Mathematical Optimization Framework

1 Introduction

This document presents a Python implementation of a mathematical optimization framework. The framework includes classes for defining generic mathematical functions and a Uzawa solver for solving constrained optimization problems.

2 Code Listings

2.1 Function Class

```
import numpy as np
import matplotlib.pyplot as plt
from typing import Callable, List

class Function(object):
    def __init__(self, compute: Callable, gradient: Callable, hessian: Callable,
        ↪ constraints=[]) -> None:
        self.compute = compute
        self.gradient = gradient
        self.hessian = hessian
        self.constraints = constraints

    def __add__(self, other):
        # ... (__add__ method code)

    def __rmul__(self, value):
        # ... (__rmul__ method code)

    def __getitem__(self, x):
        return self.compute(x)

    def __compute__(self, x):
        # ... (__compute__ method code)

    def __gradient__(self, x):
        # ... (__gradient__ method code)

    def __hessian__(self, x):
        # ... (__hessian__ method code)

    def add_constraint(self, f: 'Function'):
        self.constraints.append(f)

    def plot_contour(self):
        # ... (plot_contour method code)

    def plot_as_constraint(self):
```

```python
        print('hello')


class UzawaPlotter(object):
    def __init__(self, solver: "UzawaSolver"):
        # ... (UzawaPlotter class code)

    def plot(self, plot_f=True, plot_constraints=True, plot_gradients=True):
        # ... (plot method code)

    def plot_lambda_history(self):
        # ... (plot_lambda_history method code)

    def plot_f_increments(self, use_log=False):
        # ... (plot_f_increments method code)

    def plot_lagrangian_increments(self, use_log=False):
        # ... (plot_lagrangian_increments method code)

    def plot_tau_history(self):
        # ... (plot_tau_history method code)

class GradientDescent(object):
    def __init__(self, f) -> None:
        # ... (GradientDescent class code)

    def continue_condition(self, current_iteration, max_iteration, current_norm, epsilon, use_epsilon):
        # ... (continue_condition method code)

    def update_x(self, current_x, current_gradient, alpha):
        # ... (update_x method code)

    def solve(self, x0: np.array, max_iter=50, alpha=0.1, epsilon=0.01, use_epsilon=False):
        # ... (solve method code)

class MyFunction(object):
    def __init__(self, lambda_) -> None:
        self.lambda_ = lambda_

    def generate(self, f):
        return

class UzawaSolver(object):
    def __init__(self, f: Function, constraints):
        # ... (UzawaSolver class code)

    def solve_min(self, x0_internal, _lambda: np.array, tau=0.01, alpha=0.01,
                  max_iter=50, use_epsilon=False, epsilon=0.01, decay_type=None, decay_param=None):
        # ... (solve_min method code)
```

3 Conclusion

This mathematical optimization framework provides a versatile set of tools for solving optimization problems, including functions, constraints, and a Uzawa solver for constrained optimization.