hw4-2-5

October 26, 2025

1 Executive Summary

We implement a 2s10s yield curve flattener strategy, constructed by shorting the 2-year Treasury and longing the 10-year Treasury in a DV01-neutral method. The position is rebalanced weekly using yields generated from the Nelson-Siegel-Svensson (NSS) model. The strategy's capital is managed with a fixed margin ratio (10% and 2% for the last question), and performance is decomposed into spread, convexity, and time components to analyze returns. To summarize, the flattener strategy will earn a profit when the yield curve flattens and benefits from the positive convexity of the long period.

2 Introduction

We choose a flattener strategy because the 2s10s yield spread is one of the most reliable indicators of economic cycles and interest rate expectations. In the following sections, we will present the our model logic based on the Nelson-Siegel-Svensson (NSS) model and the weekly backtesting methodology. We then evaluate the strategy's long-term behavior and show a performance decomposition into spread, convexity, and time returns. Finally, we compare the cumulative returns under different margin requirements.

3 Discussion

3.1 Plot the cumulative return for your trading strategy.

First, we use code to import data analysis and plotting libraries we usually use and define the data file path, time range, initial capital, margin rate and the maturities for the front and back legs of the yield curve trade.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

file_path = "gsw_yields_2025.csv"
start_date = pd.Timestamp("1983-12-30")
end_date = pd.Timestamp("2025-09-05")
initial_equity = 1000000
margin_rate = 0.10
front_T = 2.0
back_T = 10.0
```

Here we use NSS model to calculate the yield for given maturity. `term1`, `term2` and `term3` represent the level, slope, and curvature components respectively. Attention, we set the minimum of t by `t = max(T, 1e-8)` to avoid 0-dividing problem.

[2]:
```
def nss_yield_row(b0, b1, b2, b3, t1, t2, T):
    t = max(T, 1e-8)
    term1 = (1 - np.exp(-t / t1)) / (t / t1) if t1 > 0 else 1
    term2 = term1 - np.exp(-t / t1) if t1 > 0 else 0
    term3 = (1 - np.exp(-t / t2)) / (t / t2) - np.exp(-t / t2) if t2 > 0 and t2␣
 ↪< 1e5 else 0
    return b0 + b1 * term1 + b2 * term2 + b3 * term3
```

Defines functions to calculate the price of ZCB and DV01, where `1e-4` stands for 1 basis point.

[3]:
```
def zcb_price(y, T):
    return np.exp(-y * T)

def dv01_per_dollar(y, T):
    return T * np.exp(-y * T) * 1e-4
```

Loads and cleans yield-curve parameter data, then computes yields by NSS model we just setup to generate our zero-coupon prices and per-dollar DV01s. Using Friday's closing values to represent the week's state helps reduce noise.

[4]:
```
df = pd.read_csv(file_path, parse_dates=["Date"])
df = df.rename(columns={c: c.upper() for c in df.columns})
df = df.set_index("DATE").sort_index().ffill().bfill()

df["y_f"] = df.apply(lambda r: nss_yield_row(r.BETA0, r.BETA1, r.BETA2, r.
 ↪BETA3, r.TAU1, r.TAU2, front_T) / 100, axis=1)
df["y_b"] = df.apply(lambda r: nss_yield_row(r.BETA0, r.BETA1, r.BETA2, r.
 ↪BETA3, r.TAU1, r.TAU2, back_T) / 100, axis=1)
df["y_1w"] = df.apply(lambda r: nss_yield_row(r.BETA0, r.BETA1, r.BETA2, r.
 ↪BETA3, r.TAU1, r.TAU2, 7 / 365) / 100, axis=1)

df["p_f"] = np.exp(-df.y_f * front_T)
df["p_b"] = np.exp(-df.y_b * back_T)
df["dvf"] = dv01_per_dollar(df.y_f, front_T)
df["dvb"] = dv01_per_dollar(df.y_b, back_T)

weekly = df.resample("W-FRI").last().loc[start_date:end_date].copy()

