Risk-Averse MPC for Wine Fermentation:

CVaR Minimization under Parametric Uncertainty

Based on analysis of `MPC_Freddo_CVaR_Cellar_Temp.ipynb`

January 22, 2026

Abstract

We formulate a risk-averse optimal control problem for the Oenoferm Freddo fermentation process. The control objective is to minimize the Conditional Value-at-Risk (CVaR) of the tracking error and energy cost under uncertain ambient cellar temperatures. We utilize the Rockafellar-Uryasev formulation to cast the stochastic problem as a deterministic Nonlinear Program (NLP), solvable directly via CasADi and IPOPT. This method ensures robust performance across a uniform distribution of environmental conditions, specifically protecting against the "tail risk" of stuck fermentations in cold weather.

1 Model Description

The underlying system describes the fermentation kinetics of yeast in a temperature-controlled tank. The dynamics are defined by the state vector $x = [X, N, E, S, O, T]^\top$ (concentrations of Yeast, Nitrogen, Ethanol, Sugar, Oxygen, and Temperature).

The system evolves according to parametric ODEs:

$$\dot{x}(t) = f(x(t), u(t), \theta) \tag{1}$$

where $u(t) \in [0, 1]$ is the cooling control input and θ represents the uncertain cellar temperature parameter (p_{17}).

1.1 Uncertainty Modeling

The uncertainty is modeled as a random parameter θ following a uniform distribution:

$$\theta \sim \mathcal{U}[10.0, 19.0] \quad (^\circ\text{C}) \tag{2}$$

The Deterministic MPC ignores this distribution, assuming a nominal $\bar{\theta} = 14.5^\circ$C. The CVaR MPC explicitly accounts for the distribution by sampling $N_{scen} = 50$ scenarios.

2 CVaR Formulation

To ensure robustness, we minimize the Conditional Value-at-Risk (CVaR) at confidence level $\alpha = 0.05$. This metric corresponds to the expected cost of the worst 5% of scenarios.

2.1 Optimization Problem

Using the sample average approximation and the variational definition of CVaR, the problem is formulated as:

$$\min_{U, \eta, \{z_s\}} \quad \eta + \frac{1}{\alpha N_{scen}} \sum_{s=1}^{N_{scen}} z_s \tag{3}$$

Subject to:

$$x_{k+1}^{(s)} = F_{\text{RK4}}(x_k^{(s)}, u_k, \theta_s) \qquad \forall s \in \{1, \dots, N_{\text{scen}}\},\ k \in \{0, \dots, N-1\} \tag{4}$$

$$J^{(s)}(U) = \sum_{k=0}^{N-1} \left[\lambda_u u_k + \lambda_S \left(S_k^{(s)} - S_{\text{ref}} \right)^2 + \lambda_r \left(S_k^{(s)} - S_k^{\text{ref}} \right)^2 \right] \qquad \forall s \in \{1, \dots, N_{\text{scen}}\} \tag{5}$$

$$z_s \geq J^{(s)}(U) - \eta \qquad \forall s \in \{1, \dots, N_{\text{scen}}\} \tag{6}$$

$$z_s \geq 0 \qquad \forall s \in \{1, \dots, N_{\text{scen}}\} \tag{7}$$

$$x_{\min} \leq x_k^{(s)} \leq x_{\max} \qquad \forall s \in \{1, \dots, N_{\text{scen}}\},\ k \in \{0, \dots, N\} \tag{8}$$

Where:

- $U = \{u_0, \dots, u_{N-1}\}$ is the single control trajectory applied robustly to all scenarios.
- η is the Value-at-Risk (VaR) auxiliary variable (the $(1-\alpha)$-quantile).
- z_s represents the positive deviation of the cost of scenario s from the VaR η (the tail loss).
- $J^{(s)}(U)$ is the **total cost** for scenario s over the prediction horizon, which includes:
  - $\lambda_u = p_{18} = 2.0$: control effort penalty weight
  - $\lambda_S = 1.0$: terminal tracking error weight
  - $\lambda_r = 0.01$: reference trajectory tracking weight
  - $S_k^{ref} = S_0 + (S_{ref} - S_0)\frac{k}{N}$: linear reference profile
- $\theta_s \sim \mathcal{U}[25, 30]$ is the sampled cellar temperature for scenario s.
- $\alpha = 0.05$ (confidence level), $N_{scen} = 50$ (number of scenarios), $N = 21$ (prediction horizon).

3 Implementation in CasADi

The following listing illustrates the specific implementation of the cost function and constraints found in the notebook.

```
# Decision variables
U_cvar = SX.sym('U_cvar', n_controls, n)
eta = SX.sym('eta')           # Value at Risk
z = SX.sym('z', n_scenarios)  # Tail loss variables

costs_cvar = []
g_cvar = []

# Scenario Simulation Loop
for s in range(n_scenarios):
    # Propagate dynamics for this specific temperature T_s
    # ... integration code ...

    # Calculate Cost for this scenario
    cost_s = sum(tracking_error + control_penalty)
    costs_cvar.append(cost_s)

    # Enforce constraints for this scenario
    g_cvar.extend([state_constraints])

# CVaR Objective (Rockafellar-Uryasev)
alpha = 0.05
cvar_obj = eta + (1.0 / (alpha * n_scenarios)) * sum1(z)

# CVaR Constraints
for s in range(n_scenarios):
```

```
    # z[s] must be greater than (Cost - VaR)
    g_cvar.append(z[s] - (costs_cvar[s] - eta))

# Solve NLP
nlp = {'x': vertcat(U_cvar, eta, z), 'f': cvar_obj, 'g': vertcat(*g_cvar)}
solver = nlpsol('solver', 'ipopt', nlp)
```

Listing 1: Python/CasADi Implementation Code

4 Conclusion

The CVaR formulation effectively transforms the stochastic control problem into a deterministic NLP. By penalizing the tail of the cost distribution, the controller adopts a conservative cooling strategy. This prevents the "stuck fermentation" failure mode common in cold scenarios ($10 - 12°C$), which the risk-neutral deterministic controller fails to anticipate.