UNIVERSITY OF PISA

MSc in Computer Engineering

Project for Performance Evaluation of Computer Systems and Networks

Multi-core scheduling

Professors:

Prof. Giovanni Stea

Ing. Giovanni Nardini

Students:

Taulant Arapi (645308)

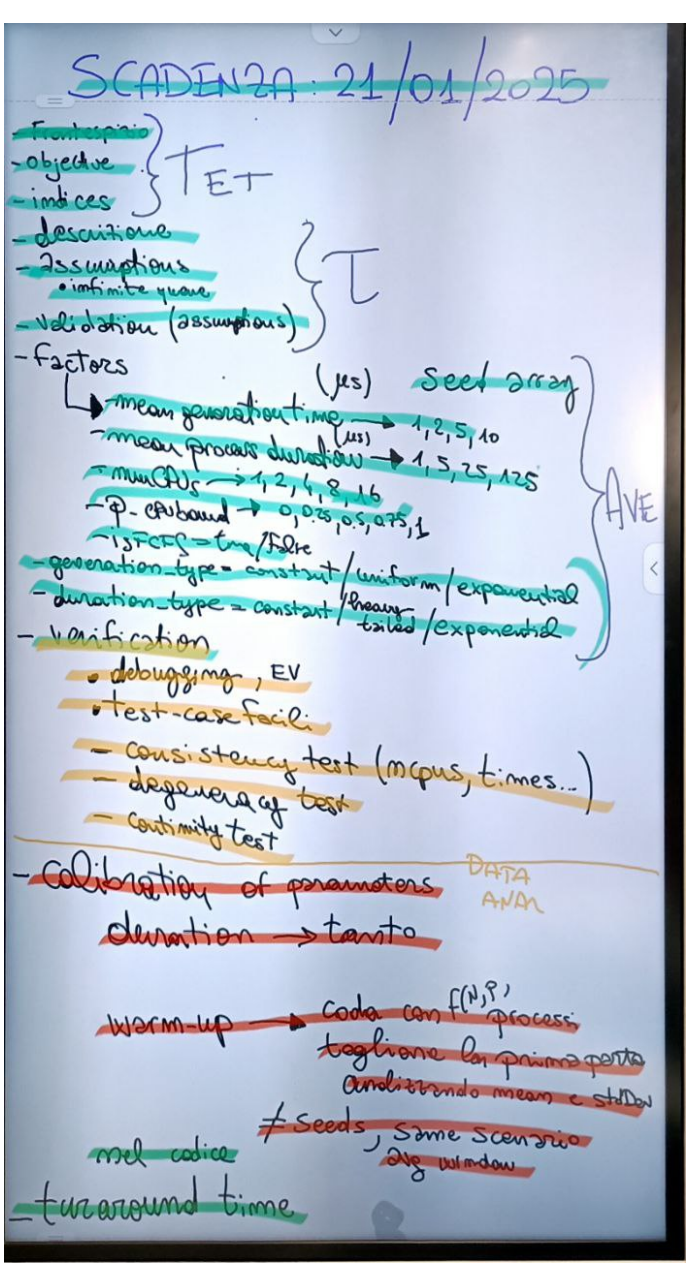
Francesco Barcherini (645413)

Antonio Ciociola (645324)

