RL use case: Retail store management problem

At each month $t \geq 1$:

- x_t units in the warehouse
- a_t : action = nb of units ordered at the end of the month
- D_t : demand for that month (assume all units are sold at the end of the month)

Problem

- p = selling price of 1 unit
- c = cost for ordering 1 unit (buying price) $(c < p)$
- h = storage cost of 1 unit
- K = overall transportation cost

Constraints:

- if demand $D_t >$ nb of available units $x_t + a_t$, then some clients are left aside.
- maximum storage capacity is of M units

Casting this setting as a Markov Decision Process:

- State space: $S = \{0, 1, \dots, M\}$ of maximum storage capacity
- Actions available from state $x \in S$: $A(x) = \{0, 1, \dots, M-x\}$
- Dynamics:
  - D_t is random: $D_t \overset{iid}{\sim} P_D$
  - next state: $x_{t+1} = \left(\min\{x_t + a_t, M\} - D_t\right)_+$

* reward: R_{t+1} given by

$$R_{t+1} = p \cdot \underbrace{\min\left\{D_t, \underbrace{\min\{x_t + a_t, M\}}_{\text{what we have before selling}}\right\}}_{\text{what we actually sold}} - c \cdot \left(\underbrace{\min\{x_t + a_t, M\} - x_t}_{\text{number of units we actually bought}}\right)$$

$$\underbrace{- h x_t}_{\text{storing cost}} \quad \underbrace{- K \mathbb{1}_{\{a_t > 0\}}}_{\text{fixed transportation cost}}$$

```r
# re-initializing variables
rm(list=ls())

# setting inventory parameters
M <- 15
gamma <- 0.99;
q <- 0.1;
pD <- q*(1-q)^(0:(M-1));
pD[1+M] <- 1 - sum(pD);
```

$p_D = (p_D(0), p_D(1), \dots, p_D(M))$

```r
# random demand (drawn from p, starting at 0)
rdemand <- function(p){
  sum(runif(1)>cumsum(p));
}
```

returns a random draw D from p

```r
# reward function
reward <- function(x, a, d){
  K <- 0.8 # transportation cost
  c <- 0.5 # cost per unit
  h <- 0.3 # stocking cost per unit
  p <- 1 # selling price per unit

  -K * (a>0) -c*max(0, min(x+a, M)-x) - h*x + p * min(d, x+a, M);
}
```

$\underbrace{-K * (a>0)}_{\text{fixed cost}}$ $\underbrace{\max(0, \min(x+a, M)-x)}_{\text{what we actually ordered}}$

$\min(d, x+a, M) = \min(d, \underbrace{\min(x+a, M)}_{\text{what we have before selling}})$

$\rightarrow R_{t+1}$ as a function of (x_t, a_t, D_t)

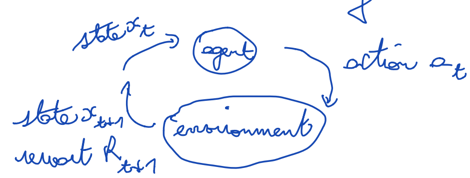
```r
# transition function
nextState <- function(x, a, d){
  max(0, min(x+a, M)-d);
}
```

$x_{t+1} = \left(\min\{x_t + a_t, M\} - D_t\right)_+$

```r
# simulating the inventory sales
simu <- function(n, pi){
  R <- array(0,n);
  X <- M;
  for (t in 1:n){
    D <- rdemand(pD);
    R[t] <- reward(X, pi[X+1], D);
    X <- nextState(X, pi[X+1], D);
  }
  R;
}
```

π: deterministic policy $\quad \pi : S \to A,\ x \mapsto \pi(x)$

simulates n iterations of




```r
# First policy: always command 2 machines
pi = array(2, M+1);
n <- 200;
```

$\forall x \in S,\ \pi(x) = 2 \iff \pi = \underbrace{(2, 2, \dots, 2)}_{M+1}$

```r
# simulation
R <- simu(n, pi);
V <- cumsum(R * gamma^(0:(n-1)));
plot(0:n, c(0,V), type='l')
```

```
# building the transition kernel and the expected reward function
trans <- list()
rew <- list()
for (a in 0:M){
  P <- matrix(0, ncol = 1+M, nrow = 1+M);
  r <- vector(mode='numeric', length=1+M)
  for (x in 0:M) for (d in 0:M){
    P[1+x, 1+nextState(x, a, d)] <- P[1+x, 1+nextState(x, a, d)] + pD[1+d];
    r[1+x] <- r[1+x] + pD[1+d] * reward(x, a, d);
  }
  trans[[1+a]] <- P;
  rew[[1+a]] <- r;
}
```

$p(s'|s,a)$

$\sum_{s'}\sum_{r} r \, p(s',r|s,a)$

$\vec{v}_\pi = \vec{r} + \gamma P \vec{v}_\pi$

$(Id - \gamma P)\vec{v}_\pi = \vec{r}$

$\vec{v}_\pi = (Id - \gamma P)^{-1}\vec{r}$

```
# policy evaluation
policyValue <- function(pol){
  P <- matrix(0, ncol=1+M, nrow=1+M);
  r <- as.vector(rep(0, 1+M));
  for (x in 0:M){
    a <- pol[1+x];
    P[1+x,] <- trans[[1+a]][1+x,];
    r[1+x] <- rew[[1+a]][1+x];
  }
  solve(diag(1+M)-gamma*P, r);
}

print(policyValue(pi));
```

policy evaluation

---

planning

```
# searching for the best policy:
# Bellman operator
BellmanOperator <- function(V){
  res = list()
  res$V <- V;
  res$pol <- rep(0, 1+M);
  for (x in 0:M){
    Q <- rep(0, 1+M);
    for (a in 0:M){
      Q[1+a] <- rew[[1+a]][1+x] + gamma * trans[[1+a]][1+x,] %*% V;
    }
    res$V[1+x] <- max(Q);
    res$pol[1+x] <- -1+which.max(Q);
  }
  res
}

# Value iteration
valueIteration <- function(){
  res = list();
  res$V <- as.vector(rep(0, 1+M));
  res$pol <- rep(0, 1+M);
  oV <- res$V+1;
  while (max(abs(oV-res$V))>1e-4){
    oV <- res$V;
```

```
    res <- BellmanOperator(res$V);
  }
  res
}

# finding the optimal solution by value iteration
sol <- valueIteration();
print(sol$V);
n <- 5/(1-gamma);
plot(c(0, n), sol$V[1+M]*c(1,1), xlim=c(0,n), ylim=c(0, 1.5*sol$V[1+M]), type='l', col='red', lwd=3)
for(k in 1:50){
  R <- simu(n, sol$pol);
  V <- cumsum(R * gamma^(0:(n-1)));
  lines(0:n, c(0,V), type='l')
}

# finding the optimal policy by policy iteration
policyIteration <- function(){
  res <- list();
  res$V <- as.vector(rep(0, 1+M));
  res$pol <- floor((1+M)*runif(1+M));
  opol <- res$pol+1;
  while (any(opol != res$pol)){
    opol <- res$pol;
    res <- BellmanOperator(policyValue(opol));
  }
  res
}
```

```
# if the parameters of the MDP are unknown: Q-learning

Qlearning <- function(n){
  Q <- matrix(0, nrow = 1+M, ncol = 1+M)
  epsilon <- 0.9# epsilon-greedy policy
  X <- M # we start full
  for (t in 1:n){
    A <- -1+which.max(Q[X,])
    if (runif(1)<epsilon) {A <- floor((1+M)*runif(1))}
    D <- rdemand(pD)
    R <- reward(X, A, D);
    nX <- nextState(X, A, D);
    #if (X==0) print(c(X, A, D, R, nX))
    alpha = 1/t^0.3; #1/sqrt(t);
    delta <- R + gamma * max(Q[1+nX,]) - Q[1+X, 1+A]
    Q[1+X, 1+A] <- Q[1+X, 1+A] + alpha * delta
    X <- nX
  }
  res <- list()
  res$Q <- Q
  res$pol <- rep(0, 1+M)
  for (x in 0:M){ res$pol[1+x] <- -1+which.max(Q[1+x,])}
  res
}
```

```
# Let us now compare the optimal solution...
sol <- policyIteration()

# ... with the solution obtained by Q-learning:
res <- Qlearning(50000)

# results:
sol$pol
res$pol
c(policyValue(res$pol)[1+M], sol$V[1+M])
```