Exercise – Understanding the law of large numbers (LLN) using coin toss simulation

1. Aim of the exercise

Deterministic convergence, stochastic convergence, and the LLN are typically taught theoretically. This exercise presents the theory but uses simulation to demonstrate these abstract concepts to achieve concrete understanding of them.

2. Theory

A sequence of numbers $\{x_n\}$ is said to converge to a constant μ if for any $\varepsilon > 0$ there exists a $N = N_\varepsilon$ such that for all $n > N$,

$$|x_n - \mu| < \varepsilon.$$

We express this as

$$\lim_{n\to\infty} x_n = \mu.$$

Here, a formula that depends on n should generate the nth member of sequence $\{x_n\}$ for any n. For example, $x_n = \frac{1}{2^n}$. This makes the sequence $\{x_n\}$ deterministic. Therefore, in the definition above, the limit and the convergence are deterministic. Note also that the definition requires that ε is chosen before N, and that N depends on ε. In most cases N is smaller when ε higher.

Consider a sequence of 'random' numbers $\{x_n\}$. In this case we cannot be certain that $|x_n - \mu| < \varepsilon$ for 'all' n. For some n this will be true, for others it will not. Therefore we need to consider $|x_n - \mu| < \varepsilon$ with a probability, and then check if this probability becomes arbitrarily close to 1 when n is sufficiently large. Consider the following definition. A sequence of random numbers $\{x_n\}$ is said to converge in probability to a constant μ if for any number $\varepsilon > 0$

$$\lim_{n\to\infty} P\left(|x_n - \mu| < \varepsilon\right) = 1.$$

We express this as

$$\operatorname*{plim}_{n\to\infty} x_n = \mu.$$

Consider the sequence of random numbers $\{\overline{x}_n\}$ where $\overline{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$, and x_i is an i.i.d. random variable with $\mathrm{E}[x_i] = \mu$ and $\mathrm{Var}[x_i] = \sigma^2$. Here $\overline{x}_n$ represents the sample mean at n. Since we have a $\overline{x}_n$ at each n, $\{\overline{x}_n\}$ is a sequence of sample means. The weak LLN states that the sample mean of a random variable converges in probability to the population mean of that random variable when the sample size is sufficiently large. In formal terms, for any $\varepsilon > 0$,

$$\lim_{n\to\infty} P\left(|\overline{x}_n - \mu| < \varepsilon\right) = 1.$$

We express this as

$$\operatorname*{plim}_{n\to\infty} \overline{x}_n = \mu.$$

3. Convergence of the sample mean of coin tosses to the population mean

3.1. Clear the memory

Clear the memory from possible calculations from an earlier session.

```
clear;
```

3.2. Set the number of tosses

In this exercise we will simulate the coin toss experiment which is typically used as an example to demonstrate the LLN. We will toss a coin for a number of times, and analyse its mean behaviour as we increase the number of tosses. Ideally we would let the number of tosses increase indefinitely. But this is computationally intensive and not necessary to demonstrate the LLN. It is enough if we let the number of tosses increase up to a large number of `N_tosses`.

What would happen if you increase this number?

```
% 3.2. Set the number of tosses
N_tosses = 25000;
```

3.3. Create a sequence of tosses

The exercise requires us to create a sequence of coin tosses. The same sequence will need to be used throughout the exercise. Let an integer value of 1 to represent the heads, and 0 the tails in a coin toss. Creating the sequence of coin tosses then boils down to generating a random sequence of binary integers 1 and 0 from the discrete uniform distribution. We can generate this sequence using the built-in MATLAB function `random`. We generate random numbers from the binomial distribution. We need to supply the function with four input arguments. The first input argument specifies the number of trials for each individual experiment. In this case, we are using the Binomial distribution with a single trial, which simplifies to the Bernoulli distribution. Each trial can result in either 0 or 1. The second input argument specifies the probability of success (in this case, generating a 1) for each trial. A value of 0.5 means that there is an equal chance (50%) of generating a 1 or a 0. The third input argument specifies the dimension of the matrix array to contain the random draws. Since we are interested in a single sequence of `N_tosses` random draws, a column array with dimension `N_tosses` by `1` would serve to the purpose. The second input argument can then be specified as `[N_tosses,1]`.

