# Punto 1

## Pricing

## Pricing e bidding interagiscono

L’algoritmo di bidding deve calcolare la propria valuation e biddare in base ai risultati dell’algoritmo di pricing. Ogni giorno, per 100 giorni, viene deciso un price e vengono effettuate 1000 bid. In particolare:

1. Viene deciso un prezzo dall’algoritmo di pricing in base alle informazioni precedenti
2. Una volta scelto il prezzo, viene decisa la valuation corrispondente. La valuation dovrebbe essere l’utility corrispondente ad ogni click sull’ad, quindi corrisponde al prezzo per la probabilità che l’utente compri a quel prezzo.

Questo prodotto è uguale alla reward curve, che l’agent stima ad ogni time step.

Il problema è che la stima della valuation deve avvenire prima che si verifichino le effettive aste, e quindi prima di poter aggiornare la reward curve con i dati delle bid. Inizializzando la media delle reward a 0 come prassi, il risultato è che la valuation viene sempre messa a zero, non vengono mai vinte aste, e quindi non viene mai aggiornata la reward curve.

La soluzione che utilizziamo è, anziché usare la media, di usare l’UCB se per il pricing usiamo GP-UCB (UCB limitato dal prezzo corrispettivo, per le prime auctions dove è probabile che andiamo in perdita) e fare sampling se usiamo GP-TS.

1. L’algoritmo di bidding prende come input la valuation fissata, e usa multiplicative pacing per fare 1000 aste al giorno, vincendone un numero *nWins*.
2. Per ogni win delle aste, un utente visita il sito e con una probabilità dettata dalla demand curve acquista effettivamente il prodotto. Se non vengono vinte bid (la valuation è troppo bassa per poter competere con gli altri, dato che l’algoritmo di bidding non bidda mai più della propria valuation per evitare rischi), viene messo 0 alla reward del prezzo.
3. L’algoritmo di pricing usa le informazioni sui price settati in precedenza e i Gaussian Processes per decidere i prezzi dei giorni successivi.

# Punto 2

## Pricing

Per settare una pricing strategy discretizzando i prezzi uso UCB + sliding window. Per decidere il numero di arm (la discretizzazione) non utilizzo però T come avrei fatto in uno stochastic environment, bensì W, che è la “memoria” di SWUCB.

Sliding window UCB soffre nelle performance in concomitanza con multipli della dimensione della window, perché escono dalla window i pull iniziali degli arm sconvenienti e quindi li deve riesplorare tutti. L’andamento è perciò a “ondine”.

Attenzione! Per calcolare il regret ho utilizzato la stessa discretizzazione sia per SW UCB che per la baseline

Immagine che contiene testo, linea, Diagramma, schermata

Descrizione generata automaticamenteImmagine che contiene testo, schermata, linea, Diagramma

Descrizione generata automaticamente

Per CUMSUM UCB utilizzo una discretizzazione diversa: posso permettermi di usare più prezzi rispetto a SW UCB. Supponendo di conoscere il numero di cambiamenti di prezzo (assunzione usata anche per calcolare la dimensione della window W), discretizzo con sensibilità