Algorithmica
Research AB

CAPM, Black-Litterman and portfolio optimization
using Quantlab 3.0

Last update: 2009-01-22 written by Robert Thorén.

CAPM, Black-Litterman, and portfolio optimization

Here we show how easy it is to create a simple workspace that implements Harry Markowitz CAPM theory of equilibrium returns[1] together with the Black-Litterman[2] way of combining an investor view to produce well balanced portfolios using classical constrained mean-variance portfolio optimization.

Background

As well known by the investment community, portfolio optimization in practise, does not behave well when using historical returns or investor-view returns as input in the mean-variance optimization. So called "corner portfolios" where most weight is allocated to one or two assets with the highest expected return is very common.

A simple way, used by many practitioners to get around the "corner portfolio" problem is to simply set constraints on the maximum weight on single or groups of assets. This however, often means that the optimal solution will "get stuck" on any of these boundary conditions. Such "optimal portfolios" could usually have been guessed a priori and the whole procedure is then just for show.

A natural starting point of estimating expected returns can be found in the CAPM model. In simple terms, investors want compensation for taking on higher than average risk. By using the assets individual risk and their correlation and hence their risk contribution to the "benchmark" portfolio, an excess return premium can be calculated. The risk premium can be viewed as a mean excess return given that no investors have any other views than what is implied by the assets risk and correlation.

The Black-Litterman model gives us the tools to combine a CAPM equilibrium return estimate with the investor's own views. Mathematically what Black-Litterman suggested was that the two views could be joined together using Bayesian inference theory. This gives the investor a chance to combine not only the return view but also a confidence level of that view. Further, some assets can be left without a view and then they will be adjusted to account for the correlation with other assets.

