Markowitz_PyPortfolioOpt_GUIDE

slide 1

Title: FixYahoo.py

We are going to present a suite of programs from the Python module PyPortfolioOpt,
coupled with a backtesting program and a data-downloading program.
These programs will be easier to use than the programs in the previous presentation
as they only require changing a few lines of code.

The data downloading program is in a file called FixYahoo.py.
FixYahoo was the original name of the y-finance module that is used in this program.
You can find documentation of y-finance in the link on the slide.
We mention this because y-finance has been updated since we first used it in our programs,
so it is now possible to do things with it that were not possible before,
like downloading many tickers at once without a loop.
In any case, the changes to y-finance have been backward compatible,
so the code in FixYahoo.py still runs.

slide 2

Title: FixYahoo.py

Lines 23 and 24:
Enter the start date and end date.
Line 50:
Enter the list of tickers you want to download.
Lines 69 and 83:
Enter the path to the directory where the CSV file will be used.
There are two types of prices, downloaded separately:
Close prices and adjusted close prices.
Adjusted close prices are re-calculated every day so as to include
the impact of splits, dividends, coupon inflows, etc..
Close prices are not re-calculated.
It makes sense to use close prices for the calculation of trading signals,
especially if trading decisions are daily or even weekly.
Adjusted close prices can be used to calculate the trading system's equity.
This is not a hard and fast rule.
Sometimes we will make this distinction, but not always, depending on
the complexity of the system.

slide 3

Title: cplex_MAIN.py (cplex.rar) 1

This slide presents a basic backtesting Python program
for a Markowitz portfolio optimization that uses
the IBM ILOG CPLEX OPTIMIZATION optimizer and the cvxpy wrapper.
IBM Cplex is a robust proprietary optimizer,
but its use is not intuitive like the solvers we seen so far,
so we are going to "wrap" IBM Cplex in another library called the cvxpy interface,
which is much easier to use and which can "wrap" a long list of Python optimizers.
Our program actually consists of two parts,
the main function in cplex_MAIN.py and
a helper module called cplex_MODULE.py
Our program finds the portfolio that minimizes the variance

while trying to meet the minimum required return desired.
So it does the same thing as 1.ClassicMarkowitz_FixedRequiredReturnMinimizedVariancePortfolio.py,
except that this new version optimizes the portfolio every month in a loop over a number of years,
and then plots the equity curve and the annual portfolio statistics.
The loop is in cplex_MAIN.py, whereas the optimization function is in cplex_MODULE.py.
There are some important aspects of the loop so we will discuss it first.

Lines 36 to 47:
Uses the yfinance utility to download a list of stock tickers,

line 41:
defines the start and end dates of the backtesting period.
This can be changed.

line 43:
Defines the stock list containing the tickers to download.
The stock tickers are the 28 Dow Jones stocks (optionally plus the BND (general bonds) ETF and the SHY (extremely short bonds) ETF)
This can be changed.

slide 4

Title: cplex_MAIN.py 2

lines 49 to 56
Define the parameters of the optimization.
min_return is the required annual return.
interest_rate is the fixed annual interest rate of the risk-free asset
which is added as an additional asset to the portfolio.
cash specifies the fraction of capital to keep as cash.
shift controls two variables: the holding period (for rebalancing) and
the period of the returns that will be calculated.
Here shift is set to 21, which means the portfolio will be optimized monthly
and the optimization calculations will be done with monthly returns.
The lookback is set to 60 meaning 60 returns will be used
to estimate the covariance matrix and the vector of expected returns
which will serve as input to the optimization.
All these parameters can be changed.

Just remember that the lookback has to be at least as long as the number of stocks,
otherwise, the covariance matrix will not be well specified for the optimization problem.