ACADEMIC YEAR 2024/2025

Contents

1 Photos


SCADENZA: 21/01/2025
- Frontespizio
- obiettivo
- indici
- descrizione
- assumptions
• infinite queue
- validation (assumptions)
TEST
T
- factors
(yes) Seed array
- mean generation time (us) → 1, 2, 5, 10
- mean process duration → 1, 5, 25, 125
- numCPUs → 1, 2, 4, 8, 16
- p-cpubound → 0, 0.25, 0.5, 0.75, 1
- SJFCFS = true/false
- generation_type = constant/uniform/exponential
- duration_type = constant/heavy-tailed/exponential
AVE
- verification
• debugging, EV
• test-case facili
- consistency test (ncpus, times...)
- degenerate test
- continuity test
- Calibration of parameters
DATA ANA
durations → tanto
warm-up → coda con f(N,P) processi
togliere la prima parte
analizzando mean e stdDev
≠ seeds, same scenario
avg window
nel codice
- turnaround time
- generation_type = constant/uniform/exponential
- duration_type = constant/heavy-tailed/exponential
- verification
• debugging, EV
• test-case facili
- consistency test (ncpus, times...)
- degenerate test
- continuity test
- Calibration of parameters
DATA ANA
durations → tanto
warm-up → coda con f(N,P) processi
togliere la prima parte
analizzando mean e stdDev
≠ seeds, same scenario
avg window
nel codice
- turnaround time
- waiting time
- utilizzazione CPU
- CPU attive nel tempo (t)
- Processi in coda pronti (t)
AVE
Dopo warm-up grafici

