Monte Carlo Methods: Project

Dan Wolff, Alexandre Zenou

October 2024

1 Question 1

1.1 Density Condition

To determine whether f is a probability density function, it must satisfy the following conditions:

- f should be piecewise continuous and define as: $\forall (a,b) \in \mathbb{R}^2, \int_{[a,b]} f(x)\,dx = P(a < X < b)$
- Normalization: $\int_{\mathbb{R}} f(x)\,dx = 1$
- Non-negativity: $\forall x \in \mathbb{R} : f(x) \geq 0$

We want to show that $\forall x \in \mathbb{R} : f(x) \geq 0$ especially at the tails. Since we have $f(x) = c(f_1(x) - af_2(x))$ as f is proportional to $f_1 - af_2$, we want $c \in \mathbb{R}$ such that

$$\int_{\mathbb{R}} cf(x)\,dx = 1.$$

1.2 Analysis of f at the Tails

We consider the behavior of f as $x \to \pm\infty$. For positivity, we require:

$$c(f_1(x) - af_2(x)) \geq 0 \quad \Rightarrow \quad \frac{f_1(x)}{f_2(x)} \geq a. \quad \iff \quad \frac{-(x-\mu_1)^2\sigma_2^2 + (x-\mu_2)^2\sigma_1^2}{2\sigma_1^2\sigma_2^2} - \log(\frac{a\sigma_1}{\sigma_2}) \geq 0$$

It follows:

$$x^2(\sigma_1^2 - \sigma_2^2) + 2x(\sigma_2^2\mu_1 - \sigma_1^2\mu_2) + \sigma_1^2\mu_2^2 - 2\sigma_2^2\sigma_1^2 \log\left(\frac{a\sigma_1}{\sigma_2}\right) \geq 0.$$

So, for f to be positive at the tails, as $\lim_{x\to\pm\infty} x^2 = +\infty$, we want $\sigma_1^2 - \sigma_2^2 \geq 0 \iff \sigma_1^2 \geq \sigma_2^2$

2 Question 2

2.1 Upper Band of a

We want an upper band of a and we know that $\frac{f_1(x)}{f_2(x)} \geq a$. Let denote $a^* = \min_{x\in\mathbb{R}} \frac{f_1(x)}{f_2(x)}$. We can derive it and the minimizing problem is equivalent to $\min_{x\in\mathbb{R}} \frac{(x-\mu_2)^2}{2\sigma_2^2} - \frac{(x-\mu_1)^2}{2\sigma_1^2}$ as the exponential function is strictly increasing.

Let $h(x) = \frac{(x-\mu_2)^2}{2\sigma_2^2} - \frac{(x-\mu_1)^2}{2\sigma_1^2}$,

$$h'(x) = 0 \iff x = \frac{\mu_1\sigma_2^2 - \mu_2\sigma_1^2}{\sigma_2^2 - \sigma_1^2}$$

It follows that the best bound we can provide is $a^* = \frac{f_1(x^*)}{f_2(x^*)}$

2.2 Value of c

f is a density function so :

$$\int_{\mathbb{R}} f(x)dx = 1 \iff c(\int_{\mathbb{R}} f_1(x)dx + a\int_{\mathbb{R}} f_2(x)dx) = 1$$

f_1 and f_2 both are density function so it implies:

$$c(1+a) = 1$$

$$\iff c = \frac{1}{1+a}$$

3 Question 3

3.1 Compatibility of the fixed value

We have $\sigma_1^2 = 9$, $\sigma_2^2 = 1$. So $\sigma_1^2 \geq \sigma_2^2$.
Also, $a = 0,2$ and $a^* \approx 0.313$. So $a \leq a^*$

3.2 Impact of a and of σ_2

Additionally, when σ_2 is small, the density f has sharper, more concentrated peaks. As σ_2 increases, f spreads out, resembling a normal distribution. However, if $\sigma_2^2 > \sigma_1^2$, the function becomes negative in the tails, and f ceases to be a valid density function.

For small values of a , our density function f closely resembles a normal distribution, dominated by $f_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$. As a increases, f becomes a combination of two Gaussian distributions. At $a = a^*$, f reaches its maximum value where it remains positive; for $a > a^*$, the term $-af_2(x)$ becomes significant, causing f to take on negative values, making it no longer a valid density.

