TDT4171 Artificial Intelligence Methods

Lecture 6 – Making Complex Decisions

Norwegian University of Science and Technology

Helge Langseth
Gamle Fysikk 255
helge.langseth@ntnu.no

NTNU

1. **Reference group meeting *fairly* soon**
Help out by sending them some comments, e.g., propose three ways to improve the course.

2. **Background for ML-part**
Next week will summarize stuff you need to know. Just a recap if you have already taken "TDT4172 Intro to ML". Will cover all you need for this course.

3. **I want to talk about the assignment**
More details on separate slide-set (`Oving.pdf`)

Summary from last week

- Rational agents can always use **utilities** to make decisions
- The **MEU principle** tells us how to behave
- It can be quite laborious to elicit preference structures from domain experts
  - $\Rightarrow$ **structured approaches** are available
- **Value of Information** helps focus information gathering for rational agents
- **Decision Networks/Influence diagrams** are extensions to BNs that let us make rational decisions.

Chapter 16 – Learning goals

Understanding the relationship between

- One-shot decisions
- Sequential decisions
  - Decisions for finite horizon
  - Decisions for infinite horizon

Being familiar with:

- Markov Decision Processes
- Value iteration

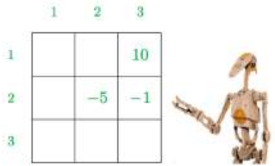
Know about:

- Partially Observable Markov Decision Processes
- Policy iteration

Decision problems with an unbounded time horizon

Examples of decision problems with an unbounded time horizon:

- Fishing in the North Sea.
- Playing poker.
- Robot navigation – find path from current position to a certain goal position.



Decision problems with an unbounded time horizon

Examples of decision problems with an unbounded time horizon:

- Fishing in the North Sea.
- Playing poker.
- Robot navigation – find path from current position to a certain goal position.

	1	2	3
1			10
2		-5	-1
3			

Characteristics:

- at each step we are faced with the same type of decision,
- at each step we are given a certain reward (possibly negative) determined by the chosen decision and the state of the world,
- the outcome of a decision may be uncertain,
- the time horizon of the decision problem is unbounded.

How can we solve problems like these?

Characteristics:

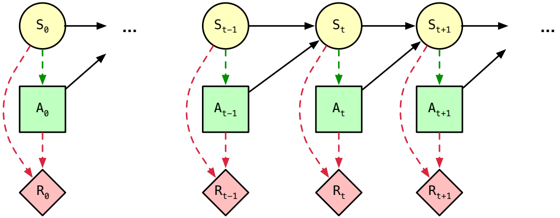
- at each step we are faced with the same type of decision,
- at each step we are given a certain reward (possibly negative) determined by the chosen decision and the state of the world,
- the outcome of a decision may be uncertain,
- the time horizon of the decision problem is unbounded.

Formalization:

- The state of the world at time t is S_t, which we **can observe**.
- At each time t we choose an action A_t.
- We get a state/action history $\sigma_t = \{S_0, A_0, S_1, A_1, \ldots, S_t\}$.
- We want a $\pi_t(\sigma_t)$ such that $\pi_t(\sigma_t)$ gives A_t for any "input" σ_t.

Can this be modelled as a decision network?
What assumptions are useful (or needed)?
Discuss with your neighbour for a couple of minutes...

Answer: Markov Decision Process



Our end-result will be a **Markov Decision Process**.

To get this model we assume **observability**, the **Markov property**, **stationarity**, and **additive rewards**.

Anyway, before we get there and understand what it all means, there are a number of details we need to consider. . .

What are we trying to obtain?

To solve these decision-problems we need...

- A "mapping" from any state/action history $\sigma_t = \{S_0, A_0, S_1, A_1, S_2, A_2, \ldots, S_t\}$ to A_t (the next action).
  - $\pi_t(\sigma_t) = a_t$ means "*If you've seen* σ_t *at* t*, then do* a_t".
  - We want to simplify – e.g., to have $\pi_t(\sigma_t) = \pi_t(S_t) = \pi(S_t)$.

	1	2	3
1	$\rightarrow$	$\rightarrow$	$\times$
2	$\uparrow$	$\uparrow$	$\uparrow$
3	$\uparrow$	$\leftarrow$	$\uparrow$

Possible representation of $\pi(S_t)$ in "robot-domain"

- We need **Markov** + **Stationarity** assumptions. **Optimal?**
- What about the **MEU** principle we discussed last time?

What are we trying to obtain?

To solve these decision-problems we need...

- A "mapping" from any state/action history $\sigma_t = \{S_0, A_0, S_1, A_1, S_2, A_2, \ldots, S_t\}$ to A_t (the next action).
  - $\pi_t(\sigma_t) = a_t$ means "*If you've seen* σ_t *at* t*, then do* a_t".
  - We want to simplify – e.g., to have $\pi_t(\sigma_t) = \pi_t(S_t) = \pi(S_t)$.
- We can think of this in two steps:
  1. Find a **utility function** $U_t^*(\sigma_t) = U_t^*(S_t)$ representing how good it is to have done σ_t and end up in S_t.
  2. Define π_t so that it **maximizes the expected utility** at each step

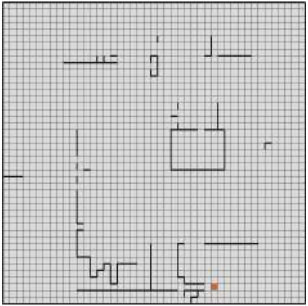
	1	2	3
1	6.2	7.5	10
2	4.7	1.1	6.5
3	3.9	4.0	5.3

Possible repr. $U_t^*(S_t)$

	1	2	3
1	→	→	×
2	↑	↑	↑
3	↑	→	↑

Corresponding $\pi_t(S_t)$

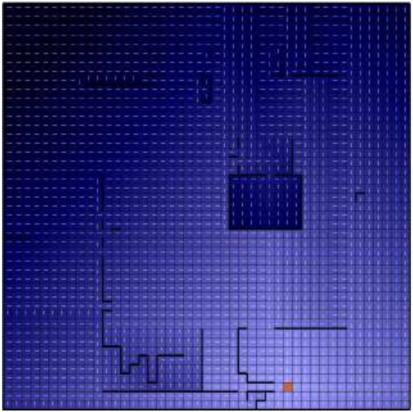
Relation to Planning



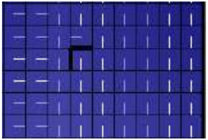
Solving such problems relate to solving a ***planning task***.
Find shortest path in maze – Golden square is goal.

