ROB-GY 6323
reinforcement learning and optimal control for robotics

Lecture 1
Introduction

Ludovic Righetti

Associate Professor
Mechanical and Aerospace Engineering
Electrical and Computer Engineering

Contact

ludovic.righetti@nyu.edu
370 Jay Street, Room 801

Office Hours

Wednesday 3 to 4pm
Any other time => questions online or by appointment

Understanding the fundamental algorithmic principles of autonomous robotic locomotion and manipulation

MACHINES IN MOTION

Huaijiang Zhu

Avadesh Meduri

Quim Ortiz

Quang Nam Nguyen

Stacy Ashlyn

Armand Jordana

Sarmad Mehrdad

Joseph Amigo

Jianghan Zhang


KUKA
KUKA


Initial policy
(Zero forces)
F1-3
F1-3

Goals of the class how to get a robot to move "optimally"

1 Algorithms to compute complex robot movements
Using optimal control and reinforcement learning

2 Practical application of algorithms in real world applications
(used in many applications beyond robotics)

Necessary background

- Calculus (derivatives, gradients, Taylor series, etc)
- Linear algebra (positive definiteness, inverse, etc)
- Python

What is optimal control?

What is reinforcement learning?

Model-Predictive Control

compute optimal
trajectories in real-time

Reinforcement Learning

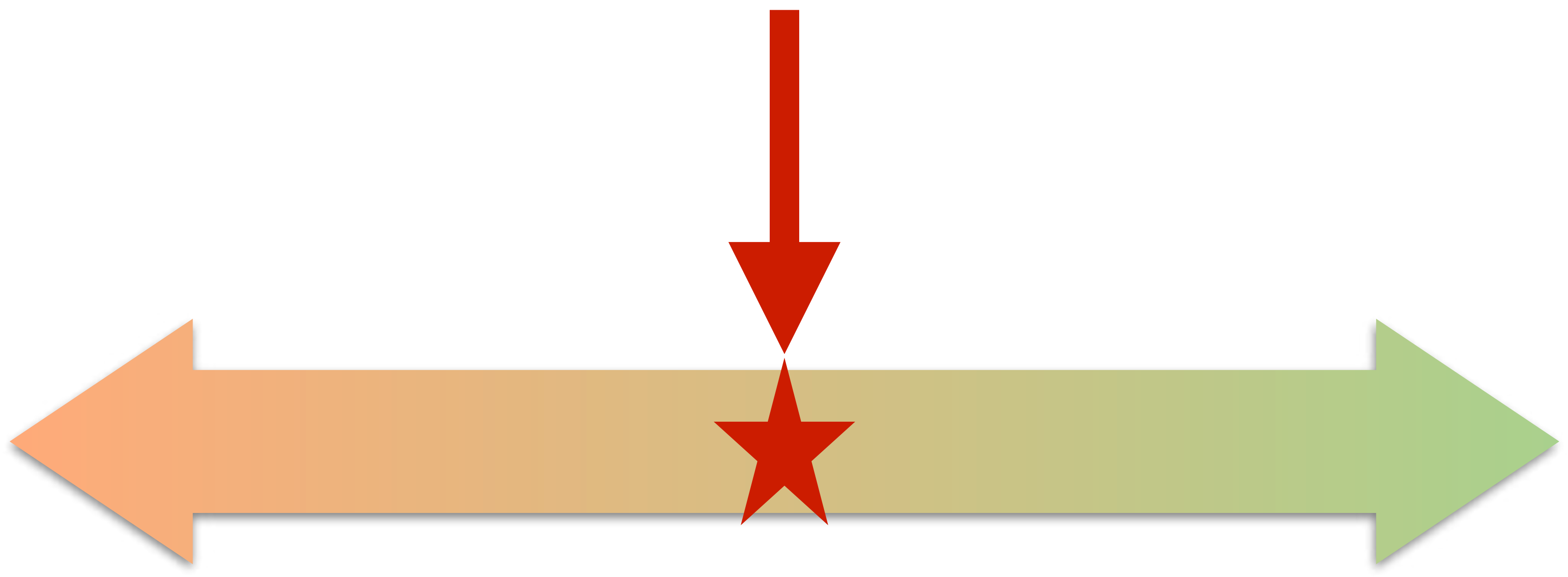
pre-compute
a policy offline

[Boston Dynamics 2022]

[Huang et al. 2022]

Model-Predictive Control

compute optimal trajectories in real-time



Reinforcement Learning

pre-compute a policy offline

$$u = \pi(x, \text{obstacles})$$

$$\min \sum_{n=0}^{N} l_n(x_n, u_n)$$

$$x_{n+1} = x_n + \Delta t \cdot f(x_n, u_n)$$

$$g(x_n, u_n) \leq 0$$

explicit dynamics

Both approaches solve optimal control problems => numerical optimization

simulator

Tentative list of topics

- Basics of optimization

- Trajectory optimization:
  - Linear problems (LQR)
  - Nonlinear problems (SQP)
  - Model predictive control
  - Zero order methods (MPPI)

- Policy optimization:
  - Bellman’s principle of optimality and dynamic programming
  - Value- and Policy-iteration
  - Deep Q-learning
  - Policy gradients and actor-critic algorithms (PPO)
  - Monte-Carlo tree search

Your turn…

…what do you expect from the class?

Course Project(s) (50%)
Goal: implement in simulation algorithms seen in class

Homework (25%)
To be handed in every few weeks
(theory + programming exercises in Python)

Paper Report (25%)

Course material

All necessary material will be posted on Brightspace
Code will be posted on the Github site of the class

https://github.com/righetti/optlearningcontrol

Discussions/Forum with Slack or Discord?

Contact

ludovic.righetti@nyu.edu
Office hours in person
Wednesday 3pm to 4pm
370 Jay street - room 801
(except next week)

