

PlsWork5

January 15, 2026

```
[43]: import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import statsmodels.api as sm
from scipy import stats

# -----
# 0) Paths / filenames
# -----
DATA_DIR = "."

CPI_FILE    = os.path.join(DATA_DIR, "FREDCPIAUCSL.csv")
MICH_FILE   = os.path.join(DATA_DIR, "MICH.csv")
MUNI_FILE   = os.path.join(DATA_DIR, "SP500MUNIBOND.xls")
GREEN_FILE  = os.path.join(DATA_DIR, "SP5500GREENBOND.xls")
DGS10_FILE  = os.path.join(DATA_DIR, "DGS10.csv")

# -----
# 1) Parsing + month-end alignment
# -----
def to_month_end_index(dates: pd.Series) -> pd.DatetimeIndex:

    return dates.dt.to_period("M").dt.to_timestamp("M")

def read_fred_csv(path: str, value_col: str) -> pd.DataFrame:

    df = pd.read_csv(path)
    df["date"] = pd.to_datetime(df["observation_date"], dayfirst=True, errors="coerce")
    df[value_col] = pd.to_numeric(df[value_col], errors="coerce")
    df = df.dropna(subset=["date"]).sort_values("date")

    # collapse to one observation per month
    df["month_end"] = to_month_end_index(df["date"])
    out = df.groupby("month_end")[value_col].last().to_frame()
```

```

        out.index.name = "date"
        return out

def read_sp_index_xls(path: str, level_name: str = "level") -> pd.Series:
    raw = pd.read_excel(path, header=None)

    date_col = raw.iloc[:, 0]
    val_col = raw.iloc[:, 1]

    parsed_dates = pd.to_datetime(date_col, errors="coerce")
    parsed_vals = pd.to_numeric(val_col, errors="coerce")

    mask = parsed_dates.notna() & parsed_vals.notna()
    data = pd.DataFrame({"date": parsed_dates[mask], level_name: ↴
    ↪parsed_vals[mask]})

    data = data.sort_values("date").drop_duplicates(subset=["date"])
    s = data.set_index("date")[level_name]
    s.index = pd.DatetimeIndex(s.index)
    s.name = level_name
    return s

def month_end_last(series_daily: pd.Series) -> pd.Series:
    return series_daily.resample("ME").last()

# -----
# 2) Building macro variables: CPI YoY and MICH expectations
# -----
def build_cpi_yoy(cpi_me: pd.DataFrame) -> pd.Series:
    cpi = cpi_me.copy()
    cpi["infl_yoy"] = 100.0 * (np.log(cpi["CPIAUCSL"]) - np.log(cpi["CPIAUCSL"].shift(12)))
    return cpi["infl_yoy"]

def build_mich_expectations(mich_me: pd.DataFrame) -> pd.Series:
    mich = mich_me.copy()
    mich["exp_for_infl_yoy_t"] = mich["MICH"].shift(12)
    return mich["exp_for_infl_yoy_t"]

# -----
# 3) Returns & main dataset
# -----
def log_return_from_level(level_me: pd.Series, name: str) -> pd.Series:

```

```

r = np.log(level_me).diff()
r.name = name
return r

# -----
# 4) Local Projections: Frequentist (HAC) and Bayesian (Ridge prior)
# -----
def run_lp_frequentist(df, y_col, shock_col="infl_surprise",
                      h_max=24, L=6, mode="single"):

    base = df.copy()

    for l in range(1, L+1):
        base[f"{y_col}_l{l}"] = base[y_col].shift(l)
        base[f"{shock_col}_l{l}"] = base[shock_col].shift(l)

    horizons = range(0, h_max+1) if mode == "single" else range(1, h_max+1)
    rows = []

    for h in horizons:
        d = base.copy()
        if mode == "single":
            d[f"{y_col}_f{h}"] = d[y_col].shift(-h)
            y = d[f"{y_col}_f{h}"]
        else:
            d[f"cum_{y_col}_{h}"] = sum(d[y_col].shift(-j) for j in range(1, h+1))
            y = d[f"cum_{y_col}_{h}"]

        X_cols = [shock_col] + \
                  [f"{y_col}_l{l}" for l in range(1, L+1)] + \
                  [f"{shock_col}_l{l}" for l in range(1, L+1)]

        tmp = pd.concat([y, d[X_cols]], axis=1).dropna()
        n = len(tmp)
        if n == 0:
            rows.append(dict(h=h, beta=np.nan, se_hac=np.nan, nobs=0,
                             t=np.nan, ci_low=np.nan, ci_high=np.nan))
            continue

        Y = tmp[y.name]
        X = sm.add_constant(tmp[X_cols]) # DataFrame with names
        mod = sm.OLS(Y, X).fit()

        bw = max(1, h)
        cov = mod.get_robustcov_results(cov_type="HAC", maxlags=bw)

```

```

# robust to statsmodels returning arrays
shock_pos = list(X.columns).index(shock_col)
beta = cov.params[shock_pos]
se = cov.bse[shock_pos]
tval = beta / se if se > 0 else np.nan

rows.append(dict(
    h=h, beta=beta, se_hac=se, nobs=int(cov.nobs),
    t=tval, ci_low=beta-1.96*se, ci_high=beta+1.96*se
))

return pd.DataFrame(rows).set_index("h")

def bayes_ridge_conjugate(y, X, lam=10.0, a0=0.0, d0=0.0,
                           ↪penalize_intercept=False):
    y = np.asarray(y).reshape(-1,1)
    X = np.asarray(X)
    n, k = X.shape

    V0_inv = lam * np.eye(k)
    if not penalize_intercept:
        V0_inv[0,0] = 0.0

    XtX = X.T @ X
    Xty = X.T @ y

    Vn = np.linalg.inv(XtX + V0_inv)
    bn = Vn @ Xty

    a_n = a0 + n/2.0
    quad = (y.T @ y - bn.T @ (XtX + V0_inv) @ bn).item()
    d_n = d0 + 0.5 * quad

    df_t = 2.0 * a_n
    cov_marg = (d_n / a_n) * Vn
    return bn.flatten(), cov_marg, df_t

def run_lp_bayes(df, y_col, shock_col="infl_surprise",
                  h_max=24, L=6, mode="single", lam=10.0):
    """
    Bayesian LP: same regressors as frequentist LP, but with ridge prior.
    """
    base = df.copy()

