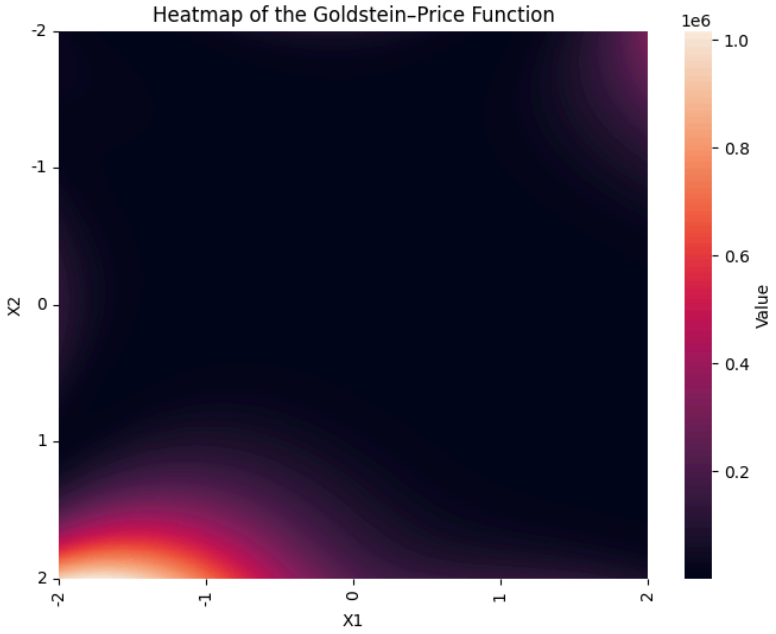
CSE 515 Final Project Report

Authors: Zhoujun Cao, Yiding Chen
Washington University in St. Louis
1 Brookings Dr, St. Louis, MO 63130
c.zhoujun@wustl.edu , c.yiding@wustl.edu

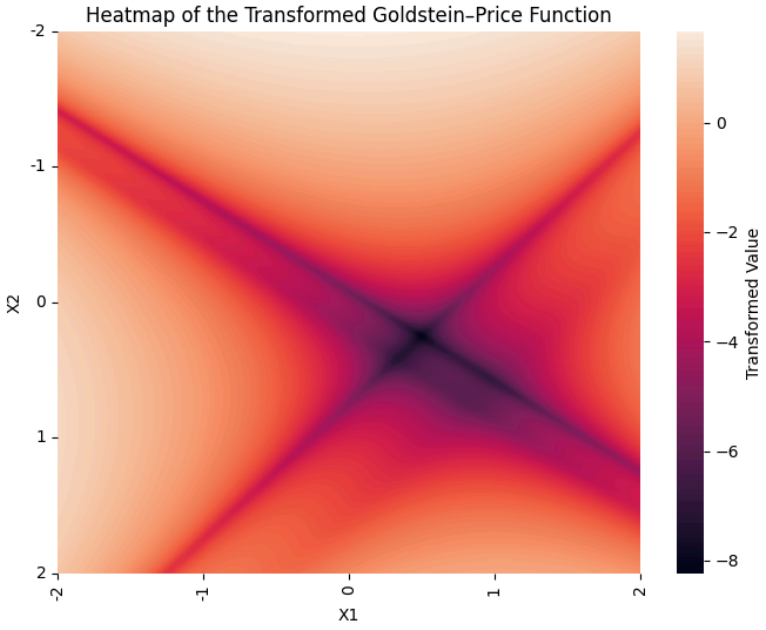
Data visualization

- Make a heatmap of the value of the Goldstein–Price function over the domain X = [−2, 2] × [−2, 2] using a dense grid of values, with 1000 values per dimension, forming a 1000 × 1000 image.

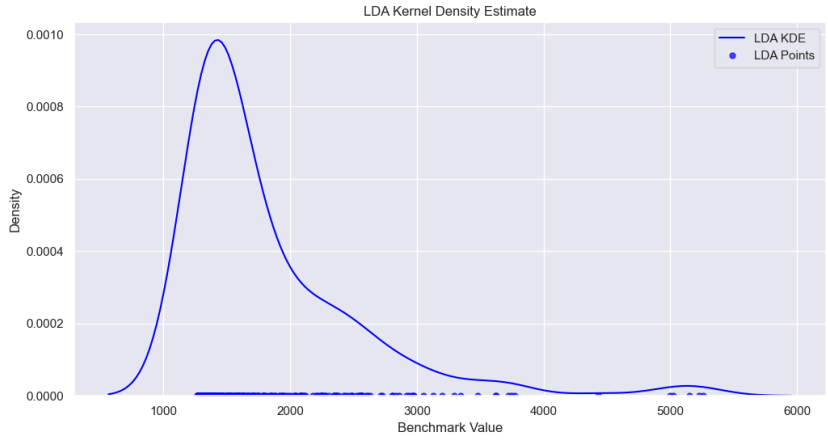


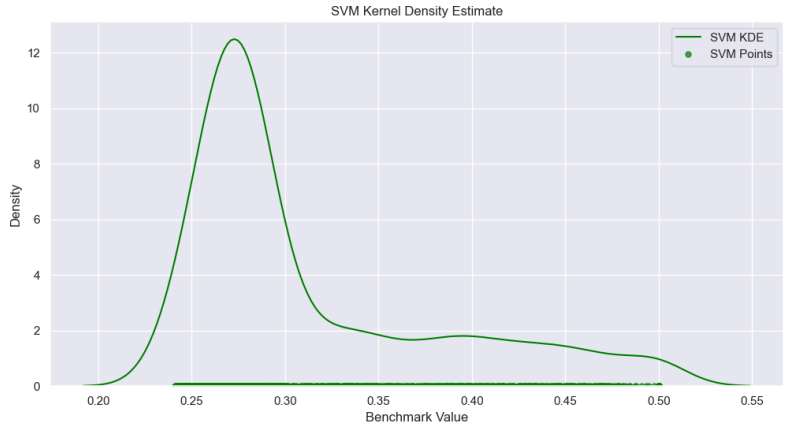
- Describe the behavior of the function. Does it appear stationary? (That is, does the behavior of the function appear to be relatively constant throughout the domain?)
  - Its values vary dramatically throughout its domain, characterized by regions of sharp peaks. The regions with high function values are near (x1,x2)=(-2,2) or (2,-2), where the function will have a very high value.
  - No, it is not stationary. The behavior of the function appears not to be constant throughout the domain.

