Contents

MIE377 (Winter 2024) - Laboratory 2

The purpose of this laboratory is to solve a mixed-integer quadratic program (MIQP) that finds an optimal portfolio with a cardinality constraint and a buy-in threshold. We will use the optimizer Gurobi to solve this MIQP. Gurobi is an optimization software that we can call from within Matlab.

TA: David Islip Instructor: Roy H. Kwon

```
% Name: Aaryan Nagpal
% Student ID: 1007792596

clc
clear all
format short

% Program Start
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

PART 1: Data pre-processing

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Load the sample historical data
load('lab2data.mat')

% Calculate the asset and factor returns (factor models use returns, not
% prices)
rets    = prices( 2:end, : ) ./ prices( 1:end - 1, : ) - 1;
facRets = sp500price( 2:end , 1 ) ./ sp500price( 1:end - 1, 1 ) - 1;

% Number of assets
n = size(rets,2);

% Number of observations;
N = size(rets, 1);

% Calculate the factor expected excess return from historical data using
% the geometric mean
mu = (geomean(rets + 1) - 1)';

% Calculate the asset covariance matrix
Q = cov(rets);

% Calculate the factor expected excess return from historical data using
% the geometric mean. Use this as the portfolio target return
```

```
targetRet = geomean(facRets + 1) - 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

PART 2: Cardinality and buy-in thresholds

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Total of 10 out of n total assets
k = 10;

% Buy-in lower bound for each asset
lBuy = 0.05;

% Buy-in upper bound for each asset
uBuy = 0.2;

% Note:
% We are adding "n" auxiliary binary variables (one per asset). This means
% we now have the n continuous variables "x" and the n binary variables
% "y". However, in MATLAB, we treat this as a single vector with 2n
% variables, with vars = [x; y].

% Since our problem now has 2n variables, we must re-size mu and Q
% accordingly.
mu = [mu; zeros(n,1)];
Q  = [Q zeros(n); zeros(n,2*n)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

PART 3: Setup our input parameters for Gurobi

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%-------------------------------------------------------------------------
% 3.1 Inequality constraints:
% Gurobi accepts inequality constraints of the form "A x <= b" and
% "A x >= b". However, for consistency, we will keep all constraints as
% "A x <= b"
%-------------------------------------------------------------------------

% We have 2n buy-in constraints (lower and upper bounds), and each
% constraint defines the lower or upper bound by pairing a single asset
% weight with its corresponding auxiliary variable. This matrix will be of
% dimension 2n * 2n

B = [-eye(n) lBuy.*eye(n); eye(n) -uBuy.*eye(n)];

% We must also include the target return constraint. We can add another row
% onto matrix B to account for the target return constraint. Therefore, our
% complete matrix A will have dimension (2n + 1) * 2n

A = [-mu';B];

% We must also define the right-hand side coefficients of the inequality
```

```
% constraints, b, which is a column vector of dimension (2n + 1)

b = [-targetRet; zeros(2*n,1)];

%-------------------------------------------------------------------------
% 3.2 Equality constraints:
% We will define our cardinality constraint as an equality (although this
% is not always the case, we could also define it as an inequality)
%-------------------------------------------------------------------------

% We only have 2 equality constraints: the cardinality constraint (sum of
% y's) and the budget constraint (sum of x's):

Aeq = [ones(n,1) zeros(n,1); zeros(n,1) ones(n,1)]';

% We must also define the right-hand side coefficients of the equality
% constraints, beq:

beq = [1 k]';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

PART 4: Setup Gurobi model

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%-------------------------------------------------------------------------
% 4.1 Define the model variables and assign them a name
%-------------------------------------------------------------------------

% Define the variable types:'C' defines a continuous variable, 'B' defines
% a binary variable
varTypes = [repmat('C', n, 1); repmat('B', n, 1)];

% Input the lower and upper bounds. Since our lower buy-in threshold is
% 0.05, this means we are not allowed to short-sell.
lb = zeros(2*n, 1);
ub = ones(2*n, 1);

% Assign tags for the model variables. The Matlab variable 'tickers' was
% loaded with the rest of the market data. The tickers are the tags we will
% use to identify the assets.

% Append '_c' to cont var. names
namesCont = cellfun(@(c)[c '_c'], tickers(1:n), 'uni', false);

% Append '_b' to binary var. names
namesBin = cellfun(@(c)[c '_b'], tickers(1:n), 'uni', false);

% Combine both name vectors
names = [namesCont namesBin];

%-------------------------------------------------------------------------
% 4.1 Setup the Gurobi model
%-------------------------------------------------------------------------
clear model;
```

```matlab
% Assign the variable names
model.varnames = names;

% Gurobi accepts an objective function of the following form:
% f(x) = x' Q x + c' x

% Define the Q matrix in the objective
model.Q = sparse(Q);

% define the c vector in the objective (which is a vector of zeros since
% there is no linear term in our objective)
model.obj = zeros(1, 2*n);

% Gurobi only accepts a single A matrix, with both inequality and equality
% constraints
model.A = [sparse(A); sparse(Aeq)];

% Define the right-hand side vector b
model.rhs = full([b; beq]);

% Indicate whether the constraints are ">=", "<=", or "="
model.sense = [ repmat('<', (2*n + 1), 1) ; repmat('=', 2, 1) ];

% Define the variable type (continuous, integer, or binary)
model.vtype = varTypes;

% Define the variable upper and lower bounds
model.lb = lb;
model.ub = ub;

% Set some Gurobi parameters to limit the runtime and to avoid printing the
% output to the console.
clear params;
params.TimeLimit = 100;
params.OutputFlag = 0;

results = gurobi(model,params);

fprintf('Optimal obj. value: %1.6f \n\nAsset weights:\n', results.objval);
for i=1:n
    if(results.x(n+i) ~= 0)
        fprintf('    (+) %3s %1.6f\n', tickers{i}, results.x(i));
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program End
```

```
Optimal obj. value: 0.000167

Asset weights:
   (+)  FB 0.050429
   (+)  PG 0.143818
   (+)   T 0.114983
   (+) WMT 0.068326
   (+) MRK 0.080193
   (+) PEP 0.130251
```

```
(+)  MO 0.057874
(+) UNH 0.096745
(+) MCD 0.174066
(+) LLY 0.083315
```

Published with MATLAB® R2023a