ATRP Polymerization Optimization with Folio

Multi-objective Bayesian optimization for polymer synthesis

Folio Demo

2026-01-12

Table of contents

1 Overview

This notebook demonstrates Folio for optimizing **Atom Transfer Radical Polymerization (ATRP)** conditions.

Optimization Goal

Find reaction conditions that produce high molecular weight polymers with narrow dispersity.

1.1 Input Variables (Reaction Stoichiometry)

Variable	Range	Units	Description
HO-EBiB	0.5–2.0	equiv	Initiator
MB	50–200	equiv	Methyl methacrylate monomer
CuBr /L	0.01–0.1	equiv	Catalyst complex
TEOA	0.1–1.0	equiv	Reducing agent

1.2 Output Variables

- **Conversion** (%): Monomer conversion by NMR
- **Mn_app** (g/mol): Number-average molecular weight from GPC
- **Dispersity** (Đ = Mw/Mn): Polydispersity index

1.3 Optimization Objectives

- **Maximize** Mn_app (higher molecular weight)
- **Minimize** Dispersity (narrower distribution)

2 Setup

```
import tempfile
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from folio.api import Folio
from folio.core.config import RecommenderConfig, TargetConfig
from folio.core.schema import InputSpec, OutputSpec

# Create a temporary database
db_path = tempfile.NamedTemporaryFile(suffix=".db", delete=False).name
folio = Folio(db_path=db_path)
print(f"Database: {db_path}")
```

```
Database: /var/folders/h9/gpnygt8x227c24jhrwxd23pr0000gn/T/tmpah0b0x04.db
```

3 Create the ATRP Project

```
# Input specifications
inputs = [
    InputSpec("HO_EBiB", "continuous", bounds=(0.5, 2.0), units="equiv"),
    InputSpec("MB", "continuous", bounds=(50.0, 200.0), units="equiv"),
    InputSpec("CuBr2_L", "continuous", bounds=(0.01, 0.1), units="equiv"),
    InputSpec("TEOA", "continuous", bounds=(0.1, 1.0), units="equiv"),
]

# Output specifications
outputs = [
    OutputSpec("conversion", units="%"),
    OutputSpec("Mn_app", units="g/mol"),
    OutputSpec("dispersity"),
]

# Multi-objective targets
target_configs = [
    TargetConfig(objective="Mn_app", objective_mode="maximize"),
    TargetConfig(objective="dispersity", objective_mode="minimize"),
]

# Reference point for hypervolume (worst acceptable values)
reference_point = [5000.0, 2.0]

# Create project with multi-task GP
folio.create_project(
    name="atrp_optimization",
    inputs=inputs,
    outputs=outputs,
    target_configs=target_configs,
    reference_point=reference_point,
    recommender_config=RecommenderConfig(
        type="bayesian",
        surrogate="multitask_gp",
        mo_acquisition="nehvi",
        n_initial=5,
    ),
)
print("Project 'atrp_optimization' created!")
```

```
Project 'atrp_optimization' created!
```

Project Configuration

The multi-task GP models correlations between objectives (Mn and dispersity), which improves predictions when objectives are related.