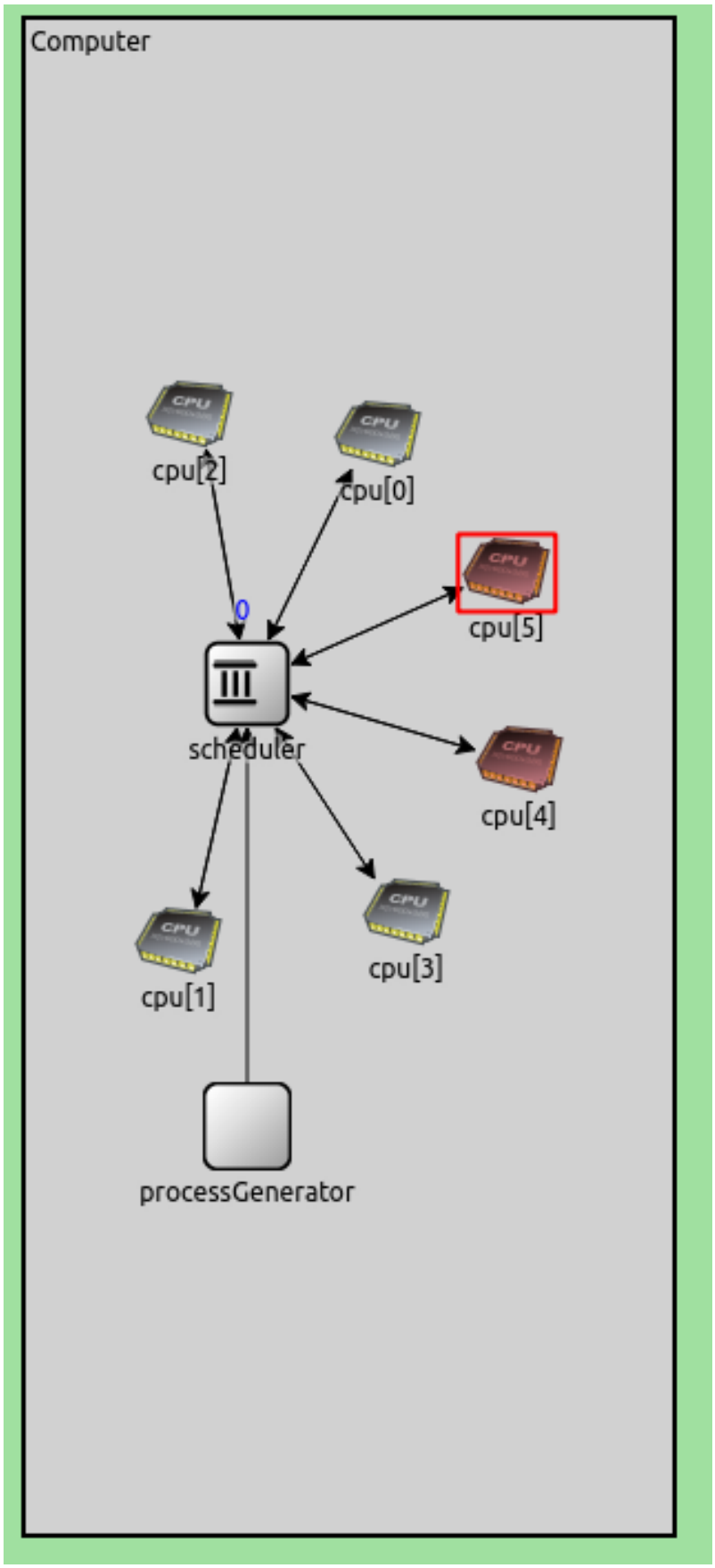


Figure 1.1: Caption

2 Introduction

2.1 Description and specification

It is required to analyze a multi-core computer equipped with N CPUs, which execute multiple interactive processes. The processes are dynamically generated at intervals of T seconds. Each process has a total duration D, which is divided into three distinct execution phases:

1. initial processing phase;
2. I/O operation phase;
3. final processing phase.

The times T and D are defined as IID random variables. The processes are categorized into two types and generated as *CPU bound* with probability p or *I/O bound* with probability $1 - p$. The difference is that:

- in CPU-bound processes the I/O operation phase constitutes 20% of the total duration D, and the other two phases both the 40%;
- in I/O-bound processes the I/O operation phase constitutes 80% of their total duration D, and the other two phases both the 10%;

Processes are assigned to CPUs by the operating system's scheduler, which selects tasks from a list of "ready" processes. When a CPU becomes idle for entering the I/O operation phase or ending the final processing phase of a process, the scheduler immediately assigns it to a new ready process, if present. Once a process completes its I/O operation phase, it is marked as ready again and is returned to the scheduling queue. A process exits the system once it has successfully completed all three phases of execution.

The scheduler can be implemented to follow two distinct policies:

- First Come First Served (FCFS): processes are scheduled in the order of their arrival after the generation or after the end of the I/O phase;
- Shortest Job First (SJF): scheduling is determined based on the shortest remaining time until either the process's I/O phase or its completion.

2.2 Objectives

The aim of this project is to assess and analyze the execution of a multi-core scheduling system under varying operational scenarios. The objective is to determine the optimal characteristics of the systems to make it work without the need to discard processes. Furthermore, the analysis aims to assess the conditions under which the best system behavior is achieved in terms of resource use efficiency and time performance evaluation. In particular, the different configurations taken into consideration are the scheduling policies, the number of CPUs in the system, the type of the processes (CPU bound or I/O bound), the generation intervals and the duration of processes.

The overarching goal is to generate insights into optimal scheduling strategies and resource allocation practices for multi-core environments, enhancing system throughput and minimizing delays.

2.3 Indices

To evaluate the system's performance comprehensively, the following indices will be analyzed in detail:

- turnaround time R: defined as the total time elapsed from the arrival of a process in the system to its completion. This metric provides a measure of the computer's overall responsiveness and of the time required for different types of processes to be executed;
- waiting time W: the cumulative time [!!: COME LO CALCOLIAMO?] a process spends in the ready queue before being assigned to a CPU for execution. This metric represents the impact on the turnaround time of the queuing policy and of the workload of the system;
- CPU utilization of the n_{th} CPU ρ_n: the fraction of time that each CPU is actively engaged in executing processes instead of being idle. This metric reflects the efficiency of resource usage within the system;
- active CPUs over time N_A: a dynamic measure of the number of CPUs actively processing tasks at any given moment. This index helps assess load distribution and system evolution;
- queue length over time N_q: tracks the size of the ready queue throughout the simulation, highlighting bottlenecks and variations in process scheduling.

These indices will be statistically analyzed to identify trends, anomalies, edge cases and key factors influencing system performance. By examining these metrics under diverse configurations of N, p and scheduler policies, the project aims to provide actionable recommendations for improving the efficiency and adaptability of multi-

core scheduling systems.