```

```

for l in range(1, L+1):
    base[f"y_col_l{l}"] = base[y_col].shift(l)
    base[f"shock_col_l{l}"] = base[shock_col].shift(l)

horizons = range(0, h_max+1) if mode=="single" else range(1, h_max+1)
rows = []

for h in horizons:
    d = base.copy()
    if mode == "single":
        d[f"y_col_f{h}"] = d[y_col].shift(-h)
        y = d[f"y_col_f{h}"]
    else:
        if h == 0:
            continue
        d[f"cum_{y_col}_{h}"] = sum(d[y_col].shift(-j) for j in range(1,h+1))
        y = d[f"cum_{y_col}_{h}"]

    X_cols = [shock_col] + \
              [f"y_col_l{l}" for l in range(1, L+1)] + \
              [f"shock_col_l{l}" for l in range(1, L+1)]

    tmp = pd.concat([y, d[X_cols]], axis=1).dropna()
    n = len(tmp)

    if n < len(X_cols) + 10:
        rows.append(dict(h=h, beta_bayes=np.nan, post_sd=np.nan,
                          ci_low=np.nan, ci_high=np.nan, df_t=np.nan,
                          nob=n))
        continue

    yv = tmp[y.name].values
    X = tmp[X_cols].values
    X = np.column_stack([np.ones(n), X])

    # Standardize regressors (not y) for ridge penalty stability
    mu = X[:, 1:].mean(axis=0)
    sd = X[:, 1:].std(axis=0, ddof=0)
    sd[sd == 0] = 1.0
    Xs = X.copy()
    Xs[:, 1:] = (Xs[:, 1:] - mu) / sd

    b_n, cov_marg, df_t = bayes_ridge_conjugate(yv, Xs, lam=lam, a0=0.0,
                                                d0=0.0)

```

```

# Map coefficients/covariance back to original regressor scale
b_n[1:] = b_n[1:] / sd
cov_marg[1:, 1:] = cov_marg[1:, 1:] / np.outer(sd, sd)

beta = b_n[1] # shock coefficient
post_sd = np.sqrt(cov_marg[1, 1])
q = stats.t.ppf(0.975, df=df_t)

rows.append(dict(
    h=h, beta_bayes=beta, post_sd=post_sd,
    ci_low=beta - q*post_sd, ci_high=beta + q*post_sd,
    df_t=df_t, nobs=n
))

return pd.DataFrame(rows).set_index("h")

def run_lp_frequentist_controls(df, y_col, shock_col="infl_surprise",
                                 controls=None, h_max=24, L=6, mode="single"):
    """
    Frequentist LP with extra controls (and their lags).
    """
    if controls is None:
        controls = []

    base = df.copy()

    for l in range(1, L+1):
        base[f"{y_col}_l{l}"] = base[y_col].shift(l)
        base[f"{shock_col}_l{l}"] = base[shock_col].shift(l)

    for c in controls:
        for l in range(1, L+1):
            base[f"{c}_l{l}"] = base[c].shift(l)

    horizons = range(0, h_max+1) if mode=="single" else range(1, h_max+1)
    rows = []

    for h in horizons:
        d = base.copy()
        if mode == "single":
            d[f"{y_col}_f{h}"] = d[y_col].shift(-h)
            y = d[f"{y_col}_f{h}"]
        else:
            d[f"cum_{y_col}_{h}"] = sum(d[y_col].shift(-j) for j in range(1, h+1))
            y = d[f"cum_{y_col}_{h}"]

        rows.append(dict(
            h=h, y=y, d=d
        ))

```

```

X_cols = [shock_col] + controls \
    + [f"{y_col}_l{l}" for l in range(1, L+1)] \
    + [f"{shock_col}_l{l}" for l in range(1, L+1)]

for c in controls:
    X_cols += [f"{c}_l{l}" for l in range(1, L+1)]

tmp = pd.concat([y, d[X_cols]], axis=1).dropna()
n = len(tmp)

if n == 0:
    rows.append(dict(h=h, beta=np.nan, se_hac=np.nan, nobs=0,
                      t=np.nan, ci_low=np.nan, ci_high=np.nan))
    continue

Y = tmp[y.name]
X = sm.add_constant(tmp[X_cols])

mod = sm.OLS(Y, X).fit()
bw = max(1, h)
cov = mod.get_robustcov_results(cov_type="HAC", maxlags=bw)

shock_pos = list(X.columns).index(shock_col)
beta = cov.params[shock_pos]
se = cov.bse[shock_pos]
tval = beta / se if se > 0 else np.nan

rows.append(dict(
    h=h, beta=beta, se_hac=se, nobs=int(cov.nobs),
    t=tval, ci_low=beta-1.96*se, ci_high=beta+1.96*se
))

return pd.DataFrame(rows).set_index("h")

# -----
# 5) Load and build the dataset
# -----
# CPI and MICH (monthly)
cpi_me = read_fred_csv(CPI_FILE, "CPIAUCSL")
mich_me = read_fred_csv(MICH_FILE, "MICH")

infl_yoy = build_cpi_yoy(cpi_me).rename("infl_yoy")
exp_infl = build_mich_expectations(mich_me).rename("exp_for_infl_yoy_t")

# Bond indices (daily -> month-end)
muni_daily = read_sp_index_xls(MUNI_FILE, level_name="level")

```

```

green_daily = read_sp_index_xls(GREEN_FILE, level_name="level")

muni_me = month_end_last(muni_daily).rename("muni_level")
green_me = month_end_last(green_daily).rename("green_level")

# Returns (monthly log)
r_muni = log_return_from_level(muni_me, "r_muni")
r_green = log_return_from_level(green_me, "r_green")

# Merge to main monthly df (month-end alignment)
df = pd.concat([r_muni, r_green, infl_yoy, exp_infl], axis=1)
df["infl_surprise"] = df["infl_yoy"] - df["exp_for_infl_yoy_t"]
df["r_spread"] = df["r_green"] - df["r_muni"]

# Keep the period where returns exist (and inflation surprise exists)
df = df.dropna(subset=["r_spread", "infl_surprise"]).copy()

print("Main df:", df.shape, df.index.min(), "to", df.index.max())
print(df[["r_muni", "r_green", "r_spread", "infl_yoy", "exp_for_infl_yoy_t", "infl_surprise"]].
    head())

# -----
# 6) Descriptive stats + correlations
# -----

desc = df[["r_green", "r_muni", "r_spread", "infl_surprise"]].describe().
    T[["mean", "std", "min", "max"]]
corr = df[["r_green", "r_muni", "r_spread", "infl_surprise"]].corr()

print("\nDescriptive stats:\n", desc)
print("\nCorrelation matrix:\n", corr)

# Plot inflation surprises
plt.figure(figsize=(10,4))
plt.plot(df.index, df["infl_surprise"])
plt.axhline(0, linewidth=1)
plt.title("Inflation Surprises (CPI YoY minus Michigan expectations)")
plt.xlabel("Date")
plt.ylabel("Inflation surprise")
plt.show()

# -----
# 7) Baseline LP results (Frequentist vs Bayesian)
# -----

H = 24
L = 6

