Simulation and Performance Evaluation
Homework 2

Blascovich Alessio and Di Noia Matteo

14th April 2025

Exercise 1

As suggested we started by setting as parameter $N = T\lambda$ with $\lambda = 200$ and $T = 1000$. Then we sampled N variates from a uniform distribution in the interval $[0, T]$, that gave us the arrival times. We also sampled N variates from an exponential distribution with parameter λ, that gave us the inter-arrival times.

From Uniform to Exponential

Starting from the uniform $U \sim \text{Uni}(0, T)$. We derived the distribution of inter-arrival times by sorting the extracted values and then, by computing:

$$\forall i \in [0, N] \quad from\_uni_i = \begin{cases} u_i & \text{if } i = 0 \\ u_i - u_{i-1} & \text{if } i \geq 1 \end{cases}$$

We had to sort the extracted values because, we wanted to calculate the inter-arrival times between two adjacent events.

By plotting 1, we compared the obtain distribution against the original inter-arrival exponential distribution, and concluded that the two distributions were empirically congruent except from a small margin of error.

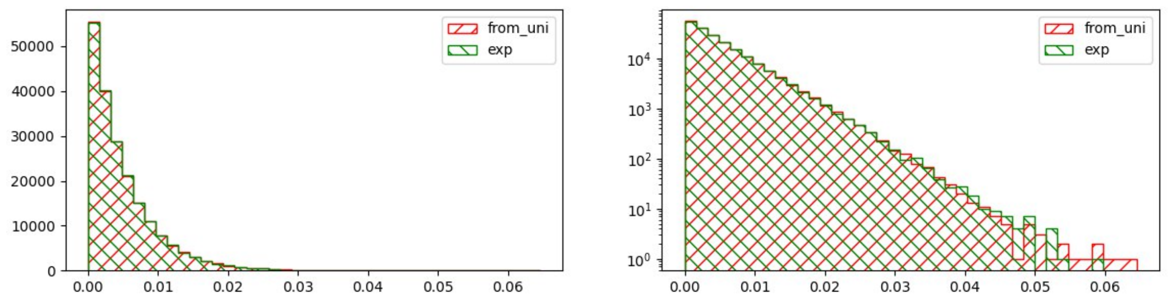

Figure 1: Exponential distributions

On the left there is the unscaled graph of the two distributions overlapping perfectly for small us. To emphasise the small errors we used, on the right graph, a logarithmic scale to represent the two exponential distributions. With this representation we saw that the errors were clustered, mostly, on larger values of u. This result was expected since, higher values of u had a lower sample rate therefore, they were less subject to the "Law of Large Numbers".

From Exponential to Uniform

Starting from the exponential $E \sim \text{Exp}(\lambda)$. We derived the distribution of arrival times by, for each sample compute the cumulative sum; up to that point.

$$\forall i \in [0, N] \quad from\_exp_i = \sum_{j=0}^{i} e_j$$

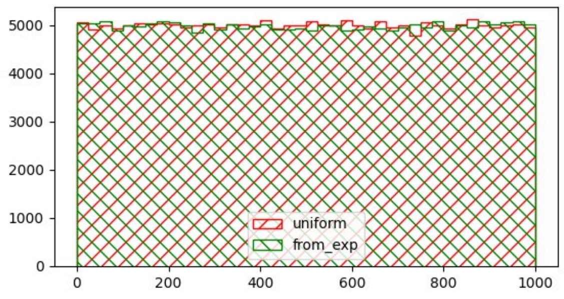

Figure 2: Uniform distributions

Note that, we had to filter out all values greater than T, this reduced the data usable for the new exponential distribution by a 0% $\sim$ 0.2%.

By plotting 2, we compared the obtain distribution against the original arrivals uniform distribution, and concluded that the two distributions are empirically congruent except from a small margin of error. In this case the error seems to be slightly bigger w.r.t. the previous plot.

Conclusions

Variating the value N as $N = 2T\lambda$. We noticed a few changes in the simulations:

- The ratio of dropped samples from the exponential to the uniform increase to reach the value of 50%, this phenomenon is visible on the leftmost graph in 3;
- The exponential distribution extracted from the uniform plunges twice as fast as the original exponential, as visible in the last two graphs in 3.

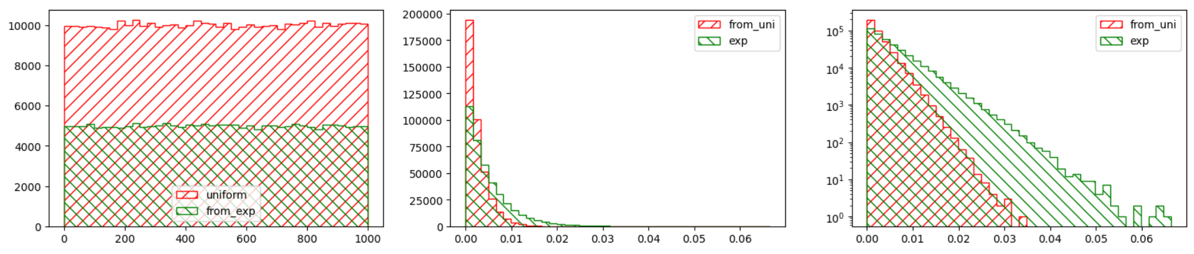

Figure 3: Distributions with $N = 2T\lambda$

Exercise 2

Sub-point 1

Because for the "rejection sample" method any scale factor is superfluous since it only cares that the given function is bounded in its domain, we ignored the parameter A. For the rest of the "rejection sampling" we used $f^n(x)$ defined as $f(x) = \frac{1}{A} f^n(x)$.