equity = initial_equity
records = []
prev_long_value = prev_short_value = prev_margin = None
prev_date = None
max_ratio = 1e4
```

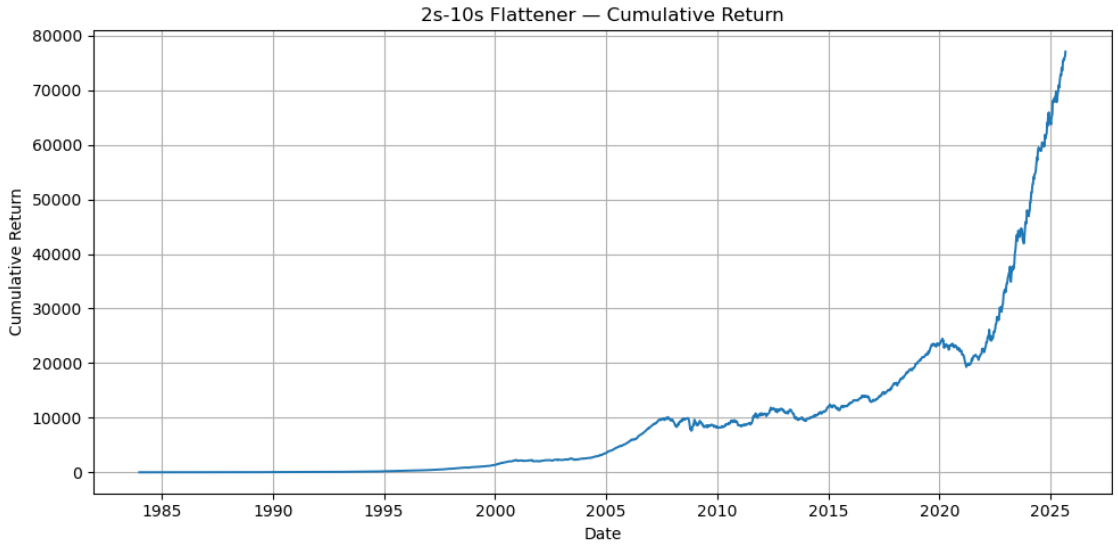
This loop runs a weekly backtest: it computes long/short P&L from price changes, accrues interest

on last period's cash via the 1-week yield, sets the hedge ratio via DV01 ratio `r2 = dvb/dvf`, targets gross exposure as `equity / margin_rate`, solves for long/short notionals and current margin/cash, logs equity, and updates state. It then builds the result DataFrame with cumulative return. The accrued interest base is `cash_prev = prev_short_value - prev_long_value + prev_margin`. When `denom` is less than 0 or not a number, `denom` will be replaced by `NaN`.

[5]:
```
for i, (date, row) in enumerate(weekly.iterrows()):
    pf, pb = float(row.p_f), float(row.p_b)
    dvf, dvb = float(row.dvf), float(row.dvb)
    y1w = float(row.y_1w)
    if i > 0:
        curr_long_value = long_notional * pb
        curr_short_value = short_notional * pf
        pnl_long = curr_long_value - prev_long_value
        pnl_short = prev_short_value - curr_short_value
        days = (date - prev_date).days
        cash_prev = prev_short_value - prev_long_value + prev_margin
        interest = cash_prev * y1w * (days / 365.0)
        equity = equity + pnl_long + pnl_short + interest
    r2 = (dvb / dvf) if dvf > 0 else 1.0
    if not np.isfinite(r2):
        r2 = 1.0
    r2 = min(max(r2, 1e-6), max_ratio)
    target_gross = equity / margin_rate
    denom = r2 * pf + pb
    if denom <= 0 or not np.isfinite(denom):
        long_notional = np.nan
        short_notional = np.nan
        long_value = short_value = margin = cash = np.nan
    else:
        long_notional = target_gross / denom
        short_notional = r2 * long_notional
        long_value = long_notional * pb
        short_value = short_notional * pf
        margin = (abs(long_value) + abs(short_value)) * margin_rate
        cash = short_value - long_value + margin
    records.append({"date": date, "equity": equity})
    prev_long_value = long_value
    prev_short_value = short_value
    prev_margin = margin
    prev_date = date

