Final Project
STAT 4110 Statistical Simulation
Winter 2026 Semester

School of Mathematical and Computational Sciences
University of Prince Edward Island
Ali Raisolsadat

November 21, 2025

Overview

This project is designed to familiarize students with real-world applications of statistical simulation, stochastic modelling, and analytical reasoning. The available options for the final project may involve developing a computational model, implementing simulation techniques covered in class, and using simulation-based or analytical methods to study system behaviour. Students are free to use any dataset or programming language of their choice, unless otherwise specified.

Choose **one** of the following three project directions:

1. Use your modelling skills to model and implement **one of the following three** real world problems:
   - Bayesian Neural Networks using Markov Chain Monte Carlo
   - Numerical Option Pricing: Crank–Nicolson PDE vs. Monte Carlo Simulation
   - Simulation of Realistic Website Queueing and Load Management Systems
2. Replicating the full design and implementation of **one** published paper related to our course material
3. Literature review of **5–15 papers** on a topic related to the course

Projects will be completed in teams of three members. Each team will be responsible for defining the scope, data sources, and motivation of their work.

Deliverables

For any of the project choices, the deliverables are as follows:

1. **Introductory Summary** (2 pages max, 5/30 points): Includes a brief description of the chosen project, references and papers that the group will use, and the roles of each student in the group.
2. **Presentation** (15 minutes + 5 minutes Q&A, 10/30 points): A concise presentation of the methodology, results, and insights from the project.

3. **Summary Report** (5 pages max, 15/30 points): A detailed report summarizing the approach, implementation, results, and discussion. Include any supporting figures, tables, or appendices as needed. You are also responsible to attach a ZIP file containing all code used in the project. If you are using Git (GitHub or GitLab), please add me as the owner, so I can review your code.

Disclaimer: The use of LLMs, including GPT, is permitted for coding assistance only. Any use of such tools must be explicitly acknowledged in the report.

Project Choice 1: Modelling Option

Purpose

The purpose of this project choice is to give you hands-on experience in developing and analysing statistical/mathematical models for real-world problems. By working on a modelling-based project, you will learn how to formalize assumptions, implement computational methods, and interpret model outcomes. This choice emphasizes on using the material learnt throughout the course and understanding the behaviour of complex systems under uncertainty

Learning Objectives

- Design and implement a computational model for a selected application.
- Apply simulation or numerical methods to approximate solutions when analytic solutions are impossible.
- Compare model predictions with alternative approaches or benchmarks.
- Communicate methodology, results, and insights effectively through written reports and presentations.

Scope

You are expected to choose **one of the three** modelling-based problems:

1. Bayesian Neural Networks using Markov Chain Monte Carlo
2. Numerical Option Pricing: Crank–Nicolson PDE vs. Monte Carlo Simulation
3. Simulation of Realistic Website Queueing and Load Management Systems

As a team, you are responsible for selecting appropriate datasets, defining the project scope, and documenting their methodology and results.

Option 1: Bayesian Neural Networks via MCMC

1. Background and Motivation

Traditional neural networks optimize weights to single point estimates, usually via gradient descent. This approach does not quantify uncertainty, which is important when:

- making decisions in risk-sensitive applications (medicine, finance, engineering)
- detecting anomalies or rare events
- forecasting with small datasets
- requiring confidence intervals or credible predictions

Bayesian neural networks (BNNs) treat the network weights as random variables with prior distributions. The posterior distribution of the weights given observed data allows us to capture uncertainty:

$$p(w \mid X, y) = \frac{p(y \mid X, w)\, p(w)}{p(X, y)},$$

where w are the weights, X the inputs, y the observed outputs, $p(w)$ the prior, and $p(y \mid X, w)$ the likelihood.

Because the posterior is usually intractable, we approximate it using Markov Chain Monte Carlo (MCMC) methods such as:

- Metropolis–Hastings
- Gibbs sampling
- Hamiltonian Monte Carlo (HMC)

2. Model Structure

Consider a simple feedforward neural network with input x, one hidden layer of H neurons, and output y. Denote the network function as $f(x; w)$.