2.4 Assumptions?

To perform the analysis, the following assumptions are made:

- There is sufficient memory to store the list of processes ready for execution, so the ready queue will be considered as infinite.
- The scheduler itself has a negligible load on the CPUs; there is no delay between the executions of the modeled processes. [!!: assumiamo che sia zero? if not, quanto facciamo?]
- With the shortest job first (SJF) scheduling, an accurate estimate of the execution times is known a priori. If the scheduler is FCFS, this data is not used.

3 Implementation

We implemented the Computer system in OMNeT++.
Our implementation consists of the following modules, also shown in Figure 3.1:

- `ProcessGenerator`: this module generates processes following the specified time distribution and sends them to the scheduler. The duration of each processing phase is also defined here.
- `scheduler`: this module handles the scheduling logic and keeps track of processes in their I-O phase.
  It contains an infinite-capacity queue. Depending on the `isFCFS` parameter, the queue is either a FIFO queue, when the parameter is set to true, or a priority queue whose elements are ordered by increasing duration.
- `cpu`: each of the N CPUs, when it is free, receives a process from the scheduler and simulates its execution. When it finishes the execution phase (i.e. the I-O phase is reached or the process terminates), a message is sent to the scheduler, indicating that the CPU is now free.

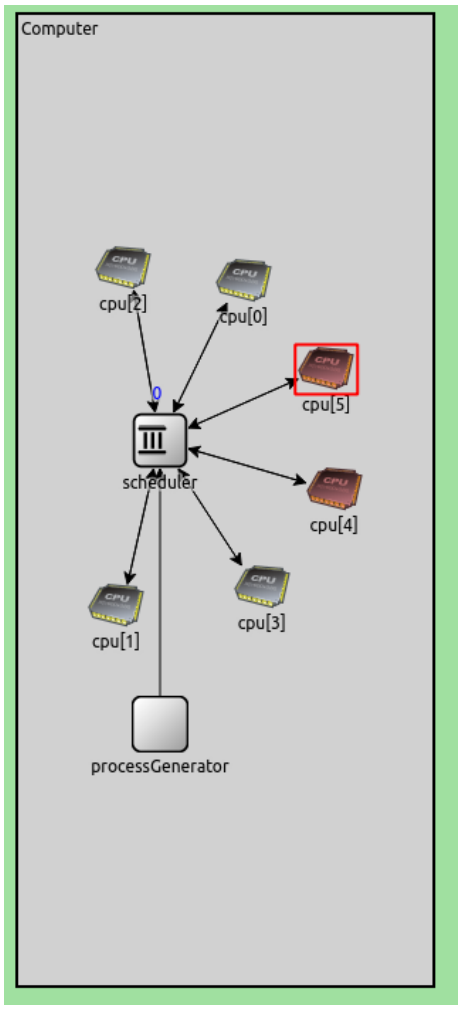


Figure 3.1: View of the system implementation in OMNeT++

4 Verification

5 Analysis

5.1 Calibration of the parameters

Having a lot of parameters to calibrate, we decided to limit the number of possible values for each parameter. This choice was made to reduce the number of simulations to be performed and to have a more manageable number of results to analyze. The parameters that we decided to calibrate are the following:

- `meanGenerationTime`: {20ms, 50ms}
  The mean for the generation times of the processes was chosen by considering a system under heavy and medium load.
- `meanProcessDuration`: {40ms, 100ms, 200ms}
  The mean for the duration of the processes was chosen after the estimations on the system stability. These values permit to have some combination of parameters that lead to a stable system and others that lead to an unstable system.
- `pCpuBound`: {0.25, 0.75}
  The probability of a process being CPU-bound has two values, one for a system with a high number of I-O bound processes and the other for a system with a high number of CPU-bound processes.
- `numCpus`: {1, 4, 12}
  The number of CPUs was chosen to simulate single-core, quad-core, and newer systems with 12 cores.
- `isFCFS`: {true, false}
  The scheduling policy that the scheduler uses can be either First-Come-First-Served or Shortest-Job-First.
- `generationType`: {"exponential", "uniform"}
  In addition to the exponential distribution, we also considered the uniform distribution for the generation of processes.
- `durationType`: {"exponential", "uniform"}
  In addition to the exponential distribution, we also considered the uniform distribution for the duration of the processes.