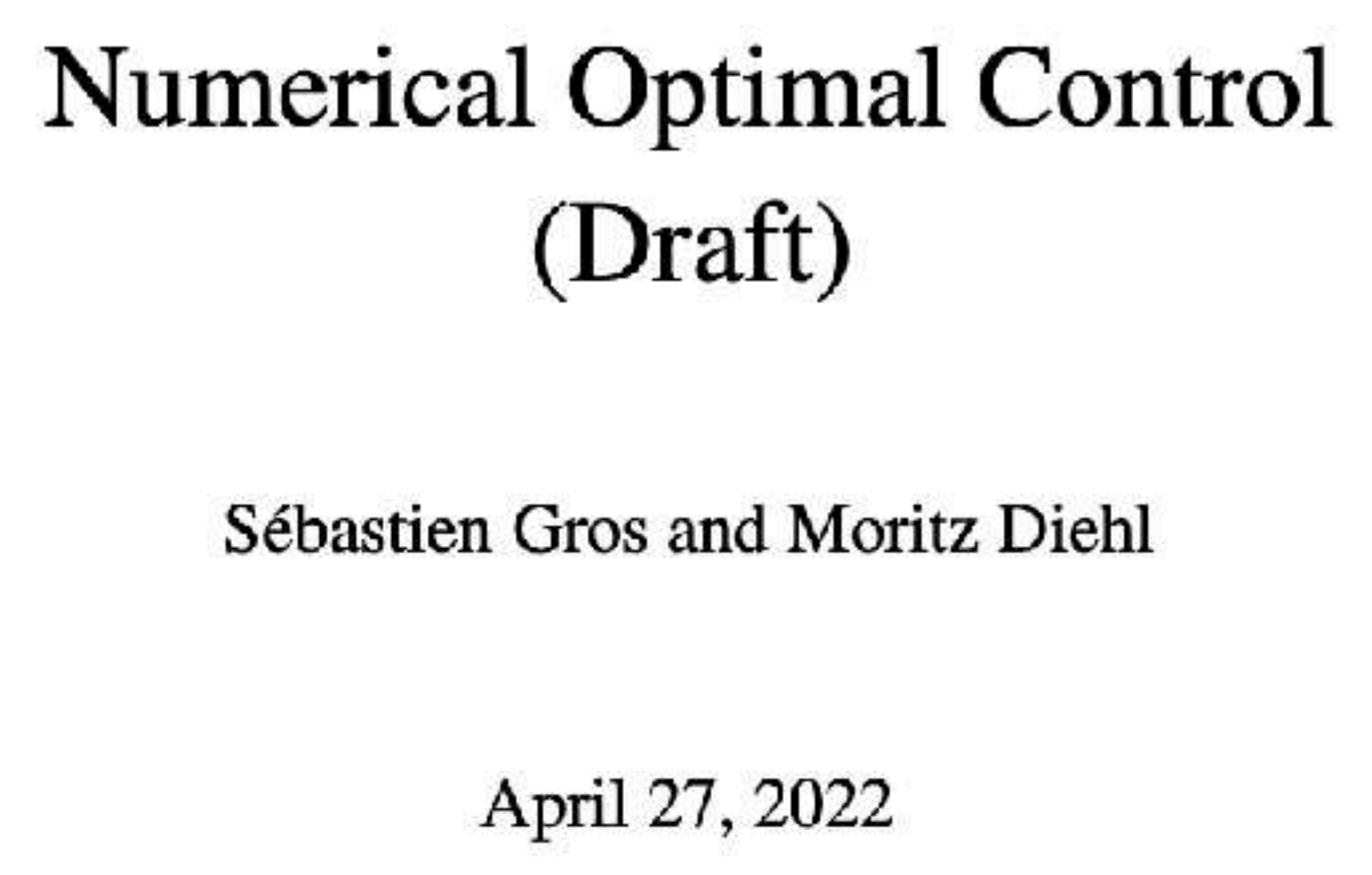
Course Assistant

Armand Jordana
aj2988@nyu.edu
Office hours Monday 1pm to 2pm
Rogers Hall 515

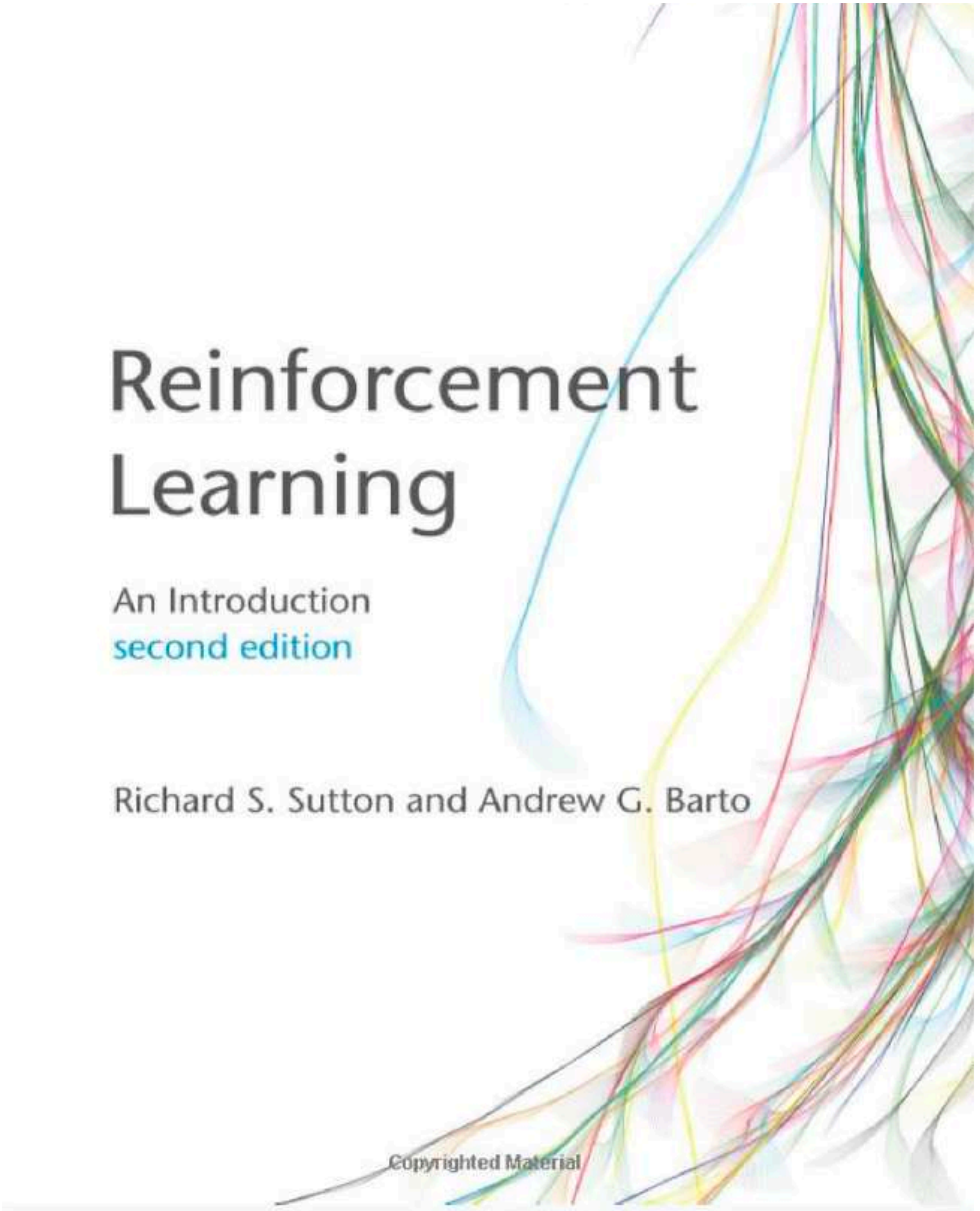
any other time by appointment only

The class will not follow any particular book

But these resources are good references



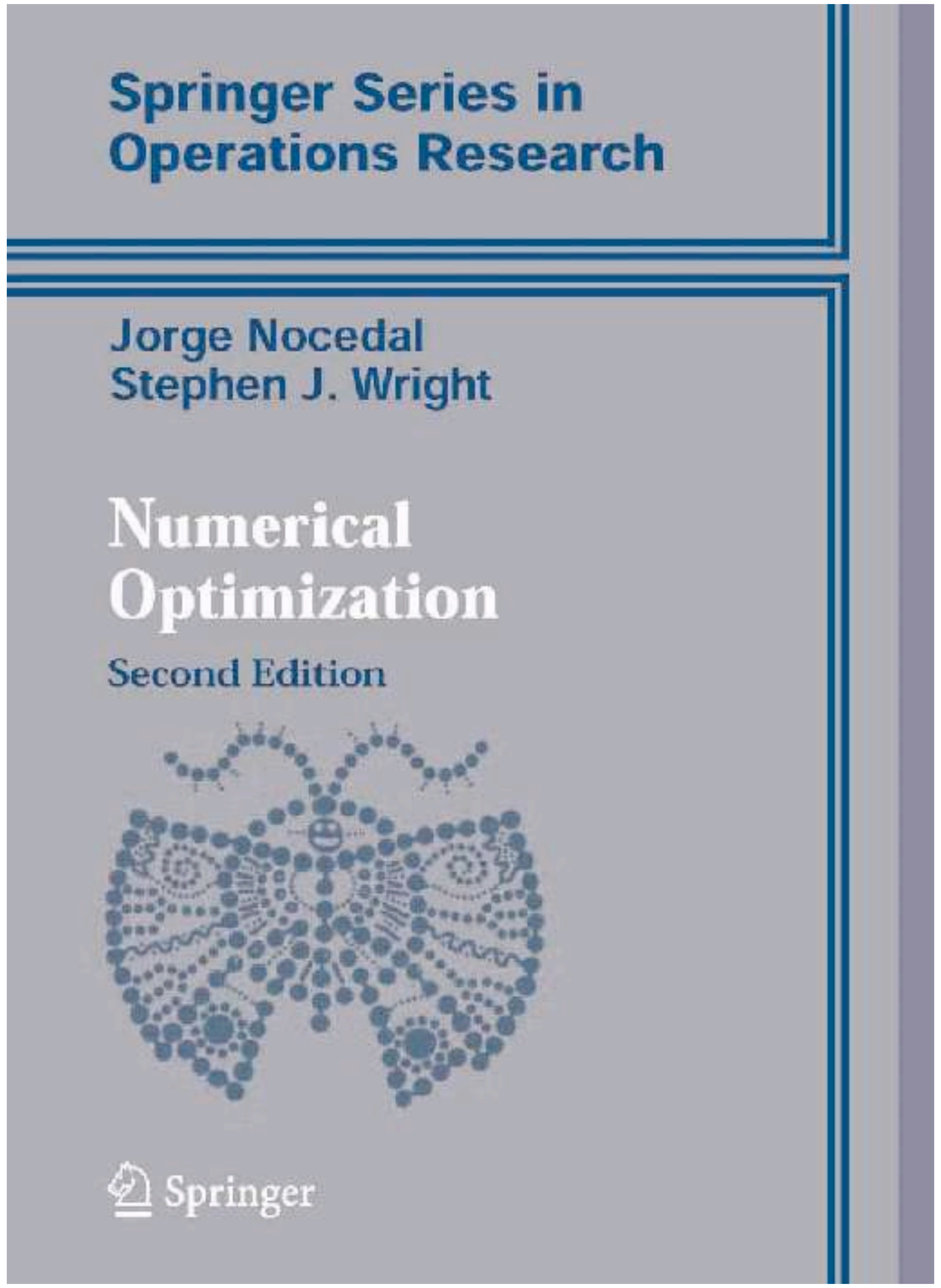
https://www.syscop.de/files/2020ss/NOC/book-NOCSE.pdf