Priors: Assign independent Gaussian priors to weights and biases:

$$w \sim \mathcal{N}(0, \sigma_w^2), \quad b \sim \mathcal{N}(0, \sigma_b^2).$$

Likelihood (for regression): Assume Gaussian noise on outputs:

$$y \mid x, w \sim \mathcal{N}(f(x; w), \sigma^2).$$

Posterior: The posterior distribution of the weights is proportional to:

$$p(w \mid X, y) \propto p(y \mid X, w)p(w).$$

Posterior predictive distribution: For a new input x_{new}, approximate the predictive distribution by averaging over posterior samples:

$$\hat{y}(x_{\text{new}}) \approx \frac{1}{S} \sum_{s=1}^{S} f(x_{\text{new}}; w^{(s)}),$$

where $w^{(1)}, \ldots, w^{(S)}$ are sampled from the posterior using MCMC.

3. Datasets

You can use any dataset, but it is recommended to choose **small or simple datasets** so that MCMC sampling completes in reasonable time. Examples include:

- **Tabular / regression:** Boston Housing, Diabetes, or synthetic datasets such as $y = \sin(x) + \epsilon$.
- **Time series:** Stock prices (subset), daily sales, or temperature data.
- **Classification / small images:** Iris dataset, or a subset of MNIST (e.g., 1000–5000 images).

The key is to keep the dataset size manageable for a few thousand MCMC iterations.

4. Suggested Experiments

- Vary the prior variance σ_w^2 and observe the effect on predictive uncertainty.
- Change the noise variance σ^2 and compare predictive intervals.
- Compare performance and uncertainty estimates between MCMC BNN and standard NN.
- Optional: Vary the number of hidden neurons or layers and observe convergence of MCMC samples.

5. Student Tasks

1. Implement a Bayesian neural network using MCMC.
2. Generate posterior samples for the weights and biases.
3. Compute posterior predictive distributions for test inputs.
4. Investigate how predictions and uncertainty change when:
    - the prior variance changes
    - the output noise changes
    - the network size changes
5. Compare accuracy and efficiency with a standard neural network.

Option 2: Black–Scholes Model, Monte Carlo Pricing, and Crank–Nicolson Finite Differences

1. What is the Black–Scholes model and why use it?

The Black–Scholes model is a continuous-time stochastic model for the evolution of a financial asset price S_t. It assumes that the asset price follows a geometric Brownian motion (GBM):

$$dS_t = \mu S_t\, dt + \sigma S_t\, dW_t,$$

where

- μ is the instantaneous expected return (drift),
- $\sigma > 0$ is the volatility (instantaneous standard deviation),
- W_t is a standard Brownian motion.

The model is used because (i) it is analytically tractable in many cases, (ii) under risk-neutral valuation it leads to a linear parabolic PDE for derivative prices, and (iii) it captures multiplicative stochastic growth and log-normal marginal distributions for S_t. The Black–Scholes framework underpins much of modern quantitative finance and provides a baseline for comparing numerical methods (PDE solvers, Monte Carlo, lattice models, etc.).

2. Risk-neutral pricing and derivation of the Black–Scholes PDE

Consider a derivative security with payoff at maturity T given by $\Phi(S_T)$. The *no-arbitrage* (risk-neutral) pricing formula states that the arbitrage-free price at time t is the discounted risk-neutral expectation:

$$V(S_t, t) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\big[\, \Phi(S_T) \mid S_t \,\big],$$

where r is the risk-free rate and $\mathbb{Q}$ denotes the risk-neutral measure. Under the risk-neutral measure the asset dynamics become

$$dS_t = rS_t\, dt + \sigma S_t\, dW_t^{\mathbb{Q}}.$$

Applying Itô's formula to $V(S_t, t)$ gives

$$dV = \left(\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \tfrac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}\right) dt + \sigma S \frac{\partial V}{\partial S} dW_t^{\mathbb{Q}}.$$

Under risk-neutral valuation, the discounted derivative price must be a martingale, which leads to the Black–Scholes backward PDE:

$$\boxed{\frac{\partial V}{\partial t} + \tfrac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0}.$$

Terminal condition: $V(S, T) = \Phi(S)$.