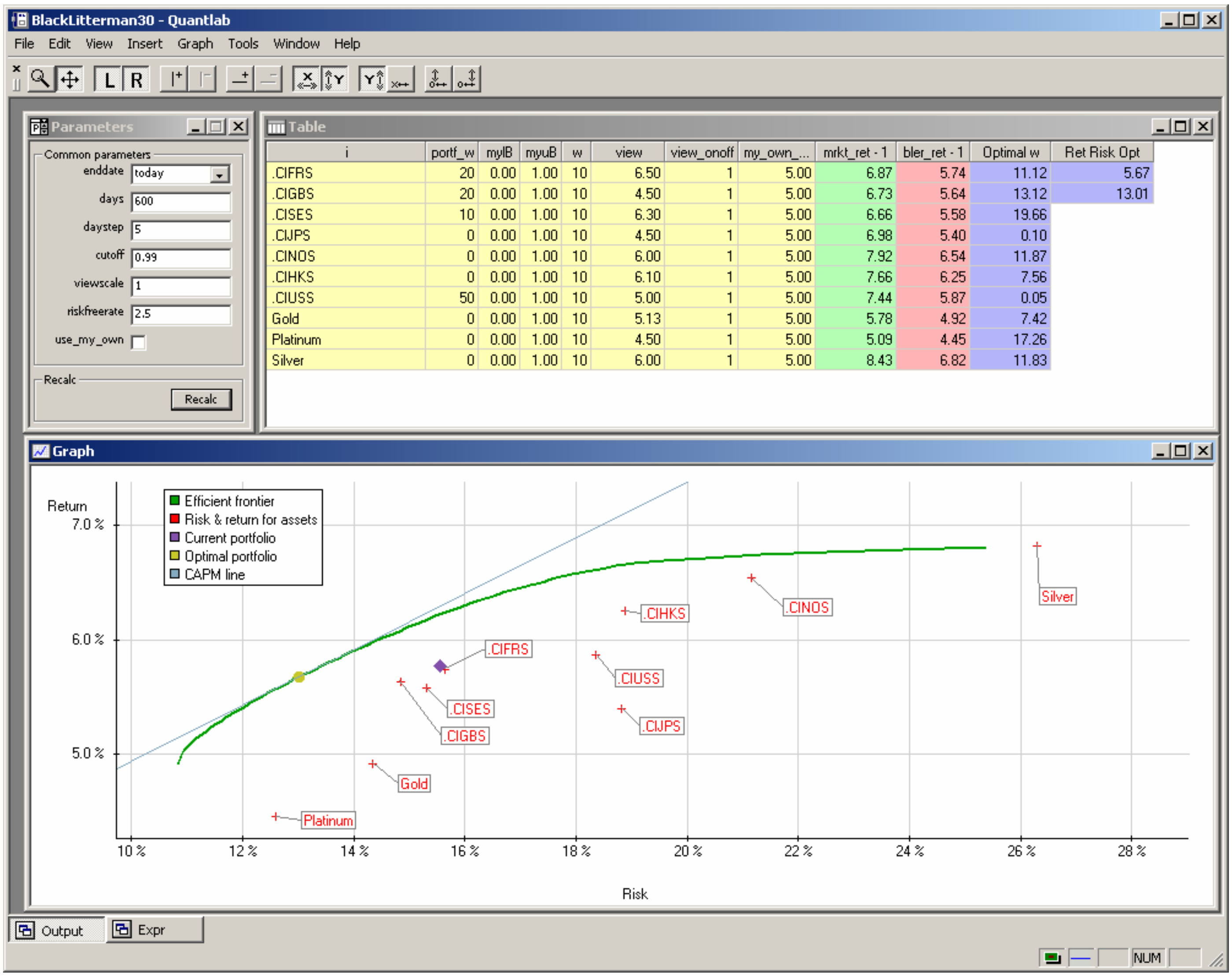
A practical example

Let's look at a Quantlab 3.0 workspace and the *key* functions used to create it.

We have used some arbitrary[3] data series representing assets or asset classes. The input vector of assets or indices is expandable simply by right clicking on the column header. By double clicking on the cell with the index name or asset a security selection dialog will appear. Quantlab keeps track of available instruments and their time-series etc.

[1] See Markowitz, Harry, "Portfolio Selection", Journal of Finance, 1952
[2] See Black, Fischer and Litterman, Robert, "Asset Allocation: Combining Investor Views With Market Equilibrium", Goldman Sachs, Fixed Income Research, 1990
[3] In fact, here we have chosen some MSCI country indices together with some commodities. ".CIFRS" is a French index and ".CIGBS" is a Great Britain index.

The left dialog box show common parameters. Time series used in the covariance calculation is set from today and back 600 business days using a 5 day step size. In order to use very large sets of potential investable indices or securities, we use a singular value decomposition procedure[4] within the Black-Litterman calibration. A cut-off can be used to control the percentage of total variance to account for when cutting off data that is linearly dependent. “Viewscale” is the common parameter to assert a certain confidence in the investor views. The higher the value the more weight will be given to the investor view in favour of the equilibrium view.



Common parameters

enddate	today
days	600
daystep	5
cutoff	0.99
viewscale	1
riskfreerate	2.5
use_my_own	☐

Recalc

i	portf_w	mylB	myuB	w	view	view_onoff	my_own_...	mrkt_ret - 1	bler_ret - 1	Optimal w	Ret Risk Opt
.CIFRS	20	0.00	1.00	10	6.50	1	5.00	6.87	5.74	11.12	5.67
.CIGBS	20	0.00	1.00	10	4.50	1	5.00	6.73	5.64	13.12	13.01
.CISES	10	0.00	1.00	10	6.30	1	5.00	6.66	5.58	19.66	
.CIJPS	0	0.00	1.00	10	4.50	1	5.00	6.98	5.40	0.10	
.CINOS	0	0.00	1.00	10	6.00	1	5.00	7.92	6.54	11.87	
.CIHKS	0	0.00	1.00	10	6.10	1	5.00	7.66	6.25	7.56	
.CIUSS	50	0.00	1.00	10	5.00	1	5.00	7.44	5.87	0.05	
Gold	0	0.00	1.00	10	5.13	1	5.00	5.78	4.92	7.42	
Platinum	0	0.00	1.00	10	4.50	1	5.00	5.09	4.45	17.26	
Silver	0	0.00	1.00	10	6.00	1	5.00	8.43	6.82	11.83	

The table has the following input / output columns
i: the instrument or asset list
Portf_w: a current/arbitrary portfolio to compare the resulting with
myLB: a lower bound for the resulting portfolio weights
myUB: an upper bound for the resulting portfolio weights
w: benchmark portfolio / market cap weighs to calculate the equilibrium returns from CAPM
view: the investor expected return view including the risk free rate expressed as a percentage
view_onoff: a possibility to turn a particular investor view on or off. 1 = on, 0 = off.

[4] An extension of the original Black-Litterman procedure by Peter Alaton, Partner and Head of Quantitative Research, Algorithmica Research AB

my_own: a possibility to enter an arbitrary alternate view that can be used for comparison
mrkt_ret: calculated CAPM expected return including the risk free rate
bler_ret: Black-Litterman expected return using the view and the mrkt_ret as input
Optimal_w: the resulting optimal asset weights for the super efficient portfolio
Ret Risk opt: the resulting portfolio expected return and risk for the super efficient portfolio

Some coding assistance in Quantlab 3.0

So what do we need to create such a workspace and which functions does Quantlab 3.0 help us with?