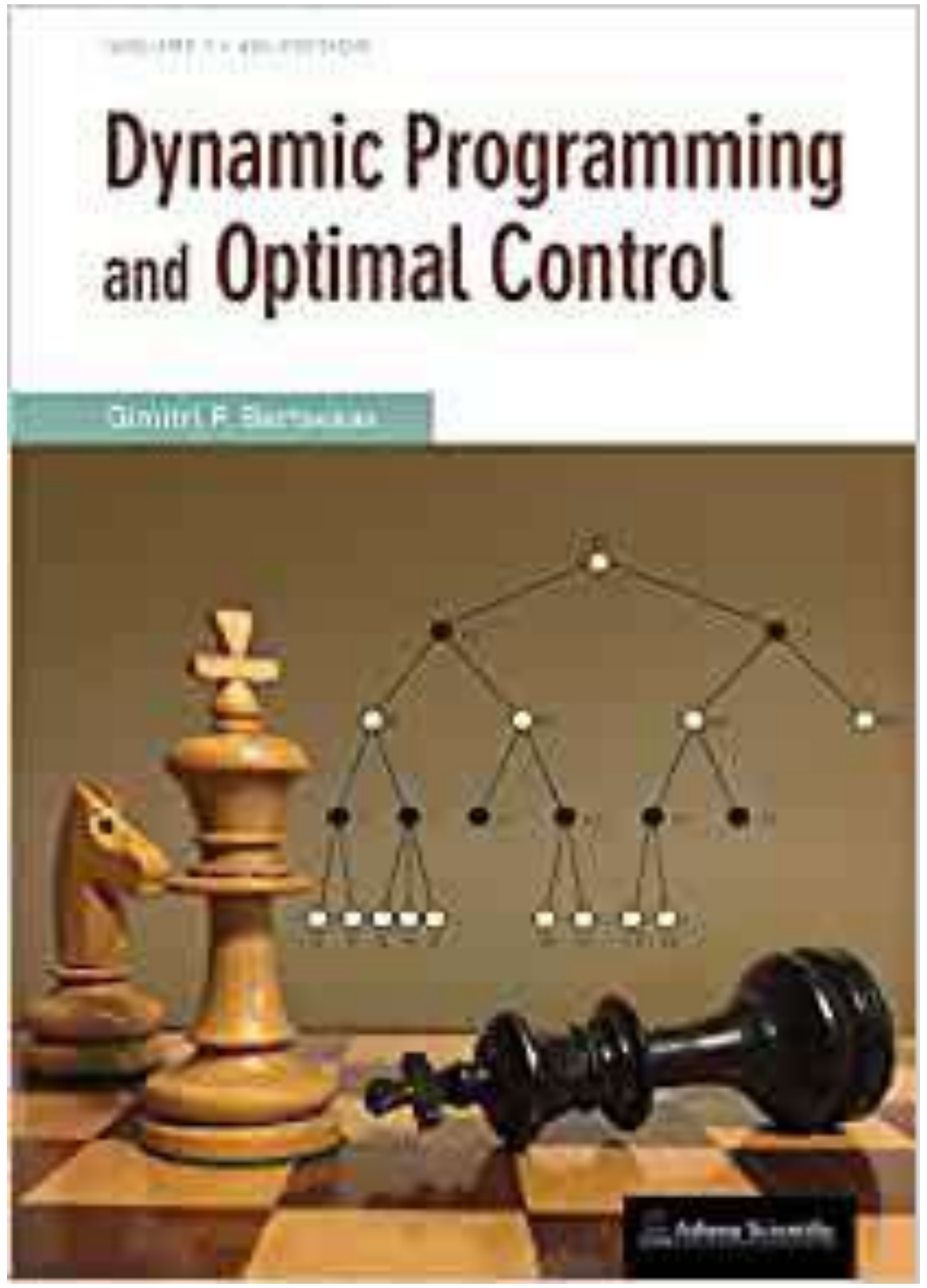


http://incompleteideas.net/book/the-book-2nd.html

The class will not follow any particular book

But these resources are good references



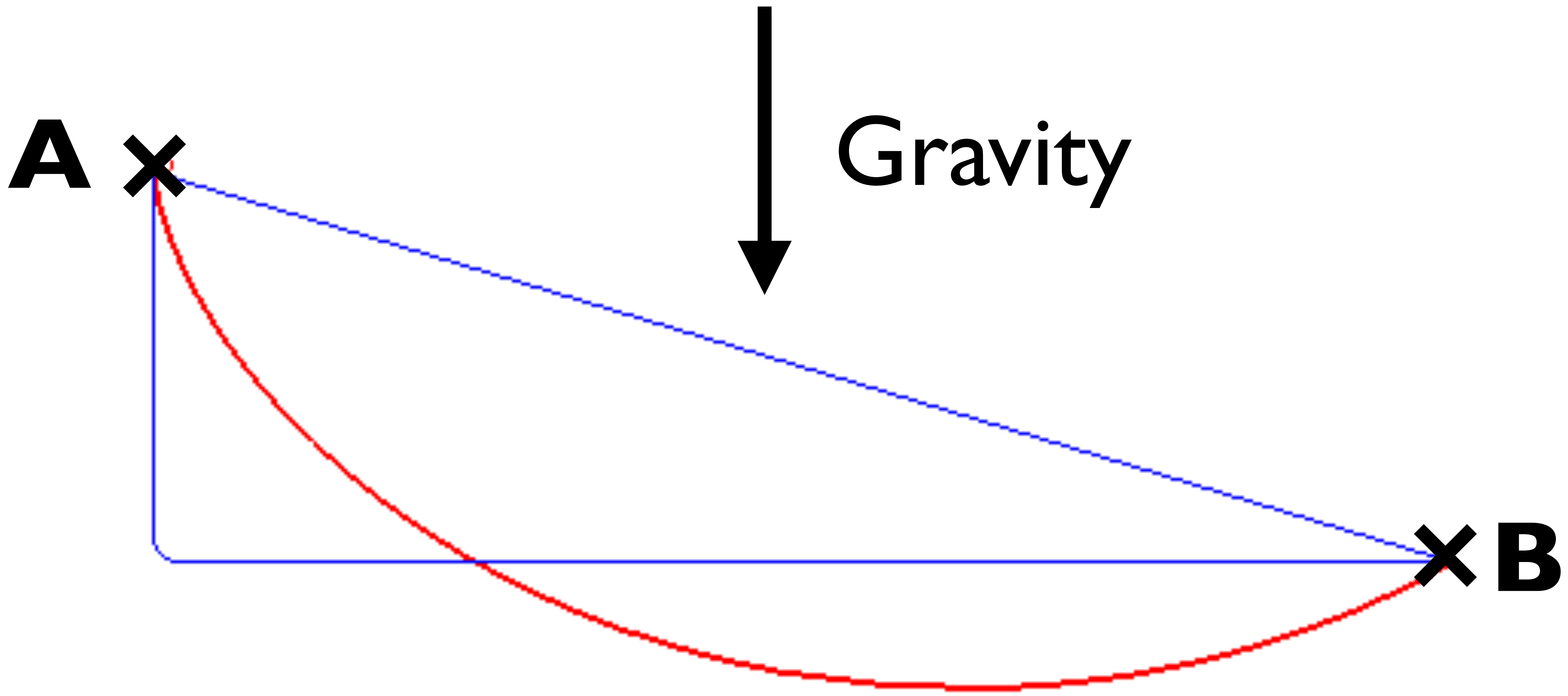


D. Bertsekas (Vol. 1+2)

Questions?

Optimal Control Problems date back to XVIIth century

Johann Bernoulli posed the question in 1696: "what is the shape of the ground that allows a ball to reach B from A in the fastest time?"



The optimal curve (red) is called the Brachistochrone curve

Actually, it is not quite yet an optimal control problem

It was originally solved by 5 mathematicians independently

Jacob Bernoulli, Isaac Newton, Gottfried Leibniz, Ehrenfried W. von Tschirnhaus and Guillaume de l'Hôpital





The theory of optimal control dates from the 1950s

In 1951 H.W Kuhn and A.W. Tucker formulated first order necessary conditions to find a minimum of a nonlinear optimization problem - W. Karush had found them in his M.Sc. thesis in 1939

In 1954, Richard Bellman formulated a "principle of optimality" to provide necessary conditions to solve optimal control problems
(aka the dynamic programming algorithm)

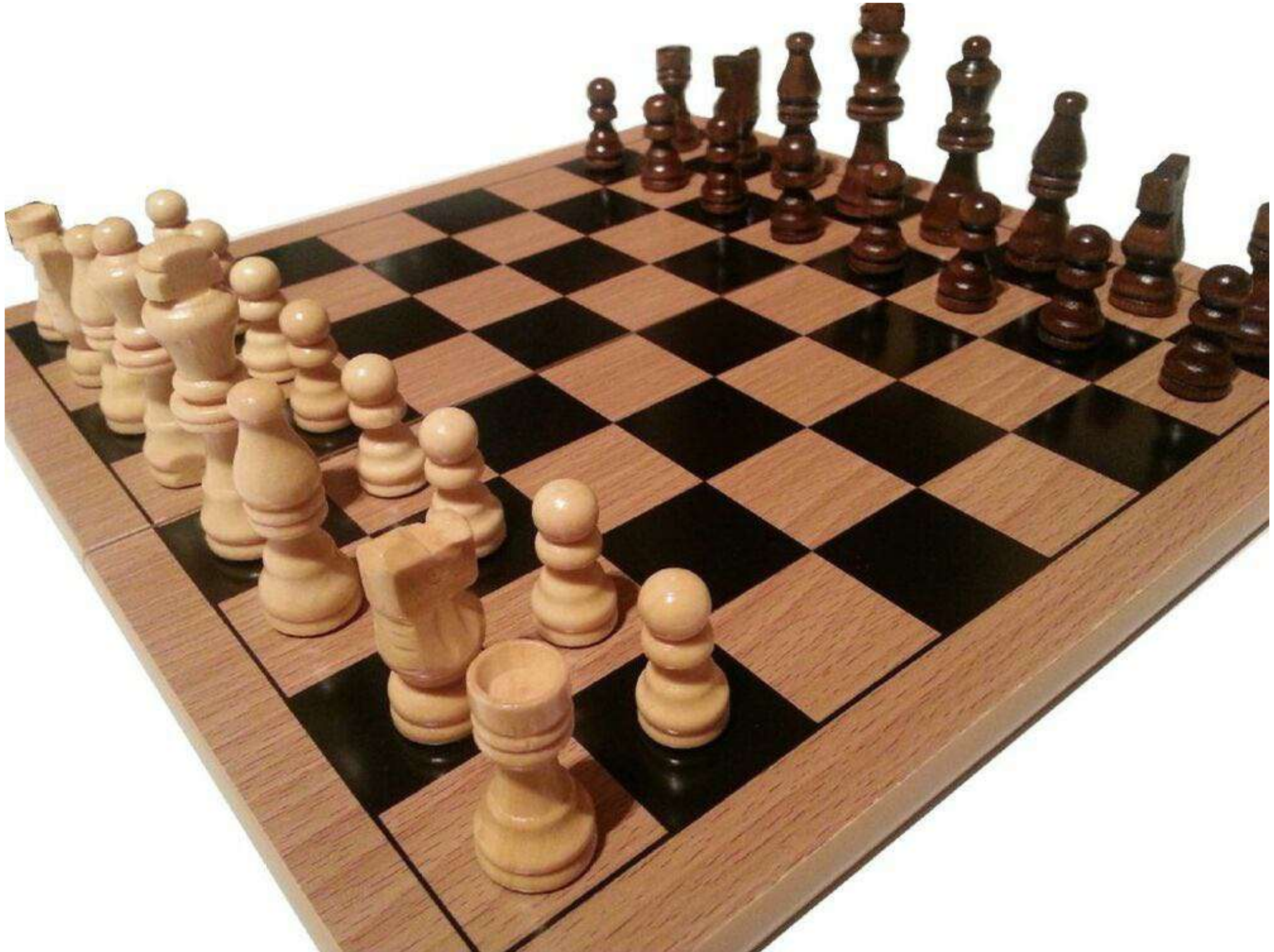
In 1956, Lev Semyonovich Pontryagin formulated a "maximum principle" to provide necessary conditions to solve optimal control problems

These ideas are the foundations of most modern optimal control and reinforcement learning algorithms

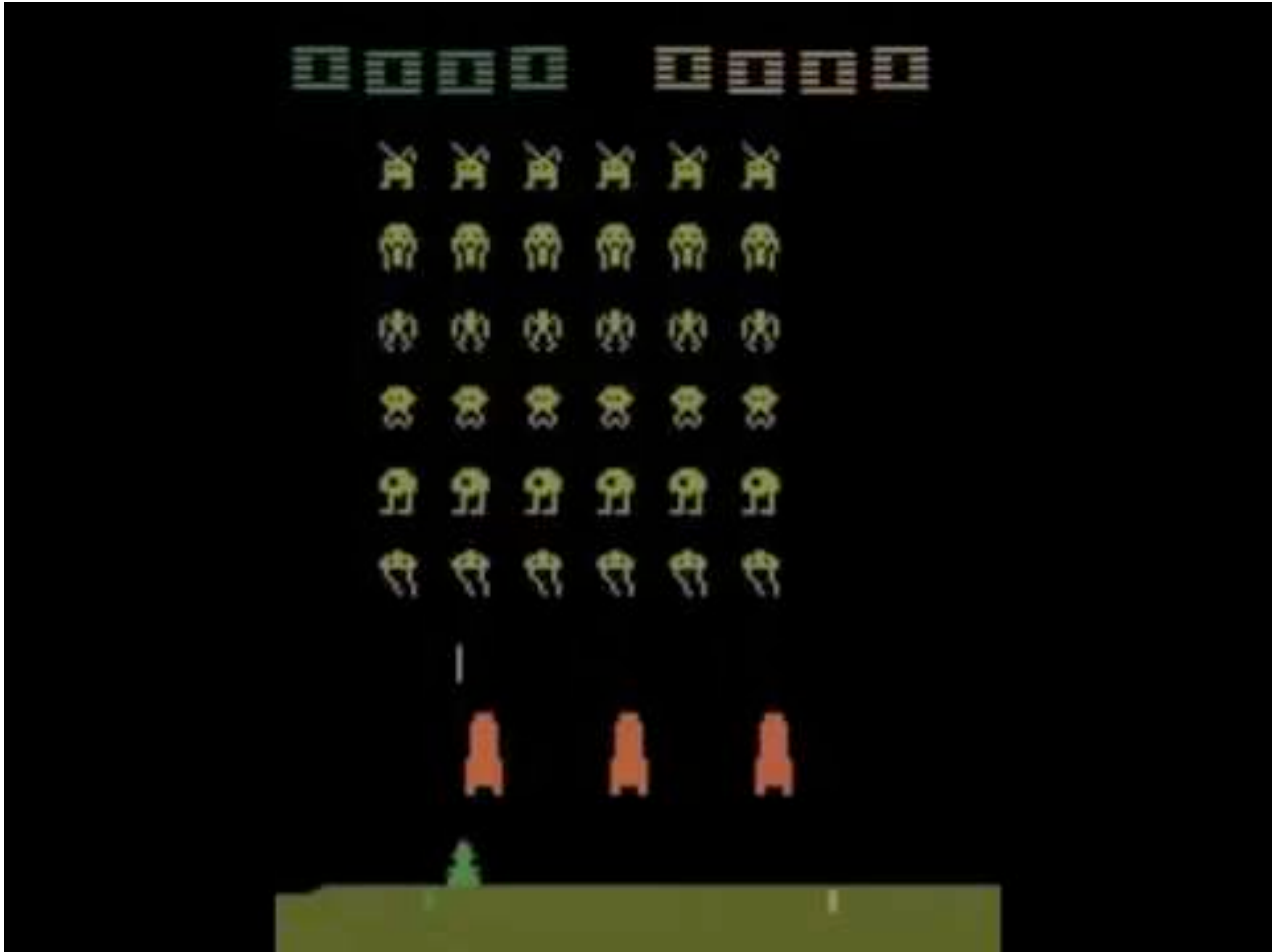
Optimal control $\simeq$ Reinforcement learning

