```python
# -*- coding: utf-8 -*-
"""
Deltahedger for IB papertrading. Purpose of script is to instantiate an IB connection,
retrieve portfolio information, create risk overview in terms of greeks for each underlying
and hedge delta (for now) of the underlying by creating orders in the underlying.
@author: Jens
"""

from ib_insync import *
from tkinter import *
import pandas as pd
import datetime
import pytz
import sys
util.patchAsyncio()

conn_type="TWS"

# Connect to IB if connection not already established
try:
    if ib.isConnected()==True:
        pass
except NameError:
    ib = IB()
    if conn_type == "TWS": # Connect to TWS
        ib.connect('127.0.0.1', 7497, clientId=2)
    elif conn_type == "IBG": # Connect to IB Gateway
        ib.connect('127.0.0.1', 4002, clientId=1)
    else:
        print("please specify connection type. script exited.")
        sys.exit()

# Define several global variables needed
global target_delta_dic, hedge_threshold_dic, portfolio, portfolio_greeks

directory = "C:/Users/Jens/AnacondaProjects/IB/ibsync/Deltahedger/" # current  directory
data_directory = "C:/Users/Jens/AnacondaProjects/IB/ibsync/Deltahedger/datadirectory/" # Directo

def load_acc_values():
    # Function to load Current Account Values (Liquidity, margin etc) into a dictionary
    net_liq = [v for v in ib.accountValues() if v.tag == 'NetLiquidationByCurrency' and v.curren
    exc_liq = [v for v in ib.accountValues() if v.tag == 'ExcessLiquidity' and v.currency == 'EU
    acc_rdy = [v for v in ib.accountValues() if v.tag == 'AccountReady'][0].value
    gross_value = [v for v in ib.accountValues() if v.tag == 'GrossPositionValue' and v.currency
    prev_day_eq_w_loan= [v for v in ib.accountValues() if v.tag == 'PreviousDayEquityWithLoanVal
    reg_t_margin= [v for v in ib.accountValues() if v.tag == 'RegTMargin' and v.currency == 'EUR
    sma= [v for v in ib.accountValues() if v.tag == 'SMA' and v.currency == 'EUR'][0].value
    return {"net_liq": net_liq, "exc_liq": exc_liq, "acc_rdy": acc_rdy,
            "gr_value": gross_value, "prev_d_eq_w_loan": prev_day_eq_w_loan,
            "reg_t_margin": reg_t_margin, "sma": sma}

acc_values = load_acc_values() # Save current account values in acc_values

pd.options.mode.chained_assignment = None # Eliminate a random pandas error
positions =ib.positions() # Receive current positions
portfolio =util.df(positions)  # save positions as Pandas df
portfolio_greeks = pd.DataFrame() # Create a df which will be hosting greeks

#aggregated_delta={} # Create dictionary for aggregated delta / Out of use
#aggregated_gamma={} # Create dictionary for aggregated gamme / Out of use
```

```
def refresh_target_delta():
    global target_delta_dic, hedge_threshold_dic
    # Fill target_delta_dic and hedge_threshold with values from imported csv
    target_delta_dic={} # Create dicitonary which will contain target delta information. key = s
    hedge_threshold_dic={} # Create dictionary with hedge threshold. key = symbol
    target_delta_pd=pd.read_csv(directory + "target_delta.csv") # Import csv containing target
    counter=0
    for sy in target_delta_pd["symbol"]:
        if sy not in target_delta_dic:
            target_delta_dic[sy]=0

        target_delta_dic[sy]= target_delta_pd["target_delta"][counter]
        hedge_threshold_dic[sy]=target_delta_pd["threshold"][counter]
        counter+=1

    return target_delta_dic, hedge_threshold_dic

refresh_target_delta()

def active_trading():
    # Check if USA is currently trading. Currently not used
    t=datetime.datetime.now()
    if (t.weekday()==5) or (t.weekday()==6):
       print("Its weekend. Get a life.")
