Regime Switching in Quadratic Programming Optimization Using CVXPY

May 20, 2025

Abstract

This document provides a comprehensive explanation and implementation of regime switching in quadratic programming (QP) problems using the CVXPY optimization framework. The regime switching mechanism is incorporated into the objective function, allowing different quadratic objectives to be selected dynamically based on binary regime indicators. We explore binary variable modeling, Big-M reformulation, multi-time-step portfolio weight regime switching, and vectorization techniques to make the approach efficient. Detailed Python examples illustrate each concept.

Contents

1 Introduction

Quadratic Programming (QP) is a foundational technique in optimization where the objective function is quadratic and the constraints are linear. In many practical applications, such as portfolio optimization, the problem environment may switch between different regimes — e.g., bull and bear markets — that affect the objective function.

In this document, we explore how to incorporate **regime switching** directly into the QP formulation using CVXPY, a popular Python convex optimization library. We will cover how to model binary regime variables, how to handle regime-dependent objectives, and how to extend this to portfolio weights that change over time or asset-wise.

2 Basic Regime Switching Idea in QP

Suppose we have two regimes with distinct quadratic objective functions:

$$\text{Regime 1 Objective: } f_1(\mathbf{x}) = \mathbf{x}^T Q_1 \mathbf{x} + \mathbf{c}_1^T \mathbf{x} \tag{1}$$

$$\text{Regime 2 Objective: } f_2(\mathbf{x}) = \mathbf{x}^T Q_2 \mathbf{x} + \mathbf{c}_2^T \mathbf{x} \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the decision variable, $Q_1, Q_2 \in \mathbb{R}^{n \times n}$ are positive semidefinite matrices, and $\mathbf{c}_1, \mathbf{c}_2 \in \mathbb{R}^n$ are coefficient vectors.

We introduce a binary variable

$$z \in \{0, 1\}$$

to select the regime: $z = 0$ selects regime 1 and $z = 1$ selects regime 2.

2.1 Challenge: Mixed-Integer QP

CVXPY supports convex problems but to include binary variables, the problem becomes a *Mixed-Integer Quadratic Program* (MIQP). Solvers like GUROBI, CPLEX, or ECOS_BB are needed.

2.2 Big-M Reformulation

We use the Big-M method to model regime switching in the objective. Define an auxiliary variable t and add constraints:

$$t \geq f_1(\mathbf{x}) - Mz \tag{3}$$

$$t \geq f_2(\mathbf{x}) - M(1 - z) \tag{4}$$

where M is a sufficiently large constant.

When $z = 0$, the first constraint is $t \geq f_1(\mathbf{x})$ and the second is relaxed due to $-M(1 - 0) = -M$. Minimizing t effectively minimizes f_1.

Similarly, for $z = 1$, $t \geq f_2(\mathbf{x})$ is enforced.

2.3 Example Code

```
import cvxpy as cp
import numpy as np

n = 3
x = cp.Variable(n)
z = cp.Variable(boolean=True)

Q1 = np.eye(n)
Q2 = 2 * np.eye(n)
c1 = np.array([1, 2, 3])
c2 = np.array([-1, -1, -1])

M = 1000
obj1 = cp.quad_form(x, Q1) + c1 @ x
obj2 = cp.quad_form(x, Q2) + c2 @ x
t = cp.Variable()

constraints = [
    t >= obj1 - M * z,
    t >= obj2 - M * (1 - z),
    x >= 0,
    cp.sum(x) == 1
]

problem = cp.Problem(cp.Minimize(t), constraints)
problem.solve(solver=cp.GUROBI)

print("Optimal x:", x.value)
print("Regime:", int(z.value))
print("Objective value:", problem.value)
```

Listing 1: Basic regime switching in CVXPY

3 Why Use Auxiliary Variable t?

The auxiliary variable t acts as an upper bound on the selected regime's objective. By minimizing t, the solver is forced to pick the regime with the lower objective value and enforce its corresponding constraint.

3.1 Alternative (Non-DCP Compliant) Formulation

One might try:

$$\min \sum_i z_i \cdot f_i(\mathbf{x}) \tag{5}$$

where z_i are binaries. But this involves product of binaries and convex functions, violating CVXPY's Disciplined Convex Programming (DCP) rules. The Big-M approach is preferred.

4 Regime Switching Based on Portfolio Weights Over Time

Suppose portfolio weights are $\mathbf{W} \in \mathbb{R}^{n \times T}$ over T time steps.

Define regime switching at each time t by:

$$\text{Regime 1 if } \|\mathbf{W}_t\|_1 \leq \tau, \quad \text{Regime 2 otherwise}$$

where $\mathbf{W}_t$ is the t-th column of $\mathbf{W}$ and τ is a threshold.

Introduce binary variables $\mathbf{z} \in \{0,1\}^T$ with $z_t = 1$ indicating regime 2 at time t.

4.1 Modeling in CVXPY

Use the Big-M trick and constraints:

$$\begin{aligned}
t_t &\geq f_1(\mathbf{W}_t) - M z_t && (6) \\
t_t &\geq f_2(\mathbf{W}_t) - M(1 - z_t) && (7) \\
\|\mathbf{W}_t\|_1 - \tau &\leq M z_t && (8) \\
\|\mathbf{W}_t\|_1 - \tau &\geq -M(1 - z_t) && (9)
\end{aligned}$$

and optimize sum of t_t over t.

4.2 Code Example

```python
import cvxpy as cp
import numpy as np

n, T = 3, 4
tau = 0.6
M = 1e3

W = cp.Variable((n, T))
z = cp.Variable(T, boolean=True)

Q1 = np.eye(n)
Q2 = 2 * np.eye(n)
c1 = np.array([0.1, 0.2, 0.3])
c2 = np.array([-0.1, -0.2, -0.1])

constraints = []
regime_obj = 0

for t in range(T):
    w_t = W[:, t]
    obj1 = cp.quad_form(w_t, Q1) - c1 @ w_t
    obj2 = cp.quad_form(w_t, Q2) - c2 @ w_t
    t_var = cp.Variable()

    constraints += [
        t_var >= obj1 - M * z[t],
        t_var >= obj2 - M * (1 - z[t]),
        cp.norm1(w_t) - tau <= M * z[t],
```

```
        cp.norm1(w_t) - tau >= -M * (1 - z[t]),
        cp.sum(w_t) == 1,
        w_t >= 0
    ]

    regime_obj += t_var

prob = cp.Problem(cp.Minimize(regime_obj), constraints)
prob.solve(solver=cp.GUROBI)

print("Optimal Weights:\n", W.value)
print("Regimes:\n", z.value)
```

Listing 2: Regime switching over time based on portfolio weights

5 Vectorized Regime Switching Model

Consider portfolio weights $W \in \mathbb{R}^{n \times T}$ over T time steps, and binary regime indicators $z \in \{0, 1\}^T$.

We define the auxiliary variables $t \in \mathbb{R}^T$ for each time step and use the following Big-M constraints in vectorized form:

$$t \geq f_1(W_t) - M z_t, \quad \forall t = 1, \dots, T \tag{10}$$

$$t \geq f_2(W_t) - M(1 - z_t), \quad \forall t = 1, \dots, T \tag{11}$$

$$\|W_t\|_1 \leq \tau + M z_t, \quad \forall t = 1, \dots, T \tag{12}$$

$$\|W_t\|_1 \geq \tau - M(1 - z_t), \quad \forall t = 1, \dots, T \tag{13}$$

$$\sum_{i=1}^{n} W_{i,t} = 1, \quad \forall t = 1, \dots, T \tag{14}$$

$$W_{i,t} \geq 0, \quad \forall i = 1, \dots, n, \ t = 1, \dots, T \tag{15}$$

where the quadratic forms f_1 and f_2 are defined as:

$$f_j(W_t) = W_t^\top Q_j W_t + c_j^\top W_t, \quad j = 1, 2.$$

—

5.1 Vectorized CVXPY Implementation

```
import cvxpy as cp
import numpy as np

n, T = 3, 4
tau = 0.6
M = 1e3

W = cp.Variable((n, T))
z = cp.Variable(T, boolean=True)
t = cp.Variable(T)

Q1 = np.eye(n)
```

```python
Q2 = 2 * np.eye(n)
c1 = np.array([0.1, 0.2, 0.3])
c2 = np.array([-0.1, -0.2, -0.1])

def batch_quad_form(W, Q):
    # Computes quadratic forms w_t^T Q w_t for each time t
    return cp.sum(cp.multiply(Q @ W, W), axis=0)

qf1 = batch_quad_form(W, Q1)
qf2 = batch_quad_form(W, Q2)

obj1 = qf1 + c1 @ W
obj2 = qf2 + c2 @ W

constraints = [
    t >= obj1 - M * z,
    t >= obj2 - M * (1 - z),
    cp.norm1(W) <= tau + M * z,
    cp.norm1(W) >= tau - M * (1 - z),
    cp.sum(W, axis=0) == 1,
    W >= 0
]

prob = cp.Problem(cp.Minimize(cp.sum(t)), constraints)
prob.solve(solver=cp.GUROBI)

print("Optimal Weights:\n", W.value)
print("Regimes:\n", z.value)
```

Listing 3: Vectorized regime switching using CVXPY

—

This vectorized form allows efficient modeling of regime switching across multiple time steps without explicit Python loops, leveraging CVXPY's elementwise operations.

6 Vectorized Regime Switching with Four Regimes

Suppose we have $K = 4$ regimes, each with its own quadratic objective:

$$f_k(W_t) = W_t^\top Q_k W_t + c_k^\top W_t, \quad k = 1, 2, 3, 4$$

Define binary indicator variables $z_{k,t} \in \{0, 1\}$ for each regime and time step, with the constraint that exactly one regime is active at each time:

$$\sum_{k=1}^{4} z_{k,t} = 1, \quad \forall t = 1, \dots, T.$$

Introduce auxiliary variables t_t representing the selected regime objective at time t. The Big-M constraints are:

$$t_t \geq f_k(W_t) - M(1 - z_{k,t}), \quad k = 1, \ldots, 4, \; t = 1, \ldots, T \quad (16)$$

$$\sum_{k=1}^{4} z_{k,t} = 1, \quad t = 1, \ldots, T \quad (17)$$

Additional constraints on W_t as required. (18)

—

6.1 CVXPY Code for 4-Regime Switching

```
import cvxpy as cp
import numpy as np

n, T = 3, 5
K = 4  # Number of regimes
M = 1e4

W = cp.Variable((n, T))
z = cp.Variable((K, T), boolean=True)  # One-hot regime indicators
t = cp.Variable(T)

# Define Q_k and c_k for each regime k
Qs = [np.eye(n),
      2 * np.eye(n),
      3 * np.eye(n),
      4 * np.eye(n)]

cs = [np.array([0.1, 0.2, 0.3]),
      np.array([-0.1, -0.2, -0.1]),
      np.array([0.05, 0.05, 0.05]),
      np.array([-0.05, -0.1, -0.2])]

def batch_quad_form(W, Q):
    return cp.sum(cp.multiply(Q @ W, W), axis=0)

# Compute all objectives f_k(W_t) for all k and t
obj = []
for k in range(K):
    qf = batch_quad_form(W, Qs[k])
    lin = cs[k] @ W
    obj.append(qf + lin)  # shape: (T,)

obj = cp.vstack(obj)  # shape: (K, T)

constraints = []

# Big-M constraints for each regime and time step
for k in range(K):
    constraints += [t >= obj[k, :] - M * (1 - z[k, :])]

# Exactly one regime active at each time
constraints += [cp.sum(z, axis=0) == 1]

```

```
# Additional constraints on W (example: weights sum to 1 and
    nonnegative)
constraints += [
    cp.sum(W, axis=0) == 1,
    W >= 0
]

prob = cp.Problem(cp.Minimize(cp.sum(t)), constraints)
prob.solve(solver=cp.GUROBI)

print("Optimal Weights:\n", W.value)
print("Active regimes (z):\n", z.value)
```

Listing 4: 4-Regime regime switching vectorized in CVXPY

—

This approach models multiple regimes cleanly by enforcing a one-hot regime vector at each time and using Big-M constraints to pick the active regime objective.