# CS726: Programming Assignment 4

April 4, 2025

## General Instructions

1. Plagiarism will be strictly penalized including but not limited to reporting to DADAC and zero in assignments. If you use tools such as ChatGPT, Copilot, you must explicitly acknowledge their usage in your report. If you use external sources (e.g., tutorials, papers, or open-source code), you must cite them properly in your report and comments in the code.

2. Submission: Submit a report explaining your approach, implementation details, and results. Clearly mention the contributions of each team member in the report. Submit your code and report as a compressed `<TeamName>_<student1rollno>_<student2rollno>_<student3rollno>.zip` file. Fill a student roll number as `NOPE` if less than 3 members.

3. Start well ahead of the deadline. Submissions up to two days late will be capped at 80% of the total marks, and no marks will be awarded beyond that.

4. Do not modify the environment provided. Any runtime errors during evaluations will result in zero marks. `README.md` provides instructions and tips to set up the environment and run the code.

5. For most of the assignment, you have to fill in your code in already existing files. Apart from the report, do not submit any additional models and files unless explicitly asked. Any additional files should be placed in the top-level directory.

```
cs726_assgmt4/
    +- TASK-0-1/
        +- get_results.py
        +- sampling_algos.py
    +- Task-02
        +-gaussianEstimate.py
    +- report.pdf [NEW]
```

6. STRICTLY FOLLOW THE SUBMISSION GUIDELINES. Any deviation from these guidelines will result in penalties.

   **Github link:** https://github.com/Nikita12200/CS726-Assignment-04
   **Link to drive:** This folder has Trained Regressor model and test data for TASK 0

# 1 Introduction

Energy-Based Models (EBMs) provide a flexible framework for representing complex probability distributions. Instead of defining a normalized probability density directly, an EBM defines an energy function $E_\theta(x)$, often parameterized by a neural network with parameters $\theta$. The probability distribution is then implicitly defined as:

$$p_\theta(x) = \frac{\exp(-E_\theta(x))}{Z_\theta}$$

where $Z_\theta = \int \exp(-E_\theta(x'))dx'$ is the partition function, which is typically intractable to compute.

While training EBMs presents challenges (often addressed by contrastive methods), sampling from a *trained* EBM, $p(x) \propto \exp(-E(x))$, is also non-trivial. Markov Chain Monte Carlo (MCMC) methods are commonly employed for this task. This assignment focuses on using Langevin dynamics-based MCMC algorithms to draw samples from a probability distribution defined by a pre-trained neural network energy function.

# 2 Provided Materials

You will be provided with the following files:

- `TASK-0-1/get_results.py`: A Python script defining the neural network architecture used as the energy function. And a Tester class.
- `TASK-0-1/sampling_algos.py`: A file where you are supposed to implement samping algorithms as explained in Task 1.

Assume the input $X$ to the network is a flattened vector (e.g., $X \in \mathbb{R}^{784}$ for flattened MNIST images).

# 3 Tasks

## 3.1 Task 0: Environment Setup and Result Reproduction

1. Set up a Python environment with the necessary libraries (NOTE: You can not use any libraries apart from below mentioned libraries:
   (a) PyTorch
   (b) Numpy
   (c) Matplotlib
2. Load the EnergyRegressor model in `get_results.py`.
3. Load the pre-trained weights from `trained_model_weights.pth` into the network instance.
4. Run the provided `get_results.py` script.
5. In your report, include the output generated by `get_results.py`.

## 3.2 Task 1: MCMC Sampling Implementation

For this task, consider the loaded, pre-trained neural network $NN(X)$ as the energy function $E(X) = NN(X)$. The target probability distribution is $p(X) \propto \exp(-E(X))$. You will implement three MCMC algorithms to sample from $p(X)$.

1. **Implement the Algorithms:**

**Algorithm 1**

Initialize $X_0$
**for** $t = 0$ to $N-1$ **do**
  Compute gradient $g_t = \nabla_X E(X_t)$
  Sample noise $\xi_t \sim \mathcal{N}(0, I)$
  Propose $X' = X_t - \frac{\tau}{2} g_t + \sqrt{\tau}\xi_t$
  Compute gradient at proposal $g' = \nabla_X E(X')$
  Compute acceptance probability:

$$\log q(X|X') = -\frac{1}{4\tau}\|X - (X' - \frac{\tau}{2}g')\|^2$$

$$\log q(X'|X_t) = -\frac{1}{4\tau}\|X' - (X_t - \frac{\tau}{2}g_t)\|^2$$

$$\alpha = \min\left(1, \exp\left(E(X_t) - E(X') + \log q(X_t|X') - \log q(X'|X_t)\right)\right)$$

  Sample $u \sim \text{Uniform}(0, 1)$
  **if** $u < \alpha$ **then**
    $X_{t+1} \leftarrow X'$
  **else**
    $X_{t+1} \leftarrow X_t$
  **end if**
**end for**
**return** samples $\{X_t\}_{t>\text{burn-in}}$

**Algorithm 2**

Initialize $X_0$
**for** $t = 0$ to $N-1$ **do**
  Compute gradient $g_t = \nabla_X E(X_t)$
  Sample noise $\xi_t \sim \mathcal{N}(0, I)$
  Update $X_{t+1} = X_t - \frac{\tau}{2} g_t + \sqrt{\tau}\xi_t$
**end for**
**return** samples $\{X_t\}_{t>\text{burn-in}}$

2. **Generate Samples:** Run your implementations to generate a chain of samples for each algorithm.

