Monte-Carlo Simulations

Asset Pricing I

Yvan Lengwiler, University of Basel

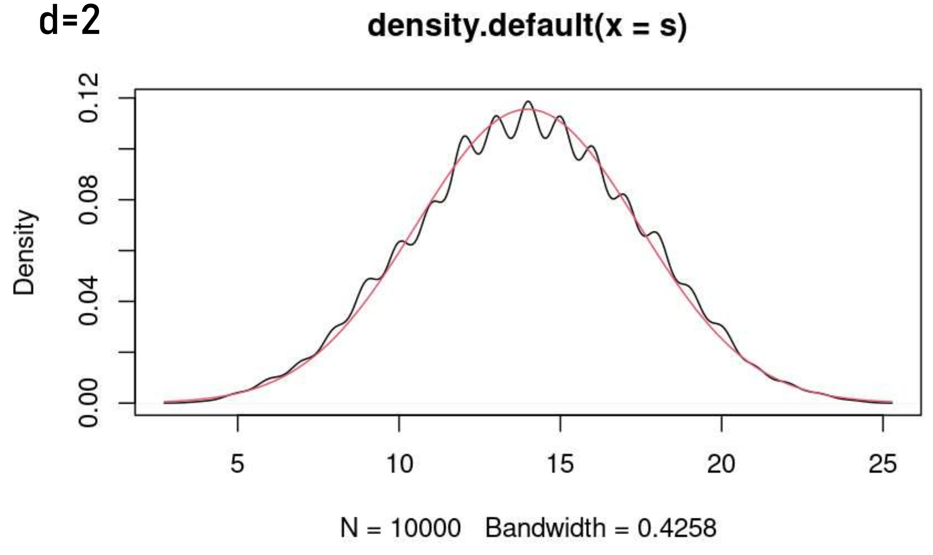
1 Task. Start from the file 'dices-unfinished.r'. This script already contains the code for the simulation of one dice. Your task is to extend this to multiple dices. Note that if you trow d dices n times, this is equivalent to throwing one dice d times n times. The details of what you need to do are in the script. The script also already contains the code to produce a kernel estimation.

```
# **** parameters
s ← 6      # maximum number of eyes on the dices
n ← 10000  # number of simulations
# **** compute Monte-Carlo simulation assuming uniform distribution
s ← floor(runif(n, min=1, max=s+1))
# **** compute probabilities
prob ← c()
for ( t in seq(1,s) ) {
    count ← length(throws[throws==t])
    prob ← c(prob, count/n)
}
barplot(prob)
# **** extend this to multiple dices
# 1) Simulate d dices.
# 2) For each throw, compute the sum of the d dices (store
#    these sums in a vector called ‘m’)
# 3) Plot the density, as we did last week (with 'density' and'plot’)
# 4) Run the following for different d: 2,4,10,100

d ← 4       # number of dices

WHAT DO YOU PUT HERE?

# plot density of the sum of d dices
density_estimate ← density(s)
plot(density_estimate, type='l’)
# plot density of normal distrbution with same moments
mu ← mean(s)
sigma ← sd(s)
curve(dnorm(x, mean = mu, sd = sigma), add = TRUE, col = 2)
```

