02565 Exam Instructions
Version 0.1

Tue Herlau
tuhe@dtu.dk

June 18, 2025

1 General information

Evaluation form: The evaluation is graded on the 7-step scale. The grade is determined by an overall assessment of your project work and a written exam. The project work will be weighted at approximately one fifth and the written exam at approximately four fifths.

Project: Details about the project can be found through DTU Learn.

Questions: Please do not hesitate to contact me by email tuhe@dtu.dk or Discord if you want clarification of any questions you may have about the exam (instructions, hand-in format, etc.).

2 The written exam

Form: The four hour written exam consist of pen-and-pencil questions and programming questions. The programming questions will require you to write code. Therefore, please bring a computer that has been set up to run the exercise code beforehand.

Previous exams: The exam from last year will be uploaded during the semester. I will also upload two so-called midterms, but taken together these are slightly longer than a single exam.

Hand-in and scoring: The hand-in and scoring procedure is described on the front page of the previous years exam. Please read this carefully before the exam and carefully check that you can generate hand-ins according to this instruction on your computer.

Aids: All normal aids are allowed for the exam (i.e., books, notes and computers), but **not** open internet. This means your computer must be set up correctly before you attend the exam.

When/where: The exam location may differ for students enrolled on the same course. You can find the exam schedule here: `https://www.dtu.dk/uddannelse/kursusbasen/eksamensskema`. You can check your exam enrollments on exam location and times on `https://eksamensplan.dtu.dk`. Check your exam enrollments well before the exam.

3 Syllabus

The course syllabus is the reading material from [Her25, SB18], the exercises, projects, course toolbox, documentation and lecture slides. It is important you have the toolbox installed and working on your computer.

4 Terminology

The conceptual questions will typically ask you to compute a number or a function. In those problems, a correct answer is an answer which has been *reasonably* simplified, and often this is emphasized in the problem text using words such as *simple* or *closed-form* expression.

For instance, consider this physics problem:

- Suppose that water is drained from a barrel at a rate of $f(t) = \frac{1}{t}$ where t is the time. Determine a simple expression for how much water is drained from time $t = 1$ to $t = T$?

In this case the answer can be found as $\int_1^T \frac{1}{t} dt$, however, simply writing the integral itself will not get full credit since it has not been evaluated – i.e., the expression is not simple in the sense it cannot be readily implemented in a computer. The right answer (which will get full credit) is $\log T$ (the value of the integral).

The reason is that the solution to many problems can often be stated in an abstract way, for instance by re-stating the dynamical programming algorithm or a Bellman equation, and the purpose of the exam is to test if you can apply these equations to a concrete situation. This is best done by computing a reduced (simplified) answer. To emphasize this, I will often use words such as *simple* or *closed form* to indicate that you should insert relevant constants into the expressions, evaluate integrals, and otherwise provide a reasonably simplified expression.

What I *don't* care that much about is symbol manipulation: Some will say that $\frac{1}{\sqrt{2}}$ should be written as $\frac{\sqrt{2}}{2}$, or that one should always remember that $\cos\frac{\pi}{4}$ is really $\frac{1}{\sqrt{2}}$, or even that $\frac{9}{2}$ should be written as $4 + \frac{1}{2}$ (or visa-versa). Don't worry about it.

Numerical value Some questions will ask you for numerical values. In these cases you cannot give the answer using symbols (such as k^2 etc.), but should rather provide a number, for instance 5.3. Sometimes I might ask you to write both a function $f(x)$ and also provide a numerical value such as $f(2)$. The reason for this is not to double your work, but rather to let *you* know that the function is really just a concrete function of x (such as $f(x) = 2 + x^2$) that readily allows you to compute that e.g. $f(2) = 6$. By asking this way I hope this can avoid certain problems such as forgetting to insert constants in an expression and so on.

5 Tips for preparing for the exam

Although all subjects in the course are exam relevant and may be the subject of questions, I think it is helpful to provide a list of subjects that perhaps deserve a second glance. Note most of these subjects are already covered by the two midterms/previous years exam.

Understand in depth: Understand the notation and mathematics behind these subjects to the degree you can apply them to simple examples step-by-step and intuitively understand what they do and reason about them.
From the dynamical programming section:

- The DP problem formulation f_k, g_k, action and state spaces, etc.
- Tail-cost functions $J_{\pi,k}$ and optimality
- The DP algorithm; be able to reason about it and apply it to simple problems by hand.