```

```

res_spread_single = run_lp_frequentist(df, "r_spread", h_max=H, L=L, mode="single")
bayes_spread_single = run_lp_bayes(df, "r_spread", h_max=H, L=L, mode="single", lam=10.0)

print("\nFrequentist LP head:\n", res_spread_single.head())
print("\nBayesian LP head:\n", bayes_spread_single.head())

# Main IRF plot
h = res_spread_single.index.values

plt.figure(figsize=(9,5))
plt.plot(h, res_spread_single["beta"], label="Frequentist LP (OLS)")
plt.fill_between(h, res_spread_single["ci_low"], res_spread_single["ci_high"], alpha=0.2)

plt.plot(h, bayes_spread_single["beta_bayes"], label="Bayesian LP (Ridge prior)")
plt.fill_between(h, bayes_spread_single["ci_low"], bayes_spread_single["ci_high"], alpha=0.2)

plt.axhline(0, linewidth=1)
plt.title("LP Impulse Responses to Inflation Surprise")
plt.xlabel("Horizon h (months)")
plt.ylabel("Response of spread return")
plt.legend()
plt.show()

# -----
# 8) Cumulative IRF plot
# -----
freq_cum = run_lp_frequentist(df, "r_spread", h_max=H, L=L, mode="cumulative")
bayes_cum = run_lp_bayes(df, "r_spread", h_max=H, L=L, mode="cumulative", lam=10.0)

plt.figure(figsize=(9,5))
plt.plot(freq_cum.index, freq_cum["beta"], label="Frequentist (cumulative)")
plt.fill_between(freq_cum.index, freq_cum["ci_low"], freq_cum["ci_high"], alpha=0.2)

plt.plot(bayes_cum.index, bayes_cum["beta_bayes"], label="Bayesian (cumulative)")
plt.fill_between(bayes_cum.index, bayes_cum["ci_low"], bayes_cum["ci_high"], alpha=0.2)

plt.axhline(0, linewidth=1)

```

```

plt.title("Cumulative Response of Spread")
plt.xlabel("Horizon h (months)")
plt.ylabel("Cumulative response")
plt.legend()
plt.show()

# -----
# 9) Lambda sensitivity plot (Bayesian)
# -----
lams = [0.0, 1.0, 10.0, 100.0]
plt.figure(figsize=(9,5))
for lam in lams:
    out = run_lp_bayes(df, "r_spread", h_max=H, L=L, mode="single", lam=lam if lam > 0 else 1e-12)
    plt.plot(out.index, out["beta_bayes"], label=f"={lam:g}")
plt.axhline(0, linewidth=1)
plt.title("Sensitivity to Shrinkage (Bayesian LP)")
plt.xlabel("Horizon h (months)")
plt.ylabel("Response of spread return")
plt.legend()
plt.show()

# -----
# 10) Peak effects table
# -----
peak = pd.DataFrame({
    "Estimator": ["Frequentist", "Bayesian"],
    "Peak_h": [
        int(res_spread_single["beta"].abs().idxmax()),
        int(bayes_spread_single["beta_bayes"].abs().idxmax())
    ],
    "Peak_value": [
        float(res_spread_single["beta"].loc[res_spread_single["beta"].abs().idxmax()]),
        float(bayes_spread_single["beta_bayes"].loc[bayes_spread_single["beta_bayes"].abs().idxmax()])
    ]
})
print("\nPeak effects:\n", peak)

# -----
# 11) DGS10 extension (discount-rate control)
# -----
# Load DGS10 daily -> month-end -> monthly change in bp
dgs10 = pd.read_csv(DGS10_FILE)

```

```

dgs10["date"] = pd.to_datetime(dgs10["observation_date"], dayfirst=True, errors="coerce")
dgs10["DGS10"] = pd.to_numeric(dgs10["DGS10"], errors="coerce")
dgs10 = dgs10.dropna(subset=["date"]).sort_values("date").set_index("date")

y10_me_pct = dgs10["DGS10"].resample("ME").last().ffill()
dy10_bp = (y10_me_pct.diff() * 100.0).rename("dy10_bp") # percent-point change
# -> bp

df_rate = df.join(dy10_bp, how="inner").dropna(subset=["dy10_bp"])

print("\nDGS10 control df_rate shape:", df_rate.shape)

# Baseline vs controlled LP (frequentist)
res_base = run_lp_frequentist_controls(df_rate, "r_spread", controls=[], h_max=H, L=L, mode="single")
res_rate = run_lp_frequentist_controls(df_rate, "r_spread", controls=[ "dy10_bp"], h_max=H, L=L, mode="single")

print("\nBaseline LP head (rate df):\n", res_base.head())
print("\nRate-controlled LP head:\n", res_rate.head())

plt.figure(figsize=(9,5))
plt.plot(res_base.index, res_base["beta"], label="Baseline LP")
plt.plot(res_rate.index, res_rate["beta"], label="LP controlling for Δ10yield")
plt.axhline(0, linewidth=1)
plt.title("Baseline vs Rate-Controlled IRFs (DGS10 decomposition)")
plt.xlabel("Horizon h (months)")
plt.ylabel("Response of spread return")
plt.legend()
plt.show()

```

C:\Users\chait\AppData\Local\Temp\ipykernel_32160\3165260118.py:47: UserWarning:
 Could not infer format, so each element will be parsed individually, falling
 back to `dateutil`. To ensure parsing is consistent and as-expected, please
 specify a format.

```

  parsed_dates = pd.to_datetime(date_col, errors="coerce")
C:\Users\chait\AppData\Local\Temp\ipykernel_32160\3165260118.py:47: UserWarning:  

  Could not infer format, so each element will be parsed individually, falling  

  back to `dateutil`. To ensure parsing is consistent and as-expected, please  

  specify a format.
  parsed_dates = pd.to_datetime(date_col, errors="coerce")

Main df: (108, 6) 2016-11-30 00:00:00 to 2025-11-30 00:00:00
          r_muni   r_green   r_spread   infl_yoy  exp_for_infl_yoy_t \
date
2016-11-30 -0.035247 -0.056948 -0.021701  1.670306                  2.7

