www.portfolioeffect.com
High Frequency Portfolio Analytics

User Manual

PortfolioEffectHFT Package for R Software

Andrey Kostin
andrey.kostin@portfolioeffect.com

August 20, 2015

Contents

1 Package Installation

PortfolioEffectHFT package for R relies on the rJava package, which assumes that Java runtime is installed and configured on your system. To install Java runtime and to configure your R engine to work with it, follow these steps:

1.1 Install Latest JDK/JRE Runtime

Download and install latest Java distribution (JDK or JRE) for your platform from Oracle's website

1.2 Configure Java Environment (Optional)

If you are using Windows, installation wizard from the previous step should have done everything for you. If you are on Linux or Mac and you used a tarball file, you will need to manually append the following lines to /etc/environment using your favorite text editor:

```
export JAVA_HOME=/path/to/java/folder
export PATH=$PATH:$JAVA_HOME/bin
```

Apply environment changes:

```
source /etc/environment
```

To complete with the set-up of Java environment inside R, run the following line:

```
sudo R CMD javareconf
```

1.3 Install Required Packages (Optional)

If you are manually installing PortfolioEffectHFT package (you don't want to use CRAN repositories for some reason), you would need to install all required package dependencies first. Start R from the command line or in your GUI editor and type

```
install.packages(c("rJava", "ggplot2"))
```

You are now ready to install the PortfolioEffectHFT package directly from www.portfolioeffect.com downloads section.

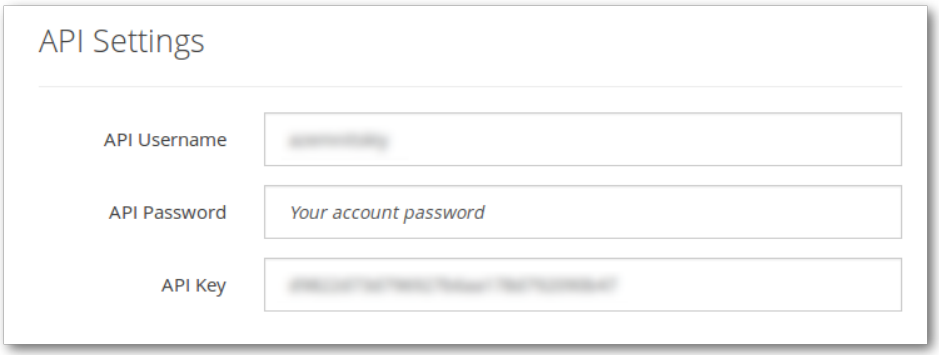
2 API Credentials

All portfolio computations are performed on PortfolioEffect cloud servers. To obtain a free non-professional account, you need to follow a quick sign-up process on our website: www.portfolioeffect.com/registration.

Please use a valid sign-up address - it will be used to email your account activation link.

2.1 Locate API Credentials

Log in to you account and locate your API credentials on the main page



2.2 Set API Credentials in R

Run the following commands to set your account API credentials for the PortfolioEffectHFT R Package installed. You will need to do it only once as your credentials are stored between sessions on your local machine to speed up future logons.

You would need to repeat this procedure if you change your account password or install PortfolioEffectHFT package on another computer.

```
require(PortfolioEffectHFT)
util_setCredentials("API Username", "API Password", "API Key")
```

You are now ready to call PortfolioEffectHFT methods.

3 Portfolio Construction

3.1 User Data

Users may supply their own historical datasets for index and position entries. This external data could be one a OHLC bar column element (e.g. 1-second close prices) or a vector of actual transaction prices that contains non-equidistant data points. You might want to pre-pend at least N =(4 x windowLength) data points to the beginning of the interval of interest which would be used for initial calibration of portfolio metrics.