4 Add Screening Data

Initial experiments exploring the parameter space:

```
screening_data = [
    # (HO_EBiB, MB, CuBr2_L, TEOA, conversion, Mn_app, dispersity, notes)
    (1.0, 100.0, 0.05, 0.5, 78.5, 18500, 1.25, "Standard conditions, good control"),
    (1.0, 150.0, 0.05, 0.5, 72.3, 26800, 1.32, "Higher monomer, increased Mn"),
    (1.0, 200.0, 0.05, 0.5, 65.1, 32100, 1.45, "High monomer loading, broader PDI"),
    (0.5, 100.0, 0.05, 0.5, 85.2, 35200, 1.38, "Low initiator, high Mn achieved"),
    (2.0, 100.0, 0.05, 0.5, 91.5, 9800, 1.18, "High initiator, low Mn but narrow PDI"),
    (1.0, 100.0, 0.01, 0.5, 45.2, 12300, 1.65, "Low catalyst, incomplete conversion"),
    (1.0, 100.0, 0.1, 0.5, 92.8, 19200, 1.22, "High catalyst, fast reaction"),
    (1.0, 100.0, 0.05, 0.1, 52.1, 14500, 1.72, "Low reducing agent, poor control"),
    (1.0, 100.0, 0.05, 1.0, 88.9, 20100, 1.28, "High TEOA, good conversion"),
    (0.5, 150.0, 0.08, 0.7, 76.4, 42500, 1.35, "Promising high-Mn conditions"),
    (0.75, 175.0, 0.06, 0.6, 71.8, 38900, 1.42, None),
    (1.5, 75.0, 0.04, 0.4, 82.3, 11200, 1.21, "Low Mn target, excellent control"),
]

for ho_ebib, mb, cubr2, teoa, conv, mn, disp, notes in screening_data:
    folio.add_observation(
        project_name="atrp_optimization",
        inputs={"HO_EBiB": ho_ebib, "MB": mb, "CuBr2_L": cubr2, "TEOA": teoa},
        outputs={"conversion": conv, "Mn_app": mn, "dispersity": disp},
        tag="screening",
        notes=notes,
    )

# Add edge cases
folio.add_observation(
    project_name="atrp_optimization",
    inputs={"HO_EBiB": 0.5, "MB": 200.0, "CuBr2_L": 0.02, "TEOA": 0.2},
    outputs={"conversion": 35.8, "Mn_app": 28500, "dispersity": 1.85},
    tag="screening",
    notes="Gel formed at high conversion, stopped early",
)

folio.add_observation(
    project_name="atrp_optimization",
```

```
    inputs={"HO_EBiB": 1.0, "MB": 50.0, "CuBr2_L": 0.03, "TEOA": 0.3},
    outputs={"conversion": 95.2, "Mn_app": 8900, "dispersity": 1.15},
    tag="screening",
    notes="Low monomer, very narrow dispersity",
)

print(f"Added {len(folio.get_observations('atrp_optimization'))} screening observations")
```

```
Added 14 screening observations
```

5 Run Optimization Loop

5.1 Automated Loop

```
def simulate_atrp(inputs: dict) -> dict:
    """Simulate ATRP results (for demo purposes)."""
    ho_ebib = inputs["HO_EBiB"]
    mb = inputs["MB"]
    cubr2 = inputs["CuBr2_L"]
    teoa = inputs["TEOA"]

    target_dp = mb / ho_ebib
    mn_base = target_dp * 100

    conv = 50 + 40 * cubr2 / 0.1 * teoa / 1.0
    conv = min(95, conv) + np.random.normal(0, 2)

    mn = mn_base * (conv / 100) + np.random.normal(0, 1000)
    mn = max(5000, mn)

    disp = 1.05 + 0.3 * (1 - cubr2 / 0.1) + 0.2 * (1 - teoa / 1.0)
    disp += 0.1 * (mb / 200)
    disp += np.random.normal(0, 0.03)
    disp = max(1.05, min(2.0, disp))

    return {
        "conversion": round(conv, 1),
        "Mn_app": round(mn, 0),
        "dispersity": round(disp, 2),
    }

# Run optimization
print("Optimization iterations:")
for i in range(5):
    suggestion = folio.suggest("atrp_optimization")[0]
```

```
    results = simulate_atrp(suggestion)

    folio.add_observation(
        project_name="atrp_optimization",
        inputs=suggestion,
        outputs=results,
        tag="optimization",
        notes=f"BO iteration {i+1}",
    )

    print(f"Iter {i+1}: MB={suggestion['MB']:.1f}, HO-EBiB={suggestion['HO_EBiB']:.2f} "
          f"→ Mn={results['Mn_app']:.0f}, Đ={results['dispersity']:.2f}")
```

```
Optimization iterations:

Iter 1: MB=200.0, HO-EBiB=0.50 → Mn=27277, Đ=1.33
Iter 2: MB=129.9, HO-EBiB=0.51 → Mn=16858, Đ=1.32
Iter 3: MB=199.8, HO-EBiB=0.50 → Mn=27803, Đ=1.29
Iter 4: MB=160.3, HO-EBiB=0.52 → Mn=21911, Đ=1.26
Iter 5: MB=117.3, HO-EBiB=0.50 → Mn=16339, Đ=1.19
```

5.2 Manual Entry Workflow

In practice, you enter data after each experiment:

```
# Get next suggestion
next_exp = folio.suggest("atrp_optimization")[0]
print("Suggested conditions:")
for k, v in next_exp.items():
    print(f"  {k}: {v:.4f}")

# Record result (simulated here)
folio.add_observation(
    project_name="atrp_optimization",
    inputs=next_exp,
    outputs={"conversion": 81.3, "Mn_app": 29800, "dispersity": 1.28},
    tag="manual_entry",
    notes="Reaction time: 4h at 60C under N2. GPC: THF, PS standards.",
)
print("\nObservation recorded!")
```

```
Suggested conditions:
  HO_EBiB: 0.5000
  MB: 182.4508
  CuBr2_L: 0.0333
```

```
  TEOA: 0.4943

Observation recorded!
```