```

2016-12-31	0.009931	0.022096	0.012165	2.030053	2.6
2017-01-31	0.005210	0.003882	-0.001328	2.479401	2.5
2017-02-28	0.006690	0.005611	-0.001079	2.771596	2.5
2017-03-31	0.002149	0.003114	0.000965	2.411875	2.7

infl_surprise

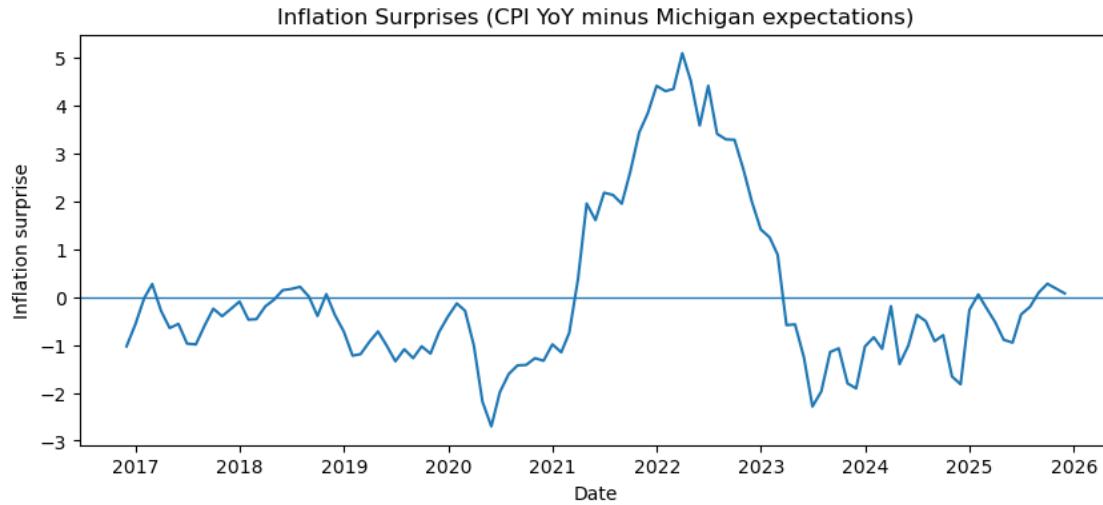
date	infl_surprise
2016-11-30	-1.029694
2016-12-31	-0.569947
2017-01-31	-0.020599
2017-02-28	0.271596
2017-03-31	-0.288125

Descriptive stats:

	mean	std	min	max
r_green	0.001615	0.018587	-0.056948	0.062985
r_muni	0.001807	0.014554	-0.036969	0.057344
r_spread	-0.000192	0.004909	-0.021701	0.012165
infl_surprise	0.027506	1.739809	-2.701995	5.095577

Correlation matrix:

	r_green	r_muni	r_spread	infl_surprise
r_green	1.000000	0.985536	0.864612	-0.226051
r_muni	0.985536	1.000000	0.766960	-0.231885
r_spread	0.864612	0.766960	1.000000	-0.168455
infl_surprise	-0.226051	-0.231885	-0.168455	1.000000



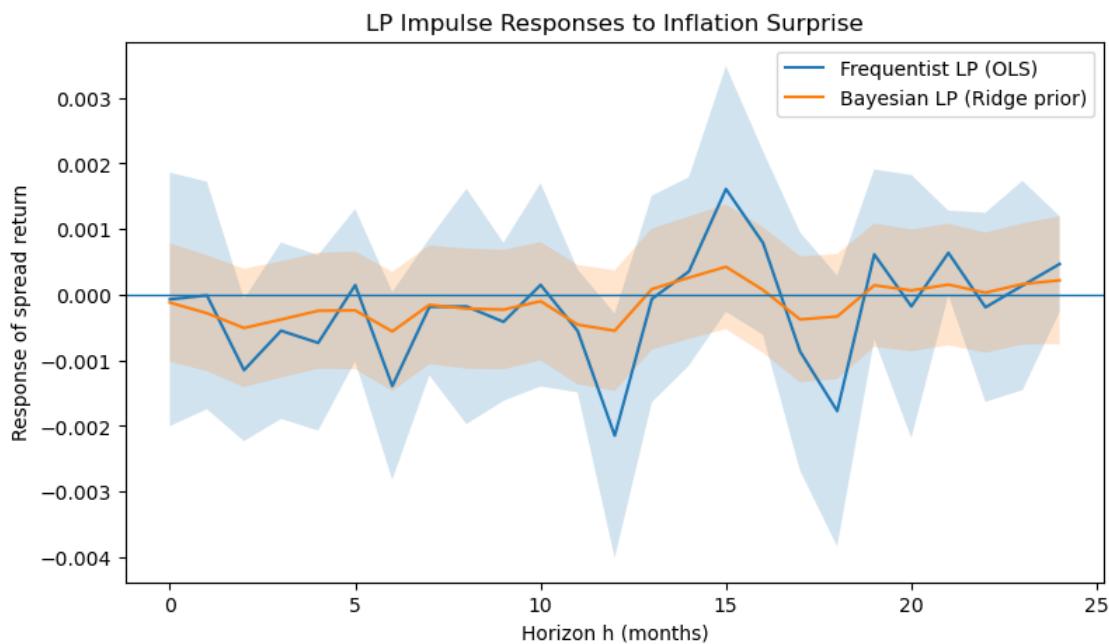
Frequentist LP head:

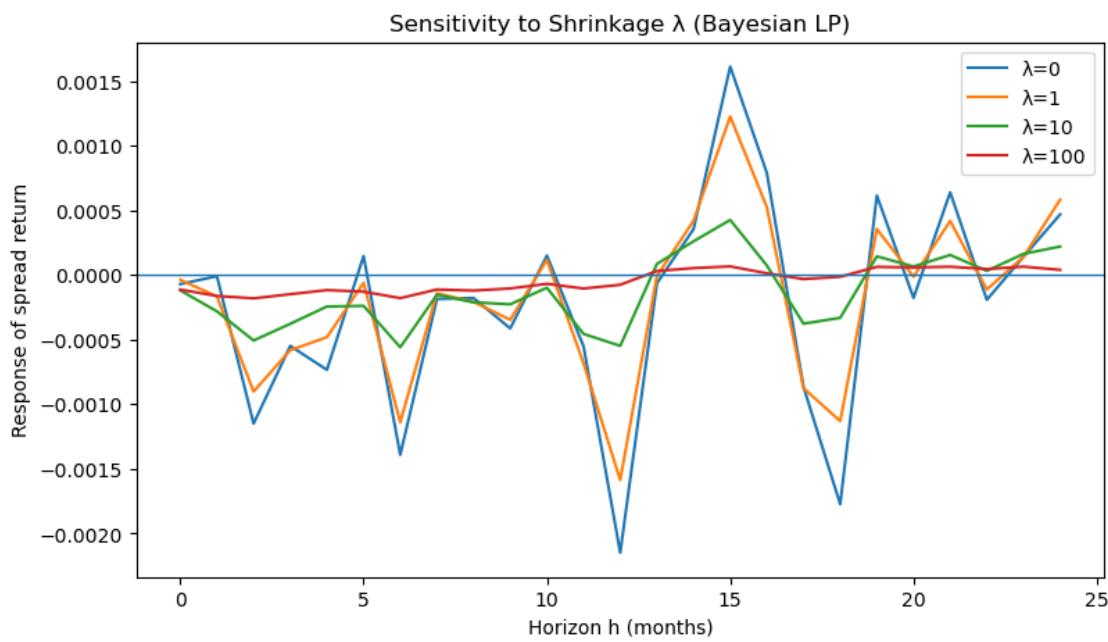
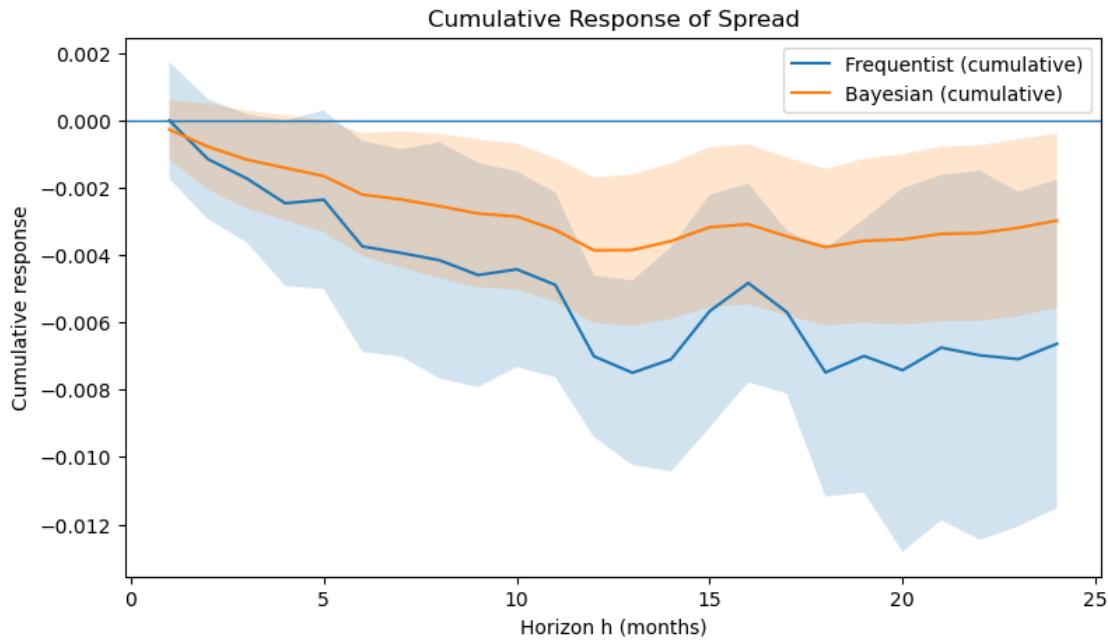
beta	se_hac	nobs	t	ci_low	ci_high
------	--------	------	---	--------	---------

h						
0	-0.000070	0.000985	102	-0.071369	-0.002002	0.001861
1	-0.000010	0.000884	101	-0.010769	-0.001743	0.001724
2	-0.001149	0.000552	100	-2.081255	-0.002231	-0.000067
3	-0.000548	0.000686	99	-0.799744	-0.001892	0.000795
4	-0.000734	0.000682	98	-1.075765	-0.002070	0.000603

Bayesian LP head:

	beta_bayes	post_sd	ci_low	ci_high	df_t	nobs
h						
0	-0.000119	0.000455	-0.001021	0.000783	102.0	102
1	-0.000281	0.000445	-0.001163	0.000601	101.0	101
2	-0.000507	0.000454	-0.001407	0.000394	100.0	100
3	-0.000377	0.000446	-0.001263	0.000509	99.0	99
4	-0.000244	0.000445	-0.001126	0.000639	98.0	98





Peak effects:

	Estimator	Peak_h	Peak_value
0	Frequentist	12	-0.002148
1	Bayesian	6	-0.000559

```
DGS10 control df_rate shape: (108, 7)
```

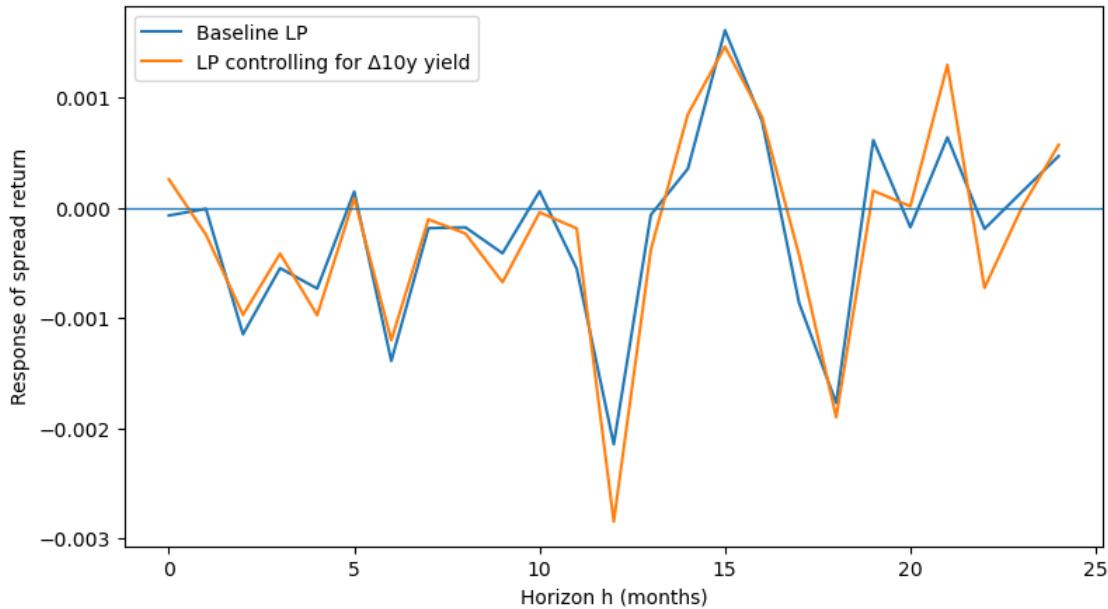
```
Baseline LP head (rate df):
```

	beta	se_hac	nobs	t	ci_low	ci_high
h						
0	-0.000070	0.000985	102	-0.071369	-0.002002	0.001861
1	-0.000010	0.000884	101	-0.010769	-0.001743	0.001724
2	-0.001149	0.000552	100	-2.081255	-0.002231	-0.000067
3	-0.000548	0.000686	99	-0.799744	-0.001892	0.000795
4	-0.000734	0.000682	98	-1.075765	-0.002070	0.000603

```
Rate-controlled LP head:
```

