Running New Optimiser as a Server

1. Download dotnet version 6.
2. Clone the optimiser from <http://username@bitasource64:7990/scm/bita/com.bita.starng.optimiser.git>.
3. From a command prompt, change directory to the top directory of the optimiser clone and run
4. dotnet build

This takes over 5 mins when starting from scratch and will build the optimiser library, BitaServer program and BITAPlus program.

If this fails try dotnet nuget locals all –clear then do dotnet build again

1. To integrate the optimiser with BITA20, use BitaServer running as a service. This adopts json input and output.

Creating and running BitaServer service

1. dotnet dev-certs https –trust

Run this command once to identify localhost as trusted

1. Create the directory c:\server
2. Open a new command prompt as administrator and change directory to the BitaServer directory and type;

dotnet publish -c Release -r win-x64 --sc -o c:\server

this will build BitaServer in c:\server. The files created here do not require dotnet at all.

1. Create a service

sc create "BITA+" binPath= c:\server\BitaServer.exe DisplayName="Optimiser Server"

this only needs to be done once (but must be done as administrator)

1. Start the service

sc start “Bita+”

1. The service should now be working on <http://localhost:7777/optimise/general>
2. curl -X POST -H "Content-type: application/json" -d @input.json [http://localhost:7777/optimise/general -o output.json](http://localhost:7777/optimise/general%20%20-o%20output.json)

will run an optimisation with json data in input.json and produce output in output.json

There are several examples for input files in the top directory for the optimiser.

1. The optimiser needs to be licensed, this is done by running the reglicenceR.exe program. This can be downloaded from JIRA ticket BITA20-2843.

This program is similar to the one used by Ibrahim’s team except that it must be run as administrator because it copies the licence key into the root

Area rather than the user area.

Another way to license the optimiser is to run

dotnet test

as administrator; one of the tests writes a licence key.

The attributes in the input json file are defined by the input parameters the following method in BlasLike\Portfolio.cs;

The main method for performing the portfolio optimisations is

public static int OptimiseGeneral(  
int n, //Number of assets  
int nfac, //Number of factors, or -1 if historic covariance risk  
model  
string[] names, //Names of assets (can be null)  
double[] w, OUTPUT //Output asset weights  
int m, //number of linear constraints  
double[] A, //Constraint array A[i\*m+j] is the exposure of the j'th  
contraint to the i'th asset  
double[] L, //Array of lower bounds L[i] is the lower bound for asset i  
if i<n otherwise L[i+n] is the lower bound for the i'th  
constraint  
double[] U, //Array of upper bounds U[i] is the upper bound for  
asset i if i<n otherwise U[i+n] is the upperbound for the  
i'th constraint  
double[] alpha, //Array of asset expected returns  
double[] benchmark, //Array of asset benchmark weights  
double[] Q, //If nfac is -1 array of n\*(n+1)/2 historic asset covariances  
double gamma, //The utility is -gamma/(1-gamma)\*expected relative  
return +0.5\*relative variance  
double[] initial, //Array of initial weights  
double delta, //Desired turnover  
double[] buy, //Array of asset buy costs  
double[] sell, //Array of asset sell costs  
double kappa, //Cost part of utility is kappa/(1-kappa)\*cost. If kappa is  
negative cost part of utility is gamma/(1-gamma)\*cost  
int basket, //maximum number of non-zero asset weights  
int trades, //Maximum number of non-zero asset trades  
double min\_holding, //lowest non-zero asset weight  
double min\_trade, //lowest non-zero asset trade  
double Rmin, //lowest value for -(total short weight)/(total long weight)  
(positive)  
double Rmax, //highest value for -(total short weight)/(total long  
weight) (positive)  
int round, //Use round lots if round=1, otherwise 0  
double[] min\_lot, //Asset weight w[i] must be on the "ladder"  
min\_lot[i]+k\*size\_lot[i] where k is integer  
double[] size\_lot,  
int[] shake, OUTPUT //For rounding (or min\_hold/min\_thresh) shake[i]=i for  
assets whose weights could not be rounded properly  
double LSValue, //Maximum total long weight in a long/short portfolio  
int nabs, //Number of absolute value constraints in a long/short  
portfolio defined in Abs\_A  
double[] Abs\_A, //Array Abs\_A[i\*n+j] is the exposure of the i'th asset to the  
j'th absolute constraint  
int mabs, //Number of absolute value constraints in a long/short  
portfolio defined in I\_A  
int[] I\_A, //Integer array, I\_A[i]=j means that the i'th absolute  
constraint in I\_A is defined using the data for the j'th  
constraint in array A  
double[] Abs\_U, //Array of upper bounds for absolute constraints, (length  
nabs+mabs)  
double[] FC, //Array of nfac\*(nfac+1)/2 factor covariances (if nfac>-1)  
double[] FL, //Array of factor betas FL[i+i\*n] is the exposure of the j'th  
factor for the i'th asset  
double[] SV, //Array of n specific variances  
double minRisk, //minimum risk or negative value for no constraint  
double maxRisk, //maximum risk or negative value for no constraint  
ref double ogamma, OUTPUT //the value of gamma that gives the desired risk  
double[] mask, //array length n used to keep certain assets from turnover  
constraint. null if not needed  
int longbasket, //number non zero assets with positive weight  
int shortbasket, //number non zero assets with negative weight  
int tradebuy, //number non zero trades that are buys  
int tradesell, //number non zero trades that are sells  
double LSValuel, //Minimum total long weight in a long/short portfolio  
double[] Abs\_L, //Array of lower bounds for absolute constraints, (length  
nabs+mabs)  
double[] breakdown, OUTPUT //breakdown per asset of ETL or LOSS  
ref bool CVARGLprob, OUTPUT//ETL or LOSS did not converge properly  
int tlen, //Length of historic period for ETL or LOSS  
double DATAlambda, //Strength of ETL or LOSS in the utility function (probably 1)  
double[] DATA, //For ETL array of losses (i.e. -return), for GAIN LOSS array of  
returns; DATA[i\*tlen+j] is the return in the j'th period for the i'th  
asset otherwise null  
double tail, //for ETL this is the probability of being in the distribution tail  
(0.05)  
double[] targetR, //null for ETL, array of tlen target returns for GAIN LOSS  
bool ETLorLOSSconstraint, //if true, set up constraint for ETL or LOSS  
double ETLorLOSSmin, //lower bound for ETL or LOSS constraint  
double ETLorLOSSmax, //upper bound for ETL or LOSS constraint  
string logfile, //either "" or string to define a file to hold log information  
int revise, //if 0 min\_lot and size\_lot apply to assets weights, if 1 they apply  
to asset trades  
)