6 Data Analysis

6.1 Export to DataFrame

```python
def observations_to_dataframe(observations: list) -> pd.DataFrame:
    """Convert observations to pandas DataFrame."""
    records = []
    for obs in observations:
        record = {"id": obs.id, "tag": obs.tag}
        record.update(obs.inputs)
        record.update(obs.outputs)
        records.append(record)
    return pd.DataFrame(records)

observations = folio.get_observations("atrp_optimization")
df = observations_to_dataframe(observations)
```

6.2 Summary Statistics

```python
numeric_cols = ["HO_EBiB", "MB", "CuBr2_L", "TEOA", "conversion", "Mn_app", "dispersity"]
df[numeric_cols].describe().round(2)
```

	HO_EBiB	MB	CuBr2_L	TEOA	conversion	Mn_app	dispersity
count	20.00	20.00	20.00	20.00	20.00	20.00	20.00
mean	0.84	134.49	0.05	0.53	73.39	22924.40	1.36
std	0.40	46.77	0.02	0.20	15.49	9793.38	0.19
min	0.50	50.00	0.01	0.10	35.80	8900.00	1.15
25%	0.50	100.00	0.05	0.50	68.68	15879.25	1.24
50%	0.88	123.58	0.05	0.50	74.00	21005.50	1.31
75%	1.00	176.86	0.07	0.68	83.02	28825.00	1.39
max	2.00	200.00	0.10	1.00	95.20	42500.00	1.85

6.3 Correlation Matrix

```
corr = df[numeric_cols].corr()

fig, ax = plt.subplots(figsize=(8, 6))
im = ax.imshow(corr, cmap="RdBu_r", vmin=-1, vmax=1)
ax.set_xticks(range(len(numeric_cols)))
ax.set_yticks(range(len(numeric_cols)))
ax.set_xticklabels(numeric_cols, rotation=45, ha="right")
ax.set_yticklabels(numeric_cols)
plt.colorbar(im, label="Correlation")
ax.set_title("Input-Output Correlation Matrix")
plt.tight_layout()
plt.show()
```