Finding the "best" way to solve a sequential decision making problem

A sequential decision making problem



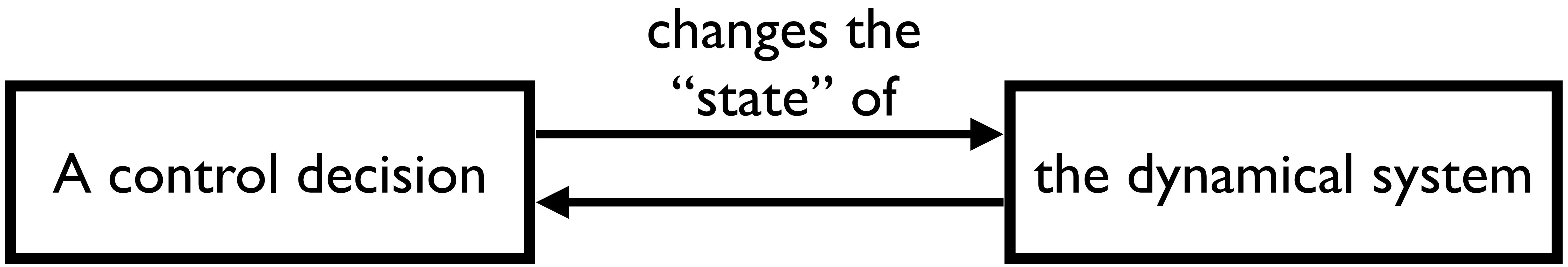
A sequential decision making problem

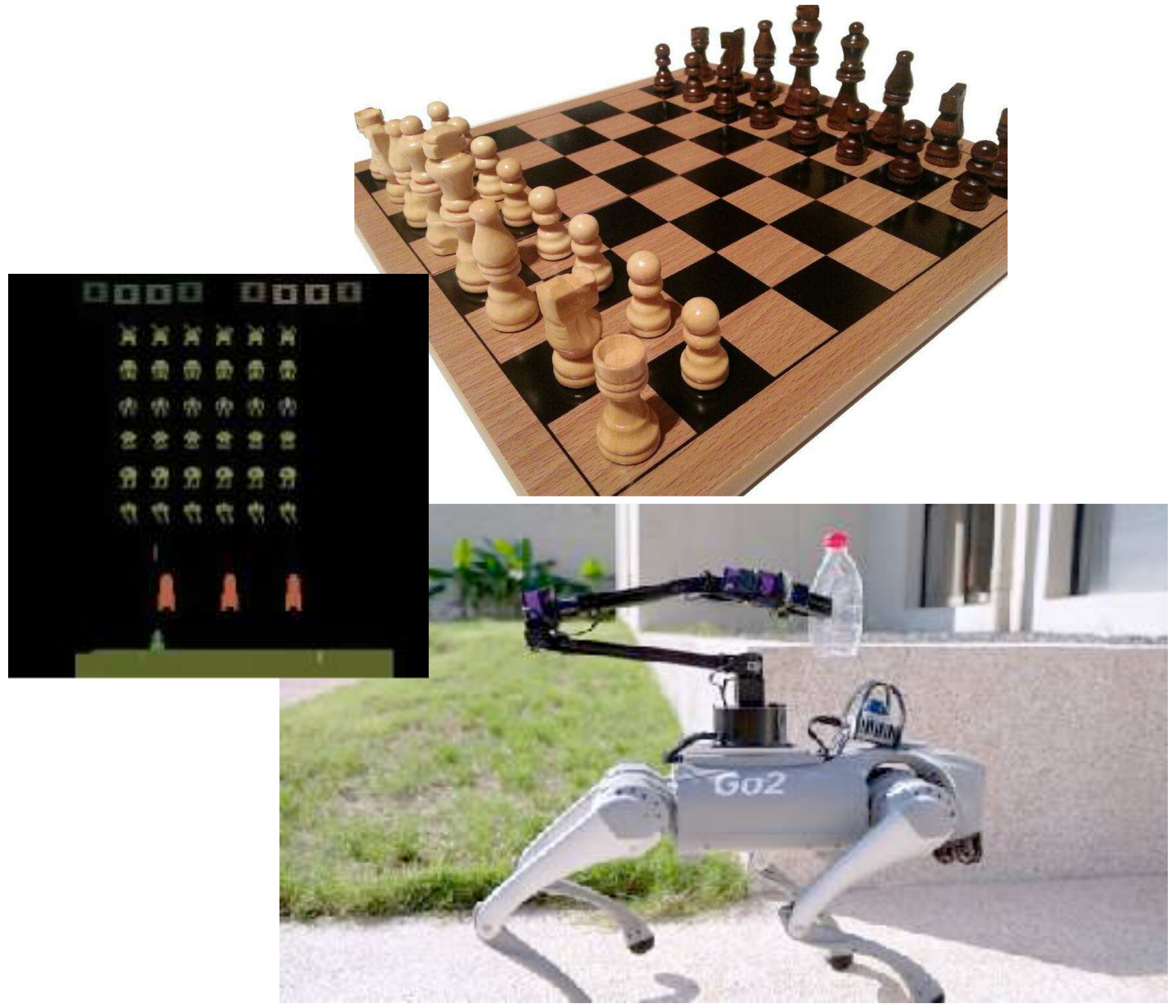


A sequential decision making problem

Luo et al. 2024

A sequential decision making problem



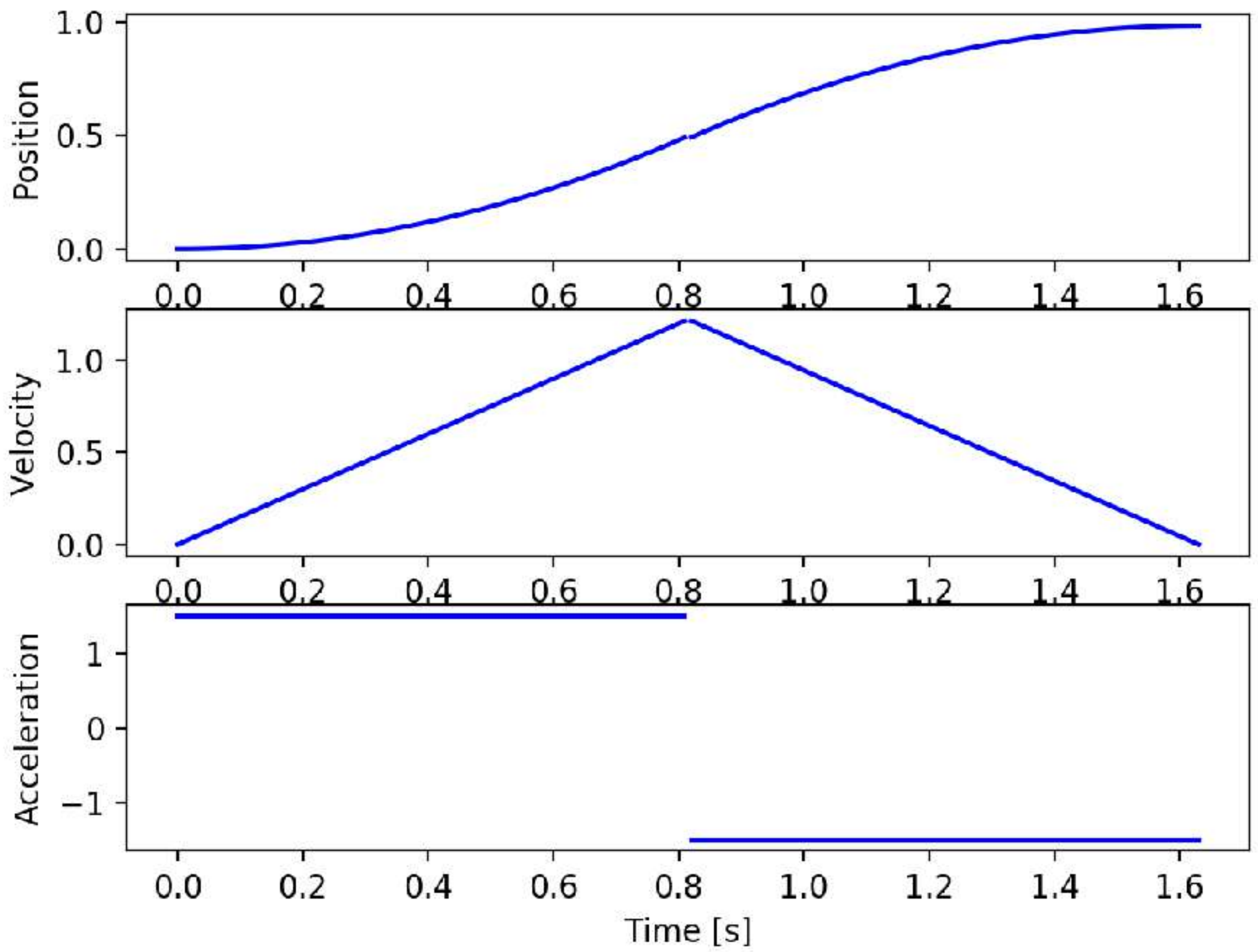


The problem: find the "**best sequence of actions**" to make the **dynamical system** behave as desired (e.g. win the game or do a backflip)

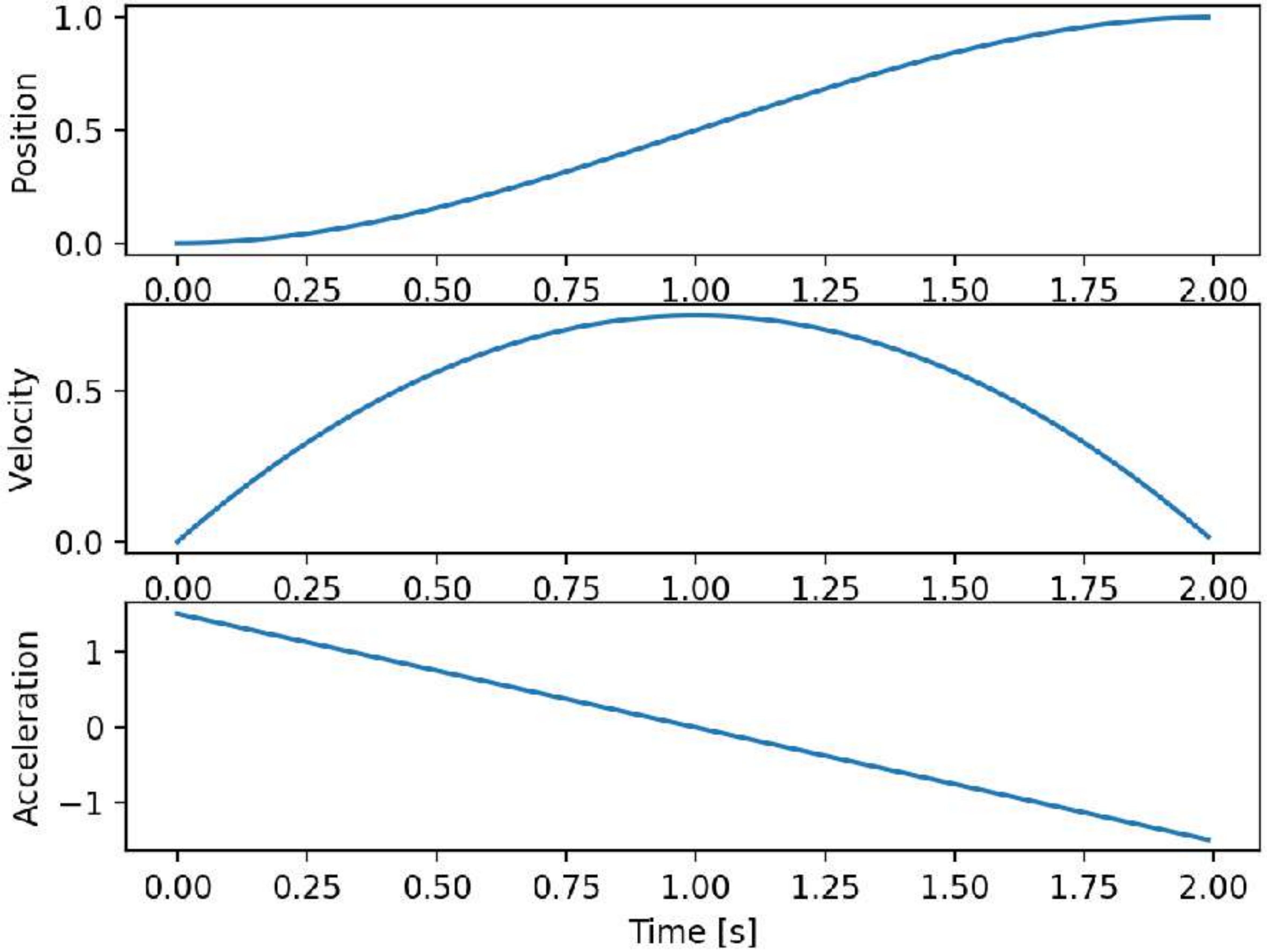
Best sequence of actions?

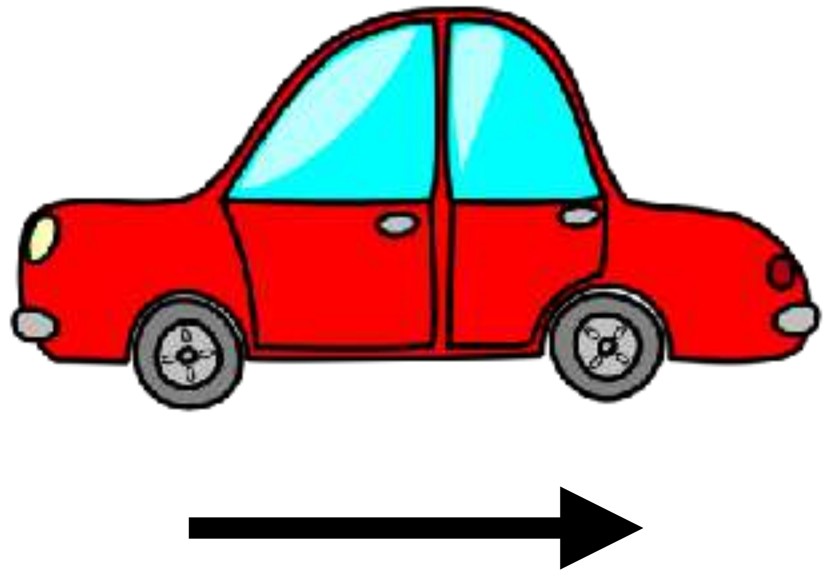
- How to accelerate a car to reach a goal in minimum time?
- How to accelerate a car to reach a goal in minimum acceleration?
- How to accelerate a car to minimize fuel consumption?
- How to accelerate a car to maximize passenger comfort?

minimum time?



minimum acceleration?

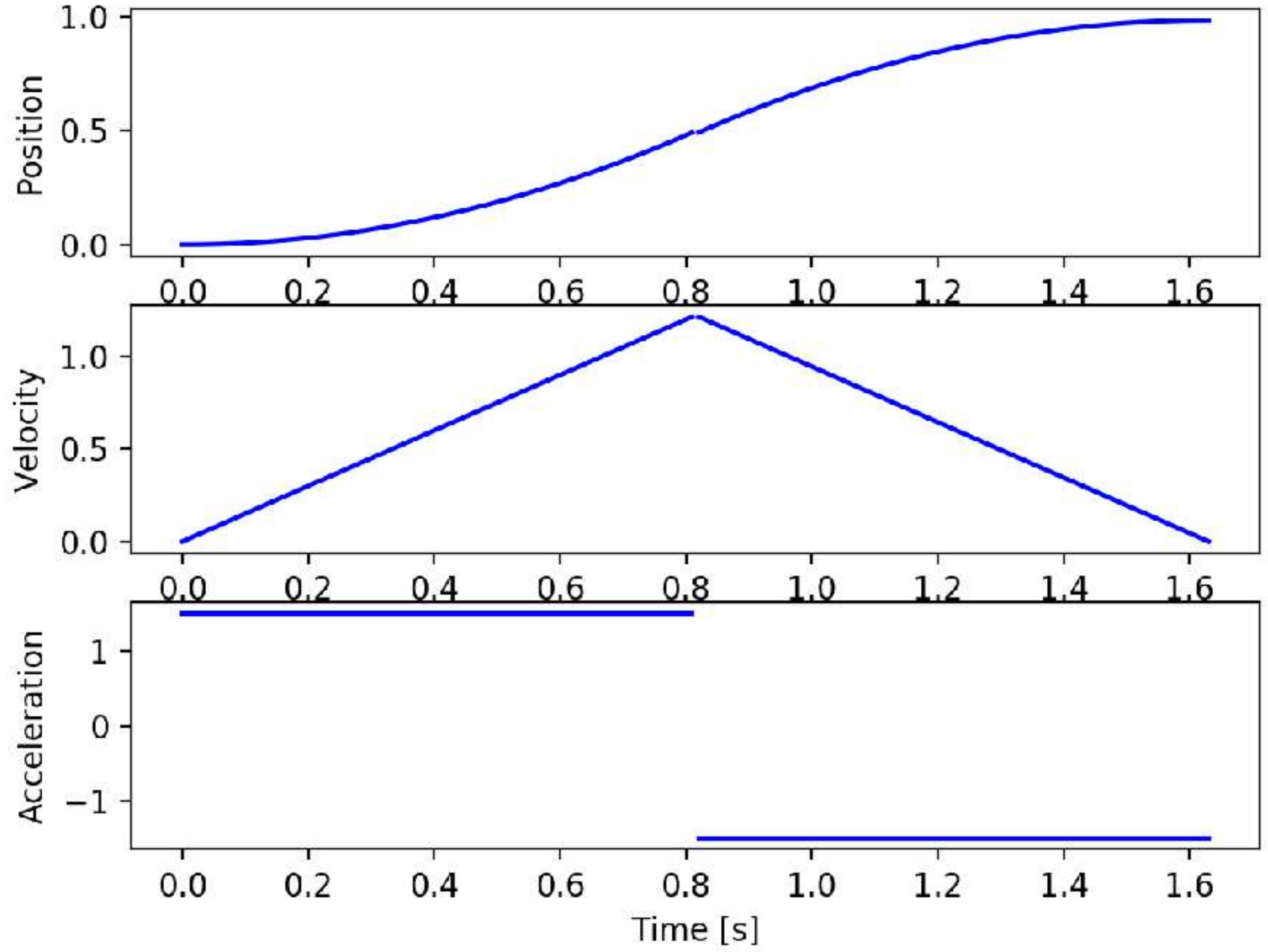




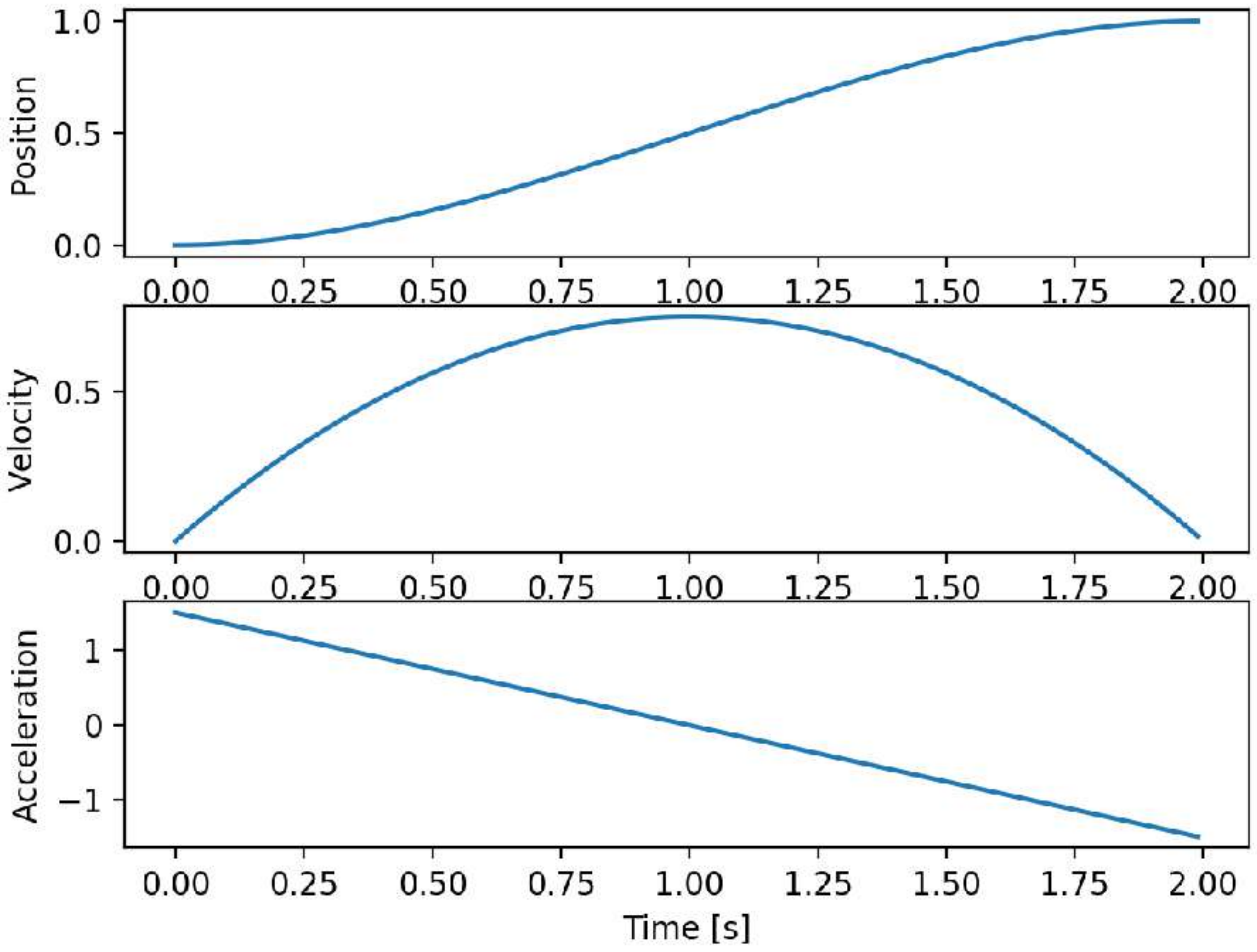
Goal
×

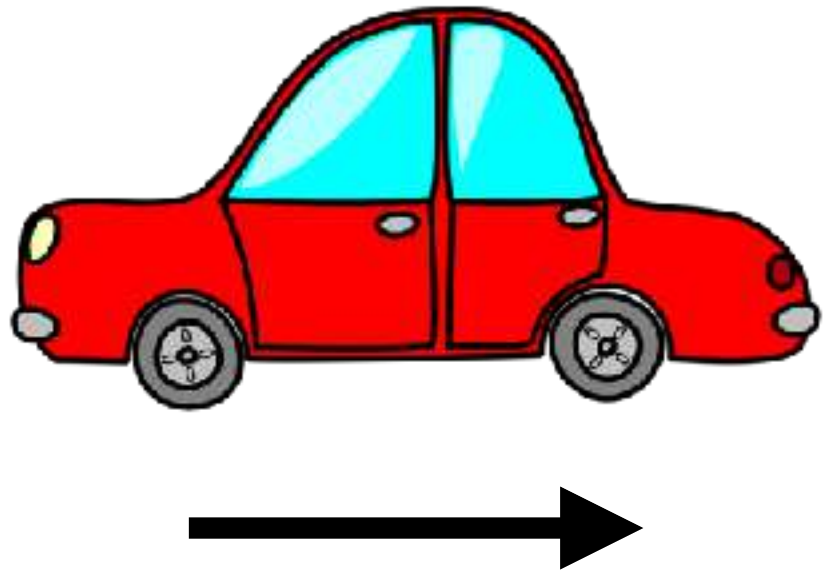
Different measures of "best" lead to very different answers

minimum time?



minimum acceleration?