3. Report sampling Time: Time taken to finish burn-in for each algorithm.

4. **Visualization / Qualitative Results:** Use t-SNE or similar method to plot this high dimensional samples in 2d or 3d space to visialize the distribution for both methods.

## 3.3 Task 2: Approximating a Black-Box Function Using Gaussian Processes

The objective of this task is to approximate a given black-box function using Gaussian Processes (GPs) without relying on machine learning libraries like `scikit-learn`. Instead, you will implement the Gaussian Process regression algorithm manually using only `NumPy` for numerical computations and `Matplotlib` for visualization. You will experiment with different kernels, vary the number of training samples, and analyze how these choices affect the GP's ability to model the function. Additionally, you will implement and compare acquisition functions to guide the selection of new training points, enhancing the GP's approximation through active learning. This exercise will provide insights into the flexibility, limitations, and practical implementation challenges of Gaussian Processes in regression tasks.

**Black-Box Function: The Branin-Hoo Function**

The black-box function to approximate is the Branin-Hoo function, defined as:

$$f(x_1, x_2) = a\left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10, \tag{1}$$

where $a = 1$, and the input domain is:

$$x_1 \in [-5, 10], \quad x_2 \in [0, 15].$$

Your goal is to approximate this function using a manually implemented Gaussian Process regression algorithm.

Complete the file `gaussianEstimate.py` as follows:

**Task Breakdown**

- **Implement the Branin-Hoo Function:** Write a Python function to compute the Branin-Hoo function, which will serve as the black-box function.
- **Generate Training Data:** Sample $n_{\text{samples}}$ points uniformly from the input space, experimenting with the following values: $n_{\text{samples}} = 10, 20, 50, 100$.
- **Implement Gaussian Process Regression Manually:**
  - Use `NumPy` to implement the Gaussian Process regression algorithm from scratch, including the computation of the posterior mean and variance.
  - Implement the following kernel types manually:
    * **Radial Basis Function (RBF) Kernel:** $k(x, x') = \sigma_f^2 \exp\left(-\frac{\|x-x'\|^2}{2\ell^2}\right)$, where $\ell$ is the length scale and $\sigma_f$ is the signal variance.
    * **Matérn Kernel ($\nu = 1.5$):**

      $$k(x, x') = \sigma_f^2 \left(1 + \frac{\sqrt{3}\|x - x'\|}{\ell}\right) \exp\left(-\frac{\sqrt{3}\|x - x'\|}{\ell}\right),$$

      where $\ell$ is the length scale and $\sigma_f$ is the signal variance.
    * **Rational Quadratic Kernel:**

      $$k(x, x') = \sigma_f^2 \left(1 + \frac{\|x - x'\|^2}{2\alpha\ell^2}\right)^{-\alpha},$$

      where $\ell$ is the length scale, $\sigma_f$ is the signal variance, and $\alpha$ is the scale mixture parameter (set $\alpha = 1$).
  - Optimize the kernel hyperparameters ($\ell$, $\sigma_f$, and noise variance $\sigma_n^2$) by maximizing the log-marginal likelihood using a grid search approach (optional for better predictions).
- **Implement Acquisition Functions:**
  - Implement two acquisition function heuristics to select the next point to evaluate:
    * **Expected Improvement (EI):** $\text{EI}(x) = (\mu(x) - f(x_{\text{best}}) - \xi)\Phi(z) + \sigma(x)\phi(z)$, where $z = \frac{\mu(x) - f(x_{\text{best}}) - \xi}{\sigma(x)}$, $\Phi$ is the cumulative distribution function, and $\phi$ is the probability density function of the standard normal distribution, with $\xi = 0.01$.
    * **Probability of Improvement (PI):** $\text{PI}(x) = \Phi\left(\frac{\mu(x) - f(x_{\text{best}}) - \xi}{\sigma(x)}\right)$, with $\xi = 0.01$.
  - Compare these with a random acquisition strategy, where the next point is chosen randomly from the input domain.
  - After selecting a new point using each acquisition function, update the training set and refit the GP.

- **Visualize and Compare Results:** Use `Matplotlib` to generate plots comparing the GP's predicted mean and uncertainty (standard deviation) against the true Branin-Hoo function for each kernel, sample size, and acquisition strategy. Create filled contour plots with a colorbar for each visualization.
- **Analyze Model Performance:** Examine how the choice of kernel, number of training samples, and acquisition functions influence the GP's approximation ability. Discuss the trade-offs between different kernels, the impact of sample size on prediction accuracy and uncertainty, and the effectiveness of each acquisition strategy.

**Submission Requirements**

Submit the following materials:

1. Python code (`gaussianEstimate.py`) implementing all tasks, including the manual GP implementation, kernel functions, acquisition functions, hyperparameter optimization (if implemented), and plotting.
2. Plots for each combination of kernel (RBF, Matérn, Rational Quadratic), sample size ($n_{\text{samples}} = 10, 20, 50, 100$), and acquisition strategy (Random, EI, PI), showing:
   (a) The true Branin-Hoo function.
   (b) The GP predicted mean.
   (c) The GP predicted uncertainty (standard deviation).
3. A brief report analyzing the results, including:
   - A comparison of the performance of different kernels.
   - The effect of varying the number of training samples on the quality of the approximation.
   - Observations about the uncertainty estimates and their reliability.
   - A comparison of the effectiveness of the acquisition functions (Random, EI, PI) in improving the GP's approximation.