Assignment 4 Report, CS 726: Spring 2024-25

Anupam Rawat
IIT Bombay
22b3982@iitb.ac.in

Aryan Gupta
IIT Bombay
22b2255@iitb.ac.in

Satush Parikh
IIT Bombay
21d070062@iitb.ac.in

April 17, 2025

Contents

1 Task 0: Environment Setup and Result Reproduction

1.1 Results and Conclusion

The shape of the input is $x \in \mathbb{R}^{100000\times784}$ and the dimension of the output vector is energy $\in \mathbb{R}^{100000\times1}$
The reported loss is 288.2319

2 Task 1: MCMC Sampling Implementation

2.1 Introduction

The task requires implementation of Markov Chain Monte Carlo (MCMC) algorithms for sampling from a probability distribution defined by a pre-trained neural network. The neural network acts as an energy function E(X) = NN(X). Now, the probability distribution is given by p(X) $\propto$ exp(-E(X)). The goal is to generate samples that follow this distribution using MCMC-based methods.

Markov Chain Monte Carlo (MCMC) methods are a class of algorithms used to generate samples from complex, high-dimensional probability distributions where direct sampling is infeasible. The core idea is to construct a Markov chain whose stationary distribution matches the desired target distribution (p(X)). By simulating this chain over time, the samples it generates will approximate the target distribution. Common techniques include the Metropolis-Hastings algorithm, Gibbs sampling, and Langevin dynamics. These methods are widely used in Bayesian inference, statistical physics, and generative modeling.

2.2 Algorithms

Algorithm 1 Metropolis-Adjusted Langevin Algorithm (MALA)

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:
    $\log q(X_t|X') = -\frac{1}{4\tau}\left\|X_t - (X' - \frac{\tau}{2}g')\right\|^2$
    $\log q(X'|X_t) = -\frac{1}{4\tau}\left\|X' - (X_t - \frac{\tau}{2}g_t)\right\|^2$
    $\alpha = \min\left(1, \exp(E(X_t) - E(X') + \log q(X_t|X') - \log q(X'|X_t))\right)$
  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 $\{X_t\}_{t>\text{burn-in}}$ =0

Algorithm 2 Unadjusted Langevin Algorithm (ULA)

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 $\{X_t\}_{t>\text{burn-in}}$ =0

Algorithm 3 Hamiltonian Monte Carlo (HMC)

Input: Step size ϵ, number of leapfrog steps L, number of samples N
Initialize X_0
for $t = 0$ to $N - 1$ **do**
Sample momentum $p_t \sim \mathcal{N}(0, I)$
Set $x \leftarrow X_t$, $p \leftarrow p_t$
Compute initial energy $H_0 = E(x) + \frac{1}{2} p^\top p$
Leapfrog Integration:
    $p \leftarrow p - \frac{\epsilon}{2} \nabla_x E(x)$
  for $i = 1$ to L **do**
$x \leftarrow x + \epsilon p$
    if $i < L$ **then**
$p \leftarrow p - \epsilon \nabla_x E(x)$
    end if
  end for
$p \leftarrow p - \frac{\epsilon}{2} \nabla_x E(x)$
Negate momentum: $p \leftarrow -p$
Compute proposed energy $H_1 = E(x) + \frac{1}{2} p^\top p$
Compute acceptance probability $\alpha = \min(1, \exp(H_0 - H_1))$
Sample $u \sim \mathcal{U}(0, 1)$
  if $u < \alpha$ **then**
$X_{t+1} \leftarrow x$
  else
$X_{t+1} \leftarrow X_t$
  end if
end for
Return: Samples $\{X_t\}_{t>\text{burn-in}}$ =0

The **Metropolis Adjusted Langevin Algorithm (MALA)** is a Markov Chain Monte Carlo (MCMC) method that combines gradient-based updates with a Metropolis-Hastings correction. It proposes new samples using **Langevin dynamics**, where the next state is influenced by both the **gradient of the energy function** and **Gaussian noise**. The Metropolis-Hastings step ensures that the samples are drawn from the correct target distribution, making MALA asymptotically unbiased. While more computationally intensive than ULA due to the acceptance-rejection step, MALA generally produces better quality samples with reduced drift.

The **Unadjusted Langevin Algorithm (ULA)** simplifies MALA by removing the acceptance-rejection step. It updates samples using a **combination of gradient descent and Gaussian noise**, making it **computationally cheaper** and easier to implement. However, ULA does not correct for discretization errors, which can introduce **bias**, especially if the step size is not sufficiently small. This makes ULA suitable for **fast approximate sampling** when small biases are acceptable, but **less reliable** for tasks requiring high precision.

The **Hamiltonian Monte Carlo (HMC)** algorithm introduces **auxiliary momentum variables** and simulates **Hamiltonian dynamics** to explore the target distribution efficiently. Using the leapfrog integrator, it generates proposals that can traverse the sample space more effectively than random-walk methods, reducing autocorrelation in the samples. HMC includes an acceptance step based on conservation of Hamiltonian energy, ensuring **unbiased sampling**. Although computationally more expensive due to **multiple gradient evaluations per iteration**, HMC excels in high-dimensional settings and is often considered one of the most efficient MCMC methods.

2.3 Results and Conclusion

The samples are stored in a .npy file in the outputs folder. Since, the samples are in a very higher dimension, a t-SNE graph is plotted to visualize them.
Sampling time of the Algorithms

- Algo-1 Burn-in Time: 11.97s
- Algo-2 Burn-in Time: 6.33s
- Algo-3 Burn-in Time: 11.97s

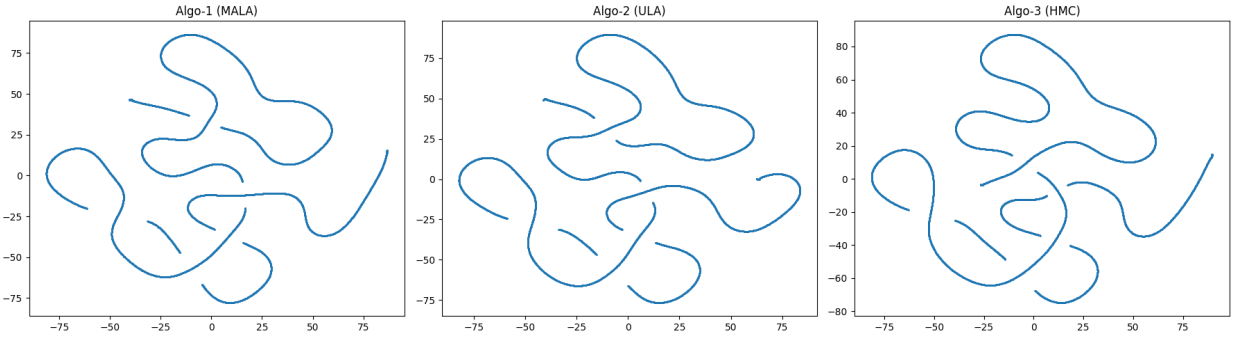


Figure 1: t-SNE visualization of the results of using two MCMC sampling algorithms

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

3.1 Introduction to Gaussian Processes & Black-Box Function

A Gaussian Process is a **non-parametric Bayesian approach** used for regression and probabilistic modeling. It defines a distribution over functions, where any finite collection of function values follows a multivariate Gaussian distribution.

A Gaussian Process helps approximate a black-box function by modeling it as a *probabilistic distribution over functions*—meaning it doesn't assume a fixed functional form, but instead provides a distribution of possible functions that fit the observed data.

The **Branin-Hoo function** is a **2D, non-convex** function with **multiple global minima**. It's smooth and continuous, which suits the assumptions of a typical GP using RBF or Matérn kernels. For the approximation task, we start with a few initial evaluations of the Branin-Hoo function at different 2D input locations. The GP is trained on this data and forms a posterior distribution over possible functions that fit these observations. Now the function can predict the black box function at unseen points. We can use acquisition functions to choose sampling points in areas of higher uncertainitiy to achieve an efficient optimization of the Black Box Function.

3.2 Kernel Functions

The following Kernel Functions were implemented to compute the covariance matrix between data points:

3.2.1 Radial Basis Function (RBF) Kernel

The RBF kernel is given by:

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

where σ_f is the signal variance, and ℓ is the length scale that controls the smoothness of the function. This kernel is widely used for its simplicity and its ability to represent smooth functions.

3.2.2 Matérn Kernel (with $\nu = 1.5$)

The Matérn kernel is given by:

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

This kernel has an additional parameter ν (in this case, set to 1.5), which controls the smoothness of the function. The Matérn kernel is useful for modeling functions with rougher behavior compared to the RBF kernel.

3.2.3 Rational Quadratic Kernel

The Rational Quadratic kernel is given by:

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

where $\alpha = 1$, σ_f is the signal variance, and ℓ is the length scale. This kernel can be interpreted as a scale mixture of RBF kernels and is useful for modeling functions that have varying degrees of smoothness over different scales.

3.3 Acquisition Strategies

3.3.1 Expected Improvement (EI)

Expected Improvement measures the expected amount of improvement over the current best observed value (e.g., the minimum or maximum). The acquisition function computes how much a new point would improve the objective function, averaged over the posterior distribution of the function.

$$\text{EI(x)} = (\mu(x) - y_{best} - \xi) \cdot \Phi(z) + \sigma(x) \cdot \phi(z)$$

where $\mu(x)$ represents the predicted mean at point x, $\sigma(x)$ represents the predicted standard deviation, y_{best} as the name suggests is the best observed value so far. $\Phi(z)$ represents the Cumulative Distribution Function of the standard normal distribution. $\phi(z)$ represents the probability distribution function of the standard normal distribution and

$$z = \frac{\mu(x) - y_{best} - \xi}{\sigma(x)}$$

EI is particularly useful when there is a need to balance exploration (testing uncertain areas) and exploitation (focusing on areas likely to yield the best results). It tends to prioritize regions where the model has high uncertainty and where the potential for improvement is large.

3.3.2 Probability of Improvement (PI)

Probability of Improvement measures the probability that a new candidate point will yield a better result than the best observed value so far. Unlike EI, which directly quantifies the improvement, PI only gives a probability estimate of improvement.

$$\text{PI(x)} = \Phi\left(\frac{\mu(x) - y_{best} - \xi}{\sigma(x)}\right)$$

where symbols have the same meanings as for EI.

PI is simpler and computationally cheaper than EI. It tends to be more aggressive in selecting regions where the model predicts high probability of improvement, which can lead to faster optimization, but with less exploration of uncertain regions compared to EI.

3.3.3 Random Acquisition

Random acquisition is the simplest approach, where new points are selected randomly from the search space without any consideration of the model's predictions. This strategy does not use any acquisition function.

Approximations in Code

To avoid dependence on the *scipy* library the below approximations are used:

$$\Phi(z) \approx \frac{1}{1 + exp(-1.702 \cdot z)}$$

$$\phi(z) \approx \frac{1.702 \cdot exp(-1.702 \cdot z)}{(1 + exp(-1.702 \cdot z))^2}$$

3.4 Results and Conclusion

Below are the plots of various kernels and acquisition functions:

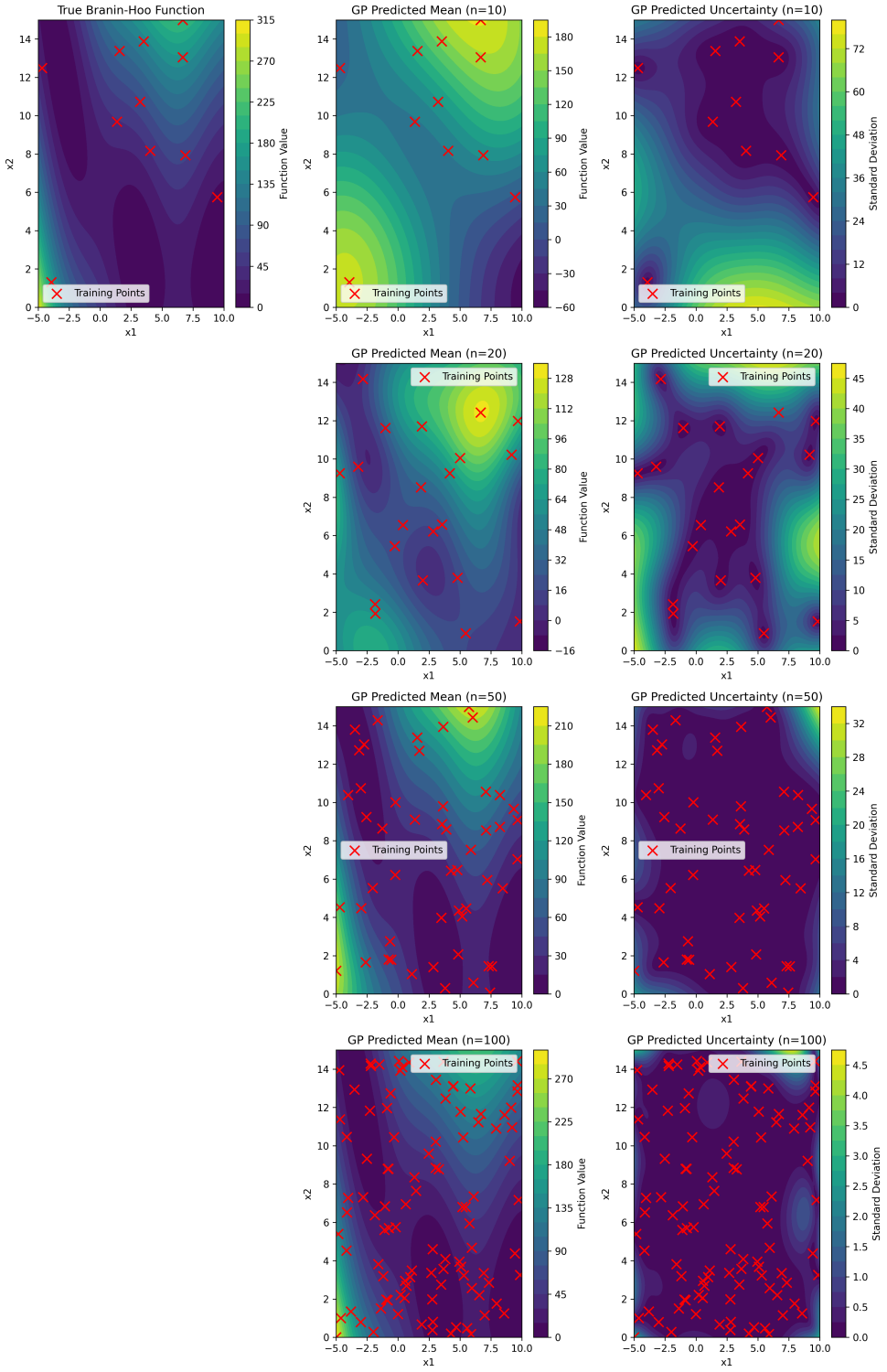


Figure 2: Progression of RBF Kernel with EI Acquisition

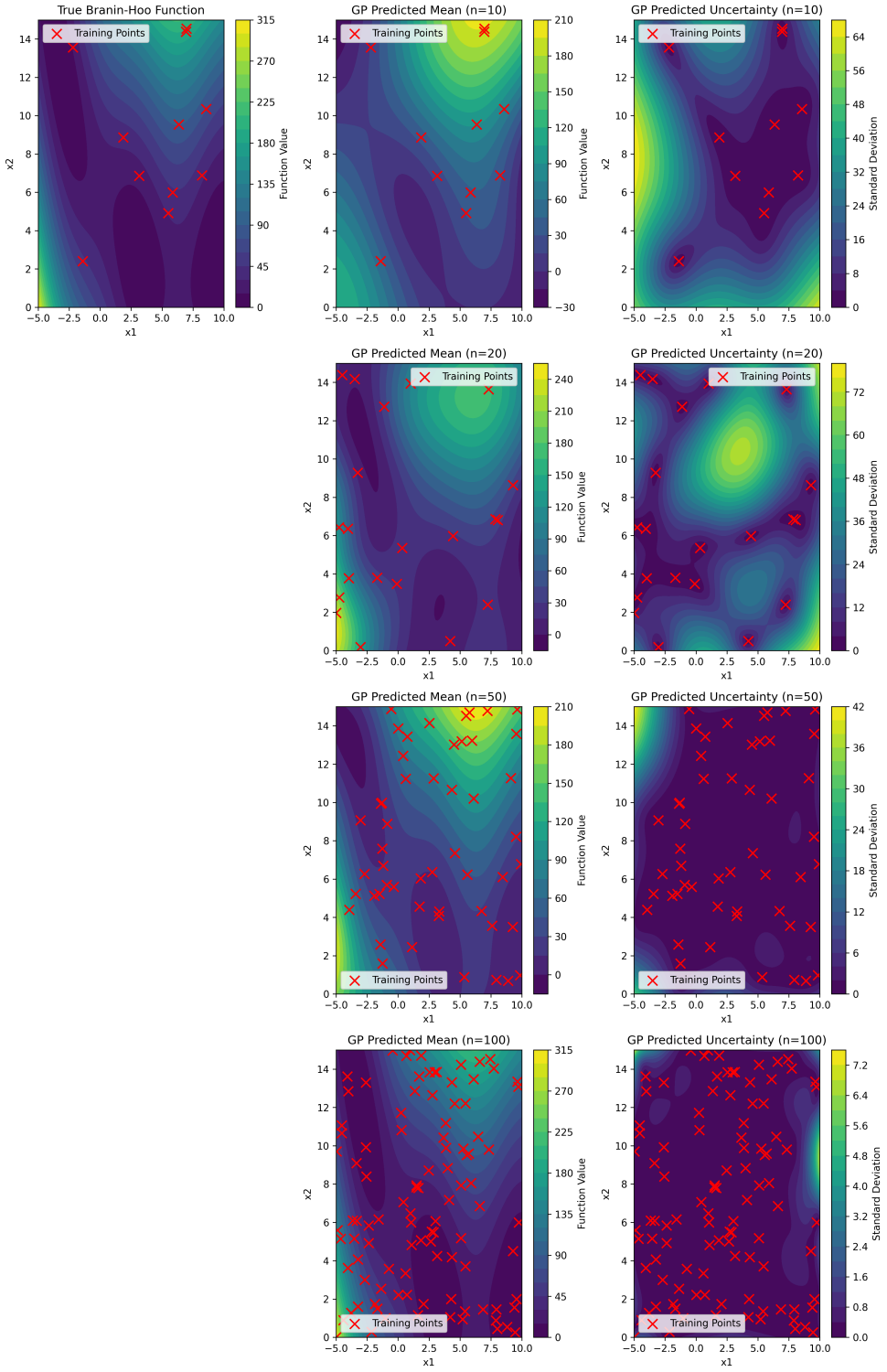


Figure 3: Progression of RBF Kernel with PI Acquisition

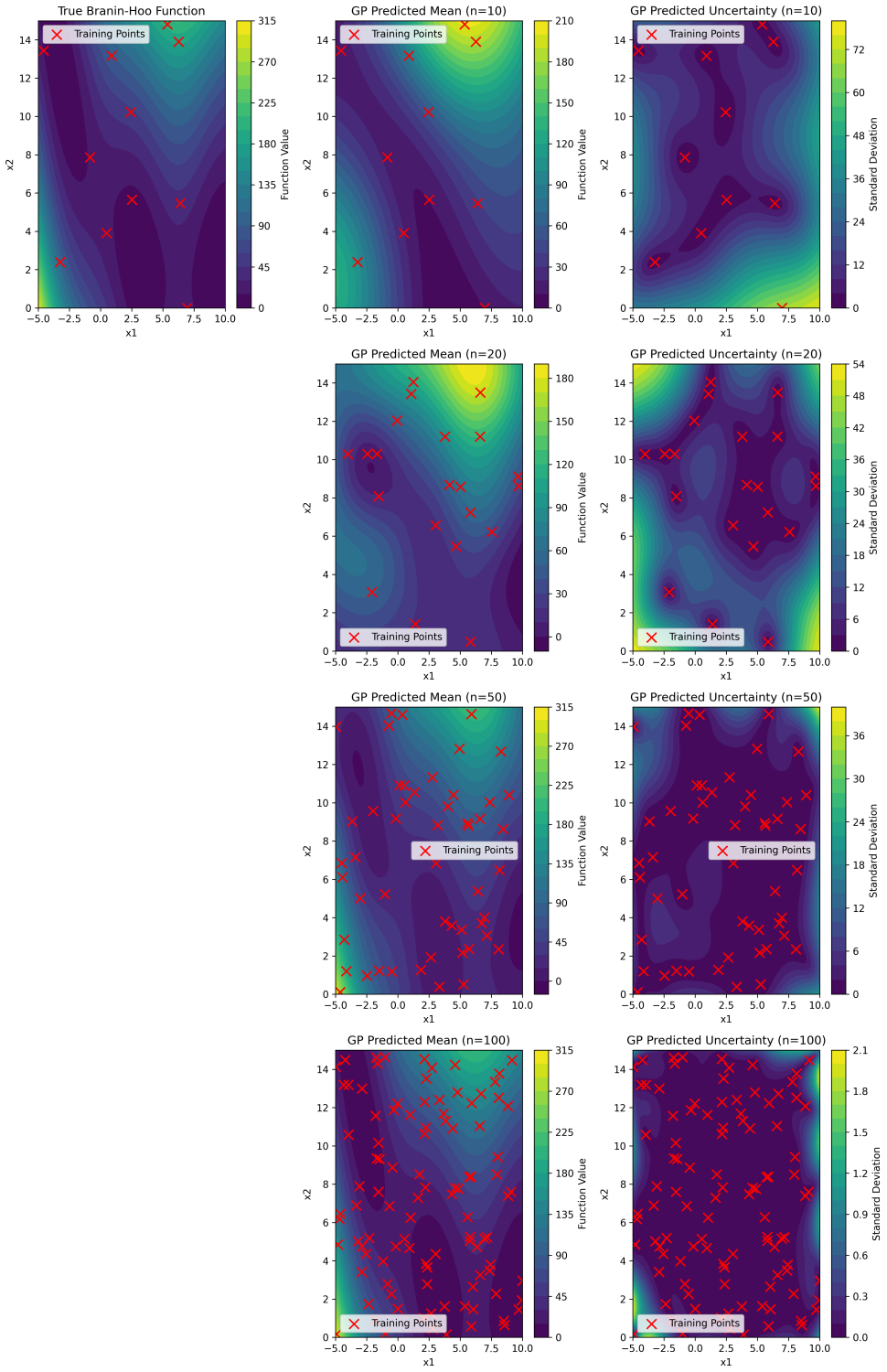


Figure 4: Progression of RBF Kernel with Random Acquisition

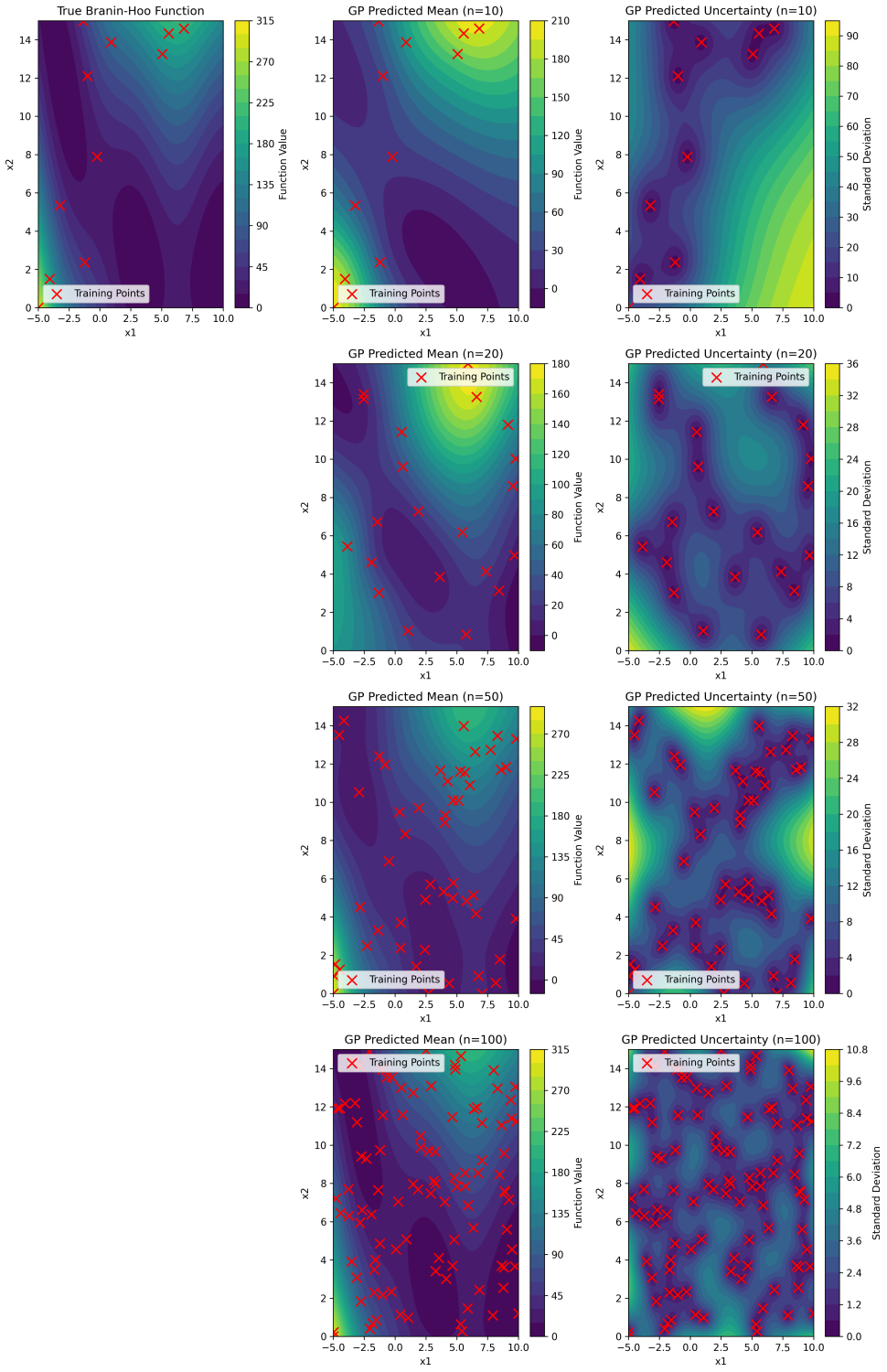


Figure 5: Progression of Matern Kernel with EI Acquisition

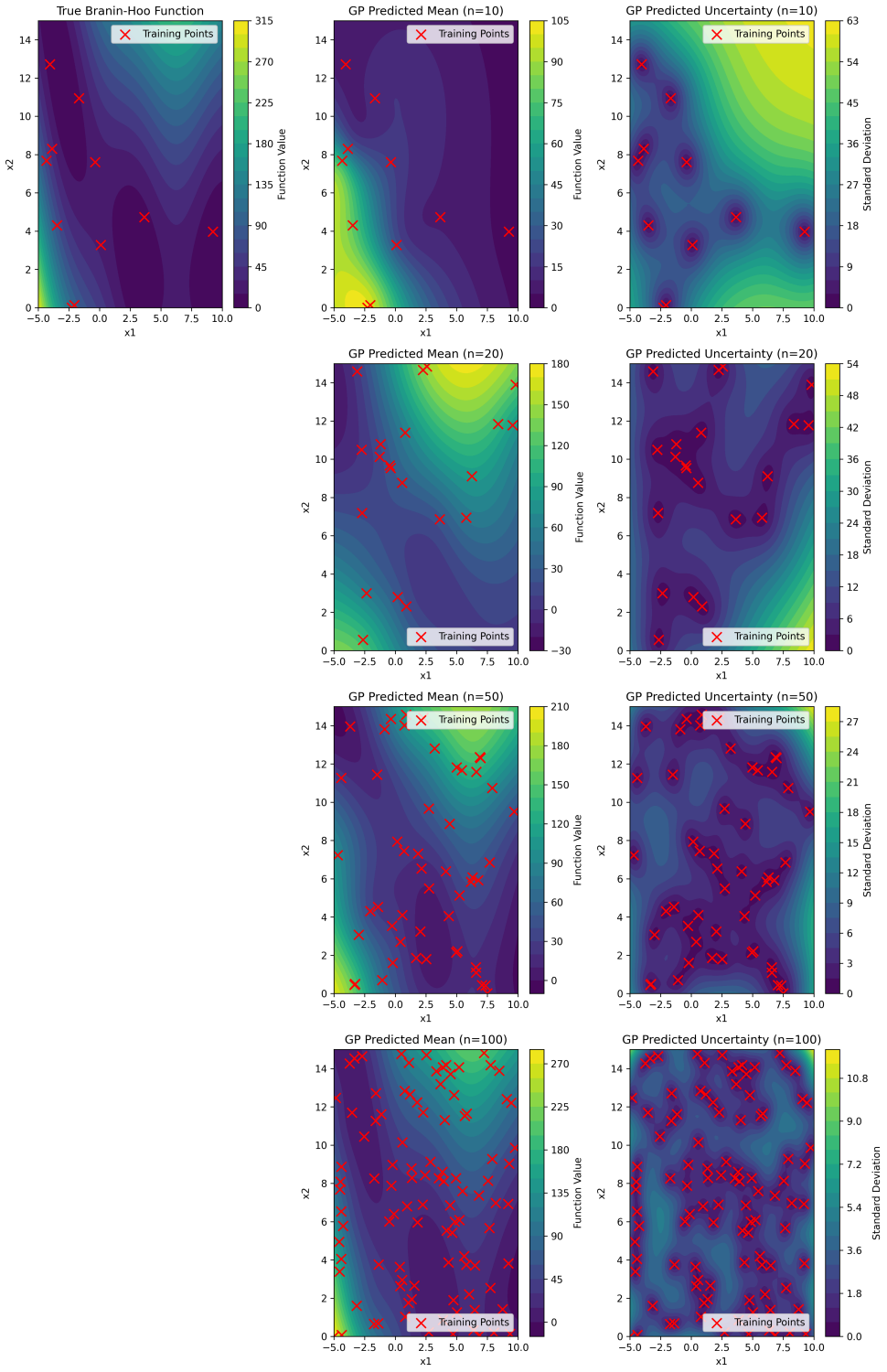


Figure 6: Progression of Matern Kernel with PI Acquisition

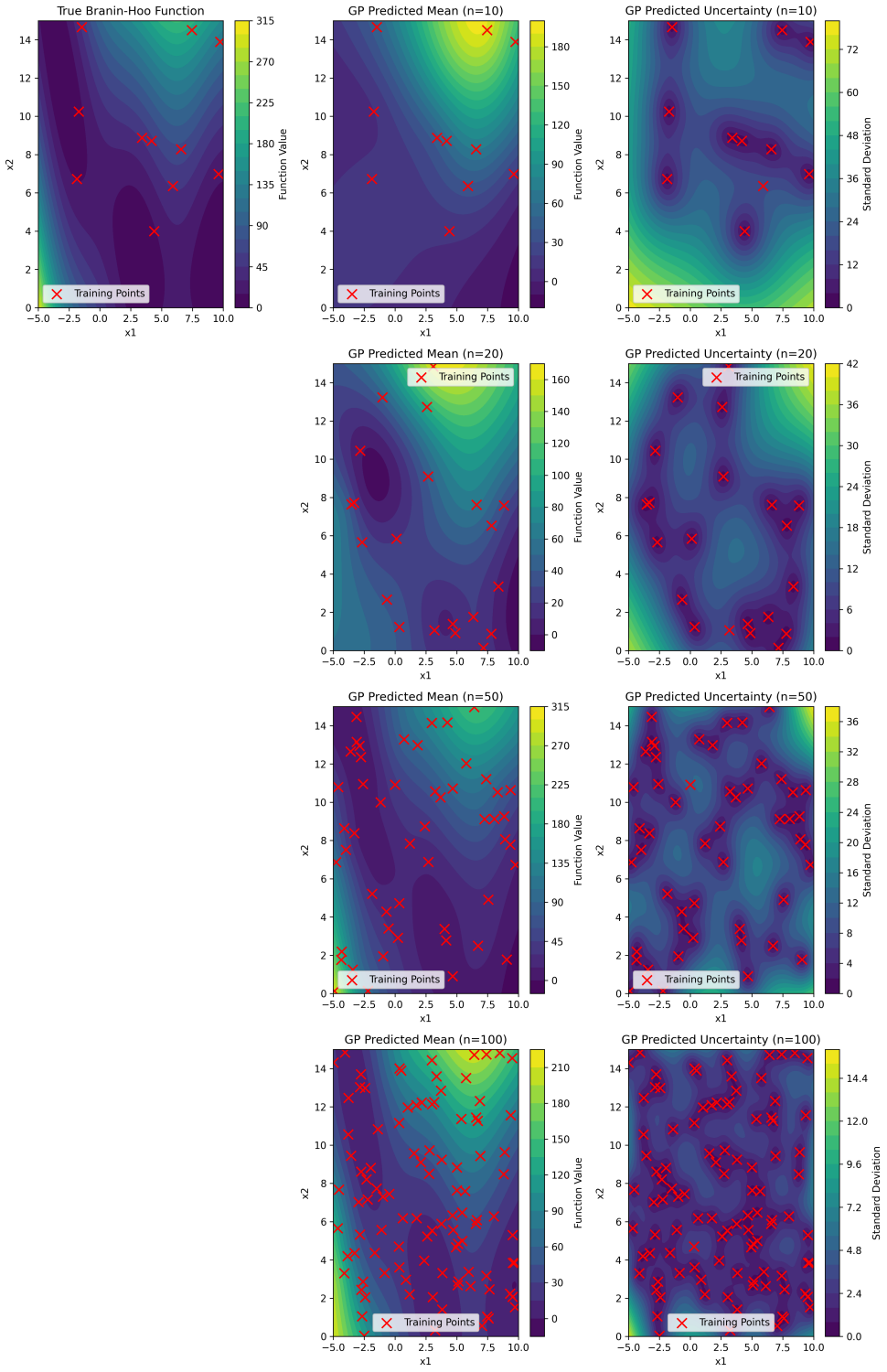


Figure 7: Progression of Matern Kernel with Random Acquisition

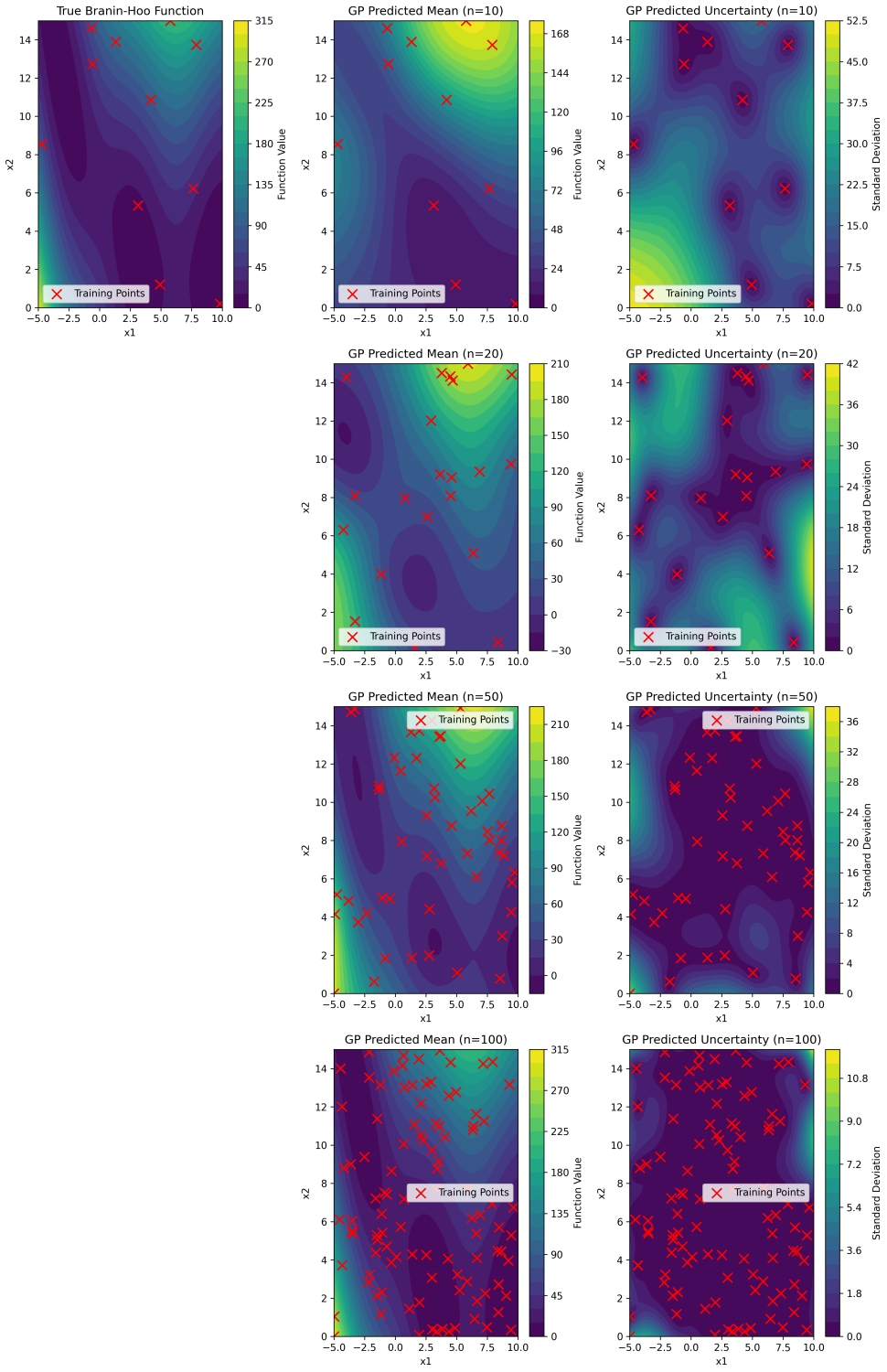


Figure 8: Progression of Rational Quadratic Kernel with EI Acquisition

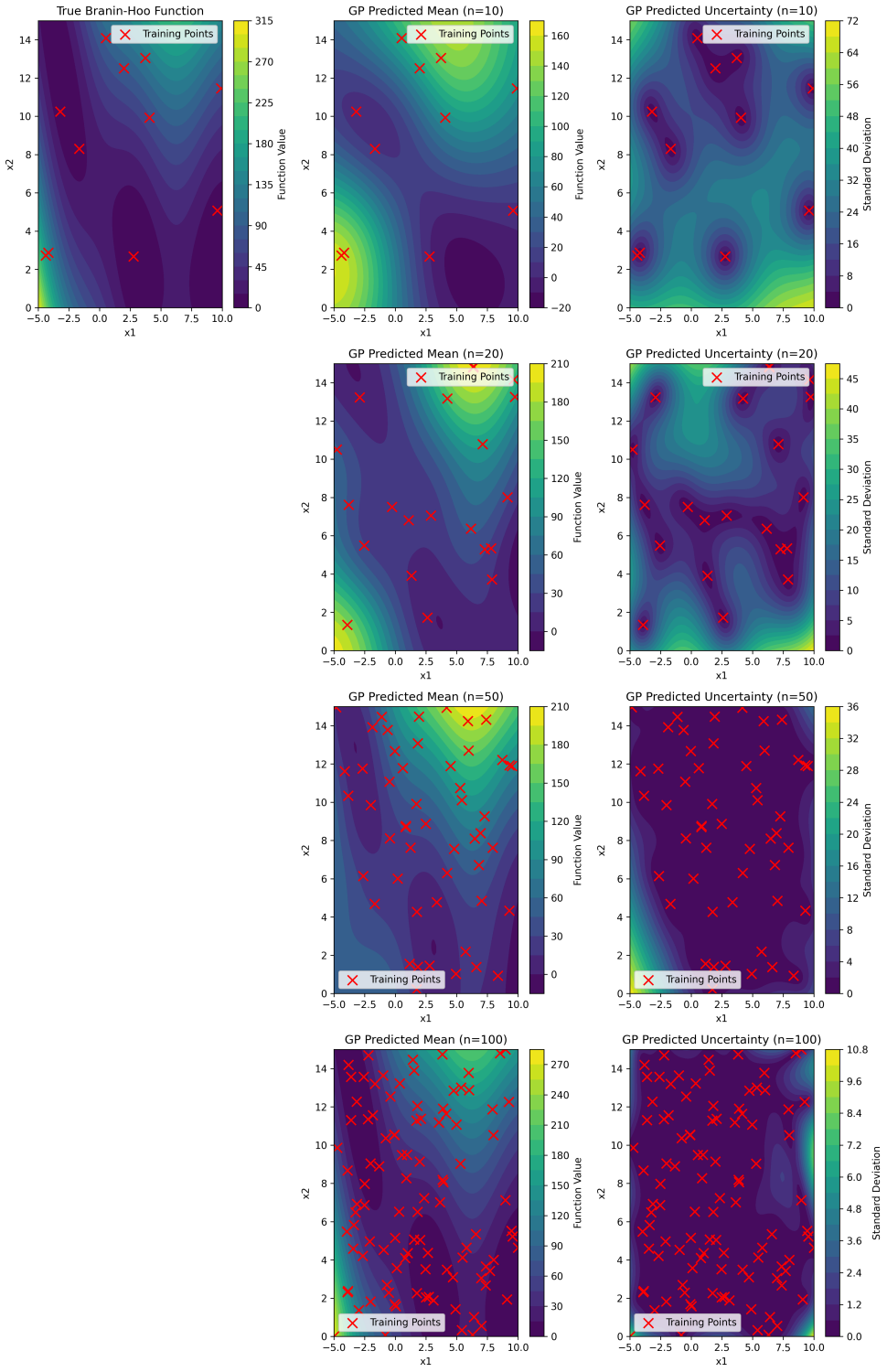


Figure 9: Progression of Rational Quadratic Kernel with PI Acquisition

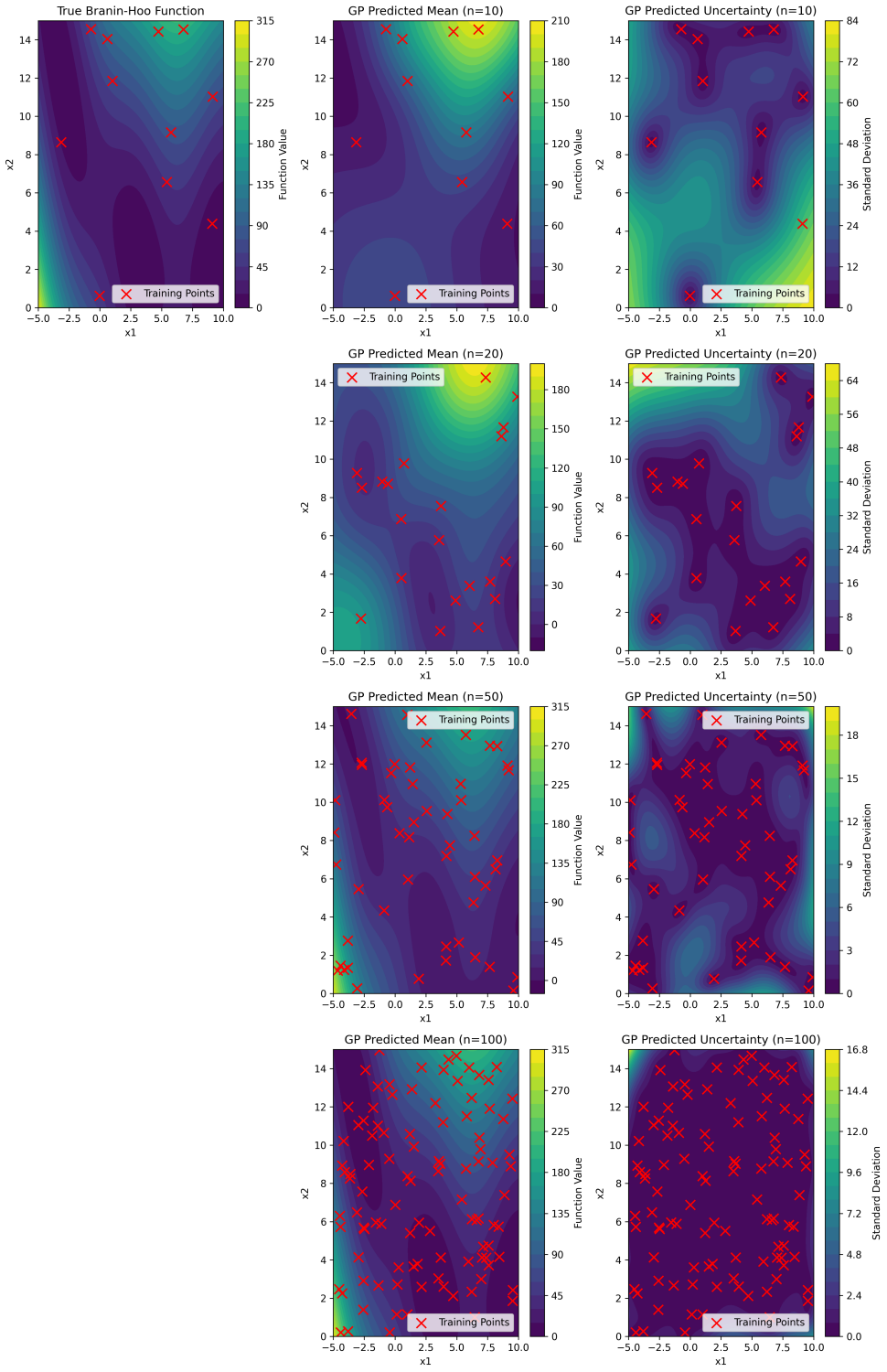


Figure 10: Progression of Rational Quadratic Kernel with Random Acquisition

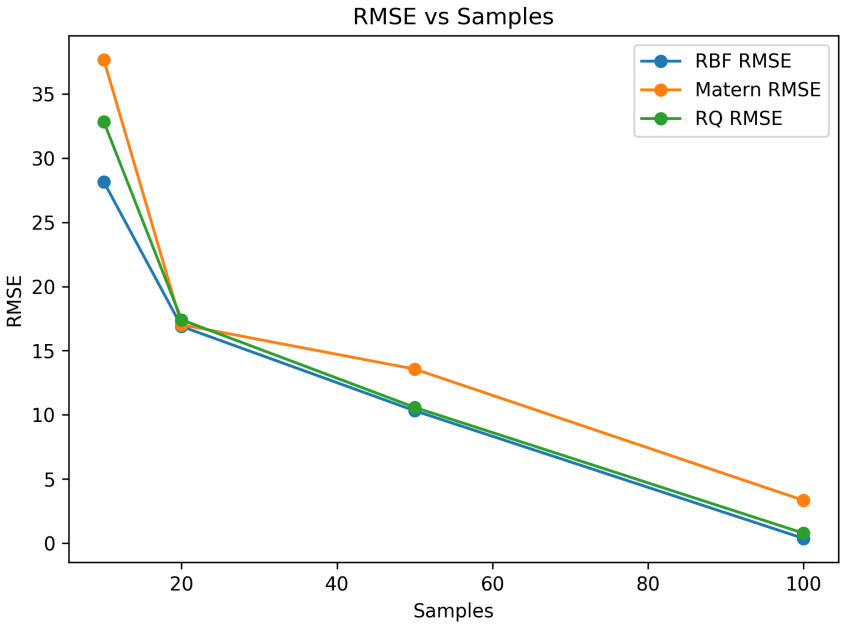
4 Performance Comparison

Table 1: Performance on Branin (various sample sizes)

Samples	RMSE			Avg. Variance		
	Matérn	RBF	RQ	Matérn	RBF	RQ
10	37.6787	28.1557	32.8413	873.655	746.486	390.078
20	17.0205	16.8878	17.4070	92.747	51.656	231.923
50	13.5645	10.3171	10.5780	38.575	24.141	41.298
100	3.3368	0.3704	0.7775	9.210	0.635	1.335

Key takeaways:

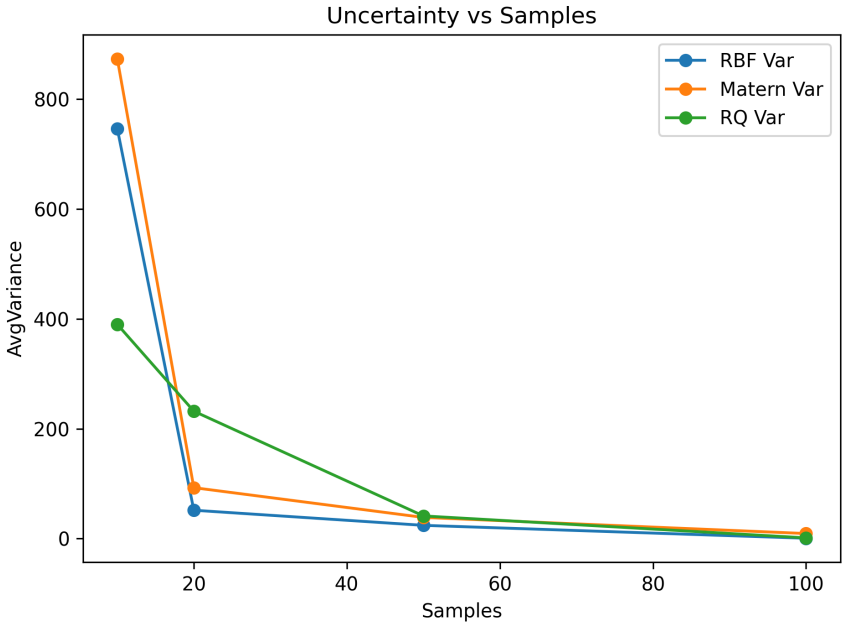
- **RBF** achieves the lowest RMSE at every sample size ≥ 20 and by 100 samples attains an RMSE of only 0.37 and an average variance of 0.64.
- **Rational Quadratic (RQ)** is competitive at small samples in uncertainty (390 vs. 746 for RBF at 10 samples) but its RMSE remains above RBF once $n \geq 10$. At 100 samples RQ has RMSE 0.78 and variance 1.33.
- **Matérn ($\nu = 1.5$)** underfits when data are scarce (RMSE 37.7, Var 873 at $n = 10$) and retains substantially higher uncertainty across all n (e.g. Var 9.21 at $n = 100$).



4.1 Effect of Sample Size

Prediction Accuracy

- All three kernels see a large drop in RMSE from 10 to 20 samples (e.g. RBF: 28.16 to 16.89).
- Beyond 50 samples, RBF and RQ continue to improve sharply, whereas Matérn plateaus until high n, only reaching RMSE 3.34 at $n = 100$.
- Diminishing returns set in after 50–100 samples, especially for Matérn.



Uncertainty Estimates

- RBF becomes extremely confident by $n = 100$ (Var 0.64), while RQ and especially Matérn remain more conservative.
- The bulk of confidence gains occur by $n = 50$, after which uncertainty converges.

4.2 Reliability of Uncertainty

Predictive variance trends mirror RMSE reductions:

- Matérn honestly reports huge uncertainty under data scarcity.
- RBF and RQ display overconfidence at $n = 10$ (variance $\ll$ RMSE) but achieve good calibration by $n \gtrsim 50$.

4.3 Acquisition Function Effectiveness

Aspect	Expected Improvement (EI)	Probability of Improvement (PI)
Exploration	Balances exploration and exploitation well	Tends to exploit unless exploration term (ξ) is tuned
Sensitivity to $\sigma(x)$	Encourages sampling uncertain regions	Can ignore high uncertainty if probability of improvement is low
Stability	More robust in practice	Can prematurely converge (greedy behavior)
Computational Cost	Slightly higher (due to EI formula)	Lower
Effectiveness	Generally outperforms PI in noisy, multimodal problems	Works well in low-noise or convex-like settings
Hyperparameter Tuning	More forgiving to choice of ξ	Very sensitive; small changes in ξ affect results

Table 2: Comparison between Expected Improvement (EI) and Probability of Improvement (PI) acquisition functions in Bayesian Optimization

4.4 Trade-Offs & Recommendations

Kernel Choice

- RBF: best overall accuracy and tightest confidence at moderate-to-large n.
- RQ: good middle ground in uncertainty at small n, though RMSE trails RBF.
- Matérn: useful if you need conservative variance estimates, but expect higher error when samples are few.

Sample Budget

- **10** $\rightarrow$ **20** samples yield the largest RMSE uncertainty reductions.
- **50** samples deliver most of the benefit; **100** samples fine-tune both RMSE and variance.

Acquisition Strategy

- EI for reliable early gains.
- PI to squeeze out the best final optimum.
- Avoid pure random once you have an approximate region.

5 AI Contribution

In the development of this project, **ChatGPT** & **Deepseek** was utilized for occasional debugging and assistance in the implementation of various functions. The LLMs contributed by providing initial drafts for these functions, which were then reviewed, debugged, and refined to align with the problem requirements.

6 Participant Contributions

The project involved multiple stages of development, debugging, and report preparation. The following contributions were made by the participants:

6.1 Aryan's Contributions

- Implemented and debugged the code for :
  - Task-0
  - Task-1

6.2 Anupam's Contributions

- Implemented and debugged the code for :
  - HMC Sampling Algorithm in Task-1
- Made the Report for the project.

6.3 Satush's Contributions

- Implementing, Debugging and refining the implementations of:
  - Task 2 : Using Gaussian Processes to approximate Black Box Functions

The combination of AI-assisted function generation and participant debugging ensured that the project met its intended objectives.