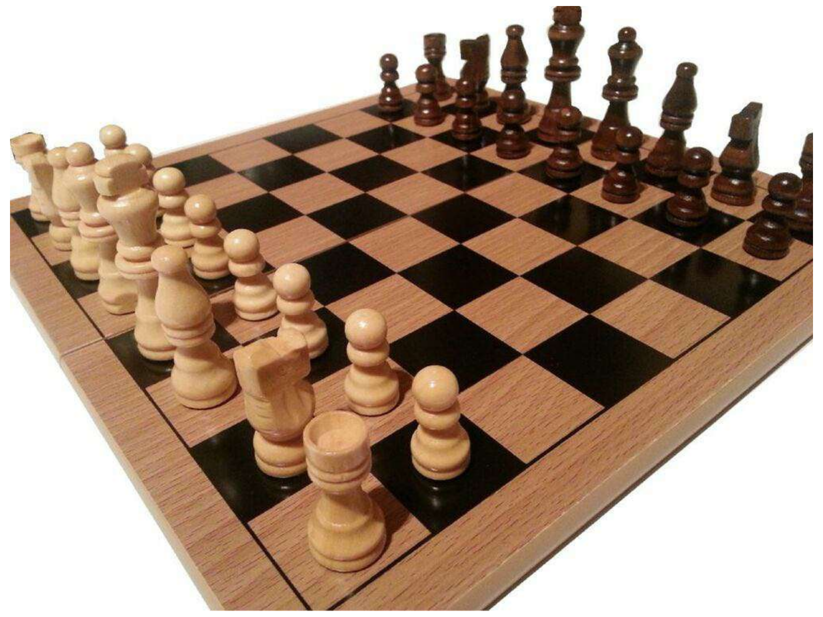


Goal
×

A sequential decision making problem

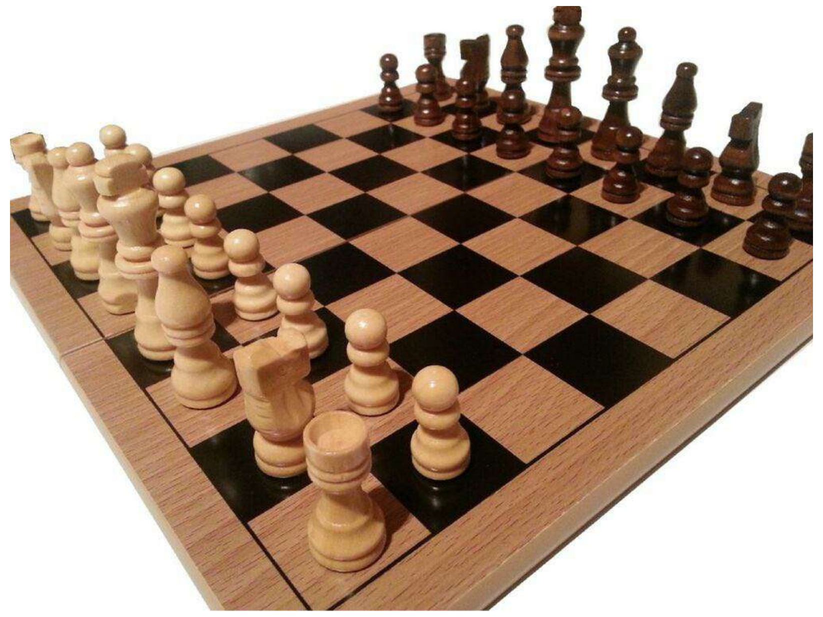
A control decision → changes the "state" of → the dynamical system

the dynamical system → A control decision



The problem: find the "**best sequence of actions**" to make the **dynamical system** behave as desired (e.g. win the game or do a backflip)

Dynamical systems are all over the place




The state of a dynamical system changes according to some “rules”

- The rules of the game and the opponent moves (chess)
- Newton’s law of motion (robot)
- Energy production rate and consumption (smart grid)
- etc

Robots are continuous time dynamical systems

Discrete vs. continuous time dynamical systems

Discrete vs. continuous time dynamical systems

A first optimal control problem

Structure of an optimal control problem

$$\boxed{\min_{u_0, u_1, \cdots, u_N} \sum_{i=0}^{N} g_i(x_i, u_i)}$$

Find actions that optimize a performance cost

Subject to:

$$\boxed{\begin{array}{l} x_{n+1} = f(x_n, u_n) \\ \\ h_n(x_n, u_n) \leq 0 \end{array}}$$

to control a dynamical system (maybe with constraints)

This is an optimization problem

Basics of optimization

Off-the-shelf optimization algorithms

SciPy

Fundamental algorithms for scientific computing in Python

GET STARTED

```
scipy.optimize.minimize(fun, x0, args=(), method=None, jac=None, hess=None,
hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None)
```

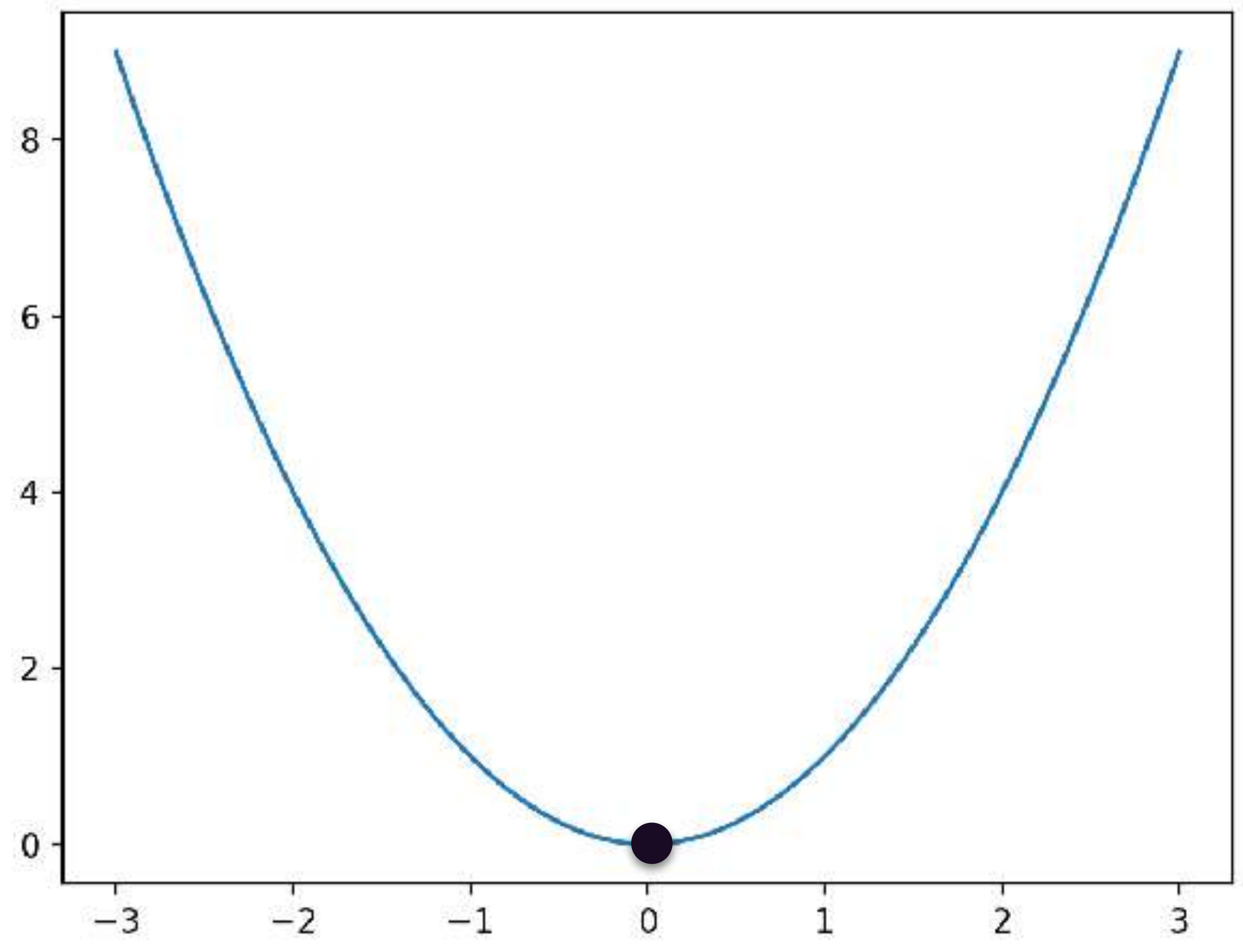
Optimization (`scipy.optimize`)

- Unconstrained minimization of multivariate scalar functions (`minimize`)
  - Nelder-Mead Simplex algorithm (`method='Nelder-Mead'`)
  - Broyden-Fletcher-Goldfarb-Shanno algorithm (`method='BFGS'`)
    - Avoiding Redundant Calculation
  - Newton-Conjugate-Gradient algorithm (`method='Newton-CG'`)
    - Full Hessian example:
    - Hessian product example:
  - Trust-Region Newton-Conjugate-Gradient Algorithm (`method='trust-ncg'`)
    - Full Hessian example:
    - Hessian product example:
  - Trust-Region Truncated Generalized Lanczos / Conjugate Gradient Algorithm (`method='trust-krylov'`)
    - Full Hessian example:
    - Hessian product example:
  - Trust-Region Nearly Exact Algorithm (`method='trust-exact'`)
- Constrained minimization of multivariate scalar functions (`minimize`)
  - Trust-Region Constrained Algorithm (`method='trust-constr'`)
    - Defining Bounds Constraints:
    - Defining Linear Constraints:
    - Defining Nonlinear Constraints:
    - Solving the Optimization Problem:
  - Sequential Least SQuares Programming (SLSQP) Algorithm (`method='SLSQP'`)
- Global optimization
- Least-squares minimization (`least_squares`)
  - Example of solving a fitting problem
  - Further examples
- Univariate function minimizers (`minimize_scalar`)
  - Unconstrained minimization (`method='brent'`)
  - Bounded minimization (`method='bounded'`)
- Custom minimizers
- Root finding
  - Scalar functions
  - Fixed-point solving
  - Sets of equations
  - Root finding for large problems
  - Still too slow? Preconditioning.
- Linear programming (`linprog`)
  - Linear programming example
- Assignment problems
  - Linear sum assignment problem example
- Mixed integer linear programming
  - Knapsack problem example