Line 62:
Calculates the returns according to the shift specified,
and uses the percentage formula (required by portfolio calculations).

slide 5

Title: SPY_volatility_lookback_monthly_overlapped.xlsx

We want to make a comment about choosing an appropriate lookback.
Open SPY_volatility_lookback_monthly_overlapped.xlsx.
It has SPY daily price data from 1993 to 2020.
We calculated the monthly percent returns of the open prices in column H.
Note that the monthly percent returns overlap each other because they are based on daily price data.
In column I, we calculated the standard deviation of these returns,

increasing the size of the sample as the row increases.
We graphed the resulting standard deviation in column I, with the title:
Standard deviation as a function of sampling size.
You can see the graph to the right of the spreadsheet.
In the spreadsheet, we did this experiment (and graphed it) twice,
once starting 1993 (top of file) and again starting 2015 (see row 106).
Note that many rows are hidden in the spreadsheet.
Looking at the plots, we thought that a reasonable lookback (sample size) would be 60.
We chose a lookback (sample size) of 60 because we wanted to choose a lookback
where the standard deviation reaches a stable steady state,
which we imagine occurs within the yellow band in the plots.
A very short lookback of less than 10 is not a good idea because the standard deviation
is whiplashing up.
But, you may differ as there is no hard and fast rule about this.

Mean variance optimization algorithms used to calculate the portfolio weights
depend on the covariance matrix of the returs not being singular.
Others (e.g. Critical Line Algorithm, Hierarchical Clustering) do not have this limitation
and are built to circumvent the Karush-Kuhn-Tucker conditions required for
finding the optimal point in a manifold.
The general rule to prevent the covariance matrix from becoming singular is
to require the amount of data to be at least 1/2*N*(N+1) independent and identically distributed observations,
where N is the number of assets.
This means that if we want the window to be quite short (e.g. 60 days) with the objective of
allowing the portfolio to benefit from the most relevant momentum information, then
the number of assets must be 11 or less; otherwise, the covariance matrix could become singular.
But we saw in EffectOfDiversification.pptx that we want to have between 15 to 30 assets.
One way around this problem is to keep the number of assets at 11 but use a mixture of ETFs and stocks or
to use only ETFs.

slide 6

Title: cplex_MAIN.py 3

Lines 66 to 68:
Do some pre-processing of the returns.
This is especially important when estimating future returns and the covariance matrix of the returns
based on historical data, which tends to be noisy.
Two options are presented here:
A smoothing of the returns by a simple moving average, to make them less choppy or
An exponential smoothing of the returns by the exponentially weighted moving average function.
You can try both of these and check the difference in the results, which is quite visible.
Exponential smoothing is explained in the next slide.

slide 7

Title: ExponentiallyWeightedMovingAverage.ipynb

Open the notebook in the title of this slide.
You will see function defined there is is very close to what Pandas exponential moving average function is doing.
The exponentially moving average function generates a set of weights that decay exponentially.
The graph of the weights is at the bottom of the slide.
The idea is to multiply the returns with these weights in such a way that
returns closer to the present are multiplied by larger weights and thus given more importance.
Pandas ewm function header is provided in the slide to show you that
although the result is always a set of weights similar to those in the graph,

it is possible to specify the speed of the decay using
span, halflife, or alpha, and the choice is a matter of preference.
You can read about these parameters in the link given on the slide.
in our portfolio optimization program, we have chosen
a span equal to the lookback,
but we could have chosen a halflife equal to half of the lookback.

slide 8

Title: cplex_MAIN.py 4

Line 85:
Calculates the covariance matrix based on the exponentially weighted mean average returns.
But you can uncomment and try other options in lines 71 to line 79.

Line 108:
Calculates the expected returns based on the exponentially weighted mean average returns.
But you can uncomment and try other options in lines 109 to line 110.

Lin 112:
Sends the covariance matrix, expected returns, and the min_return required to the optimizer
to calculate the weights of the stocks in the portfolio.
The weights are assigned to w.

slide 9

Title: cplex_MAIN.py 5

Line 124:
Calculates the cumulative return.

