Dual–SPMA Framework for Convex MDPs

Speaker Script with Q&A Bank

Two-Column Format: Script + Anticipated Questions

Pegah Aryadoost   Danielle Nguyen   Shervin Khamooshian   Ahmed Magd

December 2025

Speaker Assignments

Speaker	Section	Slides
Pegah	Background & Motivation	1–9
Danielle	Problem Formulation & Fenchel Duality	10–17
Shervin	Related Work & Our Method	18–27
Ahmed	Experiments & Conclusion	28–36

Notation Pronunciation Guide

Pronunciation Guide

Greek Letters: π [pie] (policy) — γ [gamma] (discount) — ρ [rho] (initial dist.) — α [alpha] (step size) — β [beta] (dual step) — λ [lambda] (Lagrange mult.) — μ [mu] (penalty weight) — τ [tau] (threshold) — η [eta] (learning rate) — ω [omega] (dual params) — ϕ [phi] (features)
Calligraphic: $\mathcal{S}$ [script-S] (states) — $\mathcal{A}$ [script-A] (actions) — $\mathcal{D}$ [script-D] (occupancy set)
Functions: f^* [f-star] (conjugate) — ∇ [nabla/gradient] — sup [soup/supremum] — $\langle\cdot,\cdot\rangle$ [inner product] — $[\cdot]_+$ [positive part]
Key Terms: d_π [d-pi] (occupancy) — $J(\pi)$ [J of pi] (return) — $L(\pi, y)$ [Lagrangian] — r_y [r-sub-y] (shaped reward) — $\hat{d}_\pi$ [d-hat-pi] (estimated occupancy)

Section 1: Background & Motivation (Pegah, Slides 1–9)

Slide 1 — *A Dual–SPMA Framework for Convex MDPs*

Speaker Script (Pegah)

- "Hi everyone, thanks for being here. I'm Pegah, and this project is joint work with Ahmed, Shervin, and Danielle from SFU's School of Computing Science."
- "Our title is 'A Dual–SPMA Framework for Convex MDPs'."
- "The one-sentence claim: **combining Fenchel duality with a fast policy optimizer, SPMA, gives a simple and fairly competitive way to solve convex MDPs**."
- "We'll compare this 'Dual–SPMA' recipe against a strong baseline called NPG–PD."

Related Q&A

Q: What's the main takeaway?
A: If you take the convex-MDP framework of Zahavy et al. and plug in SPMA as the policy optimizer, you get a simple "Dual–SPMA" recipe that (i) is easy to implement as shaped-reward RL, and (ii) performs competitively with NPG–PD on constrained safety tasks.

Q: Is Dual–SPMA supposed to *beat* NPG–PD?
A: No, we show SPMA is a **viable policy player** inside the convex-MDP dual framework: it's stable, respects constraints, and gives comparable reward.

Slide 2 — *Outline*

Speaker Script (Pegah)

- "Here's the plan for the next 20–25 minutes."
- "First, I'll quickly review basic RL background."
- "Then I'll motivate **convex MDPs**—why we care about them beyond standard RL."
- "Danielle: core math (Fenchel duality → saddle-point)."
- "Shervin: related work and Dual–SPMA method."
- "Ahmed: experiments and results."
- "Finally: takeaways and future work."

Related Q&A

Q: Why compare against NPG–PD specifically? Why not PPO or TRPO?
A: Our work is about **convex objectives & dual variables**, not generic performance on Atari. NPG–PD is a principled *primal–dual* algorithm specifically designed for constrained MDPs, with guarantees on both optimality and constraint violation. It also uses a geometry-aware policy update (natural gradient), which makes the comparison with SPMA conceptually clean. PPO/TRPO are great empirically, but they're not formulated directly as primal–dual CMDP algorithms.

Slide 3 — *What is Reinforcement Learning?*

Speaker Script (Pegah)

- "Let's start with a quick refresh on RL."
- "In RL, an **agent** interacts with an **environment** repeatedly over time."
- "At each time step t [t]: (1) observe state s_t [s-sub-t], (2) choose action a_t [a-sub-t], (3) receive reward r_t [r-sub-t], (4) transition to s_{t+1} [s-sub-t-plus-one]."
- "Goal: learn a policy that maximizes long-term reward."

Related Q&A

(Basic RL background—unlikely to get technical questions here.)

Q: What's the difference between y and λ?
A: y [y] is the general Fenchel dual variable over state–action pairs. λ [lambda] is the scalar Lagrange multiplier for CMDP safety constraints. In the constrained case: $y_\lambda(s,a) = \lambda c(s,a) - r(s,a)$.

Slide 4 — *Markov Decision Process (MDP): Formal Definition*

Speaker Script (Pegah)

- "Formally, the environment is a **Markov Decision Process**."
- "MDP: $(\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho)$ [script-S, script-A, P, r, gamma, rho]"
- "$\mathcal{S}$: state space; $\mathcal{A}$: action space"
- "$P(s' \mid s, a)$ [P of s-prime given s, a]: transition probability"
- "$r(s,a)$: reward; γ [gamma]: discount factor (0 to 1)"
- "$\rho(s)$: initial state distribution"
- "Policy $\pi(a \mid s)$ [pi of a given s]: probability distribution over actions"

Related Q&A

Q: What assumptions do you need for your convex MDP formulation?
A: We assume a discounted MDP with $\gamma < 1$, bounded reward and cost, and that we restrict to stationary policies. For convexity, we require f to be proper, closed, and convex on the occupancy polytope $\mathcal{D}$. For the Fenchel reduction, we also need standard conditions that guarantee the biconjugate equality and existence of a saddle point—essentially the Fenchel–Moreau assumptions.

Slide 5 — *Trajectory and Return*

Speaker Script (Pegah)

- "Running a policy gives a **trajectory**: sequence of states, actions, rewards."
- "$s_0 \to a_0 \to r_0 \to s_1 \to \dots$"
- "The **discounted return**: $J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$ [J of pi equals E-sub-pi of the sum...]"
- "Classical RL objective: find π^* [pi-star] that maximizes this."

Related Q&A

Q: Can you restate the key identity relating $J(\pi)$ and d_π?

A: Yes: we define

$$d_\pi(s,a) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \Pr_\pi(s_t = s, a_t = a).$$

Then

$$\langle r, d_\pi \rangle = \sum_{s,a} r(s,a)\, d_\pi(s,a) = (1-\gamma)J(\pi).$$

So up to the constant factor $1-\gamma$, the RL objective is just a linear function of the occupancy measure.

Slide 6 — *Occupancy Measure: Where the Policy Spends Time*

Speaker Script (Pegah)

- "Now we introduce the **discounted occupancy measure**."
- "$d_\pi(s,a) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \Pr_\pi(s_t = s, a_t = a)$ [d-sub-pi of s, a...]"
- "Intuitively: **normalized discounted frequency** of visiting (s,a) under π."
- "$(1-\gamma)$ normalizes so $\sum_{s,a} d_\pi(s,a) = 1$."
- "Think of d_π as a heatmap over states and actions."

Related Q&A

Q: Why introduce occupancy measures at all? Why not just work with $J(\pi)$?

A: Because in terms of policy parameters, $J(\pi)$ is generally non-convex. In terms of occupancy, return is **linear**: $\langle r, d_\pi \rangle = (1-\gamma)J(\pi)$. Many interesting extras (entropy, penalties, constraints) are convex in d. The set of valid occupancies is a convex polytope defined by linear flow constraints. So in d-space we get a clean convex program: minimize a convex $f(d)$ over a convex set $\mathcal{D}$.