       return False
    elif 15 < t.hour >= 22 or (t.minute <30 and t.hour ==15):
       print("No trading hours")
       return False
    else:
        return True

def update_positions():
    global portfolio, portfolio_greeks
    # Function to update all positions in portfolio df. Deletes portfolio and greeks and refills
    counter=0

    refresh_target_delta() # Renew target delta of symbols and their hedge thresholds (in deltas

    portfolio = portfolio.iloc[0:0]
    portfolio_greeks = portfolio_greeks.iloc[0:0]

    positions=ib.reqPositions() # Checken ob req notwendig da zeitintesniver
    portfolio=util.df(positions) # Save positions as Pandas df
    #Add necessary columns into dataframe
    portfolio.insert(loc=len(portfolio.columns), column="ticker", value="")
    portfolio.insert(loc=len(portfolio.columns), column="con_details", value="")
    portfolio.insert(loc=len(portfolio.columns), column="market_active", value="")
    portfolio.insert(loc=len(portfolio.columns), column="delta", value=0.0)
    portfolio.insert(loc=len(portfolio.columns), column="share_delta", value=0.0)
    portfolio.insert(loc=len(portfolio.columns), column="ddelta", value=0.0)
    portfolio.insert(loc=len(portfolio.columns), column="gamma", value=0.0)
    portfolio.insert(loc=len(portfolio.columns), column="dgamma", value=0.0)
    portfolio.insert(loc=len(portfolio.columns), column="vega", value=0.0)
    portfolio.insert(loc=len(portfolio.columns), column="dvega", value=0.0)
    portfolio.insert(loc=len(portfolio.columns), column="theta", value=0.0)
    portfolio.insert(loc=len(portfolio.columns), column="dtheta", value=0.0)
    portfolio.insert(loc=len(portfolio.columns), column="implied_vol", value=0.0)
    portfolio.insert(loc=len(portfolio.columns), column="symbol", value="")
    portfolio.insert(loc=len(portfolio.columns), column="is_trading", value="")

    # Add several pieces of information to portfolio df
    for length in range(len(portfolio)):
        portfolio["contract"][length].exchange="SMART" # Add SMART exchange to contracts
        portfolio["con_details"][length]=ib.reqContractDetails(portfolio["contract"][length]) #
        portfolio["symbol"][length] = portfolio["contract"][length].symbol # Add symbol (of unde
        # Extract trading hours and liquid hours
        eu_time = pytz.timezone('Europe/Amsterdam')
        us_time = pytz.timezone('US/Eastern')
        liq_hours = portfolio["con_details"][length][0].liquidHours.split(';')
        today_hours = liq_hours[0]
        # Check if contract is trading and set value in portfolio to True or False
        if liq_hours[0].split(':')[1] == 'CLOSED':
            portfolio["is_trading"][length] = False # If underlying not trading today then False
        else:
            #If thre is trading today, check if we are within tradinghours
            now_time = datetime.datetime.now(pytz.timezone('US/Eastern')) # Current time in US
            start = datetime.datetime(year=int(today_hours[:4]), month=int(today_hours[4:6]), da
                                   hour= int(today_hours[9:11]), minute = int(today_hours[11:13
            start = start.astimezone(us_time) # Convert to US Time
            end = datetime.datetime(year=int(today_hours[:4]), month=int(today_hours[4:6]), day=
                                   hour= int(today_hours[23:25]), minute = int(today_hours[25:2
            end = end.astimezone(us_time) # Convert to US Time
            if end > now_time > start:
                portfolio["is_trading"][length] = True # If now_time is within trading hours, i
            else:
                portfolio["is_trading"][length] = False # if not -> is_trading = False

    portfolio_greeks=pd.DataFrame(index = portfolio["symbol"].unique()) # Create portfolio_gree
    # Fill dataframe with relevant columns and prefill with float 0
    portfolio_greeks.insert(loc=len(portfolio_greeks.columns), column="aggr_delta", value=0.0)
    portfolio_greeks.insert(loc=len(portfolio_greeks.columns), column="aggr_ddelta", value=0.0)
    portfolio_greeks.insert(loc=len(portfolio_greeks.columns), column="aggr_gamma", value=0.0)
    portfolio_greeks.insert(loc=len(portfolio_greeks.columns), column="aggr_dgamma", value=0.0)
    portfolio_greeks.insert(loc=len(portfolio_greeks.columns), column="aggr_vega", value=0.0)
    portfolio_greeks.insert(loc=len(portfolio_greeks.columns), column="aggr_dvega", value=0.0)
    portfolio_greeks.insert(loc=len(portfolio_greeks.columns), column="aggr_theta", value=0.0)
    portfolio_greeks.insert(loc=len(portfolio_greeks.columns), column="aggr_dtheta", value=0.0)
    portfolio_greeks.insert(loc=len(portfolio_greeks.columns), column="is_trading", value="")
    try:
        portfolio.to_csv(path_or_buf = directory + "portfolio.csv") # try to save file (not nece
    except PermissionError:
        print("Not saved to file. No permission. Code resumed")

    return

update_positions() # Create / update portfolio and portfolio_greeks

def mid_greek(ticker, greek):
    # Function which either returns modelGreek or self calculated midgreek if model not availab
    try:
        midgreek = float(getattr(ticker.modelGreeks,greek))
    except AttributeError:
        midgreek = (float(getattr(ticker.bidGreeks, greek)) + float(getattr(ticker.askGreeks, gr
    return midgreek

def update_greeks():
    # Function to update greeks for every contract in portfolio dataframe
    global portfolio, portfolio_greeks, target_delta_dic, hedge_threshold_dic # set globals, nee
    #aggregated_delta.clear() # reset aggregated deltas. currently dic not in use
    queue = [] # define queue variable to come back to unsolved symbols

    # Request Market data for every contract in portfolio
```

```
for length in range(len(portfolio)):
    portfolio["ticker"][length] = ib.reqMktData(portfolio["contract"][length], "", True)

# Loop through portfolio to assemble and calculate data / greeks
# If not add to queue (not functional yet)
for length in range(len(portfolio)):
    counter = 0
    no_data = True # Needs to be set to False to continue with symbol
    # Check if contract is trading. Else next contract.
    if portfolio["is_trading"][length] == True:
        portfolio_greeks["is_trading"][portfolio["symbol"][length]] = True
        # Check if market data available. If not wait until 5s then proceed
        # After 50 attempts add ticker to queue and proceed with rest
        while no_data == True:
            try:
                if portfolio["ticker"][length].contract.secType == "OPT":
                    check = mid_greek(portfolio["ticker"][length],"delta")
                    no_data = False # if no error received, break while loop
                elif portfolio["ticker"][length].contract.secType == "STK":
                    check = portfolio["ticker"][length].open
                    no_data = False # if no error received, break while loop
            except AttributeError:
                # catch if market data is not delivered
                counter += 1
                ib.sleep(0.1)
                if counter > 50:
                    print("Keine Daten für %s %s empfangen" % (portfolio["symbol"][length],
                    queue.append(length)
                    break # Stop checking if data is delivered. Breaks while loop
    elif no_data == True: # If no data has been received
        portfolio_greeks["is_trading"][portfolio["symbol"][length]] = False # Set underlyin
        #print("%s gequeued" % portfolio["ticker"][length].contract.symbol)
        continue # go to next contract

    # Determine greeks and add to correct column. Also calculate dollargreeks
    if portfolio["ticker"][length].contract.secType == "OPT":
        portfolio["delta"][length] = mid_greek(portfolio["ticker"][length],"delta")
        portfolio["share_delta"][length] = mid_greek(portfolio["ticker"][length],"delta")*fl
        portfolio["ddelta"][length] = mid_greek(portfolio["ticker"][length],"delta")*mid_gre
        portfolio["gamma"][length] = mid_greek(portfolio["ticker"][length],"gamma")
        portfolio["dgamma"][length] = mid_greek(portfolio["ticker"][length],"gamma")*float(p
        portfolio["theta"][length] = mid_greek(portfolio["ticker"][length],"theta")
        portfolio["dtheta"][length] = mid_greek(portfolio["ticker"][length],"theta")*float(p
        portfolio["vega"][length] = mid_greek(portfolio["ticker"][length],"vega")
        portfolio["dvega"][length] = mid_greek(portfolio["ticker"][length],"vega")*float(po
        portfolio["implied_vol"][length] = mid_greek(portfolio["ticker"][length],"impliedVo
    elif portfolio["ticker"][length].contract.secType == "STK":
        portfolio["delta"][length] = 1.0
        portfolio["share_delta"][length] = float(portfolio["position"][length])
        portfolio["ddelta"][length] = float(portfolio["position"][length])*((portfolio["tic
    else:
        print("Security is neither STK nor OPT. No values added")

# Leaves portfolio loop here
# Write resulting portfolio to csv
try:
    portfolio.to_csv(path_or_buf = directory + "portfolio.csv")
except PermissionError:
    print("portfolio not saved to file. No permission")
# aggregate greeks in portfolio_greeks
counter = 0
```

```
    for sy in portfolio["symbol"]: # Loop through every underlying and add greeks to each other
        if portfolio["share_delta"][counter] is None:
            print("None for counter %d" % counter)
            pass # only use values if they are floats
        else:
            portfolio_greeks["aggr_delta"][sy] += float(portfolio["share_delta"][counter])
            portfolio_greeks["aggr_ddelta"][sy] += float(portfolio["ddelta"][counter])
            portfolio_greeks["aggr_gamma"][sy] += float(portfolio["gamma"][counter])
            portfolio_greeks["aggr_dgamma"][sy] += float(portfolio["dgamma"][counter])
            portfolio_greeks["aggr_theta"][sy] += float(portfolio["theta"][counter])
            portfolio_greeks["aggr_dtheta"][sy] += float(portfolio["dtheta"][counter])
            portfolio_greeks["aggr_vega"][sy] += float(portfolio["vega"][counter])
            portfolio_greeks["aggr_dvega"][sy] += float(portfolio["dvega"][counter])

        counter += 1

    # Write resulting portfolio_greeks to csv
    try:
        portfolio_greeks.to_csv(path_or_buf = directory + "portfolio_greeks.csv", header = True,
    except PermissionError:
        print("portfolio_greeks not saved to file. No permission. Code resumed")

def create_deltahedges():
    global portfolio, portfolio_greeks, target_delta_dic, hedge_threshold_dic # set globals, nee
    refresh_target_delta()
    # Function which checks for every underlying portfolio_greeks if it is in target_delta_dic
    # If yes Calls a function to create an order to rebalance delta as configured in target_delt
    if acc_values["acc_rdy"] == False: # Only resume if account is ready for trading
        print("Account not ready for trading. Deltahedge aborted")
        return
    for row in portfolio_greeks.itertuples(): # Loop through all underlyings with greek exposure
        if (row.Index in target_delta_dic) and (portfolio_greeks["is_trading"][row.Index]==True)
            # Determine if buy or sell order needed
            if target_delta_dic[row.Index]< row.aggr_delta:
                buy_sell="SELL"
            else:
                buy_sell="BUY"
            # Call deltahedge on underlying if hedge threshold crossed
            if (abs((row.aggr_delta-target_delta_dic[row.Index])) > hedge_threshold_dic[row.Inde
                deltahedge(buy_sell,row.Index,row.aggr_delta,target_delta_dic[row.Index])
            else:
                print("No hedge in %s required. Delta is %d while threshold %d over target %d" %
                      (row.Index, row.aggr_delta, hedge_threshold_dic[row.Index], target_delta_d

    # After all orders are submitted, monitor them and amend if necessary
    order_fulfill()
    # To do: add all orders into a custom tradelog

def deltahedge(buy_sell,symbol,current_delta,target_delta):
    # Function which creates a limitorder to deltahedge an underlying
    amt=int(abs(round(target_delta-current_delta))) # Determine amount of stock needed to trade
    print("Create deltahedge for %s for %d stock" % (symbol, amt))
    lmt=0
    test=0
    contract = Stock(symbol, 'SMART', 'USD') # Create contract of underlying
    ib.qualifyContracts(contract) # qualify contract, fills in missing stuff like newly generate
    mktdata = ib.reqMktData(contract, "", True) # Request market data for contract
    #ib.sleep(1)
    while True: # Loop to wait for market data of contract
        try:
            test = mktdata.bid # try to use marketdata
            break # if no error, break While and proceed
        except AttributeError:
            ib.sleep(0.1) # if error occurs, sleep and go back into while loop until data arrive

    # Create liquidityadding limitorder on bid or ask of security
    # Future: gammadependent
    if buy_sell == "BUY":
        lmt = mktdata.bid
    elif buy_sell == "SELL":
        lmt = mktdata.ask

    # Create and print order. Does not place order yet
    order = LimitOrder(buy_sell, amt,lmt)
    print(symbol, order)

    trade = ib.placeOrder(contract, order) # Place order and finish function
    return

def order_fulfill():
    # Function to monitor and amend deltahedge orders until fulfilled or aborted
    counter=1
    orders=ib.openTrades() # receive openorders
    if orders == []:
        print("No open orders") # if all orders are executed or none have been created, exit fu
        return
    else: # if orders open
        while orders != []: # loop as long as orders are open or exit conditions fulfilled
            for t in orders: # t becomes an order object
                contract = t.contract # define contract for t
                mktdata = ib.reqMktData(contract, "", True) # request current marketdata
                while True: # Loop to wait for market data of contract
                    try:
                        test = mktdata.bid # try to use marketdata
                        break # if no error, break While and proceed
                    except AttributeError:
                        ib.sleep(0.1) # if error occurs, sleep and go back into while loop unti
                print("Changing order attempt %d / 5" % counter)
                # Amend order up to 5 times
                if counter <= 5:
                    try: # Changes order. Try necessary if order has been executed meanwhile
                        if t.order.action == "BUY":
                            t.order.lmtPrice = mktdata.bid
                            ib.placeOrder(contract, t.order)
                        elif t.order.action == "Sell":
                            t.order.lmtPrice = mktdata.ask
                            ib.placeOrder(contract, t.order)
                    except AssertionError: # if order already executed, refresh opentrades and 
                        orders=ib.openTrades()
                        continue
                # If spread > threshold cancel orders and wait for next deltahedge
                elif (counter >= 5) and (((mktdata.ask/mktdata.bid)-1) > 0.0010):
                    ib.cancelOrder(t.order) # cancel order
                    print("Order not fulfilled / not amended / cancelled")
                # If spread small cross spread via market order
                else:
                    new_order = MarketOrder(t.order.action, t.orderStatus.remaining)
                    ib.cancelOrder(t.order) # cancel old order
                    trade = ib.placeOrder(contract, new_order)
                    print("Amended to market order")

                counter += 1
            # Wait 5 seconds for order execution, request openorders and repeat if necessary
            ib.sleep(5)
            orders=ib.openTrades()
            # Avoid bugs by exiting after 7 attempts and cancelling order
            if counter > 7:
                ib.cancelOrder(t.order)
                print("Too many attempts -> aborted. Still orders open %s" % orders)
                break

def create_chain(underlying,sectype,exchange, *args):
    # Function to create option chain of an underlying
    # Arguments = symbol of underlying, Security type, Exchange
    save = True # whether to save as csv or not
    stuff=sectype(underlying,exchange)
    ib.qualifyContracts(stuff)
    ticker = ib.reqTickers(stuff) # Request data
    uvalue = ticker[0].marketPrice() # current marketprice. Just for validating
    global chains # make global to work with output while coding
    chains = ib.reqSecDefOptParams(stuff.symbol, '', stuff.secType, stuff.conId) # Receive opti
    chains = util.df(chains) # Transofrm into dataframe
    #print(chains)
    if save == True: # save if wanted
        chains.to_csv(path_or_buf = data_directory + underlying + ".csv")
    return chains

def clean_chain(df):
    # Function to clean chains and reduce expirations / strikes to those wanted.
    # Function not live yet
    markprice = 2770
    columns=["expirations","strikes"]
    for c in columns:
        counter1 = 0
        for rows in df[c]:
            str_list=[]
            for strikes in df[c][counter1]:
                str_list.append(strikes)
            df[c][counter1]=str_list

            counter1 += 1

def testchain():
    # Create a chain to work with in python shell and test stuff
    create_chain("SPX",Index,"CBOE")
    df=chains
    clean_chain(df)

def cr_order():
    # Create order to test stuff / Buy 100 AAPL at limit 101.26 (non-executable)
    order = LimitOrder("BUY", 100, 101.26)
    contract = Stock("AAPL", 'SMART', 'USD')
    trade = ib.placeOrder(contract, order)

def t():
    # Function to create a testorder and monitor it
    # For testing purposes
    cr_order()
    order_fulfill()

def hedge():
    # Update all information and deltahedge or not depending on relevant factors
    # Reduces type work when testing / hedging by calling 3 functions after another
    update_positions()
    update_greeks()
    create_deltahedges()

#keep_hedging()

#while active_trading() == True:
#    print("do stuff")


#def load_portfolio_from_csv():
#    # Does not worksince contracts are loaded as strings
#    global portfolio
#    aggregated_delta={}
#    counter=0
#    # Avoid reqTicker waittime
#    portfolio=pd.read_csv(directory + "portfolio.csv")
#    for sy in portfolio["symbol"]:
#        if sy not in aggregated_delta:
#            aggregated_delta[sy]=0
#
#        aggregated_delta[sy]+= portfolio["share_delta"][counter]
#        counter+=1
```