FX-Manager

Release 1.0.16

Abdullah Bahi

Dec 02, 2021

CONTENTS

FX-Manager is a python package for developing and testing forex algorithmic trading strategies.

CHAPTER

ONE

FEATURES

1. **Historical market data collection and pre-processing**
   - with FX-Manager, you can easily collect historical data from MT4 history center and preprocess it in just a few lines of code.
2. **Flexibility with historical data formats**
   - whether the data is OHLCV or bid-ask, whether it's daily or over a period of time, all what it takes is a simple configuration and the data will be ready to be used in FX-Managers applications.
3. **Open virtual trading account with no limitations**
   - most forex and stock brokers offer virtual trading accounts for new traders to learn trading with no risk of losing money. this might be sufficient for manual trading, but when it comes to algorithmic trading, it's not practical to use broker's virtual accounts to test trading strategies due to their limitations in terms of balance and time it takes to open a new account when an account is exhausted.
   - with FX-Manager, there is no need to be constrained with broker's limitations, instead of performing actual trading in your broker's account each time a trading strategy is tested, FX-Manager creates an instance of a virtual account with all properties as an actual account and performs trading simulation on this instance without affecting the actual account on MT4 trading platform.
   - user can initialize this instance with any amount of balance and any account type with no limitations.
4. **Run real-time trading simulation using real market prices**
   - using FX-Manager's virtual accounts, you can test your user-defined trading strategy by runinig real-time trading simulation with real market prices from MT4 live price feed.
   - this allows you to run multiple simulations in the same time because each simulation uses it's own account instance without affecting the account connected with MT4. MT4 only acts as a price feed source for all running instances.
   - the simulation shows account state every time period that is defined by user.
5. **Run historical trading simulation using historical market prices**
   - in addition to real-time testing, you can also test your trading strategies using historical data and preview results in a very nice and well-stractured manner.
6. **Biult-in portfolio construction and optimization**
   - what currency pairs to invest in and what time frame should be used for each currency pair are both the most important questions for any forex trader.
   - with FX-Manager portfolio construction feature. you can construct optimized portfolios consisting of diversified collection of currency pairs in the best time frame for each pair. these portfolios are constructed with historical data and can be used for historical and live trading simulations.

CHAPTER TWO

INSTALLATION

Fx-Manager framework is mainly used to create client server applications where the client is the trading bot written in python using FX-Manager's APIs, and the server is an expert advisor that is installed on MT4 trading platform. To utilize the full capacity of the framework, the Following packages & softwares need to be installed and configured.

- **FX-Manager python framework**
- **MetaTrader4 (MT4) Trading Platform**
- **Darwenix MQL4 Expert Advisor on MT4**

In this step-by-step Guild, we'll install all the above requirments.

1. **Installing FX-Manager python framework.**

    - **Using pip** *pip install FX-Manager*

2. **Installing MetaTrader4 (MT4) Trading Platform.**

    > **NOTE**: If you already have installed MT4 software and registered an active trading account, feel free to skip to next step.

    In order to use MT4 trading platform, you need to create virtual or actual trading account with a broker that supports the platform. then log in with the account on MT4 platform. There are many broker's available that support MT4, for this guild we will use a demo acoount offered by XM.COM.

    Follow the steps in this Youtube Tutorial1 by The Forex Tutor to create a demo account on XM.COM, download and install MT4 platform and finally log in with the account on the platform.

3. **Installing Darwenix MQL4 Expert Advisor on MT4.**

    FX-Manager uses Darwinex ZeroMQ Connector to connect python clients to MetaTrader. So we need to install and configure Darwinex's Expert Advisor on MT4 platform.

    Follow the steps in this Youtube Tutorial2 by Darwinex's Official channel to install and configure the EA.

    > **NOTE**: Skip the part from 1:30 to 4:10 in the tutorial.

Now you are all setup to start developing algorithmic trading strategies in python using FX-Manager. Refer to **Tutorials** Section to get started.

CHAPTER

THREE

DOCUMENTATION

3.1 API Reference

3.1.1 Subpackages

fxmanager.basic package

Package contents

This sub-package contains mudules of helper functions used intrnally in all FX-Manager's sub-packages.

in addition to the helper functions, the sub-package contains functions used for setting up new FX-Manager projects and preprocessing data to be compaitable with FX-Manager. it also contains the module used for simulating the trading accounts.

Modules:

- util : This module contains helper functions used internally in other fxmanager sub-packages. it also includes public functions to be used by the user to setup fxmanager project structure and pre-process the data.
- account : This module contains 'Account' class which is used for creating a fully functional virtual forex trading accounts.

fxmanager.basic.account module

This module contains 'Account' class which is used for creating a fully functional virtual forex trading accounts.

used for trading simulation and strategies backtesting. first the user creates an account instance with the required paramaters, then this instance is passed to fxmanager's built-in historical or live simulators.

Classes:

- Account: Class that simulates forex trading accounts.

`class fxmanager.basic.account.`**`Account`**(*balance=100000.0*, *account_type='standard'*, *account_currency='usd'*, *leverage=0.01*, *volume_bounds=(0.01, 8.0)*)

Bases: `object`

Class that simulates forex trading accounts.

Args:

- balance : float with initial balance in the account.

- **account_type** [string idicating account type, supported account types are]
  - standard: lot size = 100,000
  - mini : lot size = 10,000
  - micro : lot size = 1000
  - nano : lot size = 100
- account_currency: string indicating the account currency (currency of initial balance).
- leverage : float indicating the leverage provided by the broker
- volume_bounds : tuple of 2 floats, minimum and maximum availabe trading volumes per one position.

Public Methods:

- open_pisition() : opens a new position and updates the state of the account accordingly.
- close_position(): closes an opened position and updates the state of the account accordingly.
- update() : updates the state of the account (including oppened positions) with the current market prices.

`close_position`(*ticket*, *prices*)
closes an opened position and updates the state of the account accordingly.

`open_pisition`(*ticket*, *base_currency*, *quote_currency*, *time_frame*, *weight*, *SL*, *TP*, *order_type*, *prices*, *period*, *order_idx*, *risk_factor=0.8*)
opens a new position and updates the state of the account accordingly.

`update`(*prices*, *dynamic_sltp=False*)
updates the state of the account (including oppened positions) with the current market prices.

fxmanager.basic.util module

This module contains helper functions used internally in other fxmanager sub-packages. it also includes public functions to be used by the user to setup fxmanager project structure and pre-process the data.

Public Functions:

- get_portfolios(): gets the portfolios to be used in historical backtesting of strategies.
- setup() : creates the required folders and files for any new project according to application type.
- preprocess() : preprocesses the raw data files to prepare it for analysis and simumlations.

`fxmanager.basic.util.expected_return`(*day=0*, *data_dir=None*, *time_frames=[]*, *currency_pairs=[]*, *weights=[]*, *leverage=0.01*, *win_rate=0.5*, *risk_reward=1*, *n_scenarios=100*)
calculates the average expected daily return from a portfolio given a win rate and a risk-reward ratio.

Args:

- day : integer with range from 0 to (number of working days in the dataset-1).
- data_dir : string with the directory or full path to the directory in which application data is kept. If the setup() function is used to create the recommended project structure, the default None value should be used.
- time_frames : list of portfolio timeframes.
- currency_pairs: list of portfolio currency pairs.
- weights : list of portfolio weights.

- leverage : float indicating the leverage. if no leverage is used set to 1.
- win_rate : float indicating the percentage of winning positions to the total number of positions.
- risk_reward : float indicating ratio of reward to risk.
- n_scenarios : integer indicating the number of scenarios across which the expected return is averaged.

Returns:

- avg_expected_return: float indicating the average expected return.
- **win_probability** [float indicating the ratio of number of scenarios in which the expected return is positive to the total number] of scenarios.

Usage:

- used in 'fxmanage.optimization.eqw_optimization_test.run_test()' function to calculate the expected returns using the optimized portfolios.

`fxmanager.basic.util.`**`findCombinations`**`(`*n*`)`

finds all possible combinations that sum up to n.

Args:

- n: integer idicating the number that returned cambinations sum up to.

Returns:

- every combination is printed in a new line to standard output. to save the results to a file, change sys.stdout to the path of the file.

Usage:

- used in 'fxmanager.basic.util.setup()' function to save required weight combinations to 'data_dirweigts' directory.

`fxmanager.basic.util.`**`findCombinationsUtil`**`(`*arr*, *index*, *num*, *reducedNum*`)`

helper function for findCombinations() function.

`fxmanager.basic.util.`**`get_avg_rets`**`(`*day*, *data_dir=None*, *time_frames=[]*, *currency_pairs=[]*`)`

gets the average best & worst returns of the passed currency pairs & timeframes from the preprocessed data in the 'data_dirtime_frames' directory.

Args:

- day : integer indicating day number - starts from 0 and ends with the total number of work days in the dataset.
- data_dir : string with the directory or full path to the directory in which application data is kept. If the setup() function is used to create the recommended project structure, the default None value should be used.
- time_frames : a list of strings of the timeframes from which the data is read.
- currency_pairs: a list of strings of the currency pairs from which the data is read.

Returns:

- avg_best_rets: pandas dataframe of average best return for each asset, columns are time_frames and index is currency_pairs.
- avg_wrst_rets: pandas dataframe of average worst return for each asset, columns are time_frames and index is currency_pairs.

Usage:

- used in ‘fxmanager.simulation.live.run()’ function if the construct_portfolio is set to True.
- used in ‘fxmanager.optimization.w_optimization_test’ and ‘fxmanager.optimization.eqw_optimization_test’ modules.

`fxmanager.basic.util.`**`get_portfolios`**`(`*data_dir=None*, *num_days=1*, *single_portfolio={}*, *portfolios=[]*, *optimized_portfolios=False*, *use_single_porfolio=False*`)`

gets the portfolios to be used in historical backtesting of strategies.

Args:

- data_dir : string with the directory or full path to the directory in which application data is kept. If the setup() function is used to create the recommended project structure, the default None value should be used.
- num_days : total number of work days in the dataset.
- single_portfolio: dictionary of the portfolio to be used for backtesting with keys (currency_pairs, time_frames, weights) used if ‘optimized_portfolios’ is False and ‘use_single_porfolio’ is True.
- portfolios : list of dictionries like ‘single_portfolio’. Lenght of the list = num_days-1. used if ‘optimized_portfolios’ is False and ‘use_single_porfolio’ is False.
- optimized_portfolios: boolean flag, set to True if an optimization test is run and the portfolios to be used are saved in ‘data_dirstatsOptimizer_Test_portfolios.csv’ file.
- use_single_porfolio: boolean flag, used if ‘optimized_portfolios’ is False, set to true if you passed a single portfolio in the ‘single_portfolio’ argument, if False, ‘portfolios’ argument is used instead.

Returns:

- df: pandas dataframe with columns (currency_pairs, time_frames, weights) and range index of length = num_days

Usage:

- used by user before calling the function ‘fxmanager.simulation.historic.run()’.

`fxmanager.basic.util.`**`get_weights`**`(`*data_dir=None*, *num_assets=1*`)`

reads the pre-calculated weights from ‘data_dirweights' directory to be used for weight optimization.

Args:

- data_dir : string with the directory or full path to the directory in which application data is kept. If the setup() function is used to create the recommended project structure, the default None value should be used.
- num_assets : number of assets in the portfolio being optimized. An asset is a currency pair in a specific timeframe.

Returns:

- weight_combinations_list: list of all possible weight combinations for the passed number of assets.

Usage:

- used in ‘fxmanager.optimization.weight_optimizer’ module.

`fxmanager.basic.util.`**`ncr`**`(`*n*, *r*`)`

calculates the number of combinations nCr

Args:

- n: integer indicating the total number of items to be combined.
- r: integer indicating the length of every combination.

Returns:

- res: integer with the result.

`fxmanager.basic.util.`**`preprocess`**(*data_dir=None*, *app_type=''*, *raw_data_type=''*, *raw_data_format={}*, *save_logs=True*)

preprocesses the raw data files to prepare it for analysis and simumlations.

Args:

- data_dir : string with the directory or full path to the directory in which application data is kept. If the setup() function is used to create the recommended project structure, the default None value should be used.
- **app_type** [string indicating the application type. supported app types are]
  - back_tester : if you want to test trading strategies using histoical data on FX-Manager virtual account.
  - portfolio_optimizer : if you want to asses different portfolio optimization methods and objectives using histoical data.
  - live_simulator : if you want to test trading strategies in real time using MT4 Live data on FX-Manager virtual account.
  - all_in_one : if you want to use all the previous features.
- **raw_data_type** [string with the type of raw data used, supported raw data types are]
  - daily_bid_ask : Bid/Ask ticks in daily seperate files (a file for each currency_pair in each day)
  - daily_candles : 1-min candles in daily seperate files (a file for each currency_pair in each day)
  - period_candles : 1-min candles over a period (a file for each currency_pair for all the days)
- **raw_data_format** [dictionary with information about row data files, the required keys vary according to raw_data_type argument, the following are exambles of how the dictionary should be formatted for each type]
  - daily_bid_ask: {'date_col':'Gmt Time', 'date_format':'%d.%m.%Y %H:%M:%S.%f', 'ask_col':'Ask', 'bid_col':'Bid'}
  - daily_candles or period_candles: {'date_col':'time', 'date_format':'%Y.%m.%d %H:%M', 'open_col':'open', 'high_col':'high', 'low_col':'low', 'close_col':'close', 'tick_vol_col':'tick_volume'}
- save_logs : boolean flag, if True, the program logs are saved to 'data_dirlogspreprocessing_logs.txt' file.

Returns:

- None

`fxmanager.basic.util.`**`process_daily_bid_ask`**(*df*, *tf*)

helper function for 'fxmanager.basic.util.preprocess()' function.

used to group the given Bid/Ask data by the given timeframe and calculate open and close prices for the new candle sticks in the new timeframe.

`fxmanager.basic.util.`**`process_ohlc`**(*df*, *tf*)

helper function for 'fxmanager.basic.util.preprocess()' function.

used to group the given OHLCV data by the given timeframe and calculate open and close prices for the new candle sticks in the new timeframe.

`fxmanager.basic.util.`**`setup`**(*app_type=''*, *raw_data_dir=''*, *live_raw_data_dir=''*, *raw_data_type=''*, *file_name_format=''*, *num_days=1*, *num_cps=1*, *num_tfs=1*)

creates the required folders and files for any new project according to application type.

Args:

- **app_type** [string indicating the application type. supported app types are:]
  - back_tester : if you want to test trading strategies using histoical data on FX-Manager virtual account.
  - portfolio_optimizer : if you want to asses different portfolio optimization methods and objectives using histoical data.
  - live_simulator : if you want to test trading strategies in real time using MT4 Live data on FX-Manager virtual account.
  - all_in_one : if you want to use all the previous features.
- raw_data_dir : string containig the path to directory in which the raw data files are kept.
- live_raw_data_dir: string containig the path to directory in which the raw data files from the previous day are kept. used to construct a portfolio using built-in fxmanager optimization for live simulation.
- **raw_data_type** [string with the type of raw data used, supported raw data types are:]
  - daily_bid_ask : Bid/Ask ticks in daily seperate files (a file for each currency_pair in each day)
  - daily_candles : 1-min candles in daily seperate files (a file for each currency_pair in each day)
  - period_candles : 1-min candles over a period (a file for each currency_pair for all the days)
- **file_name_format** [string of white spaces with the same length as raw data file names. Givin that each file name has information about date and currency pair symbol, the white spaces in the same location as these information are replaced as follows:]
  - day, month and year in the date are replaced with (dd), (mm), (yyyy) respectively.
  - currency pair symbol is replaced with (cccccc). For example, if the file name is 'AUDUSD_Ticks_02.08.2021-02.08.2021.csv' then the file_name_format is: 'cccccc dd mm yyyy '
- num_days : total number of work days in the dataset.
- num_cps : integer indicating the number of currency pairs in the dataset
- num_tfs : integer indicating the number of timeframes in the dataset.

`fxmanager.basic.util.`**`top_n_timeframes`**(*best*, *wrst*, *objective='min_win_rate'*, *n=1*)

selects the best (n) of time frames according to the 'objective' argument.

Args:

- best : pandas dataframe with the return of the 'get_avg_rets()' function defined above. note tha the index must be transformed first to range index.
- wrst : pandas dataframe with the return of the 'get_avg_rets()' function defined above. note tha the index must be transformed first to range index.
- objective: string indecating the criteria with which the best time frames will be chosed, supported objectives are (min_win_rate, min_risk_reward)
- n : integer indicating the number of time frames to return

Returns:

- top_n: numpy array with the names of selected best time frames.

- W_R : pandas series with required win rates for every time frame.
- R_R : pandas series with required risk rewards for every time frame.

Usage:

- used in 'fxmanager.optimization.weight_optimizer' and 'fxmanager.optimization.eq_weight_optimizer' modules.

fxmanager.optimization package

Package contents

This sub-package contains the mudules used for portfolio construction and optimization.

Modules:

- eq_weight_optimizer : This module contains helper functions used internally in this module and other modules in fxmanager's optimization sub-package.
- eqw_optimization_test : This module contains a function that rans an equally weighted optimization test on historical data.
- weight_optimizer : This module contains a function that rans a weighted optimization test on historical data.
- w_optimization_test : This module contains helper functions used internally in this module and other modules in fxmanager's optimization sub-package.

fxmanager.optimization.eq_weight_optimizer module

This module contains helper functions used internally in this module and other modules in fxmanager's optimization sub-package.

the functions defined in this module are designed for constructing optimal equally weighted portfolios. a portfolio consists of one or more assets where an asset in this context refers to a currency pair being traded in a specefic time frame. for example, if the currency pair 'EURUSD' is being traded in two different time frames, 5 and 10 minutes, where a new position is opened every 5 or 10 minutes, in this case the 'EURUSD_5min' and 'EURUSD_10min' are two different assets.

Public Functions:

- **optimize(): runs equally weighted portfolio optimization using data from a single day given an optimization method a**

  ** this function is designed to be used internally in 'fxmanager.optimization.eqw_optimization_test.run()' function. However, it can be used by the user separately to construct a portfolio for a given day without running the optimization test.

`fxmanager.optimization.eq_weight_optimizer.`**`mixed_timframes_portfolio`**(*bestN*, *wrstN*, *objective='min_win_rate'*, *print_progress=False*)

Helper function for 'fxmanager.optimization.eq_weight_optimizer.optimize()' function.

selects the optimal currency pair from each timeframe in the given data, then selects the optimal equally weighted combination of these timeframes.

`fxmanager.optimization.eq_weight_optimizer.`**`optimize`**`(`*best_rets_df*, *wrst_rets_df*, *method='mixed_timeframes'*, *objective='min_win_rate'*, *test_run=True*`)`

runs equally weighted portfolio optimization using data from a single day given an optimization method and objective.

Args:

- best_rets_df: pandas dataframe of average best return for each asset, columns are time_frames and index is a range index.
- wrst_rets_df: pandas dataframe of average worst return for each asset, columns are time_frames and index is a range index.
- **method: string with optimization method to be used. The supported optimization methods are:**
  - mixed_timeframes: selects the optimal currency pair from each available timeframe, then selects the optimal equally weighted combination of these time frames.
  - single_timeframe: selects the optimal time frame from the available timeframes, then selects the optimal equally weighted combination of currency pairs in the selected time frame.
- **objective: string with optimization objective to be used. The supported optimization objectives are:**
  - min_win_rate : minimizes the required win rate for the portfolio to acheive a positive returns.
  - min_risk_reward: minimizes the required risk reward ratio required for the portfolio to acheive a positive returns.
- test_run: boolean flag used for testing purposes. If true, the constructed portfolios will consist of only one asset.

Returns:

- optimized_portfolio_idxs : tuple with length equal to 'optimized_n' and values of integers that repesent the indices corresponding to the optimal currency pairs.
- optimized_portfolio_tfs : numpy array with length of 'optimized_n' and values of optimal timeframes.
- optimized_portfolio_weights: numpy array with length equal to 'optimized_n' and values of equal floats that sum up to 1.
- optimized_n : integer indicating the number of assets in the constructed portfolio.
- optimized_W_R : float indicating the minimum required win rate for the portfolio to acheive positive return.
- optimized_R_R : float indicating the minimum required risk reward for the portfolio to acheive positive return.

** this function is designed to be used internally in 'fxmanager.optimization.eqw_optimization_test.run()' function. However, it can be used by the user separately to construct a portfolio for a given day without running the optimization test.

`fxmanager.optimization.eq_weight_optimizer.`**`single_timeframe_portfolio`**`(`*best_tf*, *wrst_tf*, *num_assets*, *objective='min_win_rate'*, *print_progress=False*`)`

Helper function for 'fxmanager.optimization.eq_weight_optimizer.optimize()' function.

selects the optimal equally weighted combination of currency pairs in the given timeframe data.

fxmanager.optimization.eqw_optimization_test module

This module contains a function that rans an equally weighted optimization test on historical data.

Functions:

- run_test(): Runs an optimization test on historical data given an 'optimization_method' and 'optimization_objective'.

`fxmanager.optimization.eqw_optimization_test.`**`run_test`**(*data_dir=None*, *optimization_method=''*, *optimization_objective=''*, *scenarios_per_day=100*, *risk_rewards=[]*, *leverage=0.01*, *test_run=False*, *save_plots=False*, *save_logs=False*)

Runs an optimization test given an 'optimization_method' and 'optimization_objective'.

The test uses preprocessed historical data to construct a portfolio using data from day (n) and then we use this portfolio to calculate average expected returns in day (n+1). The expected returns are calculated for every single element in 'risk_rewards' list and then averaged over 'scenarios_per_day' every day.

Args:

- data_dir : string with the directory or full path to the directory in which application data is kept. If the setup() function is used to create the recommended project structure, the default None value should be used.
- **optimization_method** [string with optimization method to be used. The supported optimization methods are:]
  - mixed_timeframes : selects the optimal currency pair from each available timeframe, then selects the optimal equally weighted combination of these time frames.
  - single_timeframe : selects the optimal time frame from the available timeframes, then selects the optimal equally weighted combination of currency pairs in the selected time frame.
- **optimization_objective: string with optimization objective to be used. The supported optimization objectives are**
  - min_win_rate : minimizes the required win rate for the portfolio to acheive a positive returns.
  - min_risk_reward : minimizes the required risk reward ratio required for the portfolio to acheive a positive returns.
- scenarios_per_day : integer indicating the number of scenarios over which the expected return is averaged every day.
- risk_rewards : list of floats. the average expected return is calculated for each value in this list every day.
- leverage : float indicating the leverage. if no leverage is used set to 1.
- test_run : boolean flag used for testing purposes. If true, the constructed portfolios will consist of only one asset.
- save_plots : boolean flag. If True, the program will save result plots to 'data_dirvisualizations' directory.
- save_logs : boolean flag. If True, the program will save result plots to 'data_dirlogsequally_weighted_optimizer_logs.txt' file.

Returns:

- None

Saved Results: test results are saved to 'data_dirvisualizations' for plots and 'data_dirstats' for csv files.

- Optimizer_Test_Win_Probs_&_Expexted_Rets_vs_Offsets.png : image with 2 sub-plots sharing the X-axis for the offsets, and Y axis for win probabilites and average expected returns. Offsets are linearly separated floats added to the 'min_win_rate' argument, starting from (0.01, 0.02, . . .), the value of 'min_win_rate' is incremented until the win probabilty becomes greater than or equal to 1. the plot might also contain several lines per plot, each line represents calculations a single risk-reward ratio.
- Optimizer_Test_Required_Win_Rates.png: image with a plot of constructed portfolios win rates on the Y-axis vs days on the X-axis.
- Optimizer_Test_Risk _Reward_Ratios.png: image with a plot of constructed portfolios risk rewards on the Y-axis vs days on the X-axis.
- Optimizer_Test_portfolios.csv : csv file with the constructed portfolios for every day. the index starts from 1 because every row represents the portfolio created using data from the day before so day 0 does not count.
- Optimizer_Test_win_probs.csv : csv file with the data frame used for plotting the first sub-plot in 'Optimizer_Test_Win_Probs_&_Expexted_Rets_vs_Offsets.png'
- Optimizer_Test_expexted_returns.csv : csv file with the data frame used for plotting the second sub-plot in 'Optimizer_Test_Win_Probs_&_Expexted_Rets_vs_Offsets.png'

fxmanager.optimization.w_optimization_test module

This module contains a function that rans a weighted optimization test on historical data.

Functions:

- run_test(): Runs an optimization test on historical data given an 'optimization_method' and 'optimization_objective'.

`fxmanager.optimization.w_optimization_test.`**`run_test`**`(`*data_dir=None*, *optimization_method=''*, *optimization_objective=''*, *scenarios_per_day=100*, *risk_rewards=[]*, *leverage=0.01*, *test_run=False*, *save_plots=False*, *save_logs=False*`)`

Runs an optimization test given an 'optimization_method' and 'optimization_objective'.

The test uses preprocessed historical data to construct a portfolio using data from day (n) and then we use this portfolio to calculate average expected returns in day (n+1). The expected returns are calculated for every single element in 'risk_rewards' list and then averaged over 'scenarios_per_day' every day.

Args:

- data_dir : string with the directory or full path to the directory in which application data is kept. If the setup() function is used to create the recommended project structure, the default None value should be used.
- **optimization_method** [string with optimization method to be used. The supported optimization methods are:]
  - mixed_timeframes : selects the optimal currency pair from each available timeframe, then selects the optimal combination of these time frames and the optimal weight for each time frame.
  - single_timeframe : selects the optimal time frame from the available timeframes, then selects the optimal combination of currency pairs in the selected time frame and the optimal weight for each currency pair.
- **optimization_objective: string with optimization objective to be used. The supported optimization objectives are**

  - min_win_rate : minimizes the required win rate for the portfolio to acheive a positive returns.
  - min_risk_reward : minimizes the required risk reward ratio required for the portfolio to acheive a positive returns.
- scenarios_per_day : integer indicating the number of scenarios over which the expected return is averaged every day.
- risk_rewards : list of integers. the average expected return is calculated for every integer in this list every day.
- leverage : float indicating the leverage. if no leverage is used set to 1.
- test_run : boolean flag used for testing purposes. If true, the constructed portfolios will consist of only one asset.
- save_plots : boolean flag. If True, the program will save result plots to 'data_dirvisualizations' directory.
- save_logs : boolean flag. If True, the program will save result plots to 'data_dirlogsequally_weighted_optimizer_logs.txt' file.

Returns:

- None

Saved Results: test results are saved to 'data_dirvisualizations' for plots and 'data_dirstats' for csv files.

- Optimizer_Test_Win_Probs_&_Expexted_Rets_vs_Offsets.png : image with 2 sub-plots sharing the X-axis for the offsets, and Y axis for win probabilites and average expected returns. Offsets are linearly separated floats added to the 'min_win_rate' argument, starting from (0.01, 0.02, …), the value of 'min_win_rate' is incremented until the win probabilty becomes greater than or equal to 1. The plot might also contain several lines per plot, each line represents calculations a single risk-reward ratio.
- Optimizer_Test_Required_Win_Rates.png : image with a plot of constructed portfolios win rates on the Y-axis vs days on the X-axis.
- Optimizer_Test_Risk _Reward_Ratios.png : image with a plot of constructed portfolios risk rewards on the Y-axis vs days on the X-axis.
- Optimizer_Test_portfolios.csv : csv file with the constructed portfolios for every day. the index starts from 1 because every row represents the portfolio created using data from the day before and tested on the current day.
- Optimizer_Test_win_probs.csv : csv file with the data frame used for plotting the first sub-plot in 'Optimizer_Test_Win_Probs_&_Expexted_Rets_vs_Offsets.png'
- Optimizer_Test_expexted_returns.csv : csv file with the data frame used for plotting the second sub-plot in 'Optimizer_Test_Win_Probs_&_Expexted_Rets_vs_Offsets.png'

fxmanager.optimization.weight_optimizer module

This module contains helper functions used internally in this module and other modules in fxmanager's optimization sub-package.

the functions defined in this module are designed for constructing optimal weighted portfolios. a portfolio consists of one or more assets where an asset in this context refers to a currency pair being traded in a specefic time frame. for example, if the currency pair 'EURUSD' is being traded in two different time frames, 5 and 10 minutes, where a new position is opened every 5 or 10 minutes, in this case the 'EURUSD_5min' and 'EURUSD_10min' are two different assets.

Public Functions:

- **optimize(): runs weighted portfolio optimization using data from a single day given an optimization method and obje**
  ** this function is designed to be used internally in 'fxmanager.optimization.eqw_optimization_test.run()' function. However, it can be used by the user separately to construct a portfolio for a given day without running the optimization test.

`fxmanager.optimization.weight_optimizer.`**`mixed_timeframes_portfolio`**(*bestN*, *wrstN*, *data_dir*, *objective='min_win_rate'*, *print_progress=False*)

Helper function for 'fxmanager.optimization.weight_optimizer.optimize()' function.

selects the optimal currency pair from each timeframe in the given data, then selects the optimal weighted combination of these timeframes.

`fxmanager.optimization.weight_optimizer.`**`optimize`**(*best_rets_df*, *wrst_rets_df*, *data_dir=None*, *method='mixed_timeframes'*, *objective='min_win_rate'*, *test_run=True*)

runs weighted portfolio optimization using data from a single day given an optimization method and objective.

Args:

- best_rets_df: pandas dataframe of average best return for each asset, columns are time_frames and index is a range index.
- wrst_rets_df: pandas dataframe of average worst return for each asset, columns are time_frames and index is a range index.
- **data_dir** [string with the directory name or full path to the directory in which application data is kept. If the setup()] function is used to create the recommended project structure, the default None value should be used.
- **method: string with optimization method to be used. The supported optimization methods are:**
  - **mixed_timeframes: selects the optimal currency pair from each available timeframe, then selects the optir**
    combination of these time frames.
  - **single_timeframe: selects the optimal time frame from the available timeframes, then selects the optimal e**
    combination of currency pairs in the selected time frame.
- **objective: string with optimization objective to be used. The supported optimization objectives are:**
  - min_win_rate : minimizes the required win rate for the portfolio to acheive a positive returns.
  - min_risk_reward: minimizes the required risk reward ratio required for the portfolio to acheive a positive returns.
- test_run: boolean flag used for testing purposes. If true, the constructed portfolios will consist of only one asset.

Returns:

- **optimized_portfolio_idxs** [tuple with length equal to 'optimized_n' and values of integers that repesent the indices corresponding] to the optimal currency pairs.
- optimized_portfolio_tfs : numpy array with length of 'optimized_n' and values of optimal timeframes.
- optimized_portfolio_weights: numpy array with length equal to 'optimized_n' and values of equal floats that sum up to 1.
- optimized_n : integer indicating the number of assets in the constructed portfolio.
- optimized_W_R : float indicating the minimum required win rate for the portfolio to acheive positive return.

- optimized_R_R : float indicating the minimum required risk reward for the portfolio to acheive positive return.

**** this function is designed to be used internally in 'fxmanager.optimization.eqw_optimization_test.run()' function. Howe**
it can be used by the user separately to construct a portfolio for a given day without running the optimization test.

fxmanager.optimization.weight_optimizer.**single_timeframe_portfolio**(*best_tf*, *wrst_tf*, *data_dir*, *num_assets*, *objective='min_win_rate'*, *print_progress=False*)

Helper function for 'fxmanager.optimization.eq_weight_optimizer.optimize()' function.

selects the optimal weighted combination of currency pairs in the given timeframe data.

fxmanager.simulation package

Package contents

This sub-package contains the modules used for historical and live trading sumulations.

Modules:

- historic: This module contains helper functions used internally in this module and a public function to run historical simulation.
- live : This module contains helper functions used internally in this module and a public function to run live simulation with live price feed from MT4.

fxmanager.simulation.historic module

This module contains helper functions used internally in this module and a public function to run historical simulation.

Public Functions:

- run(): starts trading simulation with historic prices.

fxmanager.simulation.historic.**close_position**(*account*, *ticket*, *portfolio_prices*, *portfolio_orders*, *wins*, *losses*)

helper function to 'fxmanager.simulation.historic.run()' function. closes an opened position by ticket.

Returns:

- portfolio_orders: pandas dataframe with history of closed positions. Each row represents a closed position.
- wins : integer indicating number of closed positions with positive profit.
- losses : integer indicating number of closed positions with nigative profit.
- win_rate : float indicating percentage of wins to (wins+losses).

fxmanager.simulation.historic.**get_prices**(*day*, *data_dir=None*, *time_frame=''*, *currency_pair=''*)

helper function to 'fxmanager.simulation.historic.run()' function. gets the bid/ask prices of the given currency_pair & time_frame in the selected day from the 'data_dirtime_frames' directory.

Args:

- day : integer indicating day number - starts from 0 and ends with the total number of work days in the dataset (num_days-1).

- data_dir : string with the directory or full path to the directory in which application data is kept. If the setup() function is used to create the recommended project structure, the default None value should be used.
- time_frame : a string of the timeframe from wich the prices is read.
- currency_pair : a string of the currency pair from wich the prices is read.

Returns:

- df: pandas dataframe of the bid & ask prices with columns (ask_open, bid_open, ask_close, bid_close, tick_volume) and a datetime index from the begining to the end off the day with frequency equal to time_frame.

`fxmanager.simulation.historic.`**`print_final_state`**`(`*account*, *win_rate*`)`
helper function to 'fxmanager.simulation.historic.run()' function. prints the final state of the portfolio after the simulation is finished.

`fxmanager.simulation.historic.`**`run`**`(`*account*, *strategy*, *data_dir=None*, *portfolios={}*, *risk_factor=0.95*, *dynamic_sltp=False*, *save_logs=False*, ***kwargs*`)`
starts trading simulation with historic prices.

Args:

- account : account object with all account information.
- strategy : strategy object with the trading strategy information.
- data_dir : string with the directory or full path to the directory in which application data is kept. If the setup() function is used to create the recommended project structure, the default None value should be used.
- portfolios : pandas dataframe with columns (currency_pairs, time_frames, weights) and range index of length = num_days
- risk_factor : a float with range from 0 to 1 indicating the percentage of reinvested balance.
- dynamic_stlp : boolean flag, if True, stop losses and take profits of opened positions are updated with each simulation step.
- save_logs : boolean flag, if the program logs are saved to 'data_dirlogslive_simulation_logs.txt' file.
- kwargs : dictionary to hold any number of arguments required for the strategy object.

Returns:

- None

fxmanager.simulation.live module

This module contains helper functions used internally in this module and a public function to run live simulation with live price feed from MT4.

Public Functions:

- run(): starts trading simulation with live prices from MT4 EA.

`fxmanager.simulation.live.`**`close_position`**`(`*account*, *ticket*, *portfolio_prices*, *portfolio_orders*, *wins*, *losses*`)`
helper function to 'fxmanager.simulation.live.run()' function. closes an opened position by ticket.

Returns:

- portfolio_orders: pandas dataframe with history of closed positions. Each row represents a closed position.

- wins : integer indicating number of closed positions with positive profit.
- losses : integer indicating number of closed positions with nigative profit.
- win_rate : float indicating percentage of wins to (wins+losses).

`fxmanager.simulation.live.`**`get_price`**`(`*currency_pairs*, *price_feed*, *sleep_time*`)`

helper function to 'fxmanager.simulation.live.run()' function. gets the current prices of portfolio assets from price feed.

Returns:

- price_every_iter_df: pandas dataframe with length = 1 and (4* number of currency pairs) columns. Each asset has 4 columns as follows, (000000_bid_open, 000000_ask_open, 000000_bid_close, 000000_ask_close) where 000000 is replaced with currency pair symbol.

`fxmanager.simulation.live.`**`optimize_portfolio`**`(`*data_dir*, *raw_data_format*, *optimization_method*, *optimization_objective*, *weight_optimization=False*, *is_preprocessed=True*`)`

helper function to 'fxmanager.simulation.live.run()' function. Does portfolio optimization using data from previous day. The constructed portfolio is used for live simulation.

Returns:

- portfolio : dictionary with keys ('currency_pairs', 'time_frames', 'weights', 'optimized_n', 'optimized_W_R', 'optimized_R_R')
- currency_pairs: list with portfolio currency pairs.
- time_frames : list with portfolio timeframes.
- weights : list with portfolio weights.

`fxmanager.simulation.live.`**`print_final_state`**`(`*account*, *win_rate*`)`

helper function to 'fxmanager.simulation.live.run()' function. prints the final state of the portfolio after the simulation is finished.

`fxmanager.simulation.live.`**`print_live_state`**`(`*account*, *win_rate*`)`

helper function to 'fxmanager.simulation.live.run()' function. prints the current state of the portfolio.

`fxmanager.simulation.live.`**`run`**`(`*account*, *strategy*, *data_dir=None*, *construct_portfolio=False*, *portfolio={}*, *weight_optimization=False*, *is_preprocessed=True*, *raw_data_format=''*, *optimization_method=''*, *optimization_objective=''*, *sleep_time=1*, *risk_factor=1*, *dynamic_sltp=False*, *sync_zero=False*, *save_logs=False*, ***kwargs*`)`

starts trading simulation with live prices from MT4 EA.

Args:

- account : account object with all account information.
- strategy : strategy object with the trading strategy information.
- data_dir : string with the directory or full path to the directory in which application data is kept. If the setup() function is used to create the recommended project structure, the default None value should be used.
- construct_portfolio : boolean indicating whether portfolio optimization is used or not.
- portfolio : dictionary with the portfolio to be used for trading, keys are: (currency_pairs, time_frames, weights). This argument is only used if 'construct_portfolio' is set to False.
- weight_optimization : boolean indicating whether weight optimization is used in portfolio optimization. This argument is only used when 'construct_portfolio' is set to True.

- is_preprocessed : boolean indicating if the data of used for portfolio optimization is preprocessed or not. This argument is only used when 'construct_portfolio' is set to True.
- raw_data_format : dictionary with information about raw data used for portfolio optimization. This argument is only used when 'construct_portfolio' is set to True.
- optimization_method : string with optimization method to be used. This argument is only used when 'construct_portfolio' is set to True.
- optimization_objective: string with optimization objective to be used. This argument is only used when 'construct_portfolio' is set to True.
- sleep_time : integer indicating the time every which the portfolio is updated (in minutes).
- risk_factor : a float with range from 0 to 1 indicating the percentage of reinvested balance.
- dynamic_stlp : boolean flag, if True, stop losses and take profits of opened positions are updated with each simulation step.
- sync_zero : boolean flag, if True, the simulation starts when the seconds in current time = 0.
- save_logs : boolean flag, if the program logs are saved to 'data_dirlogslive_simulation_logs.txt' file.
- kwargs : dictionary to hold any number of arguments required for the strategy object.

Returns:

- None

fxmanager.strategies package

Package contents

This sub-package contains the mudules used for building and injecting trading strategies int FX-Manager's trading simulations.

Public Modules:

- naieve_momentum: This module contains simple implementation of momentum trading strategy, used as default for simulations.

fxmanager.strategies.naieve_momentum module

This module contains simple implementation of momentum trading strategy.

Public Functions:

- get_orders(): sample trading strategy for purposes of testing live and historic trading simulators.

`fxmanager.strategies.naieve_momentum.`**`get_orders`**`(`*prices*, ***kwargs*`)`

sample trading strategy for purposes of testing live and historic trading simulators.

Args:

- **prices: pandas DataFrame with range index and a the following columns:**
  - ask_open: open ask price of the candle stick.
  - bid_open: open bid price of the candle stick.
  - ask_close: close ask price of the candle stick.
  - bid_close: close bid price of the candle stick.

- kwargs: special dictionary to hold any number of required arguments for the function.

Returns:

- **orders: pandas DataFrame with ONE RAW and the following columns:**
  - order_type: can be one of 3 values (buy, sell, hold)
  - SL : stop loss.
  - TP : take profit.

fxmanager.strategies.template module

This module contains a class that is used by user to create a strategy object which is passed to fxmanager's built-in live and historical simulators.

Public Classes:

- strategy_template: Class for injecting user defined trading strategies into fxmanager's built-in live and historical simulators.

`class fxmanager.strategies.template.`**`strategy_template`**`(`*strategy*, *take_all_prices=False*`)`

Bases: `object`

Class for injecting user defined trading strategies into fxmanager's built-in live and historical simulators.

Args:

- **strategy** [reference to a user defined function for some trading strategy. The function MUST be structured as follows in order for the simulation to work proberly:]
  - **takes ONLY ONE positional argument called 'prices'. This argument is a pandas DataFrame with range i**
    - ask_open: open ask price of the candle stick.
    - bid_open: open bid price of the candle stick.
    - ask_close: close ask price of the candle stick.
    - bid_close: close bid price of the candle stick.
  - takes **kwargs for any other keword arguments.
  - **returns a pandas DataFrame with ONE RAW and the following columns:**
    - order_type: can be one of 3 values (buy, sell, hold)
    - SL : stop loss.
    - TP : take profit.
- **take_all_prices: boolean that controls the frequency of the range index of 'prices' DataFrame.** If True: all the prices history since the begining of the day is passed to the strategy function, the index is a normal range index (0, 1, 2, …). the time difference between every reading is determined by the 'sleep_time' argument in the 'fxmanager.simulation.live.run()' or 'fxmanager.simulation.historic.run()' functions. If False: the frequncy of the prices becomes dependant on the time frame in which we want to generate an order, for examble, if we want to open a position in 'EURUSD' in the '5min' time frame, the index will become (0, 5, 10, ..).

Methods:

- get_orders: used in 'fxmanager.simulation.live.run()' and 'fxmanager.simulation.historic.run()' functions.

`get_orders`(*prices*, ***kwargs*)

`get_portfolios_template`()
returns a list of dictionaries of length = num_days where each dictionary is a portfolio used for a given day.

`kwargs_template`()
returns a template dictionary for keyword args used in used defined strategies.

`template`(***kwargs*)
function template for the user to build trading strategies.

Args:

- **prices: pandas DataFrame with range index and the following columns:**
  - ask_open: open ask price of the candle stick.
  - bid_open: open bid price of the candle stick.
  - ask_close: close ask price of the candle stick.
  - bid_close: close bid price of the candle stick.
- kwargs: special dictionary to hold any number of required arguments for the function.

Returns:

- **orders: pandas DataFrame with ONE RAW and the following columns:**
  - order_type: can be one of 3 values (buy, sell, hold)
  - SL : stop loss.
  - TP : take profit.

fxmanager.dwx package

Package contents

This sub-package contains a modified version of Darwinex ZeroMQ connector which is used for connecting client code to MT4 trading platform.

Public Modules:

- prices_subscription : This is a modified version of an example of using the Darwinex ZeroMQ Connector for Python 3 and MT4 PULL REQUEST.
- rates_historic : This Module is used as an API to get historic data from the MT4 EA.

fxmanager.dwx.prices_subscriptions module

This is a modified version of an example of using the Darwinex ZeroMQ Connector for Python 3 and MetaTrader 4 PULL REQUEST.

The original example is referenced here, This Module is used as an API to get real time data from the MT4 EA. The user creates a 'prices_subscriptions' object with the desired symbols list in the client applicaction, then the real-time price feed can be accessed from the self._recent_prices dictionary.

Through commmand TRACK_PRICES, this client can select multiple SYMBOLS for price tracking. For example, to receive real-time bid-ask prices from symbols EURUSD and GDAXI, this client will send this command to the Server, through its PUSH channel:

"TRACK_PRICES;EURUSD;GDAXI"

Server will answer through the PULL channel with a json response like this:

{'_action':'TRACK_PRICES', '_data': {'symbol_count':2}}

or if errors, then:

{'_action':'TRACK_PRICES', '_data': {'_response':'NOT_AVAILABLE'}}

Once subscribed to this feed, it will receive through the SUB channel, prices in this format:

"EURUSD BID;ASK"

–

Original Author: raulMrello

Modified by : AbdullahBahi

`class fxmanager.dwx.prices_subscriptions.`**`prices_subscriptions`**`(`*_name='PRICES_SUBSCRIPTIONS', _symbols=['EURUSD', 'GDAXI'], _delay=0.1, _broker_gmt=3, _verbose=False*`)`

Bases: `fxmanager.dwx.DWX_ZMQ_Strategy.DWX_ZMQ_Strategy`

`isFinished``()`
Check if execution finished

`onPullData``(`*data*`)`
Callback to process new data received through the PULL port

`onSubData``(`*data*`)`
Callback to process new data received through the SUB port

`run``()`
Starts price subscriptions

`stop``()`
unsubscribe from all market symbols and exits

fxmanager.dwx.rates_historic module

This Module is used as an API to get historic data from the MT4 EA.

This is a modified version of an example of using the Darwinex ZeroMQ Connector for Python 3 and MetaTrader 4 PULL REQUEST for v2.0.1 in which a Client requests rate history from a Daily from a start date to an end date. The user creates a 'rates_historic' object and using the methods 'get_daily_candles' and 'get_period_candles' the user client can get the historic data of the desired symbols.

The original example is referenced here.

Through commmand HIST, this client can select multiple rates from an INSTRUMENT (symbol, timeframe). For example, to receive rates from instruments EURUSD(M1), between two dates, it will send this command to the Server, through its PUSH channel:

"HIST;EURUSD;1;2019.01.04 00:00:00;2019.01.14 00:00:00"

Original Author: raulMrello

Modified by : AbdullahBahi

class `fxmanager.dwx.rates_historic.`**`rates_historic`**(*_name='PRICES_SUBSCRIPTIONS'*, *_delay=0.1*, *_broker_gmt=3*, *_verbose=False*)

Bases: `fxmanager.dwx.DWX_ZMQ_Strategy.DWX_ZMQ_Strategy`

`get_daily_candles`(*save_to=''*, *currency_pair=''*, *date=''*)

Request historic data and returns data of type 'daily_candles'

`get_period_candles`(*save_to=''*, *currency_pair=''*, *start=''*, *end=''*)

Request historic data and returns data of type 'period_candles'

`isFinished`()

Check if execution finished

`onPullData`(*data*)

Callback to process new data received through the PULL port

`onSubData`(*data*)

Callback to process new data received through the SUB port

`stop`()

unsubscribe from all market symbols and exits

CHAPTER FOUR

TUTORIALS

1. Getting started
2. Data collection
3. Setting up new projects
4. Data pre-processing
5. Running portfolio optimization test on historical data
6. Running historical trading simulation
7. Running real-time trading simulation

CHAPTER

FIVE

LICENSE

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

CHAPTER

SIX

ABOUT AUTHOR

This project is built and maintained by Abdullah Bahi, Junior data analyst with a robust background in machine and deep learning. Also, a professional python developer with wide knowledge and hands-on experience with most of the packages used in AI and Data Science.

Expertise Areas

- Data Analysis
- Data Visualization
- ETL
- Data Base (SQL)
- Machine Learning
- Deep Learning
- lgorithmic Trading
- Open Source Software Development (OSS)
- Business Analysis

- Financial Data Analysis

Work Experience

For a full list of my previous work visit my central repository on GitHub from here.

General Information

- **Name :** Abdullah Bahi
- **Title :** Junior Data Analyst
- **Education :** Bachelors degree in Electronics and Communication Engineering (ECE)
- **Nationality :** Egypt

Social Channels & Contact Info

- E-mail
- LinkedIn
- Twitter
- Youtube Channel

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX