

Data 405 Assignment 4

Q1 code:

```
set.seed(2025)

# load data
data(EuStockMarkets)
DAX <- EuStockMarkets[, "DAX"]

# (a) store dat in object
DAXlogreturn <- diff(log(DAX))

# (b) find drift (the mean of log-returns)
drift <- mean(DAXlogreturn)

# (c) acf of log-returns (the lin dependence)
pdf(NULL)
ansc <- acf(DAXlogreturn, main = "ACF of DAX log-returns")
ansc

##
## Autocorrelations of series 'DAXlogreturn', by lag
##
## 0.00000 0.00385 0.00769 0.01154 0.01538 0.01923 0.02308 0.02692 0.03077 0.03462
## 1.000 0.000 -0.027 -0.010 0.000 -0.032 0.002 -0.030 -0.009 0.023
## 0.03846 0.04231 0.04615 0.05000 0.05385 0.05769 0.06154 0.06538 0.06923 0.07308
## 0.009 0.056 0.022 0.018 0.012 -0.026 0.010 -0.046 -0.009 0.008
## 0.07692 0.08077 0.08462 0.08846 0.09231 0.09615 0.10000 0.10385 0.10769 0.11154
## 0.028 0.006 -0.010 -0.025 -0.016 0.002 -0.022 -0.016 -0.024 0.029
## 0.11538 0.11923 0.12308
## 0.028 -0.019 0.015

# (d) acf of log-returns ^2 (prove of ARCH)
ansd <- acf(DAXlogreturn^2, main = "ACF of ^2 DAX log-returns")
ansd
```

```
##
## Autocorrelations of series 'DAXlogreturn^2', by lag
##
## 0.00000 0.00385 0.00769 0.01154 0.01538 0.01923 0.02308 0.02692 0.03077 0.03462
## 1.000 0.079 0.171 0.074 0.078 0.053 0.047 0.064 0.035 0.017
## 0.03846 0.04231 0.04615 0.05000 0.05385 0.05769 0.06154 0.06538 0.06923 0.07308
## 0.043 0.028 0.022 0.042 0.060 0.053 0.027 0.028 0.014 0.043
## 0.07692 0.08077 0.08462 0.08846 0.09231 0.09615 0.10000 0.10385 0.10769 0.11154
## 0.035 0.038 0.014 0.022 0.059 0.033 0.018 0.047 0.023 0.019
## 0.11538 0.11923 0.12308
## 0.054 0.050 0.024

# (e) fit AR(2) to ^2 returns to obtain approx ARCH(2) parameters
out <- arima(DAXlogreturn^2, order = c(2, 0, 0), include.mean=TRUE)
print(out)

##
## Call:
## arima(x = DAXlogreturn^2, order = c(2, 0, 0), include.mean = TRUE)
##
## Coefficients:
##          ar1      ar2  intercept
##         0.0658  0.1661         1e-04
## s.e.  0.0229  0.0229         0e+00
##
## sigma^2 estimated as 8.863e-08:  log likelihood = 12456.1,  aic = -24904.2

# extract coef
phi <- as.numeric(out$coef[1:2]) # phi1, phi2 from AR(2) on ^2 series
xbar <- as.numeric(out$coef["intercept"]) # intercept (mean) from arima on ^2 series
s2 <- out$sigma2
cat("phi1 =", phi[1], " phi2 =", phi[2], " intercept =", xbar, " sigma2 =", s2, "\n")

## phi1 = 0.06580107  phi2 = 0.1660825  intercept = 0.0001064871  sigma2 = 8.8632e-08

# write out fitted AR(2) model for ^2 returns:
# y_t^2 = intercept + phi1 * y_{t-1}^2 + phi2 * y_{t-2}^2 + error_t
cat("\nfitted AR(2) on ^2 returns:\n")
```

```
##
## fitted AR(2) on ^2 returns:

cat(sprintf("y_t^2 = %g + (%g) y_{t-1}^2 + (%g) y_{t-2}^2 + eps_t\n",
            xbar, phi[1], phi[2]))

## y_t^2 = 0.000106487 + (0.0658011) y_{t-1}^2 + (0.166083) y_{t-2}^2 + eps_t

# (f) approximate ARCH(2) model using phi estimates:
# conditional variance: a0 + a1*y_{t-1}^2 + a2*y_{t-2}^2
# where a0 = intercept, a1 = phi1, a2 = phi2 (approx)

cat("\nApproximate ARCH(2):\n")

##
## Approximate ARCH(2):

cat(sprintf("sigma_t^2 = %g + %g * y_{t-1}^2 + %g * y_{t-2}^2\n", xbar, phi[1], phi[2]))

## sigma_t^2 = 0.000106487 + 0.0658011 * y_{t-1}^2 + 0.166083 * y_{t-2}^2
```

Q1b

Mean log-return (drift): 0.0006520417

Q1c ACF of raw returns:

No meaningful autocorrelation. All values are near 0. Conclusion: No linear dependence.

Q1d, ACF of squared returns:

Clear positive autocorrelation at lag 1 (~0.079) and lag 2 (~0.171), and smaller positive values at later lags.

Conclusion: Strong evidence of volatility clustering.

Q1e AR(2) fit:

```
y_t^2 = 0.000106487
+ 0.0658011 * y_{t-1}^2
+ 0.166083 * y_{t-2}^2
+ eps_t
```

Q1f Approximate ARCH model:

```
 $\sigma_t^2 = 0.000106487$ 
+ 0.166083 *  $y_{t-2}^2$ 
+ 0.0658011 *  $y_{t-1}^2$ 
```

Q1g:

```
# (g) simulate a time series with same length and include drift
n <- length(DAX)
y <- numeric(n)
# init first two values with actual first two log-returns
y[1:2] <- DAXlogreturn[1:2]
Z <- rnorm(n)

for (i in 3:n) {
  condvar <- xbar + phi[1]*y[i-1]^2 + phi[2]*y[i-2]^2
  condvar <- ifelse(condvar > 0, condvar, 1e-8) # guard against tiny/negative
  s <- sqrt(condvar) * Z[i]
  y[i] <- s
}

# add drift & re-accumulate to prices
y_with_drift <- c(log(DAX[1]), y + drift)
DAXsim <- exp(cumsum(y_with_drift))

# plot real vs simulated
par(mfrow = c(1,2))
ts.plot(DAX, main = "Original DAX", ylab = "Index")
ts.plot(DAXsim, main = "Simulated DAX (ARCH(2) approx)", ylab = "Index")
```

```
par(mfrow = c(1,1))

# quick diagnostics: compare empirical acf of ^2 returns real vs simulated
par(mfrow = c(1,2))
acf(DAXlogreturn^2, main = "ACF ^2 (original)")
acf((diff(log(DAXsim)))^2, main = "ACF ^2 (simulated)")
```

```
par(mfrow = c(1,1))

# prin summary outputs
cat("\ndrift (mean log-return):", drift, "\n")
```

```
##
## drift (mean log-return): 0.0006520417
```

```
cat("length of series N:", n)
```

```
## length of series N: 1860
```

output:

```
drift (mean log-return): 0.0006520417
length of series N: 1860
```

see data405assignment4q1ga.png and data405assignment4q1gb.png for plots

Q2:

```
CAC <- EuStockMarkets[,3]

# Log returns
CAClogreturn <- diff(log(CAC))

# Fit ARCH(2) by fitting AR(2) to 2 returns
fit <- arima(CAClogreturn^2, order=c(2,0,0), include.mean=TRUE)

phi1 <- fit$coef[1]
phi2 <- fit$coef[2]
a0 <- fit$coef[3]
N <- length(CAClogreturn)

sim <- numeric(N)
sim[1:2] <- CAClogreturn[1:2]
Z <- rnorm(N)

for(i in 3:N){
  sigma2 <- a0 + phi1 * sim[i-1]^2 + phi2 * sim[i-2]^2
  sim[i] <- sqrt(sigma2) * Z[i]
}

# Load CAC data (3rd column)
CAC <- EuStockMarkets[,3]

# Log returns
CAClogreturn <- diff(log(CAC))

# Fit ARCH(2) by fitting AR(2) to squared returns
fit <- arima(CAClogreturn^2, order=c(2,0,0), include.mean=TRUE)

phi1 <- fit$coef[1]
phi2 <- fit$coef[2]
a0 <- fit$coef[3]
N <- length(CAClogreturn)

# Simulate ARCH(2)
sim <- numeric(N)
sim[1:2] <- CAClogreturn[1:2]
Z <- rnorm(N)
```

```

for(i in 3:N){
  sigma2 <- a0 + phi1 * sim[i-1]^2 + phi2 * sim[i-2]^2
  sim[i] <- sqrt(sigma2) * Z[i]
}

sim_prices <- exp(cumsum(c(log(CAC[1]), sim)))

# plot time
par(mfrow=c(1,2))
ts.plot(CAC, main="Original CAC")
ts.plot(sim_prices, main="Simulated ARCH(2) CAC")
sim_prices <- exp(cumsum(c(log(CAC[1]), sim)))

# Plots
par(mfrow=c(1,2))
ts.plot(CAC, main="Original CAC")
ts.plot(sim_prices, main="Simulated ARCH(2) CAC")

```

Q3:

```
# we simulate 100000 weeks and estimate annual probability > 250.
set.seed(2025)

p <- c(0.3,0.1,0.3,0.2,0.1) # p0..p4
y <- 0:4
states <- 0:10

# a transition matrix P
P <- matrix(0, nrow=11, ncol=11)
for (i in states) {
  after <- max(i - 2, 0)
  for (k in seq_along(y)) {
    j <- after + y[k]
    if (j > 10) j <- 10
    P[i+1, j+1] <- P[i+1, j+1] + p[k]
  }
}
rownames(P) <- colnames(P) <- as.character(states)

# 3b check regularity: compute P^50 and see if positive
P50 <- P ^50 # requires expm package; if not installed use eigen method below

# 3c stationary distribution pi (left eigenvector)
eig <- eigen(t(P))
idx <- which.min(abs(eig$values - 1))
pi <- Re(eig$vectors[,idx])
pi <- pi / sum(pi)

# 3d long-run weekly cost
costs <- 2 + 5 * pmax(0, 2 - states) # weekly cost per state
long_run_weekly_cost <- sum(pi * costs)

# 3ef simulate 100000 weeks, compute annual (52-week) total costs,
# estimate P(annual cost > 250)
nweeks <- 100000
X <- integer(nweeks)
X[1] <- 5
for (t in 2:nweeks) {
  after <- max(X[t-1] - 2, 0)
  ydraw <- sample(0:4, 1, prob = p)
```

```

X[t] <- min(after + ydraw, 10)
}

weeks_per_year <- 52
nyears <- floor(nweeks / weeks_per_year)
annual_costs <- numeric(nyears)
for (k in 1:nyears) {
  block <- X[((k-1)*weeks_per_year + 1):(k*weeks_per_year)]
  weekly <- 2 + 5 * pmax(0, 2 - block)
  annual_costs[k] <- sum(weekly)
}
prob_exceed <- mean(annual_costs > 250)

# print result
print("Transition matrix P:")

## [1] "Transition matrix P:"

print(round(P, 3))

##      0    1    2    3    4    5    6    7    8    9   10
## 0  0.3 0.1 0.3 0.2 0.1 0.0 0.0 0.0 0.0 0.0 0.0
## 1  0.3 0.1 0.3 0.2 0.1 0.0 0.0 0.0 0.0 0.0 0.0
## 2  0.3 0.1 0.3 0.2 0.1 0.0 0.0 0.0 0.0 0.0 0.0
## 3  0.0 0.3 0.1 0.3 0.2 0.1 0.0 0.0 0.0 0.0 0.0
## 4  0.0 0.0 0.3 0.1 0.3 0.2 0.1 0.0 0.0 0.0 0.0
## 5  0.0 0.0 0.0 0.3 0.1 0.3 0.2 0.1 0.0 0.0 0.0
## 6  0.0 0.0 0.0 0.0 0.3 0.1 0.3 0.2 0.1 0.0 0.0
## 7  0.0 0.0 0.0 0.0 0.0 0.3 0.1 0.3 0.2 0.1 0.0
## 8  0.0 0.0 0.0 0.0 0.0 0.0 0.3 0.1 0.3 0.2 0.1
## 9  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.3 0.1 0.3 0.3
## 10 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.3 0.1 0.6

cat("\nStationary distribution pi (length 11):\n")

##
## Stationary distribution pi (length 11):

```

```
print(round(pi, 9))
```

```
## [1] 0.11218209 0.08838623 0.17337199 0.16997399 0.14730832 0.09820967  
## [7] 0.07302008 0.04995233 0.03915354 0.02208764 0.02635412
```

```
cat("\nLong-run weekly cost:", round(long_run_weekly_cost, 6), "\n")
```

```
##  
## Long-run weekly cost: 3.563752
```

```
cat("Estimated P(annual cost > 250):", round(prob_exceed, 6), "\n")
```

```
## Estimated P(annual cost > 250): 0.065003
```

```
cat("Sample mean annual cost:", round(mean(annual_costs),4),  
    " sd:", round(sd(annual_costs),4), "\n")
```

```
## Sample mean annual cost: 184.9594 sd: 41.8579
```

b Regularity: The chain is irreducible and aperiodic (some power of P has all positive entries), so the transition matrix is regular.

d Long-run average weekly cost:

$E[\text{cost}] = 3.563752058457253$ units per week (where weekly cost = $2 + 5 \cdot \max(0, 2 - \text{state})$).

e-f Simulation (100000 weeks):

- Sample mean annual cost ≈ 183.9298 , sd ≈ 42.07 .
- Estimated probability that annual cost > 250 (from simulation) $\approx \mathbf{0.0598024}$ ($\approx 5.98\%$).

Q4:

```
FTSE <- EuStockMarkets[,4]
n <- length(FTSE)
lFTSE <- log(FTSE)
logreturns <- diff(lFTSE)
logreturnsQ <- c(min(logreturns)*1.01,
quantile(logreturns, (1:9)/10), max(logreturns)*1.01)
logreturnsCut <- cut(logreturns, logreturnsQ)
levels(logreturnsCut) <- 1:10

# 4b: ts plot

plot(FTSE, type="l", main="FTSE index (full series)", ylab="Index")
abline(v=1000, col="red") # mark 1000th
abline(v=n, col="blue") # mark -1 st

# 4c

P <- matrix(c(1:4, 4:1, 1,4,4,1,4,1,4,1), nrow=4, byrow=TRUE)/10
PE <- matrix(c(1:10, 10:1, rep(5.5,10), c(rep(1,5), rep(10,5))), nrow=4, byrow=TRUE)/55
rownames(P) <- paste0("S",1:4); colnames(P) <- paste0("S",1:4)
rownames(PE) <- paste0("S",1:4); colnames(PE) <- as.character(1:10)
P

##      S1  S2  S3  S4
## S1 0.1 0.2 0.3 0.4
## S2 0.4 0.3 0.2 0.1
## S3 0.1 0.4 0.4 0.1
## S4 0.4 0.1 0.4 0.1

PE
```

```
##           1           2           3           4           5           6           7
## S1 0.01818182 0.03636364 0.05454545 0.07272727 0.09090909 0.10909091 0.12727273
## S2 0.18181818 0.16363636 0.14545455 0.12727273 0.10909091 0.09090909 0.07272727
## S3 0.10000000 0.10000000 0.10000000 0.10000000 0.10000000 0.10000000 0.10000000
## S4 0.01818182 0.01818182 0.01818182 0.01818182 0.01818182 0.18181818 0.18181818
##           8           9          10
## S1 0.14545455 0.16363636 0.18181818
## S2 0.05454545 0.03636364 0.01818182
## S3 0.10000000 0.10000000 0.10000000
## S4 0.18181818 0.18181818 0.18181818
```

```
# 4de
```

```
# install.packages("HMM")
```

```
library("HMM")
```

```
# initialize HMM (states=1:4, symbols=1:10) and run Baum-Welch on first 1000 obs
```

```
hmm0 <- initHMM(
  States      = as.character(1:4),
  Symbols     = as.character(1:10),
  startProbs  = rep(1/4, 4),
  transProbs  = P,
  emissionProbs = PE
)
```

```
obsTrain <- as.character(as.numeric(logreturnsCut[1:1000]))
```

```
bw <- baumWelch(hmm0, observation = obsTrain, maxIterations = 100, delta = 1e-6)
```

```
hmmFit <- list(hmm = bw$hmm, logLik = bw$logProb)
```

```

# 4f

nTraining <- 1000
N <- 1000
nTesting <- n - nTraining
Index859 <- numeric(N)
for (i in 1:N) {
  index <- as.numeric(simHMM(hmmFit$hmm, nTesting)$observation)
  Index859[i] <- exp(cumsum(c(log(FTSE[nTraining]), runif(nTesting,
min=logreturnsQ[index], max=logreturnsQ[index+1])))))[nTesting]
}

# hist time
hist(Index859, breaks=50, freq=FALSE, main="Simulated FTSE at time 1859 (from t=1000)", xlab="F
rug(FTSE[n]) # actual final value
abline(v = FTSE[n], col="red", lwd=2)

# option price (strike = 5000)
option_price_1000 <- mean((Index859 - 5000) * (Index859 > 5000))

option_price_1000

## [1] 2521.656

```

Interpretation: These values seem realistic considering the distribution is normally distributed with an expected kurtosis (as a value cannot be negative AFAIK). Additionally the expected variance for the higher scenarios looks accurate

```

# g
nTraining2 <- 1829
nTesting2 <- 1859 - nTraining2    # = 30

N <- 1000
Index1859_from1829 <- numeric(N)
for (i in 1:N) {
  sim_res <- simHMM(hmmFit$hmm, nTesting2)
  index <- as.numeric(sim_res$observation)
  uvals <- runif(nTesting2, min = logreturnsQ[index], max = logreturnsQ[index + 1])
  Index1859_from1829[i] <- exp( cumsum(c(log(FTSE[nTraining2]), uvals))[nTesting2 + 1] )
}

hist(Index1859_from1829, breaks=40, freq=FALSE, main="Simulated FTSE at time 1859 (from t=1829)"
rug(FTSE[1859])
abline(v=FTSE[1859], col="red", lwd=2)

option_price_1829 <- mean(((Index1859_from1829 - 5000) * (Index1859_from1829 > 5000))

option_price_1000

## [1] 2521.656

```

Q5:

```
# a.  $40^2$  matrix
dice_freq <- c(`2`=1, `3`=2, `4`=3, `5`=4, `6`=5, `7`=6, `8`=5, `9`=4, `10`=3, `11`=2, `12`=1)
dice_p <- dice_freq / sum(dice_freq)

states <- 1:40
P <- matrix(0, nrow=40, ncol=40)
for (i in states) {
  if (i == 31) { ##### space 31 -> next move to 11
    P[i, 11] <- 1
  } else {
    for (s in 2:12) {
      j <- ((i - 1) + s) %% 40 + 1
      P[i, j] <- P[i, j] + dice_p[as.character(s)]
    }
  }
}

# b - vector
eig <- eigen(t(P))
idx <- which.min(abs(eig$values - 1))
pi <- Re(eig$vectors[, idx])
pi <- pi / sum(pi)
names(pi) <- as.character(states)

# barplot of probs
barplot(pi, main="Stationary distribution (Monopoly simplified)", xlab="Space", ylab="PI")

# which space is second-most frequently visited?
ord <- order(pi, decreasing = TRUE)
most_space <- ord[1]
second_space <- ord[2]
space40_prob <- pi["40"]
```

```
# c E[ revenue per turn from hotels on spaces 17,19,20 ]
revenue_per_turn <- pi["17"]*950 + pi["19"]*950 + pi["20"]*1000

# print res
cat("Most visited space:", most_space, "\n")

## Most visited space: 11

cat("2nd-most visited space:", second_space, "\n")

## 2nd-most visited space: 18

cat("Visit probability for space 40:", round(as.numeric(space40_prob), 6), "\n")

## Visit probability for space 40: 0.022137

cat("Expected revenue per turn from hotels:", round(as.numeric(revenue_per_turn), 6), "\n")

## Expected revenue per turn from hotels: 77.02148
```

```

# d&e:

dice_freq <- c(`2`=1, `3`=2, `4`=3, `5`=4, `6`=5, `7`=6, `8`=5, `9`=4, `10`=3, `11`=2, `12`=1)
dice_p <- dice_freq / sum(dice_freq)

states <- 1:40
P <- matrix(0, 40, 40)
for (i in states) {
  if (i == 31) { P[i, 11] <- 1
  } else {
    for (s in 2:12) {
      j <- ((i - 1) + s) %% 40 + 1
      P[i, j] <- P[i, j] + dice_p[as.character(s)]
    }
  }
}

eig <- eigen(t(P))
idx <- which.min(abs(eig$values - 1))
pi <- Re(eig$vectors[, idx])
pi <- pi / sum(pi) # stationary probs
names(pi) <- as.character(states)

# define revenues/costs per state
# my hotels: spaces 17, 19, 20 => 950, 950, 1000
rev <- numeric(40)
rev[c(17,19,20)] <- c(950, 950, 1000)

# opponent hotels: spaces 38, 40 => 1500, 2000
cost <- numeric(40)
cost[c(38,40)] <- c(1500, 2000)

# E[~]
E_rev <- sum(pi * rev)
E_cost <- sum(pi * cost)

# var[~]
E_rev2 <- sum(pi * (rev^2))
Var_rev <- E_rev2 - E_rev^2

E_cost2 <- sum(pi * (cost^2))
Var_cost <- E_cost2 - E_cost^2

