I have this code below. I’d like to make some changes specifically to the rebalance swaps function.

What I’d like to do is to is, within each of the lines below

eom\_date = add\_tenor(current\_date, "1M", "F", 'ldn', 1)

'close\_date': add\_tenor(date, "1M", "F", 'ldn', 1),

'close\_date': add\_tenor(date, "1M", "F", 'ldn', 1),

Instead of specifying the calendar manually, I would like to dynamically set the calendar in the same way that I do within the ‘price\_swaps’ function using

defaults.spec[str(self.ccy + '\_irs')]['calendar'])

could you do that for me?

import pandas as pd

import datetime as dt

from rateslib import \*

from tia.bbg import LocalTerminal

from dateutil.relativedelta import relativedelta

sids = {'gbp': 141}

# sids = {'eur': 514, 'usd': 490, 'gbp': 141, 'chf': 234, 'sek': 185, 'nok':487}

print(list(sids.keys())[0])

# exit()

all\_tenors = list(range(1, 31))

class SwapCreator():

def \_\_init\_\_(self, ccy, date=dt.today()):

self.ccy = ccy

self.curvenum = sids.get(ccy)

self.date = date

def create\_curve(self):

curve\_id = 'YCSW' + str(self.curvenum).zfill(4) + ' Index'

resp = LocalTerminal.get\_reference\_data(curve\_id, 'CURVE\_TENOR\_RATES', CURVE\_DATE=self.date.strftime('%Y%m%d'))

df = resp.as\_frame()

tenors = df['CURVE\_TENOR\_RATES'].iloc[0]['Tenor'].to\_list()

rates = df['CURVE\_TENOR\_RATES'].iloc[0]['Mid Yield'].to\_list()

data = pd.DataFrame({'Term': tenors, 'Rate': rates})

data['Termination'] = [add\_tenor(self.date, \_, "F", defaults.spec[str(self.ccy + '\_irs')]['calendar']) for \_ in data["Term"]]

curve = Curve(

id=str(self.ccy + self.ccy),

convention=defaults.spec[self.ccy + '\_irs']['convention'],

modifier=defaults.spec[self.ccy + '\_irs']['modifier'],

interpolation='log\_linear',

nodes={

\*\*{self.date: 1.0},

\*\*{\_: 1.0 for \_ in data["Termination"]},

}

)

kws = dict(

spec=str(self.ccy + '\_irs'),

curves=str(self.ccy + self.ccy)

)

solver = Solver(

curves=[curve],

instruments=[IRS(termination=\_, effective=self.date, \*\*kws) for \_ in data["Termination"]],

s=data['Rate'],

instrument\_labels=data["Term"],

id=str(self.ccy + '\_rates')

)

data['DF'] = [float(curve[\_]) for \_ in data["Termination"]]

return curve, solver

def price\_swaps(self):

curve, solver = self.create\_curve()

kws = dict(

spec=str(self.ccy + '\_irs'),

curves=str(self.ccy + self.ccy)

)

swaps\_data = []

for i in range(1, 10):

for j in range(1, 11 - i):

effective = add\_tenor(self.date, f"{i}Y", "F", defaults.spec[str(self.ccy + '\_irs')]['calendar'])

termination = add\_tenor(effective, f"{j}Y", "F", defaults.spec[str(self.ccy + '\_irs')]['calendar'])

swap = IRS(

effective=effective,

termination=termination,

notional=10e6,

\*\*kws

)

rate = float(swap.rate(solver=solver))

rolled\_rate = float(swap.rate(curve.roll("1M")))

roll = rolled\_rate - rate

swaps\_data.append((f"{self.ccy} {i}Y{j}Y", f"{i}Y{j}Y", rate, rolled\_rate, roll))

swaps\_df = pd.DataFrame(swaps\_data, columns=['name', 'term', 'rate', 'rolled\_rate', 'roll'])

swaps\_df.sort\_values('roll', inplace=True)

return swaps\_df

def get\_eom\_dates(start\_date, end\_date):

dates = []

current\_date = start\_date

while current\_date <= end\_date:

eom\_date = add\_tenor(current\_date, "1M", "F", 'ldn', 1)

if eom\_date > end\_date:

break

dates.append(eom\_date)

current\_date = eom\_date

return dates

def rebalance\_swaps(start\_date, end\_date):

eom\_dates = get\_eom\_dates(start\_date, end\_date)

positions = []

for date in eom\_dates:

swap\_creator = SwapCreator(f'{list(sids.keys())[0]}', date)

swaps\_df = swap\_creator.price\_swaps()

swaps\_df['date'] = date

receive\_swaps = swaps\_df.head(3)

pay\_swaps = swaps\_df.tail(3)

# Close out previous month's positions

if positions:

for position in positions:

if position['close\_date'] == date:

entry\_rate = position['rate']

closing\_swap = swaps\_df[swaps\_df['term'] == position['term']]

if not closing\_swap.empty:

closing\_rate = closing\_swap['rolled\_rate'].values[0]

pnl = (closing\_rate - entry\_rate) \* position['direction'] \* 100 \* 5000

position['pnl'] = pnl

else:

position['pnl'] = None

# Add new positions

for \_, swap in receive\_swaps.iterrows():

# new\_term = f"{int(swap['term'].split('Y')[0]) - 1}Y{swap['term'].split('Y')[1]}"

positions.append({

'date': date,

'term': swap['term'],

'rate': swap['rate'],

'direction': -1, # Receive

'pnl': None,

'close\_date': add\_tenor(date, "1M", "F", 'ldn', 1),

# 'new\_term': new\_term

})

for \_, swap in pay\_swaps.iterrows():

# new\_term = f"{int(swap['term'].split('Y')[0]) - 1}Y{swap['term'].split('Y')[1]}"

positions.append({

'date': date,

'term': swap['term'],

'rate': swap['rate'],

'direction': 1, # Pay

'pnl': None,

'close\_date': add\_tenor(date, "1M", "F", 'ldn', 1),

# 'new\_term': new\_term

})

# Print positions for the month

print(f"Positions for {date}")

for position in positions:

if position['date'] == date:

print(position)

return positions

start\_date = dt.today() - relativedelta(years=5)

end\_date = dt.today()

positions = rebalance\_swaps(start\_date, end\_date)

# Convert positions to DataFrame and save to CSV

positions\_df = pd.DataFrame(positions)

positions\_df.to\_csv(f'{list(sids.keys())[0]}\_swap\_strat\_pnl.csv', index=False)