From the control section

- Linear-quadratic problems (discrete and continuous-time; both dynamics and cost, i.e. A_k, B_k etc. and Q_k, R_k etc.)
- Discretization of control problems (Euler and exponential; includes that you can translate a system of differential equations to a control problem as in Midterm A, q6)
- Discrete LQR control
- Linearization of control problems around a point $\bar{x}, \bar{u}$
- PID control, hereunder application to simple problems such as car steering, pendulum-balancing, or the harmonic oscillator. Understand the role of $\boldsymbol{x}^*$, K_p, K_i and K_d and how they are used to control the action.
- Trapezoid collocation for direct control

From the RL section

- Bandits
  - What a bandit problem is. For instance, the 10-armed test-bed in [SB18, Chapter 2].
  - The simple bandit algorithm
  - The simple bandit for a non-stationary environment, i.e. using a learning rate α
  - The UCB-bandit algorithm

- MDPs
  - How the MDP is formulated mathematically ($p(s', r|s, a)$ etc.)
  - The definitions of key quantities such as v_π, q_π, v^*, q^* etc.
  - Reason about the behavior of quantities such as v_π etc. for different problem types, learning rates, etc.
  - Have a clear understanding of what the Bellman equations mean to the point where you can translate simple problems into equations (c.f. the Jar-Jar problem in part 3 of the project)
  - MDPs without actions (such as the Sarlac example from project 3, what Sutton call a Markov Reward Process)
  - Translate a general description of a simple MDP in terms of a diagrams/graphs and/or transition probabilities into a mathematical form where you can reason about them (c.f. example 6.2, example 6.4 or exercise 3.22)
- The basic Gridworld example (i.e., the black gridworld with pacman I live-demo in many of the lectures). Understand what it shows and reason about how the different values will change when we vary α, ϵ, γ, etc. while using different algorithms such as:
  - Sarsa
  - Q-learning
  - Dynamical programming algorithms (such as Value-iteration, policy-evaluation, etc.)
  - MC-learning (first visit and every-visit)
  - Tabular TD-lambda.

  Pay particular attention to the case where we are given a reward when we transition to the terminal state (and otherwise zero reward). The exam examples folder contains several demos you can look at.
- Off and on-policy, terminating vs. non-terminating environments, etc.
- Epsilon-greedy exploration
- Key algorithms (what do they do, what happens if you run them long enough and they converge, which quantities q_π, q^* can they compute):
  - TD0
  - Sarsa
  - Q-learning
  - Dynamical programming algorithms (such as Value-iteration, policy-evaluation, etc.)
  - MC-learning (first visit and every-visit)
  - Tabular TD-lambda.

Understand well enough to program/work with in code From the DP section

- The DP algorithm and related subjects, for instance how we represent a policy as a dictionary etc.
- Operations such as computing the expected value $\sum_x p(x)f(x)$ when $p(x)$ is represented as a dictionary etc.
- The DP model class, in particular implementing a problem using this class and then use the functions/functionality the class provides for other tasks
- The DP algorithm (you worked on both subjects in project 1)

From the control section

- The PID algorithm
- Linearization around a point
- Simulate a control problem with RK4 using e.g. the toolbox
- LQR
- Simple bandit algorithms and non-stationary variants (ϵ-greedy)

From the RL section

- Bellman-equation inspired algorithms such a Value-iteration, policy evaluation, etc.
- The simple RL algorithms such as TD0, Sarsa and Q-learning
- MDPs, in particular implementing a problem using the MDP problem class and then use the functions/functionality the class provides for other tasks
- Implement a problem using the MDP class and apply tabular methods to it (value iteration, policy iteration, policy evaluation)

Programming examples Refresh how these work so you can use them and potentially implement your own variants:

- The Inventory DP model
- The Chessmatch DP model
- The Pendulum control model
- The Harmonic oscillator control model
- Simple MDP examples such as the Gambler, Gridworld (the basic variant included in the week 9 exercises), etc.

References

[Her25] Tue Herlau. Sequential decision making. (Freely available online), 2025.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. (Freely available online).