res = pd.DataFrame(records).set_index("date").sort_index()
res["cumulative_return"] = res.equity / initial_equity - 1.0
```

Now we can plot the cumulative return for our trading strategy.

[6]:
```
plt.figure(figsize=(10, 5))
plt.plot(res.index, res["cumulative_return"], lw=1.4)
plt.title("2s-10s Flattener - Cumulative Return")
plt.ylabel("Cumulative Return")
plt.xlabel("Date")
plt.grid(True)
plt.tight_layout()
plt.show()
```



This chart indicates that a short-2s / long-10s (flattener) trading strategy has delivered strong positive performance especially from 1983 to 2025, returns rise steadily, with exponential growth after 2020. The steep rise suggests persistent yield curve flattening or even inversion.

3.2 Plot the convexity risk over time.

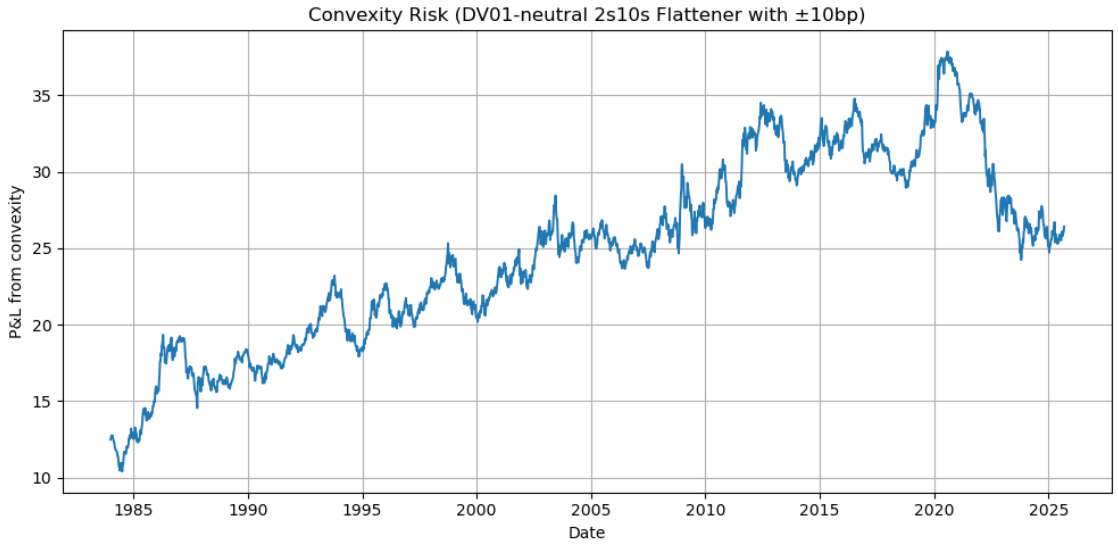
These codes builds a DV01-neutral 2s–10s position. When shocks yields by ±10 bp (parallel shift), we computes P&L for each move and apply `convexity_pnl_10bp = 0.5 * (pnl_up + pnl_dn)` to approximately calculate the value of convexity.

[7]:
```
N_back = 1000000
dY = 0.001

weekly2 = df.resample("W-FRI").last().loc[start_date:end_date].copy()
y2,y10 = weekly2["y_f"], weekly2["y_b"]
p2,p10 = weekly2["p_f"], weekly2["p_b"]
dv2,dv10 = weekly2["dvf"], weekly2["dvb"]

S_front = (dv10 / dv2) * N_back
```

```
p10_up = np.exp(-(y10 + dY) * back_T)
p2_up  = np.exp(-(y2  + dY) * front_T)
p10_dn = np.exp(-(y10 - dY) * back_T)
p2_dn  = np.exp(-(y2  - dY) * front_T)

pnl_up = N_back * (p10_up - p10) - S_front * (p2_up - p2)
pnl_dn = N_back * (p10_dn - p10) - S_front * (p2_dn - p2)
convexity_pnl_10bp = 0.5 * (pnl_up + pnl_dn)
```

After the operations above, we can plot the convexity risk.

[8]:
```
plt.figure(figsize=(10,5))
plt.plot(convexity_pnl_10bp.index, convexity_pnl_10bp.values, lw=1.4)
plt.title("Convexity Risk (DV01-neutral 2s10s Flattener with ±10bp)")
plt.ylabel("P&L from convexity")
plt.xlabel("Date")
plt.grid(True)
plt.tight_layout()
plt.show()
```



This chart shows your positive convexity premium on a DV01-neutral 2s10s flattener, which means that you earn a small profit from the higher convexity of the longer-duration leg, regardless to rates moving up or down. As long-term yields decline, this convexity gain grows larger, causing the curve to rise gradually from about USD10 to USD35.

3.3 Decompose the weekly return into Spread, Convexity Time return and Residual

Build Friday-weekly index, subset needed columns, and compute long-end yield change `dy`.

[9]:
```
idx = df.resample("W-FRI").last().loc[start_date:end_date].index
weekly = df.loc[idx, ["y_f","y_b","p_f","p_b","y_1w","dvf","dvb"]].copy()
weekly["dy"] = weekly["y_b"].diff().fillna(0.0)
```

Initialize the lists to hold data for spread, convexity, time returns.

[10]:
```
spread_pnl = []
convex_pnl = []
time_pnl = []
total_ret = []
```

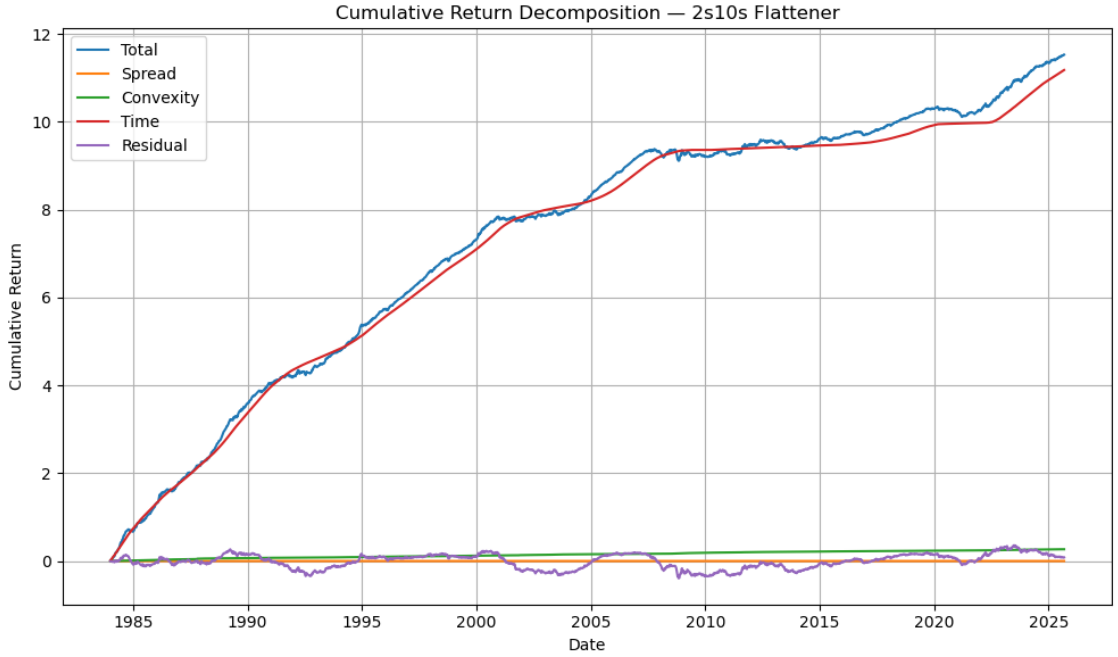
Load previous/current weekly data. Then compute gross exposure from margin and hedge with r2=dvb/dvf to get nominal long and short positions. After all preparations, we now calculate spread return: `spread_pnl.append( (DV_back - DV_front) * dy * 10000.0 )`, convexity return: `convex_pnl.append( 0.5 * (C_back*L - C_front*S) * (dy**2) )`, time return: `time_pnl.append( cash0 * y1w * (days/365.0) )` and residual by using `total_return - (spread_ret + convex_ret + time_ret)`.

[11]:
```
for i in range(1, len(weekly)):
    d0, d1 = idx[i-1], idx[i]
    y2,y10 = weekly.loc[d0, "y_f"], weekly.loc[d0, "y_b"]
    pf,pb   = weekly.loc[d0, "p_f"], weekly.loc[d0, "p_b"]
    dvf,dvb = weekly.loc[d0, "dvf"], weekly.loc[d0, "dvb"]
    dy      = weekly.loc[d1, "dy"]
    y1w     = weekly.loc[d1, "y_1w"]
    days    = (d1 - d0).days
    eq0     = res.loc[d0, "equity"]
    r2      = dvb / dvf
    gross   = eq0 / margin_rate
    denom   = r2*pf + pb
    L       = gross / denom
    S       = r2 * L
    DV_back = dvb * L
    DV_front= dvf * S
    spread_pnl.append( (DV_back - DV_front) * dy * 10000.0)
    C_back  = (back_T**2)  * np.exp(-y10*back_T)
    C_front = (front_T**2) * np.exp(-y2*front_T)
    convex_pnl.append( 0.5 * (C_back*L - C_front*S) * (dy**2))
    long_val0  = L * pb
    short_val0 = S * pf
    margin0    = (abs(long_val0)+abs(short_val0)) * margin_rate
    cash0      = short_val0 - long_val0 + margin0
    time_pnl.append( cash0 * y1w * (days/365.0))
    eq1 = res.loc[d1, "equity"]
    total_ret.append( (eq1 - eq0) / eq0 )