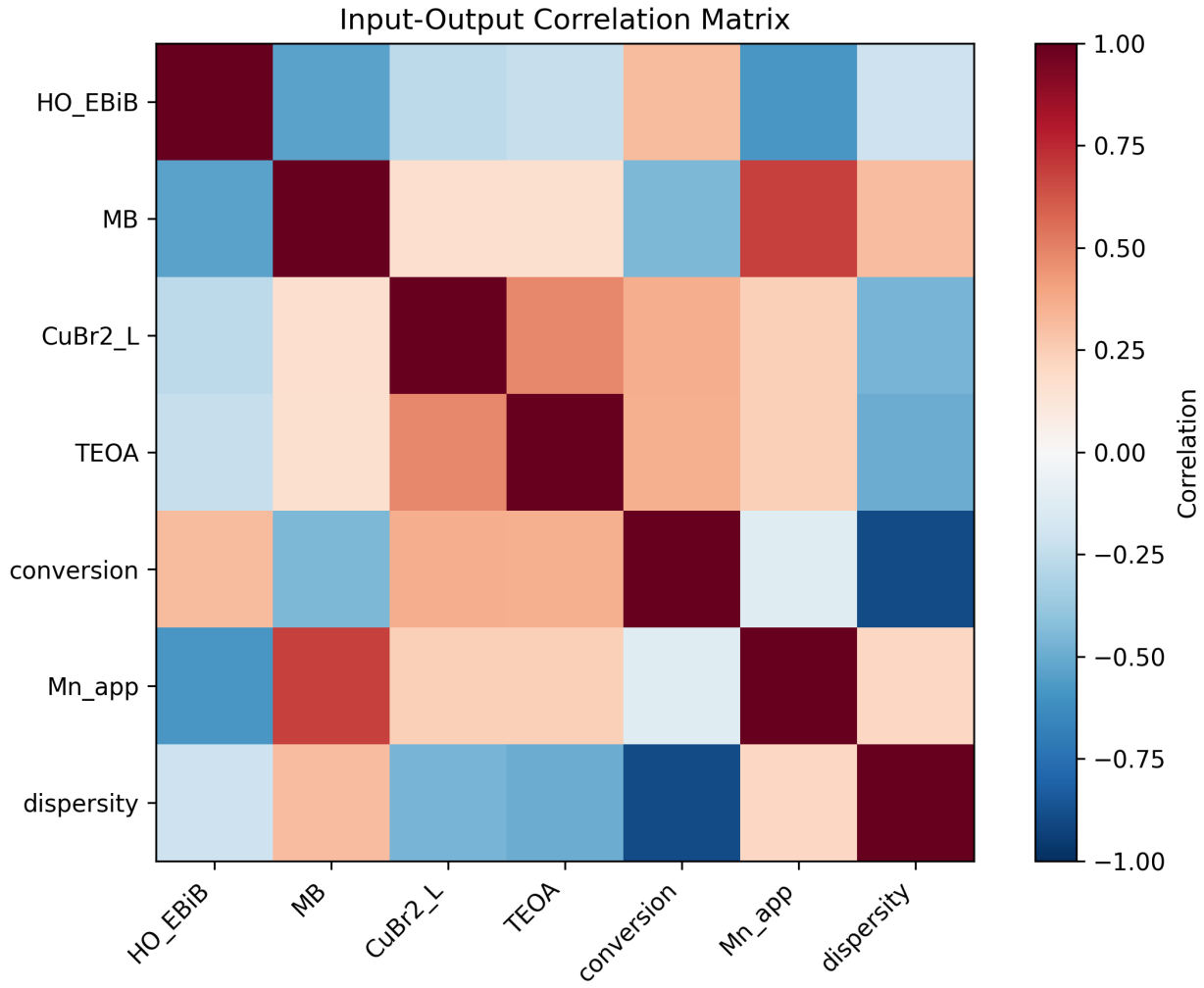

Figure 1: Input-output correlation matrix for ATRP experiments

6.4 Pairwise Relationships

```
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

ax = axes[0, 0]
sc = ax.scatter(df["MB"], df["Mn_app"], c=df["dispersity"], cmap="viridis", s=50)
ax.set_xlabel("MB (equiv)")
ax.set_ylabel("Mn_app (g/mol)")
ax.set_title("Monomer vs Molecular Weight")
plt.colorbar(sc, ax=ax, label="Dispersity")

ax = axes[0, 1]
ax.scatter(df["HO_EBiB"], df["Mn_app"], c=df["dispersity"], cmap="viridis", s=50)
ax.set_xlabel("HO-EBiB (equiv)")
ax.set_ylabel("Mn_app (g/mol)")
ax.set_title("Initiator vs Molecular Weight")

ax = axes[1, 0]
ax.scatter(df["CuBr2_L"], df["dispersity"], c=df["Mn_app"], cmap="plasma", s=50)
ax.set_xlabel("CuBr2/L (equiv)")
ax.set_ylabel("Dispersity (Đ)")
ax.set_title("Catalyst vs Dispersity")

ax = axes[1, 1]
ax.scatter(df["TEOA"], df["conversion"], c=df["Mn_app"], cmap="plasma", s=50)
ax.set_xlabel("TEOA (equiv)")
ax.set_ylabel("Conversion (%)")
ax.set_title("Reducing Agent vs Conversion")

plt.tight_layout()
plt.show()
```