4 Question 4

4.1 Dichotomie

We began our exploration using the bisection method, focusing on the equation

$$G(x) = F(x) - u,$$

where F is the cumulative distribution function (CDF). Initially, we set our search interval from $[-500; 500]$ and computed the function values at these bounds. If both values were positive or negative, we expanded the interval by $10,000$ units in each direction until we found an appropriate range containing the root.

When the upper bound exceeded $1,000,000$, we realized that the quantile we sought was likely in the tails of the distribution, allowing us to use an approximation with the quantile function. Once we established an interval where $G(a_0)$ and $G(b_0)$ had opposite signs, we calculated the midpoint, x_0.

If $G(a_0)$ or $G(b_0)$ was exactly zero, we found our solution. Otherwise, we iterated, refining our bounds until we achieved an acceptable approximation of the quantile, x_0, and returned it alongside the error, $G(x)$.

4.2 Newton-Raphson

We've also implemented a Newton-Raphson algorithm that provided good results with fast convergence for the Monte-Carlo simulations. The method can be sensitive to the initial guess and may fail if the function's derivative is zero or unstable, potentially leading to non-convergence or divergence. (cf page 5)

5 Question 5

In this section, we set $n = 10,000$ to compute a large sample of values from a random variable with a specified distribution using the inverse function method. We defined the function `inverse_cdf`, which generates n uniformly distributed random values U between 0 and 1. For each value in U, we applied the bisection method by calling `dichotomie` to find the corresponding quantile $X[i]$ based on our distribution parameters.

We also computed the density values of our distribution function for each generated quantile, storing these results in the `value` array.

Finally, we visualized our results by plotting a histogram of the generated sample values. The histogram was constructed with 50 bins, representing the frequency of each value, and displayed alongside a line representing the density of our distribution function. The black line in the plot shows that the histogram aligns well with our theoretical density function, confirming the accuracy of our inverse function method.

6 Question 6

The **accept-reject algorithm** is a widely-used method for generating random samples from a target distribution $h(x)$ when direct sampling is challenging. This approach involves finding a simpler proposal distribution $q(x)$ from which samples can easily be generated, and a constant $C > 0$ that satisfies:

$$h(x) \leq Cq(x)$$

Steps of the Accept-Reject Algorithm

1. **Choose a Proposal Distribution $q(x)$:**
   In our case, we select $q(x) = \mathcal{N}(\mu_1, \sigma_1^2)$, which is straightforward to sample from and serves as a good approximation to $h(x)$, especially when b is small.

2. **Determine the Scaling Constant C:**
   The constant C is set to
   $$C = \max_x \frac{h(x)}{q(x)},$$
   ensuring that the target distribution $h(x)$ is always below the scaled proposal $C \cdot q(x)$.

3. **Generate a Candidate Sample Y from $q(x)$:**
   Here, we sample $Y \sim \mathcal{N}(\mu_1, \sigma_1^2)$.

4. **Generate a Uniform Random Variable $V \sim U(0, 1)$.**

5. **Acceptance Criterion:**
   If
   $$V \leq \frac{h(Y)}{C \cdot q(Y)},$$
   then accept Y as a sample from $h(x)$; otherwise, reject Y and return to Step 3.

A-R and Expected Iterations

The theoretical A-R R_a of the accept-reject algorithm is given by:

$$R_a = \frac{1}{C}.$$

Thus, the probability of accepting a sample after generating it from $q(x)$ and passing the acceptance criterion is

$$\int \frac{h(x)}{C \cdot q(x)} q(x)\, dx = \Pr\left(V \leq \frac{h(Y)}{C \cdot q(Y)}\right).$$

Consequently, the expected number of iterations $E[T]$ needed to obtain a sample from $h(x)$ is C:

$$E[T] = C,$$

where T follows a geometric distribution with parameter $\frac{1}{C}$.

Maximizing the Ratio

To determine C, we maximize the ratio:

$$\frac{h(x)}{q(x)} = \frac{f_1(x) - bf_2(x)}{f_1(x)} = \frac{1}{1-b},$$

yielding $C = \frac{1}{1-b}$. For $b = 0.2$, we find $C = 1.25$.

This theoretical A-R aligns closely with the observed A-R after implementing the accept-reject algorithm.

7 Question 7

Using code without loops in R can be more efficient for several reasons. First, R is designed to work with vectors and matrices, allowing for vectorized operations. This means calculations can be performed on entire data sets in a single instruction, which is generally faster than iterating with loops. Additionally, loops introduce state management overhead, while vectorized operations allow for more optimized memory management. They can also take advantage of parallel processing, further speeding up calculations by utilizing multiple processor cores.
Here is the output of the amoutn of time both methods take :

Table 1: Performance Comparison (Unit: milliseconds)

Method	Min	LQ	Mean	Median	UQ	Max	N
Optimized	2.240994	2.620046	2.852079	2.861835	3.128039	3.513418	10
Classic	95.194832	97.700979	104.858526	100.833950	107.387926	122.114185	10

Based on the updated execution time data, it's clear that the optimized code without significant loops continues to outperform the classic method. The optimized approach has a minimum execution time of 2.241 milliseconds, while the classic method's minimum is 95.195 milliseconds. The mean execution time for the optimized method is 2.852 milliseconds, compared to 104.859 milliseconds for the classic version. This indicates that, on average, the optimized code is about 36 times faster. Additionally, the median execution time is significantly lower for the optimized method (2.862 milliseconds) compared to the classic method (100.834 milliseconds). Even in the worst-case scenario, the optimized code remains quicker, with a maximum time of 3.513 milliseconds versus 122.114 milliseconds for the classic method. Overall, these data confirm that the optimized code is not only faster but also more consistent in its performance, validating its efficiency over the classic approach.

8 Question 8

The theoretical acceptance rate depends on the parameter a , as this influences the shape of the target density $f(x)$. As a approaches its limit value a^* , the acceptance rate is likely to decrease, as the target density function becomes more difficult to approximate with the chosen proposal function. It seems that for some values of a , the empirical acceptance rate is close to 100%, while as it goes to a^*, it decreases. This suggests that the acceptance-rejection algorithm is more efficient when aa is low, but that its efficiency decreases as a increases and approaches a^* . The observations are in correlation with the plots.

Console Output

```
[1] "Empirical A-R = 100%, Theoretical A-R = 96.347%, rho = 5.207%"
[2] "Empirical A-R = 93.145%, Theoretical A-R = 89.527%, rho = 5.491%"
[3] "Empirical A-R = 89.856%, Theoretical A-R = 86.521%, rho = 5.683%"
[4] "Empirical A-R = 88.027%, Theoretical A-R = 84.799%, rho = 5.849%"
[5] "Empirical A-R = 85.301%, Theoretical A-R = 82.229%, rho = 6.058%"
[6] "Empirical A-R = 84.361%, Theoretical A-R = 80.227%, rho = 6.220%"
[7] "Empirical A-R = 66.559%, Theoretical A-R = 68.686%"
```

Newton-Raphson

Algorithm 1 Newton-Raphson Algorithm

function NEWTON_RAPHSON($u, a, \mu_1, \mu_2, \sigma_1, \sigma_2, \epsilon$, iter_max)
  iter $\leftarrow 0$
  $x_0 \leftarrow \frac{1}{1-a} \cdot (\mu_1 - a \cdot \mu_2)$
  $G_{x_0} \leftarrow F(x_0, a, \mu_1, \mu_2, \sigma_1, \sigma_2) - u$
  $dG_{x_0} \leftarrow \text{func}(x_0, a, \mu_1, \mu_2, \sigma_1, \sigma_2)$
  if $dG_{x_0} = 0$ **then**
    stop ("Error: dG(x0) needs to be different from 0.")
  end if
  while (abs$(G_{x_0}) \geq \epsilon$) **and** (iter $<$ iter_max) **do**
    $x_n \leftarrow x_0 - \frac{G_{x_0}}{dG_{x_0}}$
    $x_0 \leftarrow x_n$
    $G_{x_0} \leftarrow F(x_0, a, \mu_1, \mu_2, \sigma_1, \sigma_2) - u$
    $dG_{x_0} \leftarrow \text{func}(x_0, a, \mu_1, \mu_2, \sigma_1, \sigma_2)$
    iter $\leftarrow$ iter $+ 1$
  end while
  if iter $=$ iter_max $+ 1$ **then**
    $x_0 \leftarrow \text{qnorm}(u, \mu_1, \sigma_1)$
  end if
  return (x_0, G_{x_0})
end function