How can we represent this? How to quantify $U_t^*(\sigma_t)$?
Are Markov + Stationarity assumptions OK?
Discuss with your neighbour for a couple of minutes...

Relation to Planning

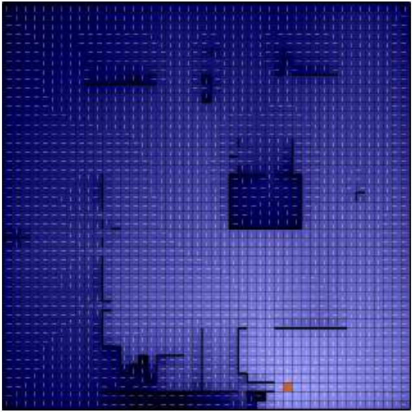


Detail: Near wall behaviour

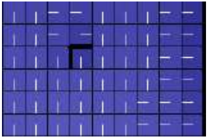


Solution – Ordinary planning. Something like
$R(s_t, a_t) = R(s_t) = -1$ unless goal-state, where $R(s_t) = +100$.
Utility can be the sum of all future rewards under optimal policy!

Relation to Planning



Detail: Near wall behaviour



Solution – Stochastic planning

Robot may fail to do correct action.

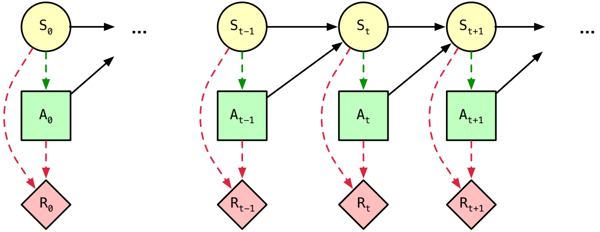
Penalty $R(s_t) = -50$ if bouncing off a wall.

Markov Decision Processes (MDPs)

The robot navigation problem can roughly be described as a loop:

1. Observe the state of the world,
2. Collect (possibly negative) reward R_t (not the same as U_t^*!),
3. Decide on the next action A_t and perform it.

This can be modelled as a **Markov decision process**:



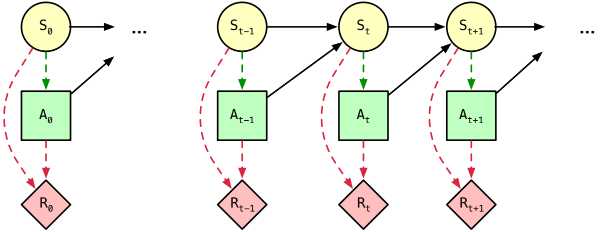
Note! The model adheres to the **Markov assumption**. In particular, only S_t is needed from σ_t to find next action: $\pi_t(\sigma_t) = \pi_t(S_t)$.

Markov Decision Processes (MDPs)

The robot navigation problem can roughly be described as a loop:

1. Observe the state of the world,
2. Collect (possibly negative) reward R_t (not the same as U_t^*!),
3. Decide on the next action A_t and perform it.

This can be modelled as a **Markov decision process**:



Note also! R_t is determined by A_t and S_t. However, as $R_t(s_t, a_t) = R_t(s_t)$ in the robot example I simplify the equations accordingly.

The quantitative part of the MDP

In order to specify the **transition probabilities** $P(S_t \mid A_{t-1}, S_{t-1})$ and the **reward function** $R(S_t)$ we need some more information about the domain:

- The robot can move `north`, `east`, `south`, and `west`.
- For each move there is a fuel expenditure of 0.1, unless we fall into one of the holes (giving values -1 or -5) or we reach the goal-state (reward $+10$).
- A move succeeds with probability 0.7; otherwise it moves in one of the other directions with equal probability.
- If it walks into a wall, the robot effectively stands still.

Rewards per position:

	1	2	3
1	-0.1	-0.1	10
2	-0.1	-5.0	-1.0
3	-0.1	-0.1	-0.1

The quantitative part of the MDP – cont'd

This gives the **transition probabilities** (for $P(S_{t+1} \mid \texttt{north}, S_t)$):

		S_t								
		$\{1,1\}$	$\{2,1\}$	$\{3,1\}$	$\{1,2\}$	$\{2,2\}$	$\{3,2\}$	$\{1,3\}$	$\{2,3\}$	$\{3,3\}$
	$\{1,1\}$	0.8	0.7	0	0.1	0	0	0	0	0
	$\{2,1\}$	0.1	0.1	0.7	0	0.1	0	0	0	0
	$\{3,1\}$	0	0.1	0.2	0	0	0.1	0	0	0
	$\{1,2\}$	0.1	0	0	0.7	0.7	0	0	0	0
S_{t+1}	$\{2,2\}$	0	0.1	0	0.1	0	0.7	0	0.1	0
	$\{3,2\}$	0	0	0.1	0	0.1	0.1	0	0	0.1
	$\{1,3\}$	0	0	0	0.1	0	0	1	0.7	0
	$\{2,3\}$	0	0	0	0	0.1	0	0	0.1	0.7
	$\{3,3\}$	0	0	0	0	0	0.1	0	0.1	0.2