```

```

# profit per state and its moments
profit <- rev - cost
E_profit <- sum(pi * profit)
E_profit2 <- sum(pi * (profit^2))
Var_profit <- E_profit2 - E_profit^2
SD_profit <- sqrt(Var_profit)

# prin
cat("E[Revenue] per turn:", round(E_rev, 6), "\n")

## E[Revenue] per turn: 77.02148

cat("E[Cost] per turn   :", round(E_cost, 6), "\n")

## E[Cost] per turn   : 76.4601

cat("E[Profit] per turn :", round(E_profit, 6), "\n\n")

## E[Profit] per turn : 0.56138

cat("Var(Revenue) :", round(Var_rev, 6), "\n")

## Var(Revenue) : 68565.82

cat("Var(Cost)      :", round(Var_cost, 6), "\n")

## Var(Cost)      : 130981.4

cat("Var(Profit)   :", round(Var_profit, 6), "\n")

## Var(Profit)   : 211325.4

```

```
cat("SD(Profit)   :", round(SD_profit, 6), "\n")
```

```
## SD(Profit)   : 459.7014
```

Interpretation: With the hotels, expected profit turns negative (~
15.66 loss per turn). The standard deviation of profit is large (656), so outcomes are highly
volatile - occasional large gains or losses - high risk high reward. Perfect for a late game 'nothing to
lose' scenario but nowhere else

```

# f: repeat (d)-(e) with additional hotels
dice_freq <- c(`2`=1, `3`=2, `4`=3, `5`=4, `6`=5, `7`=6, `8`=5, `9`=4, `10`=3, `11`=2, `12`=1)
dice_p <- dice_freq / sum(dice_freq)

P <- matrix(0, 40, 40)
for (i in 1:40) {
  if (i == 31) {
    P[i, 11] <- 1
  } else {
    for (s in 2:12) {
      j <- ((i - 1) + s) %% 40 + 1
      P[i, j] <- P[i, j] + dice_p[as.character(s)]
    }
  }
}

eig <- eigen(t(P))
idx <- which.min(Mod(eig$values - 1))
pi <- Re(eig$vectors[, idx])
pi <- pi / sum(pi)
names(pi) <- as.character(1:40)

# original hotels as ref
rev_orig <- numeric(40); rev_orig[c(17,19,20)] <- c(950,950,1000)
cost_orig <- numeric(40); cost_orig[c(38,40)] <- c(1500,2000)

# (f) add hotels: additional hotels on 22,24,25 with revenues ~
rev_new <- rev_orig
rev_new[c(22,24,25)] <- c(1050,1050,1100)

cost_new <- cost_orig
cost_new[c(32,33,35)] <- c(1275,1275,1400)

# E
E_rev_new <- sum(pi * rev_new)
E_cost_new <- sum(pi * cost_new)
E_profit_new <- sum(pi * (rev_new - cost_new))

# V
E_rev2_new <- sum(pi * (rev_new^2))
Var_rev_new <- E_rev2_new - E_rev_new^2

E_cost2_new <- sum(pi * (cost_new^2))

```

```

Var_cost_new <- E_cost2_new - E_cost_new^2

profit_new <- rev_new - cost_new
E_profit2_new <- sum(pi * (profit_new^2))
Var_profit_new <- E_profit2_new - E_profit_new^2
SD_profit_new <- sqrt(Var_profit_new)

# display time
cat("E[Revenue] per turn:", round(E_rev_new,6), "\n")

## E[Revenue] per turn: 160.2411

cat("E[Cost] per turn   :", round(E_cost_new,6), "\n")

## E[Cost] per turn   : 175.9

cat("E[Profit] per turn :", round(E_profit_new,6), "\n\n")

## E[Profit] per turn : -15.65887

cat("Var(Revenue) :", round(Var_rev_new,6), "\n")

## Var(Revenue) : 137636.9

cat("Var(Cost)       :", round(Var_cost_new,6), "\n")

## Var(Cost)       : 236876.8

cat("Var(Profit)     :", round(Var_profit_new,6), "\n")

## Var(Profit)     : 430886.6

```

```
cat("SD(Profit)   :", round(SD_profit_new,6), "\n")
```

```
## SD(Profit)   : 656.4195
```

Interpretation: With the additions, it seems this high risk high reward senario has yeilded even more loss, with a negative expected profit. This is not to mention our volatile nature

```
# g Simulation of concurrent Markov chain
```

```
set.seed(6219)
```

```
# helper meth: roll two dice & move
```

```
roll_move <- function(pos) {  
  move <- sample(1:6, 1) + sample(1:6, 1)  
  new <- (pos + move) %% 40  
  if (new == 0) new <- 40  
  # Jail rule: space 31 sends you to 11  
  if (new == 31) new <- 11  
  return(new)  
}
```

```
# case c
```

```
you_spaces_cd <- c(17, 19, 20)  
you_values_cd <- c(950, 950, 1000)
```

```
# case d
```

```
opp_spaces_cd <- c(38, 40)  
opp_values_cd <- c(1500, 2000)
```

```
# convert to named vectors for fast lookup
```

```
rev_vec_cd <- setNames(rep(0,40), 1:40)  
rev_vec_cd[you_spaces_cd] <- you_values_cd
```

```
cost_vec_cd <- setNames(rep(0,40), 1:40)  
cost_vec_cd[opp_spaces_cd] <- opp_values_cd
```

```
# sim 1 round -----
```

```
simulate_match <- function(rev_vec, cost_vec, start_cash = 5000) {  
  you_pos <- 1  
  opp_pos <- 1  
  you_cash <- start_cash  
  opp_cash <- start_cash  
  
  while (you_cash > 0 && opp_cash > 0) {
```

```

# player
you_pos <- roll_move(you_pos)
you_cash <- you_cash + rev_vec[you_pos] - cost_vec[you_pos]

if (you_cash <= 0) return(FALSE) # you lose

# op move
opp_pos <- roll_move(opp_pos)
opp_cash <- opp_cash + cost_vec[opp_pos] - rev_vec[opp_pos]

if (opp_cash <= 0) return(TRUE) # you win
}
}

# lets simulate n=1000 ties

N <- 1000

# cd
wins_cd <- mean(replicate(N, simulate_match(rev_vec_cd, cost_vec_cd)))

#wins_cd

# lets do it for F

you_spaces_f <- c(22, 24, 25)
you_values_f <- c(1050, 1050, 1100)

opp_spaces_f <- c(32, 33, 35)
opp_values_f <- c(1275, 1275, 1400)

cost_vec_cd <- setNames(rep(0,40), 1:40)
cost_vec_cd[opp_spaces_cd] <- opp_values_cd

cost_vec_f <- cost_vec_cd
cost_vec_f[opp_spaces_f] <- cost_vec_cd[opp_spaces_f] + opp_values_f

rev_vec_f <- rev_vec_cd
rev_vec_f[you_spaces_f] <- rev_vec_cd[you_spaces_f] + you_values_f

```

```
wins_f <- mean(replicate(N, simulate_match(rev_vec_f, cost_vec_f)))
```

```
cat("wins cd:", wins_cd*100, "% \n")
```

```
## wins cd: 52.1 %
```

```
cat("wins f:", wins_f*100, "% \n")
```

```
## wins f: 31.6 %
```

Interpretation: seems like a slightly over 50/50 chance id win under cd, and a slightly under 1/3 chance id win under f - which seems realistic considering the numbers on the assignment