Minimizing a function

$$\min_x x^2$$

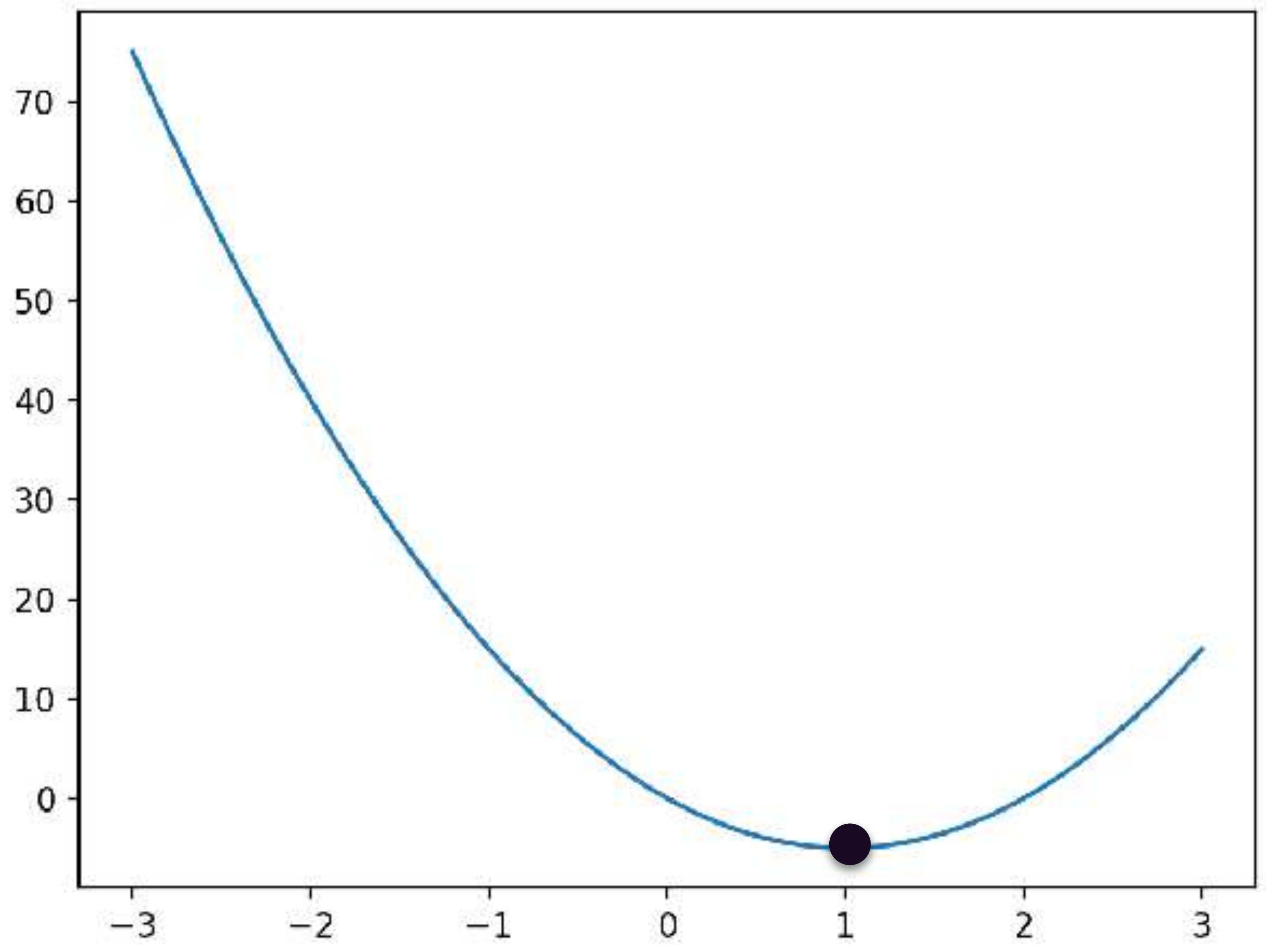
$$x^* = 0$$



Minimizing a function

$$\min_x 5x^2 - 10x$$

$$x^* = 1$$



Minimums (local and global)

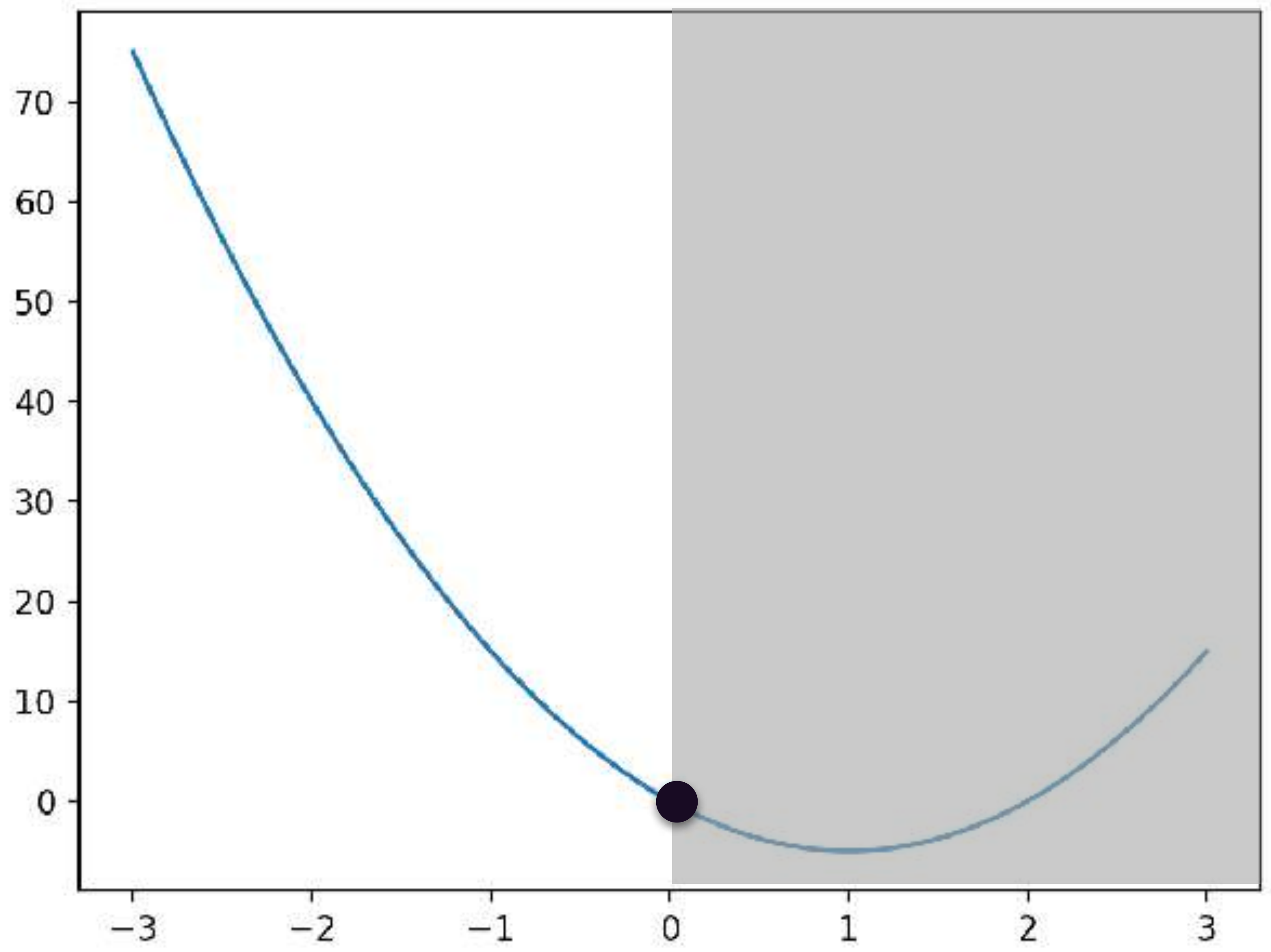
These conditions are not sufficient!