We say that $\{1,3\}$ is **absorbing** if for all A_t

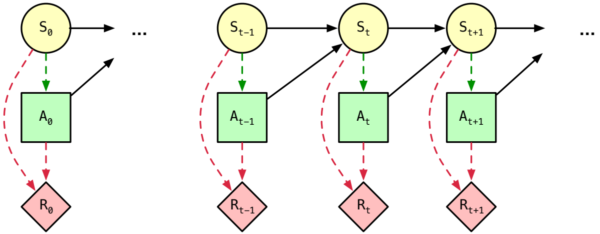
$$P(\{1,3\} \mid A_t, \{1,3\}) = 1.$$

	1	2	3
1	$\{1,1\}$	$\{1,2\}$	$\{1,3\}$
2	$\{2,1\}$	$\{2,2\}$	$\{2,3\}$
3	$\{3,1\}$	$\{3,2\}$	$\{3,3\}$

MDPs in general

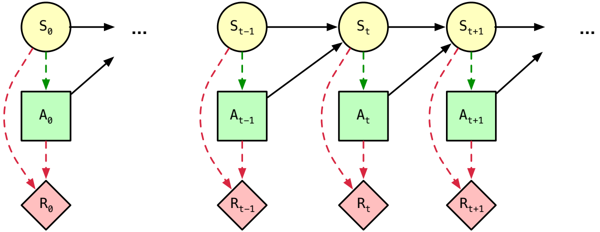
In general, in a Markov decision process . . .

- The world is **fully observable**,
- Any uncertainty in the system is due to **non-deterministic actions**
- For each decision we get a **reward** (which may be negative); may depend on current world state and chosen action, but is independent of time (stationarity of reward-model).



Decision policies

A decision policy for A_t is in general a function over the entire past, $\{S_0, A_0, S_1, A_1, S_2, A_2, \ldots, S_t\}$.

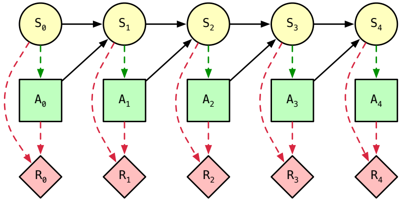


However, from the conditional independence properties we see that the **relevant** past is reduced to S_t.

Note! While we do not consider S_{t-1} when choosing A_t, we **do** think about the future (S_{t+1}, A_{t+1}, S_{t+2}, A_{t+2}, $\ldots$).
This is similar to **prediction** in the dynamic models.

Types of strategies: a bounded time horizon

The (approximated) North Sea fishing example over a **five year period**; S_t is amount of fish in the sea at the start of time period t, A_t is the amount being fished at during period t.



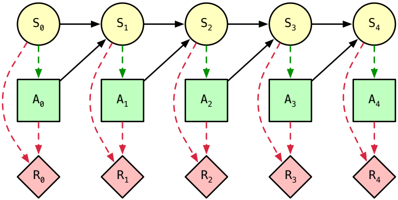
What we want to understand with this example:

Strategy is **Markovian**: $\pi_t(\sigma_t) = \pi_t(S_t)$.

What will it take to guarantee that it is **stationary**, $\pi_t(S_t) = \pi(S_t)$, as well?

Types of strategies: a bounded time horizon

The (approximated) North Sea fishing example over a **five year period**; S_t is amount of fish in the sea at the start of time period t, A_t is the amount being fished at during period t.



Even if S_0 and S_4 are the same state:

- At $t = 0$ we may specify a conservative number to ensure that there is enough fish in the coming years.
- At $t = 4$ we have no concerns about the future, and catch as much as we can.

$\Longrightarrow$ **The optimal policy for A_t depends on the time t!**

Length of horizon vs. Optimal strategy

Optimal strategy changes as the the time-horizon increases:
Consider the robot navigation task with the add-on that the game ends when the goal is reached or k time-steps have passed.

	1	2	3
1	-0.1	-0.1	10
2	-0.1	-5	-1
3	-0.1	-0.1	-0.1

$R(S_t)$

	1	2	3
1	→	→	×
2	↑	↑	↑
3	←	→	↑

$k = 3$

	1	2	3
1	→	→	×
2	↑	↑	↑
3	↑	←	←

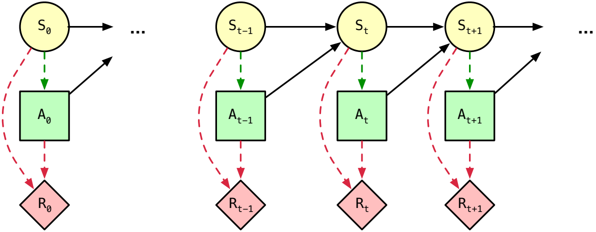
$k \geq 6$

- For $k = 3$ and start position $\{3, 2\}$ we have to accept the penalty of -1 in $\{2, 3\}$ to make it to the goal state on time.
- For larger k we have time to take the long route.

$\Longrightarrow$ **Non-stationarity again!**

Types of strategies: an unbounded time horizon

The (approximated) North Sea fishing example with an **unbounded** time horizon:



The optimal policy for A_t depends on the current state and what may happen in the future. If two time steps, say year $t = 0$ and $t = 4$, are in the same state then they have the same possibilities in the future.

$\Longrightarrow$ The optimal policies for A_0 and A_4 are the same!

The strategy is said to be **stationary** if $\pi_t(S_t) = \pi(S_t)$.