spread_ret   = pd.Series(spread_pnl, index=idx[1:]) / res["equity"].shift(1).
 ↪loc[idx[1:]]
```

```
convex_ret   = pd.Series(convex_pnl, index=idx[1:]) / res["equity"].shift(1).
 ↪loc[idx[1:]]
time_ret     = pd.Series(time_pnl,   index=idx[1:]) / res["equity"].shift(1).
 ↪loc[idx[1:]]
total_return = pd.Series(total_ret,  index=idx[1:])
residual_ret = total_return - (spread_ret + convex_ret + time_ret)

cum = pd.DataFrame({
    "Total": total_return.cumsum(),
    "Spread": spread_ret.cumsum(),
    "Convexity": convex_ret.cumsum(),
    "Time": time_ret.cumsum(),
    "Residual": residual_ret.cumsum()
})
```

Draw the figure of spread return, convexity return, time return and residual. Put them in the same chart.

[12]:
```
plt.figure(figsize=(10,6))
for c in ["Total","Spread","Convexity","Time","Residual"]:
    plt.plot(cum.index, cum[c], label=c)
plt.legend()
plt.title("Cumulative Return Decomposition - 2s10s Flattener")
plt.ylabel("Cumulative Return")
plt.xlabel("Date")
plt.grid(True)
plt.tight_layout()
plt.show()
```



- Total: Shows the strategy's overall cumulative return — the sum of all return sources (spread, convexity, time, and residual).
- Spread: Reflects the linear return from changes in the 10Y–2Y yield spread; usually small under DV01 neutrality.
- Convexity: Represents the return from second-order rate effects (convexity), typically a stable positive contributor.
- Time: Captures the contribution from the passage of time and interest earned on cash, providing the most steady and persistent positive return.

3.4 Plot the cumulative total return of the 2% to the 10% margin requirement.

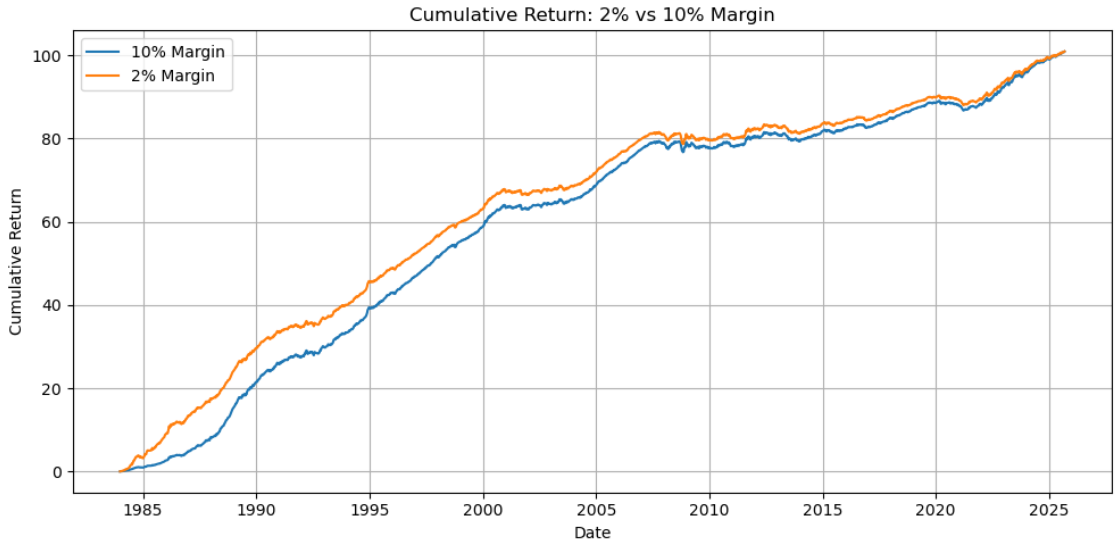
First, we define a backtest function here with guards. Since in the first time, my margin exploded and result in a problematic figure, here I have to apply `r2_min/max`, `denom_floor`, `max_leverage_cap`, `max_gross_cap` to avoid such problems. Being the same, we compute long/short P&L from price moves and accrue interest on last period cash. Then we calculated DV01-neutral hedge ratio, set `target_gross = min(eq*max_lev, max_gross_cap)` and solve notionals Long, Short positions. Finally compute market values and margin.

[14]:
```
def run_backtest_with_guards(margin, weekly, initial_equity):
    eq = float(initial_equity)
    rec = []
    prev_Lv = prev_Sv = prev_M = None
    prev_date = None
    r2_min, r2_max = 0.2, 5.0
    denom_floor = 1e-6
    max_leverage_cap = 25.0
    max_gross_cap = 100.0 * initial_equity

    for i, (date, row) in enumerate(weekly.iterrows()):
        pf, pb = float(row.p_f), float(row.p_b)
        dvf, dvb = float(row.dvf), float(row.dvb)
        y1w = float(row.y_1w)

        if i > 0:
            Lv = L * pb
            Sv = S * pf
            pnl_long  = Lv - prev_Lv
            pnl_short = prev_Sv - Sv
            days = (date - prev_date).days
            cash_prev = prev_Sv - prev_Lv + prev_M
            eq += pnl_long + pnl_short + cash_prev * y1w * (days/365.0)

        r2 = dvb / dvf if dvf > 0 else 1.0
        r2 = float(np.clip(r2, r2_min, r2_max))
        denom = max(r2*pf + pb, denom_floor)
```

```
        max_lev = min(1.0/margin, max_leverage_cap)
        target_gross = min(eq * max_lev, max_gross_cap)

        L = target_gross / denom          # long 10y face
        S = r2 * L                        # short 2y face
        Lv, Sv = L*pb, S*pf
        M = (abs(Lv)+abs(Sv)) * margin

        rec.append((date, eq))
        prev_Lv, prev_Sv, prev_M, prev_date = Lv, Sv, M, date

    out = pd.DataFrame(rec, columns=["date","equity"]).set_index("date")
    out["cum"] = out["equity"]/initial_equity - 1.0
    return out["cum"]

cum_10 = run_backtest_with_guards(0.10, weekly, initial_equity)
cum_02 = run_backtest_with_guards(0.02, weekly, initial_equity)
```

Here we show the figures of the comparason between cumulative under the constraint of 2% and 10% margin.

[15]:
```
plt.figure(figsize=(10,5))
plt.plot(cum_10.index, cum_10, label="10% Margin")
plt.plot(cum_02.index, cum_02, label="2% Margin")
plt.title("Cumulative Return: 2% vs 10% Margin")
plt.ylabel("Cumulative Return")
plt.xlabel("Date")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



A lower margin magnifies both leverage and drawdowns, making the cumulative return curve swing much more dramatically — and increasing the risk of a margin call.

4 Conclusion

Our DV01-neutral "flattener" strategy exhibits consistent long-term profitability, mainly driven by the positive convexity premium and steady time value. Due to the choice of hedging neutrality, spread returns remain slight. However, the convexity effect contributes persistent gains as long-term yields decline. Although lower margin ratios can amplify gains within a period, it also increases volatility and thereby enhence the potential risk of forced liquidation. Overall, the strategy demonstrates how yield-curve dynamics, duration matching, and convexity exposure can combine to produce sustained returns with manageable risk.

Gen AI Disclosure: I finished the homework with the help of Gen AI focusing on: NSS model selection, coding grammer correction, guarding method for 2% margin requirement.