	beta	se_hac	nobs	t	ci_low	ci_high
h						
0	0.000259	0.000880	102	0.293850	-0.001466	0.001983
1	-0.000244	0.000741	101	-0.329604	-0.001696	0.001208
2	-0.000973	0.000574	100	-1.695062	-0.002097	0.000152
3	-0.000416	0.000750	99	-0.554951	-0.001885	0.001053
4	-0.000976	0.000753	98	-1.295472	-0.002451	0.000500

Baseline vs Rate-Controlled IRFs (DGS10 decomposition)



```
[9]: # Baseline vs controlled LP (frequentist)
res_base = run_lp_frequentist_controls(df_rate, "r_spread", controls=[], ↴
                                         h_max=H, L=L, mode="single")
res_rate = run_lp_frequentist_controls(df_rate, "r_spread", ↴
                                         controls=["dy10_bp"], h_max=H, L=L, mode="single")
```

```

print("\nBaseline LP head (rate df):\n", res_base.head())
print("\nRate-controlled LP head:\n", res_rate.head())

plt.figure(figsize=(9,5))
plt.plot(res_base.index, res_base["beta"], label="Baseline LP")
plt.fill_between(res_base.index, res_base["ci_low"], res_base["ci_high"], alpha=0.15)
plt.fill_between(res_rate.index, res_rate["ci_low"], res_rate["ci_high"], alpha=0.15)
plt.plot(res_rate.index, res_rate["beta"], label="LP controlling for Δ10yield")
plt.axhline(0, linewidth=1)
plt.title("Baseline vs Rate-Controlled IRFs (DGS10 decomposition)")
plt.xlabel("Horizon h (months)")
plt.ylabel("Response of spread return")
plt.legend()
plt.show()

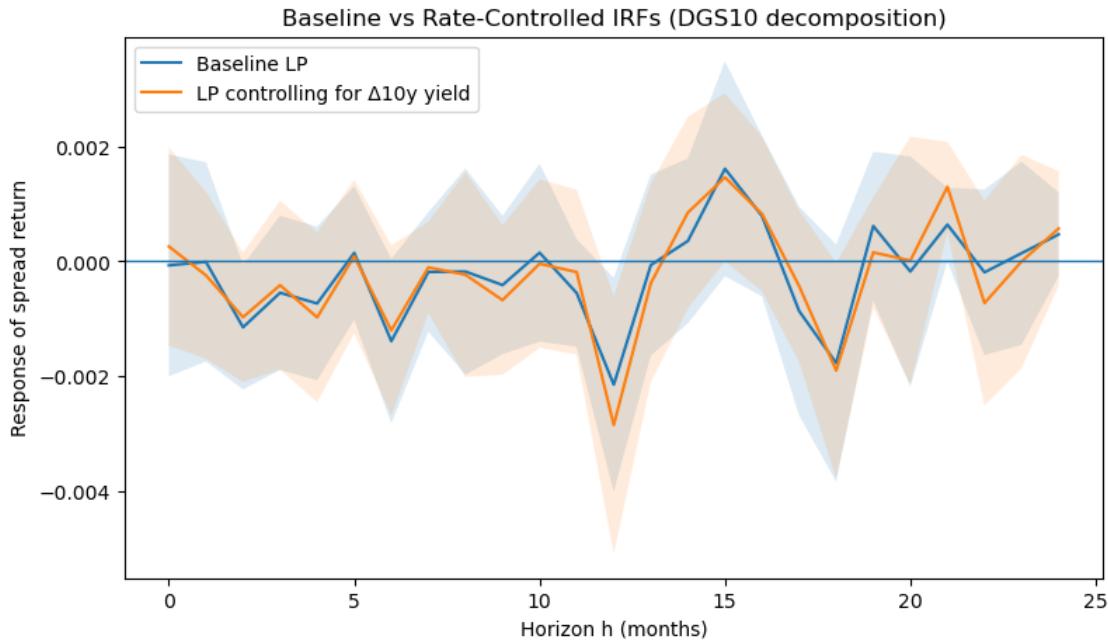
```

Baseline LP head (rate df):

	beta	se_hac	nobs	t	ci_low	ci_high
h						
0	-0.000070	0.000985	102	-0.071369	-0.002002	0.001861
1	-0.000010	0.000884	101	-0.010769	-0.001743	0.001724
2	-0.001149	0.000552	100	-2.081255	-0.002231	-0.000067
3	-0.000548	0.000686	99	-0.799744	-0.001892	0.000795
4	-0.000734	0.000682	98	-1.075765	-0.002070	0.000603

Rate-controlled LP head:

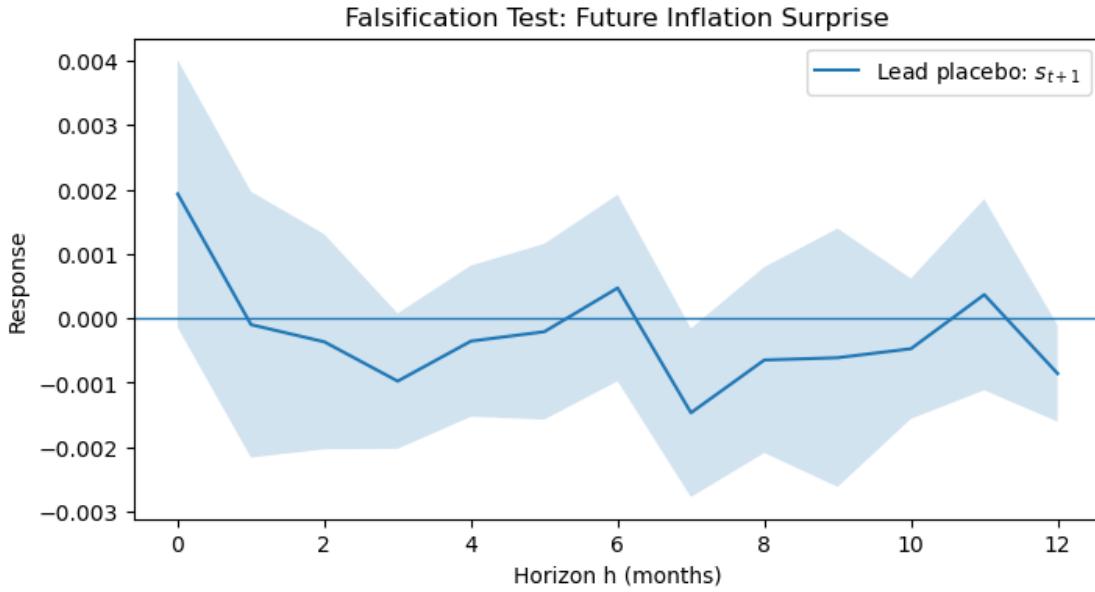
	beta	se_hac	nobs	t	ci_low	ci_high
h						
0	0.000259	0.000880	102	0.293850	-0.001466	0.001983
1	-0.000244	0.000741	101	-0.329604	-0.001696	0.001208
2	-0.000973	0.000574	100	-1.695062	-0.002097	0.000152
3	-0.000416	0.000750	99	-0.554951	-0.001885	0.001053
4	-0.000976	0.000753	98	-1.295472	-0.002451	0.000500



```
[11]: # Lead placebo: using  $s_{t+1}$  as shock
df_lead = df.copy()
df_lead["infl_surprise_lead1"] = df_lead["infl_surprise"].shift(-1)

lead_placebo = run_lp_frequentist(
    df_lead.dropna(subset=["infl_surprise_lead1"]),
    y_col="r_spread",
    shock_col="infl_surprise_lead1",
    h_max=12,
    L=6,
    mode="single"
)

plt.figure(figsize=(8,4))
plt.plot(lead_placebo.index, lead_placebo["beta"], label="Lead placebo:  $\beta_{s_{t+1}}$ ")
plt.fill_between(lead_placebo.index, lead_placebo["ci_low"], lead_placebo["ci_high"], alpha=0.2)
plt.axhline(0, linewidth=1)
plt.xlabel("Horizon h (months)")
plt.ylabel("Response")
plt.title("Falsification Test: Future Inflation Surprise")
plt.legend()
plt.show()
```



```
[17]: lag_choices = [3, 6, 12]
robust_results = {}

for L in lag_choices:
    res_L = run_lp_frequentist(
        df,
        y_col="r_spread",
        shock_col="infl_surprise",
        h_max=24,
        L=L,
        mode="single"
    )
    robust_results[L] = res_L

plt.figure(figsize=(8,5))

# Baseline (L=6) with bands
base = robust_results[6]
plt.plot(base.index, base["beta"], label="L = 6 (baseline)", linewidth=2)
plt.fill_between(base.index, base["ci_low"], base["ci_high"], alpha=0.25)

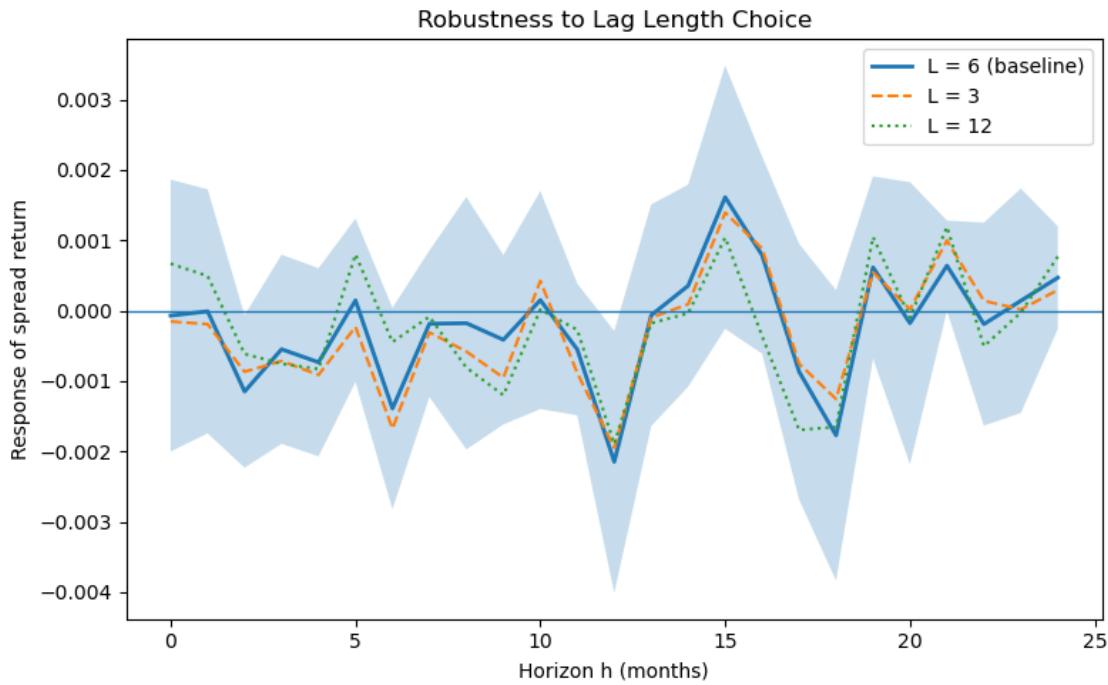
# Other lag lengths (no bands to keep clean)
plt.plot(robust_results[3].index, robust_results[3]["beta"], label="L = 3", 
         linestyle="--")
plt.plot(robust_results[12].index, robust_results[12]["beta"], label="L = 12", 
         linestyle=":")

```

```

plt.axhline(0, linewidth=1)
plt.xlabel("Horizon h (months)")
plt.ylabel("Response of spread return")
plt.title("Robustness to Lag Length Choice")
plt.legend()
plt.tight_layout()
plt.show()

```



```

[37]: def run_lp_frequentist(df, y_col, shock_col="infl_surprise",
                           h_max=24, L=6, mode="single",
                           hac_mode="h", hac_lag=None):
    """
    Baseline frequentist LP:
     $y_{t+h} = a_h + \beta_h \text{shock}_t + \sum_{l=1..L} \phi_{h,l} y_{t-l} + \sum_{l=1..L} \psi_{h,l} \text{shock}_{t-l} + u_{t+h}$ 

    HAC (Newey-West):
     $\text{hac\_mode} = "h" \rightarrow \text{maxlags} = \max(1, h) \quad (\text{baseline})$ 
     $\text{hac\_mode} = "fixed" \rightarrow \text{maxlags} = \text{hac\_lag} \quad (\text{user-specified})$ 
     $\text{hac\_mode} = "auto" \rightarrow \text{maxlags} = \text{floor}(4 * (T/100)^{(2/9)}) \quad (\text{rule-of-thumb})$ 

    Returns DataFrame indexed by horizon h with beta/se/CI.
    """
    import numpy as np