As bounding function we used $g(x) = 6.36$; this function represent the maximum of the $f^n(x)$ in its domain. Using this bounding function the average number of attempts to get a valid sample is:

$$\frac{\int_{-3}^{3} g(x)}{\int_{-3}^{3} f^n(x)} \approx 4.31283$$

```
while True:
    x = float(np.random.uniform(-3, 3, 1)[0])
    u = float(np.random.uniform(0, 6.36, 1)[0])
    if u <= weird_function(x):
        return x
```

In this snippet of code `weird_function` represents $f^n(x)$.

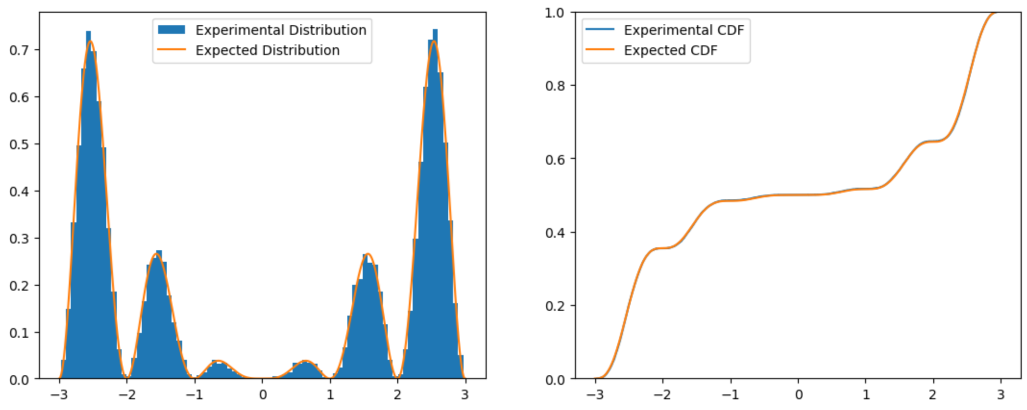

Figure 4: “Weird” PDF and CDF

Sub-point 2

To have a sample for the following sub point we drawn 20000 variates already in this stage of the simulation.

Observing 4, on the left there are the **normalised** draws obtained by the “rejection sampling” compared against the real shape of $f(x)$. With more draws the result could have been better fitting the real curve, but with this batch the fitting was already evident.

For the CDFs the non-overlapping parts are so small that they are barely visible without zooming on a particular area of the graph.

Sub-point 3

For the median and the 0.9-quantile we used the method explained in class since we had a large n and all the variables were IID. So we sorted the sub-sample and we used the standard value $\eta = 1.96$, give the fact that the we were asked for 95% confidence intervals, i.e., $\gamma = 0.95$.

```python
def es3quantile(arr_sorted: list, q: float): # for the median q = 0.5
    n = len(arr_sorted)
    quant = np.quantile(arr_sorted, q)
    low = math.floor(n * q - ETA95 * math.sqrt(n * q * (1 - q)))
    up = math.ceil(n * q + ETA95 * math.sqrt(n * q * (1 - q)) + 1)
    ci_boot = bootstrap_procedure(arr_sorted, 25, 0.95, lambda arr: np.quantile(arr, q))
    print(...)
```

For the mean we used the asymptotic method because we had a large n and non-wild distribution. So we computed arithmetic mean μ_n and standard deviation s_n then we obtained the interval $\mu_n \pm \eta \frac{s_n}{\sqrt{n}}$, with n the size of the sample.

For the “boostrap” method we used an higher-order function and the standard value $r_0 = 25$.

```python
def bootstrap_procedure(array: list, r0: int, gamma: float, st_func):
    r = math.ceil(2*r0/(1-gamma))-1
    stat_calculated = []
    for _ in range(r):
        draws = np.random.choice(array, len(array))
        stat_calculated.append(st_func(draws))
    stat_calculated.sort()
    return stat_calculated[r0], stat_calculated[r + 1 - r0]
```

The function takes in input the sample `array`, the value `r0`, the probability `gamma` and the function `st_func` describing the parameter the parameter to estimate.

Sub-point 4

Since every disjointed set is formed by 200 values we decided to use the asymptotic estimation as done before for the mean. Depending on the run the percentage of sets with an arithmetic mean within the confidentiality interval was 90% $\sim$ 100% which is around the real probability of confidence interval, i.e., 95%.

```python
def es4(samples):
    n = len(samples)
    correct = 0
    for subset in np.reshape(samples, (100, 200)):
        mean = float(np.mean(subset))
        std_dev = float(np.std(subset))
        err = ETA95 * std_dev / math.sqrt(n)
        if mean - err <= 0 <= mean + err:
            correct += 1
    print("True mean is in interval ", correct, "% of the times")
```