**Initial Codebase**

The initial codebase I developed for the project was primarily focused on generating input data suitable for submission to the black-box optimization competition. Given that all inputs had to be within the range of 0 to 1 and that I couldn't evaluate the functions locally, my primary concern was creating a flexible system to generate and manage these inputs effectively. I utilized Python for scripting and relied on basic libraries like NumPy to handle numerical operations and data manipulation.

For instance, the core function I used to generate random inputs was:

A black background with white text

Description automatically generated

This function allowed me to create random input vectors for any of the eight functions, adjusting the dimensions parameter according to each function's requirements.

**Decision-Making Process**

I chose this straightforward approach because, under the constraints of the competition, simplicity and reliability were paramount. Since I couldn't test or evaluate the functions on my end, it was essential to have a method that could produce valid inputs without introducing unnecessary complexity. The code needed to be easily adjustable as I gathered more data from the weekly submissions, allowing for incremental improvements based on the limited feedback received.

**Initial Strategy**

In the early stages, my strategy revolved around exploring the input space as broadly as possible. With submissions limited to once a week per function, each input had to be chosen carefully to maximize the potential for gaining useful information. Random sampling across the entire input range seemed like a logical starting point to achieve a wide coverage.

I generated initial inputs for each function as follows:A computer screen with text on it

Description automatically generated

These inputs were then formatted according to the submission guidelines and stored for tracking purposes.

**Initial Setup**

The setup process involved organizing these inputs systematically. I created separate directories or files for each function to keep the data organized. After submitting the inputs, I waited for the weekly results, which I then logged alongside the corresponding inputs for future reference.

**Getting Comfortable with the Process**

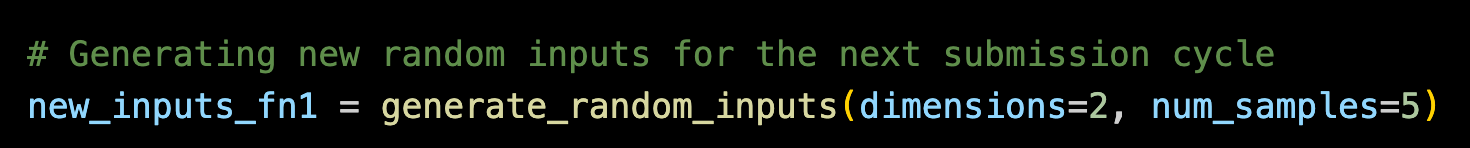
Adjusting to the weekly submission cycle required patience and careful planning. I realized early on that meticulous record-keeping would be crucial. Each week, I made sure to:

* **Submit on Time**: Missing a submission would mean losing a week's worth of data.
* **Log Results Accurately**: Inputs and their corresponding outputs were recorded diligently.
* **Reflect on Outcomes**: I spent time analysing any patterns or notable results from the submissions.

**Code Modification**

As I gathered more data from the weekly submissions, I began to modify my code to incorporate insights gleaned from previous results. Although the data was sparse, certain trends started to emerge, prompting me to adjust my input generation strategy.

**Early Exploration: Random Sampling**

Initially, I continued with random sampling to maintain a broad search across the input space. However, I started to pay closer attention to the outputs received, noting any inputs that resulted in comparatively better outputs.

**Transition to Structured Inputs**

After a few weeks, I noticed that some inputs consistently performed better than others. Although the data was limited, it suggested that certain regions of the input space might be more promising.

To explore this, I adjusted the input generation function to focus on narrower ranges:A screen shot of a computer code

Description automatically generated

By narrowing the input ranges based on previous outputs, I aimed to explore into areas that might yield better results.

**Incorporating Surrogate Models**

Given the inability to evaluate the functions directly, I experimented with simple surrogate models to predict potential outputs based on the data collected. With the limited dataset, I tried basic interpolation and regression techniques.A screen shot of a computer code

Description automatically generated

This approach was tentative, given the small sample size, but it provided a rudimentary way to prioritize certain inputs over others.

**Adjustments for Higher Dimensionality**

For functions with more dimensions, such as Function 7 (6-dimensional) and Function 8 (8-dimensional), the challenges intensified. The vastness of the input space made random sampling less effective.

To address this, I:

* **Fixed Certain Variables**: Based on the minimal data, I fixed some variables at values that previously resulted in better outputs.A screen shot of a computer code

  Description automatically generated
* **Reduced Sampling Range**: I narrowed the input ranges for certain variables to focus on promising regions.

**Iterative Refinement**

Each subsequent week, I integrated new data into my analysis:

* **Data Analysis**: I plotted inputs versus outputs to identify any visible trends.
* **Adaptive Input Generation**: Adjusted input ranges and sampling methods based on observed patterns.
* **Surrogate Model Updates**: Retrained the surrogate models with the new data to improve predictions.

**Practical Adjustments**

Recognizing the limitations imposed by the competition's structure, I made several practical adjustments:

* **Enhanced Data Logging**: Improved the logging mechanism to include more details, such as the date of submission and any relevant notes.A screen shot of a computer code

  Description automatically generated
* **Time Management**: Set reminders for submission deadlines to ensure no opportunities were missed.
* **Function Prioritization**: Allocated more effort to functions where progress seemed attainable, given the constraints.

**Incorporating Advanced Surrogate Models**

As the competition progressed and I accumulated more data, it became evident that basic regression techniques and simple interpolation were insufficient for capturing the nuances of the black-box functions. To enhance my predictive capabilities, I decided to implement a more advanced surrogate model using Gaussian Process Regression (GPR). This approach is well-suited for modeling complex functions with limited data, as it provides a probabilistic framework that can quantify uncertainty in predictions.

I utilized the GaussianProcessRegressor from scikit-learn, opting for the Matern kernel due to its flexibility and ability to model a wide range of functions:A screen shot of a computer program

Description automatically generated

To select the most promising inputs for the next submission, I calculated the Expected Improvement (EI) acquisition function:A computer screen shot of text

Description automatically generated

By integrating this advanced surrogate model and acquisition function, I aimed to make more informed decisions on which inputs to submit, despite the limitations of sparse data and delayed feedback. The probabilistic nature of GPR allowed me to quantify uncertainty and focus on inputs that not only had high predicted outputs but also offered significant potential for improvement.

**Final Phase: Focused Optimization**

As the competition approached its conclusion, I concentrated my efforts on fine-tuning inputs for functions where I had observed the most promising results.

* **Local Perturbations**: Made small adjustments around the best-performing inputs to explore the immediate vicinity.A computer screen with text

  Description automatically generated
* **Repeated Submissions**: For functions showing consistent performance, I repeated similar inputs to verify the stability of the outputs.

**Final Result and Performance Reflections**

By the end of the competition, I found myself somewhat dissatisfied with my overall performance. While there were moments of progress, I couldn't shake the feeling that the results didn't fully reflect the effort and strategies I had employed. Several factors contributed to this outcome.

Firstly, managing eight separate functions simultaneously proved more challenging than I had anticipated. Each function had its own characteristics and required individual attention. Balancing my focus across all of them meant that I couldn't delve deeply into any single function, potentially limiting the effectiveness of my optimization efforts. This spread-thin approach may have prevented me from achieving significant breakthroughs in any particular area.

Secondly, I suspect that there might have been undetected bugs or flaws in my code. The advanced surrogate models, while theoretically sound, didn't always yield the expected improvements. In some cases, the predictions seemed inconsistent with the observed outcomes, suggesting possible issues with the model implementation or data handling processes. Unfortunately, the competition's structure—specifically, the inability to evaluate functions locally and the time-gated submissions—made it difficult to thoroughly debug and validate my code within the given timeframe.

Moreover, the limited amount of data and the sparse feedback loop exacerbated these challenges. With only weekly results to inform my adjustments, any errors in the surrogate models or data processing could have gone unnoticed, compounding over time and undermining the overall effectiveness of my strategy.

In hindsight, the combination of managing multiple functions, potential code issues, and the constraints of the competition created a perfect storm that hindered my ability to achieve the desired results. While the experience was undoubtedly valuable and provided numerous learning opportunities, it's clear that there were areas where my approach could have been improved.

**General Performance Observations**

Throughout the competition, several observations stood out:

* **Data Limitations**: The sparse data hindered the ability to make robust predictions or build reliable models.
* **Function Complexity**: Some functions appeared to have a high degree of complexity or noise, making patterns less discernible.
* **Dimensionality Impact**: Lower-dimensional functions were more amenable to the strategies employed, while higher-dimensional ones were less responsive.

**Challenges Encountered**

The competition presented several inherent challenges:

* **Weekly Submission Cycle**: The infrequent feedback made iterative improvement slow and required careful planning.
* **Inability to Evaluate Locally**: Without local evaluation, I couldn't test hypotheses or validate models outside of the submission process.
* **Limited Data Points**: With only a handful of data points per function, statistical methods were less effective.

**Lessons Learned**

1. **Strategic Patience**: The competition emphasized the importance of patience and long-term planning over quick iterations.
2. **Record-Keeping**: Detailed logs were invaluable for tracking progress and making informed decisions.
3. **Model Limitations**: Recognized the limitations of surrogate models with sparse data and adjusted expectations accordingly.
4. **Flexibility**: Adapting strategies in response to new data was crucial, even if changes were incremental.

**Future Improvements**

Reflecting on the experience, several areas for improvement emerged:

* **Enhanced Experimental Design**: Employ methods like Bayesian experimental design to select inputs that maximize expected information gain.
* **Focus on Uncertainty Quantification**: Incorporate uncertainty estimates to prioritize inputs with the highest potential impact.

**Conclusion**

Participating in the black-box optimization competition was both challenging and enlightening. The constraints imposed forced me to think carefully about each decision and highlighted the difficulties of optimization without direct feedback. While the results were mixed, the process provided valuable lessons in patience, data management, and strategic planning under uncertainty.