3. Probabilistic (Monte Carlo) representation

The derivative price can be approximated by the risk-neutral expectation:

$$V(S_0, 0) = e^{-rT}\mathbb{E}^{\mathbb{Q}}[\Phi(S_T) \mid S_0].$$

Under GBM,

$$S_T = S_0 \exp\Big(\big(r - \tfrac{1}{2}\sigma^2\big)T + \sigma\sqrt{T}\, Z\Big), \qquad Z \sim \mathcal{N}(0, 1).$$

3.1 Euler–Maruyama discretization

Discretize the interval $[0, T]$ into N steps $\Delta t = T/N$. The Euler–Maruyama scheme:

$$S_{n+1} = S_n + rS_n\Delta t + \sigma S_n\sqrt{\Delta t}Z_n, \quad Z_n \sim N(0, 1),$$

or the log-Euler form:

$$S_{n+1} = S_n \exp\Big((r - \tfrac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}\, Z_n\Big).$$

3.2 Monte Carlo pricing of European options

Simulate M paths $S_T^{(m)}$ and compute:

$$V_0 \approx e^{-rT}\frac{1}{M}\sum_{m=1}^{M}\Phi\big(S_T^{(m)}\big).$$

Remarks:

- Euler–Maruyama is first-order accurate in Δt for strong convergence.
- Monte Carlo converges as $O(M^{-1/2})$; variance reduction can improve efficiency.
- Flexible for path-dependent options and multi-asset problems.

4. Numerical solution of the PDE: Crank–Nicolson finite differences

Finite-difference methods discretize the PDE on a grid in asset price S and time t. Crank–Nicolson (CN) is a widely used implicit scheme that is second-order accurate in both space and time and is unconditionally stable (for the linear parabolic PDEs we consider).

Below we derive the CN discretization and obtain the tridiagonal linear system that must be solved at each time step.

4.1 Domain and grid

Truncate the infinite spatial domain $S \in (0, \infty)$ to a large finite domain $S \in [S_{\min}, S_{\max}]$. A common choice is $S_{\min} = 0$ and $S_{\max} = LS_0$ with $L = 3$ or 4, chosen so the option value is insensitive to further increases in $S_{\max}$.

Define a uniform spatial grid with $M + 1$ nodes:

$$S_i = S_{\min} + i\Delta S, \qquad \Delta S = \frac{S_{\max} - S_{\min}}{M}, \quad i = 0, 1, \dots, M.$$

Define a temporal grid stepping backward from T to 0 with N time-steps:

$$t^n = n\Delta t, \qquad \Delta t = \frac{T}{N}, \quad n = 0, 1, \dots, N.$$

We will denote the numerical approximation to $V(S_i, t^n)$ by V_i^n. (Because the PDE is typically integrated backward in time from $t = T$ to $t = 0$, many authors index time reversely; here we take $n = 0$ for time $t = 0$ and note the CN update moves from known V^n to V^{n+1}.)

4.2 Spatial derivatives approximation

Approximate first and second derivatives by centered finite differences:

$$\left.\frac{\partial V}{\partial S}\right|_{S_i,t^n} \approx \frac{V_{i+1}^n - V_{i-1}^n}{2\Delta S}, \qquad \left.\frac{\partial^2 V}{\partial S^2}\right|_{S_i,t^n} \approx \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n}{(\Delta S)^2}.$$

4.3 Semi-discrete operator

Plugging these into the PDE, at grid node (S_i, t^n) the differential operator

$$\mathcal{L}V := \tfrac{1}{2}\sigma^2 S^2 V_{SS} + rSV_S - rV$$

is approximated by

$$(\mathcal{L}V)_i^n \approx \tfrac{1}{2}\sigma^2 S_i^2 \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n}{(\Delta S)^2} + rS_i \frac{V_{i+1}^n - V_{i-1}^n}{2\Delta S} - rV_i^n.$$

It is convenient to rewrite this as a linear combination

$$(\mathcal{L}V)_i^n \approx \alpha_i^n V_{i-1}^n + \beta_i^n V_i^n + \gamma_i^n V_{i+1}^n,$$

with coefficients (dropping the explicit time index for the coefficients since they depend only on S_i):

$$\alpha_i := \frac{1}{2}\sigma^2 S_i^2 \frac{1}{(\Delta S)^2} - \frac{rS_i}{2\Delta S},$$

$$\beta_i := -\frac{1}{2}\sigma^2 S_i^2 \frac{2}{(\Delta S)^2} - r,$$

$$\gamma_i := \frac{1}{2}\sigma^2 S_i^2 \frac{1}{(\Delta S)^2} + \frac{rS_i}{2\Delta S}.$$

Thus

$$(\mathcal{L}V)_i^n \approx \alpha_i V_{i-1}^n + \beta_i V_i^n + \gamma_i V_{i+1}^n.$$