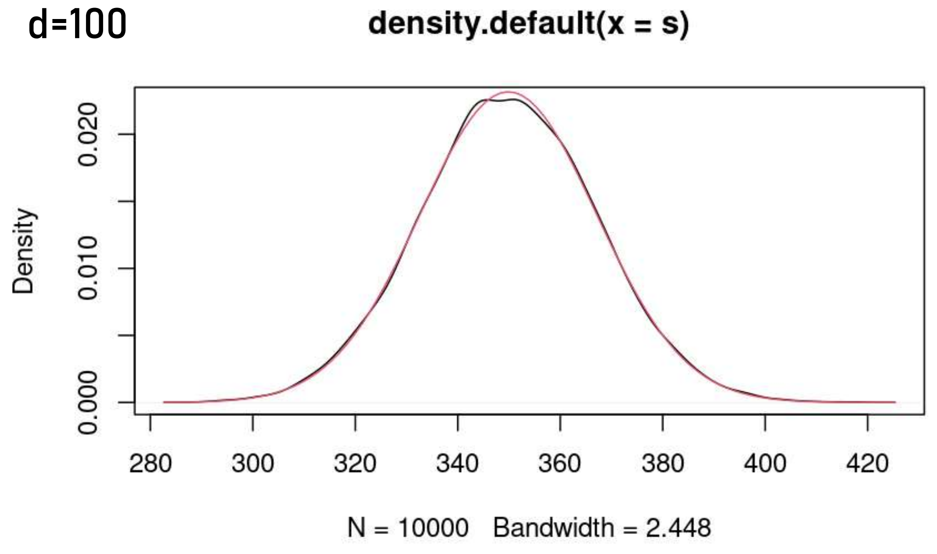
1 Task. Start from the file 'dices-unfinished.r'. This script already contains the code for the simulation of one dice. Your task is to extend this to multiple dices. Note that if you trow d dices n times, this is equivalent to throwing one dice d times n times. The details of what you need to do are in the script. The script also already contains the code to produce a kernel estimation.

```
# **** parameters
s ← 6      # maximum number of eyes on the dices
n ← 10000  # number of simulations
# **** compute Monte-Carlo simulation assuming uniform distrib
s ← floor(runif(n, min=1, max=s+1))
# **** compute probabilities
prob ← c()for ( t in seq(1,s) ) {
    count ← length(throws[throws==t])
    prob ← c(prob, count/n)
}
barplot(prob)
# **** extend this to multiple dices
# 1) Simulate d dices.
# 2) For each throw, compute the sum of the d dices (store these sums in
#    a vector called ‘m’)
# 3) Plot the density, as we did last week (with 'density' and 'plot')
# 4) Run the following for different d: 2,4,10,100

d ← 4       # number of dices

# n * d throws of the dice
throws ← floor(runif(n*d, min=1, max=s+1))
# organize as a n x d matrix and take row-wise sums
s ← rowSums(matrix(throws, nrow=n))

# plot density of the sum of d dices
density_estimate ← density(m)
plot(density_estimate, type='l’)
# plot density of normal distrbution with same moments
mu ← mean(s)
sigma ← sd(s)
curve(dnorm(x, mean = mu, sd = sigma), add = TRUE, col = 2)
```

d=2



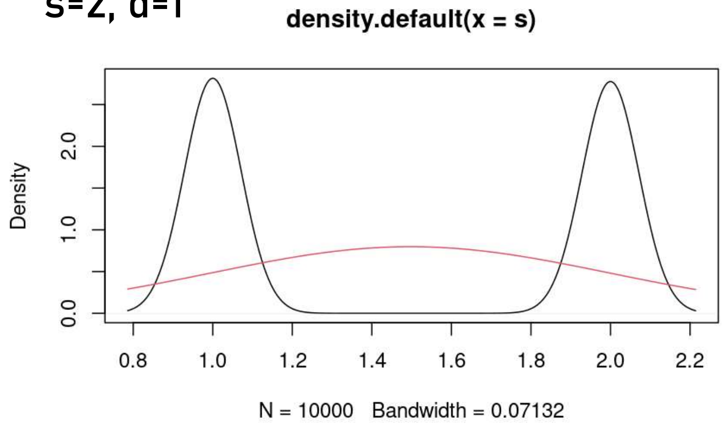
d=100



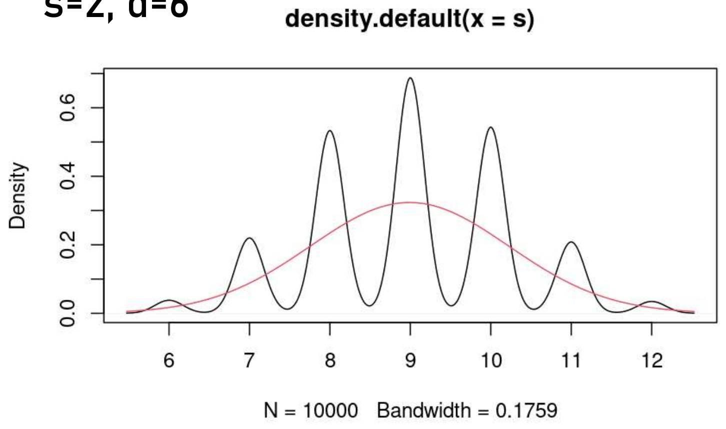
If we set s=2 (number of sides of the dice), we are simulating a coin toss instead of a dice. This reveals the mechanics of the central limit theorem even better.

Central Limit Theorem in Action

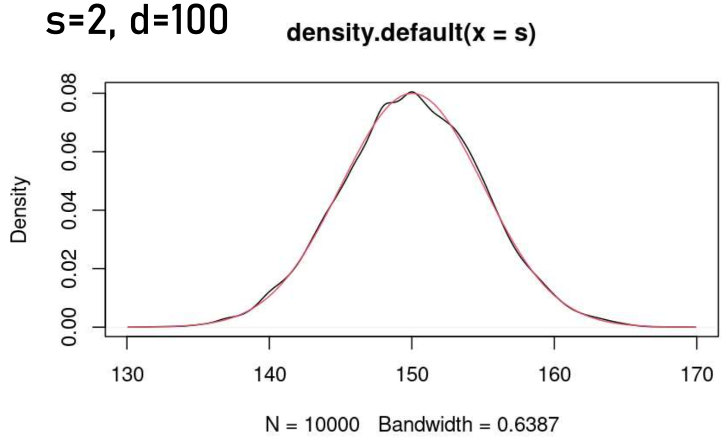
s=2, d=1



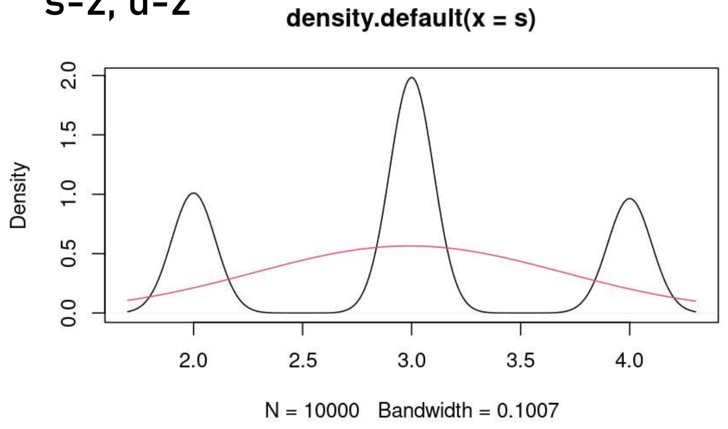
s=2, d=6



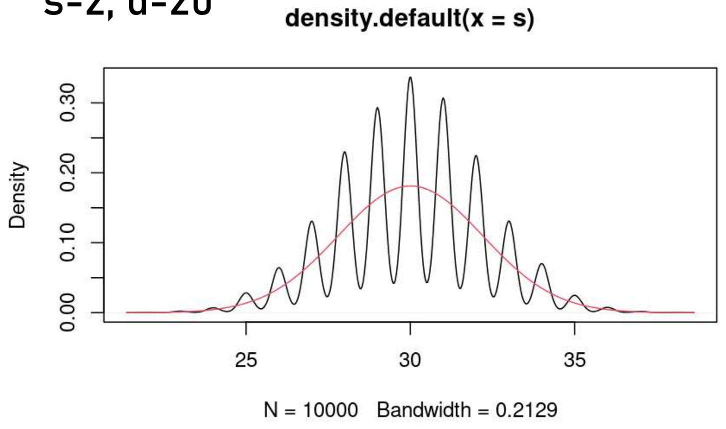
s=2, d=100



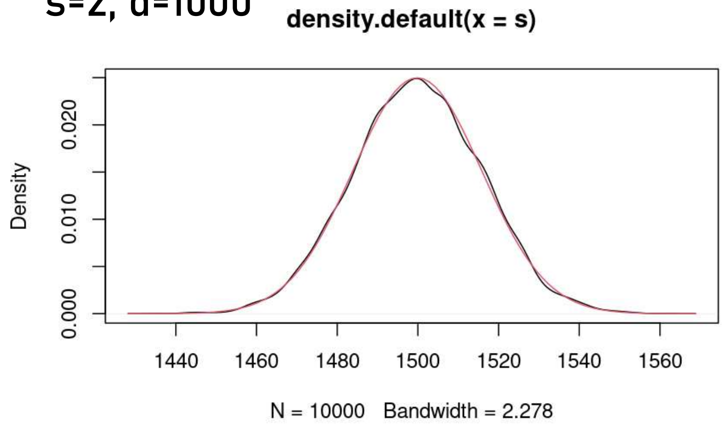
s=2, d=2



s=2, d=20



s=2, d=1000

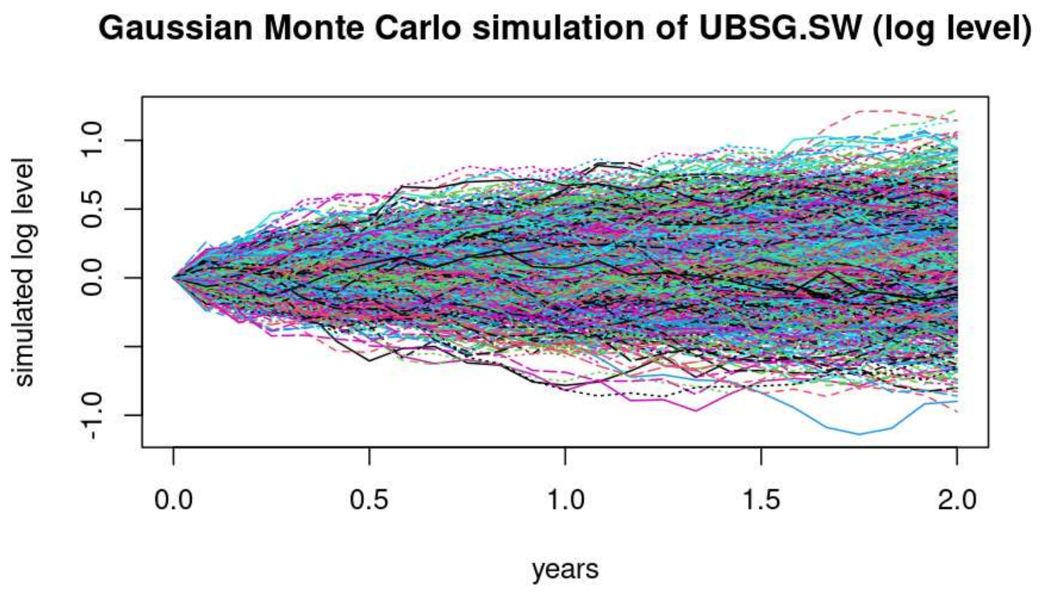


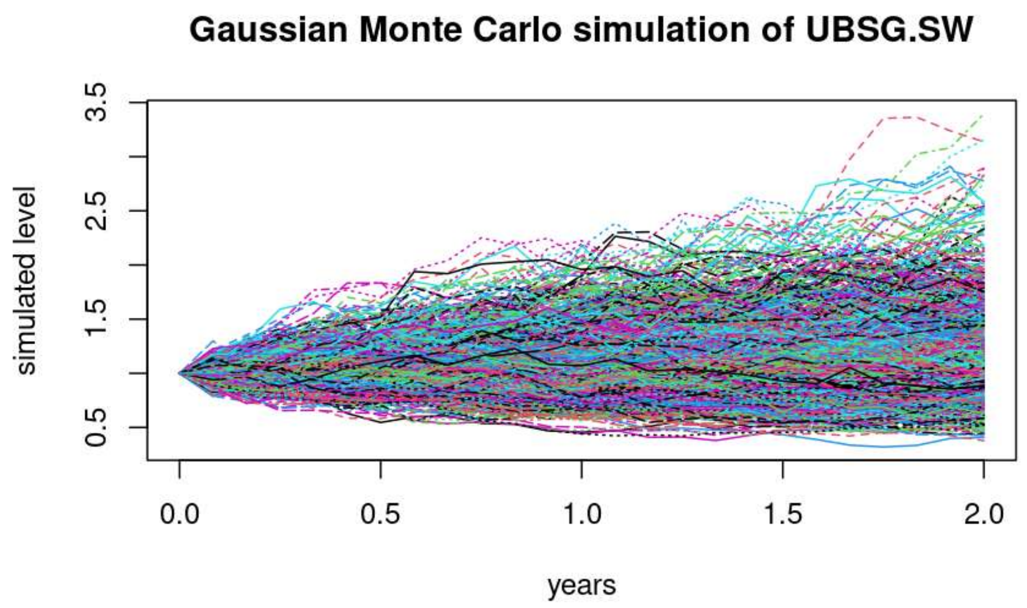
2 Task. Start from the file 'MC-unfinished.r'. Please complete the code at the indicated places so that simulations of the stochastic returns are generated. The simulated returns should be normally distributed and have the same mean and variance as the empirical data. A simulated path should cover two calendar years and you should generate 1000 paths:

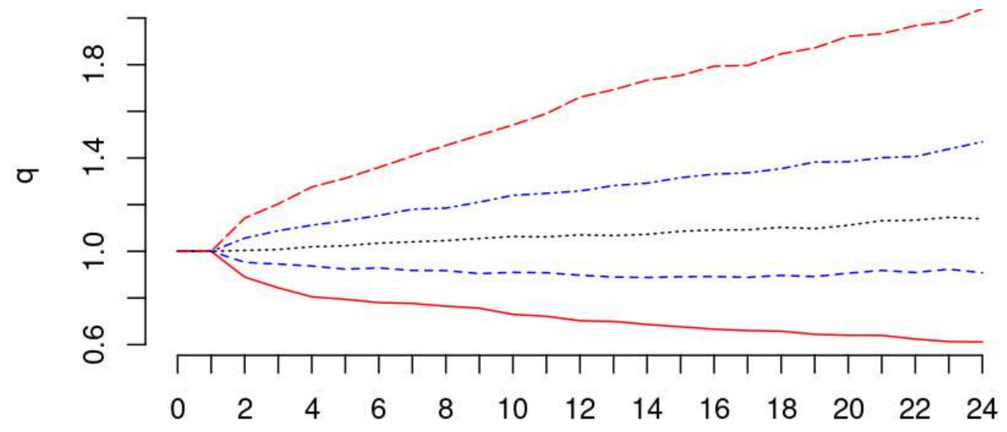
```
    sim_years <- 2 # length of simulation in years
    n <- 1000 # number of simulations
```

From the random returns, you can also compute the resulting logarithmic level simply by cumulating the returns ($x_t = x_1 + r_2 + \cdots + r_t$). The function `cumsum` is quite helpful here. Also make charts showing the simulated logarithmic and non-logarithmic paths.

```
# we download data from yahoo
#  ...
price ← as.numeric(data$price_adjusted)
dates ← as.Date(data$ref_date)

# compute the yields
yield ← diff(log(price)) * factor

# compute Monte-Carlo assuming normal distrib
# first and second moment of the data
mu ← mean(yield)
sigma ← sd(yield)

# how to continue?
```


Gaussian Monte Carlo simulation of UBSG.SW (log level)
simulated log level
-1.0
0.0
0.5
1.0
0.0
0.5
1.0
1.5
2.0
years


Gaussian Monte Carlo simulation of UBSG.SW
simulated level
0.5
1.5
2.5
3.5
0.0
0.5
1.0
1.5
2.0
years


q
0.6
1.0
1.4
1.8
0
2
4
6
8
10
12
14
16
18
20
22
24

3 Question. Remember the discussion about the central limit theorem. Elaborate about the importance of the distribution you use to underlie your simulation. How important is it that the distribution function you use in the Monte Carlo simulation fits the empirical distribution?