3.1.1 Create Portfolio

Method portfolio_create(). takes a vector of index prices in the format (UTC timestamp, price) with UTC timestamp expressed in milliseconds from 1970-01-01 00:00:00 EST.

```
      Time          Value
 [1,] 1412256601000 99.30
 [2,] 1412256602000 99.33
 [3,] 1412256603000 99.30
 [4,] 1412256604000 99.26
 [5,] 1412256605000 99.36
 [6,] 1412256606000 99.36
 [7,] 1412256607000 99.36
 [8,] 1412256608000 99.38
 [9,] 1412256609000 99.40
[10,] 1412256610000 99.37
```

If index symbol is specified, it is silently ignored.

```
data(spy.data)

# Create portfolio
portfolio=portfolio_create(priceDataIx=spy.data)
```

3.1.2 Add Positions

Positions are added using portfolio_addPosition() with 'priceData' in the same format as index price.

```
data(goog.data)
data(aapl.data)

# Single position without rebalancing
portfolio_addPosition(portfolio,
            symbol='GOOG',
            quantity=100,
            priceData=goog.data)

# Single position with rebalancing
portfolio_addPosition(portfolio,
            symbol='AAPL',
            quantity=c(300,150),
```

```
            time=c("2014-09-01 09:00:00","2014-09-07 14:30:00"),
            priceData=aapl.data)
```

3.2 Server Data

At PortfolioEffect we are capturing and storing 1-second intraday bar history for a all NASDAQ traded equites. This server-side dataset spans from January 2013 to the latest trading time minus five minutes. It could be used to construct asset portfolios and compute intraday portfolio metrics.

3.2.1 Create Portfolio

Method portfolio_create() creates new asset portfolio or overwrites an existing portfolio object with the same name.

When using server-side data, it only requires a time interval that would be treated as a default position holding period unless positions are added with rebalancing. Index symbol could be specified as well with a default value of "SPY" - SPDR S&P 500 ETF Trust.

Interval boundaries are passed in the following format:

- "yyyy-MM-dd HH:MM:SS" (e.g. "2014-10-01 09:30:00")
- "yyyy-MM-dd" (e.g. "2014-10-01")
- "t-N" (e.g. "t-5" is latest trading time minus 5 days)
- UTC timestamp in milliseconds (mills from "1970-01-01 00:00:00") in EST time zone

```
# Timestamp in "yyyy-MM-dd HH:MM:SS" format
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

# Timestamp in "yyyy-MM-dd" format
portfolio=portfolio_create(fromTime="2014-10-01",
                           toTime="2014-10-02")

# Timestamp in "t-N" format
portfolio=portfolio_create(fromTime="t-5",
                           toTime="t")
```

3.2.2 Get Symbols List

Once portfolio is created, portfolio_availableSymbols() method could be called to receive the list of all available symbols for position creation. Each symbol is accompanied by a full company/instrument description and listing exchange name.

```
portfolio_availableSymbols(portfolio)
```

```
     id      description                                              exchange
 [1,] "BBC"   "BioShares Biotechnology Clinical Trials Fund"           "NASDAQ"
 [2,] "SCS"   "Steelcase Inc. Common Stock"                            "NYSE"
 [3,] "BBD"   "Banco Bradesco Sa American Depositary Shares"           "NYSE"
 [4,] "BBG"   "Bill Barrett Corporation Common Stock"                  "NYSE"
 [5,] "STPP"  "Barclays PLC - iPath US Treasury Steepener ETN"         "NASDAQ"
 [6,] "BBF"   "BlackRock Municipal Income Investment Trust"            "NYSE"
```

```
 [8,] "BBH"   "Market Vectors Biotech ETF"                              "NYSEARCA"
 [9,] "SCON"  "Superconductor Technologies Inc. - Common Stock"         "NASDAQ"
[10,] "SCX"   "L.S. Starrett Company (The) Common Stock"                "NYSE"
[11,] "BBK"   "Blackrock Municipal Bond Trust"                          "NYSE"
```

3.2.3 Add Positions

Positions are added by calling portfolio_addPosition() method on a portfolio object with a list of symbols and quantities. For positions that were rebalanced or had non-default holding periods a 'time' argument could be used to specify rebalancing timestamps.

```
# Single position without rebalancing
portfolio_addPosition(portfolio,
            symbol='GOOG',
            quantity=100)

# Multiple positions without rebalancing
portfolio_addPosition(portfolio,
            symbol=c('C','GOOG'),
            quantity=c(500,600))

# Single position with rebalancing
portfolio_addPosition(portfolio,
            symbol='AAPL',
            quantity=c(300,150),
            time=c("2014-09-01 09:00:00","2014-09-07 14:30:00"))
```

4 Portfolio Settings

4.1 Portfolio Metrics

These settings regulate how portfolio returns and return moments are computed

4.1.1 Portfolio Metrics Mode

One of the two modes for collecting portfolio metrics that could be used:

- "portfolio"- portfolio metrics are computed using previous history of position rebalancing. Portfolio risk and performance metrics account for the periods with no market exposure (i.e. when no positions are held) depending on the holding periods accounting settings (see holding periods mode below).
- "price" - at any given point of time, both position and portfolio metrics are computed for a buy-and-hold strategy. This mode is a common for classic portfolio theory and is often used in academic literature for portfolio optimization or when computing price statistics.

By default, mode is set to "portfolio".

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

portfolio_addPosition(portfolio,
            symbol='AAPL', quantity=c(300,150),
            time=c("2014-10-01 09:30:00","2014-10-02 09:30:00"))

portfolio_addPosition(portfolio,
            symbol='GOOG', quantity=c(100,150),
            time=c("2014-10-01 09:30:00","2014-10-02 09:30:00"))

# "price" mode
portfolio_settings(portfolio, portfolioMetricsMode="price")
variance_price=portfolio_variance(portfolio);

# "portfolio" mode
portfolio_settings(portfolio, portfolioMetricsMode="portfolio")
variance_portfolio=portfolio_variance(portfolio);
util_plot2d(variance_portfolio,title="Variance, portfolioMetricsMode",legend="price")+
    util_line2d(variance_price,legend="portfolio")
```

4.1.2 Holding Periods Only

This setting should only be used when portfolio metrics mode is set to "portfolio". When holdingPeriodsOnly is set to FALSE, trading strategy risk and performance metrics will be annualized to include time intervals when strategy had no market exposure at certain points (i.e. when position quantity were zero). When set to TRUE, trading strategy metrics are annualized only based on actual holding intervals.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

portfolio_addPosition(portfolio,'AAPL',
                      c(0,300,0,150),
                      time=c("2014-09-30 09:30:00",
                      "2014-10-01 09:30:00",
                      "2014-10-01 15:30:00",
                      "2014-10-02 11:30:00"))

# enable holdingPeriodsOnly
portfolio_settings(portfolio, holdingPeriodsOnly=TRUE)
variance_holdingPeriodsOnly_TRUE=portfolio_variance(portfolio);


# disable holdingPeriodsOnly
portfolio_settings(portfolio, holdingPeriodsOnly=FALSE)
variance_holdingPeriodsOnly_FALSE=portfolio_variance(portfolio);

util_plot2d(variance_holdingPeriodsOnly_TRUE,title="Variance,holdingPeriodsOnly",legend="TRUE")+ util_line2d(
    variance_holdingPeriodsOnly_FALSE,legend="FALSE")
```

4.1.3 Short Sales Mode

This setting is used to specify how position weights are computed. Available modes are:

- "lintner" - the sum of absolute weights is equal to 1 (Lintner assumption)
- "markowitz" - the sum of weights must equal to 1 (Markowitz assumption)

Defaults to "lintner", which implies that the sum of absolute weights is used to normalize investment weights.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

portfolio_addPosition(portfolio,
                      c('C','GOOG'),
                      c(-500,600))

# weights are normalized based on a simple sum (Markowitz)
portfolio_settings(portfolio, shortSalesMode="markowitz")
variance_markowitz=portfolio_variance(portfolio);

# weights are normalized based on a sum of absolute values (Lintner)
portfolio_settings(portfolio, shortSalesMode="lintner")
variance_lintner=portfolio_variance(portfolio);

util_plot2d(variance_markowitz,title="Variance,shortSalesMode",legend="markowitz")+ util_line2d(variance_lintner,
    legend="lintner")
```

4.2 Data Sampling

These settings regulate how results of portfolio computations are returned. Depending on your usage scenario, some of them might bring significantly imporvement to speed of your portfolio computations

4.2.1 Results Sampling Interval

Interval to be used for sampling computed results before returning them to the caller. Available interval values are:

- "Xs" - seconds
- "Xm" - minutes
- "Xh" - hours
- "Xd" - trading days (6.5 hours in a trading day)
- "Xw" - weeks (5 trading days in 1 week)
- "Xmo" - month (21 trading day in 1 month)
- "Xy" - years (256 trading days in 1 year)
- "none" - no sampling.
- "last" - only the very last data point is returned

Large sampling interval would produce smaller vector of results and would require less time spent on data transfer. Default value of "1s" indicates that data is returned for every second during trading hours.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-01 16:00:00")

portfolio_addPosition(portfolio,c('C','GOOG'),c(500,600))

# sample results every 30 seconds
portfolio_settings(portfolio, resultsSamplingInterval="30s")
variance_30s=portfolio_variance(portfolio);

# sample results every 5 minutes
portfolio_settings(portfolio, resultsSamplingInterval="15m")
variance_15m=portfolio_variance(portfolio);

util_plot2d(variance_30s,title="Variance,resultsSamplingInterval",legend="30s")+ util_line2d(variance_15m,legend="
    15m")
```

4.2.2 Input Sampling Interval

Interval to be used as a minimum step for sampling input prices. Available interval values are:

- "Xs" - seconds
- "Xm" - minutes
- "Xh" - hours
- "Xd" - trading days (6.5 hours in a trading day)
- "Xw" - weeks (5 trading days in 1 week)
- "Xmo" - month (21 trading day in 1 month)
- "Xy" - years (256 trading days in 1 year)
- "none" - no sampling

Default value is “none”, which indicates that no sampling is applied.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

portfolio_addPosition(portfolio,c('C','GOOG'),c(500,600))

# sample input prices every 30 seconds
portfolio_settings(portfolio, inputSamplingInterval="30s")
variance_30s=portfolio_variance(portfolio);

# sample input prices every 5 min
portfolio_settings(portfolio, inputSamplingInterval="5m")
variance_5m=portfolio_variance(portfolio);

util_plot2d(variance_30s,title="Variance,inputSamplingInterval",legend="30s")+ util_line2d(variance_5m,legend="5m"
    )
```

4.3 Model Pipeline

4.3.1 Window Length

Specifies rolling window length that should be used for computing portfolio and position metrics. When portfolio mode is set to “portfolio”, it is also the length of rebalancing history window to be used. Available interval values are:

- “Xs” - seconds
- “Xm” - minutes
- “Xh” - hours
- “Xd” - trading days (6.5 calendar hours in a trading day)
- “Xw” - weeks (5 trading days in 1 week)
- “Xmo” - month (21 trading day in 1 month)
- “Xy” - years (256 trading days in 1 year)
- “all” - all observations are used

Default value is “1d” - one trading day.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

portfolio_addPosition(portfolio,c('C','GOOG'),c(500,600))

# 1 hour rolling window
portfolio_settings(portfolio, windowLength="1h")
variance_1h=portfolio_variance(portfolio);

# 1 week rolling window
portfolio_settings(portfolio, windowLength="1d")
variance_1d=portfolio_variance(portfolio);

util_plot2d(variance_1h,title="Variance,windowLength",legend="1h")+ util_line2d(variance_1d,legend="1d")
```

4.3.2 Time Scale

Interval to be used for scaling return distribution statistics and producing metrics forecasts at different horizons. Available interval values are:

- "Xs" - seconds
- "Xm" - minutes
- "Xh" - hours
- "Xd" - trading days (6.5 hours in a trading day)
- "Xw" - weeks (5 trading days in 1 week)
- "Xmo" - month (21 trading day in 1 month)
- "Xy" - years (256 trading days in 1 year)
- "all" - actual interval specified during portfolio creation.

Default value is "1d" - one trading day.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

portfolio_addPosition(portfolio,c('C','GOOG'),c(500,600))

# 1 hour time scale
portfolio_settings(portfolio, timeScale="1h")
variance_1h=portfolio_variance(portfolio);

# 1 week time scale
portfolio_settings(portfolio, timeScale="1d")
variance_1d=portfolio_variance(portfolio);

util_plot2d(variance_1h,title="Variance,timeScale",legend="1h")+ util_line2d(variance_1d,legend="1d")
```

4.3.3 Microstructure Noise Model

Enables market mirostructure noise model of distribution returns.
Defaults to TRUE, which means that microstructure effects are modeled and resulting HF noise is removed from metric caluclations. When FALSE, HF microstructure noise is not separated from asset returns, which at high trading frequences could yield noise-contaminated results.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

portfolio_addPosition(portfolio,c('C','GOOG'),c(500,600))

# HF noise model is enabled
portfolio_settings(portfolio, noiseModel=TRUE)
variance_noiseModel_TRUE=portfolio_variance(portfolio);

# HF noise model is disabled
portfolio_settings(portfolio, noiseModel=FALSE)
variance_noiseModel_FALSE=portfolio_variance(portfolio);

util_plot2d(variance_noiseModel_TRUE,title="Variance,noiseModel",legend="TRUE")+ util_line2d(variance_noiseModel_
    FALSE,legend="FALSE")
```

4.3.4 Jumps/Outliers Model

Used to select jump filtering mode when computing return statistics. Available modes are:

- "none" - price jumps are not filtered anywhere
- "moments" - price jumps are filtered only when computing return moments (i.e. for expected return, variance, skewness, kurtosis and derived metrics)
- "all" - price jumps are filtered from computed returns, prices and all return metrics.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

portfolio_addPosition(portfolio,c('C','GOOG'),c(500,600))

# Price jumps detection is enabled for returns and moments
portfolio_settings(portfolio, jumpsModel="all")
variance_all=portfolio_variance(portfolio);

# Price jumps detection is disabled
portfolio_settings(portfolio, jumpsModel="none")
variance_none=portfolio_variance(portfolio);

util_plot2d(variance_all,title="Variance,jumpsModel",legend="all")+ util_line2d(variance_none,legend="none")
```

4.3.5 Density Model

Used to select density approximation model of return distribution. Available models are:

- "GLD" - Generalized Lambda Distribution
- "CORNER_FISHER" - Corner-Fisher approximation
- "NORMAL" - Gaussian distribution

Defaults to "GLD", which would fit a very broad range of distribution shapes.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

portfolio_addPosition(portfolio,c('C','GOOG'),c(500,600))

# Using normal density
portfolio_settings(portfolio, densityModel="NORMAL")
util_plotDensity(portfolio_pdf(portfolio,pValueLeft=0.6,pValueRight=1,nPoints=100,addNormalDensity=TRUE))

# Using Generalized Lambda density
portfolio_settings(portfolio, densityModel="GLD")
util_plotDensity(portfolio_pdf(portfolio,pValueLeft=0.6,pValueRight=1,nPoints=100,addNormalDensity=TRUE))
```

4.3.6 Factor Model

Factor model to be used when computing portfolio metrics. Available models are:

- "sim" - portfolio metrics are computed using the Single Index Model

- "direct" - portfolio metrics are computed using portfolio value itself (experimental)

Defaults to "sim", which implies that the Single Index Model is used to compute portfolio metrics.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

portfolio_addPosition(portfolio,c('C','GOOG'),c(500,600))

# Single Index Model is used
portfolio_settings(portfolio, factorModel="sim")
variance_sim=portfolio_variance(portfolio);

# Direct model is used
portfolio_settings(portfolio, factorModel="direct")
variance_direct=portfolio_variance(portfolio);

util_plot2d(variance_sim,title="Variance,factorModel",legend="sim")+ util_line2d(variance_direct,legend="direct")
```

4.3.7 Fractal Price Model

Used to enable mono-fractal price assumptions (fGBM) when time scaling return moments. Defaults to TRUE, which implies that computed Hurst exponent is used to scale return moments. When FALSE, price is assumed to follow regular GBM with Hurst exponent = 0.5.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

portfolio_addPosition(portfolio,c('C','GOOG'),c(500,600))

# Fractal price model is enabled
portfolio_settings(portfolio, fractalPriceModel=TRUE)
variance_fractal=portfolio_variance(portfolio);

# Fractal price model is disabled
portfolio_settings(portfolio, fractalPriceModel=FALSE)
variance_nonfractal=portfolio_variance(portfolio);

util_plot2d(variance_fractal,title="Variance,fractalPriceModel",legend="enabled")+ util_line2d(variance_nonfractal
    ,legend="disabled")
```

4.3.8 Drift Term

Used to enable drift term (expected return) when computing probability density approximation and related metrics (e.g. CVaR, Omega Ratio, etc.). Defaults to FALSE, which implies that distribution is centered around expected return.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00", toTime="2014-10-02 16:00:00")
portfolio_addPosition(portfolio,c('C','GOOG'),c(500,600))

# Drift term is enabled
portfolio_settings(portfolio, driftTerm=TRUE)
CVaR_driftTerm_TRUE=portfolio_CVaR(portfolio);

# Drift term is disabled
portfolio_settings(portfolio, driftTerm=FALSE)
CVaR_driftTerm_FALSE=portfolio_CVaR(portfolio);
```

```
util_plot2d(CVaR_driftTerm_TRUE,title="CVaR,driftTerm",legend="TRUE")+ util_line2d(CVaR_driftTerm_FALSE,legend="
    FALSE")
```

4.4 Transactional Costs

These settings provide a framework for adding variable and fixed transactional costs into return, expected return and profit calculations. All metrics based on expected return like Sharpe Ratio, VaR (with drift term enabled) would reflect transactional costsin their computations.

4.4.1 Cost Per Share

Amount of transaction costs per share. Default value is 0.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")
portfolio_addPosition(portfolio,c('C','GOOG'),c(500,600))

# Transactional costs per share are 0.5 cent
portfolio_settings(portfolio, txnCostPerShare=0.005)
return_5=portfolio_return(portfolio)

# Transactional costs per share are 0.1 cent
portfolio_settings(portfolio, txnCostPerShare=0.001)
return_1=portfolio_return(portfolio)

util_plot2d(return_5,title="Return,txnCostPerShare",legend="0.005")+
util_line2d(return_1,legend="0.001")
```

4.4.2 Cost Per Transaction

Amount of fixed costs per transaction. Defaults to 0.

```
require(PortfolioEffectHFT)
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")
portfolio_addPosition(portfolio,'AAPL',
                      c(100,300,500,150),
                      time=c("2014-10-01 09:30:00",
                      "2014-10-01 11:30:00",
                      "2014-10-01 15:30:00",
                      "2014-10-02 11:30:00"))

# Fixed costs per transaction are 9 dollars
portfolio_settings(portfolio, txnCostFixed=19.0)
return_19=portfolio_return(portfolio)

# Fixed costs per transaction are 1 dollar
portfolio_settings(portfolio, txnCostFixed=1.0)
return_1=portfolio_return(portfolio)

util_plot2d(return_19,title="Return,txnCostFixed",legend="19.0")+
util_line2d(return_1,legend="1.0")
```

5 Portfolio Optimization

5.1 Optimization Goals & Constraints

A classic problem of constructing a portfolio that meets certain maximization/minimization goals and constraints is addressed in our version of a multi-start portfolio optimization algorithm. At every time step optimization algorithm tries to find position weights that best meet optimization goals and constraints.

5.1.1 Key Features

- A multi-start approach is used to compare local optima with each other and select a global optimum. Local optima are computed using a modified method of parallel tangents (PARTAN).
- When optimization algorithm is supplied with mutually exclusive constraints, it would try to produce result that is equally close (in absolute terms) to all constraint boundaries. For instance, constraints "x $>$ 6" and "x $<$ 4" are mutually exclusive, so the optimization algorithm would choose "x = 5", which is a value that has the smallest distance to both constraints.
- Portfolio metrics change over time, but optimization uses only the latest value in the time series. Therefore, the faster metric series would change, the more likely current optimal weights would deviate from the optimal weights at the next time step.
- Optimization results depend on provided portfolio settings. For example, short windowLength would produce "spot" versions of portfolio metrics and computed optimal weights would change faster to reflect shortened metric horizon.

5.1.2 Optimization Goals

Optimization algorithm requires a single maximization/minimization goal to be set using optimization_goal() method that operates on a portfolio (see portfolio construction). Returned optimizer object could be used to add optional optimization costraints and then passed to the optimization_run method to launch portfolio optimization.

```
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00", toTime="2014-10-02 16:00:00")
portfolio_addPosition(portfolio, symbol=c('C','GOOG'),  quantity=c(500,600))
portfolio_settings(portfolio,
                   portfolioMetricsMode='price',
                   resultsSamplingInterval='30m')

# set optimization goal
optimizer=optimization_goal(portfolio,
                            goal="Return", direction="maximize")

# launch optimization and obtain optimal portfolio
optimalPortfolio=optimization_run(optimizer)

util_plot2d(portfolio_return(portfolio),title="Portfolio Return",legend="Simple Portfolio")+ util_line2d(portfolio
    _return(optimalPortfolio),legend="Optimal Portfolio")
```

The following portfolio metrics could currently be used as optimization goals:

“Variance”
: portfolio returns variance

“VaR”
: portfolio Value-at-Risk

“CVaR”
: portfolio Conditional Value-at-Risk (Expected Tail Loss)

“ExpectedReturn”
: portfolio expected return

“Return”
: portfolio return

“SharpeRatio”
: portfolio Sharpe Ratio

“ModifiedSharpeRatio”
: portfolio modified Sharpe Ratio

“StarrRatio”
: portfolio STARR Ratio

“ContraintsOnly”
: no optimization is performed. This is used for returning an arbitrary portfolio that meets specified set of constraints

“EquiWeight”
: no optimization is performed and constraints are not processes. Portfolio positions are returned with equal weights

5.1.3 Adding Constraints

Optimization constraints cover both metric-based and weight-based constraints.Metric-based constraints limit portfolio-level metrics to a certain range of values. For example, zero beta constraint would produce market-neutral optimal portfolio. Weight-based constraints operate on optimal position weights or sum of weights to give control over position concentration risks or short-sales assumptions.

Constraint methods could be chained to produce complex optimization rules:

Since position quantities are integer numbers and weights are decimals, a discretization error is introduced while converting optimal position weights to corresponding quantities. By default, optimal portfolio starts with a value of the initial portfolio. Portfolio value could be fixed to a constant level at every optimization step (see corresponding constraint below). Higher portfolio value could be used to keep difference between computed optimal weights and effective weights based on position quantities small. Lower portfolio value or higher asset price would normally increase discretization error.

```
# create portfolio and add positions
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00", toTime="2014-10-02 16:00:00")
portfolio_addPosition(portfolio, symbol=c('C','GOOG'),  quantity=c(500,600))
portfolio_settings(portfolio,
                   portfolioMetricsMode='price',
                   resultsSamplingInterval='30m')

# set optimization goal
optimizer=optimization_goal(portfolio, goal="SharpeRatio", direction="maximize")
```

```
# add constraints
optimization_constraint_beta(optimizer,"=",0)
optimization_constraint_weight(optimizer,">=",0.5, "GOOG")
optimization_constraint_variance(optimizer,"<=",0.02)

# launch optimization and obtain optimal portfolio
optimalPortfolio=optimization_run(optimizer)

util_plot2d(portfolio_sharpeRatio(portfolio),title="Sharpe Ratio",legend="Simple
Portfolio")+ util_line2d(portfolio_sharpeRatio(optimalPortfolio),legend="Optimal Portfolio")

util_plot2d(portfolio_beta(portfolio),title="Beta",legend="Simple Portfolio")+
util_line2d(portfolio_beta(optimalPortfolio),legend="Optimal Portfolio")

util_plot2d(portfolio_variance(portfolio),title="Variance",legend="Simple Portfolio")+
util_line2d(portfolio_variance(optimalPortfolio),legend="Optimal Portfolio")
```

The following constraint methods are available:

optimization_constraint_allWeights
: portfolio weights of all positions

optimization_constraint_weight
: portfolio position weights

optimization_constraint_sumOfAbsWeights
: portfolio's sum of absolute positions weights for selected positions

optimization_constraint_return
: portfolio return

optimization_constraint_expectedReturn
: portfolio expected return

optimization_constraint_variance
: portfolio returns variance

optimization_constraint_beta
: portfolio beta

optimization_constraint_VaR
: portfolio Value-at-Risk

optimization_constraint_CVaR
: portfolio Conditional Value-at-Risk (Expected Tail Loss)

optimization_constraint_modifiedSharpeRatio
: portfolio modified Sharpe Ratio

optimization_constraint_sharpeRatio
: portfolio Sharpe Ratio

optimization_constraint_starrRatio
: portfolio STARR Ratio

5.1.4 Scalar Constraints

Scalar constraints are the simplest type of optimization boundaries. They require a single constant that is applied over a full time span of portfolio optimization. An example below sets portfolio beta constraint to be greater or equal to 0.1.

```
# create portfolio and add positions
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")

portfolio_addPosition(portfolio,
                      symbol=c('C','GOOG'),
                      quantity=c(500,600))

# rebalancing every minute, static portfolio ignores rebalancing history
portfolio_settings(portfolio,
                   portfolioMetricsMode='price',
                   resultsSamplingInterval='30m')

# set optimization goal and define constraints
optimizer=optimization_goal(portfolio,
                            goal="SharpeRatio",
                            direction="maximize")
optimization_constraint_beta(optimizer, "<=", constraintValue = 0.1)

# run optimization
optimalPortfolio=optimization_run(optimizer)

# plot results
util_plot2d(portfolio_beta(optimalPortfolio), title="Beta", legend="Optimal Beta") +
util_line2d(portfolio_beta(portfolio), legend="Original Beta")
```

5.1.5 Vector Constraints

Instead of using a single scalar, one could specify an vector of constraint values with corresponding timestamps. Optimization algorithm would then automatically determine when certain constraint value should be applied based on the current rebalancing time.

```
# create portfolio and add positions
portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                           toTime="2014-10-02 16:00:00")
portfolio_addPosition(portfolio,
                      symbol=c("AAPL","GOOG"),
                      quantity=c(500,600))

# rebalancing every minute, static portfolio ignores rebalancing history
portfolio_settings(portfolio,
                   portfolioMetricsMode='price',
                   resultsSamplingInterval='30m')

betaVector = cbind(c("2014-10-01 09:30:00","2014-10-01 12:30:00"), c(0.1,0.5))

# set optimization goal and define constraints
optimizer=optimization_goal(portfolio,
                            goal="SharpeRatio",
                            direction="maximize")
optimizer=optimization_constraint_beta(optimizer, "<=",
                                       constraintValue = betaVector)

# run optimization
optimalPortfolio=optimization_run(optimizer)

# plot results
util_plot2d(portfolio_beta(optimalPortfolio), title="Beta", legend="Optimal Beta") +
util_line2d(portfolio_beta(portfolio), legend="Original Beta")
```

5.1.6 User-Defined Constraints

User-defined constraint relies on a provided function, which is called during portfolio optimization at specified rebalancing times. Function should return a scalar constraint value and would optionally receive portfolio and current time, if they are specified as arguments.

- function () - automatically converted to a scalar constraint
- function (time) - automatically converted to a vector constraint
- function (portfolio, time) - evaluated during optimization procedure

Note: When functional constraint specifies portfolio object as a required arguement, it could no longer be quickly converted to a scalar or vector-based constraint. In such case, optimization procedure would take a performance hit.

```
# create portfolio and add positions
my_portfolio=portfolio_create(fromTime="2014-10-01 09:30:00",
                              toTime="2014-10-02 16:00:00")
portfolio_addPosition(my_portfolio,
                      symbol=c('AAPL','GOOG'),
                      quantity=c(500,600))

# rebalancing every 30m, static portfolio ignores rebalancing history
portfolio_settings(my_portfolio,
                   portfolioMetricsMode='price',
                   resultsSamplingInterval='30m')

# define custom optimization constraint method for beta

myConstraint<-function(portfolio){
  # custom function
  portfolio_beta(portfolio)[,2]-0.2
}
# set optimization goal and define constraints
optimizer=optimization_goal(my_portfolio, goal="SharpeRatio", direction="maximize")
optimizer=optimization_constraint_beta(optimizer, ">=",
                                       constraintValue = myConstraint)

# run optimization
optimalPortfolio=optimization_run(optimizer)

# plot results
util_plot2d(portfolio_beta(optimalPortfolio), title="Beta", legend="Optimal Beta") +
util_line2d(portfolio_beta(my_portfolio), legend="Original Beta")
```