目　录

双均线策略(期货)

```
# coding=utf-8
from __future__ import print_function, absolute_import
from gm.api import *
import talib
import time

'''
本策略以DCE.i1801为交易标的，根据其一分钟(即60s频度）bar数据建立双均线模型，
短周期为30，长周期为60，当短期均线由上向下穿越长期均线时做空，
当短期均线由下向上穿越长期均线时做多,每次开仓前先平掉所持仓位，再开仓。
回测数据为:DCE.i1801的60s频度bar数据
回测时间为:2017-09-01 09:00:00到2017-09-30 15:00:00
'''

def init(context):
    context.FAST = 30                                                    # 短周期
    context.SLOW = 60                                                    # 长周期
    context.symbol = 'DCE.i1801'                                         # 订阅&交易标的
    context.period = context.SLOW + 1                                    # 订阅数据滑窗长度
    subscribe(context.symbol, '60s', count=context.period)               # 订阅行情

def on_bar(context, bars):
    print (bars[0].bob)
    # 获取数据
    prices = context.data('DCE.i1801', '60s', context.period, fields='close')
    # 计算长短周期均线
    fast_avg = talib.SMA(prices.values.reshape(context.period), context.FAST)
    slow_avg = talib.SMA(prices.values.reshape(context.period), context.SLOW)

    # 均线下穿，做空
    if slow_avg[-2] < fast_avg[-2] and slow_avg[-1] >= fast_avg[-1]:
      # 平多仓
        order_target_percent(symbol=context.symbol, percent=0,
```

```
    position_side=1, order_type=2)
34.       # 开空仓
35.         order_target_percent(symbol=context.symbol, percent=0.1,
    position_side=2, order_type=2)
36.
37.     # 均线上穿，做多
38.     if fast_avg[-2] < slow_avg[-2] and fast_avg[-1] >= slow_avg[-1]:
39.       # 平空仓
40.         order_target_percent(symbol=context.symbol, percent=0,
    position_side=2,order_type=2)
41.       # 开多仓
42.         order_target_percent(symbol=context.symbol, percent=0.1,
    position_side=1,order_type=2)
43.
44.
45. def on_execution_report(context, execrpt):
46.     # 打印委托执行回报
47.     print(execrpt)
48.
49.
50. if __name__ == '__main__':
51.     '''
52.     strategy_id策略ID,由系统生成
53.     filename文件名,请与本文件名保持一致
54.     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
55.     token绑定计算机的ID,可在系统设置-密钥管理中生成
56.     backtest_start_time回测开始时间
57.     backtest_end_time回测结束时间
58.     backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复
    权:ADJUST_POST
59.     backtest_initial_cash回测初始资金
60.     backtest_commission_ratio回测佣金比例
61.     backtest_slippage_ratio回测滑点比例
62.     '''
63.     run(strategy_id='strategy_id',
64.         filename='main.py',
65.         mode=MODE_BACKTEST,
66.         token='token_id',
67.         backtest_start_time='2017-09-01 09:00:00',
68.         backtest_end_time='2017-09-30 15:00:00',
69.         backtest_adjust=ADJUST_NONE,
70.         backtest_initial_cash=10000000,
71.         backtest_commission_ratio=0.0001,
72.         backtest_slippage_ratio=0.0001)
```

回测时间：2017-09-01 09:00:00 - 2017-09-30 15:00:00 | 期初资金：¥10,000,000.00元 | 成交比例：100.00% | 手续费率：0.01% | 滑点比例：0.01% | 复权方式：不复权 | 状态：回测

期初资金	总资产	累计盈亏	累计手续费	累计收益率	基准收益率(沪深300)	年化收益率	最大回撤	卡玛比率	夏普比率	开仓次数	平仓次数	胜率
10,000,000.00元	11,208,105.37元	1,208,105.37元	310,297.36元	12.08%	0.16%	152.05%	3.51%	43.29	4.01	149	147	35.37%

alpha对冲(股票+期货)

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
from gm.api import *

'''
本策略每隔1个月定时触发计算SHSE.000300成份股的过去的EV/EBITDA并选取EV/EBITDA大于0的股票
随后平掉排名EV/EBITDA不在最小的30的股票持仓并等权购买EV/EBITDA最小排名在前30的股票
并用相应的CFFEX.IF对应的真实合约等额对冲
回测数据为:SHSE.000300和他们的成份股和CFFEX.IF对应的真实合约
回测时间为:2017-07-01 08:00:00到2017-10-01 16:00:00
'''


def init(context):
    # 每月第一个交易日09:40:00的定时执行algo任务
    schedule(schedule_func=algo, date_rule='1m', time_rule='09:40:00')

    # 设置开仓在股票和期货的资金百分比(期货在后面自动进行杠杆相关的调整)
    context.percentage_stock = 0.4
    context.percentage_futures = 0.4


def algo(context):
    # 获取当前时刻
    now = context.now
    # 获取上一个交易日
    last_day = get_previous_trading_date(exchange='SHSE', date=now)
    # 获取沪深300成份股
    stock300 = get_history_constituents(index='SHSE.000300', start_date=last_day,
                                        end_date=last_day)[0]['constituents'].keys()
    # 获取上一个工作日的CFFEX.IF对应的合约
    index_futures = get_continuous_contracts(csymbol='CFFEX.IF', start_date=last_day, end_date=last_day)[-1]['symbol']
    # 获取当天有交易的股票
    not_suspended_info = get_history_instruments(symbols=stock300,
```

```
    start_date=now, end_date=now)
    not_suspended_symbols = [item['symbol'] for item in not_suspended_info if not item['is_suspended']]
    # 获取成份股EV/EBITDA大于0并为最小的30个
    fin = get_fundamentals(table='tq_sk_finindic', symbols=not_suspended_symbols,
                           start_date=now, end_date=now, fields='EVEBITDA',
                           filter='EVEBITDA>0', order_by='EVEBITDA', limit=30, df=True)
    fin.index = fin.symbol
    # 获取当前仓位
    positions = context.account().positions()
    # 平不在标的池或不为当前股指期货主力合约对应真实合约的标的
    for position in positions:
        symbol = position['symbol']
        sec_type = get_instrumentinfos(symbols=symbol)[0]['sec_type']
        # 若类型为期货且不在标的池则平仓
        if sec_type == SEC_TYPE_FUTURE and symbol != index_futures:
            order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Short)
            print('市价单平不在标的池的', symbol)
        elif symbol not in fin.index:
            order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print('市价单平不在标的池的', symbol)

    # 获取股票的权重
    percent = context.percentage_stock / len(fin.index)
    # 买在标的池中的股票
    for symbol in fin.index:
        order_target_percent(symbol=symbol, percent=percent, order_type=OrderType_Market,
                             position_side=PositionSide_Long)
        print(symbol, '以市价单调多仓到仓位', percent)

    # 获取股指期货的保证金比率
    ratio = get_history_instruments(symbols=index_futures, start_date=last_day, end_date=last_day)[0]['margin_ratio']
    # 更新股指期货的权重
    percent = context.percentage_futures * ratio
```

```
    # 买入股指期货对冲
    order_target_percent(symbol=index_futures, percent=percent, order_type=OrderType_Market,
                         position_side=PositionSide_Short)
    print(index_futures, '以市价单调空仓到仓位', percent)


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-07-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)
```


收益图
源代码
期初资金
10,000,000.00元
总资产
9,986,949.20元
累计盈亏
-13,050.80元
累计手续费
2,405.89元
累计收益率
-0.13%
基准收益率(沪深300)
5.09%
年化收益率
-0.54%
最大回撤
1.45%
卡玛比率
-0.37
夏普比率
-0.16
开仓次数
65
平仓次数
38
胜率
71.05%
策略收益
每日盈亏
每日买入
每日卖出
基准
请输入基准信号
沪深300
8.00%
6.00%
4.00%
2.00%
-2.00%
60,000元
40,000元
20,000元
-20,000元
-40,000元
-60,000元
9,000,000元
6,000,000元
3,000,000元
-3,000,000元
-6,000,000元
07-03 2017
07-13 2017
08-13 2017
09-13 2017
09-29 2017

集合竞价选股(股票)

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
from gm.api import *

'''
本策略通过获取SHSE.000300沪深300的成份股数据并统计其30天内
开盘价大于前收盘价的天数,并在该天数大于阈值10的时候加入股票池
随后对不在股票池的股票平仓并等权配置股票池的标的,每次交易间隔1个月.
回测数据为:SHSE.000300在2015-01-15的成份股
回测时间为:2017-07-01 08:00:00到2017-10-01 16:00:00
'''

def init(context):
    # 每月第一个交易日的09:40 定时执行algo任务
    schedule(schedule_func=algo, date_rule='1m', time_rule='09:40:00')
    # context.count_bench累计天数阙值
    context.count_bench = 10
    # 用于对比的天数
    context.count = 30
    # 最大交易资金比例
    context.ratio = 0.8

def algo(context):
    # 获取当前时间
    now = context.now
    # 获取上一个交易日
    last_day = get_previous_trading_date(exchange='SHSE', date=now)
    # 获取沪深300成份股
    context.stock300 = get_history_constituents(index='SHSE.000300', start_date=last_day,
                                                end_date=last_day)[0]['constituents'].keys()
    # 获取当天有交易的股票
    not_suspended_info = get_history_instruments(symbols=context.stock300, start_date=now, end_date=now)
    not_suspended_symbols = [item['symbol'] for item in
```

```
    not_suspended_info if not item['is_suspended']]

    trade_symbols = []
    if not not_suspended_symbols:
        print('没有当日交易的待选股票')
        return

    for stock in not_suspended_symbols:
        recent_data = history_n(symbol=stock, frequency='1d', count=context.count, fields='pre_close,open',
                                fill_missing='Last', adjust=ADJUST_PREV, end_time=now, df=True)
        diff = recent_data['open'] - recent_data['pre_close']
        # 获取累计天数超过阙值的标的池.并剔除当天没有交易的股票
        if len(diff[diff > 0]) >= context.count_bench:
            trade_symbols.append(stock)

    print('本次股票池有股票数目: ', len(trade_symbols))
    # 计算权重
    percent = 1.0 / len(trade_symbols) * context.ratio
    # 获取当前所有仓位
    positions = context.account().positions()
    # 如标的池有仓位,平不在标的池的仓位
    for position in positions:
        symbol = position['symbol']
        if symbol not in trade_symbols:
            order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print('市价单平不在标的池的', symbol)

    # 对标的池进行操作
    for symbol in trade_symbols:
        order_target_percent(symbol=symbol, percent=percent, order_type=OrderType_Market,
                             position_side=PositionSide_Long)
        print(symbol, '以市价单调整至权重', percent)


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
```

```
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-07-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)
```

[

]

多因子选股(股票)

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals

import numpy as np
from gm.api import *
from pandas import DataFrame

'''
本策略每隔1个月定时触发,根据Fama-French三因子模型对每只股票进行回归，得到其alpha值。
假设Fama-French三因子模型可以完全解释市场，则alpha为负表明市场低估该股，因此应该买入。
策略思路：
计算市场收益率、个股的账面市值比和市值,并对后两个进行了分类,
根据分类得到的组合分别计算其市值加权收益率、SMB和HML.
对各个股票进行回归(假设无风险收益率等于0)得到alpha值.
选取alpha值小于0并为最小的10只股票进入标的池
平掉不在标的池的股票并等权买入在标的池的股票
回测数据:SHSE.000300的成份股
回测时间:2017-07-01 08:00:00到2017-10-01 16:00:00
'''


def init(context):
    # 每月第一个交易日的09:40 定时执行algo任务
    schedule(schedule_func=algo, date_rule='1m', time_rule='09:40:00')
    print(order_target_percent(symbol='SHSE.600000', percent=0.5, order_type=OrderType_Market,
                               position_side=PositionSide_Long))
    # 数据滑窗
    context.date = 20
    # 设置开仓的最大资金量
    context.ratio = 0.8
    # 账面市值比的大/中/小分类
    context.BM_BIG = 3.0
    context.BM_MID = 2.0
    context.BM_SMA = 1.0
    # 市值大/小分类
    context.MV_BIG = 2.0
```

```
37.     context.MV_SMA = 1.0
38.     
39. 
40. # 计算市值加权的收益率,MV为市值的分类,BM为账目市值比的分类
41. def market_value_weighted(stocks, MV, BM):
42.     select = stocks[(stocks.NEGOTIABLEMV == MV) & (stocks.BM == BM)]
43.     market_value = select['mv'].values
44.     mv_total = np.sum(market_value)
45.     mv_weighted = [mv / mv_total for mv in market_value]
46.     stock_return = select['return'].values
47.     # 返回市值加权的收益率的和
48.     return_total = []
49.     for i in range(len(mv_weighted)):
50.         return_total.append(mv_weighted[i] * stock_return[i])
51.     return_total = np.sum(return_total)
52.     return return_total
53. 
54. 
55. def algo(context):
56.     # 获取上一个交易日的日期
57.     last_day = get_previous_trading_date(exchange='SHSE',
    date=context.now)
58.     # 获取沪深300成份股
59.     context.stock300 = get_history_constituents(index='SHSE.000300',
    start_date=last_day,
60.                                                 end_date=last_day)
    [0]['constituents'].keys()
61.     # 获取当天有交易的股票
62.     not_suspended =
    get_history_instruments(symbols=context.stock300,
    start_date=last_day, end_date=last_day)
63.     not_suspended = [item['symbol'] for item in not_suspended if not
    item['is_suspended']]
64.     fin = get_fundamentals(table='tq_sk_finindic',
    symbols=not_suspended, start_date=last_day, end_date=last_day,
65.                            fields='PB,NEGOTIABLEMV', df=True)
66.     
67.     # 计算账面市值比,为P/B的倒数
68.     fin['PB'] = (fin['PB'] ** -1)
69.     # 计算市值的50%的分位点,用于后面的分类
70.     size_gate = fin['NEGOTIABLEMV'].quantile(0.50)
71.     # 计算账面市值比的30%和70%分位点,用于后面的分类
72.     bm_gate = [fin['PB'].quantile(0.30), fin['PB'].quantile(0.70)]
73.     fin.index = fin.symbol
74.     x_return = []
```

```
    # 对未停牌的股票进行处理
    for symbol in not_suspended:
        # 计算收益率
        close = history_n(symbol=symbol, frequency='1d', count=context.date + 1, end_time=last_day, fields='close',
                          skip_suspended=True, fill_missing='Last', adjust=ADJUST_PREV, df=True)['close'].values
        stock_return = close[-1] / close[0] - 1
        pb = fin['PB'][symbol]
        market_value = fin['NEGOTIABLEMV'][symbol]
        # 获取[股票代码. 股票收益率, 账面市值比的分类, 市值的分类, 流通市值]
        if pb < bm_gate[0]:
            if market_value < size_gate:
                label = [symbol, stock_return, context.BM_SMA, context.MV_SMA, market_value]
            else:
                label = [symbol, stock_return, context.BM_SMA, context.MV_BIG, market_value]
        elif pb < bm_gate[1]:
            if market_value < size_gate:
                label = [symbol, stock_return, context.BM_MID, context.MV_SMA, market_value]
            else:
                label = [symbol, stock_return, context.BM_MID, context.MV_BIG, market_value]
        elif market_value < size_gate:
            label = [symbol, stock_return, context.BM_BIG, context.MV_SMA, market_value]
        else:
            label = [symbol, stock_return, context.BM_BIG, context.MV_BIG, market_value]
        if len(x_return) == 0:
            x_return = label
        else:
            x_return = np.vstack([x_return, label])

    stocks = DataFrame(data=x_return, columns=['symbol', 'return', 'BM', 'NEGOTIABLEMV', 'mv'])
    stocks.index = stocks.symbol
    columns = ['return', 'BM', 'NEGOTIABLEMV', 'mv']
    for column in columns:
        stocks[column] = stocks[column].astype(np.float64)
    # 计算SMB.HML和市场收益率
    # 获取小市值组合的市值加权组合收益率
```

```
    smb_s = (market_value_weighted(stocks, context.MV_SMA, context.BM_SMA) +
                market_value_weighted(stocks, context.MV_SMA, context.BM_MID) +
                market_value_weighted(stocks, context.MV_SMA, context.BM_BIG)) / 3

    # 获取大市值组合的市值加权组合收益率
    smb_b = (market_value_weighted(stocks, context.MV_BIG, context.BM_SMA) +
                market_value_weighted(stocks, context.MV_BIG, context.BM_MID) +
                market_value_weighted(stocks, context.MV_BIG, context.BM_BIG)) / 3

    smb = smb_s - smb_b
    # 获取大账面市值比组合的市值加权组合收益率
    hml_b = (market_value_weighted(stocks, context.MV_SMA, 3) +
                market_value_weighted(stocks, context.MV_BIG, context.BM_BIG)) / 2
    # 获取小账面市值比组合的市值加权组合收益率
    hml_s = (market_value_weighted(stocks, context.MV_SMA, context.BM_SMA) +
                market_value_weighted(stocks, context.MV_BIG, context.BM_SMA)) / 2

    hml = hml_b - hml_s
    close = history_n(symbol='SHSE.000300', frequency='1d', count=context.date + 1,
                          end_time=last_day, fields='close', skip_suspended=True,
                          fill_missing='Last', adjust=ADJUST_PREV, df=True)['close'].values
    market_return = close[-1] / close[0] - 1
    coff_pool = []
    # 对每只股票进行回归获取其alpha值
    for stock in stocks.index:
        x_value = np.array([[market_return], [smb], [hml], [1.0]])
        y_value = np.array([stocks['return'][stock]])
        # OLS估计系数
        coff = np.linalg.lstsq(x_value.T, y_value)[0][3]
        coff_pool.append(coff)

    # 获取alpha最小并且小于0的10只的股票进行操作(若少于10只则全部买入)
    stocks['alpha'] = coff_pool
```

```
    stocks = stocks[stocks.alpha < 0].sort_values(by='alpha').head(10)

    symbols_pool = stocks.index.tolist()
    positions = context.account().positions()

    # 平不在标的池的股票
    for position in positions:
        symbol = position['symbol']
        if symbol not in symbols_pool:
            order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print('市价单平不在标的池的', symbol)

    # 获取股票的权重
    percent = context.ratio / len(symbols_pool)
    # 买在标的池中的股票
    for symbol in symbols_pool:
        order_target_percent(symbol=symbol, percent=percent, order_type=OrderType_Market,
                             position_side=PositionSide_Long)
        print(symbol, '以市价单调多仓到仓位', percent)


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-07-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
```

```
184.         backtest_adjust=ADJUST_PREV,
185.         backtest_initial_cash=10000000,
186.         backtest_commission_ratio=0.0001,
187.         backtest_slippage_ratio=0.0001)
```

[

]

网格交易(期货)

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals

import numpy as np
import pandas as pd
from gm.api import *

'''
本策略首先计算了过去300个价格数据的均值和标准差
并根据均值加减1和2个标准差得到网格的区间分界线,
并分别配以0.3和0.5的仓位权重
然后根据价格所在的区间来配置仓位(+/-40为上下界,无实际意义):
(-40,-3],(-3,-2],(-2,2],(2,3],(3,40](具体价格等于均值+数字倍标准差)
[-0.5, -0.3, 0.0, 0.3, 0.5](资金比例,此处负号表示开空仓)
回测数据为:SHFE.rb1801的1min数据
回测时间为:2017-07-01 08:00:00到2017-10-01 16:00:00
'''


def init(context):
    context.symbol = 'SHFE.rb1801'
    # 订阅SHFE.rb1801, bar频率为1min
    subscribe(symbols=context.symbol, frequency='60s')
    # 获取过去300个价格数据
    timeseries = history_n(symbol=context.symbol, frequency='60s', count=300, fields='close', fill_missing='Last',
                           end_time='2017-07-01 08:00:00', df=True)['close'].values
    # 获取网格区间分界线
    context.band = np.mean(timeseries) + np.array([-40, -3, -2, 2, 3, 40]) * np.std(timeseries)
    # 设置网格的仓位
    context.weight = [0.5, 0.3, 0.0, 0.3, 0.5]


def on_bar(context, bars):
    bar = bars[0]
    # 根据价格落在(-40,-3],(-3,-2],(-2,2],(2,3],(3,40]的区间范围来获取最新收盘价所在的价格区间
    grid = pd.cut([bar.close], context.band, labels=[0, 1, 2, 3, 4])
```

```
[0]
    # 获取多仓仓位
    position_long = context.account().position(symbol=context.symbol, side=PositionSide_Long)
    # 获取空仓仓位
    position_short = context.account().position(symbol=context.symbol, side=PositionSide_Short)
    # 若无仓位且价格突破则按照设置好的区间开仓
    if not position_long and not position_short and grid != 2:
        # 大于3为在中间网格的上方,做多
        if grid >= 3:
            order_target_percent(symbol=context.symbol, percent=context.weight[grid], order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print(context.symbol, '以市价单开多仓到仓位', context.weight[grid])
        if grid <= 1:
            order_target_percent(symbol=context.symbol, percent=context.weight[grid], order_type=OrderType_Market,
                                 position_side=PositionSide_Short)
            print(context.symbol, '以市价单开空仓到仓位', context.weight[grid])
    # 持有多仓的处理
    elif position_long:
        if grid >= 3:
            order_target_percent(symbol=context.symbol, percent=context.weight[grid], order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print(context.symbol, '以市价单调多仓到仓位', context.weight[grid])
        # 等于2为在中间网格,平仓
        elif grid == 2:
            order_target_percent(symbol=context.symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print(context.symbol, '以市价单全平多仓')
        # 小于1为在中间网格的下方,做空
        elif grid <= 1:
            order_target_percent(symbol=context.symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print(context.symbol, '以市价单全平多仓')
```

```
            order_target_percent(symbol=context.symbol, percent=context.weight[grid], order_type=OrderType_Market,
                                 position_side=PositionSide_Short)
            print(context.symbol, '以市价单开空仓到仓位', context.weight[grid])
    # 持有空仓的处理
    elif position_short:
        # 小于1为在中间网格的下方,做空
        if grid <= 1:
            order_target_percent(symbol=context.symbol, percent=context.weight[grid], order_type=OrderType_Market,
                                 position_side=PositionSide_Short)
            print(context.symbol, '以市价单调空仓到仓位', context.weight[grid])
        # 等于2为在中间网格,平仓
        elif grid == 2:
            order_target_percent(symbol=context.symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Short)
            print(context.symbol, '以市价单全平空仓')
        # 大于3为在中间网格的上方,做多
        elif grid >= 3:
            order_target_percent(symbol=context.symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Short)
            print(context.symbol, '以市价单全平空仓')
            order_target_percent(symbol=context.symbol, percent=context.weight[grid], order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print(context.symbol, '以市价单开多仓到仓位', context.weight[grid])


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
```

```
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-07-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)
```

[

]

指数增强(股票)

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
import numpy as np
from gm.api import *
from pandas import DataFrame

'''
本策略以0.8为初始权重跟踪指数标的沪深300中权重大于0.35%的成份股.
个股所占的百分比为(0.8*成份股权重)*100%.然后根据个股是否:
1.连续上涨5天 2.连续下跌5天
来判定个股是否为强势股/弱势股,并对其把权重由0.8调至1.0或0.6
回测时间为:2017-07-01 08:50:00到2017-10-01 17:00:00
'''


def init(context):
    # 资产配置的初始权重,配比为0.6-0.8-1.0
    context.ratio = 0.8
    # 获取沪深300当时的成份股和相关数据
    stock300 = get_history_constituents(index='SHSE.000300', start_date='2017-06-30', end_date='2017-06-30')[0][
        'constituents']
    stock300_symbol = []
    stock300_weight = []

    for key in stock300:
        # 保留权重大于0.35%的成份股
        if (stock300[key] / 100) > 0.0035:
            stock300_symbol.append(key)
            stock300_weight.append(stock300[key] / 100)

    context.stock300 = DataFrame([stock300_weight], columns=stock300_symbol, index=['weight']).T
    print('选择的成分股权重总和为: ', np.sum(stock300_weight))
    subscribe(symbols=stock300_symbol, frequency='1d', count=5, wait_group=True)


def on_bar(context, bars):
    # 若没有仓位则按照初始权重开仓
    for bar in bars:
```

```
        symbol = bar['symbol']
        position = context.account().position(symbol=symbol, side=PositionSide_Long)
        if not position:
            buy_percent = context.stock300['weight'][symbol] * context.ratio
            order_target_percent(symbol=symbol, percent=buy_percent, order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print(symbol, '以市价单开多仓至仓位:', buy_percent)
        else:
            # 获取过去5天的价格数据,若连续上涨则为强势股,权重+0.2;若连续下跌则为弱势股,权重-0.2
            recent_data = context.data(symbol=symbol, frequency='1d', count=5, fields='close')['close'].tolist()
            if all(np.diff(recent_data) > 0):
                buy_percent = context.stock300['weight'][symbol] * (context.ratio + 0.2)
                order_target_percent(symbol=symbol, percent=buy_percent, order_type=OrderType_Market,

position_side=PositionSide_Long)
                print('强势股', symbol, '以市价单调多仓至仓位:', buy_percent)
            elif all(np.diff(recent_data) < 0):
                buy_percent = context.stock300['weight'][symbol] * (context.ratio - 0.2)
                order_target_percent(symbol=symbol, percent=buy_percent, order_type=OrderType_Market,

position_side=PositionSide_Long)
                print('弱势股', symbol, '以市价单调多仓至仓位:', buy_percent)


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复
```

```
    权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-07-01 08:50:00',
        backtest_end_time='2017-10-01 17:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)
```

跨品种套利(期货)

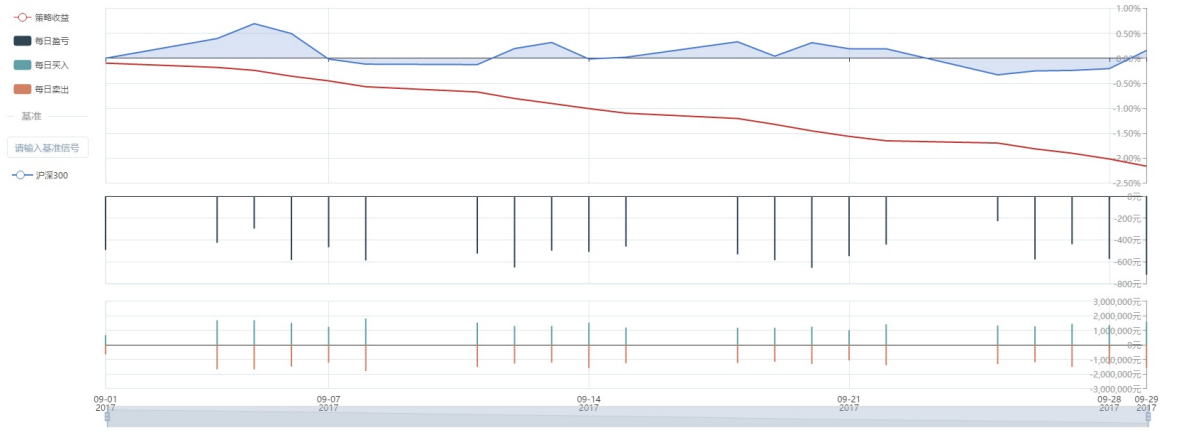
```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
from gm.api import *
import numpy as np

'''
本策略根据计算滚动的.过去的30个1min的bar的均值正负2个标准差得到布林线
并在最新价差上穿上轨来做空价差,下穿下轨来做多价差
并在回归至上下轨水平内的时候平仓
回测数据为:SHFE.rb1801和SHFE.hc1801的1min数据
回测时间为:2017-09-01 08:00:00到2017-10-01 16:00:00
'''


def init(context):
    # 进行套利的品种
    context.goods = ['SHFE.rb1801', 'SHFE.hc1801']
    # 订阅行情
    subscribe(symbols=context.goods, frequency='60s', count=31, wait_group=True)


def on_bar(context, bars):
    # 获取两个品种的时间序列
    data_rb = context.data(symbol=context.goods[0], frequency='60s', count=31, fields='close')
    close_rb = data_rb.values
    data_hc = context.data(symbol=context.goods[1], frequency='60s', count=31, fields='close')
    close_hc = data_hc.values
    # 计算价差
    spread = close_rb[:-1] - close_hc[:-1]
    # 计算布林带的上下轨
    up = np.mean(spread) + 2 * np.std(spread)
    down = np.mean(spread) - 2 * np.std(spread)
    # 计算最新价差
    spread_now = close_rb[-1] - close_hc[-1]
    # 无交易时若价差上(下)穿布林带上(下)轨则做空(多)价差
    position_rb_long = context.account().position(symbol=context.goods[0],
```

```
side=PositionSide_Long)
    position_rb_short = context.account().position(symbol=context.goods[0], side=PositionSide_Short)
    if not position_rb_long and not position_rb_short:
        if spread_now > up:
            order_target_volume(symbol=context.goods[0], volume=1, order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[0], '以市价单开空仓一手')
            order_target_volume(symbol=context.goods[1], volume=1, order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            print(context.goods[1], '以市价单开多仓一手')
        if spread_now < down:
            order_target_volume(symbol=context.goods[0], volume=1, order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            print(context.goods[0], '以市价单开多仓一手')
            order_target_volume(symbol=context.goods[1], volume=1, order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[1], '以市价单开空仓一手')
    # 价差回归时平仓
    elif position_rb_short:
        if spread_now <= up:
            order_close_all()
            print('价格回归,平所有仓位')
            # 跌破下轨反向开仓
        if spread_now < down:
            order_target_volume(symbol=context.goods[0], volume=1, order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            print(context.goods[0], '以市价单开多仓一手')
            order_target_volume(symbol=context.goods[1], volume=1, order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[1], '以市价单开空仓一手')
    elif position_rb_long:
        if spread_now >= down:
            order_close_all()
            print('价格回归,平所有仓位')
            # 涨破上轨反向开仓
        if spread_now > up:
```

```
            order_target_volume(symbol=context.goods[0], volume=1, order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[0], '以市价单开空仓一手')
            order_target_volume(symbol=context.goods[1], volume=1, order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            print(context.goods[1], '以市价单开多仓一手')


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-09-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=500000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)
```

收益图　源代码

期初资金	总资产	累计盈亏	累计手续费	累计收益率	基准收益率(沪深300)	年化收益率	最大回撤	卡玛比率	夏普比率	开仓次数	平仓次数	胜率
500,000.00元	489,192.63元	-10,807.37元	8,598.69元	-2.16%	0.16%	-27.20%	2.07%	-13.17	-71.32	1090	1090	51.19%



跨期套利(期货)

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals

import numpy as np
from gm.api import *

try:
    import statsmodels.tsa.stattools as ts
except:
    print('请安装statsmodels库')

'''
本策略根据EG两步法(1.序列同阶单整2.OLS残差平稳)判断序列具有协整关系之后(若无协整关系则全平仓位不进行操作)
通过计算两个真实价格序列回归残差的0.9个标准差上下轨,并在价差突破上轨的时候做空价差,价差突破下轨的时候做多价差
并在回归至标准差水平内的时候平仓
回测数据为:SHFE.rb1801和SHFE.rb1805的1min数据
回测时间为:2017-09-25 08:00:00到2017-10-01 15:00:00
'''


# 协整检验的函数
def cointegration_test(series01, series02):
    urt_rb1801 = ts.adfuller(np.array(series01), 1)[1]
    urt_rb1805 = ts.adfuller(np.array(series01), 1)[1]
    # 同时平稳或不平稳则差分再次检验
    if (urt_rb1801 > 0.1 and urt_rb1805 > 0.1) or (urt_rb1801 < 0.1 and urt_rb1805 < 0.1):
        urt_diff_rb1801 = ts.adfuller(np.diff(np.array(series01)), 1)[1]
        urt_diff_rb1805 = ts.adfuller(np.diff(np.array(series01), 1))[1]
        # 同时差分平稳进行OLS回归的残差平稳检验
        if urt_diff_rb1801 < 0.1 and urt_diff_rb1805 < 0.1:
            matrix = np.vstack([series02, np.ones(len(series02))]).T
            beta, c = np.linalg.lstsq(matrix, series01)[0]
            resid = series01 - beta * series02 - c
            if ts.adfuller(np.array(resid), 1)[1] > 0.1:
                result = 0.0
```

```
            else:
                result = 1.0
            return beta, c, resid, result

        else:
            result = 0.0
            return 0.0, 0.0, 0.0, result

    else:
        result = 0.0
        return 0.0, 0.0, 0.0, result


def init(context):
    context.goods = ['SHFE.rb1801', 'SHFE.rb1805']
    # 订阅品种
    subscribe(symbols=context.goods, frequency='60s', count=801, wait_group=True)


def on_bar(context, bars):
    # 获取过去800个60s的收盘价数据
    close_01 = context.data(symbol=context.goods[0], frequency='60s', count=801, fields='close')['close'].values
    close_02 = context.data(symbol=context.goods[1], frequency='60s', count=801, fields='close')['close'].values
    # 展示两个价格序列的协整检验的结果
    beta, c, resid, result = cointegration_test(close_01, close_02)
    # 如果返回协整检验不通过的结果则全平仓位等待
    if not result:
        print('协整检验不通过,全平所有仓位')
        order_close_all()
        return

    # 计算残差的标准差上下轨
    mean = np.mean(resid)
    up = mean + 0.9 * np.std(resid)
    down = mean - 0.9 * np.std(resid)
    # 计算新残差
    resid_new = close_01[-1] - beta * close_02[-1] - c
    # 获取rb1801的多空仓位
    position_01_long = context.account().position(symbol=context.goods[0], side=PositionSide_Long)
    position_01_short =
```

```
    context.account().position(symbol=context.goods[0],
    side=PositionSide_Short)
76.     if not position_01_long and not position_01_short:
77.         # 上穿上轨时做空新残差
78.         if resid_new > up:
79.             order_target_volume(symbol=context.goods[0], volume=1,
    order_type=OrderType_Market,
80.                                 position_side=PositionSide_Short)
81.             print(context.goods[0] + '以市价单开空仓1手')
82.             order_target_volume(symbol=context.goods[1], volume=1,
    order_type=OrderType_Market,
83.                                 position_side=PositionSide_Long)
84.             print(context.goods[1] + '以市价单开多仓1手')
85.         # 下穿下轨时做多新残差
86.         if resid_new < down:
87.             order_target_volume(symbol=context.goods[0], volume=1,
    order_type=OrderType_Market,
88.                                 position_side=PositionSide_Long)
89.             print(context.goods[0], '以市价单开多仓1手')
90.             order_target_volume(symbol=context.goods[1], volume=1,
    order_type=OrderType_Market,
91.                                 position_side=PositionSide_Short)
92.             print(context.goods[1], '以市价单开空仓1手')
93.     # 新残差回归时平仓
94.     elif position_01_short:
95.         if resid_new <= up:
96.             order_close_all()
97.             print('价格回归,平掉所有仓位')
98.         # 突破下轨反向开仓
99.         if resid_new < down:
100.            order_target_volume(symbol=context.goods[0], volume=1,
    order_type=OrderType_Market,
101.                                position_side=PositionSide_Long)
102.            print(context.goods[0], '以市价单开多仓1手')
103.            order_target_volume(symbol=context.goods[1], volume=1,
    order_type=OrderType_Market,
104.                                position_side=PositionSide_Short)
105.            print(context.goods[1], '以市价单开空仓1手')
106.    elif position_01_long:
107.        if resid_new >= down:
108.            order_close_all()
109.            print('价格回归,平所有仓位')
110.        # 突破上轨反向开仓
111.        if resid_new > up:
```

```
            order_target_volume(symbol=context.goods[0], volume=1, order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[0], '以市价单开空仓1手')
            order_target_volume(symbol=context.goods[1], volume=1, order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            print(context.goods[1], '以市价单开多仓1手')


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-09-25 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=500000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)
```

收益图 源代码

期初资金	总资产	累计盈亏	累计手续费	累计收益率	基准收益率(沪深300)	年化收益率	最大回撤	卡玛比率	夏普比率	开仓次数	平仓次数	胜率
500,000.00元	497,797.02元	-2,202.98元	1,326.49元	-0.44%	0.49%	-32.16%	0.40%	-81.17	-23.00	188	186	51.61%

日内回转交易(股票)

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
try:
    import talib
except:
    print('请安装TA-Lib库')
from gm.api import *

'''
本策略首先买入SHSE.600000股票10000股
随后根据60s的数据来计算MACD(12,26,9)线,并在MACD>0的时候买入100股,MACD<0的时候卖出100股
但每日操作的股票数不超过原有仓位,并于收盘前把仓位调整至开盘前的仓位
回测数据为:SHSE.600000的60s数据
回测时间为:2017-09-01 08:00:00到2017-10-01 16:00:00
'''


def init(context):
    # 设置标的股票
    context.symbol = 'SHSE.600000'
    # 用于判定第一个仓位是否成功开仓
    context.first = 0
    # 订阅浦发银行, bar频率为1min
    subscribe(symbols=context.symbol, frequency='60s', count=35)
    # 日内回转每次交易100股
    context.trade_n = 100
    # 获取昨今天的时间
    context.day = [0, 0]
    # 用于判断是否触发了回转逻辑的计时
    context.ending = 0


def on_bar(context, bars):
    bar = bars[0]
    if context.first == 0:
        # 最开始配置仓位
        # 需要保持的总仓位
        context.total = 10000
        # 购买10000股浦发银行股票
```

```
        order_volume(symbol=context.symbol, volume=context.total, side=PositionSide_Long,
                     order_type=OrderType_Market, position_effect=PositionEffect_Open)
        print(context.symbol, '以市价单开多仓10000股')
        context.first = 1.
        day = bar.bob.strftime('%Y-%m-%d')
        context.day[-1] = day[-2:]
        # 每天的仓位操作
        context.turnaround = [0, 0]
        return

    # 更新最新的日期
    day = bar.bob.strftime('%Y-%m-%d %H:%M:%S')
    context.day[0] = bar.bob.day
    # 若为新的一天,获取可用于回转的昨仓
    if context.day[0] != context.day[-1]:
        context.ending = 0
        context.turnaround = [0, 0]
    if context.ending == 1:
        return

    # 若有可用的昨仓则操作
    if context.total >= 0:
        # 获取时间序列数据
        symbol = bar['symbol']
        recent_data = context.data(symbol=symbol, frequency='60s', count=35, fields='close')
        # 计算MACD线
        macd = talib.MACD(recent_data['close'].values)[0][-1]
        # 根据MACD>0则开仓,小于0则平仓
        if macd > 0:
            # 多空单向操作都不能超过昨仓位,否则最后无法调回原仓位
            if context.turnaround[0] + context.trade_n < context.total:
                # 计算累计仓位
                context.turnaround[0] += context.trade_n
                order_volume(symbol=context.symbol, volume=context.trade_n, side=PositionSide_Long,
                             order_type=OrderType_Market, position_effect=PositionEffect_Open)
                print(symbol, '市价单开多仓', context.trade_n, '股')
        elif macd < 0:
            if context.turnaround[1] + context.trade_n <
```

```
    context.total:
78.                 context.turnaround[1] += context.trade_n
79.                 order_volume(symbol=context.symbol,
    volume=context.trade_n, side=PositionSide_Short,
80.                              order_type=OrderType_Market,
    position_effect=PositionEffect_Close)
81.                 print(symbol, '市价单平多仓', context.trade_n, '股')
82.         # 临近收盘时若仓位数不等于昨仓则回转所有仓位
83.         if day[11:16] == '14:55' or day[11:16] == '14:57':
84.             position =
    context.account().position(symbol=context.symbol,
    side=PositionSide_Long)
85.             if position['volume'] != context.total:
86.                 order_target_volume(symbol=context.symbol,
    volume=context.total, order_type=OrderType_Market,
87.                                     position_side=PositionSide_Long)
88.                 print('市价单回转仓位操作...')
89.                 context.ending = 1
90.         # 更新过去的日期数据
91.         context.day[-1] = context.day[0]
92.
93.
94. if __name__ == '__main__':
95.     '''
96.     strategy_id策略ID,由系统生成
97.     filename文件名,请与本文件名保持一致
98.     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
99.     token绑定计算机的ID,可在系统设置-密钥管理中生成
100.    backtest_start_time回测开始时间
101.    backtest_end_time回测结束时间
102.    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复
    权:ADJUST_POST
103.    backtest_initial_cash回测初始资金
104.    backtest_commission_ratio回测佣金比例
105.    backtest_slippage_ratio回测滑点比例
106.    '''
107.    run(strategy_id='strategy_id',
108.        filename='main.py',
109.        mode=MODE_BACKTEST,
110.        token='token_id',
111.        backtest_start_time='2017-09-01 08:00:00',
112.        backtest_end_time='2017-10-01 16:00:00',
113.        backtest_adjust=ADJUST_PREV,
114.        backtest_initial_cash=2000000,
```

```
115.         backtest_commission_ratio=0.0001,
116.         backtest_slippage_ratio=0.0001)
```

做市商交易(期货)

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals

from gm.api import *

'''
本策略通过不断对CZCE.CF801进行:
买(卖)一价现价单开多(空)仓和卖(买)一价平多(空)仓来做市
并以此赚取差价
回测数据为:CZCE.CF801的tick数据
回测时间为:2017-09-29 11:25:00到2017-09-29 11:30:00
需要特别注意的是:本平台对于回测对限价单固定完全成交,本例子 仅供参考.
敬请通过适当调整回测参数
1.backtest_commission_ratio回测佣金比例
2.backtest_slippage_ratio回测滑点比例
3.backtest_transaction_ratio回测成交比例
以及优化策略逻辑来达到更贴近实际的回测效果
'''


def init(context):
    # 订阅CZCE.CF801的tick数据
    context.symbol = 'CZCE.CF801'
    subscribe(symbols=context.symbol, frequency='tick')


def on_tick(context, tick):
    quotes = tick['quotes'][0]
    # 获取持有的多仓
    positio_long = context.account().position(symbol=context.symbol, side=PositionSide_Long)
    # 获取持有的空仓
    position_short = context.account().position(symbol=context.symbol, side=PositionSide_Short)
    print(quotes['bid_p'])
    print(quotes['ask_p'])
    # 没有仓位则双向开限价单
    # 若有仓位则限价单平仓
    if not positio_long:
```

```
        # 获取买一价
        price = quotes['bid_p']
        print('买一价为: ', price)
        order_target_volume(symbol=context.symbol, volume=1,
price=price, order_type=OrderType_Limit,
                            position_side=PositionSide_Long)
        print('CZCE.CF801开限价单多仓1手')
    else:
        # 获取卖一价
        price = quotes['ask_p']
        print('卖一价为: ', price)
        order_target_volume(symbol=context.symbol, volume=0,
price=price, order_type=OrderType_Limit,
                            position_side=PositionSide_Long)
        print('CZCE.CF801平限价单多仓1手')
    if not position_short:
        # 获取卖一价
        price = quotes['ask_p']
        print('卖一价为: ', price)
        order_target_volume(symbol=context.symbol, volume=1,
price=price, order_type=OrderType_Limit,
                            position_side=PositionSide_Short)
        print('CZCE.CF801卖一价开限价单空仓')
    else:
        # 获取买一价
        price = quotes['bid_p']
        print('买一价为: ', price)
        order_target_volume(symbol=context.symbol, volume=0,
price=price, order_type=OrderType_Limit,
                            position_side=PositionSide_Short)
        print('CZCE.CF801买一价平限价单空仓')


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复
权:ADJUST_POST
    backtest_initial_cash回测初始资金
```

```
77.     backtest_commission_ratio回测佣金比例
78.     backtest_slippage_ratio回测滑点比例
79.     backtest_transaction_ratio回测成交比例
80.     '''
81.     run(strategy_id='strategy_id',
82.         filename='main.py',
83.         mode=MODE_BACKTEST,
84.         token='token_id',
85.         backtest_start_time='2017-09-29 11:25:00',
86.         backtest_end_time='2017-09-29 11:30:00',
87.         backtest_adjust=ADJUST_PREV,
88.         backtest_initial_cash=500000,
89.         backtest_commission_ratio=0.00006,
90.         backtest_slippage_ratio=0.0001,
91.         backtest_transaction_ratio=0.5)
```

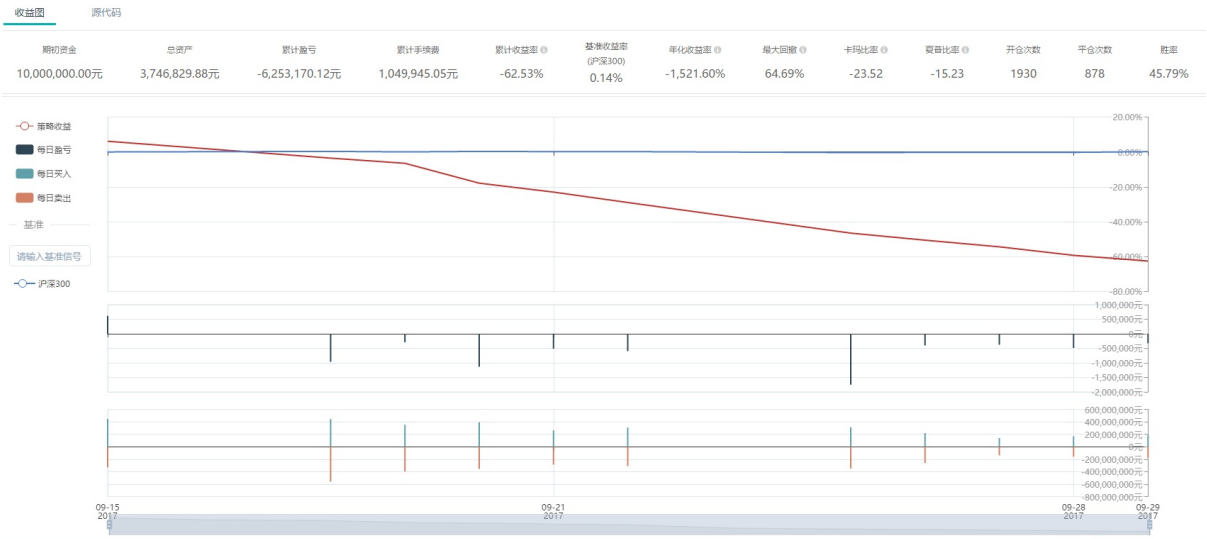
海龟交易法(期货)

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals

import numpy as np
import pandas as pd
try:
    import talib
except:
    print('请安装TA-Lib库')
from gm.api import *

'''
本策略通过计算CZCE.FG801和SHFE.rb1801的ATR.唐奇安通道和MA线,并:
上穿唐奇安通道且短MA在长MA上方则开多仓,下穿唐奇安通道且短MA在长MA下方则开空仓
若有 多/空 仓位则分别:
价格 跌/涨 破唐奇安平仓通道 上/下 轨则全平仓位,否则
根据 跌/涨 破持仓均价 -/+ x(x=0.5,1,1.5,2)倍ATR把仓位
回测数据为:CZCE.FG801和SHFE.rb1801的1min数据
回测时间为:2017-09-15 09:15:00到2017-10-01 15:00:00
'''


def init(context):
    # context.parameter分别为唐奇安开仓通道.唐奇安平仓通道.短ma.长ma.ATR的参数
    context.parameter = [55, 20, 10, 60, 20]
    context.tar = context.parameter[4]
    # context.goods交易的品种
    context.goods = ['CZCE.FG801', 'SHFE.rb1801']
    # context.ratio交易最大资金比率
    context.ratio = 0.8
    # 订阅context.goods里面的品种, bar频率为1min
    subscribe(symbols=context.goods, frequency='60s', count=101)
    # 止损的比例区间


def on_bar(context, bars):
    bar = bars[0]
    symbol = bar['symbol']
    recent_data = context.data(symbol=symbol, frequency='60s',
```

```
    count=101, fields='close,high,low')
40.     close = recent_data['close'].values[-1]
41.     # 计算ATR
42.     atr = talib.ATR(recent_data['high'].values,
    recent_data['low'].values, recent_data['close'].values,
43.                     timeperiod=context.tar)[-1]
44.     # 计算唐奇安开仓和平仓通道
45.     context.don_open = context.parameter[0] + 1
46.     upper_band = talib.MAX(recent_data['close'].values[:-1],
    timeperiod=context.don_open)[-1]
47.     context.don_close = context.parameter[1] + 1
48.     lower_band = talib.MIN(recent_data['close'].values[:-1],
    timeperiod=context.don_close)[-1]
49.     # 计算开仓的资金比例
50.     percent = context.ratio / float(len(context.goods))
51.     # 若没有仓位则开仓
52.     position_long = context.account().position(symbol=symbol,
    side=PositionSide_Long)
53.     position_short = context.account().position(symbol=symbol,
    side=PositionSide_Short)
54.     if not position_long and not position_short:
55.         # 计算长短ma线.DIF
56.         ma_short = talib.MA(recent_data['close'].values, timeperiod=
    (context.parameter[2] + 1))[-1]
57.         ma_long = talib.MA(recent_data['close'].values, timeperiod=
    (context.parameter[3] + 1))[-1]
58.         dif = ma_short - ma_long
59.         # 获取当前价格
60.         # 上穿唐奇安通道且短ma在长ma上方则开多仓
61.         if close > upper_band and (dif > 0):
62.             order_target_percent(symbol=symbol, percent=percent,
    order_type=OrderType_Market,
63.                                  position_side=PositionSide_Long)
64.             print(symbol, '市价单开多仓到比例: ', percent)
65.         # 下穿唐奇安通道且短ma在长ma下方则开空仓
66.         if close < lower_band and (dif < 0):
67.             order_target_percent(symbol=symbol, percent=percent,
    order_type=OrderType_Market,
68.                                  position_side=PositionSide_Short)
69.             print(symbol, '市价单开空仓到比例: ', percent)
70.     elif position_long:
71.         # 价格跌破唐奇安平仓通道全平仓位止损
72.         if close < lower_band:
73.             order_close_all()
```

```
            print(symbol, '市价单全平仓位')
        else:
            # 获取持仓均价
            vwap = position_long['vwap']
            # 获取持仓的资金
            money = position_long['cost']
            # 获取平仓的区间
            band = vwap - np.array([200, 2, 1.5, 1, 0.5, -100]) *
atr
            grid_percent = float(pd.cut([close], band, labels=[0,
0.25, 0.5, 0.75, 1])[0]) * percent
            # 选择现有百分比和区间百分比中较小的值(避免开仓)
            target_percent = np.minimum(money /
context.account().cash['nav'], grid_percent)
            if target_percent != 1.0:
                print(symbol, '市价单平多仓到比例: ', target_percent)
                order_target_percent(symbol=symbol,
percent=target_percent, order_type=OrderType_Market,

position_side=PositionSide_Long)

    elif position_short:
        # 价格涨破唐奇安平仓通道或价格涨破持仓均价加两倍ATR平空仓
        if close > upper_band:
            order_close_all()
            print(symbol, '市价单全平仓位')
        else:
            # 获取持仓均价
            vwap = position_short['vwap']
            # 获取持仓的资金
            money = position_short['cost']
            # 获取平仓的区间
            band = vwap + np.array([-100, 0.5, 1, 1.5, 2, 200]) *
atr
            grid_percent = float(pd.cut([close], band, labels=[1,
0.75, 0.5, 0.25, 0])[0]) * percent
            # 选择现有百分比和区间百分比中较小的值(避免开仓)
            target_percent = np.minimum(money /
context.account().cash['nav'], grid_percent)
            if target_percent != 1.0:
                order_target_percent(symbol=symbol,
percent=target_percent, order_type=OrderType_Market,

position_side=PositionSide_Short)
```

```
                print(symbol, '市价单平空仓到比例: ', target_percent)


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-09-15 09:15:00',
        backtest_end_time='2017-10-01 15:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)
```



行业轮动(股票)

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
import numpy as np
from gm.api import *

'''
本策略每隔1个月定时触发计算SHSE.000910.SHSE.000909.SHSE.000911.SHSE.000912.SHSE.000913.SHSE.00091
(300工业.300材料.300可选.300消费.300医药.300金融)这几个行业指数过去
20个交易日的收益率并选取了收益率最高的指数的成份股获取并获取了他们的市值数据
随后把仓位调整至市值最大的5只股票上
回测数据为:SHSE.000910.SHSE.000909.SHSE.000911.SHSE.000912.SHSE.000913.SHSE.000
和他们的成份股
回测时间为:2017-07-01 08:00:00到2017-10-01 16:00:00
'''


def init(context):
    # 每月第一个交易日的09:40 定时执行algo任务
    schedule(schedule_func=algo, date_rule='1m', time_rule='09:40:00')
    # 用于筛选的行业指数
    context.index = ['SHSE.000910', 'SHSE.000909', 'SHSE.000911', 'SHSE.000912', 'SHSE.000913', 'SHSE.000914']
    # 用于统计数据的天数
    context.date = 20
    # 最大下单资金比例
    context.ratio = 0.8


def algo(context):
    # 获取当天的日期
    today = context.now
    # 获取上一个交易日
    last_day = get_previous_trading_date(exchange='SHSE', date=today)
    return_index = []
    # 获取并计算行业指数收益率

```

```
    for i in context.index:
        return_index_his = history_n(symbol=i, frequency='1d', count=context.date, fields='close,bob',
                                     fill_missing='Last', adjust=ADJUST_PREV, end_time=last_day, df=True)
        return_index_his = return_index_his['close'].values
        return_index.append(return_index_his[-1] / return_index_his[0] - 1)
    # 获取指定数内收益率表现最好的行业
    sector = context.index[np.argmax(return_index)]
    print('最佳行业指数是: ', sector)
    # 获取最佳行业指数成份股
    symbols = get_history_constituents(index=sector, start_date=last_day, end_date=last_day)[0]['constituents'].keys()
    # 获取当天有交易的股票
    not_suspended_info = get_history_instruments(symbols=symbols, start_date=today, end_date=today)
    not_suspended_symbols = [item['symbol'] for item in not_suspended_info if not item['is_suspended']]

    # 获取最佳行业指数成份股的市值，从大到小排序并选取市值最大的5只股票
    fin = get_fundamentals(table='tq_sk_finindic', symbols=not_suspended_symbols, start_date=last_day,
                           end_date=last_day, limit=5, fields='NEGOTIABLEMV', order_by='-NEGOTIABLEMV', df=True)
    fin.index = fin['symbol']
    # 计算权重
    percent = 1.0 / len(fin.index) * context.ratio
    # 获取当前所有仓位
    positions = context.account().positions()
    # 如标的池有仓位,平不在标的池的仓位
    for position in positions:
        symbol = position['symbol']
        if symbol not in fin.index:
            order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print('市价单平不在标的池的', symbol)
    # 对标的池进行操作
    for symbol in fin.index:
        order_target_percent(symbol=symbol, percent=percent, order_type=OrderType_Market,
                             position_side=PositionSide_Long)
        print(symbol, '以市价单调整至仓位', percent)
```

```
69.
70.
71. if __name__ == '__main__':
72.     '''
73.     strategy_id策略ID,由系统生成
74.     filename文件名,请与本文件名保持一致
75.     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
76.     token绑定计算机的ID,可在系统设置-密钥管理中生成
77.     backtest_start_time回测开始时间
78.     backtest_end_time回测结束时间
79.     backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复
    权:ADJUST_POST
80.     backtest_initial_cash回测初始资金
81.     backtest_commission_ratio回测佣金比例
82.     backtest_slippage_ratio回测滑点比例
83.     '''
84.     run(strategy_id='strategy_id',
85.         filename='main.py',
86.         mode=MODE_BACKTEST,
87.         token='token_id',
88.         backtest_start_time='2017-07-01 08:00:00',
89.         backtest_end_time='2017-10-01 16:00:00',
90.         backtest_adjust=ADJUST_PREV,
91.         backtest_initial_cash=10000000,
92.         backtest_commission_ratio=0.0001,
93.         backtest_slippage_ratio=0.0001)
```

机器学习(股票)

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
from datetime import datetime
import numpy as np
from gm.api import *
import sys
try:
    from sklearn import svm
except:
    print('请安装scikit-learn库和带mkl的numpy')
    sys.exit(-1)

'''
本策略选取了七个特征变量组成了滑动窗口长度为15天的训练集,随后训练了一个二分类(上涨/下跌)的支持向量机模型.
若没有仓位则在每个星期一的时候输入标的股票近15个交易日的特征变量进行预测,并在预测结果为上涨的时候购买标的.
若已经持有仓位则在盈利大于10%的时候止盈,在星期五损失大于2%的时候止损.
特征变量为:1.收盘价/均值2.现量/均量3.最高价/均价4.最低价/均价5.现量6.区间收益率7.区间标准差
训练数据为:SHSE.600000浦发银行,时间从2016-03-01到2017-06-30
回测时间为:2017-07-01 09:00:00到2017-10-01 09:00:00
'''


def init(context):
    # 订阅浦发银行的分钟bar行情
    context.symbol = 'SHSE.600000'
    subscribe(symbols=context.symbol, frequency='60s')
    start_date = '2016-03-01'  # SVM训练起始时间
    end_date = '2017-06-30'  # SVM训练终止时间
    # 用于记录工作日
    # 获取目标股票的daily历史行情
    recent_data = history(context.symbol, frequency='1d', start_time=start_date, end_time=end_date, fill_missing='last',
                          df=True)
    days_value = recent_data['bob'].values
    days_close = recent_data['close'].values
    days = []
    # 获取行情日期列表
```

```
    print('准备数据训练SVM')
    for i in range(len(days_value)):
        days.append(str(days_value[i])[0:10])

    x_all = []
    y_all = []
    for index in range(15, (len(days) - 5)):
        # 计算三星期共15个交易日相关数据
        start_day = days[index - 15]
        end_day = days[index]
        data = history(context.symbol, frequency='1d',
start_time=start_day, end_time=end_day, fill_missing='last',
                       df=True)
        close = data['close'].values
        max_x = data['high'].values
        min_n = data['low'].values
        amount = data['amount'].values
        volume = []
        for i in range(len(close)):
            volume_temp = amount[i] / close[i]
            volume.append(volume_temp)

        close_mean = close[-1] / np.mean(close)  # 收盘价/均值
        volume_mean = volume[-1] / np.mean(volume)  # 现量/均量
        max_mean = max_x[-1] / np.mean(max_x)  # 最高价/均价
        min_mean = min_n[-1] / np.mean(min_n)  # 最低价/均价
        vol = volume[-1]  # 现量
        return_now = close[-1] / close[0]  # 区间收益率
        std = np.std(np.array(close), axis=0)  # 区间标准差

        # 将计算出的指标添加到训练集X
        # features用于存放因子
        features = [close_mean, volume_mean, max_mean, min_mean,
vol, return_now, std]
        x_all.append(features)

    # 准备算法需要用到的数据
    for i in range(len(days_close) - 20):
        if days_close[i + 20] > days_close[i + 15]:
            label = 1
        else:
            label = 0
        y_all.append(label)

    x_train = x_all[: -1]
```

```
    y_train = y_all[: -1]
    # 训练SVM
    context.clf = svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
                          tol=0.001, cache_size=200, verbose=False, max_iter=-1,
                          decision_function_shape='ovr', random_state=None)
    context.clf.fit(x_train, y_train)
    print('训练完成!')


def on_bar(context, bars):
    bar = bars[0]
    # 获取当前年月日
    today = bar.bob.strftime('%Y-%m-%d')
    # 获取数据并计算相应的因子
    # 于星期一的09:31:00进行操作
    # 当前bar的工作日
    weekday = datetime.strptime(today, '%Y-%m-%d').isoweekday()
    # 获取模型相关的数据
    # 获取持仓
    position = context.account().position(symbol=context.symbol, side=PositionSide_Long)
    # 如果bar是新的星期一且没有仓位则开始预测
    if not position and weekday == 1:
        # 获取预测用的历史数据
        data = history_n(symbol=context.symbol, frequency='1d', end_time=today, count=15,
                         fill_missing='last', df=True)
        close = data['close'].values
        train_max_x = data['high'].values
        train_min_n = data['low'].values
        train_amount = data['amount'].values
        volume = []
        for i in range(len(close)):
            volume_temp = train_amount[i] / close[i]
            volume.append(volume_temp)

        close_mean = close[-1] / np.mean(close)
        volume_mean = volume[-1] / np.mean(volume)
        max_mean = train_max_x[-1] / np.mean(train_max_x)
        min_mean = train_min_n[-1] / np.mean(train_min_n)
        vol = volume[-1]
```

```
119.         return_now = close[-1] / close[0]
120.         std = np.std(np.array(close), axis=0)
121. 
122.         # 得到本次输入模型的因子
123.         features = [close_mean, volume_mean, max_mean, min_mean,
    vol, return_now, std]
124.         features = np.array(features).reshape(1, -1)
125.         prediction = context.clf.predict(features)[0]
126.         # 若预测值为上涨则开仓
127.         if prediction == 1:
128.             # 获取昨收盘价
129.             context.price = close[-1]
130.             # 把浦发银行的仓位调至95%
131.             order_target_percent(symbol=context.symbol,
    percent=0.95, order_type=OrderType_Market,
132.                                  position_side=PositionSide_Long)
133.             print('SHSE.600000以市价单开多仓到仓位0.95')
134.     # 当涨幅大于10%,平掉所有仓位止盈
135.     elif position and bar.close / context.price >= 1.10:
136.         order_close_all()
137.         print('SHSE.600000以市价单全平多仓止盈')
138.     # 当时间为周五并且跌幅大于2%时,平掉所有仓位止损
139.     elif position and bar.close / context.price < 1.02 and weekday
    == 5:
140.         order_close_all()
141.         print('SHSE.600000以市价单全平多仓止损')
142. 
143. 
144. if __name__ == '__main__':
145.     '''
146.     strategy_id策略ID,由系统生成
147.     filename文件名,请与本文件名保持一致
148.     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
149.     token绑定计算机的ID,可在系统设置-密钥管理中生成
150.     backtest_start_time回测开始时间
151.     backtest_end_time回测结束时间
152.     backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复
    权:ADJUST_POST
153.     backtest_initial_cash回测初始资金
154.     backtest_commission_ratio回测佣金比例
155.     backtest_slippage_ratio回测滑点比例
156.     '''
157.     run(strategy_id='strategy_id',
158.         filename='main.py',
```

```
159.         mode=MODE_BACKTEST,
160.         token='token_id',
161.         backtest_start_time='2017-07-01 09:00:00',
162.         backtest_end_time='2017-10-01 09:00:00',
163.         backtest_adjust=ADJUST_PREV,
164.         backtest_initial_cash=10000000,
165.         backtest_commission_ratio=0.0001,
166.         backtest_slippage_ratio=0.0001)
```