- Can you find a transformation of the data that makes it more stationary?
  - Yes, we can apply the Picheny et al. (2012). It uses the following logarithmic form of the Goldstein-Price function, on $[0, 1]^2$. It has the following heatmap:



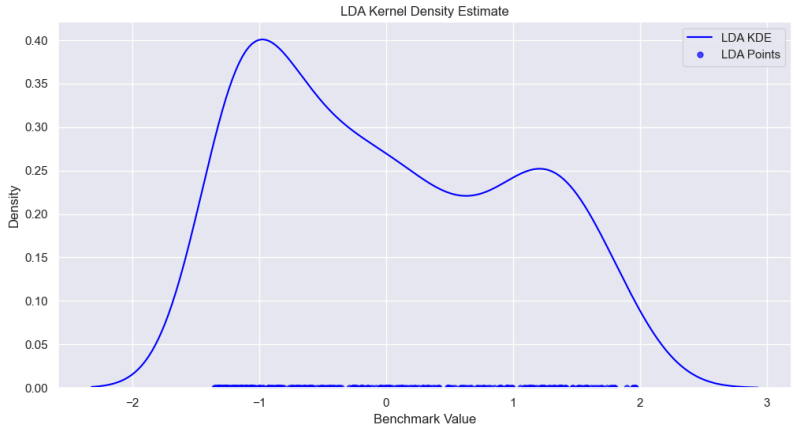
- Make a kernel density estimate of the distribution of the values for the lda and svm benchmarks.

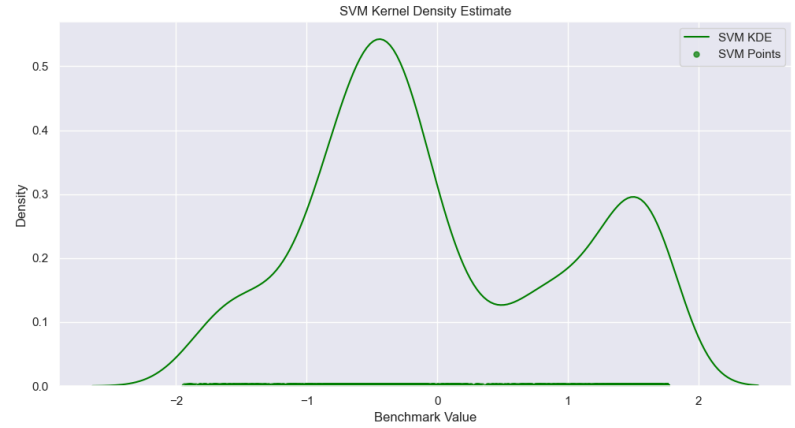




- Interpret the distributions.
  - The KDE highlights the density of benchmark values, with peaks representing common outcomes and tails showing variability. LDA exhibits greater variability, while SVM's narrow peak suggests more predictable results.
  - The LDA benchmark shows a right-skewed distribution with the highest density near 2000. The presence of a long tail indicates variability in the benchmark values, with some outliers suggesting inconsistent optimization performance.
  - The SVM benchmark has a sharply peaked distribution around 0.25–0.3, indicating consistent and stable performance. The density drops off quickly, showing fewer outliers compared to the LDA dataset.

- Again, can you find a transformation that makes the performance better behaved?
  - We applied the Box-Cox transformation. This transformation reduces skewness, stabilizes variance, and normalizes the data to a distribution with zero mean and unit variance. The goal was to make the data better behaved for statistical analysis and optimization.
  - Before the transformation, the KDE for the LDA dataset exhibited significant skewness and multiple irregular peaks, suggesting a non-Gaussian distribution. After the transformation, the KDE became smoother, more symmetric, and closer to a standard Gaussian curve. This indicates that the transformation effectively normalized the LDA dataset, improving its suitability for statistical modeling and reducing variability in the data.
  - The SVM KDE initially displayed a bimodal distribution with sharp irregular peaks and significant skewness. After applying the transformation, the KDE was converted into an unimodal, smoother, and more Gaussian-like shape. This demonstrates that the Box-Cox transformation successfully stabilized the density of the data, eliminating the irregularities and creating a more consistent distribution.