5.2 Time parameters setup

5.2.1 Warm-up period

To determine the warm-up period, we observed the development of the mean number of busy CPUs over time, through 10 independent runs and with different parameters configurations. The number of busy CPUs is a good indicator of the system's stability, as it doesn't depend on the scheduling policy. Our observations showed that the system stabilizes well before 200 s in all tested configurations. To handle variability and ensure robustness in worse cases, we set the warm-up period to 200 s.

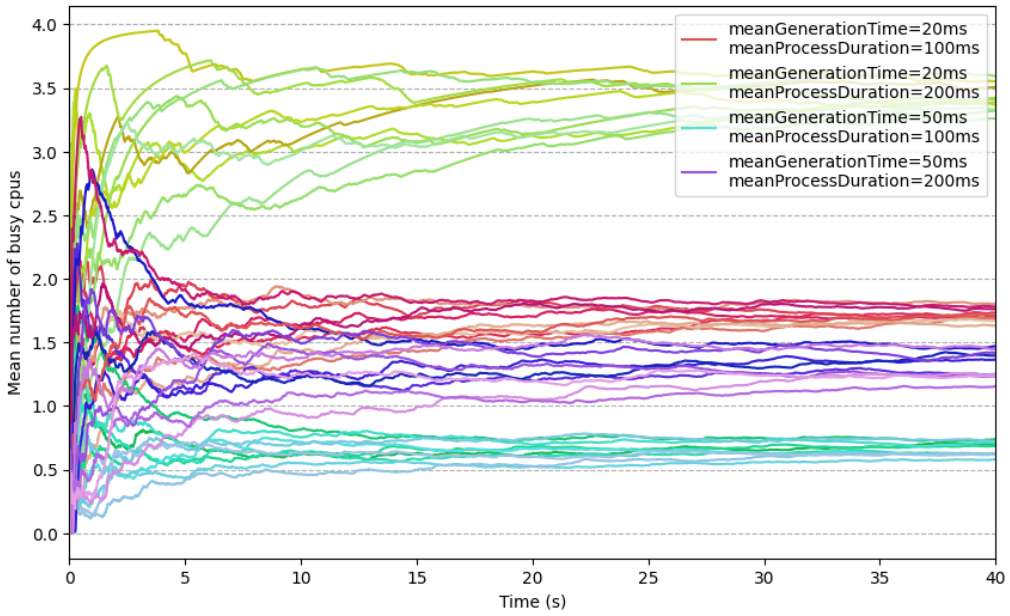


Figure 5.1: Line chart of the mean number of busy CPUs over time for different configurations of meanGenerationTime and meanProcessDuration and multiple repetitions.