Each element of the vector `sequence_of_tosses` refers to x_i in the theoretical exposition above.

```
% 3.3. Create a sequence of tosses
sequence_of_tosses = random('Binomial',1,0.5,[N_tosses,1]);
```

3.4. Preallocate a matrix

Preallocate a matrix to store the sequence of sample means of cumulating tosses in the sequence of tosses.

```
% 3.4. Preallocate the matrix for storing the sequence of sample means
mean_sample = NaN(N_tosses,1);
```

3.5. Create the sequence of sample means of cumulating tosses in the sequence of tosses

Suppose that we are about to toss the coin four times. What would we expect the mean

of the tosses to be in this sequence of tosses? Perhaps 0.5 because a coin has a 0.5 probability of returning heads or tails, and we would expect to obtain two heads and two tails, and then to obtain the mean as $(0+0+1+1)/4 = 0.5$. Suppose that the outcome of the four coin tosses is three heads and one tails. Then, the mean of this sequence of four tosses is 0.75. This is not quite the mean we expected.

If we keep on tossing the coin after the first four times, how would we expect the sample mean to behave? That is, how would we expect the means of the cumulating tosses to behave? We would expect them to converge to 0.5. Below we will study whether this is true.

We need to calculate the means of cumulating tosses in the sequence of tosses created above. The first line of the code presented at the end of this section creates an empty matrix that will store the means of cumulating tosses to be calculated shortly.

The for loop presented at the end of the section calculates the mean of tosses at given numbers of tosses. The first line of the for loop specifies the index of the loop. We want to calculate the means of cumulating tosses, and the tosses keep on cumulating until when the total number of tosses is `N_tosses`. Therefore, the index of the for loop starts from `1` and runs until `N_tosses`.

In the second line of the for loop, we calculate the means of cumulating tosses. In particular, at a given iteration of the for loop, we first select the toss outcomes from the first toss until the toss dictated by the iteration using array indexing on the column array `sequence_of_tosses`. That is, the row subscript `1:i` of the array selects the rows from the first row until row `i`, and the column subscript `1` of the array selects the only column of the array. Hence, `sequence_of_tosses(1:i,1)` selects the sequence of coin tosses at iteration `i`. Next, we take the mean of the selected array using the `mean` function. The mean at iteration `i` is calculated by `mean(sequence_of_tosses(1:i,1))`. This mean is then filled in row number `i` of the empty column array `mean_sample` at iteration `i` of the for loop. The last line of the for loop closes the loop.

Open the `mean_sample` array in the Workspace to inspect its content. Each row contains the mean of a sequence of tosses. For example, row number five contains the mean of the first five coin tosses. The next row contains the mean of the 'first five tosses plus an additional toss'. The vector `mean_sample` refers to $\{\overline{x}_n\}$ in the theoretical exposition above.

```
% 3.5. Create the sequence of sample means
for i = 1:N_tosses
    mean_sample(i,1) = mean(sequence_of_tosses(1:i,1));
end
```

3.6. Define the population mean

As discussed above, we expect the means of cumulating tosses to converge to 0.5. Assign this value to a scalar array which we will use to create the plot of the means of cumulating tosses in a later section.

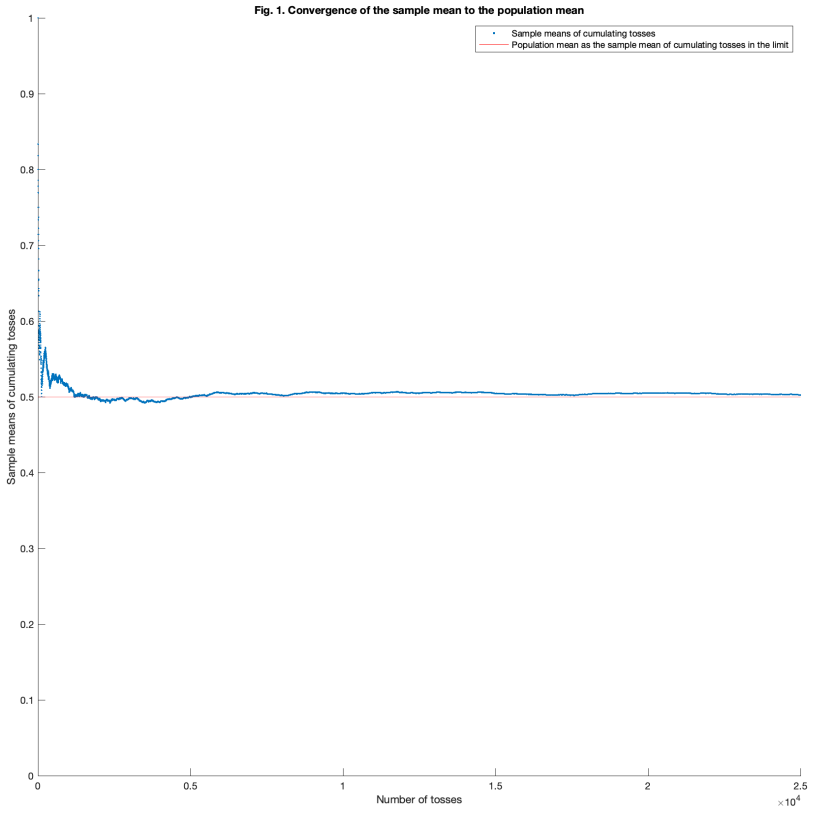
```
% 3.6. Define the population mean
mean_population = 0.5;
```

3.7. Plot the sample mean and the population mean against the number of tosses

The code presented at the end of the section plots the means of the cumulating coin tosses

against the number of tosses in each cumulation. The plot also marks the mean of the cumulating coin tosses in the limit which is equal to 0.5. `mean_population` refers to μ in the theoretical exposition above. The plot demonstrates that, as the number of coin tosses increase, the sample means of cumulating tosses are more often close to the population mean of tosses. The population mean of tosses is nothing but the expected value of a coin toss.

```
% 3.7. Plot the sample mean and the population mean against the number
% of tosses
scatter(1:N_tosses,mean_sample,0.001,'Marker','.')
ylim([0 1]) % Set the range of y axis.
hold on
line([0,N_tosses],[mean_population,mean_population],'Color','red')
hold off
title('Fig. 1. Convergence of the sample mean to the population mean')
legend('Sample means of cumulating tosses',['Population mean ' ...
    'as the sample mean of cumulating tosses in the limit'])
% Or, probability of heads if 1 represents heads.
ylabel('Sample means of cumulating tosses')
xlabel('Number of tosses')
```



4. Indicator of whether the absolute difference is small enough

4.1. Define the absolute difference and the tolerance parameter

In our simulation exercise, $\overline{x}_n$ refers to `mean_sample`, and μ refers to `mean_population`. Then, we can define $|\overline{x}_n - \mu|$ as `abs(mean_sample-mean_population)`. Name this expression as `abs_dif`. We can define the tolerance parameter ε as `e`, and set it equal to 0.01. The choice of this certain value for `e` is not all that arbitrary, and depends on how large the `n` is. The larger the `n` is, the smaller `e` can be. This is because as `n` increases, the closer the sample mean will be to the population mean, allowing us to set a more restrictive tolerance parameter.

```
% 4.1. Define the absolute difference and the tolerance parameter
abs_dif = abs(mean_sample-mean_population);
e = 0.01;
```

4.2. Create an indicator of whether the absolute difference is smaller than the tolerance parameter

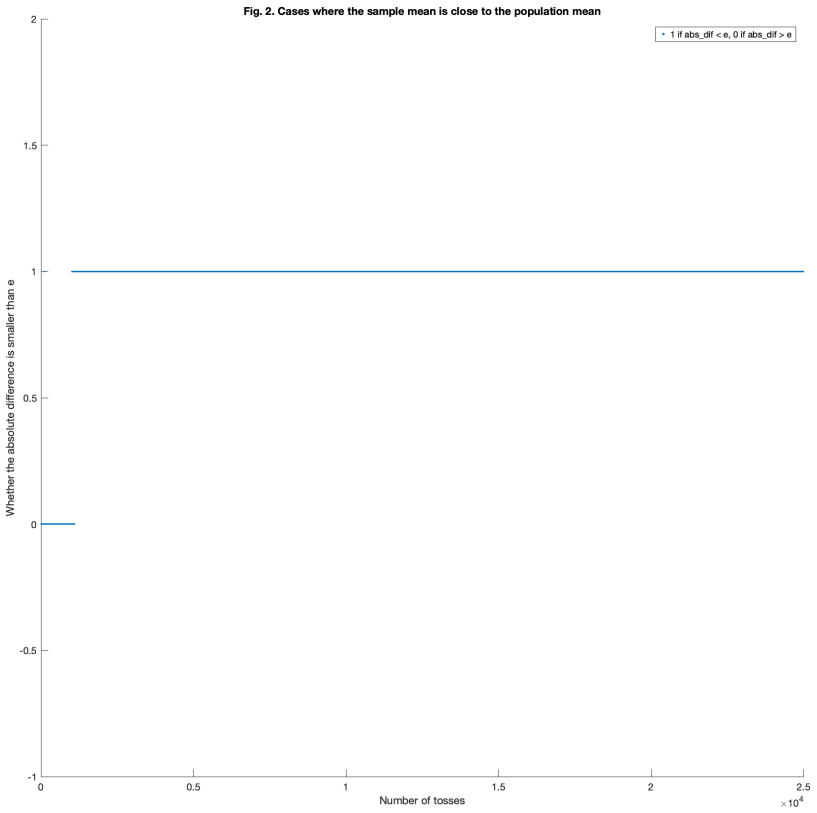
Consider the condition $|\overline{x}_n - \mu| < \varepsilon$. Given the definitions in the previous section, we can define this condition as `abs_dif < e`. In fact, in MATLAB, this syntax creates a dummy variable that takes a value of 1 if the condition is satisfied, and 0 otherwise.

```
% 4.2. Create the indicator
abs_dif_smaller_than_e_dummy = abs_dif < e;
```

4.3. Plot the indicator against the number of tosses

The code at the end of the section creates a scatter plot of the indicator against the number of tosses. Note that there are cases where the indicator takes a value of 0. These are the cases where the sample and the population means are not close enough based on the specified tolerance. This is not surprising. The members of our sequence $\{\overline{x}_n\}$ are random, and we cannot expect that $|\overline{x}_n - \mu| < \varepsilon$ for all n. Hence, we have a case where convergence cannot be deterministic. In the next section we consider $|\overline{x}_n - \mu| < \varepsilon$ with a probability at each n, and make a case where convergence is stochastic.

```
% 4.3. Plot the indicator against the number of tosses
scatter(1:N_tosses,abs_dif_smaller_than_e_dummy,0.001,'Marker','.')
ylim([-1 2])
title(['Fig. 2. Cases where the sample mean is close to ' ...
    'the population mean'])
legend('1 if abs\_dif < e, 0 if abs\_dif > e')
ylabel('Whether the absolute difference is smaller than e')
xlabel('Number of tosses')
```



5. Convergence of the sample mean to the population mean in probability

5.1. Preallocate the matrix for storing probabilities

Preallocate the matrix with dimension `N_tosses` by `1` that will store `N_tosses` probability values to be calculated in the next section.

```
% 5.1. Create an empty matrix for storing probabilities
prob_of_abs_dif_smaller_than_e = NaN(N_tosses,1);
```

5.2. Probability that the absolute difference is smaller than the tolerance parameter

The for loop presented at the end of this section calculates the mean of the dummy variable `abs_dif_smaller_than_e_dummy` at given numbers of tosses. This gives the probability that the absolute value of the difference between `mean_sample` and `mean_population` is smaller than `e` at given numbers of tosses. This refers to $P\left(|\overline{x}_n - \mu| > \varepsilon\right)$ at given numbers of tosses. In the for loop we will analyse the behaviour of the stated probability as the number of tosses increase.

The first line of the for loop specifies the index of the loop. We want to calculate the means of the dummy variable `abs_dif_smaller_than_e_dummy` as the number of tosses `N_tosses` cumulate. Therefore, the index of the for loop starts from `1` and runs until `N_tosses`.

In the second line of the for loop, we calculate the mean of the observations of the dummy variable `abs_dif_smaller_than_e_dummy` at a given sequence of coin tosses. In particular,

at a given iteration of the for loop, we first select the observations of the dummy variable corresponding to the first toss until the toss dictated by the iteration using array indexing on the column array `abs_dif_smaller_than_e_dummy`. That is, the row subscript `1:i` of the array selects the rows from the first row until row `i`, and the column subscript `1` of the array selects the only column of the array. Hence, `abs_dif_smaller_than_e_dummy(1:i,1)` selects the observations of the dummy variable corresponding to the sequence of coin tosses at iteration `i`. Next, we take the mean of the selected array using the `mean` function. The mean at iteration `i` is calculated by `mean(abs_dif_smaller_than_e_dummy(1:i,1))`. This mean is then filled in row number `i` of the empty column array `prob_of_abs_dif_smaller_than_e` at iteration `i` of the for loop.

