



## **Introduction to conic optimization**

June 20th 2016 - June 22th 2016

[e.d.andersen@mosek.com](mailto:e.d.andersen@mosek.com)

[www.mosek.com](http://www.mosek.com)





- More applications.
  - Minimal circle problem.
  - Portfolio optimization (or how to get rich maybe).
  - Robust linear optimization.
- Solving examples using **MOSEK** Fusion.

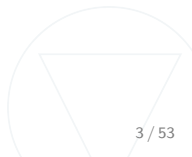
## Section 1

Applications continued





- Problem: Assume you should place a fire station so the maximum distance from the station to each house in the district is minimized.
- This is a minimal enclosing circle problem.
- Wikipedia info:  
`en.wikipedia.org/wiki/Smallest-circle\_problem`





Let  $n$  points

$$p^i \in \mathbb{R}^2$$

be given. Find the smallest enclosing circle.

Let  $x$  be the center of a circle, then we want to find  $x$  such that

$$\|p^i - x\|$$

is minimized.

Observe

$$\begin{pmatrix} r \\ p^i - x \end{pmatrix} \in \mathcal{K}_q$$

implies

$$\|p^i - x\| \leq r.$$

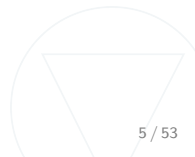


CQ model:

$$\begin{array}{ll} \text{minimize} & r \\ \text{subject to} & \begin{pmatrix} r \\ p^i - x \end{pmatrix} \in \mathcal{K}_q, \quad \forall i. \end{array}$$

[1] solves it using GAMS+(Knitro,Conopt,MINOS)

- Formulation is nonconvex.
- Must choose a good starting point or optimizer faces difficulties for  $n = 100$ .





MOSEK Fusion is a framework for solving

$$\begin{array}{ll} \text{minimize} & \sum_k (c^k)^T x^k \\ \text{subject to} & A^i x^k + b^i \in \mathcal{K}^i, i = 1, \dots \end{array}$$

where

$$\mathcal{K}^i$$

is

- a linear cone,
- a quadratic cone,
- a semidefinite cone,
- or the zero vector. (The linear equality case)

Fusion is available for

- C++
- Java
- Matlab
- Python
- .NET

We will use

- Python version.
- From a Jupiter notebook.







A few import Python facts:

- Python is interpreted language. `python.org`.
- Index origin is 0.
- `[[1.0, 2.0], [3.0, 4.0]]` is a 2 by 2 matrix stored rowwise.

See Jupyter notebook:

- `minball.ipynb`



Recall duality i.e. the primal

$$\begin{array}{ll} \text{minimize} & \sum (c^k)^T x^k \\ \text{subject to} & \sum_k A^k x^k = b, \\ & x^k \in \mathcal{K}^k, \quad \forall k \end{array}$$

has the dual

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & c^k - (A^k)^T y \in \mathcal{K}^k, \forall k. \end{array}$$



Reformulated problem

$$\begin{array}{ll} \text{maximize} & \hat{r} \\ \text{subject to} & \begin{pmatrix} 0 \\ p^i \end{pmatrix} - I \begin{pmatrix} \hat{r} \\ x \end{pmatrix} \in \mathcal{K}_q, \quad \forall i \end{array}$$

where  $r = -\hat{r}$  has problem has dual form.

Therefore, the dual problem is

$$\begin{aligned} & \text{minimize} && \sum_i \begin{bmatrix} 0 \\ p^i \end{bmatrix}^T y^i \\ & \text{subject to} && \sum_i y^i = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \\ & && y^i \in \mathcal{K}_q, \quad \forall i. \end{aligned}$$





See Jupyter notebook:

- `minball_primal_dual.ipynb`

## Section 2

Finance application: Portfolio optimization





An investor can invest in  $n$  stocks or assets to be held over a period of time. What is the optimal portfolio?

Now assume a stochastic model where the return of the assets is a random variable

$$r$$

with known mean

$$\mu = \mathbf{E}r$$

and covariance

$$\Sigma = \mathbf{E}(r - \mu)(r - \mu)^T.$$



Let  $x_j$  amount invested in asset  $j$ . Moreover, the expected return is:

$$\mathbf{E}y = \mu^T x$$

and variance of the return is

$$(y - \mathbf{E}y)^2 = x^T \Sigma x.$$

The investors optimization problem:

$$\begin{array}{ll} \text{maximize} & \mu^T x \\ \text{subject to} & e^T x = w + e^T x^0, \\ & x^T \Sigma x \leq \gamma^2, \\ & x \geq 0, \end{array}$$



where

- $e$  is the vector of all ones.
- $w$  investors initial wealth.
- $x^0$  investors initial portfolio.
- Objective maximize expected return.
- Constraints:
  - Budget constraint.  $(e^T x = \sum_{j=1}^n x_j)$ .
  - Risk constraint.  $\gamma$  is chosen by the investor.
  - Only buy a positive amount i.e. no short-selling.



The covariance matrix  $\Sigma$  is positive semidefinite by definition.  
Therefore,

$$\exists G : \quad \Sigma = GG^T.$$

CQ reformulation:

$$\begin{array}{ll} \text{maximize} & \mu^T x \\ \text{subject to} & e^T x = w + e^T x^0, \\ & [\gamma; G^T x] \in \mathcal{K}_q^{n+1}, \\ & x \geq 0. \end{array}$$



# Python program

## portfolio\_basic.py



```
import mosek
import sys

from mosek.fusion import *
from portfolio_data import *

def BasicMarkowitz(n,mu,GT,x0,w,gamma):
    with Model("Basic Markowitz") as M:

        # Redirect log output from the solver to stdout for debugging.
        # if uncommented.
        M.setLogHandler(sys.stdout)

        # Defines the variables (holdings). Shortselling is not allowed.
        x = M.variable("x", n, Domain.greaterThan(0.0))

        # Maximize expected return
        M.objective('obj', ObjectiveSense.Maximize, Expr.dot(mu,x))

        # The amount invested must be identical to initial wealth
        M.constraint('budget', Expr.sum(x), Domain.equalsTo(w+sum(x0)))

        # Imposes a bound on the risk
        M.constraint('risk', Expr.vstack( gamma,Expr.mul(GT,x)), Domain.inQCone())

        M.solve()

        return (M.primalObjValue(), x.level())

if __name__ == '__main__':

    (expret,x) = BasicMarkowitz(n,mu,GT,x0,w,gamma)
    print("Expected return: %e" % expret)
    print("x: "),
    print(x)
```



```
n      = 3;
w      = 1.0;
mu     = [0.1073,0.0737,0.0627]
x0     = [0.0,0.0,0.0]
gamma  = 0.04
GT     = [[ 0.166673333200005, 0.0232190712557243 , 0.0012599496030238 ],
          [ 0.0                , 0.102863378954911 , -0.00222873156550421],
          [ 0.0                , 0.0                , 0.0338148677744977 ]]
```



Running

```
python portfolio_basic.py
```



```

Optimizer - threads                : 4
Optimizer - solved problem        : the primal
Optimizer - Constraints            : 3
Optimizer - Cones                  : 1
Optimizer - Scalar variables      : 6          conic                : 4
Optimizer - Semi-definite variables: 0          scalarized             : 0
Factor    - setup time            : 0.00       dense det. time        : 0.00
Factor    - ML order time         : 0.00       GP order time         : 0.00
Factor    - nonzeros before factor : 6          after factor          : 6
Factor    - dense dim.            : 0          flops                  : 7.00e+001

ITE PFEAS  DFEAS  GFEAS  PRSTATUS  POBJ          DOBJ          MU          TIME
0   1.0e+000 1.0e+000 1.0e+000 0.00e+000 0.000000000e+000 0.000000000e+000 1.0e+000 0.00
1   1.7e-001 1.7e-001 4.4e-001 9.46e-001 1.259822223e-001 2.171837612e-001 1.7e-001 0.00
2   4.0e-002 4.0e-002 5.6e-001 1.56e+000 8.104070951e-002 1.693911786e-001 4.0e-002 0.00
3   1.4e-002 1.4e-002 2.9e-001 3.00e+000 7.268285567e-002 8.146211968e-002 1.4e-002 0.00
4   1.3e-003 1.3e-003 1.1e-001 1.43e+000 7.102726686e-002 7.178857777e-002 1.3e-003 0.00
5   1.7e-004 1.7e-004 3.9e-002 1.05e+000 7.101472221e-002 7.111329525e-002 1.7e-004 0.00
6   7.7e-006 7.7e-006 8.5e-003 1.01e+000 7.099770619e-002 7.100232290e-002 7.7e-006 0.00
7   6.0e-007 6.0e-007 2.4e-003 1.00e+000 7.099794084e-002 7.099830405e-002 6.0e-007 0.00
8   1.7e-008 1.7e-008 4.0e-004 1.00e+000 7.099799652e-002 7.099800667e-002 1.7e-008 0.00
Interior-point optimizer terminated. Time: 0.00.

```

Optimizer terminated. Time: 0.01

Expected return: 7.099800e-02

x:

[ 0.15518625 0.12515363 0.71966011]



- Question: What is the right  $\gamma$ ?
- Answer: Show the investor the optimal expected return for all  $\gamma$ 's.

I.e. solve

$$\begin{array}{ll} \text{maximize} & \mu^T x - \alpha s \\ \text{subject to} & e^T x = w + e^T x^0, \\ & [s; G^T x] \in \mathcal{K}_q^{n+1}, \\ & x \geq 0 \end{array}$$

for all  $\alpha \in [0, \infty[$ .

# Python program

## portfolio\_frontier.py



```
import mosek
import sys

from mosek.fusion import *
from portfolio_data import *

def EfficientFrontier(n,mu,GT,x0,w,alphas):
    with Model("Efficient frontier") as M:
        # Defines the variables (holdings). Shortselling is not allowed.
        x = M.variable("x", n, Domain.greaterThan(0.0)) # Portfolio variables
        s = M.variable("s", 1, Domain.unbounded()) # Risk variable

        M.constraint('budget', Expr.sum(x), Domain.equalsTo(w+sum(x0)))

        # Computes the risk
        M.constraint('risk', Expr.vstack(s,Expr.mul(GT,x)),Domain.inQCone())

    frontier = []

    mudotx = Expr.dot(mu,x) # Is reused.

    for i,alpha in enumerate(alphas):

        # Define objective as a weighted combination of return and risk
        M.objective('obj', ObjectiveSense.Maximize, Expr.sub(mudotx,Expr.mul(alpha,s)))

        M.solve()

        frontier.append((alpha,M.primalObjValue(),s.level()[0]))

    return frontier

if __name__ == '__main__':
    alphas = [x * 0.1 for x in range(0, 21)]
    frontier = EfficientFrontier(n,mu,GT,x0,w,alphas)
    print('%-14s %-14s %-14s %-14s' % ('alpha','obj','exp. ret', 'std. dev.'))
    for f in frontier:
        print("%-14.2e %-14.2e %-14.2e %-14.2e" % (f[0],f[1],f[1]+f[0]*f[2],f[2])),
```





- The whole model is not rebuild for each  $\alpha$ .
- Leads to better efficiency.



- Question: Is prices on asserts independent of trade volume.
- Answer: No. Why?

A common assumption about market impact costs are where  $m_j \geq 0$  is a constant that is estimated in some way [2][p. 452]. Therefore,

$$T_j(x_j - x_j^0) = m_j |x_j - x_j^0| \sqrt{|x_j - x_j^0|} = m_j |x_j - x_j^0|^{3/2}$$

can be seen as transaction cost.

Recall from lecture 1:

$$\{(t, z) : t \geq z^{3/2}, z \geq 0\} = \{(t, z) : (s, t, z), (z, 1/8, s) \in \mathcal{K}_r^3\}.$$

So

$$\begin{aligned} z_j &= |x_j - x_j^0|, \\ (s_j, t_j, z_j), (z_j, 1/8, s_j) &\in \mathcal{K}_r^n, \\ \sum_{j=1}^n T(x_j - x_j^0) &= \sum_{j=1}^n t_j. \end{aligned}$$

and the relaxation

$$\begin{aligned} z_j &\geq |x_j - x_j^0|, \\ (s_j, t_j, z_j), (z_j, 1/8, s_j) &\in \mathcal{K}_r^n, \\ \sum_{j=1}^n T(x_j - x_j^0) &= \sum_{j=1}^n t_j \end{aligned}$$

is good enough.

Now

$$z_j \geq |x_j - x_j^0|.$$

is the same as

$$\begin{aligned} z_j &\geq x_j - x_j^0, \\ z_j &\geq -(x_j - x_j^0). \end{aligned}$$





$$\begin{array}{ll}
 \text{maximize} & \mu^T x \\
 \text{subject to} & e^T x + m^T t = w + e^T x^0, \\
 & (\gamma, G^T x) \in \mathcal{K}_q^{n+1}, \\
 & z_j \geq x_j - x_j^0, \quad j = 1, \dots, n, \\
 & z_j \geq x_j^0 - x_j, \quad j = 1, \dots, n, \\
 & [v_j; t_j; z_j], [z_j; 1/8; v_j] \in \mathcal{K}_r^3, \quad j = 1, \dots, n, \\
 & x \geq 0.
 \end{array}$$

The revised budget constraint is

$$e^T x = w + e^T x^0 - m^T t$$

where

$$m^T t$$

is the total market impact cost that must be paid too.



# Python program

## portfolio\_marketimpact.py



```
import mosek
import numpy
import sys

from mosek.fusion import *
from portfolio_data import *

def MarkowitzWithMarketImpact(n,mu,GT,x0,w,gamma,m):
    with Model("Markowitz portfolio with market impact") as M:

        #M.setLogHandler(sys.stdout)

        # Defines the variables. No shortselling is allowed.
        x = M.variable("x", n, Domain.greaterThan(0.0))

        # Additional "helper" variables
        t = M.variable("t", n, Domain.unbounded())
        z = M.variable("z", n, Domain.unbounded())
        v = M.variable("v", n, Domain.unbounded())

        # Maximize expected return
        M.objective('obj', ObjectiveSense.Maximize, Expr.dot(mu,x))

        # Invested amount + slippage cost = initial wealth
        M.constraint('budget', Expr.add(Expr.sum(x),Expr.dot(m,t)), Domain.equalsTo(w+sum(x0)))

        # Imposes a bound on the risk
        M.constraint('risk', Expr.vstack(gamma,Expr.mul(GT,x)), Domain.inQCone())

        # z >= |x-x0|
        M.constraint('buy', Expr.sub(z,Expr.sub(x,x0)),Domain.greaterThan(0.0))
        M.constraint('sell', Expr.sub(z,Expr.sub(x0,x)),Domain.greaterThan(0.0))

        # t >= z^1.5, z >= 0.0. Needs two rotated quadratic cones to model this term
        M.constraint('ta', Expr.hstack(v,t,z),Domain.inRotatedQCone())
        M.constraint('tb', Expr.hstack(z,Expr.constTerm(n,1.0/8.0),v),Domain.inRotatedQCone())

        M.solve()

        print('Expected return: %.4e Std. deviation: %.4e Market impact cost: %.4e' % \
              (M.primalObjValue(),gamma,numpy.dot(m,t.level()))))

if __name__ == '__main__':
    m = n*[1.0e-2]
    MarkowitzWithMarketImpact(n,mu,GT,x0,w,gamma,m)
```

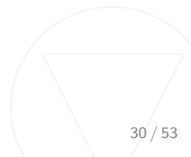


Now assume there is a cost associated with trading asset  $j$  and the cost is given by

$$T_j(\Delta x_j) = \begin{cases} 0, & \Delta x_j = 0, \\ f_j + g_j |\Delta x_j|, & \text{otherwise.} \end{cases}$$

where

$$\Delta x_j = x_j - x_j^0.$$





$$\begin{aligned} & \text{maximize} && \mu^T x \\ & \text{subject to} && e^T x + \sum_{j=1}^n (f_j y_j + g_j z_j) = w + e^T x^0, \\ & && [\gamma; G^T x] \in \mathcal{K}_q^{n+1} \\ & && z_j \geq x_j - x_j^0, \quad j = 1, \dots, n, \\ & && z_j \geq x_j^0 - x_j, \quad j = 1, \dots, n, \\ & && z_j \leq U_j y_j, \quad j = 1, \dots, n, \\ & && y_j \in \{0, 1\}, \quad j = 1, \dots, n, \\ & && x \geq 0. \end{aligned}$$

Observe:

- Is mixed-integer model.  $y_j$  models purchase or not.
- We have  $z_j \geq 0$  and hence  $y_j = 0 \Rightarrow z_j = 0 \Rightarrow x_j = x_j^0$ .



# Python program

## portfolio\_transaction.py



```
import mosek
import numpy
import sys

from mosek.fusion import *
from portfolio_data import *

def MarkowitzWithTransactionsCost(n,mu,GT,x0,w,gamma,f,g):
    # Upper bound on the traded amount
    w0 = w+sum(x0)
    u = n*[w0]

    with Model("Markowitz portfolio with transaction costs") as M:
        x = M.variable("x", n, Domain.greaterThan(0.0))
        z = M.variable("z", n, Domain.unbounded())
        y = M.variable("y", n, Domain.binary())

        # Maximize expected return
        M.objective('obj', ObjectiveSense.Maximize, Expr.dot(mu,x))

        # Invest amount + transactions costs = initial wealth
        M.constraint('budget', Expr.add([ Expr.sum(x), Expr.dot(f,y),Expr.dot(g,z)] ), Domain.equalsTo(w0))

        # Imposes a bound on the risk
        M.constraint('risk', Expr.vstack( gamma,Expr.mul(GT,x)), Domain.inQCone())

        # z >= |x-x0|
        M.constraint('buy', Expr.sub(z,Expr.sub(x,x0)),Domain.greaterThan(0.0))
        M.constraint('sell', Expr.sub(z,Expr.sub(x0,x)),Domain.greaterThan(0.0))

        # Constraints for turning y off and on. z-diag(u)*y<=0 i.e. z_j <= u_j*y_j
        M.constraint('y_on_off', Expr.sub(z,Expr.mulElm(u,y)), Domain.lessThan(0.0))

        # Integer optimization problems can be very hard to solve so limiting the
        # maximum amount of time is a valuable safe guard
        M.setSolverParam('mioMaxTime', 180.0)
        M.solve()

    print('Expected return: %.4e Std. deviation: %.4e Transactions cost: %.4e' % \
          (numpy.dot(mu,x.level()),gamma,numpy.dot(f,y.level())+numpy.dot(g,z.level())))

if __name__ == '__main__':
    f = n*[0.01]
    g = n*[0.001]
    MarkowitzWithTransactionsCost(n,mu,GT,x0,w,gamma,f,g)
```

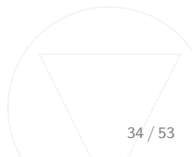
## Section 3

### Robust optimization





- A very important application of conic quadratic optimization is robust optimization.
- Robust optimization assumes the problem data e.g.  $A$  is not known exactly.
- Tries to compute a robust solution.





Consider the toy linear optimization problem:

*A company produces two kinds of drugs, DrugI and DrugII, containing a specific active agent A, which is extracted from raw materials which should be purchased on the market. The drug production data are as follows:*

<i>Drug</i>	<i>Selling price, \$ per 1000 packs</i>	<i>Content of agent A, g per 1000 packs</i>
<i>DrugI</i>	6,200	0.500
<i>DrugII</i>	6,900	0.600

<i>Drug</i>	<i>Production expenses per 1000 packs</i>		
	<i>manpower, hours</i>	<i>equipment, hours</i>	<i>operational costs, \$</i>
<i>DrugI</i>	90.0	40.0	700
<i>DrugII</i>	100.0	50.0	800

*There are two kinds of raw materials, RawI and RawII, which can be used as sources of the active agent. The related data are as follows:*

<i>Raw material</i>	<i>Purchasing price, \$ per kg</i>	<i>Content of agent A, g per kg</i>
<i>RawI</i>	<i>100.00</i>	<i>0.01</i>
<i>RawII</i>	<i>199.90</i>	<i>0.02</i>

*Finally, the per month resources dedicated to producing the drugs are as follows:*

<i>Budget, \$</i>	<i>Manpower, hours</i>	<i>Equipment, hours</i>	<i>Capacity of raw materials storage, kg</i>
<i>100,000</i>	<i>2,000</i>	<i>800</i>	<i>1,000</i>



*The problem is to find the production plan which maximizes the profit of the company.*

The problem can be immediately posed as the following linear programming program:

maximize

$$\begin{aligned} & - (100 \cdot \text{RawI} + 199.90 \cdot \text{RawII} + 700 \cdot \text{DrugI} + 800 \cdot \text{DrugII}) \text{ (cost)} \\ & + (6200 \cdot \text{DrugI} + 6900 \cdot \text{DrugII}) \text{ (income)} \end{aligned}$$



subject to

$$\begin{aligned}0.01 \cdot \text{RawI} + 0.02 \cdot \text{RawII} - 0.500 \cdot \text{DrugI} - 0.600 \cdot \text{DrugII} &\geq 0 \\ \text{RawI} + \text{RawII} &\leq 1000 \\ 90.0 \cdot \text{DrugI} + 100.0 \cdot \text{DrugII} &\leq 2000 \\ 40.0 \cdot \text{DrugI} + 50.0 \cdot \text{DrugII} &\leq 800 \\ 100.0 \cdot \text{RawI} + 199.90 \cdot \text{RawII} + 700 \cdot \text{DrugI} + 800 \cdot \text{DrugII} &\leq 100000 \\ \text{RawI}, \text{RawII}, \text{DrugI}, \text{DrugII} &\geq 0\end{aligned}$$

Explanation of constraints:

- balance of active agent
- storage restriction
- manpower restriction
- equipment restriction
- budget restriction



The optimal solution:

\*\*\* Optimal value: 8819.658

\*\*\* Optimal solution:

RawI: 0.000

RawII: 438.789

DrugI: 17.552

DrugII: 0.000

Comments

- The company makes a profit 8819 on a budget of 100,000 i.e. 8.8%.
- The balance constraint is active as could have been guessed.
- Is there anything wrong with the solution?





- Is it likely that RawI contains exactly 0.01 g per kg of agent A?
- Reasonable assumption: The contents of the active agent  $a_I$  in RawI and  $a_{II}$  in RawII in the raw materials are random variables.
- Assume instead:

$$a_I = \begin{cases} 0.0095, & p = 0.5 \\ 0.0105, & p = 0.5 \end{cases}$$

and

$$a_{II} = \begin{cases} 0.0196, & p = 0.5 \\ 0.0204, & p = 0.5 \end{cases}$$

where  $p$  is a probability.



- The optimal solution says buy 438.8 kg of RawII and produce 17552 packs of drug DrugII.
- That will be an infeasible plan with probability of 0.5.
- In that case we can only produce 17201 packs leading to a profit of 6889. A 21% decrease in the profit.
- Assuming we reduce output in the bad case and keep it constant in the good case, then the expected profit is 7854.
- Conclusion: We see that in our toy example *pretty small (and unavoidable in reality) perturbations of the data may make the optimal solution infeasible, and a straightforward adjustment to the actual solution values may heavily affect the solution quality.*





The standard linear optimization problem:

$$\begin{array}{ll}\min & c^T x \\ \text{subject to} & a_i^T x \leq b_i, \quad \forall i.\end{array}$$

Assume:

$$a_{i\cdot}^T \in \mathcal{E}_i := \{z : z = \bar{a}_{i\cdot}^T + H^i y, \quad \|y\| \leq 1\},$$

where

$$H^i \in \mathbb{R}^{n \times l_i}.$$

Observe:

- $\mathcal{E}_i$  is an ellipsoid.

For a fixed  $x$  we have

$$\begin{aligned}\max_{a_{i:} \in \mathcal{E}_i} a_{i:}x &= \max_{\|y\| \leq 1} x^T (\bar{a}_{i:}^T + H^i y) \\ &= \bar{a}_{i:}x + \max_{\|y\| \leq 1} x^T H^i y \\ &= \bar{a}_{i:}x + \left\| (H^i)^T x \right\|.\end{aligned}$$

(Why does the last equality holds?)



Therefore,

$$\begin{array}{ll} \min & c^T x \\ \text{subject to} & a_{i:} x \leq b_i, \quad a_{i:}^T \in \mathcal{E}_i, \quad \forall i \end{array}$$

and

$$\begin{array}{ll} \min & c^T x \\ \text{subject to} & \bar{a}_{i:} x + \|(H^i)^T x\| \leq b_i, \quad \forall i \end{array}$$

are equivalent.





Consider

$$\begin{array}{ll}\min & -x_1 \\ \text{subject to} & a_1x_1 + a_2x_2 \leq 1, \\ & x_1, x_2 \geq 0.\end{array}$$

Assuming that  $a_1 = a_2 = 1$  then the optimal solution is

$$x_1 = 1 \text{ and } x_2 = 0.$$

Notes

- The optimal solution is on the boundary (holds generically).
- The optimal solution is infeasible if  $a_1 > 1$  and therefore not robust.

Next consider the robust version

$$\begin{array}{ll}\min & -x_1 \\ \text{subject to} & a_1x_1 + a_2x_2 \leq 1, \quad \forall (a_1, a_2) \in \mathcal{E} \\ & x_1, x_2 \geq 0.\end{array}$$

where

$$\mathcal{E} := \{(a_1, a_2) : (a_1, a_2) = (1, 1) + \theta y, \quad \|y\| \leq 1\}$$

and  $\theta$  is a fixed nonnegative number. Equivalent robust version

$$\begin{array}{ll}\min & -x_1 \\ \text{subject to} & x_1 + x_2 + \theta \|(x_1, x_2)\| \leq 1, \\ & x_1, x_2 \geq 0.\end{array}$$



## Notes

- The optimal solution is  $(x_1, x_2) = \left(\frac{1}{1+\theta}, 0\right)$ .
- The optimal solution is in the interior of

$$\{(x_1, x_2) : x_1 + x_2 \leq 1\}$$

for  $\theta > 0$ .

- The robust version push the optimal solution into the interior of the original feasible region.
- Therefore, the optimal solution is still feasible even for small changes in the problem data.
- Clearly, there is a **tradeoff** between “robustness” and the objective value.







Assumptions:

- $a_{i:}$  are independent Gaussian random vectors i.e.

$$a_{i:} \sim N(\bar{a}_{i:}, \Sigma_i).$$

Problem:

$$\begin{array}{ll} \min & c^T x \\ \text{subject to} & \text{Prob}(a_{i:}x \leq b_i) \geq p, \forall i. \end{array}$$

Now

$$\text{Prob}(a_{i:}x \leq b_i) \geq p$$

is equivalent to

$$\text{Prob}\left(\frac{a_{i:}x - \mu}{\sigma_i} \leq \frac{b_i - \mu}{\sigma_i}\right) \geq p$$

where

$$\mu = \bar{a}_{:i}x \text{ and } \sigma_i = \left\| \Sigma_i^{\frac{1}{2}} x \right\|.$$

Clearly

$$\frac{a_{i:x} - \bar{a}_{:i}x}{\left\| \Sigma_i^{\frac{1}{2}} x \right\|} \sim N(0, 1).$$

Hence,

$$\frac{b_i - \bar{a}_{:i}x}{\left\| \Sigma_i^{\frac{1}{2}} x \right\|} \geq \Phi^{-1}(p)$$

where

$$\Phi(z) := \frac{1}{2\pi} \int_{-\infty}^z e^{-t^2/2} dt.$$

Thus

$$b_i \geq \bar{a}_{:i}x + \Phi^{-1}(p) \left\| \Sigma_i^{\frac{1}{2}} x \right\|.$$



Equivalent problem:

$$\begin{array}{ll} \min & c^T x \\ \text{subject to} & \bar{a}_i x + \Phi^{-1}(p) \left\| \Sigma_i^{1/2} x \right\| \leq b_i, \forall i. \end{array}$$

Notes:

- For  $p \geq 0.5$  then  $\Phi^{-1}(p) \geq 0$ .
- Hence, is a conic quadratic problem for  $p \geq 0.5$ .
- Is called *chance constrained* optimization.



## Section 4

### Summary





- Introduced MOSEK Fusion for Python.
- Implemented minimum circle model in Fusion.
- Introduced an portfolio optimization problem of an investor and its implementation.
- Introduced robust optimization.



- [1] Neculai Andrei.  
*Nonlinear optimization applications using GAMS technology.*  
Springer, New York, 2013.
- [2] Richard C. Grinold and Ronald N. Kahn.  
*Active portfolio management.*  
McGraw-Hill, New York, 2 edition, 2000.