Evaluating strategies with unbounded time horizons

Assume that the reward function is specified as:

	1	2	3
1	−0.1	−0.1	10
2	−0.1	−5	−1
3	−0.1	−0.1	−0.1

Imagine there is no terminal state and no uncertainty on the result of an action. Then:

$$U\left(\begin{array}{|c|c|c|}\hline \rightarrow & \rightarrow & \times \\ \hline \uparrow & \uparrow & \uparrow \\ \hline \uparrow & \rightarrow & \uparrow \\ \hline \end{array}, S_0 = \{3,3\}\right) = U\left(\begin{array}{|c|c|c|}\hline \rightarrow & \rightarrow & \times \\ \hline \uparrow & \downarrow & \leftarrow \\ \hline \uparrow & \leftarrow & \uparrow \\ \hline \end{array}, S_0 = \{3,3\}\right) = \infty$$

But which one is better?

The utility of an unbounded sequence: discounted rewards

Weigh rewards in the immediate future higher than rewards in the distant future:

$$U(s_0, s_1, s_2, \ldots) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots ,$$

where $0 \leq \gamma \leq 1$.

The utility of an unbounded sequence: discounted rewards

Weigh rewards in the immediate future higher than rewards in the distant future:

$$U(s_0, s_1, s_2, \ldots) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots ,$$

where $0 \leq \gamma \leq 1$.

Possible **interpretations** of the **discounting factor** γ:

- In economics, γ may be thought of as an interest rate of $r = (1/\gamma) - 1$.
- The decision process may terminate with probability $(1 - \gamma)$ at any point in time, e.g. the robot breaking down.

The utility of an unbounded sequence: discounted rewards

Weigh rewards in the immediate future higher than rewards in the distant future:

$$U(s_0, s_1, s_2, \ldots) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots ,$$

where $0 \leq \gamma \leq 1$.

- For $\gamma = 0$ we have a greedy strategy.
- With $0 < \gamma < 1$ and $\tilde{R} = \max_t R(s_t) < \infty$ we have

$$U(s_0, s_1, s_2, \ldots) = \sum_{i=0}^{\infty} \gamma^i R(s_i) \leq \sum_{i=0}^{\infty} \gamma^i \cdot \tilde{R} = \frac{\tilde{R}}{1-\gamma} < \infty.$$

- For $\gamma = 1$ we have normal additive rewards, quite possibly leading to infinite-valued $U(s_0, s_1, s_2, \ldots)$.

Uncertainty – Use expected utilities

The actions may be non-deterministic so a strategy may only take you to a state with a certain probability.

$$\Downarrow$$

Strategies should be compared based on the **expected** accumulated rewards they can produce – we follow **MEU principle**.

Uncertainty – Use expected utilities

The actions may be non-deterministic so a strategy may only take you to a state with a certain probability.

$$\Downarrow$$

Strategies should be compared based on the **expected** accumulated rewards they can produce – we follow **MEU principle**. Starting in s_0 and following π, the **expected discounted reward in step i** is:

$$\gamma^i \cdot \mathbb{E}\left[R(S_i) \mid \pi, S_0 = s_0\right] = \gamma^i \sum_{s_i} R(s_i) P(S_i = s_i \mid \pi, S_0 = s_0)$$

Uncertainty – Use expected utilities

The actions may be non-deterministic so a strategy may only take you to a state with a certain probability.

$$\Downarrow$$

Strategies should be compared based on the **expected** accumulated rewards they can produce – we follow **MEU principle**. Starting in s_0 and following π, the **expected discounted reward in step** i is:

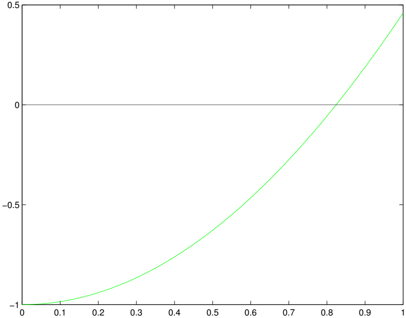
$$\gamma^i \cdot \mathbb{E}\left[R(S_i) \mid \pi, S_0 = s_0\right] = \gamma^i \sum_{s_i} R(s_i) P(S_i = s_i \mid \pi, S_0 = s_0)$$

The expected discounted reward of π is defined as:

$$U(s_0, \pi) = \sum_{i=0}^{\infty} \gamma^i \left(\sum_{s_i} R(s_i) \cdot P(S_i = s_i \mid \pi, S_0 = s_0) \right).$$

A side-step: Fix-point iterations

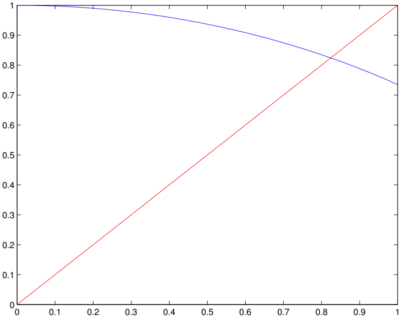
Solve the equation $x^2 - \cos(x) = 0$ on $x \in [0, 1]$.
Hint: We will use an iterative scheme to solve $x = \sqrt{\cos(x)}$.



How can we proceed to find an approximate solution?
Discuss with your neighbour for a couple of minutes...

A side-step: Fix-point iterations

We solve $x = \sqrt{\cos(x)}$ instead of $x^2 - \cos(x) = 0$.
The two have the same solution for $x \in [0, 1]$, so no worries.



Question: Why is that easier?