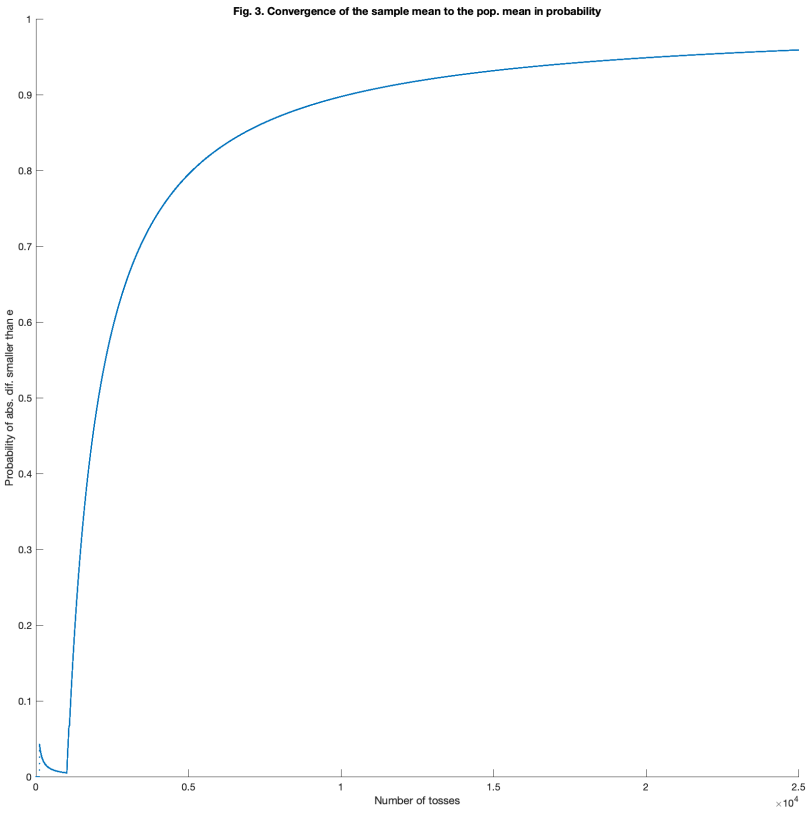
The last line of the for loop closes the loop. Open the `prob_of_abs_dif_smaller_than_e` array in the Workspace to inspect its content. A row contains the probability that the absolute difference is smaller than the tolerance parameter at a given sequences of tosses. For example, row number five contains the probability associated with the first five coin tosses. The next row contains the probability associated with the 'first five tosses plus an additional toss'.

```
% 5.2. Probability that the absolute difference is small enough
for i = 1:N_tosses
    prob_of_abs_dif_smaller_than_e(i,1) = mean( ...
        abs_dif_smaller_than_e_dummy(1:i,1));
end
```

5.3. Plot the probability that the absolute difference is small enough

The code presented at the end of the section plots the calculated probabilities for cumulating coin tosses against the number of tosses in each cumulation. The plot shows that, at a larger number of tosses, the probability that the difference between the mean of a given sequence of coin tosses and the population mean of tosses is smaller. This probability tends to go to 1 as the number of tosses in a sequence of tosses increases. In the limit this probability is 1. This demonstrates the notion of the convergence in probability introduced above. "Convergence in probability" is often referred to as "stochastic convergence".

```
% 5.3. Plot the probability that the absolute difference is larger than
% % the tolerance parameter against the number of tosses
scatter(1:N_tosses,prob_of_abs_dif_smaller_than_e,0.001,'Marker','.')
ylim([0 1])
title(['Fig. 3. Convergence of the sample mean to the pop. mean ' ...
    'in probability'])
ylabel('Probability of abs. dif. smaller than e')
xlabel('Number of tosses')
```



6. Final notes

This file is prepared and copyrighted by Tunga Kantarcı. This file and the accompanying MATLAB files are available on GitHub and can be accessed via this link.