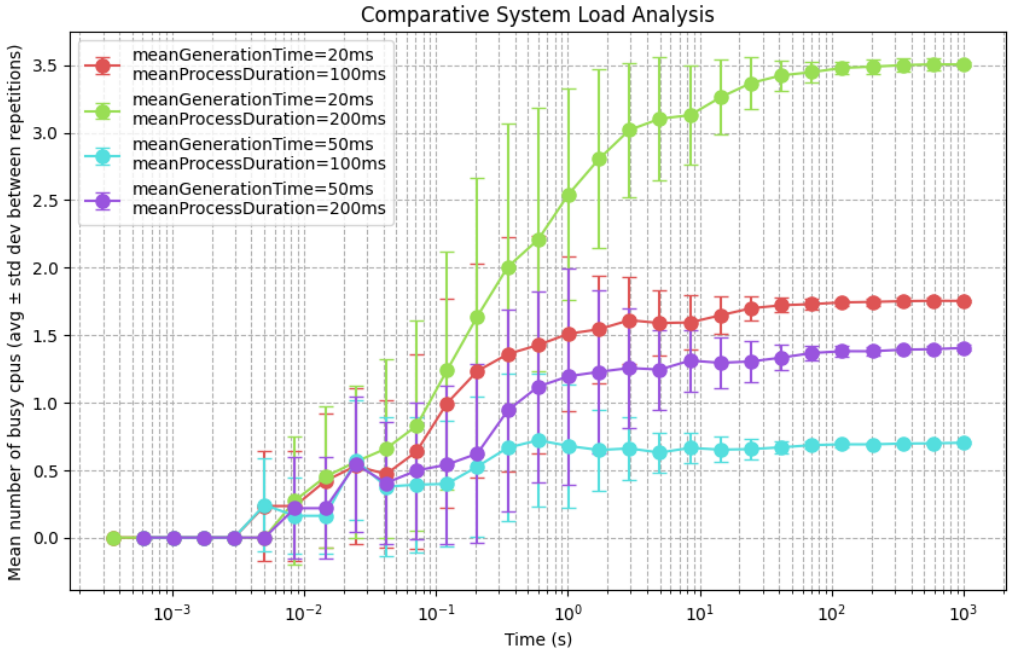


Figure 5.2: Average and Std dev of the mean number of busy CPUs over the repetitions.

	20ms, 100ms		20ms, 200ms		50ms, 100ms		50ms, 200ms	
Time (s)	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev
1	1.52	0.58	2.54	0.78	0.68	0.46	1.20	0.80
2	1.54	0.37	2.85	0.64	0.67	0.27	1.25	0.51
5	1.59	0.24	3.10	0.46	0.63	0.15	1.24	0.29
10	1.63	0.18	3.18	0.34	0.65	0.10	1.29	0.20
20	1.68	0.11	3.33	0.22	0.64	0.08	1.27	0.15
50	1.72	0.05	3.43	0.10	0.68	0.05	1.35	0.09
100	1.74	0.02	3.47	0.05	0.69	0.02	1.37	0.04
200	1.75	0.02	3.49	0.05	0.69	0.02	1.38	0.03
500	1.75	0.03	3.50	0.05	0.70	0.01	1.40	0.02
1000	1.75	0.01	3.51	0.02	0.70	0.01	1.41	0.02

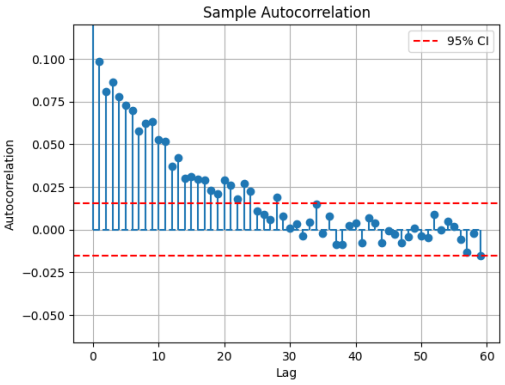
Table 5.1: Average and Std dev of the mean number of busy CPUs over the repetitions.

5.2.2 Simulation duration

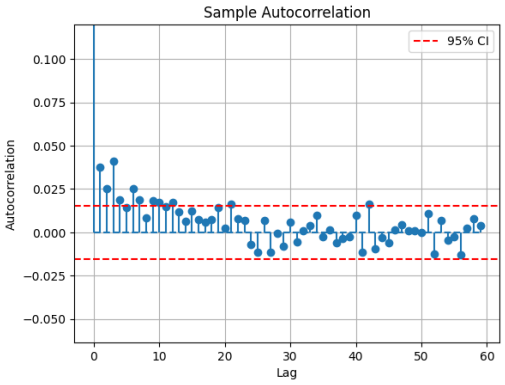
For the simulation duration, we chose to end it at 1000 s. This ensures that after discarding the initial 200 s warm-up period, there remains 800 s of simulation data for analysis. Given that the mean generation time is on the order of tenths of seconds, this duration strikes a balance between obtaining statistically meaningful results even after subsampling and maintaining reasonable simulation times.