A side-step: Fix-point iterations (cont'd)

Solve iteratively: $x_{i+1} \leftarrow \sqrt{\cos(x_i)}$

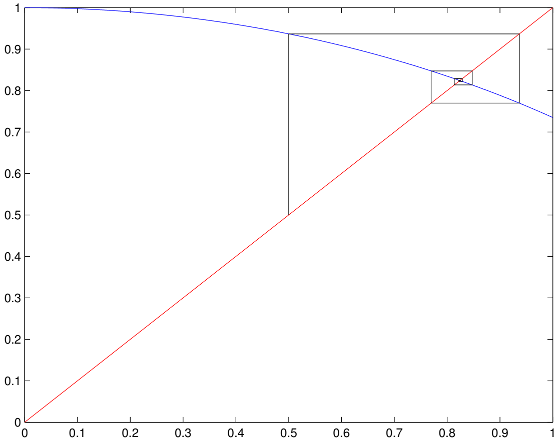
An equation $x = g(x)$ can be solved iteratively when $|g'(x)| < 1$.

Python code:

```
x = .5 # Initial value
for iter in range(no_iter):
   print(f"Iter {iter:2d}:  x = {x:.4f}")
   x = np.sqrt(np.cos(x)) # Do the update
```

A side-step: Fix-point iterations (cont'd)

Solve iteratively: $x_{i+1} \leftarrow \sqrt{\cos(x_i)}$
An equation $x = g(x)$ can be solved iteratively when $|g'(x)| < 1$.



A side-step: Fix-point iterations (cont'd)

Solve iteratively: $x_{i+1} \leftarrow \sqrt{\cos(x_i)}$
An equation $x = g(x)$ can be solved iteratively when $|g'(x)| < 1$.

Output:

Iter 0:	0.5000	Iter 8:	0.8237
Iter 1:	0.9368	Iter 9:	0.8243
Iter 2:	0.7697	Iter 10:	0.8241
Iter 3:	0.8474	Iter 11:	0.8242
Iter 4:	0.8136	Iter 12:	0.8241
Iter 5:	0.8288	Iter 13:	0.8241
Iter 6:	0.8220	Iter 14:	0.8241
Iter 7:	0.8251	Iter 15:	0.8241

A side-step: Fix-point iterations (higher dims)

It can work in higher dimensions, too!

We solve this set of equations

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} (2x^2 - 2y^3 + 1)/4 \\ (-x^4 - 4y^4 + 8y + 4)/12 \end{bmatrix}$$

simply by using

$$\begin{aligned} x_{i+1} &\leftarrow (2{x_i}^2 - 2{y_i}^3 + 1)/4 \\ y_{i+1} &\leftarrow (-{x_i}^4 - 4{y_i}^4 + 8y_i + 4)/12 \end{aligned}$$

to obtain

$$\begin{bmatrix} x \\ y \end{bmatrix} \approx \begin{bmatrix} 0.06177 \\ 0.72449 \end{bmatrix}.$$

Note indices! y_{i+1} calculated using x_i even if x_{i+1} is known.

Recap: What we are up to

To solve these decision-problems we need. . .

- A mapping from any state/action history σ_t to next action A_t.
  - We have simplified: $\pi_t(\sigma_t) \overset{\text{Markov}}{=} \pi_t(S_t) \overset{\text{Stationarity}}{=} \pi(S_t)$

- We proceed in two steps:
  1. Find a **utility function** $U^*(S_t)$ – how good it is to be in S_t
  2. Define $\pi(S_t)$ to **maximizes the expected utility**

	1	2	3
1	6.2	7.5	10
2	4.7	1.1	6.5
3	3.9	4.0	5.3

Possible repr. $U_t^*(S_t)$

	1	2	3
1	→	→	×
2	↑	↑	↑
3	↑	→	↑

Corresponding $\pi_t(S_t)$

- $U^*(S_t)$ is defined as the accumulated discounted reward following the optimal policy, but how to calculate it?
- **Fix-point iterations coming up next. . .**

Finding optimal strategies

The maximum expected utility of starting in state s_0 is:

$$U^*(s_0) = \max_\pi U(s_0, \pi) = \max_\pi \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i R(S_i) \,\middle|\, \pi, S_0 = s_0\right].$$

But how do we calculate this?

Finding optimal strategies

The maximum expected utility of starting in state s_0 is:

$$U^*(s_0) = \max_\pi U(s_0, \pi) = \max_\pi \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i R(S_i) \,\middle|\, \pi, S_0 = s_0\right].$$

But how do we calculate this?

In any state we choose the action maximizing the expected utility:

$$\pi(s) = \arg\max_a \sum_{s'} P(s' \mid s, a) \cdot U^*(s').$$

Thus, $U^*(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a) \cdot U^*(s')$.

Finding optimal strategies

The maximum expected utility of starting in state s_0 is:

$$U^*(s_0) = \max_\pi U(s_0, \pi) = \max_\pi \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i R(S_i) \,\middle|\, \pi, S_0 = s_0\right].$$

But how do we calculate this?

In any state we choose the action maximizing the expected utility:

$$\pi(s) = \arg\max_a \sum_{s'} P(s' \mid s, a) \cdot U^*(s').$$

Thus, $U^*(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a) \cdot U^*(s')$.

We now have:

- n **non-linear** equations; n unknowns ($n =$ no. states)
- A solution to these equations correspond to U^*.

The link back to fix-point iterations

We have found that for $U^*(s)$ it must hold that

$$U^*(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a) \cdot U^*(s').$$

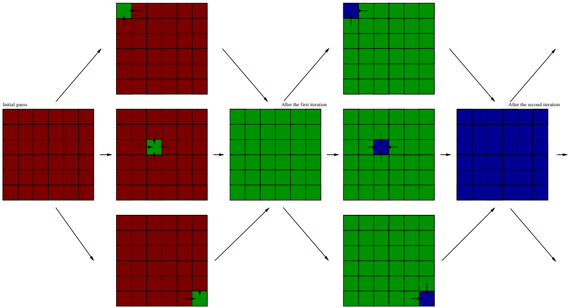
That is, we have a set of equations represented as "$U^* = g(U^*)$" (where $g(\cdot)$ follows from above), and it can be shown that fixed-point iterations will work for this setup.

