OVERVIEW

SG NOMA with Q Learning

GOAL

- Task: 12 users, 4 triplets that maximized NOMA utility
- Learning Signal:
    - Finish triplet
    - Get reward Uw (sum rate U times mean weight w)
    - No reward for partial choices = sparse rewards that are aligned to goal

STEP 1: STATE

- remaining: users still unassigned
- current: partial triplets being built (0, 1 or 2 users chosen so far)
- makeStateKey(remaining, current):
    - Convert state to string key for Q table
    - remaining is a bitstring (ex. 110010…)
    - current is an orderless printout of chosen indices and sorted with sort(current)
    - Concatenated via remaining bits | current indices
    - Orderless so that state is unchanged if same two users added in different orders

STEP 2: ACTION

- Actions = find(remaining)
- Action a: add user a to current triplet
- Pick action a:
    - remaining(a) = false; current = [current , a];
    - If numel(current) < 3: no reward yet (partial triplet)
    - If == 3: complete, then compute reward with tripletUtility(current points, weights) and reset current = [], start forming next triplet

STEP 3: REWARD

- THIS IS BASICALLY OUR PREVIOUS WORK
- tripletUtility(current points, weights)
- NOMA rates for triplets using our NOMA model (pathless, rayleigh, power splits, etc.)
- U = R1 + R2 + R3
- Matches calcScore, training objective = evaluation metric

STEP 4: UPDATING Q TABLE

- Q is hashmap (containers.Map) from (state,action) -> value
    - sKey = makeStateKey()
    - qakey = makeQAKey(sKey, a) -> <state>|a:<index>
- Training policy
    - epsilon greedy over available actions
    - Probability epsilon to pick random action (explore), else pick argmaxQ action (exploit)
    - epsilon decays each episode: eps = max(eps end, eps start * eps decay^t)

- Update (Bellman)
    - Q(s,a) <- (1-alpha)(Q(s,a)) + alpha(r + gamma*maxQnext)
        - r triplet reward
        - maxQnext = max_{a'} Q(s',a'). If terminal (no actions, maxQnext = 0)
        - alpha = learning rate, gamma = discount (keep some value for future triplets)

STEP 5: LOOP EPISODES & LOGGING

- One episode = start with all users (remaining = true)
- Form all 4 triplets via actions until no users remain
- Score is the utility of an episode (sum of rewards)
- episode best score to keep track for the best score so far
- logs will print it every few checkpoints

STEP 6: POLICY AFTER TRAINING

- Set remaining = true(1,N); current=[]; triplets=[];
- At each state select argMaxQ action (no epsilon exploration)
- Build triplets until done
- If unable to produce N/3 triplets, back to best result observed so far

HELPER FUNCTIONS

- makeStateKey(rem, curr): orderless current, shrink state space, reusable q table entries
- makeQAKey(sKey, a): append action to state key, (s,a) entries
- argmaxQ(Q, sKey, actions): available actions, pick action with highest stored Q val (0 for unseen keys)
- maxQ(Q, sKey, actions): return max_a Q(s,a). If no actions return 0