#平仓盈亏计算，按POSITION的PNL计算，只适用于下一BAR开盘撮合成交模式

#计算最大回撤

import talib as ta

import numpy as np

from collections import Counter

def clearance(context,bar\_dict,ID):#移仓换月

l\_RB=get\_future\_contracts(ID)

f\_list=l\_RB

for i in f\_list:

xxx=context.portfolio.positions[i].buy\_quantity

yyy=context.portfolio.positions[i].sell\_quantity

if xxx!=0 and i!=context.s1:

sell\_close(i,xxx)

buy\_open(context.s1,xxx)

if yyy!=0 and i!=context.s1:

buy\_close(i,yyy)

sell\_open(context.s1,yyy)

if xxx!=0 or yyy!=0:

context.holding=i

break

else:

context.holding=0

def init(context):

# context内引入全局变量s1

context.fired = False

context.instru=['AG','AL','AU','BU','CU','HC','NI','PB','RB','RU','SN','ZN','PP','V','Y','A','C','CS','I','J','JM','JD','L','M','P','CF','FG','MA','OI','RM','SF','SM','SR','TA','ZC']

context.r =0

context.lot =0.005

context.level ='1d'

context.ma =40

context.N =4

context.atr =20

context.account\_initial=1000000

context.max =context.account\_initial

context.min =context.account\_initial

context.major =0

context.holding =0

context.buy\_qty1 =0

context.sell\_qty1 =0

context.total\_value =0

context.realized\_pnl=0

context.day\_pnl =0

context.total\_pnl =0

context.cost =0

context.drawdown =0

context.MAX\_drawdown=0

context.dict = dict([(k, [0,0,0]) for k in context.instru])

context.pnl =[]

# 初始化时订阅合约行情。订阅之后的合约行情会在handle\_bar中进行更新。

subscribe([i+str(88) for i in context.instru])

# 实时打印日志

logger.info("RunInfo: {}".format(context.run\_info))

# before\_trading此函数会在每天策略交易开始前被调用，当天只会被调用一次

def before\_trading(context):

pass

# 你选择的期货数据更新将会触发此段逻辑，例如日线或分钟线更新

def handle\_bar(context, bar\_dict):

for i in context.instru:

if len(history\_bars(i+str(88),100,context.level,'close')) > context.ma:

context.major = i+str(88)

context.s1 = get\_dominant\_future(i)

context.buy\_qty1 = context.portfolio.positions[context.s1].buy\_quantity

context.sell\_qty1 = context.portfolio.positions[context.s1].sell\_quantity

recovery(context,bar\_dict,context.r)

#ATR

max =np.max((context.high\_r[context.ma-context.atr:context.ma],context .close\_r[context.ma-context.atr-1:context.ma-1]),axis=0)

min =np.min((context.low\_r[context.ma-context.atr:context.ma],context .close\_r[context.ma-context.atr-1:context.ma-1]),axis=0)

tr =max-min

atr =np.mean(tr)

#atr计算仓位

multiplier =instruments(context.major).contract\_multiplier

context.lot =int(context.realized\_pnl\*context.lot/(atr\*multiplier))

#ATR通道

ma =ta.MA(context.close\_r,context.ma)[-1]

high =ma+context.N\*atr

low =ma-context.N\*atr

c =context.close\_r[-1]

#交易逻辑

clearance(context,bar\_dict,i)#移仓换月

if c<ma and context.buy\_qty1!=0:#平多仓

sell\_close(context.s1,context.buy\_qty1)

elif c>ma and context.sell\_qty1!=0:#平空仓

buy\_close(context.s1,context.sell\_qty1)

elif c>high and context.holding==0:#开多仓

buy\_open(context.s1,context.lot)

elif c<low and context.holding==0:#开空仓

sell\_open(context.s1,context.lot)

#平仓盈亏计算

#判断平仓时间

context.dict[i][1]=context.dict[i][2]

context.dict[i][2]=context.portfolio.positions[context.s1].pnl

if context.dict[i][1]!=0 and context.dict[i][2]==0:

context.pnl.append(context.dict[i][1])

#手续费

cost =context.portfolio.positions[context.s1].transaction\_cost

context.cost += cost

def recovery(context,bar\_dict,r):#复权

if r==0:

#等差复权

#主连切片数据

open\_1 = history\_bars(context.major,context.ma,context.level,'open')

high\_1 = history\_bars(context.major,context.ma,context.level,'high')

low\_1 = history\_bars(context.major,context.ma,context.level,'low')

close\_1 = history\_bars(context.major,context.ma,context.level,'close')

#当前主力切片数据

open\_2 = history\_bars(context.s1,context.ma,context.level,'open')

high\_2 = history\_bars(context.s1,context.ma,context.level,'high')

low\_2 = history\_bars(context.s1,context.ma,context.level,'low')

close\_2 = history\_bars(context.s1,context.ma,context.level,'close')

#判断主力换月日期

s1\_leng =len(close\_2) #context.ma-0

#计算主力换月收盘次日开盘跳空

#主力合约数据长度不够造成两个list无法比较错误

if len(close\_2)<len(close\_1):

close=close\_1[-len(close\_2):]-close\_2

else: close=close\_1-close\_2

count =Counter(close)[0]

#未复权主连合约跳空

bias =open\_1[context.ma-count]-close\_1[context.ma-count-1]

#构造复权主力合约

price\_bias\_a =[bias]\*(context.ma-count)

price\_bias\_b =[0]\*count

price\_bias =price\_bias\_a+price\_bias\_b

context.open\_r =open\_1+price\_bias

context.high\_r =high\_1+price\_bias

context.low\_r =low\_1+price\_bias

context.close\_r =close\_1+price\_bias

if r==1:

#等比复权

#主连切片数据

open\_1 = history\_bars(context.major,context.ma,context.level,'open')

high\_1 = history\_bars(context.major,context.ma,context.level,'high')

low\_1 = history\_bars(context.major,context.ma,context.level,'low')

close\_1 = history\_bars(context.major,context.ma,context.level,'close')

#当前主力切片数据

open\_2 = history\_bars(context.s1,context.ma,context.level,'open')

high\_2 = history\_bars(context.s1,context.ma,context.level,'high')

low\_2 = history\_bars(context.s1,context.ma,context.level,'low')

close\_2 = history\_bars(context.s1,context.ma,context.level,'close')

#判断主力换月日期

s1\_leng =len(close\_2) #context.ma-0

#计算主力换月收盘次日开盘跳空

#主力合约数据长度不够造成两个list无法比较错误

if len(close\_2)<len(close\_1):

close=close\_1[-len(close\_2):]-close\_2

else: close=close\_1-close\_2

count =Counter(close)[0]

bias =open\_1[context.ma-count]/close\_1[context.ma-count-1]#复权因子

#构造复权主力合约

price\_bias\_a =[bias]\*(context.ma-count)

price\_bias\_b =[1]\*count

price\_bias =np.array(price\_bias\_a+price\_bias\_b)

context.open\_r =open\_1 \*price\_bias

context.high\_r =high\_1 \*price\_bias

context.low\_r =low\_1 \*price\_bias

context.close\_r =close\_1 \*price\_bias

plot('context.realized\_pnl', context.realized\_pnl)

plot('total\_value', context.portfolio.total\_value)

# after\_trading函数会在每天交易结束后被调用，当天只会被调用一次

def after\_trading(context):

#平仓盈亏曲线

context.day\_pnl=sum(context.pnl)

context.total\_pnl += context.day\_pnl

context.realized\_pnl = context.total\_pnl+context.account\_initial-context.cost

context.day\_pnl = 0

context.pnl = []

#最大回撤

if context.realized\_pnl > context.max:

context.max = context.realized\_pnl

context.min = context.realized\_pnl

elif context.realized\_pnl<context.max:

if context.realized\_pnl<context.min:

context.min = context.realized\_pnl

context.drawdown = (context.max-context.min)/context.max

if context.drawdown > context.MAX\_drawdown:

context.MAX\_drawdown = round(context.drawdown,3)

print(context.MAX\_drawdown)