4.4 Crank–Nicolson time discretization

Crank–Nicolson averages the spatial operator at times t^n and t^{n+1}. The time derivative is approximated by

$$\frac{V_i^{n+1} - V_i^n}{\Delta t} \approx \frac{1}{2}\big((\mathcal{L}V)_i^{n+1} + (\mathcal{L}V)_i^n\big).$$

Rearrange to obtain the CN update equation:

$$V_i^{n+1} - \frac{\Delta t}{2}(\mathcal{L}V)_i^{n+1} = V_i^n + \frac{\Delta t}{2}(\mathcal{L}V)_i^n.$$

Substituting the finite-difference representation of $\mathcal{L}$ yields a linear relation for interior nodes $i = 1, \dots, M-1$:

$$V_i^{n+1} - \frac{\Delta t}{2}\big(\alpha_i V_{i-1}^{n+1} + \beta_i V_i^{n+1} + \gamma_i V_{i+1}^{n+1}\big) = V_i^n + \frac{\Delta t}{2}\big(\alpha_i V_{i-1}^n + \beta_i V_i^n + \gamma_i V_{i+1}^n\big).$$

Collect terms for V_{i-1}^{n+1}, V_i^{n+1}, V_{i+1}^{n+1} on the left-hand side and known V^n terms on the right-hand side. Define coefficients:

$$\begin{aligned} A_i &:= -\frac{\Delta t}{2}\alpha_i, \\ B_i &:= 1 - \frac{\Delta t}{2}\beta_i, \qquad \text{(left-hand side coefficients)} \\ C_i &:= -\frac{\Delta t}{2}\gamma_i, \end{aligned}$$

and

$$\begin{aligned} \widetilde{A}_i &:= \frac{\Delta t}{2}\alpha_i, \\ \widetilde{B}_i &:= 1 + \frac{\Delta t}{2}\beta_i, \qquad \text{(right-hand side coefficients).} \\ \widetilde{C}_i &:= \frac{\Delta t}{2}\gamma_i, \end{aligned}$$

Then the CN equation becomes, for each interior i,

$$A_i V_{i-1}^{n+1} + B_i V_i^{n+1} + C_i V_{i+1}^{n+1} = \widetilde{A}_i V_{i-1}^n + \widetilde{B}_i V_i^n + \widetilde{C}_i V_{i+1}^n.$$

4.5 Matrix form

Stack the interior unknowns into a vector

$$\mathbf{V}^n = \begin{bmatrix} V_1^n \\ V_2^n \\ \vdots \\ V_{M-1}^n \end{bmatrix}.$$

For each time-step the system can be written compactly as

$$\boxed{A\,\mathbf{V}^{n+1} \;=\; B\,\mathbf{V}^n \;+\; \mathbf{d}^n},$$

where A and B are $(M-1)\times(M-1)$ tridiagonal matrices with entries given by the A_i, B_i, C_i and $\widetilde{A}_i, \widetilde{B}_i, \widetilde{C}_i$ coefficients, and $\mathbf{d}^n$ is a vector that collects boundary contributions. Explicitly, A has the form

$$A = \begin{bmatrix} B_1 & C_1 & 0 & \cdots & 0 \\ A_2 & B_2 & C_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & A_{M-2} & B_{M-2} & C_{M-2} \\ 0 & \cdots & 0 & A_{M-1} & B_{M-1} \end{bmatrix},$$

and B is similarly

$$B = \begin{bmatrix} \widetilde{B}_1 & \widetilde{C}_1 & 0 & \cdots & 0 \\ \widetilde{A}_2 & \widetilde{B}_2 & \widetilde{C}_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \widetilde{A}_{M-2} & \widetilde{B}_{M-2} & \widetilde{C}_{M-2} \\ 0 & \cdots & 0 & \widetilde{A}_{M-1} & \widetilde{B}_{M-1} \end{bmatrix}.$$

The boundary vector $\mathbf{d}^n$ accounts for known values V_0^n and V_M^n appearing in the finite-difference stencils for the first and last interior nodes. For example, the first equation (for $i = 1$) includes terms proportional to V_0^n and V_0^{n+1}; move these to the right-hand side to obtain the appropriate entry in $\mathbf{d}^n$.

Boundary conditions. Reasonable boundary conditions for European call options:

$$V(S_{\min} = 0, t) = 0, \qquad \text{(call is worthless at zero)}$$
$$V(S_{\max}, t) \approx S_{\max} - Ke^{-r(T-t)}.$$

Use these values to compute contributions to $\mathbf{d}^n$. If one uses $S_{\min} = 0$ then $V_0^n = 0$ simplifies the first equation. For the last equation, known V_M^n provides a known term on the right-hand side.