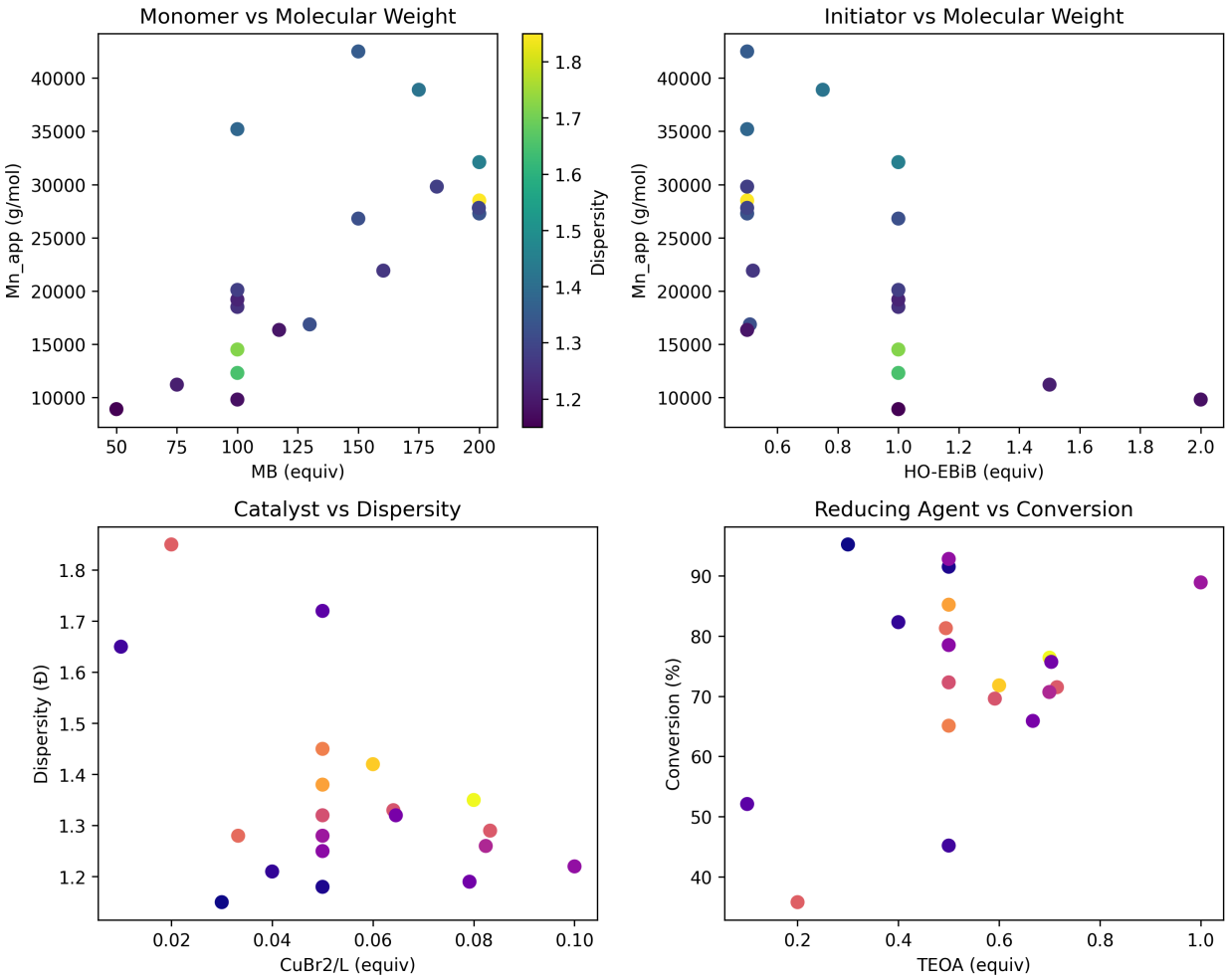

Figure 2: Pairwise plots showing relationships between inputs and outputs

7 GP Model Predictions

Visualize the trained Gaussian Process predictions and uncertainty:

```
# Ensure model is fitted
folio.suggest("atrp_optimization")
recommender = folio.get_recommender("atrp_optimization")
surrogate = recommender.surrogate

# Create prediction grid
n_grid = 20
mb_range = np.linspace(50, 200, n_grid)
ho_ebib_range = np.linspace(0.5, 2.0, n_grid)
MB_grid, HO_grid = np.meshgrid(mb_range, ho_ebib_range)

# Fixed values
```

```
cubr2_fixed, teoa_fixed = 0.05, 0.5

X_grid = np.column_stack([
    HO_grid.ravel(),
    MB_grid.ravel(),
    np.full(n_grid * n_grid, cubr2_fixed),
    np.full(n_grid * n_grid, teoa_fixed),
])

mean, std = surrogate.predict(X_grid)
```

```
mn_mean = mean[:, 0].reshape(n_grid, n_grid)
mn_std = std[:, 0].reshape(n_grid, n_grid)
disp_mean = mean[:, 1].reshape(n_grid, n_grid)
disp_std = std[:, 1].reshape(n_grid, n_grid)

fig, axes = plt.subplots(2, 2, figsize=(12, 10))

ax = axes[0, 0]
c = ax.contourf(MB_grid, HO_grid, mn_mean, levels=20, cmap="viridis")
plt.colorbar(c, ax=ax, label="Mn_app (g/mol)")