We now know:

- Solving the n equations over n unknowns gives us U^*.
- **We can iteratively solve the equations!**
- The strategy is given by which action a that maximizes $\max_a \sum_{s'} P(s' \mid s, a) \cdot U^*(s')$.

Value iteration: Fix-point iterations in "value-space"

Start with an initial guess at the utility function $U^*(s)$ for each state s, and iteratively refine this using fix-point iterations:



The updating function:

$$\hat{U}_{j+1}(s) \leftarrow R(s) + \gamma \cdot \max_a \sum_{s'} P(s' \mid a, s) \cdot \hat{U}_j(s').$$

Value iteration: The algorithm

1. Choose an $\epsilon > 0$ to regulate the stopping criterion.
2. Let U_0 be an initial estimate of the utility function (for example, initialized to zero for all states).
3. Set $i := 0$.
4. **Repeat**
   1. Let $i := i + 1$.
   2. **For** each states s in S **do**

$$\hat{U}_i(s) := R(s) + \gamma \cdot \max_a \sum_{s'} P(s' \mid a, s) \hat{U}_{i-1}(s').$$

5. **Until** $|\hat{U}_i(s) - \hat{U}_{i-1}(s)| < \epsilon(1-\gamma)/\gamma$ for all s.
6. $\hat{\pi}(s) := \mathrm{argmax}_a \sum_{s'} P(s' \mid s, a) \cdot \hat{U}_i(s')$

Value iteration: Challenge

- A robot is in a 2-state world, states called left and right.
- The robot can choose between stay and move in both states.
  - The actions always do as would be expected, e.g., $P(s' = \text{right} | s = \text{right}, a = \text{stay}) = 1$.
- The robot gets a $+1$ reward if it is in right and -1 if in left.

Your task – together with your neighbour:

1. Formalize the domain as a Markov Decision Process. What assumptions are made? Do they make sense?
2. Solve the MDP using Value iteration. Remember that

$$\hat{U}_{j+1}(s) \leftarrow R(s) + \gamma \cdot \max_a \sum_{s'} P(s' \mid a, s) \cdot \hat{U}_j(s').$$

Initialize with $U_0(s) = 0.0$ and use $\gamma = .5$.

Value iteration: Solution

Model:

- Markov, stationarity, infinite time: All OK
- Define distributions by $a =$ stay means $s' = s$ with probability 1; $a =$ move means $s' \neq s$ with probability 1.

Value iteration: Solution

Model:

- Markov, stationarity, infinite time: All OK
- Define distributions by $a =$ stay means $s' = s$ with probability 1; $a =$ move means $s' \neq s$ with probability 1.

Formulas:

- In general $\hat{U}_{j+1}(s) \leftarrow R(s) + \gamma \cdot \max_a \sum_{s'} P(s' \mid a, s) \cdot \hat{U}_j(s')$.
- Here we can use that the model is deterministic to simplify:

$$\hat{U}_{j+1}(s) \leftarrow R(s) + \gamma \cdot \max\{\hat{U}_j(\text{left}), \hat{U}_j(\text{right})\}$$

Value iteration: Solution

Model:

- Markov, stationarity, infinite time: All OK
- Define distributions by $a = \text{stay}$ means $s' = s$ with probability 1; $a = \text{move}$ means $s' \neq s$ with probability 1.

Formulas:

- In general $\hat{U}_{j+1}(s) \leftarrow R(s) + \gamma \cdot \max_a \sum_{s'} P(s' \mid a, s) \cdot \hat{U}_j(s')$.
- Here we can use that the model is deterministic to simplify:

$$\hat{U}_{j+1}(s) \leftarrow R(s) + \gamma \cdot \max\{\hat{U}_j(\text{left}), \hat{U}_j(\text{right})\}$$

Results:

- $\hat{U}_0(\text{left}) = 0.0$, $\hat{U}_0(\text{right}) = 0.0$. Our starting-point.
- $\hat{U}_1(\text{left}) = -1 + .5 \cdot \max\{0, 0\} = -1$, $\hat{U}_1(\text{right}) = +1.0$.
- $\hat{U}_2(\text{left}) = -1 + .5 \cdot \max\{-1.0, 1.0\} = -.5$, $\hat{U}_2(\text{right}) = 1.5$.
- $\hat{U}_3(\text{left}) = -1 + .5 \cdot \max\{-.5, 1.5\} = -.25$, $\hat{U}_3(\text{right}) = 1.75$.
- $\hat{U}_\infty(\text{left}) = 0$, $\hat{U}_\infty(\text{right}) = 2$ w/ strategy “Get to right state”.

Value iteration: Extended example w/ robot navigation

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

Initial guess $\hat{U}_0$

The corresponding optimal strategy is uninformative:

	1	2	3
1	←	←	×
2	←	←	←
3	←	←	←

Value iteration: Extended example w/ robot navigation

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

Initial guess $\hat{U}_0$

	1	2	3
1			
2			
3			

First iteration $\hat{U}_1$

Value iteration: Extended example w/ robot navigation

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

Initial guess $\hat{U}_0$

	1	2	3
1		-0.1	
2			
3			

First iteration $\hat{U}_1$

$$\begin{aligned}
\hat{U}_1(\{1,2\}) &= R(\{1,2\}) + \gamma \cdot \max\Bigg\{ \sum_{s'} P(s' \mid \texttt{north}, \{1,2\})\hat{U}_0(s'), \ldots, \\
&\qquad \sum_{s'} P(s' \mid \texttt{west}, \{1,2\})\hat{U}_0(s') \Bigg\} \\
&= -0.1 + 0.9 \cdot \max\{0,0,0,0\} = -0.1.
\end{aligned}$$