Q: What exactly is $\mathcal{D}$, the set of feasible occupancies?

A: $\mathcal{D}$ is the set of $d \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ that satisfy the Bellman flow constraints plus non-negativity: for every state s,

$$\sum_a d(s,a) = (1-\gamma)\rho(s) + \gamma \sum_{s',a'} P(s|s',a')\, d(s',a'), \quad d(s,a) \geq 0.$$

This says "discounted occupancy flowing *into* a state equals initial occupancy plus discounted flow from other states." It's a convex polytope.

Slide 7 — *Key Identity: RL is Linear in Occupancy*

Speaker Script (Pegah)

- "The **fundamental identity**:"
- "$\langle r, d_\pi \rangle = \sum_{s,a} r(s,a)\, d_\pi(s,a) = (1-\gamma)J(\pi)$"
- "So the RL objective is simply a linear function of d_π."
- "However, many interesting objectives are **not** linear in d_π:"
- "Safety constraints: $\langle c, d_\pi \rangle \leq \tau$"
- "Imitation learning (match expert occupancy)"
- "Exploration bonuses (entropy of d_π)"
- "That's where **convex MDPs** come in."

Related Q&A

Q: Is there a 1–1 correspondence between policies and occupancy measures?

A: Under mild regularity conditions (e.g., communicating MDP, $\gamma < 1$), each stationary policy has a unique discounted occupancy measure, and each valid occupancy in $\mathcal{D}$ corresponds to at least one stationary policy (you can recover $\pi(a|s) = d(s,a)/\sum_{a'} d(s,a')$ where the denominator is nonzero). So working in d-space is equivalent, up to degeneracies.

Q: Why do you call $f(d)$ "convex" when there's a minus reward term $-\langle r, d \rangle$?

A: Because $-\langle r, d \rangle$ is linear in d, and linear functions are both convex and concave. When we add convex terms like entropy or penalties, the sum stays convex. So all the examples on the "Convex MDP: Examples" slide fit into "convex f" without breaking anything.

Slide 8 — *Why Convex MDPs?*

Speaker Script (Pegah)

- "Linear RL is great if all we care about is maximizing reward."
- "But in practice we often care about more:"
- "**Enforce safety** constraints"
- "**Match expert behaviour** in imitation learning"
- "**Encourage exploration** or control risk"
- "Convex MDPs: instead of maximizing linear $\langle r, d_\pi \rangle$, we **minimize convex** $f(d_\pi)$."
- "Challenge: optimizing over d is hard—dimension $|\mathcal{S}||\mathcal{A}|$, complicated polytope."
- "Our approach: **Fenchel duality** → **min–max game** → shaped-reward RL."

Related Q&A

Q: Why do we even care about *convex* MDPs instead of just standard RL?
A: Standard RL maximizes a linear function of the occupancy measure—expected return—so it's great when you only care about reward. Many real problems add structure: safety constraints, exploration terms, risk penalties, imitation losses, etc. Many of those can be written as convex functionals of the occupancy d_π. Convex MDPs give a unified way to handle these goals while preserving nice optimization properties like existence of a saddle point.

Q: How is this different from just "constrained RL with a Lagrange multiplier"?
A: Classic constrained RL focuses on a particular structure: reward plus linear constraints. Convex MDPs are more general: you minimize an arbitrary convex function $f(d_\pi)$ over the convex occupancy polytope. Constrained RL is one example where f adds a penalty or indicator for constraint violations; entropy-regularized RL and some imitation objectives are *other* examples. Our framework—and the Fenchel duality step—work for all of them, not just linear constraints.

Slide 9 — *Convex MDP: Examples*

Speaker Script (Pegah)

- "Examples of convex MDP objectives:"
- "Standard RL: $f(d) = -\langle r, d \rangle$ (linear, hence convex)"
- "**Entropy-regularized RL**: $f(d) = -\langle r, d \rangle + \alpha \sum_{s,a} d(s,a) \log d(s,a)$"
- "**Constrained safety**: $f(d) = -\langle r, d \rangle + \mu \max\{0, \langle c, d \rangle - \tau\}$"
- "Here c: cost, τ [tau]: threshold, μ [mu]: penalty weight"
- "All three are **convex functions of occupancy**."
- "Next: Danielle shows **Fenchel duality**."

Related Q&A

Q: Why use μ for penalty weight instead of λ?
A: We intentionally use μ there to distinguish a **fixed penalty weight** in a penalized objective from the **Lagrange multiplier** λ that evolves as a dual variable. This avoids confusion between "tunable hyperparameter" and "optimization variable."

Q: What's the difference between y and λ?
A: y: general Fenchel dual variable over state–action pairs. λ: scalar Lagrange multiplier for CMDP constraints. In constrained case: $y_\lambda(s,a) = \lambda c(s,a) - r(s,a)$.

Section 2: Problem Formulation & Fenchel Duality (Danielle, Slides 10–17)

Slide 10 — *Roadmap (checkpoint)*

Speaker Script (Danielle)

- "Thanks, Pegah. So far: RL basics and convex MDPs motivation."
- "Next: I'll show how **Fenchel duality** turns this into a saddle-point formulation."

Related Q&A

(Transition slide—take a breath, check audience engagement.)

Slide 11 — *Fenchel Conjugate: Definition*

Speaker Script (Danielle)

- "The **Fenchel conjugate** of convex f:"
- "$f^*(y) = \sup_x\{\langle y, x\rangle - f(x)\}$ [f-star of y equals supremum...]"
- "y: dual variable; $\langle y, x\rangle$: dot product; sup [soup]: supremum"
- "Intuitively: how large can $\langle y, x\rangle$ be relative to $f(x)$?"
- "We'll use this to expose a min-max structure."

Related Q&A

Q: Why bring in Fenchel conjugates at all?
A: Fenchel conjugates give us an identity:

$$f(d) = \sup_y\{\langle y, d\rangle - f^*(y)\}.$$

Plugging this into $\min_{d\in\mathcal{D}} f(d)$ lets us rewrite the convex MDP as

$$\min_{d\in\mathcal{D}} \max_y\{\langle y, d\rangle - f^*(y)\}.$$

So we get a saddle-point game with a natural dual variable y. This structure is what allows us to run two-player algorithms: one player updates y, the other updates the policy.

Q: How do you get the conjugate $f^*(y)$ in the entropy-regularized case?
A: For

$$f(d) = -\langle r, d\rangle + \alpha \sum d \log d,$$

you can complete the square in log-space / use standard convex analysis to show

$$f^*(y) = \alpha \log \sum_{s,a} \exp((y(s,a) + r(s,a))/\alpha).$$

Its gradient is a softmax: $\nabla f^*(y) = \mathrm{softmax}((y + r)/\alpha)$. That's the expression we use in the dual update.

Slide 12 — *Fenchel–Moreau Theorem*

Speaker Script (Danielle)

- "Key result: **Fenchel–Moreau theorem**."
- "For proper, closed, convex f:"
- "$f(d) = \sup_y\{\langle y, d\rangle - f^*(y)\}$"
- "So $f = f^{**}$ [f equals f-double-star]."
- "This allows us to turn **minimizing** $f(d)$ into a **min–max optimization** over d and y."

Related Q&A

Q: What assumptions for convex MDP formulation?
A: Discounted MDP ($\gamma < 1$), bounded reward/cost, stationary policies. f must be proper, closed, convex on $\mathcal{D}$. Standard Fenchel–Moreau conditions for biconjugate equality and saddle point existence.

Slide 13 — *Applying Fenchel Duality to Convex MDPs*

Speaker Script (Danielle)

- "Start with: $\min_{d\in\mathcal{D}} f(d)$"
- "Apply Fenchel–Moreau: $f(d) = \sup_y\{\langle y, d\rangle - f^*(y)\}$"
- "Plug in: $\min_{d\in\mathcal{D}} \sup_y\{\langle y, d\rangle - f^*(y)\}$"
- "This is a **convex-concave saddle-point problem**."
- "Under standard assumptions, solving this equals solving original convex MDP."
- "Replace d by d_π: $\min_\pi \max_y L(\pi, y)$ where $L(\pi, y) = \langle y, d_\pi\rangle - f^*(y)$"

Related Q&A

Q: Why do you write $\min_d \max_y$ instead of $\max_y \min_d$? Are you allowed to swap min and max?

A: We derive the saddle-point form by directly applying Fenchel–Moreau:

$$f(d) = \sup_y\{\langle y, d\rangle - f^*(y)\},$$

then taking $\min_d$ outside. Under standard convexity/closedness assumptions, there is no duality gap and the saddle-point value equals both the minimax and maximin. So while we don't literally swap orders in the algebra, we rely on these conditions to interpret the solution as an equilibrium of the game.

Slide 14 — *Two-Player Game Interpretation*

Speaker Script (Danielle)

- "Saddle-point as a **two-player game**:"
- "$\min_\pi \max_y L(\pi, y) = \langle y, d_\pi\rangle - f^*(y)$"
- "**Policy player**: chooses π, wants to **minimize** L (RL agent)"
- "**Dual player**: chooses y, wants to **maximize** L (shapes reward)"
- "At equilibrium (saddle point): neither can unilaterally improve"
- "Policy player has π^*, dual player has y^*"

Related Q&A

Q: Why is the solution of the min–max at a saddle point?

A: Because in convex-concave problems, under suitable conditions, the minimax and maximin values coincide and are achieved at a pair (π^*, y^*) satisfying

$$L(\pi^*, y) \le L(\pi^*, y^*) \le L(\pi, y^*)$$

for all π, y. That point is simultaneously optimal for the min player (policy) and the max player (dual), and is called a saddle point. Solving the min–max is thus equivalent to finding such a saddle point.

Slide 15 — *Visualizing the Saddle Point*

Speaker Script (Danielle)

- "3D visualization: $L(d, y)$ as a surface"
- "d (horizontal): policy-side variable"
- "y (other axis): dual variable"
- "**Red point**: saddle point"
- "Along d-direction (fix y): red point is a **minimum**"
- "Along y-direction (fix d): red point is a **maximum**"
- "It's 'best response' for both players → solution of min–max game"

Related Q&A

Q: Does your algorithm *guarantee* convergence to the saddle point?

A: In the idealized tabular, exact-oracle setting, existing theory (Zahavy et al. + SPMA paper) shows that if the dual player and the policy player both use low-regret algorithms, the average iterates converge to a saddle point. In our implementation we have function approximation, noisy occupancy estimates, and finite inner loops, so we don't claim a formal convergence proof. Instead we check empirically that the saddle objective and constraints behave sensibly.

Slide 16 — *From Saddle Point to Shaped Reward*

Speaker Script (Danielle)

- "How to minimize over policies in saddle-point?"
- "For fixed y, policy subproblem: $\min_\pi \langle y, d_\pi\rangle$"
- "Using occupancy definition: $\langle y, d_\pi\rangle = (1-\gamma)\mathbb{E}_\pi[\sum_t \gamma^t y(s_t, a_t)]$"
- "So minimizing $\langle y, d_\pi\rangle$ = **maximizing** return with **shaped reward**: $r_y(s, a) = -y(s, a)$"
- "**Key insight**: policy player can use **standard RL** on shaped reward $r_y = -y$"

Related Q&A

Q: What does the dual variable y represent intuitively?

A: It's a vector of the same dimension as d—one coordinate per state–action pair. In the Fenchel dual, it's the variable that "linearizes" the convex objective. Operationally, for the policy player, y is just a shaped reward field: we treat $r_y(s, a) = -y(s, a)$ as the reward. Different choices of y encourage the policy to put occupancy where f wants it.

Q: Does the shaped reward $r_y = -y$ break the Markov property or anything?

A: No, because y is a function of the current state–action pair (s, a) only; it doesn't introduce dependence on the entire trajectory. The environment dynamics $P(s'|s, a)$ are unchanged, and the reward at each step is just a different function of (s, a). So the process is still a valid MDP, just with a different reward.

Slide 17 — *Summary: The Fenchel Dual Reduction*

Speaker Script (Danielle)

- “Summary of reduction:”
- “Started with convex MDP: $\min_\pi f(d_\pi)$”
- “Fenchel duality $\to$ saddle-point: $\min_\pi \max_y L(\pi, y)$”
- “Fixing y: minimizing over π = RL with shaped reward $r_y = -y$”
- “Simple algorithmic structure:”
- “1. **Dual step**: $y_{k+1} = y_k + \alpha(\hat{d}_{\pi_k} - \nabla f^*(y_k))$”
- “2. **Policy step**: run RL on $r_{y_k} = -y_k$ to get π_{k+1}”
- “Reduced to: **alternating convex optimization (dual) and standard RL (policy)**”

Related Q&A

Q: Why not just solve the convex program over d with CVX or some LP solver?

A: You *could* if the MDP is small and fully known. In our setting we assume a **model-free** RL scenario: we don’t know the transition probabilities, we only interact with the environment by sampling trajectories. That means we can’t write down $\mathcal{D}$ explicitly or compute exact expectations. RL gives us a way to approximate $\min_\pi \langle y, d_\pi \rangle$ from samples by treating $-y$ as a reward and running a policy optimizer.

Q: Why is alternating updates (policy step then dual step) reasonable, given that vanilla Gradient Descent–Ascent can diverge?

A: It’s true that naive GDA can oscillate. We’re not using plain GDA: we use specific online-learning style updates. The dual player uses a simple FTL/gradient step in a convex Euclidean space; the policy player uses SPMA, which is a mirror-descent method in logit space with convergence guarantees. Online saddle-point theory says that when both players have low regret, the average of their strategies converges to a saddle point, so alternating these particular updates is justified.

Section 3: Related Work & Our Method (Shervin, Slides 18–27)

Slide 18 — *Roadmap (checkpoint)*

Speaker Script (Shervin)

- "We've completed the core math using Fenchel duality."
- "Next: related work that inspired this framework and the specific policy optimizer, SPMA."

Related Q&A

(Transition slide—Shervin takes over.)

Slide 19 — *Related Work: "Reward Is Enough"*

Speaker Script (Shervin)

- "First key paper: '*Reward Is Enough for Convex MDPs*' by Zahavy et al."
- "Foundational work for convex-MDP perspective."
- "Three main contributions:"
- "1. **Fenchel dual reduction**: convex MDP $\rightarrow$ saddle-point"
- "2. **Meta-algorithm**: alternate policy & dual updates"
- "3. Many RL paradigms (imitation, constrained, entropy-reg.) are special cases"
- "Our contribution: **implement this with SPMA** and compare to NPG–PD."

Related Q&A

Q: If you can plug "any RL algorithm" into the policy player, why specifically SPMA?
A: The convex-MDP theory says any RL algorithm could in principle be used as the policy player. We choose SPMA because (i) it has **strong convergence results** in tabular MDPs, (ii) it's geometry-aware, operating in logit space with a softmax mirror map, (iii) its loss looks similar to PPO/MDPO, so it's practical to implement, and (iv) the SPMA paper already shows it's competitive with PPO/TRPO in continuous control.

Slide 20 — *Related Work: Softmax Policy Mirror Ascent (SPMA)*

Speaker Script (Shervin)

- "Second key ingredient: **SPMA** from Asad et al."
- "**Mirror ascent in logit space** using log-sum-exp mirror map."
- "In tabular MDPs: achieves **linear convergence**."
- "Tabular update: $\pi_{t+1}(a|s) = \pi_t(a|s)[1 + \eta A_{\pi_t}(s,a)]$"
- "**Geometry-aware**: respects probability simplex"
- "No explicit per-state normalization"
- "Clean extension to function approximation as convex classification"

Related Q&A

Q: How is SPMA different from vanilla policy gradient?
A: Vanilla PG minimizes $L_{\mathrm{PG}} = -\mathbb{E}[\log \pi(a|s) A(s,a)]$ and takes gradient steps in parameter space. SPMA instead uses an **exponential-gradient / mirror-descent update in logit space**. The actor loss includes a regularizer

$$\frac{1}{\eta}(\exp(\Delta \log \pi) - 1 - \Delta \log \pi)$$

which penalizes large changes in log-probabilities. This gives better control over how far the policy moves per step and leads to faster convergence in the tabular theory.

Q: Why is SPMA "geometry-aware"?
A: Because it does mirror ascent with the log-sum-exp mirror map, which is tailored to the simplex geometry. Instead of taking Euclidean steps in parameter space and then projecting back onto the simplex, SPMA treats the logits as dual variables and the softmax as the mirror map. This respects the probability constraints and tends to produce well-behaved updates when probabilities are near 0 or 1.

Slide 21 — *Related Work: NPG–PD (Our Baseline)*

Speaker Script (Shervin)

- “Main baseline: **NPG–PD** (Ding et al. 2020)”
- “Designed for constrained MDPs with Lagrangian:”
- “$L(\pi, \lambda) = J_r(\pi) + \lambda(J_c(\pi) - \tau)$, $\lambda \geq 0$”
- “**Primal**: natural policy gradient ascent (Fisher geometry)”
- “**Dual**: $\lambda_{k+1} = [\lambda_k + \beta(J_c(\pi_k) - \tau)]_+$”
- “Has $\mathcal{O}(1/\sqrt{T})$ bounds on optimality gap & constraint violation”
- “We use NPG–PD as our **comparison point**.”

Related Q&A

Q: How does SPMA compare to natural policy gradient conceptually?

A: Both are trying to account for the geometry of the policy space. Natural PG uses the Fisher information matrix to modify the gradient direction and step size; SPMA uses a Bregman divergence in logit space. In small tabular problems you can show they are closely related; in our setting we treat them as two reasonable but different choices of geometry-aware update and compare them empirically.

Slide 22–23 — *Roadmap & Our Contributions*

Speaker Script (Shervin)

Slide 22 (Roadmap):

- “So far: convex-MDP theory, Fenchel dual framework, SPMA optimizer.”
- “Next: what we actually built.”

Slide 23 (Our Contributions):

- “1. **Complete Dual–SPMA framework**: outer dual loop + SPMA policy oracle”
- “Supports entropy-regularized RL and constrained safety CMDPs”
- “2. **Three occupancy estimators**: tabular MC, feature-based MC, MLE-style”
- “3. **Faithful NPG–PD baseline**”
- “4. **Empirical comparison** of SPMA vs NPG–PD on constrained safety”

Related Q&A

Q: Could you use other policy optimizers in place of SPMA?

A: Yes. The meta-algorithm from Zahavy et al. lets you plug in any RL method as the policy player. We chose SPMA because of its theory and similarity to modern methods, but in principle you could try PPO, TRPO, or MDPO. An interesting piece of future work would be a systematic empirical comparison of different policy players inside the same convex-MDP dual framework.

Slide 24 — *Dual–SPMA Loop: High-Level View*

Speaker Script (Shervin)

- “High-level Dual–SPMA structure:”
- “Saddle-point objective: $L(\pi, y) = \langle y, d_\pi \rangle - f^*(y)$”
- “**Outer loop** (dual): $y_{k+1} = y_k + \alpha(\hat{d}_{\pi_k} - \nabla f^*(y_k))$”
- “Use estimated occupancy $\hat{d}_{\pi_k}$ from rollouts”
- “**Inner loop** (policy): run K_{inner} SPMA steps on shaped reward $r_y = -y$”
- “After fixed outer iterations, return final policy π_K”

Related Q&A

Q: Why do you use a simple gradient/FTL step for the dual variable y?

A: The dual variable lives in a straightforward Euclidean space and the dual objective is convex with a simple gradient $\hat{d}_\pi - \nabla f^*(y_k)$. For such problems, basic online convex optimization algorithms like gradient ascent or FTL are known to have sublinear regret. They’re easy to implement and come with clear theoretical guarantees, so there’s no need for more complex methods.

Q: How do you choose the SPMA step size η?

A: We use a simple Armijo line search over a small discrete set of candidate η values. For each candidate we compute the SPMA loss on a minibatch and pick the largest η that produces a sufficient decrease. This is inspired by the SPMA paper and by MDPO/TRPO line-search practices.

Slide 25 — *Policy Player: SPMA Actor Loss*

Speaker Script (Shervin)

- "SPMA policy update via custom **actor loss**."
- "Standard PG: $L_{\text{PG}} = -\mathbb{E}[\log \pi(a|s) A(s,a)]$"
- "SPMA adds 'stay close to old policy' regularizer in logit space:"
- "$L_{\text{SPMA}} = \mathbb{E}[-\Delta \log \pi \cdot A + \frac{1}{\eta}(\exp(\Delta \log \pi) - 1 - \Delta \log \pi)]$"
- "$\Delta \log \pi = \log \pi_{\text{new}} - \log \pi_{\text{old}}$"
- "Exponential term: KL-like regularizer penalizing large log-prob changes"

Related Q&A

Q: Where does your SPMA loss come from?
A: It's the convex surrogate corresponding to the tabular SPMA update. If you linearize the value function and consider an exponential-gradient step in logit space, the associated Bregman divergence gives rise to exactly that $\exp -1 - \Delta \log \pi$ term. The SPMA paper derives it formally; we implement it as a differentiable loss so we can use standard automatic differentiation.

Q: Do you enforce the $[1 + \eta A]_+$ and normalization from the tabular update?
A: In the tabular theory, the closed-form update is

$$\pi_{t+1}(a|s) \propto \pi_t(a|s)[1 + \eta A(s,a)]_+.$$

In our neural approximation we don't directly enforce this per-state formula. Instead we optimize the SPMA loss over minibatches of trajectories; the softmax layer plus the KL-like regularizer implicitly enforce a similar behaviour. So we're using the **function-approximation version** of SPMA instead of the exact tabular update.

Slide 26 — *Occupancy Estimation: MC vs MLE*

Speaker Script (Shervin)

- "Critical ingredient: occupancy estimate $\hat{d}_\pi$"
- "**Default**: Monte Carlo estimator:"
- "$\hat{d}_\pi(s,a) = \frac{1-\gamma}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \gamma^t \mathbf{1}\{s_t^{(i)} = s, a_t^{(i)} = a\}$"
- "Simple: count discounted visits. Variance grows with $|\mathcal{S}||\mathcal{A}|$."
- "Also explored **MLE-based estimator** (Barakat et al.): log-linear $\lambda_\omega(s,a) \propto \exp(\omega^\top \phi(s,a))$"
- "Error depends on feature dimension, not $|\mathcal{S}||\mathcal{A}|$"
- "Sanity check: verified $\sum_{s,a} \hat{d}_\pi(s,a) \approx 1$"

Related Q&A

Q: Why not use exact occupancies in FrozenLake instead of Monte Carlo?
A: We intentionally use Monte Carlo to simulate the **model-free** setting most RL algorithms operate in—where you don't have access to $P(s'|s,a)$. It also keeps the pipeline more consistent with what you'd do in larger environments. Using the exact occupancy is possible in FrozenLake, but we'd then be solving a much easier problem that doesn't stress the estimator.

Q: How do you ensure your Monte Carlo occupancy estimate is valid?
A: We compute

$$\hat{d}_\pi(s,a) = \frac{1-\gamma}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \gamma^t \mathbf{1}\{s_t^{(i)} = s, a_t^{(i)} = a\}.$$

By construction this is non-negative. As a sanity check, we track $\sum_{s,a} \hat{d}_\pi(s,a)$ over training and verify it stays very close to 1; that's the plot on our entropy-regularized results slide. Small deviations come from truncating trajectories at finite T.

Q: What's the point of the MLE-based occupancy estimator?
A: The tabular MC estimator's variance scales with $|\mathcal{S}||\mathcal{A}|$; in large MDPs that's bad. The MLE estimator instead fits a log-linear model $\lambda_\omega(s,a) \propto \exp(\omega^\top \phi(s,a))$ by maximum likelihood on samples. Its error depends mostly on the feature dimension, not the raw state–action count. So it's a more scalable idea we wanted to prototype, although we only briefly tested it here.

Slide 27 — *Example: Constrained Safety CMDP*

Speaker Script (Shervin)

- "How constrained MDP fits our framework:"
- "Problem: maximize $J_r(\pi)$ s.t. $J_c(\pi) \le \tau$"
- "In Dual–SPMA:"
- "1. Build dual: $y_\lambda(s,a) = \lambda c(s,a) - r(s,a)$"
- "2. Policy sees: $r_y = r - \lambda c$"
- "3. Run SPMA inner loop on r_y"
- "4. Update dual: $\lambda_{k+1} = [\lambda_k + \beta(J_c(\pi_k) - \tau)]_+$"
- "SPMA & NPG–PD have **same dual update**—difference is policy optimizer."
- "Now Ahmed presents experimental results."

Related Q&A

Q: Are there any constraints on y? Do you project it?

A: In the entropy-regularized case, y is unconstrained—we don't project. For CMDPs we sometimes parameterize the dual as $y_\lambda = \lambda c - r$ with $\lambda \ge 0$, so the only constraint is non-negativity on λ, which we enforce with a $[\cdot]_+$ projection (same as NPG–PD). More sophisticated bounds or regularizers on y are possible but we didn't explore them in this project.

Q: Is the dual step size α sensitive?

A: Yes, like most gradient methods it needs to be in a reasonable range: too small and progress is slow; too large and you can oscillate. In practice we tuned α on a log scale and picked values where the dual objective and the constraint violation decreased smoothly. Because our setting is tabular and relatively small, this wasn't too painful.

Section 4: Experiments & Conclusion (Ahmed, Slides 28–36)

Slide 28–30 — *CMDP Example, Roadmap, Experimental Setup*

Speaker Script (Ahmed)

Slide 28: CMDP example (same as Slide 27, concrete walkthrough).

Slide 29: "We've described Dual–SPMA, occupancy estimators, NPG–PD baseline. Next: experimental setup and results."

Slide 30 (Experimental Setup):

- "Tabular setting for visualizing occupancies."
- "Environment: **FrozenLake 4×4** gridworld, deterministic transitions."
- "Unsafe states (holes): cost $c = 1$."
- "Compare: **Dual–SPMA** vs **NPG–PD**"
- "Track: $J_r(\pi)$, $J_c(\pi)$, $J_c - \tau$ (violation), occupancy sum ≈ 1"
- "Settings: $\gamma = 0.99$, $\tau = 0.1$, 30 outer iterations, 2048 steps/rollout"

Related Q&A

Q: Why FrozenLake 4×4?
A: Wanted **tabular** CMDP to (i) compute/visualize occupancies, (ii) understand safety constraint intuitively (holes vs safe), (iii) debug without function approximation error. Standard simple gridworld.

Q: How many trajectories/steps per outer iteration?
A: 2048 environment steps per rollout per iteration. Enough for stable occupancy estimation and critic training in this small environment. Didn't push sample efficiency; systematic study would vary this.

Q: How do you measure "success" in your experiments?
A: Three criteria: (1) **constraint satisfaction**—$J_c(\pi) \leq \tau$; (2) **reward**—$J_r(\pi)$ as high as possible; (3) **stability**—no blow-up or wild oscillations. On these, Dual–SPMA $\approx$ NPG–PD.

Slide 31–32 — *Results: Entropy-Regularized RL & Constrained CMDP*

Speaker Script (Ahmed)

Slide 31 (Entropy-Regularized):

- "No constraints, just entropy bonus."
- "Left: L (Lagrangian) vs iteration—both methods decrease it."
- "Right: occupancy sum ≈ 1 (sanity check)."
- "SPMA and NPG–PD behave similarly."

Slide 32 (Constrained CMDP):

- "Left: reward $J_r(\pi_k)$ increases over iterations."
- "Right: violation $J_c(\pi_k) - \tau$ goes below zero (constraint satisfied)."
- "Both methods reach similar final performance."

Related Q&A

Q: Which method converges faster?
A: Depends on hyperparameters. Typically SPMA takes larger, more "aggressive" steps, can reduce violation faster initially, but slightly more sensitive to hyperparameters. NPG–PD smoother but needs careful tuning of β. Final performance broadly comparable.

Q: How do you handle exploration?
A: Stochastic policy (softmax) naturally explores early. Entropy-regularized experiments explicitly add entropy term. No extra exploration bonuses.

Slide 33–34 — *Results: SPMA vs NPG–PD & Occupancy Heatmaps*

Speaker Script (Ahmed)

Slide 33 (Comparison):

- "Left: reward J_r vs iterations—similar once constraint satisfied."
- "Right: violation—both bring close to zero, different curves."
- "SPMA: larger steps, sometimes faster, more hyperparameter-sensitive."
- "NPG–PD: smoother, needs careful β tuning."
- "Conclusion: Dual–SPMA is competitive, SPMA is viable policy optimizer."

Slide 34 (Heatmaps):

- "**Left**: unconstrained policy—explores broadly, includes unsafe states."
- "**Right**: safety-constrained—avoids unsafe states (occupancy ≈ 0)."
- "Constraint reflected in agent behaviour, not just numerically."

Related Q&A

Q: Do both methods satisfy the constraint?
A: Yes. Both reduce $J_c(\pi) - \tau$ toward zero. Violation plots approach/oscillate around zero. Learned policies' heatmaps clearly avoid unsafe states.

Q: Are advantages estimated with critic or MC?
A: Standard actor–critic: critic is value network trained with TD, advantage computed as $A = \hat{Q} - V$ via GAE or simple TD returns. Both SPMA and NPG–PD share same critic architecture.

Q: Do you use MLE estimator for main results?
A: No, we use MC estimator for main FrozenLake experiments. MLE implemented and tested briefly; full comparison is future work.

Slide 35–36 — *Takeaways, Limitations & Future Work*

Speaker Script (Ahmed)

Slide 35 (Takeaways & Limitations):

- "**Recipe**: Convex MDP → Fenchel dual → SPMA-based shaped-reward RL"
- "Built: Dual–SPMA loops, NPG–PD baseline, occupancy estimators"
- "Result: Dual–SPMA is stable, satisfies constraints, competitive with NPG–PD"
- "**Limitations**: only tabular settings; MLE estimator needs stress-testing; hyperparameter sensitivity not fully explored"

Slide 36 (Future Work):

- "**Scaling up**: larger CMDPs (MuJoCo), continuous states/actions"
- "**Better occupancy estimation**: MLE in high dimensions"
- "**More convex objectives**: risk-sensitive, imitation learning"
- "**Theory**: convergence rates and sample complexity bounds"

Related Q&A

Q: What are the main limitations of your project?
A: The big three are: (1) we only test in small tabular environments, so we don't claim scalability yet; (2) our MLE estimator is implemented but not fully stress-tested on large problems; and (3) we don't provide a formal convergence-rate analysis for the full Dual–SPMA algorithm with function approximation—only for the inner SPMA and the baseline, via existing theory.

Q: How hard would it be to extend this to MuJoCo or other continuous-control tasks?
A: Conceptually it's straightforward: we keep the same dual update and shaped-reward view, but replace the tabular policy with a neural policy over continuous actions (e.g., Gaussian). The main challenges are practical: occupancy estimation in continuous spaces (we'd rely on feature-based or MLE estimators), tuning SPMA or NPG–PD in higher dimensions, and computational cost. It's doable but beyond the scope of this course project.

Q: What would a convergence-rate analysis of Dual–SPMA look like?
A: It would likely combine: (1) regret bounds for the dual player (FTL/gradient); (2) convergence or regret bounds for SPMA as a policy optimizer under shaped rewards; and (3) error terms due to approximate occupancy estimation and finite inner loops. Those could be turned into an $\mathcal{O}(1/\sqrt{T})$ or better bound on the primal–dual gap of averaged iterates. We didn't attempt this full derivation, so we list it explicitly as future work.

Slide 37–38 — *Q&A & References*

Speaker Script

Slide 37 (Q&A):

- "Thanks for listening—that concludes our presentation."
- "We're happy to take questions."
- (Remind: "Pegah: background; Danielle: Fenchel dual; Shervin: related work & method; Ahmed: experiments.")

Slide 38 (References):

- "Main references: Zahavy et al. (convex MDPs), Asad et al. (SPMA), Ding et al. (NPG–PD), Barakat et al. (MLE occupancy)."

Related Q&A

(Use the complete Q&A bank on the next page for any questions during Q&A session.)

Complete Q&A Bank by Category

1. Big-picture & motivation (Q1–Q5)

Q1. What's the main takeaway of your project in one sentence?

A1. We show that if you take the convex-MDP framework of Zahavy et al. and plug in SPMA as the policy optimizer, you get a simple "Dual–SPMA" recipe that (i) is easy to implement as shaped-reward RL, and (ii) performs competitively with a strong primal–dual baseline, NPG–PD, on constrained safety tasks.

Q2. Why do we even care about *convex* MDPs instead of just standard RL?

A2. Standard RL maximizes a linear function of the occupancy measure—expected return—so it's great when you only care about reward. Many real problems add structure: safety constraints, exploration terms, risk penalties, imitation losses, etc. Many of those can be written as convex functionals of the occupancy d_π. Convex MDPs give a unified way to handle these goals while preserving nice optimization properties like existence of a saddle point.

Q3. How is this different from just "constrained RL with a Lagrange multiplier"?

A3. Classic constrained RL focuses on a particular structure: reward plus linear constraints. Convex MDPs are more general: you minimize an arbitrary convex function $f(d_\pi)$ over the convex occupancy polytope. Constrained RL is one example where f adds a penalty or indicator for constraint violations; entropy-regularized RL and some imitation objectives are *other* examples. Our framework—and the Fenchel duality step—work for all of them, not just linear constraints.

Q4. Why do you compare against NPG–PD in particular? Why not PPO or TRPO?

A4. The core of our work is about **convex objectives & dual variables**, not generic performance on Atari. NPG–PD is a principled *primal–dual* algorithm specifically designed for constrained MDPs, with guarantees on both optimality and constraint violation. It also uses a geometry-aware policy update (natural gradient), which makes the comparison with SPMA conceptually clean. PPO/TRPO are great empirically, but they're not formulated directly as primal–dual CMDP algorithms.

Q5. Is Dual–SPMA supposed to *beat* NPG–PD?

A5. For a term project we don't claim to beat state of the art. What we want to show is that SPMA is a **viable policy player** inside the convex-MDP dual framework: it's stable, respects constraints, and gives reward comparable to NPG–PD. Beating a heavily tuned algorithm like PPO or NPG–PD consistently would require much more engineering and experimentation than we can do here.

2. Occupancy measures & convex MDP formulation (Q6–Q11)

Q6. Why introduce occupancy measures at all? Why not just work with $J(\pi)$?

A6. Because in terms of policy parameters, $J(\pi)$ is generally non-convex. In terms of occupancy, return is **linear**: $\langle r, d_\pi \rangle = (1-\gamma)J(\pi)$. Many interesting extras (entropy, penalties, constraints) are convex in d. The set of valid occupancies is a convex polytope defined by linear flow constraints. So in d-space we get a clean convex program: minimize a convex $f(d)$ over a convex set $\mathcal{D}$.

Q7. Can you restate the key identity relating $J(\pi)$ and d_π?

A7. Yes: we define

$$d_\pi(s,a) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \Pr_\pi(s_t = s, a_t = a).$$

Then

$$\langle r, d_\pi \rangle = \sum_{s,a} r(s,a)\, d_\pi(s,a) = (1-\gamma)J(\pi).$$

So up to the constant factor $1-\gamma$, the RL objective is just a linear function of the occupancy measure.

Q8. What exactly is $\mathcal{D}$, the set of feasible occupancies?

A8. $\mathcal{D}$ is the set of $d \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ that satisfy the Bellman flow constraints plus non-negativity: for every state s,

$$\sum_a d(s,a) = (1-\gamma)\rho(s) + \gamma \sum_{s',a'} P(s|s',a')\, d(s',a'), \quad d(s,a) \geq 0.$$

This says "discounted occupancy flowing *into* a state equals initial occupancy plus discounted flow from other states." It's a convex polytope.

Q9. Is there a 1–1 correspondence between policies and occupancy measures?

A9. Under mild regularity conditions (e.g., communicating MDP, $\gamma < 1$), each stationary policy has a unique discounted occupancy measure, and each valid occupancy in $\mathcal{D}$ corresponds to at least one stationary policy (you can recover $\pi(a|s) = d(s,a)/\sum_{a'} d(s,a')$ where the denominator is nonzero). So working in d-space is equivalent, up to degeneracies.

Q10. Why do you call $f(d)$ "convex" when there's a minus reward term $-\langle r, d\rangle$?

A10. Because $-\langle r, d\rangle$ is linear in d, and linear functions are both convex and concave. When we add convex terms like entropy or penalties, the sum stays convex. So all the examples on the "Convex MDP: Examples" slide fit into "convex f" without breaking anything.

Q11. What assumptions do you need for your convex MDP formulation?

A11. We assume a discounted MDP with $\gamma < 1$, bounded reward and cost, and that we restrict to stationary policies. For convexity, we require f to be proper, closed, and convex on the occupancy polytope $\mathcal{D}$. For the Fenchel reduction, we also need standard conditions that guarantee the biconjugate equality and existence of a saddle point—essentially the Fenchel–Moreau assumptions.