You will notice that a product is being calculated and then the results are being added together.
So this is doing the same thing as the Pandas cumprod function,
given at the bottom of the slide.
Remember there are two ways of calculating the cumulative returns, depending on the type of return that was used.
If you used percentage returns (which is what we used), then you need to multiply the returns along the time dimension.
If you used log returns, then you need to add the returns along the time dimension.
When you plot the cumulative returns you get a curve that reflects the history of the trading system's bank account.

slide 10

Title: cplex_MAIN.py 6

This slide shows the plot of the cumulative returns over the years, also known as the equity curve.

slide 11

Title: Annual Return

Next, we discuss some portfolio metrics that you may be interested in calculating.
The most intuitive return is the annual return, which is the total percent change between the ending balance and
the beginning balance divided linearly by the number of years.
Because interest is not compounded, and the growth rate is calculated in a linear fashion, this formula is less exact.

slide 12

Title: CAGR

Compound annual growth rate (CAGR) is the rate of return that would be required for an investment to grow
from its beginning balance to its ending balance,
assuming the profits were reinvested at the end of each year of the investment's lifespan.

slide 13

Title: Sharpe Ratio

The Sharpe ratio is the average return earned in excess of the risk-free rate per unit of volatility or total risk.
Again there is more than one way to calculate it.
You may use the formula, just remember that the excess return is another name for the difference in the numerator,
between the portfolio return and the risk-free return.
There is more than one way to calculate the Sharpe ratio depending on how you calculate this numerator.
In the simplified version appearing below the formula,
you can just use an average of the returns, and
divide it by the standard deviation of the returns,
and annualize the result.
Whichever way you decide to do this, what is important is that you stay consistent, so as to be able to compare
across investing methodologies.

slide 14

Title: Maximum drawdown or MDD

A maximum drawdown (MDD) is the maximum observed loss
from a peak to a trough of a portfolio before a new peak is attained.
A maximum drawdown is an indicator of downside risk over a specified time period.
We do not often calculate it in the demos but it is important for you to know about it.

slide 15

Title: Maximum drawdown program

This is a little program for calculating the maximum drawdown,
should you want to do so in the future.

slide 16

Title: Calmar Ratio

The Calmar ratio is a comparison of the average annual compounded rate of return and
the maximum drawdown risk.
The lower the Calmar ratio,
the worse the investment performed on a risk-adjusted basis over the specified time period;
the higher the Calmar ratio,
the better it performed.
Again, we do not often calculate it in the demos but it is important for you to know about it.

slide 17

Title: cplex_MAIN.py 7

Lines 182 to 202:
These lines calculate and print out some of the metrics we have just discussed
for our program.
The results are acceptable:
A Sharpe ratio above 1 is considered very good.
But notice that the volatility is quite high, almost 13%.
This is, of course, a problem because you want to be able to place a roof on that volatility.
So, soon we will be looking at another program that will allow us to do this.
The point of this presentation though is to show you how a backtesting program works
and how the new optimization interface, cvxpy, works with IBM's cplex optimizer,
which is exactly where we are going next.

slide 18

Title: cplex_MODULE.py

So open the Python script with the name in the title of this slide.
This script shows the optimization function that calculates the weights of the portfolio every month.

Lines 2 to 4:
These lines import the cvxpy module and the posdef module.
The cvxpy module is an interface that allows the user to program
an optimizer using a standard format that works with all optimizers.

Line 12:
Uses nearestPD to convert the covariance matrix to the nearest positive-definite matrix so that
it can be easily inverted by the optimizer.
Actually using nearestPD is strictly speaking not necessary since IBM's cplex is a very robust optimizer,
but we would have to use it if we changed the optimizer from cplex to some other, less robust, optimizer.

Line 18:
This line uses cvxpy's Variable function to define
the weights of the portfolio that we are solving for,
corresponding to the changing cells in Excel's Solver.

Line 20:
Uses cvxpy's quad_form function to define the risk to be minimized.
Actually, using quad_form is not required, it is a convenience "atomic" function,
which you can find here: https://www.cvxpy.org/tutorial/functions/index.html
quad_form outputs the portfolio variance (the risk) w.T@Sigma@w that will be minimized (@ indicates matrix multiplication)

Line 21 to 22:
Uses cvxpy's Parameter function to define the value of the min_ret required return parameter.

Line 23 to 26:
Uses cvxpy's Problem function to set up the optimization model, and assigns the model to prob.

Line 23:
Uses cvxpy's Minimize function to set up the objective function to minimize the variance (the risk).

Line 24:
Sets up the condition that the return be greater than or equal than the required return.

Line 25:
Uses cvxpy's sum function to set up the condition that the sum of the weights be equal to 1.

Line 26:
Sets up the condition that the weights be positive

Line 28:
Uses the optimization model's solve function to solve the optimization problem.
Note that in this line we define the solver cvxpy is using under wraps as IBM's cyplex.

As you can notice, even though IBM's cyplex is VERY complicated to use
the cvxpy interface allows us to program with cyplex in an intuitive way
similar to the way one programs with Excel's Solver.
Next, we will be showing you how to use the pyPortfolioOpt module to program your portfolios.
pyPortfolioOpt is even simpler than cvxpy because you need only change a couple of lines of code!
But pyPortfolioOpt uses cvxpy and cplex under wraps so you need to know how it is working
in case you need to look at the source code of pyPortfolioOpt to clarify questions you may have or
to modify the code for your own needs.

slide 19

Title: pyopt_MAIN.py (in PyPortfolioOpt.rar)

We now present the use of the pyPortfolioOpt module.
This program actually consists of two parts,
the main function in pyopt_MAIN.py and
a helper module called pyopt_MODULE.py

pyopt_MAIN.py is practically the same main program we just presented (cplex_MAIN.py).
The program uses the IBM ILOG CPLEX OPTIMIZATION optimizer and the cvxpy interface.

Line 112:
This line is different because it calls
the correct get_weights function in the correct module.

slide 20

Title: pyopt_MODULE.py

This slide presents how you can call different portfolio optimization functions of the PyPortfolioOpt module
by choosing the correct case number in the switch

Case 1:
Calls the max_sharpe function of the optimizer.
It finds the weights of the capital market portfolio that maximizes the Sharpe ratio.

Case 2:
Calls the efficient_risk function of the optimizer (with the maximum volatility as input).
The optimizer solves Harry's problem (that is, flipped Markowitz).
It finds the weights of the capital market portfolio that maximizes the Sharpe ratio while
simultaneously placing an upper limit on the volatility.
We have a free-standing version of this program which unfortunately takes hours to run:
HarrysProblem.SPY.TLT.YFINANCE.rar
Our version is closer to the solution described in the paper, as
it minimizes the volatility while simultaneously placing a lower limit on the required return.

Case 3:
Calls the efficient_return function of the optimizer (with the required return as input).
It finds the weights of a portfolio that minimizes the volatility while

simultaneously placing a lower limit on the required return.

Case 4:
Calls the min_volatility function of the optimizer.
It finds the weights of a portfolio that minimizes the volatility with no conditions.
This is the minimum variance portfolio.

Case 5:
Calls the critical line algorithm model to find the capital market portfolio that maximizes the Sharpe ratio.
However, it tends to go into an infinite loop.
We have a free-standing version of this program that we have corrected:
markowitz-portfolio-optimization-master.CLA_LOOP.YFINANCE.PY37.rar

Note that it may not be possible for the optimizer to obey a condition because
the market simply does not permit it.
For example,
if the minimum variance portfolio has a volatility that is above 12%,
a condition that puts a cap of 10% on the volatility of the case 2 optimizer will not be able to be obeyed except rarely.
Similarly,
if the maximum expected return of all the stocks is 5%
and you set a required return condition of 7%, the case 3 optimizer will not be able to obey that condition.
Read the code carefully in cases 2 and cases 3 to see why this happens.