4.6 Solving the tridiagonal system (Thomas algorithm)

At each time-step we must solve the tridiagonal linear system

$$A\,\mathbf{V}^{\,n+1} = \mathbf{r}^{\,n}, \qquad \text{where } \mathbf{r}^{\,n} = B\,\mathbf{V}^{\,n} + \mathbf{d}^{\,n}.$$

Because A is tridiagonal and diagonally dominant for reasonable grids and parameters, the Thomas algorithm (tridiagonal matrix algorithm) solves this in $O(M)$ operations with two sweeps:

1. *Forward elimination:* modify diagonal and right-hand side entries to eliminate subdiagonal entries.
2. *Backward substitution:* compute unknowns $V_{M-1}^{n+1}, V_{M-2}^{n+1}, \ldots, V_1^{n+1}$.

This yields an efficient solver and is typically preferred to general-purpose LU decompositions for 1-D PDEs.

5. Implementation notes and practical considerations

Choice of grid and accuracy. The CN scheme is second-order accurate in both ΔS and Δt. In practice choose Δt and ΔS so that the leading truncation errors are balanced (e.g. $\Delta t \sim (\Delta S)^2$ when dominated by diffusion terms), and perform grid-refinement studies (halving ΔS, Δt) to observe convergence.

Payoff discontinuity at strike (kink). Payoffs like $\max(S - K, 0)$ are non-smooth at $S = K$. This reduces local accuracy. Remedies include grid refinement around the strike, coordinate transforms (log-price), or Rannacher smoothing (take a few fully implicit Euler steps initially before CN to dampen oscillations).

Log-price transform (optional). A common change of variables $x = \ln S$ transforms the PDE into one with constant diffusion coefficient, which can simplify discretization and allow a more uniform grid in x. After transform, adjust boundary and terminal conditions accordingly.

Comparison with Monte Carlo. Use the analytical Black–Scholes formula as a reference for plain European options. Compare CN deterministic errors (difference to exact formula) and MC statistical errors (confidence intervals). Generate work–precision plots (error vs CPU time) to compare efficiency.

Stability. CN is unconditionally stable for linear parabolic PDEs; however, accuracy still requires reasonable Δt relative to ΔS. When implementing coordinate transforms or non-uniform grids, re-check discretization consistency.

6. Tasks

You are asked to implement both Monte Carlo and Crank–Nicolson methods for European call options and perform a comparative study.

1. Constants table (suggested values):

Parameter	Value
Initial asset price S_0	100
Strike K	100
Time to maturity T	1 year
Volatility σ	0.2, 0.4
Risk-free rate r	0.01, 0.05
Number of time steps N	100
Number of Monte Carlo paths M	50,000

2. Experiments:

- Compute option prices for all combinations of σ and r.
- Compare Monte Carlo estimated prices with Crank–Nicolson numerical PDE prices.
- Measure CPU time and compare efficiency of Monte Carlo vs CN.
- Plot option price vs volatility, option price vs interest rate.
- Optionally, explore variance reduction techniques in Monte Carlo.

3. Deliverables:

- Implementations of Monte Carlo and Crank–Nicolson solvers.
- Table of results for varying σ and r.
- Analysis comparing accuracy and efficiency of the two methods.

Website Queueing System: Simplified Model

1. Motivation

High-traffic websites (ticketing, auctions, flash sales) often implement queues to prevent system overload. Modelling these queues helps answer:

- How many servers are needed to keep waiting times reasonable?
- What fraction of users leave without being served?
- How does queue size affect blocking and throughput?

2. Basic Queueing Concepts

We use these parameters:

- λ = arrival rate (users per unit time)
- μ = service rate per server
- c = number of servers
- K = maximum system capacity (including servers)

Traffic intensity: $\rho = \lambda/(c\mu)$.

Assume exponential service times (mean $1/\mu$) and exponential patience (mean $1/\theta$) if modelling abandonment.