3. Fenchel duality & saddle-point formulation (Q12–Q18)

Q12. Why bring in Fenchel conjugates at all?

A12. Fenchel conjugates give us an identity:

$$f(d) = \sup_y \{\langle y, d\rangle - f^*(y)\}.$$

Plugging this into $\min_{d\in\mathcal{D}} f(d)$ lets us rewrite the convex MDP as

$$\min_{d\in\mathcal{D}} \max_y \{\langle y, d\rangle - f^*(y)\}.$$

So we get a saddle-point game with a natural dual variable y. This structure is what allows us to run two-player algorithms: one player updates y, the other updates the policy.

Q13. What does the dual variable y represent intuitively?

A13. It's a vector of the same dimension as d—one coordinate per state–action pair. In the Fenchel dual, it's the variable that "linearizes" the convex objective. Operationally, for the policy player, y is just a shaped reward field: we treat $r_y(s, a) = -y(s, a)$ as the reward. Different choices of y encourage the policy to put occupancy where f wants it.

Q14. Why is the solution of the min–max at a saddle point?

A14. Because in convex-concave problems, under suitable conditions, the minimax and maximin values coincide and are achieved at a pair (π^*, y^*) satisfying

$$L(\pi^*, y) \le L(\pi^*, y^*) \le L(\pi, y^*)$$

for all π, y. That point is simultaneously optimal for the min player (policy) and the max player (dual), and is called a saddle point. Solving the min–max is thus equivalent to finding such a saddle point.

Q15. Does your algorithm *guarantee* convergence to the saddle point?

A15. In the idealized tabular, exact-oracle setting, existing theory (Zahavy et al. + SPMA paper) shows that if the dual player and the policy player both use low-regret algorithms, the average iterates converge to a saddle point. In our implementation we have function approximation, noisy occupancy estimates, and finite inner loops, so we don't claim a formal convergence proof. Instead we check empirically that the saddle objective and constraints behave sensibly.

Q16. Why is alternating updates (policy step then dual step) reasonable, given that vanilla Gradient Descent–Ascent can diverge?

A16. It's true that naive GDA can oscillate. We're not using plain GDA: we use specific online-learning style updates. The dual player uses a simple FTL/gradient step in a convex Euclidean space; the policy player uses SPMA, which is a mirror-descent method in logit space with convergence guarantees. Online saddle-point theory says that when both players have low regret, the average of their strategies converges to a saddle point, so alternating these particular updates is justified.

Q17. How do you get the conjugate $f^*(y)$ in the entropy-regularized case?

A17. For

$$f(d) = -\langle r, d\rangle + \alpha \sum d \log d,$$

you can complete the square in log-space / use standard convex analysis to show

$$f^*(y) = \alpha \log \sum_{s,a} \exp((y(s, a) + r(s, a))/\alpha).$$

Its gradient is a softmax: $\nabla f^*(y) = \text{softmax}((y + r)/\alpha)$. That's the expression we use in the dual update.

Q18. Why do you write $\min_d \max_y$ instead of $\max_y \min_d$? Are you allowed to swap min and max?

A18. We derive the saddle-point form by directly applying Fenchel–Moreau:

$$f(d) = \sup_y \{\langle y, d\rangle - f^*(y)\},$$

then taking $\min_d$ outside. Under standard convexity/closedness assumptions, there is no duality gap and the saddle-point value equals both the minimax and maximin. So while we don't literally swap orders in the algebra, we rely on these conditions to interpret the solution as an equilibrium of the game.

4. Shaped-reward RL and "why RL" (Q19–Q21)

Q19. Why not just solve the convex program over d with CVX or some LP solver?

A19. You *could* if the MDP is small and fully known. In our setting we assume a **model-free** RL scenario: we don't know the transition probabilities, we only interact with the environment by sampling trajectories. That means we can't write down $\mathcal{D}$ explicitly or compute exact expectations. RL gives us a way to approximate $\min_\pi \langle y, d_\pi\rangle$ from samples by treating $-y$ as a reward and running a policy optimizer.

Q20. Does the shaped reward $r_y = -y$ break the Markov property or anything?

A20. No, because y is a function of the current state–action pair (s, a) only; it doesn't introduce dependence on the entire trajectory. The environment dynamics $P(s'|s, a)$ are unchanged, and the reward at each step is just a different function of (s, a). So the process is still a valid MDP, just with a different reward.

Q21. If you can plug "any RL algorithm" into the policy player, why specifically SPMA?
A21. The convex-MDP theory says any RL algorithm could in principle be used as the policy player. We choose SPMA because (i) it has **strong convergence results** in tabular MDPs, (ii) it's geometry-aware, operating in logit space with a softmax mirror map, (iii) its loss looks similar to PPO/MDPO, so it's practical to implement, and (iv) the SPMA paper already shows it's competitive with PPO/TRPO in continuous control.

5. SPMA details & comparison to other policy gradients (Q22–Q27)

Q22. How is SPMA different from vanilla policy gradient?
A22. Vanilla PG minimizes $L_{\text{PG}} = -\mathbb{E}[\log \pi(a|s)A(s, a)]$ and takes gradient steps in parameter space. SPMA instead uses an **exponential-gradient / mirror-descent update in logit space**. The actor loss includes a regularizer

$$\frac{1}{\eta}(\exp(\Delta \log \pi) - 1 - \Delta \log \pi)$$

which penalizes large changes in log-probabilities. This gives better control over how far the policy moves per step and leads to faster convergence in the tabular theory.

Q23. Where does your SPMA loss come from?
A23. It's the convex surrogate corresponding to the tabular SPMA update. If you linearize the value function and consider an exponential-gradient step in logit space, the associated Bregman divergence gives rise to exactly that $\exp -1 - \Delta \log \pi$ term. The SPMA paper derives it formally; we implement it as a differentiable loss so we can use standard automatic differentiation.

Q24. Do you enforce the $[1 + \eta A]_+$ and normalization from the tabular update?
A24. In the tabular theory, the closed-form update is

$$\pi_{t+1}(a|s) \propto \pi_t(a|s)[1 + \eta A(s, a)]_+.$$

In our neural approximation we don't directly enforce this per-state formula. Instead we optimize the SPMA loss over mini-batches of trajectories; the softmax layer plus the KL-like regularizer implicitly enforce a similar behaviour. So we're using the **function-approximation version** of SPMA instead of the exact tabular update.

Q25. How do you choose the SPMA step size η?
A25. We use a simple Armijo line search over a small discrete set of candidate η values. For each candidate we compute the SPMA loss on a minibatch and pick the largest η that produces a sufficient decrease. This is inspired by the SPMA paper and by MDPO/TRPO line-search practices.

Q26. Why is SPMA "geometry-aware"?
A26. Because it does mirror ascent with the log-sum-exp mirror map, which is tailored to the simplex geometry. Instead of taking Euclidean steps in parameter space and then projecting back onto the simplex, SPMA treats the logits as dual variables and the softmax as the mirror map. This respects the probability constraints and tends to produce well-behaved updates when probabilities are near 0 or 1.

Q27. How does SPMA compare to natural policy gradient conceptually?
A27. Both are trying to account for the geometry of the policy space. Natural PG uses the Fisher information matrix to modify the gradient direction and step size; SPMA uses a Bregman divergence in logit space. In small tabular problems you can show they are closely related; in our setting we treat them as two reasonable but different choices of geometry-aware update and compare them empirically.

6. Dual-update & FTL choice (Q28–Q30)

Q28. Why do you use a simple gradient/FTL step for the dual variable y?
A28. The dual variable lives in a straightforward Euclidean space and the dual objective is convex with a simple gradient $\hat{d}_\pi - \nabla f^*(y_k)$. For such problems, basic online convex optimization algorithms like gradient ascent or FTL are known to have sublinear regret. They're easy to implement and come with clear theoretical guarantees, so there's no need for more complex methods.

Q29. Are there any constraints on y? Do you project it?
A29. In the entropy-regularized case, y is unconstrained—we don't project. For CMDPs we sometimes parameterize the dual as $y_\lambda = \lambda c - r$ with $\lambda \geq 0$, so the only constraint is non-negativity on λ, which we enforce with a $[\cdot]_+$ projection (same as NPG–PD). More sophisticated bounds or regularizers on y are possible but we didn't explore them in this project.

Q30. Is the dual step size α sensitive?
A30. Yes, like most gradient methods it needs to be in a reasonable range: too small and progress is slow; too large and you can oscillate. In practice we tuned α on a log scale and picked values where the dual objective and the constraint violation decreased smoothly. Because our setting is tabular and relatively small, this wasn't too painful.

7. Occupancy estimation (Q31–Q34)

Q31. Why not use exact occupancies in FrozenLake instead of Monte Carlo?

A31. We intentionally use Monte Carlo to simulate the **model-free** setting most RL algorithms operate in—where you don't have access to $P(s'|s,a)$. It also keeps the pipeline more consistent with what you'd do in larger environments. Using the exact occupancy is possible in FrozenLake, but we'd then be solving a much easier problem that doesn't stress the estimator.

Q32. How do you ensure your Monte Carlo occupancy estimate is valid?

A32. We compute

$$\hat{d}_\pi(s,a) = \frac{1-\gamma}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}\gamma^t \mathbf{1}\{s_t^{(i)} = s, a_t^{(i)} = a\}.$$

By construction this is non-negative. As a sanity check, we track $\sum_{s,a}\hat{d}_\pi(s,a)$ over training and verify it stays very close to 1; that's the plot on our entropy-regularized results slide. Small deviations come from truncating trajectories at finite T.

Q33. What's the point of the MLE-based occupancy estimator?

A33. The tabular MC estimator's variance scales with $|\mathcal{S}||\mathcal{A}|$; in large MDPs that's bad. The MLE estimator instead fits a log-linear model $\lambda_\omega(s,a) \propto \exp(\omega^\top \phi(s,a))$ by maximum likelihood on samples. Its error depends mostly on the feature dimension, not the raw state–action count. So it's a more scalable idea we wanted to prototype, although we only briefly tested it here.

Q34. Do you actually use the MLE estimator for the main results, or just MC?

A34. No, we use MC estimator for main FrozenLake experiments. MLE implemented and tested briefly; full comparison is future work.

8. Experiments & empirical behaviour (Q35–Q40)

Q35. Why did you choose FrozenLake 4×4 as your main environment?

A35. Wanted **tabular** CMDP to (i) compute/visualize occupancies, (ii) understand safety constraint intuitively (holes vs safe), (iii) debug without function approximation error. Standard simple gridworld.

Q36. Do both methods actually satisfy the safety constraint?

A36. Yes. Both reduce $J_c(\pi) - \tau$ toward zero. Violation plots approach/oscillate around zero. Learned policies' heatmaps clearly avoid unsafe states.

Q37. Which method converges faster in your experiments?

A37. Depends on hyperparameters. Typically SPMA takes larger, more "aggressive" steps, can reduce violation faster initially, but slightly more sensitive to hyperparameters. NPG–PD smoother but needs careful tuning of β. Final performance broadly comparable.

Q38. How many trajectories / steps do you need per outer iteration?

A38. 2048 environment steps per rollout per iteration. Enough for stable occupancy estimation and critic training in this small environment. Didn't push sample efficiency; systematic study would vary this.

Q39. Are the advantages estimated with a critic network or Monte Carlo?

A39. Standard actor–critic: critic is value network trained with TD, advantage computed as $A = \hat{Q} - V$ via GAE or simple TD returns. Both SPMA and NPG–PD share same critic architecture.

Q40. How do you handle exploration in these experiments?

A40. Stochastic policy (softmax) naturally explores early. Entropy-regularized experiments explicitly add entropy term. No extra exploration bonuses.

9. Limitations & future work (Q41–Q44)

Q41. What are the main limitations of your project?

A41. The big three are: (1) we only test in small tabular environments, so we don't claim scalability yet; (2) our MLE estimator is implemented but not fully stress-tested on large problems; and (3) we don't provide a formal convergence-rate analysis for the full Dual–SPMA algorithm with function approximation—only for the inner SPMA and the baseline, via existing theory.

Q42. How hard would it be to extend this to MuJoCo or other continuous-control tasks?

A42. Conceptually it's straightforward: we keep the same dual update and shaped-reward view, but replace the tabular policy with a neural policy over continuous actions (e.g., Gaussian). The main challenges are practical: occupancy estimation in continuous spaces (we'd rely on feature-based or MLE estimators), tuning SPMA or NPG–PD in higher dimensions, and computational cost. It's doable but beyond the scope of this course project.

Q43. Could you use other policy optimizers in place of SPMA?

A43. Yes. The meta-algorithm from Zahavy et al. lets you plug in any RL method as the policy player. We chose SPMA because of its theory and similarity to modern methods, but in principle you could try PPO, TRPO, or MDPO. An interesting piece of future work would be a systematic empirical comparison of different policy players inside the same convex-MDP dual framework.

Q44. What would a convergence-rate analysis of Dual–SPMA look like?

A44. It would likely combine: (1) regret bounds for the dual player (FTL/gradient); (2) convergence or regret bounds for SPMA as a policy optimizer under shaped rewards; and (3) error terms due to approximate occupancy estimation and finite inner loops. Those could be turned into an $\mathcal{O}(1/\sqrt{T})$ or better bound on the primal–dual gap of averaged iterates. We didn't attempt this full derivation, so we list it explicitly as future work.

10. Misc / notation and implementation details (Q45–Q48)

Q45. What's the difference between y and λ in your notation?
A45. y is the general Fenchel dual variable over state–action pairs in the convex-MDP formulation. λ is the scalar Lagrange multiplier for CMDP safety constraints. In the constrained case we sometimes define a structured dual variable $y_\lambda(s,a) = \lambda c(s,a) - r(s,a)$, so y is parameterized by λ.

Q46. Why does your "Convex MDP examples" slide use μ as a penalty weight instead of λ?
A46. We intentionally use μ there to distinguish a **fixed penalty weight** in a penalized objective from the **Lagrange multiplier** λ that evolves as a dual variable. This avoids confusion between "tunable hyperparameter" and "optimization variable".

Q47. What exactly are $J_r(\pi)$ and $J_c(\pi)$?
A47. They are just discounted returns with respect to reward and cost: $J_r(\pi) = \mathbb{E}_\pi[\sum_t \gamma^t r(s_t, a_t)]$, $J_c(\pi) = \mathbb{E}_\pi[\sum_t \gamma^t c(s_t, a_t)]$. We estimate them by Monte Carlo averaging over rollouts.

Q48. How do you measure "success" in your experiments?
A48. We consider three main criteria: (1) **constraint satisfaction**—$J_c(\pi)$ should be at or below τ; (2) **reward**—$J_r(\pi)$ should be as high as possible given the constraint; and (3) **stability**—dual variables and occupancies shouldn't blow up or oscillate wildly. On those metrics, Dual–SPMA behaves about as well as NPG–PD in our tests.