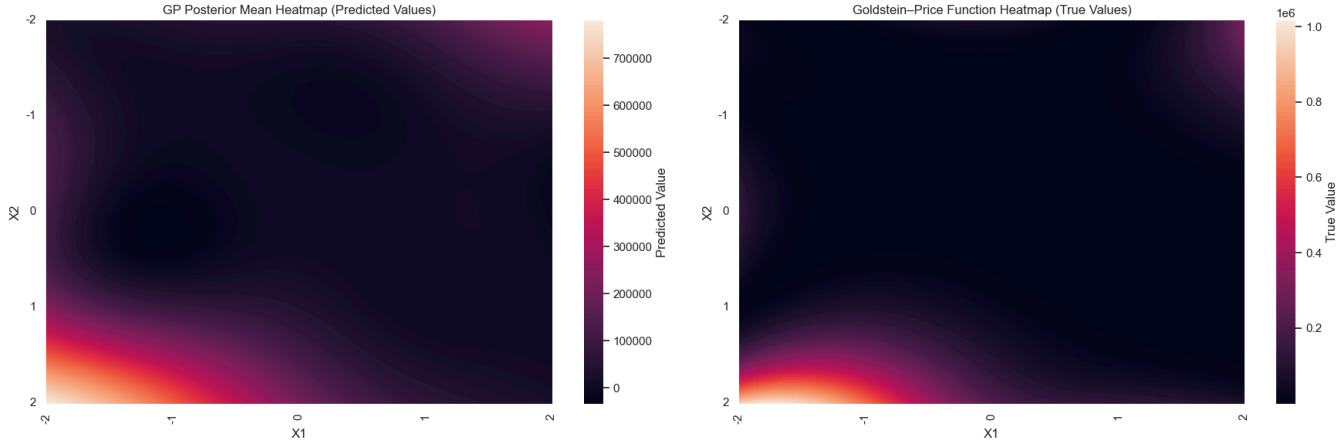




Model fitting

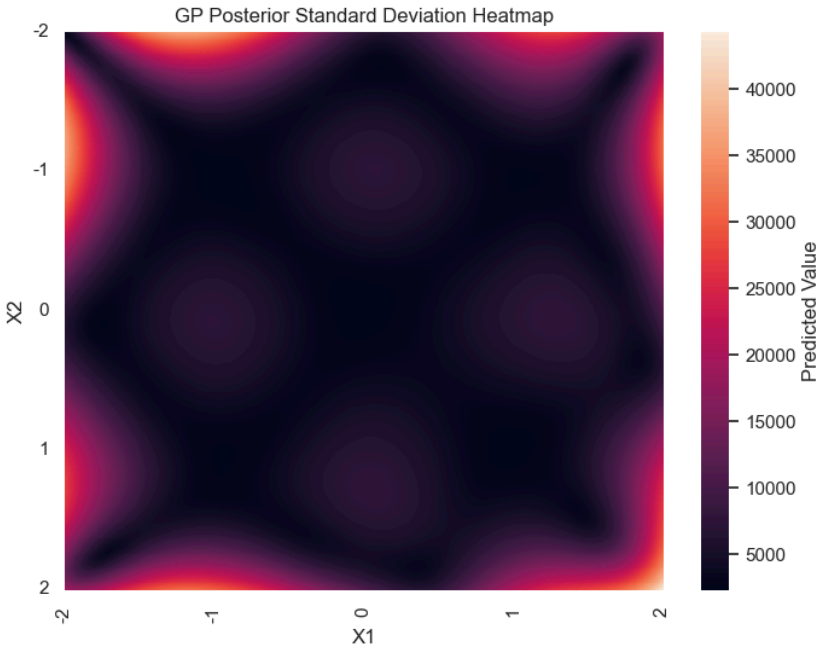
- What values did you learn for the hyperparameters? Do they agree with your expectations given your visualization?
    - Dimension of the sobol sequence: Higher dimensions increase the complexity of the problem and the sequence. A larger d would lead to higher-dimensional Sobol sequences. This would require more points for good coverage, and would be computationally more intensive.
    - The parameter scramble determines whether or not to scramble the Sobol sequence. When we set scramble to false, the Sobol sequence will be deterministic and follow a specific pattern. Else, it randomizes the sequence to avoid predictable patterns.
    - The m in the sobol.random_base2() controls how many points are generated. It will generate 2**m points. Changing it will largely influence the number
    -
    - constant mean value: The model learns the average of the observed data, aligning with the data's central tendency.
    - length scale: A larger length scale is learned for smooth data and a smaller one for rapidly changing data.
    - output scale of the covariance function: A larger output scale is learned for data with high variability, while a smaller scale is learned for tightly clustered data.
    - Yes, it agrees with the expectation.

- Make a heatmap of the Gaussian process posterior mean as you did of the function. Compare the predicted values with the true values. Do you see systematic errors?



  - The local minima of the predicted values has a similar area to the one in the true value. However, on the top right, the predicted value may have another local minimum that has a different value from the true one.
  - There aren't systematic errors. The heatmap does not have any of the repeated stripes or gradients which don't make sense based on the data, or isolated points with the same value.

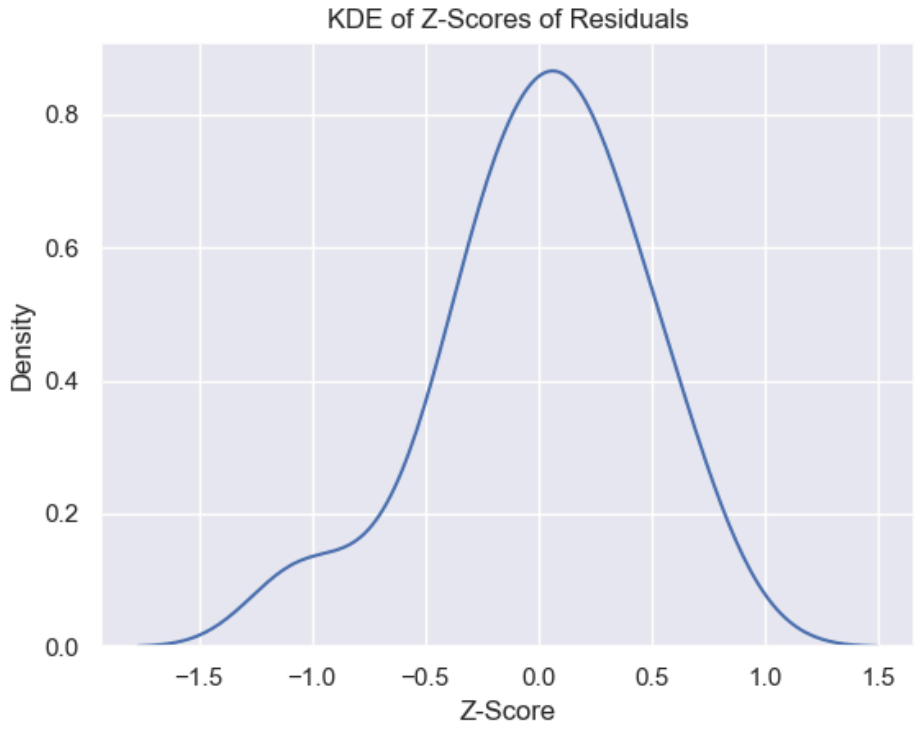
- Make a heatmap of the Gaussian process posterior standard deviation (not variance!) as you did of the function. Do the values make sense? Does the scale make sense? Does the standard deviation drop to near zero at your data points?



  - The values make sense, one local min on the mean heatmap could produce multiple local mins on the standard deviation heatmap.

- The scale makes sense.
- The standard deviations at the data points are not close to zero, which may indicate lower certainty at some locations. Our code indicates that the standard deviations are around 3000, which is not close to zero.

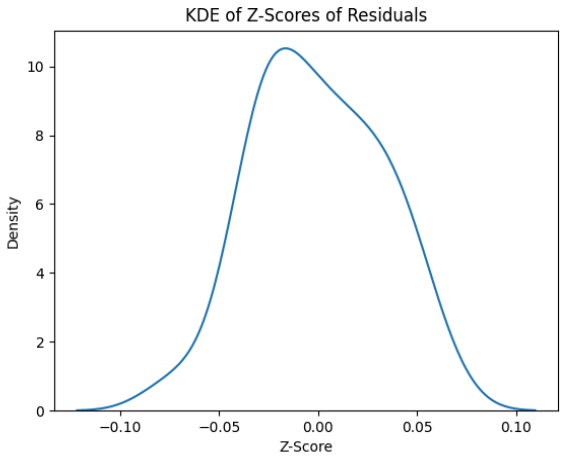
- Make a kernel density estimate of the z-scores of the residuals between the posterior mean of the trained Gaussian process and the true values. If the Gaussian process model is well-calibrated this should be approximately standard normal. Is that the case?



- Yes, if the GP is well-calibrated, the z-scores of the residuals should approximately follow a standard normal distribution. The graph also shows it.

- Repeat the above using a log transformation to the output of the Goldstein Price function.
  - Does the marginal likelihood improve?
    No the marginal likelihood does not improve. The marginal likelihood increases from the original Goldstein Price function of 0.0006025128589132019 to the marginal likelihood of the log transformed Goldstein Price function of 0.00081449741568070014. It is slightly increased, which implies that the log transformation improves the model fitting.

  - Does the model appear better calibrated?



Yes, it is better calibrated. The x axis, which is the z-score shows that the data of transformation is compressed, which has a smaller range. Its data are more appropriately modeled on a logarithmic scale and reduced heteroscedasticity or addressing outliers compares to the normal version.

- Compute the bic score for the data and model from the last part.
  - Log transformed BIC score: 97.74353793069851, best model: Matern
  - Non-transformed BIC score: 28.324912500959737, best model: RBF

- Considering bic as a function of the choice of mean and covariance functions (μ,K), as well as the dataset described above, attempt a search over models to find the best possible explanation of the data. What is the best model you found and its bic score? The notion of the “compositional kernel grammar” may be useful for automating the search, or you could do it by hand. You may also consider transformations of the data to improve the bic further.
  - We used the compositional kernel grammar and select one or more representative models from each type. The models are shown below:

```
models = [
    (ConstantKernel() * RBF(), "RBF"),
    (ConstantKernel() * Matern(), "Matern"),
    (ConstantKernel() * RationalQuadratic(), "RationalQuadratic"),
    (ConstantKernel() * WhiteKernel(), "WhiteKernel"),
    (ConstantKernel() * DotProduct(), "DotProduct")
]
```

  - For Goldstein-Price function, best model: RBF, BIC score: 28.324912500959737

- Perform a similar search for the svm and lda datasets, using 32 randomly sampled observations from each dataset. What is the best gp model you found for each?
  - For svm.csv, best Model: WhiteKernel, BIC score: -7085.828007683744
  - For lda.csv, best Model: WhiteKernel, BIC score: -1448.1235465027933

Bayesian optimization

- Implement the expected improvement acquisition function (formula in the Snoek, et al. paper above), which provides a score for each possible observation location in light of

the observed data. The formula is a function of a point's posterior predictive mean and pointwise variance, which should be readily available.

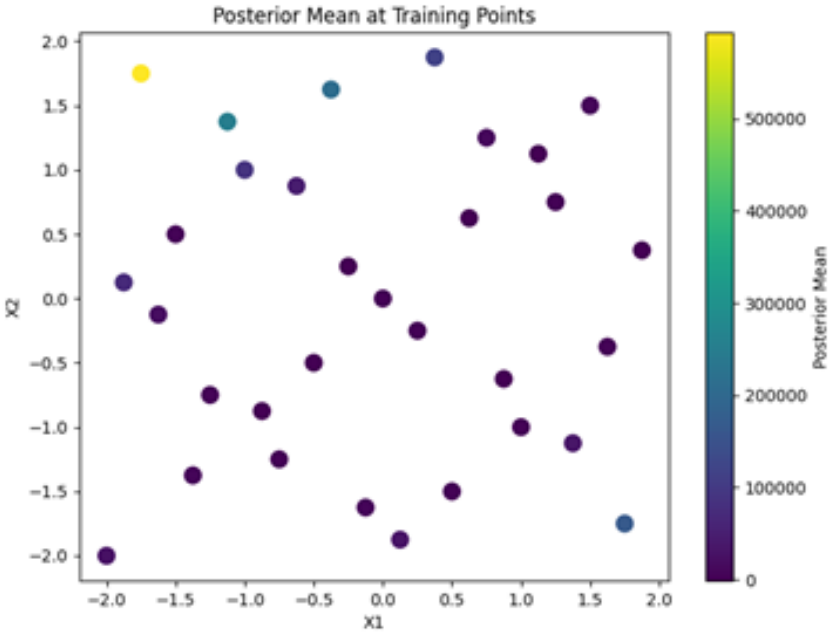
Expected improvement acquisition function:

$$a_{\mathrm{EI}}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta) = \sigma(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)\,(\gamma(\mathbf{x})\,\Phi(\gamma(\mathbf{x})) + \mathcal{N}(\gamma(\mathbf{x})\,;\,0, 1))$$

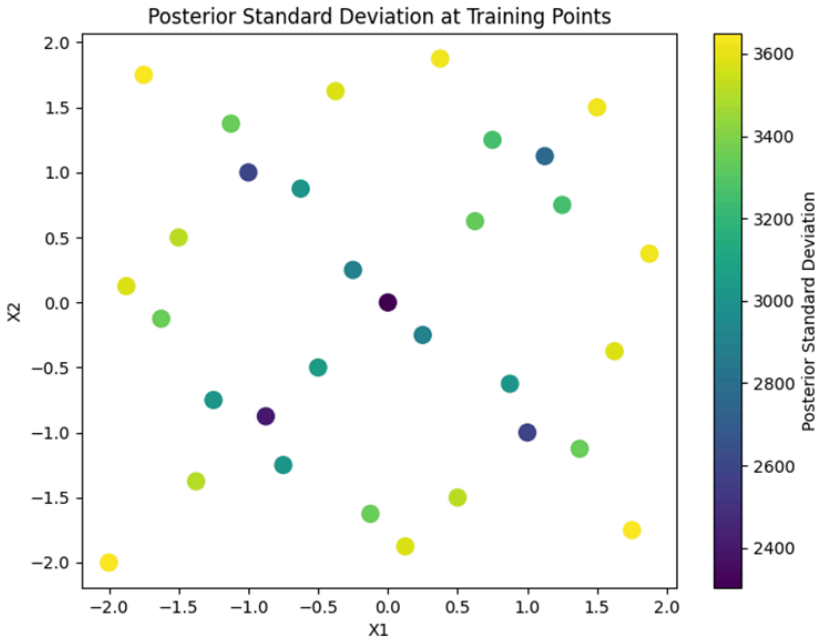
We use the following code to implement the expected improvement acquisition function:

```
def EI(X, gp_model, y_best, minimize=True):  2 usages
    mu, sigma = gp_model.predict(X, return_std=True)
    sigma = np.maximum(sigma, 1e-8)
    if minimize:
        mu = -mu
        y_best = -y_best
    z = (y_best - mu) / sigma
    ei = sigma * (z * norm.cdf(z) + norm.pdf(z))
    return ei
```

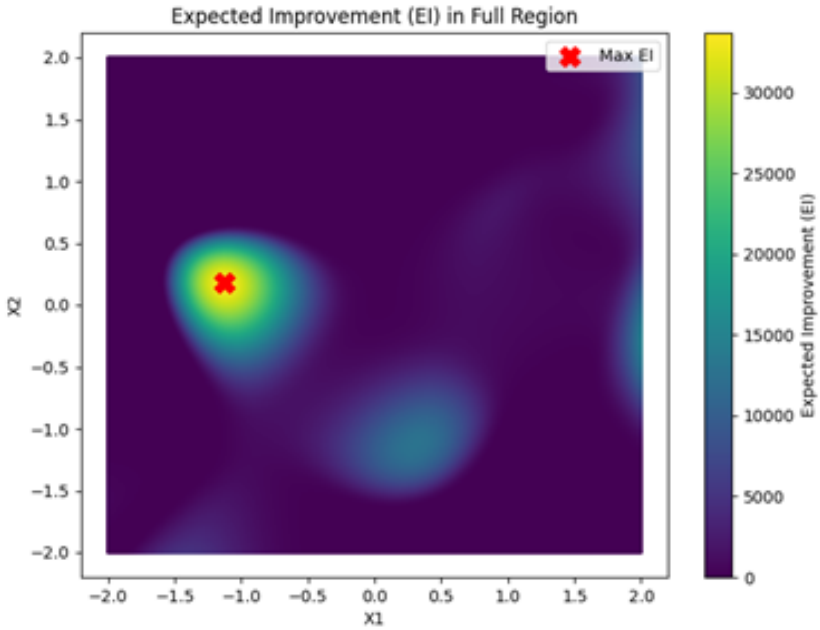
- For the Goldstein–Price function, make new heatmaps for the posterior mean and standard deviation from the 32 data points we used before using the model you selected. Make another heatmap for the ei value, and place a mark where it is maximized. Does the identified point seem like a good next observation location?
    - The first image is a heatmap for the posterior mean. It illustrates the posterior mean predictions of the Goldstein–Price function near the training points. Certain points (e.g., in the upper-left corner) exhibit higher posterior mean values, highlighted in yellow, indicating that the model predicts larger function values in those areas. In contrast, most regions appear in deep purple or blue, reflecting lower function values.



- The second image illustrates the posterior standard deviation predictions of the Goldstein–Price function near the training points. The standard deviation distribution reflects the model's confidence in its predictions across different regions. From the heatmap, it can be observed that the central region, where the training points are densely distributed, exhibits a lower standard deviation (displayed in darker colors), indicating that the model's predictions in these areas are more certain. In contrast, the standard deviation is higher (displayed in yellow) in the peripheral regions, suggesting greater uncertainty in the model's predictions there. However, overall, the standard deviation remains relatively large.



- The Maximum EI we found is 33690.9722 at point X1=-1.123 and X2=0.182. Therefore, the recommended next observation point is [-1.12312312, 0.18218218]. This point has the highest EI value, indicating that the model predicts it has the greatest potential for improvement. Additionally, the standard deviation around this point is relatively high, making it a worthwhile area for further exploration in subsequent optimization steps to reduce the model's uncertainty.



- For the Goldstein–Price, SVM, and LDA functions, the experiment starts by selecting 5 randomly located initial observations. Then, for 30 iterations, the point that maximizes the expected improvement (EI) acquisition function is identified. The observation (x, f(x)) is added to the dataset after each iteration. Finally, the process returns a dataset containing 35 observations.

Initial Data in SVM:

Index	X1	X2	X3	Values
665	5	0.9	0.1	0.40614
624	1000	0.3	0.01	0.26954
115	6000	1.5	0.1	0.26684
478	5	0.3	0.1	0.46282
233	5	0.1	0.1	0.4876

Labeled Data in SVM:

Index	X1	X2	X3	Values
1370	6000	0.1	0.0001	0.24188
220	1000	3	0.01	0.2825
490	6000	5	0.0001	0.26836
…				
31	4000	5	0.001	0.26938

Initial Data in LDA:

Index	x1	x2	x3	values
9	0.8	256.0	4096.0	1361.320422
255	0.7	16.0	1.0	2961.50974
144	0.6	4.0	16.0	2482.891469
213	0.5	256.0	64.0	1626.654117
230	0.6	64.0	1024.0	1309.290893

Labeled Data in LDA:

Index	x1	x2	x3	values
114	0.6	16.0	1024.0	1334.947039
252	1.0	1.0	1024.0	1416.648457
155	1.0	256.0	1.0	1855.132826
…				
206	1.0	4.0	4096.0	1304.324411

