Airfoil Optimization Using GP

Raktim Bhattacharya

Problem Statement

- We have aerodynamic data
  - In this case L/D as a function of **thickness** and **camber**
  - The data is *sparse* – little data since they are obtained from high fidelity CFD or experiments
  - There is a function that relates L/D with camber and thickness
- Want to find thickness and camber that minimizes L/D
  - We do not know this function – only a few noisy snapshots (the data)
  - How do we find the minimum of this function?

GP Based Approach

Steps to follow

1. Create a GP *surrogate* model (approximate representation of the function)
2. Solve for optimal thickness and camber using GP model
3. Refine GP model by adding a new data point (High fidelity CFD or experiment)
4. Optimize again with the refined model
5. If difference between two optimizations are not small, go to step 3 until convergence.

A Python Code Example

Objective Function

```
def objective_function(x):
    thickness, camber = x[:, 0], x[:, 1]
    lift_to_drag_ratio = (4*thickness - thickness**2 + 2*camber - camber**2)
    return lift_to_drag_ratio[:, None]
```

- This function represents a simplified model of the lift-to-drag ratio as a function of two design variables: thickness-to-chord ratio and camber.
- The actual function should be based on aerodynamic principles and typically involves complex CFD simulations or experimental evaluations. In your assignment, you will call XFoil to compute L/D for a NACA airfoil with a given thickness-to-chord ratio and camber.

Design Space

```
bounds = [{'name': 'thickness', 'type': 'continuous', 'domain': (0.1, 0.18)},
          {'name': 'camber', 'type': 'continuous', 'domain': (0.02, 0.05)}]
domain = bounds
```

- The design space is defined by specifying the bounds for each design variable.
- In this case, the thickness-to-chord ratio and camber have specified continuous ranges.

Bayesian Optimization Setup

```
optimizer = BayesianOptimization(f=objective_function,
                                 domain=domain,
                                 model_type='GP',
                                 acquisition_type='EI',
                                 acquisition_jitter=0.05,
                                 exact_feval=True,
                                 maximize=False)
```

- A Bayesian Optimization object is created using the `GPyOpt` library.
- The objective function and the design space (domain) are specified.
- The model_type is set to *GP*, indicating that a Gaussian Process model will be used to approximate the objective function.
- The `acquisition_type` is set to *EI*, which stands for Expected Improvement, a common acquisition function in Bayesian optimization.

- `acquisition_jitter` is a small noise term added to the acquisition function to encourage exploration.
- `exact_feval` is set to *True*, indicating that the objective function evaluations are exact (noisy evaluations can be handled by setting this to False).
- `maximize` is set to True because the goal is to maximize the lift-to-drag ratio.

Optimization Loop

```
optimizer.run_optimization(max_iter=max_iter, verbosity=True)
```

- The optimization is run for a specified number of iterations (`max_iter`).
- At each iteration, the algorithm selects a new design to evaluate based on the acquisition function (EI in this case).
- The GP model is updated with the new data point (design and its corresponding objective function value).

Results

```
print("Optimal design:", optimizer.x_opt)
print("Optimal lift-to-drag ratio:", -optimizer.fx_opt)
```

- After the optimization loop, the optimal design and the corresponding lift-to-drag ratio are printed.
- The negative sign is used to convert the value back to the original maximization problem.

Convergence Plot

```
optimizer.plot_convergence()
```

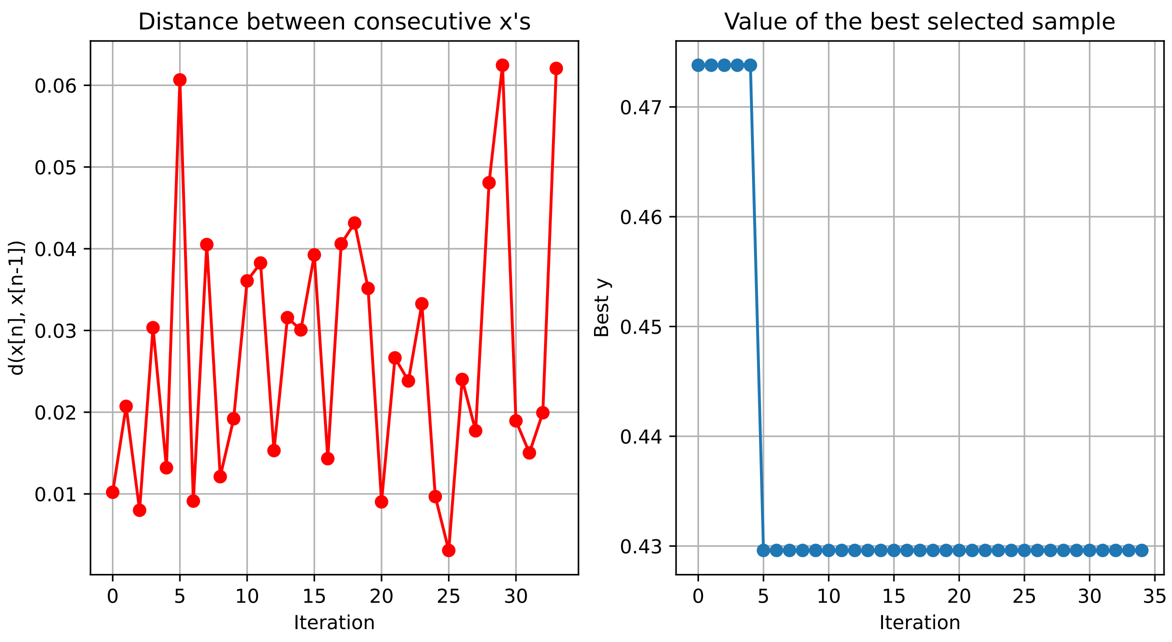
This plots the convergence of the optimization process, showing the best objective function value found so far at each iteration.

Full Python Code

```
import numpy as np
import GPyOpt
from GPyOpt.methods import BayesianOptimization

# Simplified objective function: lift-to-drag ratio of an airfoil
```

```python
# Note: In practice, this would be replaced with a CFD simulation
# or experimental evaluation.

# For this assignment, you will use Xfoil to generate L/D for a
# given thickness and camber.
# Python interface to Xfoil is available here:
#   https://pypi.org/project/xfoil/.

def objective_function(x):
    thickness, camber = x[:, 0], x[:, 1]
    # Simplified model for demonstration: assume the lift-to-drag ratio is
    # some function of thickness and camber.
    # This is just a placeholder and should be replaced with a realistic
    # aerodynamic model.
    lift_to_drag_ratio = (4*thickness - thickness**2 + 2*camber - camber**2)
    return lift_to_drag_ratio[:, None]

# Bounds of the design variables (e.g., thickness-to-chord ratio and camber)
bounds = [{'name': 'thickness', 'type': 'continuous', 'domain': (0.1, 0.18)},
          {'name': 'camber', 'type': 'continuous', 'domain': (0.02, 0.05)}]

# Optimization domain
domain = bounds

# Create a Bayesian Optimization object
optimizer = BayesianOptimization(f=objective_function,
                                 domain=domain,
                                 model_type='GP',
                                 acquisition_type='EI',  # Expected Improvement
                                 acquisition_jitter=0.05,
                                 exact_feval=True,
                                 maximize=False)

# Number of initial points and subsequent evaluations
initial_design_numdata = 10
max_iter = 30

# Run the optimization
optimizer.run_optimization(max_iter=max_iter, verbosity=True)

# Print the optimal design
print("Optimal design:", optimizer.x_opt)
print("Optimal lift-to-drag ratio:", optimizer.fx_opt)

# Plot convergence
optimizer.plot_convergence()
```

```
num acquisition: 1, time elapsed: 0.11s
num acquisition: 2, time elapsed: 0.34s
num acquisition: 3, time elapsed: 0.53s
num acquisition: 4, time elapsed: 0.80s
num acquisition: 5, time elapsed: 1.14s
num acquisition: 6, time elapsed: 1.74s
num acquisition: 7, time elapsed: 2.25s
num acquisition: 8, time elapsed: 2.69s
num acquisition: 9, time elapsed: 3.47s
num acquisition: 10, time elapsed: 4.08s
num acquisition: 11, time elapsed: 4.87s
num acquisition: 12, time elapsed: 5.53s
num acquisition: 13, time elapsed: 6.08s
num acquisition: 14, time elapsed: 6.63s
num acquisition: 15, time elapsed: 7.14s
num acquisition: 16, time elapsed: 7.66s
num acquisition: 17, time elapsed: 8.27s
num acquisition: 18, time elapsed: 8.70s
num acquisition: 19, time elapsed: 9.20s
num acquisition: 20, time elapsed: 9.67s
num acquisition: 21, time elapsed: 10.23s
num acquisition: 22, time elapsed: 10.77s
num acquisition: 23, time elapsed: 11.29s
num acquisition: 24, time elapsed: 11.84s
num acquisition: 25, time elapsed: 12.32s
num acquisition: 26, time elapsed: 12.82s
num acquisition: 27, time elapsed: 13.31s
num acquisition: 28, time elapsed: 13.80s
num acquisition: 29, time elapsed: 14.40s
num acquisition: 30, time elapsed: 14.89s
Optimal design: [0.1  0.02]
Optimal lift-to-drag ratio: 0.4296
```



Mathematical Background

- Gaussian Processes (GP) are used to model the unknown objective function. A GP is defined by a mean function (often assumed to be zero) and a covariance function (kernel), which encodes assumptions about the smoothness and variability of the function.
- The Expected Improvement (EI) acquisition function is used to balance exploration (sampling regions with high uncertainty) and exploitation (sampling regions with high predicted performance). Mathematically, EI is defined as:

$$EI(x) = \mathbb{E}\left[\max(f(x) - f(x^+), 0)\right]$$

where $f(x)$ is the GP prediction at point x, and $f(x^+)$ is the value of the best design found so far.

- Bayesian optimization iteratively selects new designs to evaluate by maximizing the acquisition function, updates the GP model with the new data, and repeats this process until convergence criteria are met.