3. Simple M/M/c and M/M/c/K Models

3.1 M/M/c (infinite queue)

- Users arrive at rate λ, served by c servers with rate μ each.
- Probability all servers are busy (user waits) can be estimated using Erlang-C formula:

$$P_{\text{wait}} \approx \frac{\frac{(\lambda/\mu)^c}{c!}\frac{c\mu}{c\mu-\lambda}}{\sum_{n=0}^{c-1}\frac{(\lambda/\mu)^n}{n!} + \frac{(\lambda/\mu)^c}{c!}\frac{c\mu}{c\mu-\lambda}}.$$

- Expected waiting time in queue: $W_q = \frac{P_{\text{wait}}}{c\mu-\lambda}$.

3.2 M/M/c/K (finite capacity)

- Maximum K users allowed in system. If full, new arrivals are blocked.
- Blocking probability (probability an arrival is rejected) can be computed numerically from simple formulas:

$$P_{\text{block}} = \frac{a^K/(c!c^{K-c})}{\sum_{n=0}^{c}\frac{a^n}{n!} + \sum_{n=c+1}^{K}\frac{a^n}{c!c^{n-c}}}, \quad a = \frac{\lambda}{\mu}.$$

- This gives an idea of how many servers or queue slots are needed.

4. Simple Discrete-Event Simulation (DES)

You can simulate the system:

- Keep track of current time, number of users in system, and queue.
- Events: `ARRIVAL` (new user), `DEPARTURE` (service completed), `ABANDONMENT` (optional, if modelling patience).
- Steps:
  1. Initialize time and servers.
  2. Schedule first arrival.
  3. While simulation not finished:
     - Pop next event (earliest time).
     - Update system state.
     - Schedule next event if applicable.

This allows observing waiting times, fraction of users served, blocked, or abandoned.

5. Suggested Experiments

- Compare average waiting time W_q and blocking probability for different numbers of servers c and system capacities K.
- Introduce a simple peak: double λ for a short time, measure transient waiting and abandonment.
- Optional: vary user patience $1/\theta$ and see effect on abandonment.

6. Example Parameters

Parameter	Suggested Value
Arrival rate λ	10 users/min (normal), 20 users/min (peak)
Service rate μ	5 users/min per server
Number of servers c	2–5
System capacity K	5–10 (if finite)
Mean patience $1/\theta$	2–5 min

7. Student Tasks

1. Implement a simple M/M/c/K simulation.
2. Measure average waiting time, fraction of blocked users, and fraction of abandoned users.
3. Compare results when changing:
   - Number of servers c
   - Maximum capacity K
   - Arrival rate λ (simulate peak traffic)
   - Mean patience $1/\theta$ (if modelling abandonment)
4. Optional: Compare DES results with M/M/c steady-state formulas to validate.

Option 2: Replicating the Design and Implementation of a Published Paper

Background and Motivation

Replication is a core aspect of scientific research. By attempting to reproduce the results of a published study, you gain a deeper understanding of modelling assumptions, computational methods, and analysis techniques. This option allows you to connect course concepts to real-world research and critically evaluate methodological choices.

Project Task

You will:

- Select a published paper related to stochastic modelling, Monte Carlo and/or Monte Carlo simulation or other topics covered in the course.
- Carefully review the paper, focusing on the problem statement, assumptions, model structure, and computational approach.
- Implement the methods or experiments described in the paper using a programming language of their choice.
- Compare your results with the published results and analyse any discrepancies.
- Document challenges encountered, adaptations made, and lessons learned from the replication process.

Learning Objectives

- Understand the translation of theoretical models into practical computational methods.
- Critically evaluate published results and methodologies.
- Develop practical skills in programming, numerical methods, and model validation.
- Gain experience in scientific documentation and reproducibility.

Option 3: Literature Review of 5–15 Papers

Background and Motivation

A literature review allows you to synthesize knowledge on a particular topic, identify trends, gaps, and open questions, and develop critical evaluation skills. This project option emphasizes scholarly analysis rather than implementation, and is ideal for you interested in exploring broader research directions.

Project Task

You will:

- Select a coherent topic or research question related to the course (e.g., stochastic simulations, Bayesian inference, neural network modelling, queueing systems).
- Search for and read 5–15 relevant academic papers, preprints, or technical reports.
- Summarize key methods, results, assumptions, and conclusions of each paper.
- Compare approaches, highlight similarities and differences, and identify gaps in the literature.
- Discuss potential future directions or applications suggested by the literature.

Learning Objectives

- Develop skills in critical reading, synthesis, and scholarly writing.
- Identify research trends and gaps within a focused area.
- Learn to communicate complex ideas clearly in written and oral form.
- Gain familiarity with academic databases, citation practices, and literature evaluation techniques.