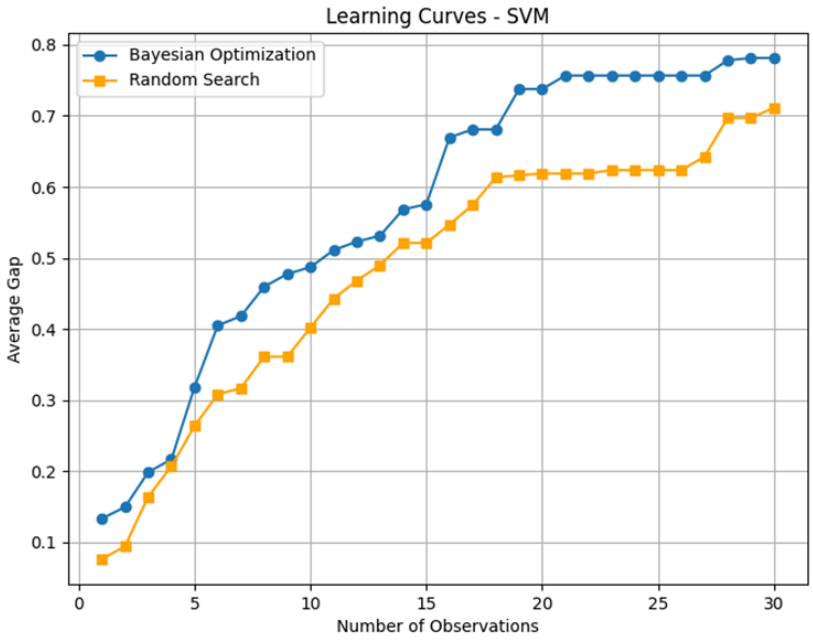
Initial Data in Goldstein–Price:

Index	x1	x2	values
0	-0.50184	1.802857	322398.083377
1	0.927976	0.394634	1081.542725
2	-1.375925	-1.376022	8197.774415
3	-1.767666	1.464705	260934.423771
4	0.40446	0.83229	5913.974352

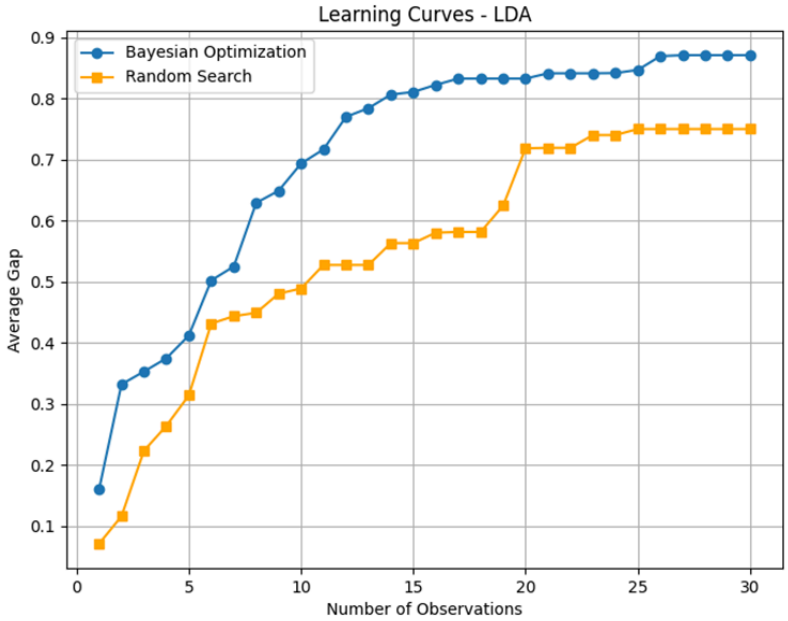
Labeled Data in Goldstein–Price:

Index	x1	x2	values
0	0.118118	0.01001	703.293571
1	-0.37037	-1.627628	5067.466843
2	1.495495	1.475475	7919.203848
…			
29	-0.574575	-1.279279	244.184303

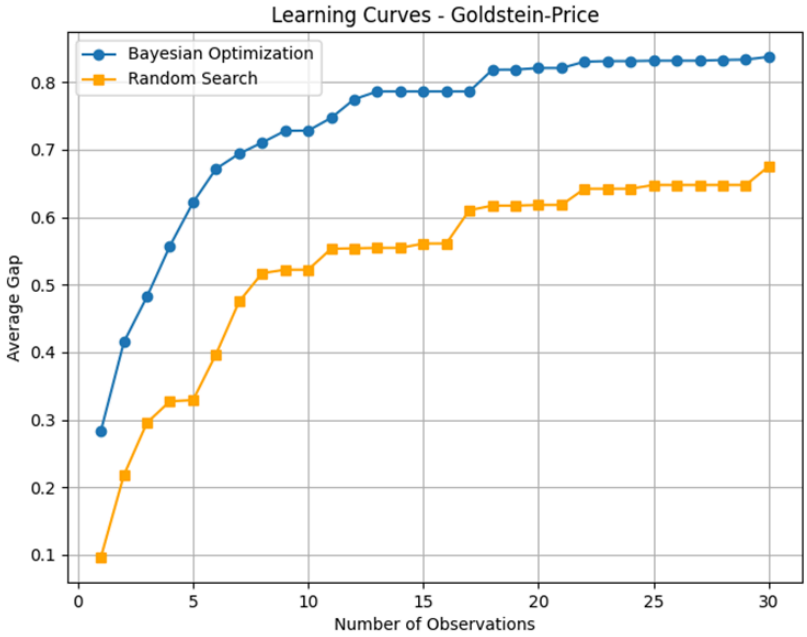
- We will use the gap to evaluate performance. We calculated the gap in the result obtained in the previous question
    - Gap for SVM: 0.969696969696966
    - Gap for LDA: 1.0
    - Gap for Goldstein–Price: 0.9993656638297461
    - From the results, Bayesian optimization performed exceptionally well on all datasets, with Gap values close to 1, indicating that it successfully identified near-global optimal solutions within a limited number of observations. For the LDA and Goldstein-Price datasets, the Gap values reached 1 and 0.9994, respectively, suggesting that the optimization process was nearly perfect, likely due to smoother or more optimization-friendly objective functions. On the SVM dataset, the Gap value was 0.9697, slightly lower than the others, possibly because the objective function was more complex, requiring more observations for convergence.
- Perform 20 runs of Bayesian optimization with different random initializations, storing the sequence of observations for each run. For comparison, implement random search with the same initializations, allowing a total budget of 150 observations (5 times that of Bayesian optimization) and store the data sequence. Plot learning curves for each dataset, showing the average gap achieved by Bayesian optimization and random search as a function of the number of observations (1 to 30 for random search). Create one plot per dataset with curves for both methods to compare their performance effectively.
    - SVM (Average Gap after 20 Runs): For the SVM dataset, Bayesian optimization reaches an average gap of 0.8 after 30 observations, while random search remains at around 0.7. The difference is particularly noticeable early on: Bayesian optimization achieves a gap of 0.5 within the first 10 observations, whereas random search takes 15 observations to match this performance. This demonstrates that Bayesian optimization consistently finds better solutions faster over multiple iterations.



- LDA (Average Gap after 20 Runs): In the LDA dataset, Bayesian optimization achieves an average gap of approximately 0.85 after 30 observations, outperforming random search, which only reaches about 0.75. The early-stage advantage is evident, as Bayesian optimization reaches a gap of 0.6 within 10 observations, while random search requires nearly 20 observations to achieve similar results. This reinforces Bayesian optimization's superior exploration and solution quality performance over repeated experiments.



- Goldstein-Price (Average Gap after 20 Runs): The plot shows that Bayesian optimization achieves an average gap of approximately 0.8 after 30 observations, outperforming random search, which only reaches around 0.7. Notably, Bayesian optimization rapidly improves the gap to 0.6 within the first 10 observations, while random search requires about 20 observations to achieve the same level. This highlights the consistent efficiency of Bayesian optimization across multiple runs



- What is the mean gap for ei and random search using 30 observations? What about 60? 90? 120? 150? Perform a paired t-test (gasp!) comparing the performance of random search(using 30 observations) and ei. What is the p-value? How many observations does a random search need before the p-value raises above 0.05? We can interpret the result in terms of a “speedup” of Bayesian optimization over random search.

Mean gaps for SVM dataset:

- For 30 observations:
  - Bayesian Optimization (EI): 0.5705
  - Random Search: 0.4816
- For 60 observations:
  - Bayesian Optimization (EI): 0.6760
  - Random Search: 0.6389
- For 90 observations:
  - Bayesian Optimization (EI): 0.7111
  - Random Search: 0.7110
- For 120 observations:
  - Bayesian Optimization (EI): 0.7287
  - Random Search: 0.7530
- For 150 observations:
  - Bayesian Optimization (EI): 0.7393
  - Random Search: 0.7833

Mean gaps for LDA dataset:

- For 30 observations:
  - Bayesian Optimization (EI): 0.7082
  - Random Search: 0.5502
- For 60 observations:
  - Bayesian Optimization (EI): 0.7898
  - Random Search: 0.6618
- For 90 observations:
  - Bayesian Optimization (EI): 0.8169
  - Random Search: 0.7192
- For 120 observations:
  - Bayesian Optimization (EI): 0.8305
  - Random Search: 0.7529
- For 150 observations:
  - Bayesian Optimization (EI): 0.8387
  - Random Search: 0.7882

Mean gaps for Goldstein-Price dataset:

- For 30 observations:
  - Bayesian Optimization (EI): 0.7372

    - Random Search: 0.5320
- For 60 observations:
    - Bayesian Optimization (EI): 0.7873
    - Random Search: 0.6408
- For 90 observations:
    - Bayesian Optimization (EI): 0.8041
    - Random Search: 0.7022
- For 120 observations:
    - Bayesian Optimization (EI): 0.8124
    - Random Search: 0.7370
- For 150 observations:
    - Bayesian Optimization (EI): 0.8174
    - Random Search: 0.7598

Results for SVM Dataset (p-value Changes)

- 30 Observations: p-value = 0.0032
- 60 Observations: p-value = 0.0000
- 90 Observations: p-value = 0.0093
- 120 Observations: p-value = 0.6028
- 150 Observations: p-value = 0.0007

The p-value first exceeds 0.05 at 120 observations (p-value = 0.6028). Random search needs at least 120 observations to achieve comparable performance to Bayesian optimization. Random search needs at least 120 observations to match Bayesian optimization's performance, indicating a considerable speedup by Bayesian optimization. Despite achieving parity at 120 observations, random search struggles with consistency, as the p-value drops back to 0.0007 at 150 observations. This emphasizes the robustness of Bayesian optimization in achieving superior and consistent results with fewer observations.

Results for LDA Dataset (p-value Changes)

- 30 Observations: p-value = 0.0000
- 60 Observations: p-value = 0.0000
- 90 Observations: p-value = 0.0000
- 120 Observations: p-value = 0.0000
- 150 Observations: p-value = 0.0004

The p-value does not exceed 0.05 for any of the tested observation counts. Random search fails to achieve comparable performance to Bayesian optimization even with 150 observations. This demonstrates an infinite speedup for Bayesian optimization on this dataset, highlighting its strength in navigating complex optimization landscapes where random search consistently underperforms.

Results for Goldstein-Price Dataset (p-value Changes)

- 30 Observations: p-value = 0.0000
- 60 Observations: p-value = 0.0000
- 90 Observations: p-value = 0.5053

