Thesis 9

Anna Lucia Lamacchia
Statistics

January 12, 2024

The Poisson Process

1 Meaning

The Poisson Process is a mathematical model that describes the timing of rare events occurring independently in a continuous time interval. These events are often characterized by their infrequency and lack of predictability between occurrences. In other words the Poisson Process serves as a mathematical framework to model the timing of rare, independent events occurring in continuous time. Named after the French mathematician Siméon Denis Poisson, the process is widely employed to model phenomena such as the arrival of customers at a service point, the decay of radioactive particles, or rare incidents in a given time period.

2 Properties

The Poisson Process has several key properties that make it a powerful and widely used tool in probability theory, statistics, and various applied fields. Some of the fundamental properties are:

1. **Stationary Increment**:
    - **Property**: The increment of the Poisson Process over a time interval depends only on the length of the interval and is independent of the starting time.
    - **Implication**: The process exhibits a consistent, time-independent behavior. This property simplifies calculations and analyses, allowing for a focus on the interval duration.

2. **Independent Increments**:

   - **Property**: Non-overlapping time intervals result in independent increments, meaning the number of events in one interval is independent of the number of events in another.
   - **Implication**: The randomness of event occurrences remains consistent across different time intervals. This property facilitates the modeling of systems where events are not influenced by past occurrences.

3. **Memoryless Property**:

   - **Property**: The time until the next event follows an exponential distribution, and the process possesses the memoryless property.
   - **Implication**: The probability of an event occurring in the next instant is independent of the past. This property is crucial for capturing the "no memory" characteristic of rare events.

4. **Time Homogeneity**:

   - **Property**: The Poisson Process is time-homogeneous, meaning the rate of event occurrences is constant over time.
   - **Implication**: The average rate λ remains constant, providing a simple and consistent parameter for modeling the process. This homogeneity makes predictions and analyses more straightforward.

5. **Infrequent Events**:

   - **Property**: The Poisson Process is designed to model rare events, where the probability of multiple events occurring in a very short time interval is negligible.
   - **Implication**: It is well-suited for scenarios involving sporadic incidents, such as arrivals at a service point or rare failures in a system.

6. **Non-Negative Integer Values**:

   - **Property**: The number of events in any fixed interval is a non-negative integer.
   - **Implication**: The Poisson Process provides a discrete framework for modeling event occurrences, making it applicable in scenarios where events can be counted.

3 Simulation in Javascript

To simulate the Poisson Point Process, this JavaScript code takes by user input the number of systems attacked M, the number of attacks N, the λ parameter and the time interval T such that the probability $p = \frac{\lambda N}{T}$ is very small. After, it simulates the score trajectory for each system and if the random probability that attack i happens is less than p, the system is penetrated, so the score trajectory is set to -1, otherwise it is set to 1. The code has been written for the Homework 5 and a more detailed explanation can be obtained following this link: Homework 5

```
function startSimulation() {
  numSystems = parseInt(numSystemsInput.value);
  numAttacks = parseInt(numAttacksInput.value);
  T = parseInt(document.getElementById("time").value);
  lambda = parseFloat(document.getElementById("successProbabilityInput").value);
  successProbabilityInput = lambda * T /(parseInt(numAttacksInput.value)) ;

  scores = [];

  scores.length = 0;

  // Simulate security scores for all systems
  for (let i = 0; i < numSystems; i++) {
    const systemScores = simulateSecurityScores(numAttacks, successProbabilityInput)[0];
    scores.push(systemScores);
  }

  drawSecurityScores(scores,canvas,ctx);
}

startSimulationButton.addEventListener("click", startSimulation);

// Function to simulate security scores for a single system
function simulateSecurityScores(numAttacks, p) {
  const scores = [];

  let score = 0;

  for (let attack = 1; attack <= numAttacks; attack++) {
    if (Math.random() < p) {
      score += 1; // System is penetrated
    }
    scores.push(score);
    }

  return [scores];
}
```

Figure 1: Code

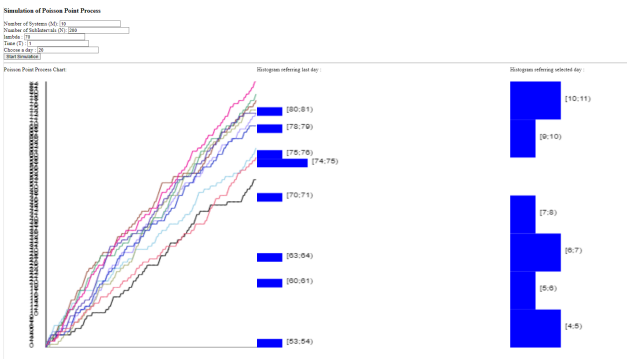

Figure 2: Chart