ax.scatter(df["MB"], df["HO_EBiB"], c="red", s=20, marker="x")
ax.set_xlabel("MB (equiv)")
ax.set_ylabel("HO-EBiB (equiv)")
ax.set_title("Predicted Mn_app (mean)")

ax = axes[0, 1]
c = ax.contourf(MB_grid, HO_grid, mn_std, levels=20, cmap="Oranges")
plt.colorbar(c, ax=ax, label="Std (g/mol)")
ax.scatter(df["MB"], df["HO_EBiB"], c="red", s=20, marker="x")
ax.set_xlabel("MB (equiv)")
ax.set_ylabel("HO-EBiB (equiv)")
ax.set_title("Mn_app Uncertainty")

ax = axes[1, 0]
c = ax.contourf(MB_grid, HO_grid, disp_mean, levels=20, cmap="viridis_r")
plt.colorbar(c, ax=ax, label="Dispersity")
ax.scatter(df["MB"], df["HO_EBiB"], c="red", s=20, marker="x")
ax.set_xlabel("MB (equiv)")
ax.set_ylabel("HO-EBiB (equiv)")
ax.set_title("Predicted Dispersity (mean)")

ax = axes[1, 1]
c = ax.contourf(MB_grid, HO_grid, disp_std, levels=20, cmap="Oranges")
plt.colorbar(c, ax=ax, label="Std")
ax.scatter(df["MB"], df["HO_EBiB"], c="red", s=20, marker="x")
ax.set_xlabel("MB (equiv)")
```

```
ax.set_ylabel("HO-EBiB (equiv)")
ax.set_title("Dispersity Uncertainty")

plt.suptitle(f"GP Predictions (CuBr2/L={cubr2_fixed}, TEOA={teoa_fixed} fixed)", y=1.02)
plt.tight_layout()
plt.show()
```

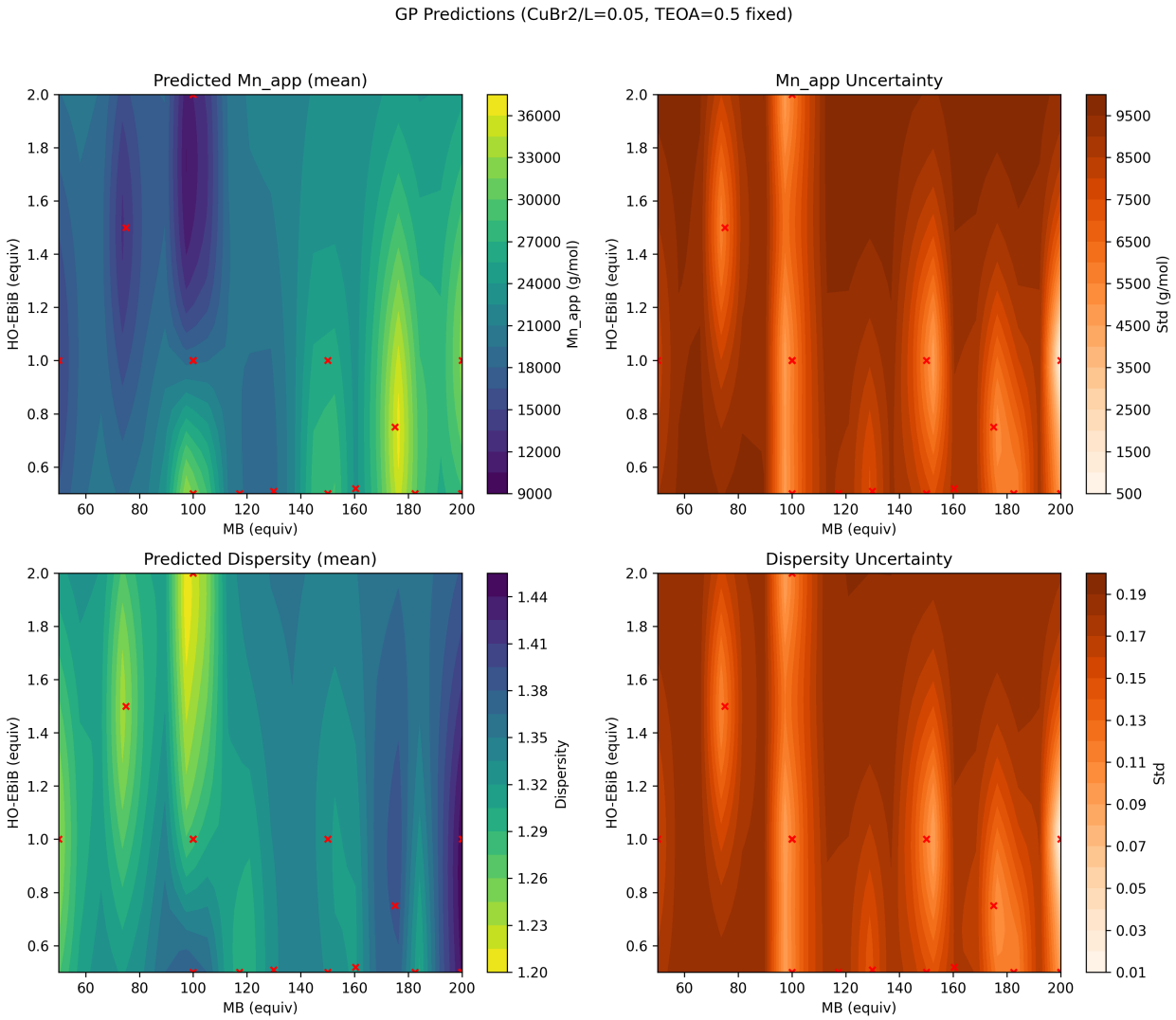

Figure 3: GP predictions for Mn and dispersity with uncertainty estimates

8 Pareto Front Analysis

Identify non-dominated solutions balancing Mn and dispersity:

```python
def find_pareto_optimal(mn_values, disp_values):
    """Find Pareto-optimal points (maximize Mn, minimize dispersity)."""
    n = len(mn_values)
    is_optimal = np.ones(n, dtype=bool)

    for i in range(n):
        for j in range(n):
            if i != j:
                if mn_values[j] >= mn_values[i] and disp_values[j] <= disp_values[i]:
                    if mn_values[j] > mn_values[i] or disp_values[j] < disp_values[i]:
                        is_optimal[i] = False
                        break
    return is_optimal

mn_values = df["Mn_app"].values
disp_values = df["dispersity"].values
pareto_mask = find_pareto_optimal(mn_values, disp_values)

print(f"Total observations: {len(df)}")
print(f"Pareto-optimal: {pareto_mask.sum()}")
```

```
Total observations: 20
Pareto-optimal: 7
```

```python
fig, ax = plt.subplots(figsize=(10, 7))

# Dominated points
ax.scatter(mn_values[~pareto_mask], disp_values[~pareto_mask],
           c="gray", alpha=0.5, s=60, label="Dominated")

# Pareto-optimal points
ax.scatter(mn_values[pareto_mask], disp_values[pareto_mask],
           c="red", s=120, marker="*", label="Pareto optimal", zorder=5)

# Connect Pareto front
pareto_mn = mn_values[pareto_mask]
pareto_disp = disp_values[pareto_mask]
sort_idx = np.argsort(pareto_mn)
ax.plot(pareto_mn[sort_idx], pareto_disp[sort_idx], "r--", alpha=0.5, linewidth=2)

# Reference point
ax.scatter([reference_point[0]], [reference_point[1]],
           c="black", marker="s", s=100, label="Reference point")
ax.axvline(reference_point[0], color="black", linestyle=":", alpha=0.3)
ax.axhline(reference_point[1], color="black", linestyle=":", alpha=0.3)
```

```
ax.set_xlabel("Mn_app (g/mol)", fontsize=12)
ax.set_ylabel("Dispersity (Đ)", fontsize=12)
ax.set_title("ATRP Optimization: Pareto Front\n(Maximize Mn, Minimize Đ)", fontsize=14)
ax.legend(loc="upper right")
ax.grid(True, alpha=0.3)
ax.invert_yaxis()

plt.tight_layout()
plt.show()
```