5.2.3 Subsampling

To analyze the data, the assumption of IID-ness will be needed, but without further modification the samples do not uphold it. For instance, if a process finishes with a large turnaround time, which is caused by a long queue, it is likely that the same thing will happen for the next processes. To address this and ensure independence between samples, subsampling has been employed. The new sample is constructed taking each point of the starting sample with probability p. $p = \frac{1}{2^k}$ and k is the smallest integer that makes $|R(j)| < z_{\alpha/2}/\sqrt{n}$ for each j between 1 and 10. This method for choosing k was chosen since only nearby points are correlated. Before trying a bigger k the condition is tested multiple times with different seeds, this makes sure that p doesn't become too small due to unlucky sampling. The closer the system is to saturation, the stronger the correlation becomes. As a result, the needed k will become larger, leading to a smaller subsample size.

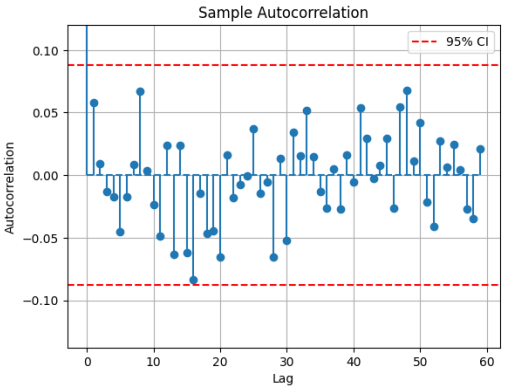


(a) System close to saturation.

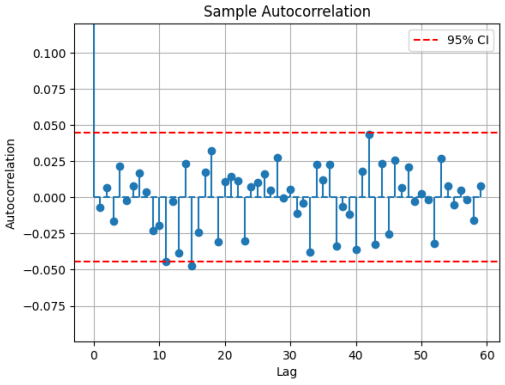


(b) System with high load but not as close to saturation.

Figure 5.3: Comparison of autocorrelation without subsampling for different loads.



(a) System close to saturation with $p = 1/32$.



(b) System with high load but not as close to saturation with $p = 1/8$.

Figure 5.4: Comparison of autocorrelation with subsampling for different loads.

6 Conclusions

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

```
1 // Your First C++ Program
2
3 #include <iostream>
4
```

```cpp
int main() {
    std::cout << "Hello World!";
    return 0;
}
```

Listing 6.1: `test.cpp`

Nunc sed pede. Praesent vitae lectus. Praesent neque justo, vehicula eget, interdum id, facilisis et, nibh. Phasellus at purus et libero lacinia dictum. Fusce aliquet. Nulla eu ante placerat leo semper dictum. Mauris metus. Curabitur lobortis. Curabitur sollicitudin hendrerit nunc. Donec ultrices lacus id ipsum.

[Persona 1: Commento 1]

[Persona 2: Commento 2]

Test cite[1].

Test footnote[1].

Figure 6.1: Un gattino.

Ciao

Nunc sed pede. Praesent vitae lectus. Praesent neque justo, vehicula eget, interdum id, facilisis et, nibh. Phasellus at purus et libero lacinia dictum. Fusce aliquet. Nulla eu ante placerat leo semper dictum. Mauris metus. Curabitur lobortis. Curabitur sollicitudin hendrerit nunc. Donec ultrices lacus id ipsum.

[1] Footnote

Bibliography

[1] Google. *Google*. 2025. https://google.com.