Optimization with constraints

$$\min_x 5x^2 - 10x$$

$$x < 0$$

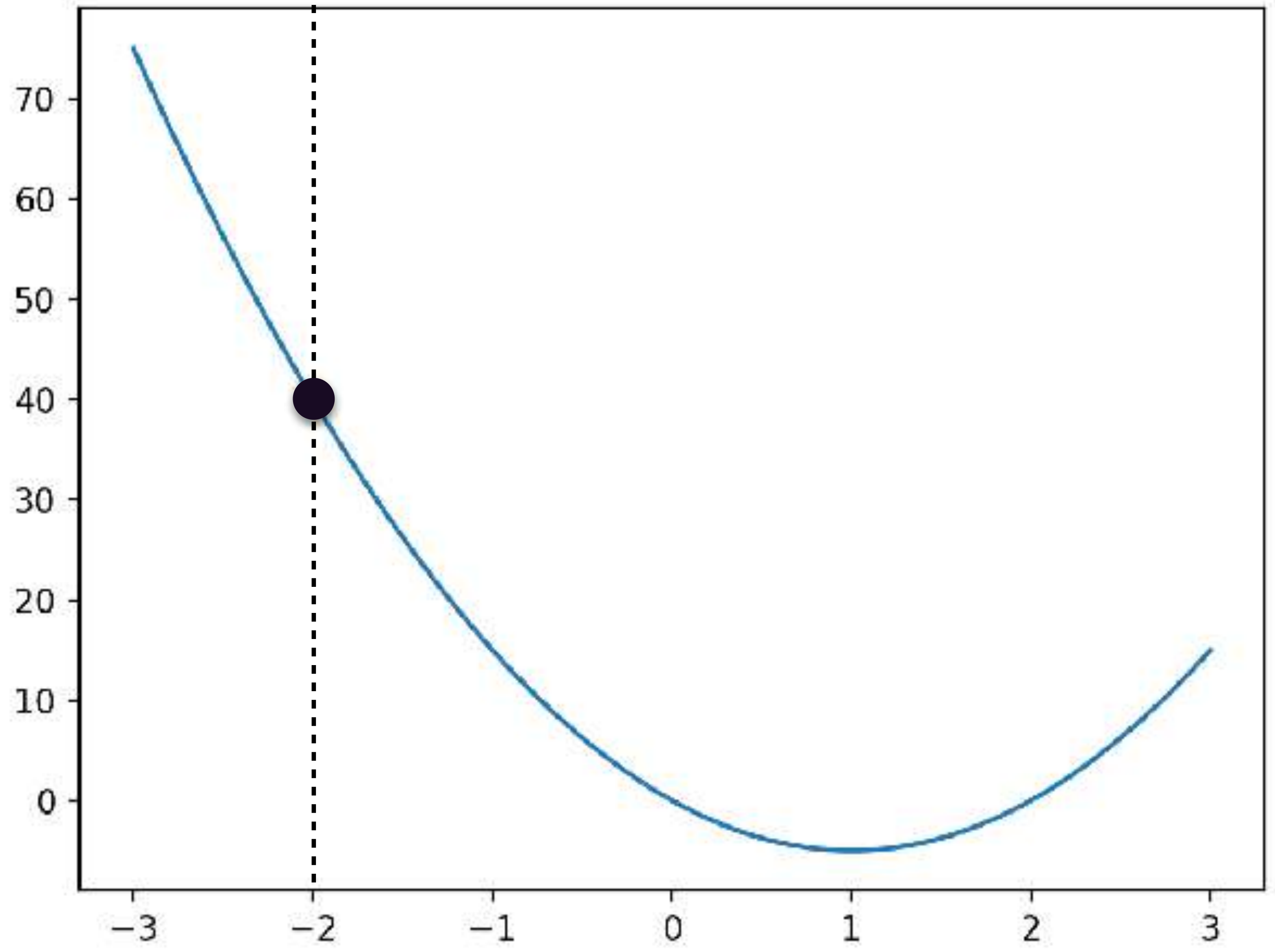
$$x^* = 0$$



Optimization with constraints

$$\min_x 5x^2 - 10x$$

$$2x + 4 = 0$$
$$x^* = -2$$

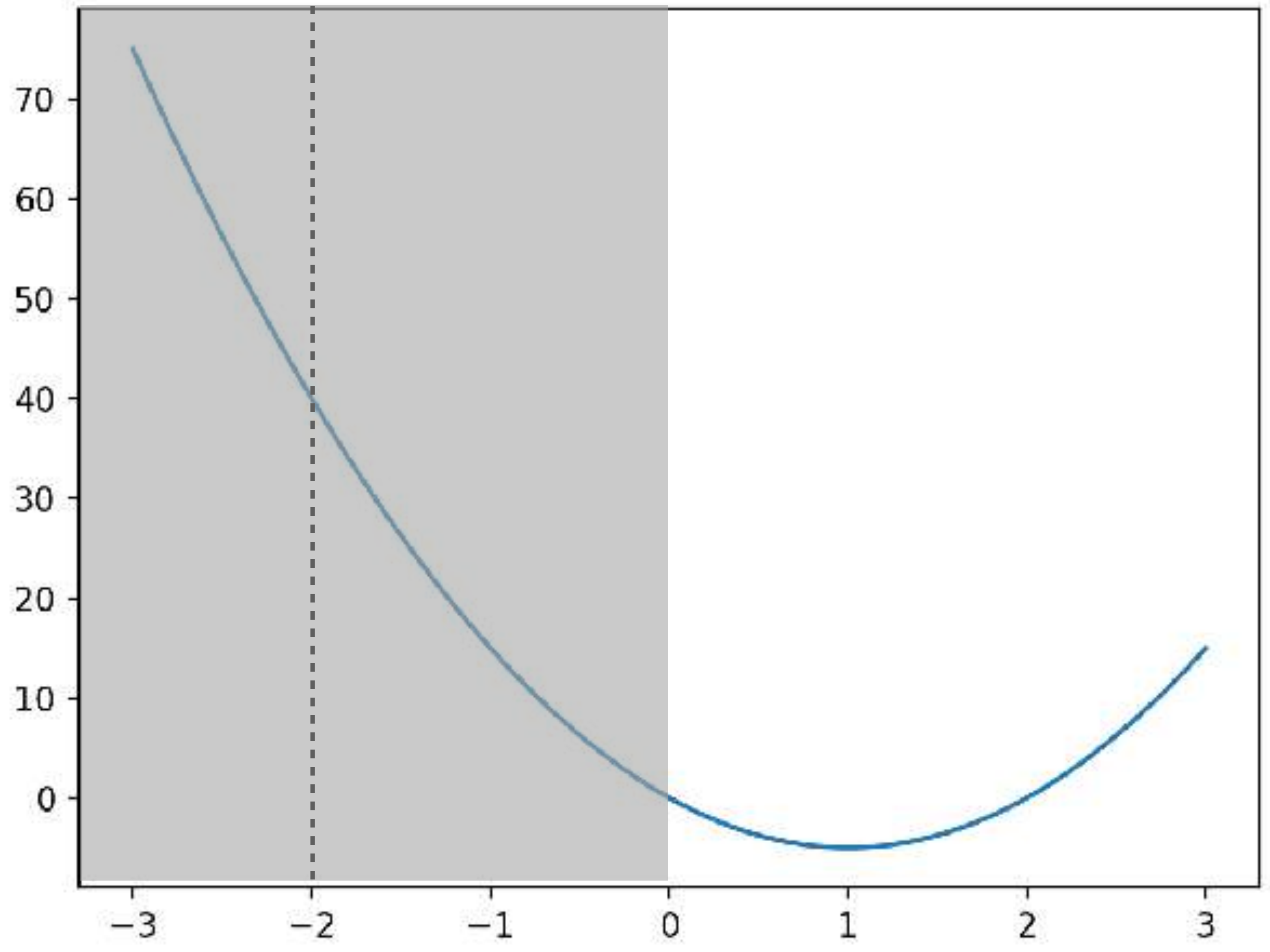


Optimization with constraints

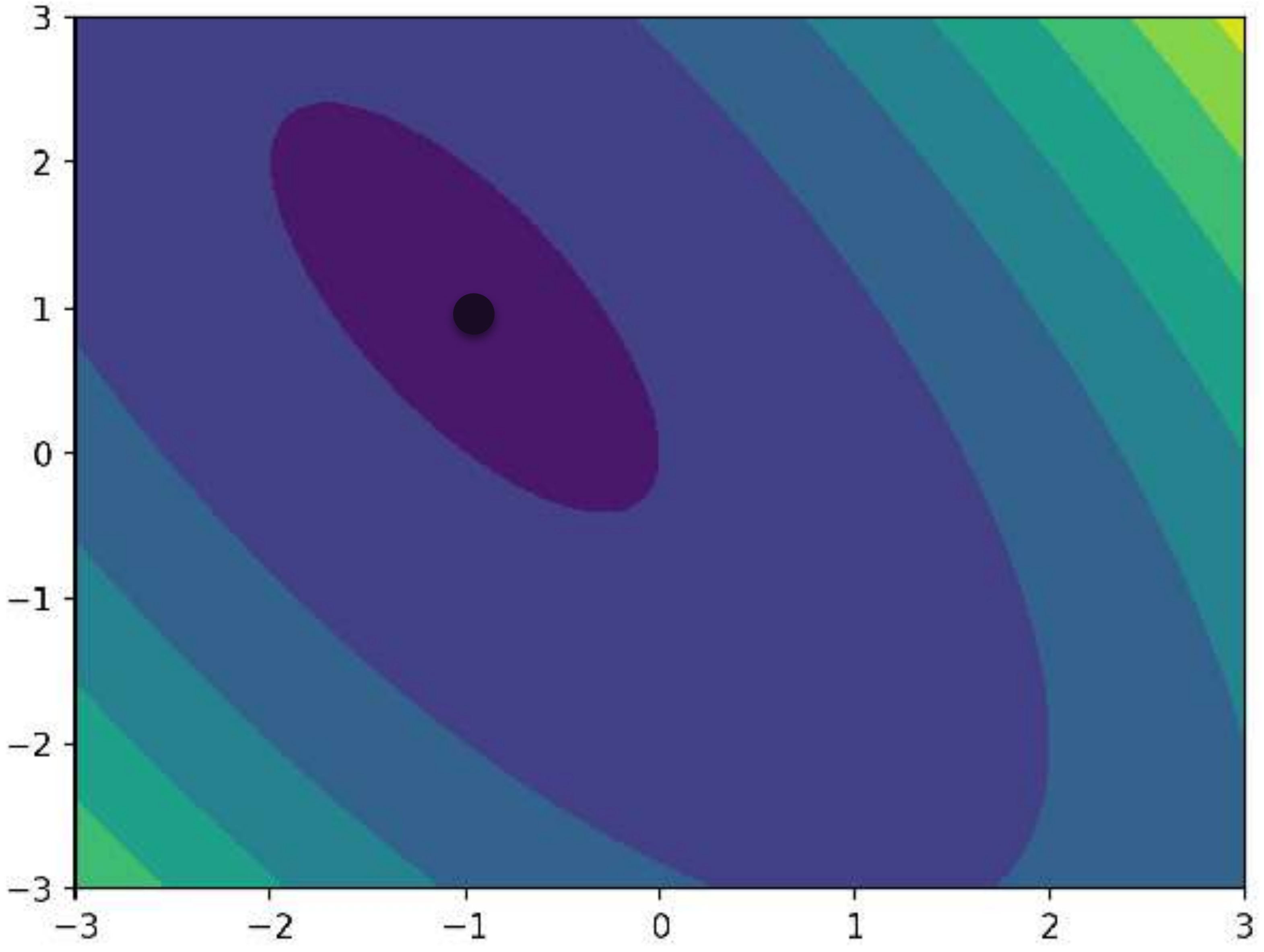
$$\min_x 5x^2 - 10x$$

$$2x + 4 = 0$$
$$-x < 0$$

Unfeasible

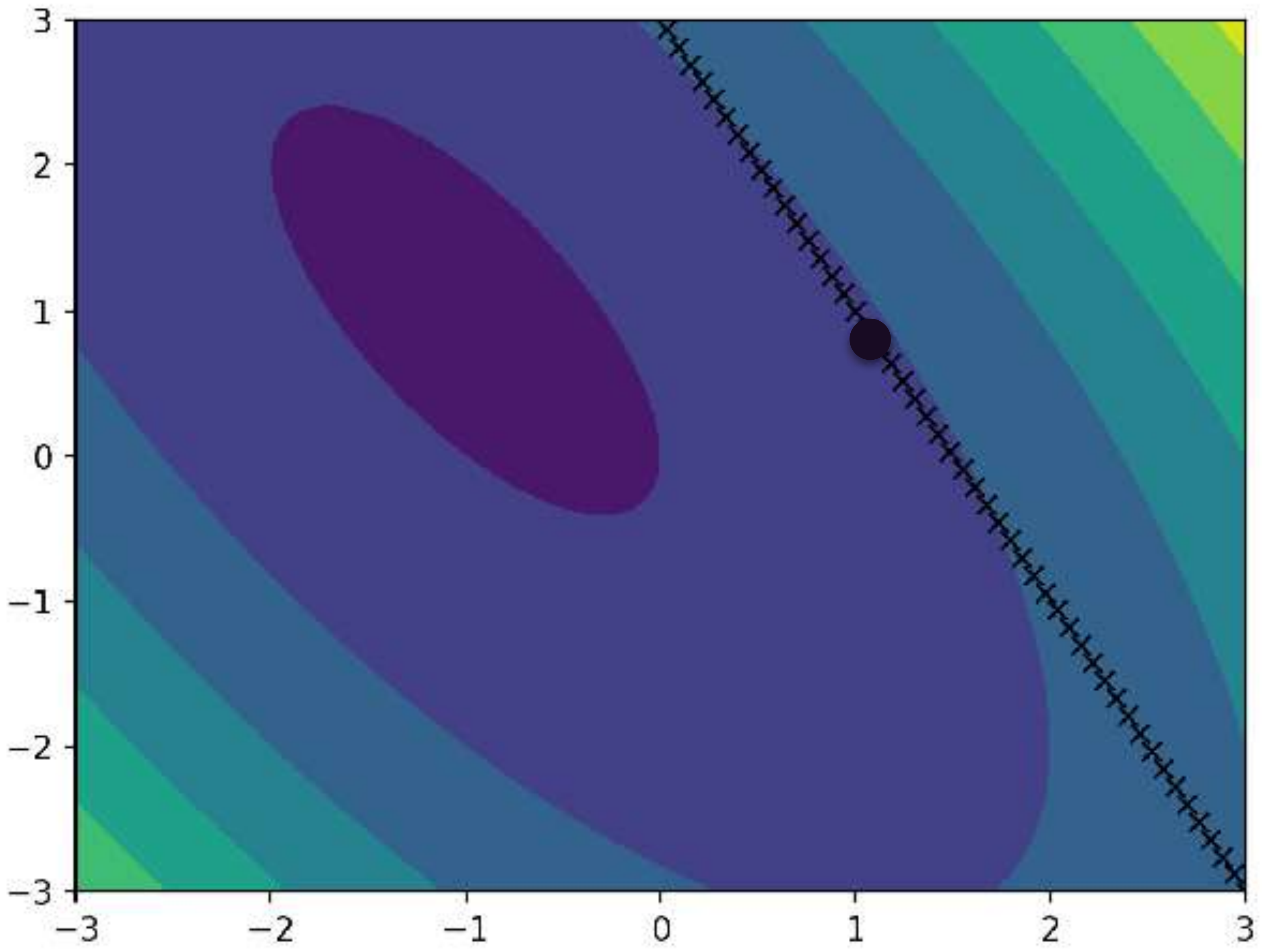


$$\min_{x_1,x_2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 2x_1$$



$$\min_{x_1,x_2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 2x_1$$

s.t. $x_1 + 2x_2 + 3 = 0$



$$\min_{x_1,x_2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 2x_1$$

s.t. $x_1 + 2x_2 + 3 = 0$

$x_1 < 0.5$