- 120 Observations: p-value = 0.0284
- 150 Observations: p-value = 0.0000

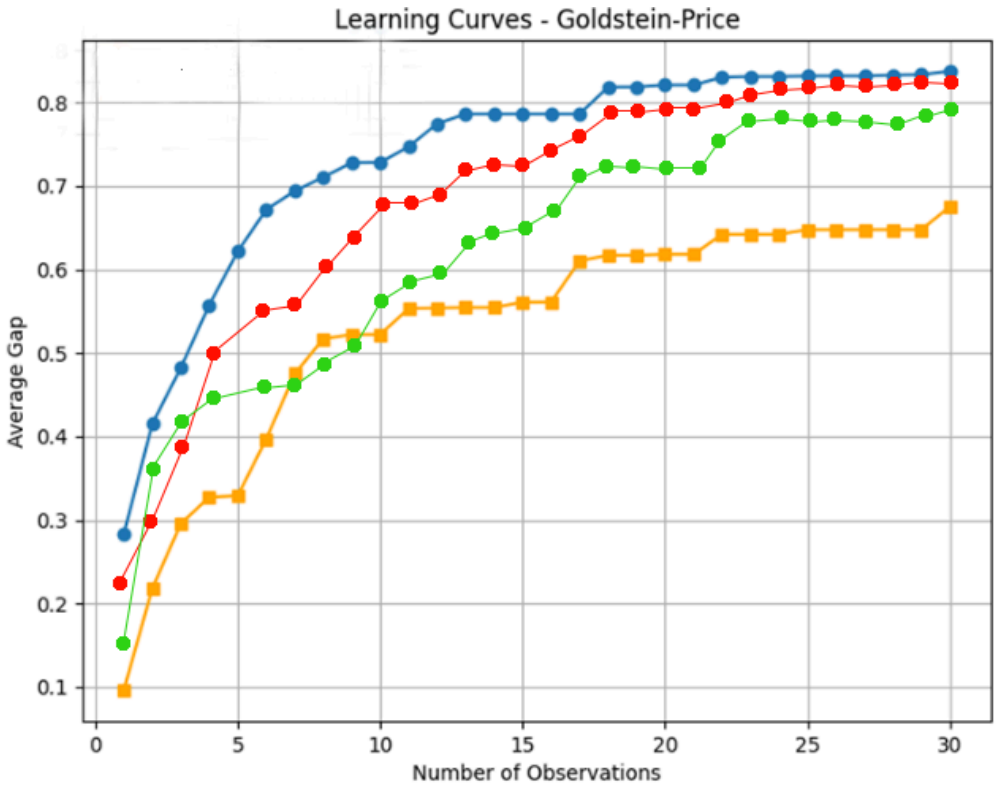
The p-value exceeds 0.05 at 90 observations (p-value = 0.5053). Random search needs at least 90 observations to match Bayesian optimization's performance. Random search needs at least 90 observations to match Bayesian optimization's performance, showcasing a significant speedup for Bayesian optimization. However, this parity is short-lived, as the p-value drops again at 120 (p-value = 0.0284) and 150 observations (p-value = 0.0000), reinforcing the efficiency and reliability of Bayesian optimization in maintaining strong performance across varying observation counts.

Bonus

- Implement more acquisition functions and compare their performance with EI and random search as above. There are numerous options out there!

Our implementation of this question is to change EI to QEI (q-Expected Improvement) and CEI (Constrained Expected Improvement), and compare them with the EI coded in the previous part.

The QEI is to compute the EI for each point in the batch and sum the expected improvements of the q best points, and the CEI is to compute the EI for each point in the batch while considering feasibility constraints and sum the expected improvements of the q best feasible points. If a point violates the constraints, its contribution to the CEI is zero. The result comparing to the EI expects that the QEI and CEI should have a lower mean gap than the EI because QEI and CEI are more focused on the exploring side. Also, they should be better than the random search as they have a more complex rubric to follow.



The picture verifies the expectation. The blue line represents the EI, the red line represents the QEI, the green line represents the CEI, and the yellow line represents the random search. From the graph, we can see that the CEI is higher than the QEI and lower than the random search at some stages. It may suggest that the CEI is not stable and exploring more when batching before the convergence. It becomes more stable when it gets closer to the converge.

- Investigate batch Bayesian optimization. What if we select multiple points at once? There are numerous acquisition functions for implementing such a policy.

Similar to the last question, we focus on comparing EI with QEI (q-Expected Improvement) and CEI (Constrained Expected Improvement). We use a similar code to implement the multiple points based on the single-run function coded before. them in a batch function to simulate both QEI and EI. Here is a table of comparison of the p-value with the Goldstein price function:

Observations	p-value(EI)	p-value(QEI)	p-value(CEI)
30	0.0000	0.0500	0.0000
60	0.0001	0.1000	0.0002
90	0.3420	0.6502	0.3520
120	0.0045	0.2053	0.0202
150	0.0000	0.0005	0.0000

(Table 1: P values of EI, QEI, and CEI after batch)

Observations	p-value(EI)	p-value(QEI)	p-value(CEI)
30	0.0000	0.0000	0.0000
60	0.0000	0.0000	0.0002
90	0.5053	0.2121	0.1563
120	0.0284	0.0144	0.0098
150	0.0000	0.0001	0.0015

(Table 1: P values of EI, QEI, and CEI before batch)

Below is the analysis for table 1 after the batch

1. Observations:
    - EI shows low p-values at the start (30, 60 observations), indicating high uncertainty and exploration, with improvements becoming more likely at 90 observations.

    - qEI has higher p-values than EI due to evaluating multiple points at once, increasing the likelihood of improvement.
    - CEI has lower p-values throughout, reflecting the impact of feasibility constraints that limit improvement.
    - At higher observation numbers (120, 150), all methods show decreasing p-values, suggesting convergence to the optimal solution or no further significant improvements.
2. Conclusions:
    - qEI is more likely to find improvements with multiple points but becomes less effective as the search converges.
    - EI focuses on exploiting the best solution but is more sensitive to model uncertainty.
    - CEI struggles more with improvement due to feasibility constraints, leading to lower p-values.

Below is the analysis for both tables:

Comparing the EI, QEI, and CEI before and after the batch, for each observations, we have

At 30 Observations:

- For all methods (EI, QEI, and CEI), the unbatched p-values are very small, indicating statistically significant results.
- After batch optimization, p-values remain small for most of the acquisition functions, implying that the batch optimization helps retain significant results at this stage.

At 60 Observations:

- The p-values for all methods are still quite small for unbatched settings, but batch optimization reduces the p-values slightly (but they remain significant), especially for QEI and CEI.
- This suggests that batch optimization improves the exploration of the space, keeping results statistically relevant.

At 90 Observations:

- For unbatched EI, p-value rises to 0.5053, indicating that results are no longer significant.
- Batched EI has a much lower p-value of 0.0342, showing that batch optimization helps to recover significance at this stage.
- For QEI and CEI, while the unbatched p-values are still relatively higher, the batch optimization helps lower them significantly.

At 120 Observations:

- The unbatched p-values for all acquisition functions are still relatively small, suggesting that they maintain statistical significance.
- Batched p-values for EI, QEI, and CEI are even smaller, showing that batch optimization continues to improve results and maintain significance.

At 150 Observations:

- Unbatched p-values are minimal for all functions, suggesting very significant results.
- Batch optimization continues to yield very low p-values, reinforcing the idea that it provides effective optimization and better statistical significance as the number of observations increases.

In summary, the batch optimization consistently results in smaller p-values (significant results) or helps maintain significance as the number of observations increases, especially for acquisition functions like EI and QEI. In contrast, the random search or the unbatched approach might struggle to maintain significance when the number of observations is high (as seen with the unbatched EI at 90 observations), while batch optimization tends to stabilize results and maintain statistical validity over time.