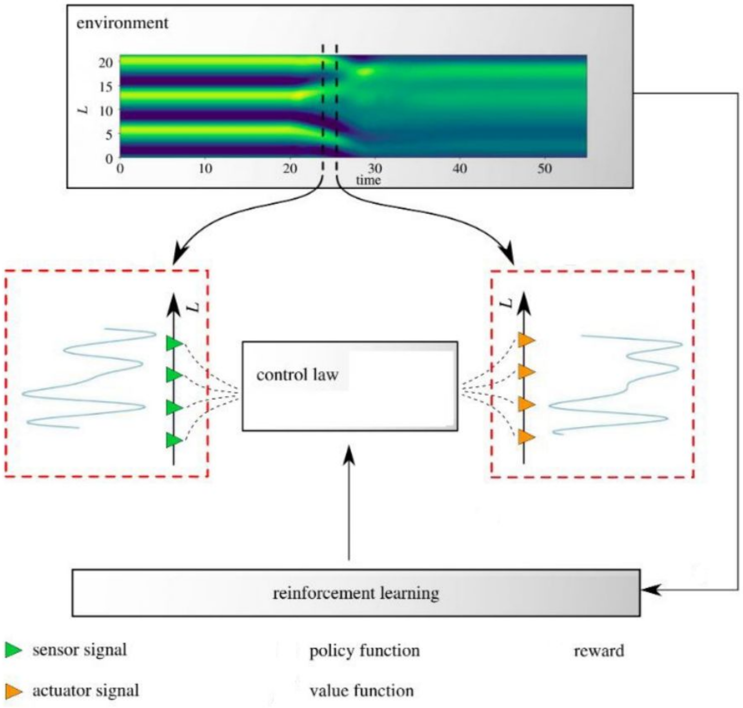
Cost is the *energy* of the system:

$$c(x) = \|x\|_2^2$$

Control is a *gaussian mixture* weighted by $U_t \sim \pi\,(X_t,\, \cdot)$:

$$\phi(U) = \sum_{i=1}^{b} U_i \frac{1}{2\pi\sigma} \exp\left(-\frac{(x - x_i^{\mathrm{a}})^2}{2\sigma^2}\right)$$

The evolution of the system is performed with spatial and temporal discretization with **exponential time-differencing**.

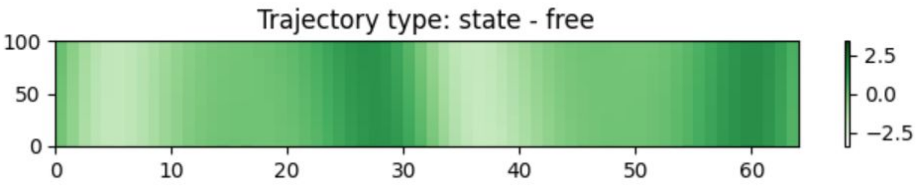


Schema of the reinforcement learning process for the Kuramoto-Sivashinsky dynamical system. Control of Chaotic Systems by Deep Reinforcement Learning, M. A. Bucci et al. (2019).

Experiments: Stabilising the dynamics

Under $L = 22$ KS, has multiple steady-state solutions.

Let fix one solution called E_2 .



Objective:

Stabilise the dynamics from E_2 to $E_0 = 0$.

Cost function $c(x) = \|x - E_2\|$

Configuration:

Method used: *Deep Deterministic Policy Gradient (DDPG)*

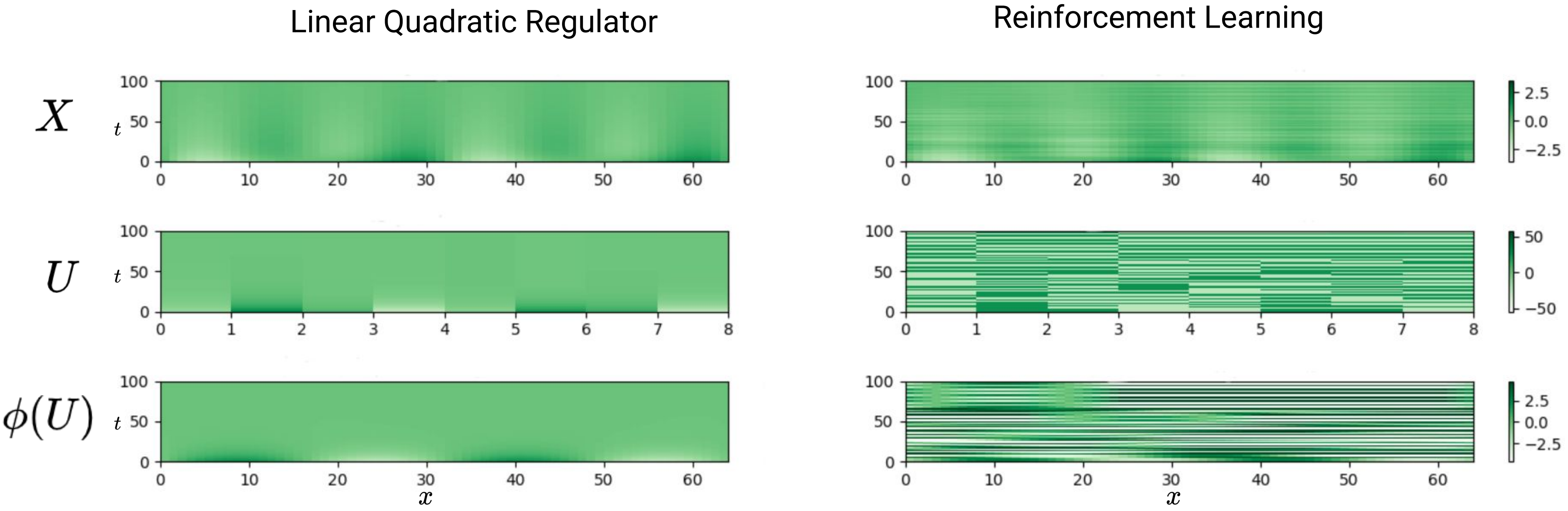
Policy (π_θ): two layers *neural network*, 64 hidden neurons

Optimiser: Adam[1], (Gradient Descent based)

Time steps: $t \in \{0, \ldots, 100\}$

Actuators: equi-spaced along x-axis, $b = 8$

Experiments: Stabilising the dynamic



Linear Quadratic Regulator:

Optimal control u of the linearised system:

$$x' = D_{KS}^{E_2}\,x + B_\phi\,u$$

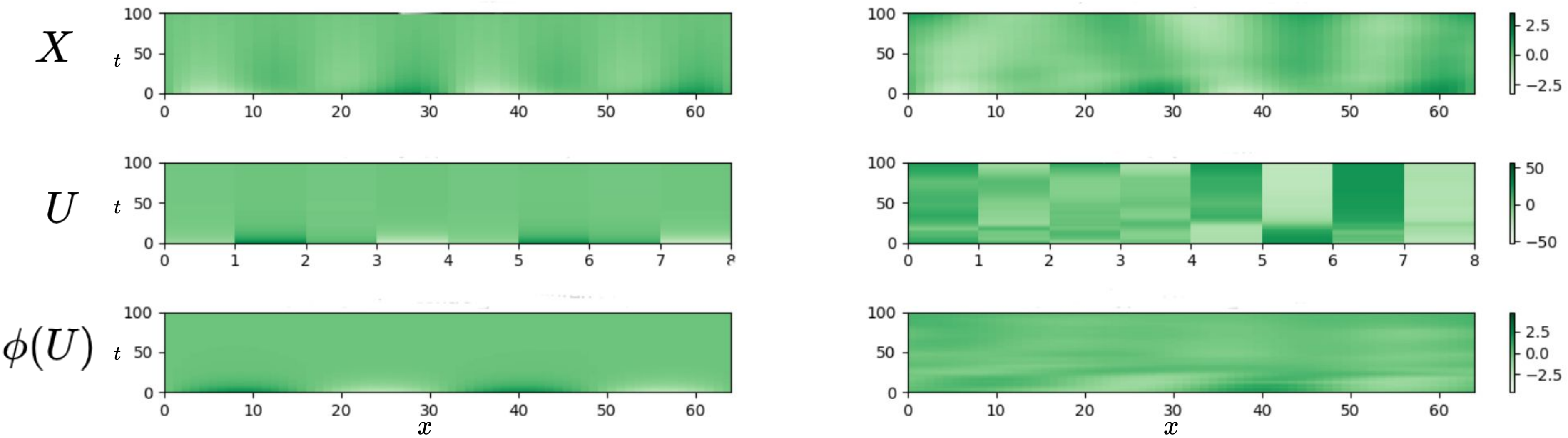
$$B_\phi\,u \simeq \phi(u)$$

Observations:

- Controlled trajectory with RL is **comparable** with LQR.
- However, LQR control is more **physically meaningful**.

Experiments: Stabilising the dynamic

Linear Quadratic Regulator | *Reinforcement Learning*



Linear Quadratic Regulator:

Optimal control u of the linearised system:

$$x' = D_{KS}^{E_2}\, x + B_\phi\, u$$

$$B_\phi\, u \simeq \phi(u)$$

Observations:

- Controlled trajectory with RL is **worst** than with LQR
- However, RL control is more **physically interpretable**.

Conclusion