Value iteration: Extended example w/ robot navigation

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

Initial guess $\hat{U}_0$

	1	2	3
1		-0.1	
2		-5	
3			

First iteration $\hat{U}_1$

$$\begin{aligned}\hat{U}_1(\{2,2\}) &= R(\{2,2\}) + \gamma \cdot \max\left\{\sum_{s'} P(s' \mid \texttt{north}, \{2,2\})\hat{U}_0(s'), \ldots, \right.\\ &\qquad \left.\sum_{s'} P(s' \mid \texttt{west}, \{2,2\})\hat{U}_0(s')\right\}\\ &= -5 + 0.9 \cdot \max\{0,0,0,0\} = -5.\end{aligned}$$

Value iteration: Extended example w/ robot navigation

Initial guess $\hat{U}_0$

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

First iteration $\hat{U}_1$

	1	2	3
1	-0.1	-0.1	10
2	-0.1	-5	-1
3	-0.1	-0.1	-0.1

The corresponding optimal strategy is still uninformed:

	1	2	3
1	←	←	×
2	←	←	←
3	←	←	←

Value iteration: Extended example w/ robot navigation

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

Initial guess $\hat{U}_0$

	1	2	3
1	−0.1	−0.1	10
2	−0.1	−5	−1
3	−0.1	−0.1	−0.1

First iteration $\hat{U}_1$

	1	2	3
1			
2			
3			

Second iteration $\hat{U}_2$

Value iteration: Extended example w/ robot navigation

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

Initial guess $\hat{U}_0$

	1	2	3
1	−0.1	−0.1	10
2	−0.1	−5	−1
3	−0.1	−0.1	−0.1

First iteration $\hat{U}_1$

	1	2	3
1		+5.7	
2			
3			

Second iteration $\hat{U}_2$

$$\begin{aligned}
\hat{U}_2(\{1,2\}) &= -0.1 + 0.9\cdot\max\{-0.7\cdot 0.1 + 0.1\cdot 10 - 0.1\cdot 5 - 0.1\cdot 0.1, \\
&\qquad 0.7\cdot 10 - 0.1\cdot 5 - 0.1\cdot 0.1 - 0.1\cdot 0.1, \\
&\qquad -0.7\cdot 5 - 0.1\cdot 0.1 - 0.1\cdot 0.1 + 0.1\cdot 10, \\
&\qquad -0.7\cdot 0.1 - 0.1\cdot 0.1 + 0.1\cdot 10 - 0.1\cdot 5\} \\
&= -0.1 + 0.9\cdot\max\{0.42, 6.48, -2.52, 0.42\} \\
&= 5.73,
\end{aligned}$$

Value iteration: Extended example w/ robot navigation

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

Initial guess $\hat{U}_0$

	1	2	3
1	−0.1	−0.1	10
2	−0.1	−5	−1
3	−0.1	−0.1	−0.1

First iteration $\hat{U}_1$

	1	2	3
1		+5.7	
2		−5.2	
3			

Second iteration $\hat{U}_2$

$$\begin{aligned}
\hat{U}_2(\{2,2\}) &= -5 + 0.9 \cdot \max\{-0.7 \cdot 0.1 - 0.1 \cdot 1 - 0.1 \cdot 0.1 - 0.1 \cdot 0.1, \\
&\qquad - 0.7 \cdot 1 - 0.1 \cdot 0.1 - 0.1 \cdot 0.1 - 0.1 \cdot 0.1, \\
&\qquad - 0.7 \cdot 0.1 - 0.1 \cdot 0.1 - 0.1 \cdot 0.1 - 0.1 \cdot 1, \\
&\qquad - 0.7 \cdot 0.1 - 0.1 \cdot 1 - 0.1 \cdot 0.1 - 0.1 \cdot 0.1\} \\
&= -5 + 0.9 \cdot \max\{-0.19, -0.73, -0.19, -0.19\} \\
&= -5.171,
\end{aligned}$$

Value iteration: Extended example w/ robot navigation

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

Initial guess $\hat{U}_0$

	1	2	3
1	−0.1	−0.1	10
2	−0.1	−5	−1
3	−0.1	−0.1	−0.1

First iteration $\hat{U}_1$

	1	2	3
1	−0.2	+5.7	+10
2	−0.6	−5.2	+4.8
3	−0.2	−0.6	−0.3

Second iteration $\hat{U}_2$

The optimal strategy corresponding to $\hat{U}_2$:

	1	2	3
1	←	→	×
2	←	←	↑
3	←	←	←

Value iteration: Extended example w/ robot navigation

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

Initial guess $\hat{U}_0$

	1	2	3
1	−0.1	−0.1	10
2	−0.1	−5	−1
3	−0.1	−0.1	−0.1

First iteration $\hat{U}_1$

	1	2	3
1	−0.2	+5.7	+10
2	−0.6	−5.2	+4.8
3	−0.2	−0.6	−0.3

Second iteration $\hat{U}_2$

. . .

The optimal strategy corresponding to $\hat{U}_\infty$:

	1	2	3
1	→	→	×
2	↑	↑	↑
3	↑	→	↑

Value iteration: The impact of the discounting factor

	1	2	3
1	6.2	7.5	10
2	4.7	1.1	6.5
3	3.9	4.0	5.3

$\hat{U}_\infty$ function ($\gamma = 0.9$)

	1	2	3
1	-0.1	0.6	10
2	-0.2	-5	-0.4
3	-0.1	-0.2	-0.1

$\hat{U}_\infty$ function ($\gamma = 0.1$)

	1	2	3
1	$\rightarrow$	$\rightarrow$	$\times$
2	$\uparrow$	$\uparrow$	$\uparrow$
3	$\uparrow$	$\rightarrow$	$\uparrow$

Optimal strategy ($\gamma = 0.9$)

Get to the goal – 'Money accumulates'

	1	2	3
1	$\rightarrow$	$\rightarrow$	$\times$
2	$\uparrow$	$\uparrow$	$\uparrow$
3	$\leftarrow$	$\leftarrow$	$\downarrow$

Optimal strategy ($\gamma = 0.1$)

'Avoid setbacks' – Future less relevant

Value iteration: Convergence

So the algorithm converges for this particular example, but does this hold in general?

Yes!

It can be proven that there is only one "true" utility function, and that value iteration is **guaranteed to converge** to this utility function.

Value iteration: Demo

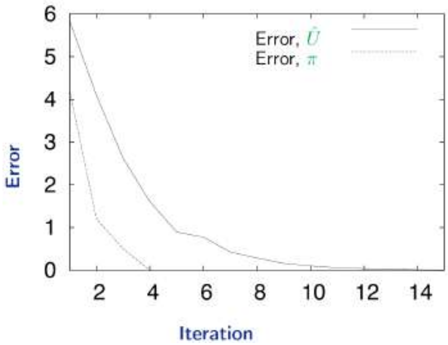
`rl_sim`-demo: `8_big.maze`

- Deterministic actions
- Stochastic actions

Value iteration: Efficiency

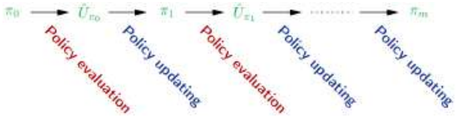
Value iteration converges, but is it efficient?

- Estimates utilities of **all** states with same requirements towards accuracy, also those states that are rarely visited.
- Convergence defined from accuracy of utility estimates, but the agent only cares about **making optimal decisions**.



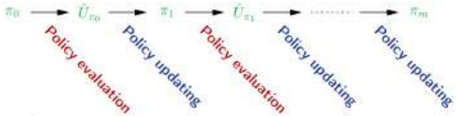
Policy iteration

Instead of updating the utility function, make an initial guess at the optimal **policy** and perform an iterative refinement of this guess:



Policy iteration

Instead of updating the utility function, make an initial guess at the optimal **policy** and perform an iterative refinement of this guess:



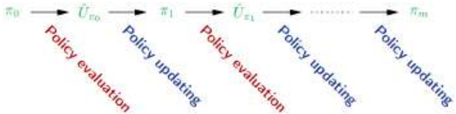
The **evaluation function**:

$$\hat{U}_{\pi_i}(s) = R(s) + \gamma \sum_{s'} P(s' \mid \pi_i(s), s) \cdot \hat{U}_{\pi_i}(s'),$$

which defines a system of **linear equalities**; the solution is $\hat{U}_{\pi_i}$.

Policy iteration

Instead of updating the utility function, make an initial guess at the optimal **policy** and perform an iterative refinement of this guess:



The **evaluation function**:

$$\hat{U}_{\pi_i}(s) = R(s) + \gamma \sum_{s'} P(s' \mid \pi_i(s), s) \cdot \hat{U}_{\pi_i}(s'),$$

which defines a system of **linear equalities**; the solution is $\hat{U}_{\pi_i}$.
The **updating function**:

$$\pi_{i+1}(s) := \arg\max_a \sum_{s'} P(s' \mid a, s) \hat{U}_{\pi_i}(s').$$

Policy iteration: the algorithm

1. Let π_0 be an initial randomly chosen policy.
2. Set $i := 0$.
3. **Repeat**
   1. Find the utility function $\hat{U}_{\pi_i}$ corresponding to the policy π_i [**Policy evaluation**].
   2. Let $i := i + 1$.
   3. **For** each s

$$\pi_i(s) := \arg\max_a \sum_{s'} P(s' \mid a, s)\hat{U}_{\pi_{i-1}}(s')[\textbf{Policy updating}].$$

4. **Until** $\pi_i = \pi_{i-1}$

Value iteration vs. Policy Iteration: Demo

`rl_sim-demo`

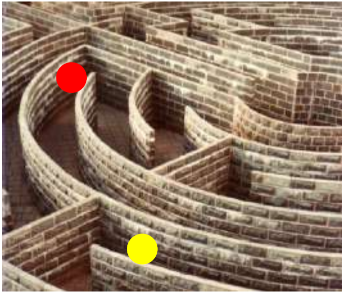
Methods:

- Value iteration
- Policy iteration

Approaches:

- **Step-by-step:** Notice number of iterations required for `1_smallest2.maze`
- **Execute:** Notice time spent per step for `8_big.maze`

Partial observability



- **Partially Observable Markov Decision Process (POMDP):** The agent does not observe the environment fully, so does not know the state it is in.
- A POMDP is like an MDP, but has an **observation model** $P(\mathbf{e}_t \mid s_t)$ defining the probability that the agent obtains evidence $\mathbf{e}_t$ when in state s_t
- Agent does not know which state it is in, so **it makes no sense to talk about policy $\pi(s)$!!**

Solving POMDPs

Theorem

The optimal policy in a POMDP is a function $\pi(b)$ *where* b *is the* **belief state** *(probability distribution over states) for the agent.*

Hence, we can convert a POMDP into an MDP in belief-state space, where $P(b_{t+1} \mid a_t, b_t)$ *is the probability that the new belief state* b_{t+1} *given that the current belief state is* b_t *and the agent does* a_t*.*

- If there are n states, b is an n-dimensional real-valued vector
  $\Rightarrow$ solving POMDPs is **very** hard! (PSPACE-hard)
- **The real world is a POMDP (with initially unknown transition and observation models)**

Summary

- **Sequential decision problems**
  - **Assumptions:** Stationarity, Markov assumption, Additive rewards, infinite horizon with discount
  - **Model class:** Markov decision problems
  - **Algorithm:** Value iteration / policy iteration
- Intuitively, MDPs combine **probabilistic models over time** (filtering, prediction) with the **maximum expected utiltiy principle**.