slide 21

Title: PyPortfolioOpt Documentation
We strongly recommend you read the entire documentation of PyPortfolioOpt.
It is extremely well written and informative.
Download the code from GitHub as a zip file from the GitHub link provided.
It is essential for a deeper understanding and thankfully, well documented, and well organized.

slide 22

Title: Title: PyPortfolioOpt Cookbook
PyPortfolioOpt has a collection of jupyter notebooks that serve as cookbook.
These are very useful too.

slide 23

Title: Returns.xlsx
This slide shows all relevant formulas with both types of returns,
percent and log returns, in Returns.xlsx.
We suggest you use log returns as much as possible since
their formulas for annualized return and Sharpe ratio are simpler.
When calculating CAGR (using log returns)
use the formula with beginning and ending balance.
When dealing with stock portfolios you must use percent returns for the individual assets
because otherwise the calculation of the portfolio returns will not work out correctly.
This requirement is sometimes ignored but doing so is a mistake.
But once you have the portfolio returns on hand you can convert them to log returns
by applying the conversion formula in row 29
and then use the simplest formulas for
the calculation of portfolio annualized return and portfolio Sharpe ratio.
This conversion step is often skipped because differences are small if you do not convert
but it is the strictly correct thing to do.
Please study this spreadsheet carefully.
In the worksheet called Formulas,

you will find the percent returns (big cap R) and log returns (small cap r) in two columns,
This worksheet carefully calculates the log and percent return version of:
cumulative returns,
average returns,
annualized returns,
variance of the returns,
annualized variance of the returns,
standard deviation of the returns,
annualized standard deviation of the returns,
the Sharpe ratio,
CAGR or Compound annual growth rate, and
the conversion between percent and log returns.
Note that these measures are calculated quite differently
for log and percent returns.
Commenting this worksheet from the top down,
some conventions apply:
Small-cap-r is taken to be a vector of returns, r-sub-i is a single small-cap-r return.
Ditto for big-cap-r.
Small cap n is the number of samples (i.e. the number of returns).
Big-cap-r (percent returns) are regularly transformed into "return factors"
by the addition of a 1,
then the "return factors" are processed in various ways
(e.g. by exponentiation or multiplication), then
the final step is to subtract the 1 that had been added
to turn the "return factors" back into returns.
This turning of the percent returns into "return factors" is obligatory.
By contrast,
Small-cap-r (log returns) do not require turning into "return factors"
(though optionally, adding one is sometimes done
to make the starting point of the equity curve equal to 1).
Also note that log returns are additive,
so the cumulative log returns involve addition, and
the average log returns involve arithmetic averaging.
By contrast cumulative percent returns involve multiplication, and
average percent returns involve geometric averaging
(of the percent "return factors" having added 1).
The annualized versions of average returns
(whether geometric or arithmetic)
involves exponentiation (or multiplication) by the "period."
In this worksheet,
the "period" is actually the sampling frequency of the returns
(our "period" is how many returns fit into a year).
For daily returns, the "period" is 252 since
there are 252 trading days in a year.
If you are dealing with weekly returns, the "period" is 52,
if you are dealing with monthly returns, the "period" is 12.
So when you divide the number of data items by the "period,"
you get the number of years, which
is essential for annualization.
In the case of CAGR,
you will notice that it is just the geometric mean
of the percent "return factors," but annualized.
The workpages large_r and small_r
show how to calculate the various measures using Excel.
Cumulative log returns are flatter (less steep) than
cumulative percent returns, an effect of the log transformation,
but annualized return, volatility and

the sharpe ratio of the log returns are actually realistic,
and comparable to the percent return version of these measures,
which you can check by checking the numbers that agree in colour.
In the end, what is important is that you be
consistent in your calculations so that you can compare
apples to apples, not apples to oranges.