DM559/DM545 – Linear and Integer Programming

Answers to Obligatory Assignment 1.1, Spring 2018

93039159 | Jeff Gyldenbrand (jegyl16)

Task 1

Notation

Let j be an asset from a set of N potential investments, and let R_j be a random variable indicating the return of the investment of $j, j = 1, ..., N$ in the next time period of investment. Furthermore let r_{jt} denote the historical return on investment j from month t to month $t + 1$

Model

$$\widehat{MAD} = \frac{1}{T} \sum_{t=1}^{T} \left[\left| \sum_{j=1}^{N} x_j (r_{tj} - \widehat{R}_j) \right| \right] \tag{1}$$

Minimize: $\widehat{MAD}$

Subject to:

$$min \frac{1}{T} \sum_{t=1}^{T} z_t \tag{2}$$

$$z_t \geq \sum_{j=1}^{N} x_j (r_{jt} - \widehat{R}_j) \qquad j = 1, ..., N, t = 1, ..., T \tag{3}$$

$$z_t \geq -\left(\sum_{j=1}^{N} x_j (r_{jt} - \widehat{R}_j) \right) \qquad j = 1, ..., N, t = 1, ..., T \tag{4}$$

$$x_j \geq 0 \tag{5}$$

$$\sum_{j=1}^{N} x_j = 1 \tag{6}$$

$$B \leq \sum_{j=1}^{N} x_j \widehat{R}_j \tag{7}$$

$$j, t \in \mathbb{N} \tag{8}$$

$$z_t \in \mathbb{R} \tag{9}$$

Explanation of the model

The model is based on equation (1) which is the estimated reward and risk of all the assets, known as the mean of the absolute deviation (MAD). The objective is to minimize $\widehat{MAD}$ as shown in equation (2), where the absolute value is denoted z_t.

The first two constraints is shown in equation (3) and (4). Together they constrain the value z_t to an absolut value, because we are only interested in the deviation.

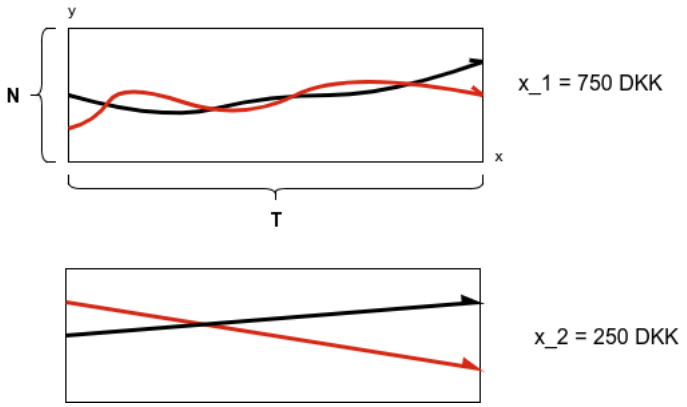

fig:1

Number of variables and constraints

Constraint (3) denotes one constraint for each t = T
Constraint (4) denotes one constraint for each t = T
= 2T

Constraint (5) denotes n constraints for each x_j = n
Constraint (6) denotes 1 constraint = 1
Constraint (7) denotes 1 constraint = 1

Total constraints: 2T + n + 2
Total variables: z_t + x_j

Max number of assets, with reason

Task 2

Your code

```
#!/usr/bin/python
from __future__ import division
from gurobipy import *
import matplotlib.pyplot as plt

class Data:
    def __init__(self, filename):
        f = open(filename, "r")
        lines=f.readlines();
        f.close();
        self.r={}
        self.R =[]
        self.max_r = 0
        self.min_r = GRB.INFINITY
        N = int(lines[0].strip("\r\n"))
        self.T = int(lines[1].strip("\r\n"))
        self.assets = range(1,N+1)
        self.times = range(1,self.T+1)
        for line in lines[2:]:
            line=line.strip("\r\n");
            parts=line.split(" ");
            j=int(parts[0])
            t=int(parts[1])
            val=float(parts[2])
            self.r[j,t]=val

        calculate(self)

def solve(data, epsilon):
    m = Model("portfolio")
    m.setParam(GRB.param.Method, 0)

    B = data.min_r + (epsilon/100)*(data.max_r-data.min_r);

    ######### BEGIN: Write here your models
    z_t = {}
    x_j = {}
    for n in range(len(data.assets)):
        j = data.assets[n]
        x_j[j] = m.addVar( vtype = GRB.CONTINUOUS, name = "x" )

        # Constraint number 3: (equation (5) in report)
        m.addConstr( 0 <= x_j[j] )

    # Constraint number 4: (equation (6) in report)
    m.addConstr( 1 == quicksum ( map ( lambda x: x_j[x], data.assets )))

    for n in range(len(data.times)):
        t = data.assets[n]
        z_t[t] = m.addVar( vtype = GRB.CONTINUOUS, name = "x" )

        # Constraint number 1: (equation (3) in report)
        m.addConstr( z_t[t] >= ( quicksum ( map ( lambda x: x_j[x] * ( data.r[x,t] -
            data.R[x-1] ), data.assets ))))

        # Constraint number 2: (equation (4) in report)
        m.addConstr( z_t[t] >= -( quicksum ( map ( lambda x: x_j[x] * ( data.r[x,t] -
            data.R[x-1] ), data.assets ))))

        # Constraint number 5: (equation (7) in report)
        m.addConstr( B <= quicksum( map ( lambda x: x_j[x] * data.R[x-1], data.assets )
            ))

    m.setObjective( quicksum( map ( lambda x: z_t[x] ,data.times )) / data.T, GRB.
        MINIMIZE )
    m.optimize()
```

```
    ######### END

    return m.objVal, B

def calculate(data):
    for j in data.assets:
        m = 0
        for t in data.times:
            m += data.r[j,t]
        m /= data.T
        if data.max_r < m:
            data.max_r = m
        if data.min_r > m:
            data.min_r = m
        data.R.append(m)

    if (0 > data.min_r):
        data.min_r = 0

def main(argv):
    if len(argv) != 1:
        usage()
    instance = Data(argv[0])

    epsilons = xrange(10,100,10)
    risks = [0 for i in epsilons]
    rewards = [0 for i in epsilons]
    i=0
    for epsilon in epsilons:
        print "=== Solve for epsilon = ", epsilon
        risks[i], rewards[i] = solve(instance, epsilon)
        i+=1
    plt.plot(risks, rewards,'-o')
    plt.xlabel('risk')
    plt.ylabel('reward')
    plt.show()

def usage():
    print "Reads data from datafilename";
    print "Usage: [\"datafilename\"]\n";
    sys.exit(1);


if __name__ == "__main__":
    main(sys.argv[1:])
```

Comments on the run of Gurobi and statistics on the solutions

For all epsilon values, we see that gurobi statistics optimizes a model with 1328 rows, 747 coloumns and 399084 nonzeros.

For epsilon value 10 1087 iterations is runned, whereas for epsilon value 20, 993 iterations, and epsilon value 30, 863 iterations, all the way to the last epsilon value 90, with only 227 iterations.

Your plot

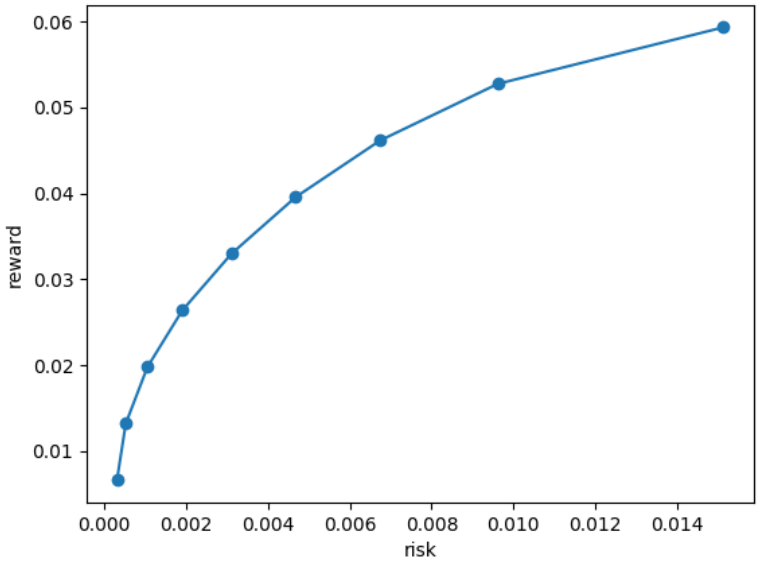

fig:2

Comments on the plot: is it as expected? What do we learn?

n/a

Task 3

Which model is linearizable?

$$maxmin_{t=1,..,T} \sum_{j=1}^{n} x_j r_{jt} \tag{10}$$

Notation

Let j be an asset from a set of n investments, and let x_j denote the actual investment amount. Furthermore let r_{jt} denote the return of the investment of the time $t = 1, .., T$

New Model

Maximize: z

$$min_{t=1,..,T} \sum_{j=1}^{n} z \tag{11}$$

Subject to:

$$z \leq \sum_{j=n}^{n} x_j r_{jt} \tag{12}$$

$$\sum_{j=n}^{n} = 1 \tag{13}$$

$$\sum_{j=n}^{n} x_j \geq 0 \tag{14}$$

$$B \leq \sum_{j=n}^{n} x_i r_{jt} \tag{15}$$

Explanation of the model

The model is based on equation (10) which maximizes the return of the portfolio in the worst period. The objective is to maximize z, as seen in equation (11), with the first constraint, that the sum of $x_j r_{jt}$ must be greater than z, as seen in equation (12). Furthermore the sum of all assets must be equal to 1, and greater than zero, as seen in equation (13) and (14). Last constraint is to make sure B is less than the sum of $x_j r_{jt}$.

Source code

```
#!/usr/bin/python
from __future__ import division
from gurobipy import *
import matplotlib.pyplot as plt

class Data:
    def __init__(self, filename):
        f = open(filename, "r")
```

```
        lines=f.readlines();
        f.close();
        self.r={}
        self.R =[]
        self.max_r = 0
        self.min_r = GRB.INFINITY
        N = int(lines[0].strip("\r\n"))
        self.T = int(lines[1].strip("\r\n"))
        self.assets = range(1,N+1)
        self.times = range(1,self.T+1)
        for line in lines[2:]:
            line=line.strip("\r\n");
            parts=line.split(" ");
            j=int(parts[0])
            t=int(parts[1])
            val=float(parts[2])
            self.r[j,t]=val

        calculate(self)

def solve(data, epsilon):
    m = Model("portfolio")
    m.setParam(GRB.param.Method, 0)

    B = data.min_r + (epsilon/100)*(data.max_r-data.min_r);

    ######### BEGIN: Write here your models
    x_j = {}
    z = {}

    for n in range(len(data.assets)):
        j = data.assets[n]
        x_j[j] = m.addVar( vtype = GRB.CONTINUOUS, name = "x" )

        # Constraint number 3: (equation (14) in report)
        m.addConstr( 0 <= x_j[j])

    z = m.addVar( vtype = GRB.CONTINUOUS, name = "x" )
    for n in range(len(data.times)):
        t = data.assets[n]

        # Constraint number 1: (equation (12) in report)
        m.addConstr( z <= quicksum( map ( lambda x: x_j[x] * data.r[x, t], data.assets)
            ))

    # Constraint number 2: (equation (13) in report)
    m.addConstr( 1 == quicksum ( map ( lambda x: x_j[x], data.assets )))

    # Constraint number 4: (equation (14) in report)
    m.addConstr( B <= quicksum( map ( lambda x: x_j[x] * data.r[x, t], data.assets )))

    m.setObjective( quicksum( map ( lambda x: z, data.times )), GRB.MAXIMIZE)
    m.optimize()
    ######### END

    return m.objVal, B

def calculate(data):
    for j in data.assets:
        m = 0
        for t in data.times:
            m += data.r[j,t]
        m /= data.T
        if data.max_r < m:
            data.max_r = m
        if data.min_r > m:
            data.min_r = m

    if (0 > data.min_r):
        data.min_r = 0

def main(argv):
```

```
    if len(argv) != 1:
        usage()
    instance = Data(argv[0])

    epsilons = xrange(10,100,10)
    risks = [0 for i in epsilons]
    rewards = [0 for i in epsilons]
    i=0
    for epsilon in epsilons:
        print "=== Solve for epsilon = ", epsilon
        risks[i], rewards[i] = solve(instance, epsilon)
        i+=1
    plt.plot(risks, rewards,'-o')
    plt.xlabel('risk')
    plt.ylabel('reward')
    plt.show()

def usage():
    print "Reads data from datafilename";
    print "Usage: [\"datafilename\"]\n";
    sys.exit(1);


if __name__ == "__main__":
    main(sys.argv[1:])
```

Plot of the forntier

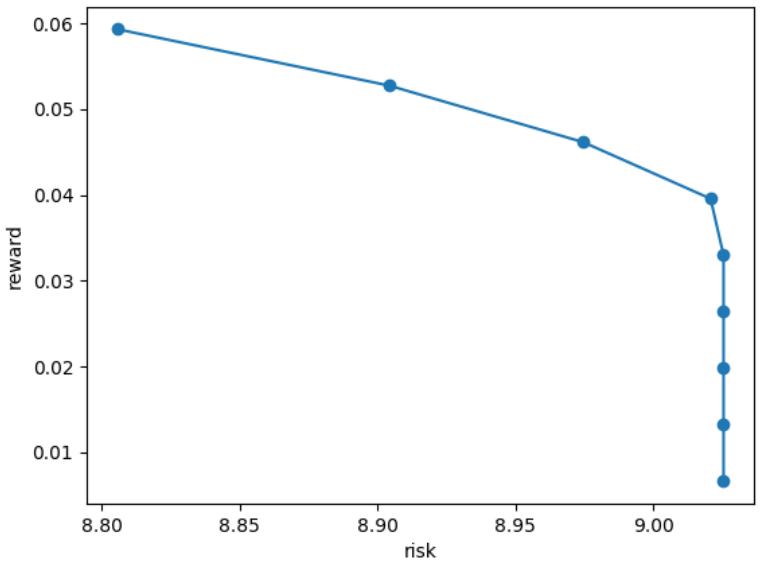

fig:2

Comments

n/a