First of all we get the Black-Litterman function adapted for large scale problems. Some nice functions to have then are; the Singular Value Decomposition (SVD), a matrix and vector based language, and some matrix inversion functions like gauss_jordan.

```
void bler(       out vector(number) r,
                 out number cutoffindex,
                 matrix(number) Sigma,
                 number tau,
                 matrix(number) P,
                 vector(number) pii,
                 vector(number) q,
                 vector(number) alpha,
                 number cutoff){

// Extended Black-Litterman estimation which takes care of the case of a (nearly) singular Sigma matrix
// Output:
// r: Black-Litterman expected return, n rows column vector
// cutoffindex: A scalar that corresponds to the number of factors used
//
// Input:
// Sigma: Variance-covariance matrix for factor returns, n*n diagonal matrix.
// tau: proportionality for the variance-covariance matrix used for equilibrium returns, scalar
// P: P is a k*n matrix that determines which securities or spreads that we have an opinion of.
// pii: Market implied returns, n rows column vector
// q: A k rows column vector expressing our views
// alpha: A k rows column vector that scales the variances in Sigma to get the variances in Omega.
// cutoff: scalar that determines how much of variance that should be taken
// care of, for example 0.95//.

… some code omitted here …

     //Singular value decomposition, S is diagonal covariance matrix
    matrix(number) U, V;
    vector(number) S;
    svd(Sigma, U, S, V);
...
    // Normalise with factor standard deviations so that all factors have variance 1.
    matrix(number) Vinv = diag_matrix(S^(-0.5))*transpose(V);
    V = V*diag_matrix(S^0.5);
...

    // Calculate market implied risk premia.
    vector(number) rf = gauss_jordan(Sinv/tau + transpose(Pf)*Omegainv/tau*piif +
    transpose(Pf)*Omegainv);

}
```

So that was nice and actually the bler() function is part of the Portfolio optimization library so you don't have to bother with the above.

But what do we use more? Well we need to fetch historical data for all the assets that we have chosen in the instrument vector. Then we also need to calculate the CAPM returns using the historical data (i.e. the covariance matrix) and the market price of risk.

```
out matrix(number) my_bler_call(… input params here …){
…
            vector(number) mkter;                       // market excess return
            vector(number) mktr;                        // market return
            vector(number) bl_er;                       // Black-Litterman excess return
…

            // GET THE HISTORICAL DATA FOR ALL INSTRUMENTS AT THE SAME TIME
            // OUTPUT IS A DATA MATRIX  FOR ALL DEFINED BUSINESS DAYS IN SWEDEN
            series<date>(vector(number)) prices = series(p:move_bus_days(enddate,-
            days,calendar('SWEDEN')),enddate,daystep;get_instr_quote(i,p));

            //CONVERT TO RETURNS
            series<date>(vector(number)) log_prices  = change(log(prices),calendar('SWEDEN'));

            // CALCULATE COVARIANCE MATRIX
            cov = covariance(log_prices)*250/daystep;

            //CALCULATE EXCESS CAPM RETURN
            mkter = price_of_risk*cov*w;

            // CALL THE BLACK-LITTERMAN procedure to get the actual calibration done on real data.
            bler(bl_er, cutoffindex, cov, 1, ViewIndex, mkter, view_er, vscale, cutoff);

            // Add the risk free rate in order to get returns
            mktr = rf + mkter;
            bl_r = rf + bl_er;

            //Return a 2xN matrix with the CAPM returns and the Black-Litterman returns expressed as
            //percent
             return transpose([mktr, bl_r])*100;

}
```

Now we can loop through the frontier and calculate all optimal portfolios along the way. We also catch the super-efficient portfolio on the way. In the following lines of code we enjoy the non-linear optimization routine find_min_risk_markowitz(…) with linear constraints also found in the Portfolio optimization library.

```
out vector(point_number) frontier(){

            vector(number) w;

            if (umo)
                      bl_r = my_r;

            number r_des;
            number n = v_size(bl_r);
            vector(point_number) front;
```

```
	// Find lowest possible risk to start with. Minimization of risk with constraint that all weights sum
	// to 1.
	w = find_min_risk_markowitz(cov, bl_r, 0, lB,uB);
	risk = sqrt(w*cov*w);
	r_des = w*bl_r;
	push_back(front, point(risk, r_des));

	// Find largest return
	number r_max = max_ret(bl_r, lB, uB);
	// Find lowest return
	number r_min = v_min(bl_r);
	number step = (r_max-r_min)/100;
	number k = 0;
	// Go through the front
	while(r_des <= r_max){
		try{
				w = find_min_risk_markowitz(cov, bl_r, r_des, lB, uB);
		} catch {
				break;
		}
		risk = sqrt(w*cov*w);

		// Catch the super efficient portfolio
		if((r_des-rf)/risk > k){
				w_optimal = w;
				ret_optimal = r_des;
				risk_optimal = risk;
				k = (r_des-rf)/risk;
		}

		push_back(front, point(risk, r_des));
		r_des = r_des + step;
	}
	return front;
}
```

The frontier function is declared with the key word “out” and can then be attached to a graph or a table. It is from this function we get our plotted frontier.

We are now ready to get to the actual job of playing around with the assets, historical time-frame and our own views in order to get portfolio suggestions out to the users. The whole implementation of the workspace shown in the screen-shot above is less than 100 lines of actual code, including the bler() function.