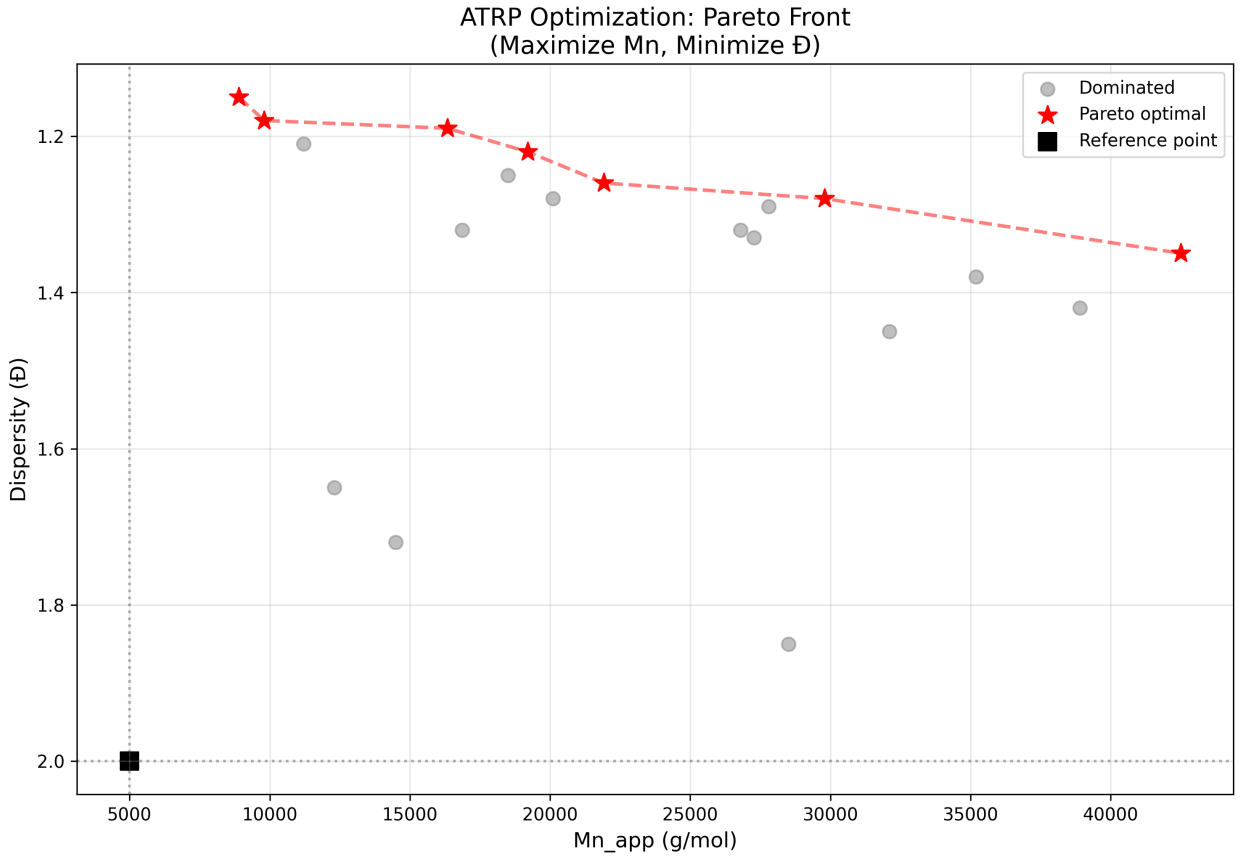

Figure 4: Pareto front showing trade-off between molecular weight and dispersity

8.1 Pareto-Optimal Conditions

```
pareto_df = df[pareto_mask].sort_values("Mn_app", ascending=False)
display_cols = ["HO_EBiB", "MB", "CuBr2_L", "TEOA", "conversion", "Mn_app", "dispersity"]
pareto_df[display_cols].round(3)
```

	HO_EBiB	MB	CuBr2_L	TEOA	conversion	Mn_app	dispersity
9	0.500	150.000	0.080	0.700	76.4	42500.0	1.35
19	0.500	182.451	0.033	0.494	81.3	29800.0	1.28
17	0.519	160.344	0.082	0.700	70.7	21911.0	1.26
6	1.000	100.000	0.100	0.500	92.8	19200.0	1.22
18	0.500	117.254	0.079	0.703	75.7	16339.0	1.19
4	2.000	100.000	0.050	0.500	91.5	9800.0	1.18
13	1.000	50.000	0.030	0.300	95.2	8900.0	1.15

! Key Finding

The Pareto front reveals a fundamental trade-off: achieving higher molecular weight (Mn > 35,000 g/mol) typically requires conditions that broaden the dispersity (Đ > 1.35). A polymer chemist should select conditions based on application requirements.

9 Summary

This demo showed the complete Folio workflow:

1. **Project setup** with multi-objective optimization (multitask GP + NEHVI)
2. **Data entry** with screening experiments, tags, and notes
3. **Bayesian optimization** to explore the Pareto front
4. **Data analysis** with pandas export and visualization
5. **GP inspection** showing predictions and uncertainty
6. **Pareto analysis** identifying optimal trade-offs

```
# Cleanup
folio.delete_project("atrp_optimization")
print("Demo complete!")
```

```
Demo complete!
```