```

```

import pandas as pd
import statsmodels.api as sm
import math

horizons = range(0, h_max + 1) if mode == "single" else range(1, h_max + 1)

out = []
base = df.copy()

# sample size used for "auto" rule
T = base.dropna(subset=[y_col, shock_col]).shape[0]
auto_bw = max(1, int(math.floor(4 * (T / 100) ** (2/9)))) if T > 0 else 1

for h in horizons:
    d = base.copy()

    # Dependent variable
    if mode == "single":
        d[f"y_col_f{h}"] = d[y_col].shift(-h)
        y = d[f"y_col_f{h}"]
    else:
        # cumulative: sum_{j=1..h} y_{t+j}
        d[f"cum_y_col_h"] = sum(d[y_col].shift(-j) for j in range(1, h+1))
        y = d[f"cum_y_col_h"]

    # Regressors: constant + shock_t + L lags of y + L lags of shock
    X = pd.DataFrame(index=d.index)
    X["const"] = 1.0
    X["shock"] = d[shock_col]

    for ell in range(1, L + 1):
        X[f"y_lag{ell}"] = d[y_col].shift(ell)
        X[f"shock_lag{ell}"] = d[shock_col].shift(ell)

    reg_df = pd.concat([y, X], axis=1).dropna()
    y_reg = reg_df.iloc[:, 0]
    X_reg = reg_df.iloc[:, 1:]

    model = sm.OLS(y_reg, X_reg).fit()

    # HAC bandwidth choice
    if hac_mode == "h":
        maxlags = max(1, h)
    elif hac_mode == "fixed":
        if hac_lag is None:

```

```

        raise ValueError("hac_lag must be provided when
↳hac_mode='fixed'")
    maxlags = int(hac_lag)
elif hac_mode == "auto":
    maxlags = auto_bw
else:
    raise ValueError("hac_mode must be one of {'h','fixed','auto'}")

rob = model.get_robustcov_results(cov_type="HAC", maxlags=maxlags)

shock_idx = list(X_reg.columns).index("shock")
beta = float(rob.params[shock_idx])
se = float(rob.bse[shock_idx])

ci_low = beta - 1.96 * se
ci_high = beta + 1.96 * se

out.append({"h": h, "beta": beta, "se": se,
            "ci_low": ci_low, "ci_high": ci_high,
            "hac_maxlags": maxlags})

return pd.DataFrame(out).set_index("h")

```

```

import matplotlib.pyplot as plt

# Baseline (my current)
res_base = run_lp_frequentist(df, "r_spread", "infl_surprise",
                               h_max=24, L=6, mode="single",
                               hac_mode="h")

# Fixed bandwidths
fixed_bandwidths = [6, 8, 12]
res_fixed = {
    bw: run_lp_frequentist(df, "r_spread", "infl_surprise",
                           h_max=24, L=6, mode="single",
                           hac_mode="fixed", hac_lag=bw)
    for bw in fixed_bandwidths
}

# Automatic bandwidth rule (data-driven)
res_auto = run_lp_frequentist(df, "r_spread", "infl_surprise",
                               h_max=24, L=6, mode="single",
                               hac_mode="auto")

plt.figure(figsize=(9,5))

```

```

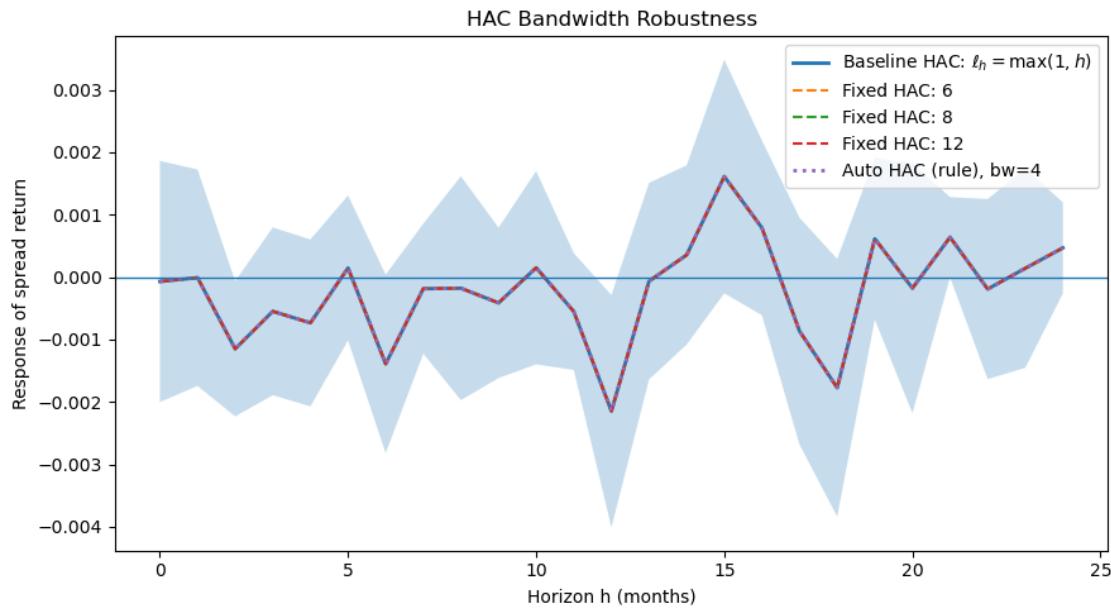
plt.plot(res_base.index, res_base["beta"], label="Baseline HAC:  $\ell_h = \max(1, h)$ ", linewidth=2)
plt.fill_between(res_base.index, res_base["ci_low"], res_base["ci_high"], alpha=0.25)

for bw in fixed_bandwidths:
    plt.plot(res_fixed[bw].index, res_fixed[bw]["beta"], linestyle="--", label=f"Fixed HAC: {bw}")

plt.plot(res_auto.index, res_auto["beta"], linestyle=":", linewidth=2,
         label=f"Auto HAC (rule), bw={int(res_auto['hac_maxlags'].iloc[0])}")

plt.axhline(0, linewidth=1)
plt.xlabel("Horizon h (months)")
plt.ylabel("Response of spread return")
plt.title("HAC Bandwidth Robustness")
plt.legend()
plt.tight_layout()
plt.show()

```



[]: