

- If $f(x)$ - strictly or strongly (different cases 😊) convex function, then S^* contains only one single point $S^* = x^*$.

Optimization with equality conditions

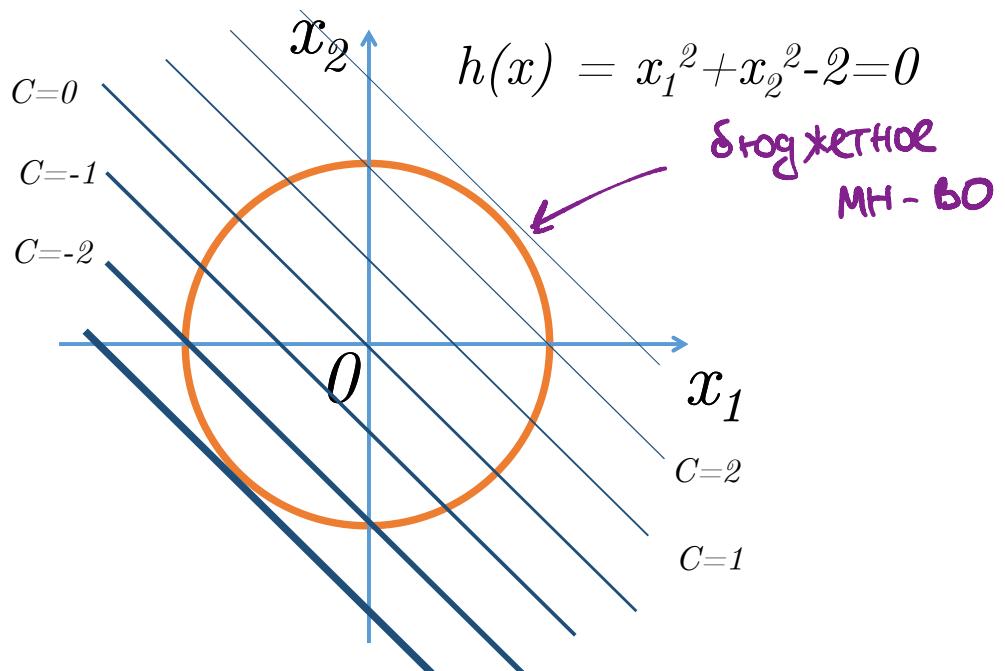
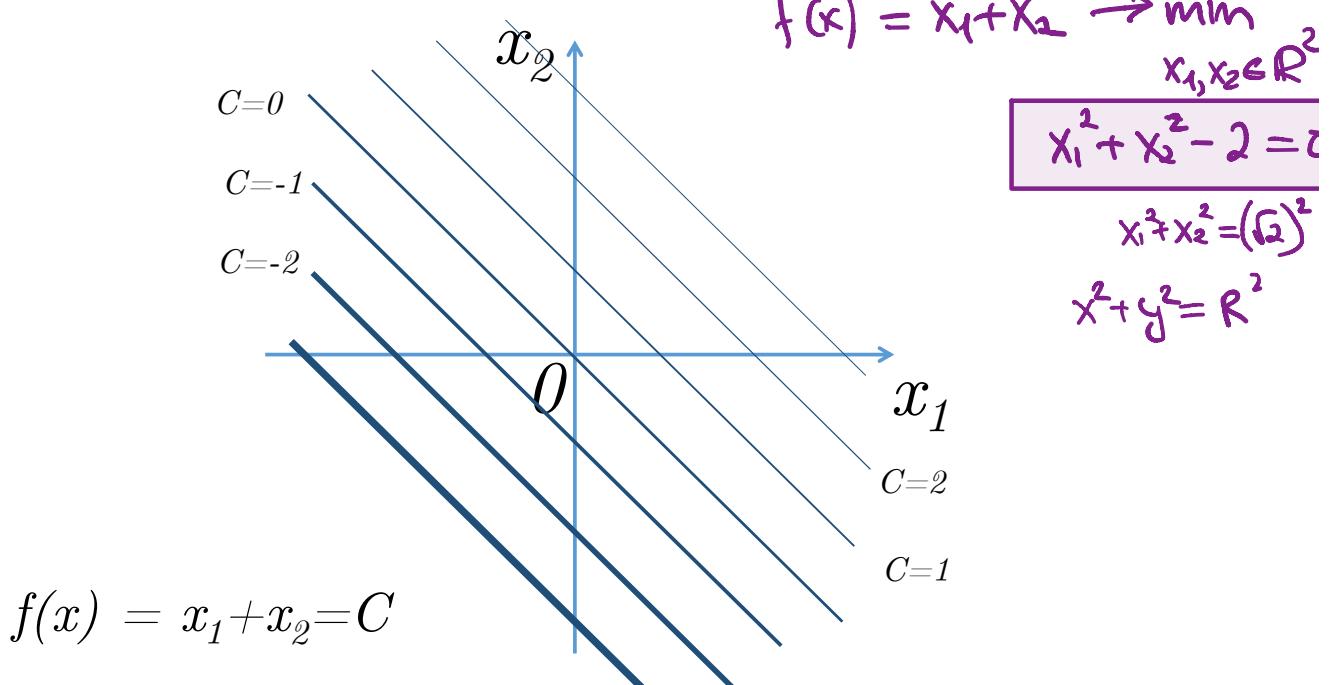
Intuition

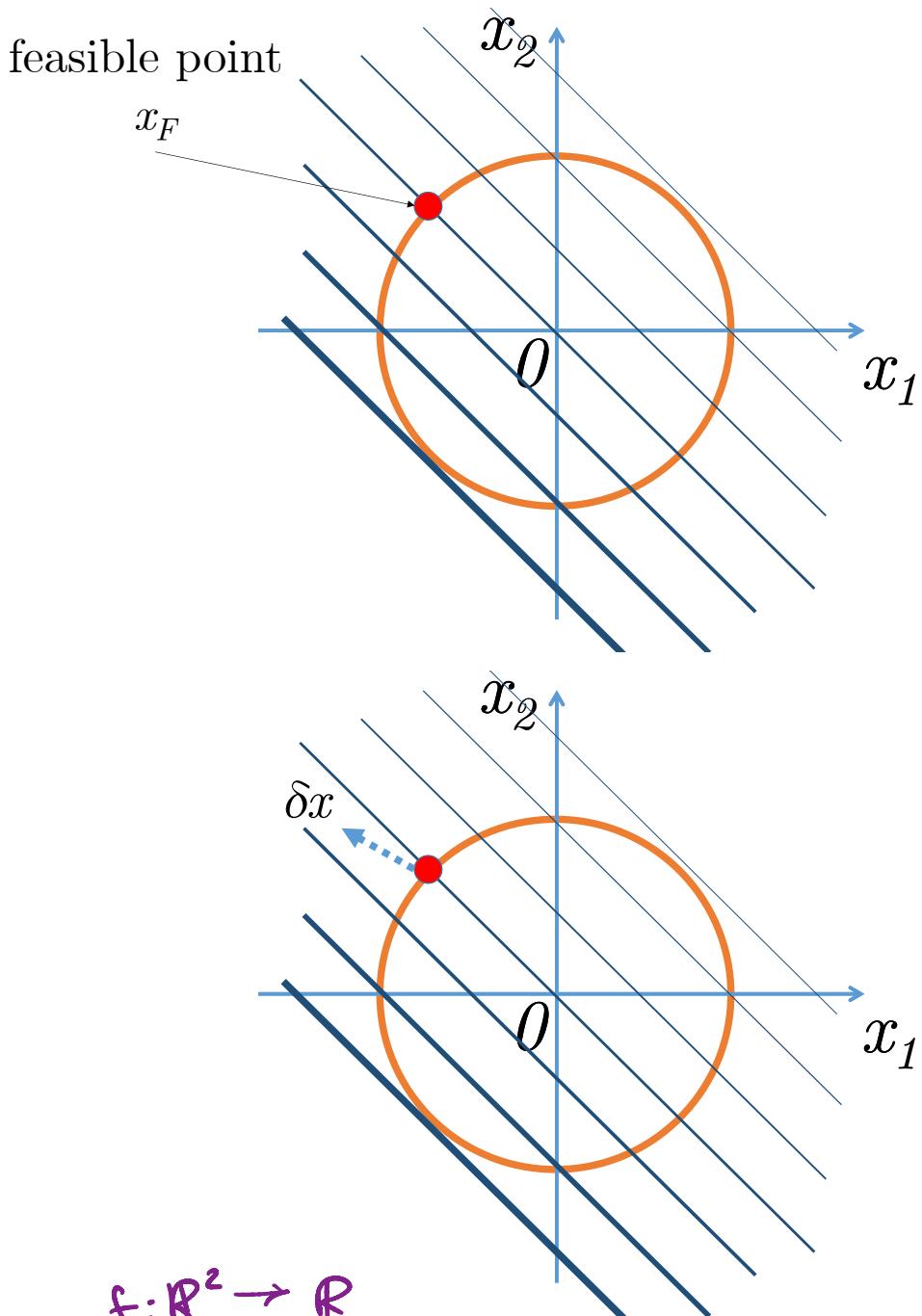
Things are pretty simple and intuitive in unconstrained problem. In this section we will add one equality constraint, i.e.

$$\begin{aligned} f(x) &\rightarrow \min_{x \in \mathbb{R}^n} \\ \text{s.t. } h(x) &= 0 \end{aligned}$$

Условная
задача
оптимизации

We will try to illustrate approach to solve this problem through the simple example with $f(x) = x_1 + x_2$ and $h(x) = x_1^2 + x_2^2 - 2$





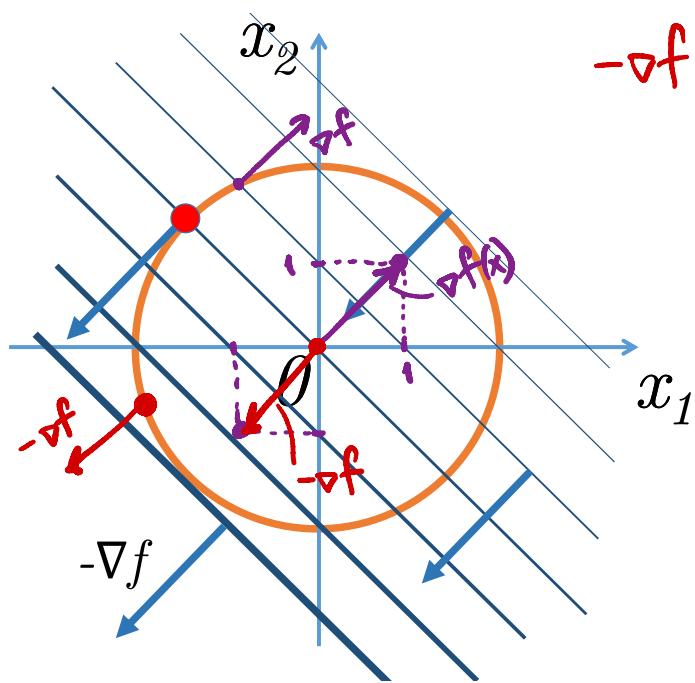
$$f: \mathbb{R}^2 \rightarrow \mathbb{R}$$

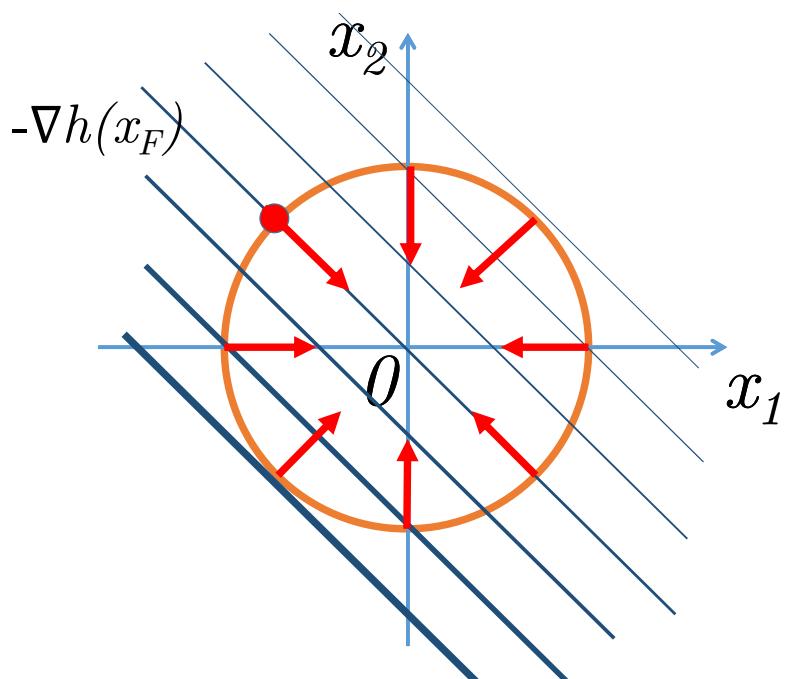
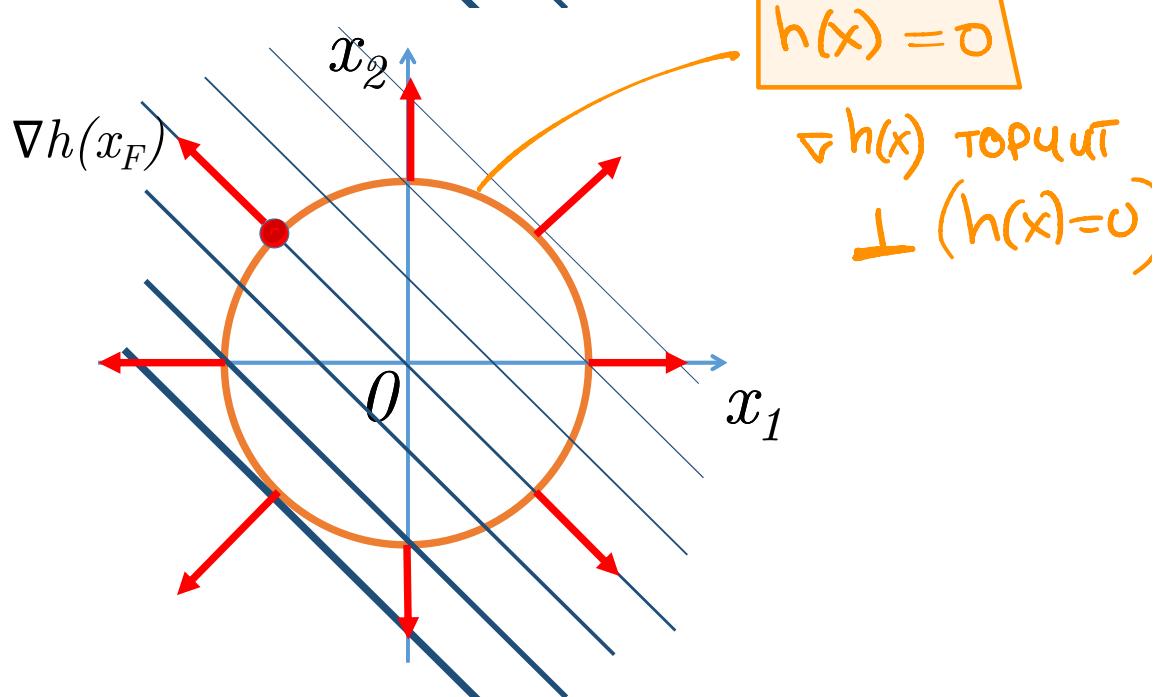
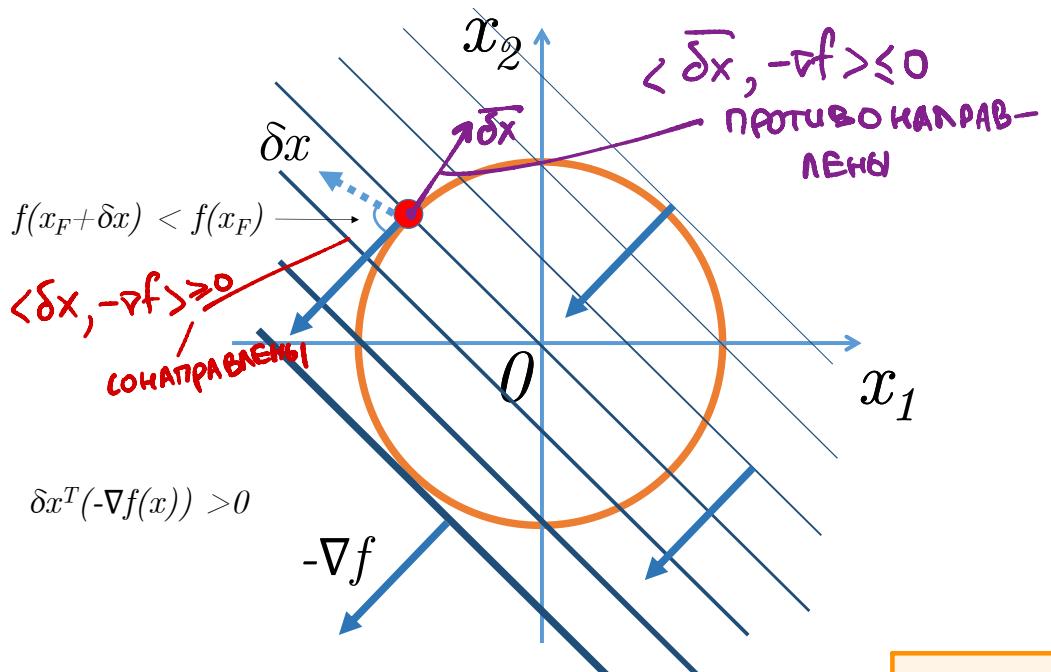
$$f(x) = x_1 + x_2$$

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{pmatrix} \in \mathbb{R}^2$$

$$\nabla f = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$-\nabla f = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$$





Generally: in order to move from x_F along the budget set towards decreasing the function, we need to guarantee two conditions:

$$\langle \delta x, \nabla h(x_F) \rangle = 0$$

$$\langle \delta x, -\nabla f(x_F) \rangle > 0$$

— НЕ ВОДОГУМ УЗ
БЮДЖЕТНОГО МН-ВА

уменьшающие
функции

Let's assume, that in the process of such a movement we have come to the point where

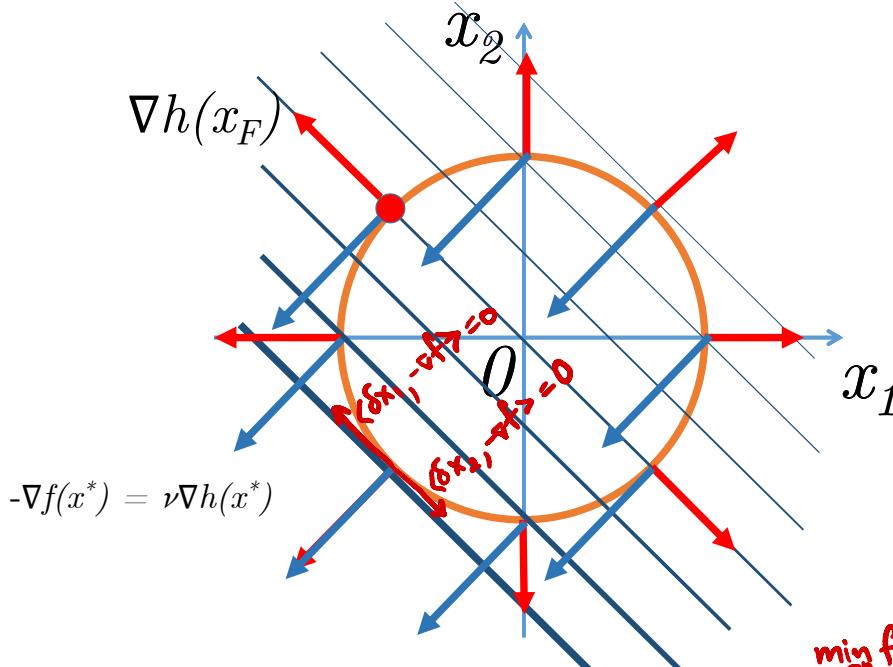
$$-\nabla f(x) = \nu \nabla h(x)$$

$$\langle \delta x, -\nabla f(x) \rangle = \langle \delta x, \nu \nabla h(x) \rangle = 0$$

— НЮ
 $\nu \in \mathbb{R}$

условие
остановы

Then we came to the point of the budget set, moving from which it will not be possible to reduce our function. This is the local minimum in the constrained problem :)



So let's define a Lagrange function (just for our convenience):

функция
Лагранжа

$$L(x, \nu) = f(x) + \nu h(x)$$

$$\begin{cases} \min_{x \in \mathbb{R}^n} f(x) \\ h(x) = 0 \end{cases}$$

недх.
услуг

Then the point x^* be the local minimum of the problem described above, if and only if:

Necessary conditions

$$\nabla_x L(x^*, \nu^*) = 0 \text{ that's written above}$$

$$\nabla_\nu L(x^*, \nu^*) = 0 \text{ budget constraint}$$

Sufficient conditions

$$\langle y, \nabla_{xx}^2 L(x^*, \nu^*) y \rangle \geq 0,$$

$$\forall y \neq 0 \in \mathbb{R}^n : \nabla h(x^*)^\top y = 0$$

$$\begin{cases} \nabla_x L = 0 \\ \nabla_\nu L = 0 \end{cases}$$

$$\nabla_x L = \nabla f(x) + \nu \cdot \nabla h(x) = 0$$

$$-\nabla f(x) = \nu \cdot \nabla h(x)$$

We should notice that $L(x^*, \nu^*) = f(x^*)$.

General formulation

$$f(x) \rightarrow \min_{x \in \mathbb{R}^n}$$

s.t. $h_i(x) = 0, i = 1, \dots, p$

$$\begin{aligned} L(x, \nu) &= f(x) + \sum_{i=1}^m \nu_i h_i(x) \\ &= f(x) + \nu^\top h(x) \end{aligned}$$

(ECP)

$\nu \in \mathbb{R}^m$
 $h \in \mathbb{R}^m$

Solution

Пример:

$$A \cdot x = b$$

$A \in \mathbb{R}^{m \times n}$ хотим
 $x \in \mathbb{R}^n$ решить
 $b \in \mathbb{R}^m$ систему лин.

$$m < n$$

ур-ши меньше,
чем неизвестных
недоопределённые
системы

$$\begin{cases} x + y = 3 \\ \dots \end{cases}$$

бесконечно
много
решений

выбрать из
них решение
с минимальной
нормой:

$$\frac{1}{2} x^T x \rightarrow \min_{x \in \mathbb{R}^n}$$

$$Ax = b$$

$$f(x) = \frac{1}{2} x^T x$$

$$h(x) = Ax - b$$

$$\textcircled{1} \quad \nabla_x L = 0$$

$$\nabla_j L = 0$$

$$\nabla_b L$$

$$m = n$$

$$x^* = A^{-1} b$$

1 решение

$$m > n$$

ур-ши больше,
чем неизвестных
 $\begin{cases} x = 5 \\ x = 7 \end{cases}$

\emptyset

НЕТ решений
всего тонкого
решения выбирает
ся наименее плохое:

$$\frac{1}{2} \|Ax - b\|_2^2 \rightarrow \min_{x \in \mathbb{R}^n}$$

$$x^* = (A^T A)^{-1} A^T b$$

$$d\left(\frac{1}{2} \langle x, x \rangle\right) = \frac{1}{2} \langle dx, x \rangle + \frac{1}{2} \langle x, dx \rangle = \langle x, dx \rangle$$

Варианты функционала Лагранжа:

$$L(x, \lambda) = f(x) + \lambda^T h(x) =$$

$$= \frac{1}{2} x^T x + \lambda^T (Ax - b)$$

$$dL = \frac{1}{2} \cdot \langle 2x, dx \rangle + \langle \lambda, Adx \rangle$$

$$\Rightarrow \boxed{\nabla_x L = x + A^T \lambda = 0}$$

$$L(x, \gamma) = \frac{1}{2} x^T x + \gamma^T (Ax - b)$$

$x \rightarrow z$
 $\gamma \rightarrow x$

$$\bar{\nabla}_{\gamma} L = \frac{1}{2} z^T z + x^T (Az - b)$$

$$dL = 0 + \langle Az - b, dx \rangle$$

$$dL = \langle Ax - b, d\gamma \rangle \Rightarrow \nabla_{\gamma} L = Ax - b = 0$$

$$\begin{cases} x + A^T \gamma = 0 \\ Ax - b = 0 \end{cases} \quad \begin{cases} x = -A^T \gamma \\ A(-A^T \gamma) - b = 0 \end{cases} \quad \begin{cases} x = -A^T \gamma \\ A \cdot A^T \gamma = -b \end{cases}$$

$| \quad (AA^T)^{-1}$

$$\begin{cases} x = -A^T \gamma \\ \gamma = -(AA^T)^{-1} \cdot b \end{cases} \quad \begin{cases} x^* = A^T (AA^T)^{-1} \cdot b \\ \gamma^* = -(AA^T)^{-1} \cdot b \end{cases}$$

Ответ: $x^* = A^T (AA^T)^{-1} \cdot b$

$$x^* = A^T b$$

A^T - не всегда обратимая

$$A^T \begin{cases} \xrightarrow{m < n} (AA^T)^{-1} \\ \xrightarrow{m = n} A^{-1} \\ \xrightarrow{m > n} (A^T A)^{-1} A^T \end{cases}$$

$$L(x, \nu) = f(x) + \sum_{i=1}^p \nu_i h_i(x) = f(x) + \nu^\top h(x)$$

Let $f(x)$ and $h_i(x)$ be twice differentiable at the point x^* and continuously differentiable in some neighborhood x^* . The local minimum conditions for $x \in \mathbb{R}^n, \nu \in \mathbb{R}^m$ are written as

ECP: Necessary conditions

$$\nabla_x L(x^*, \nu^*) = 0$$

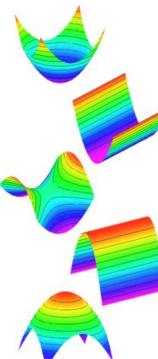
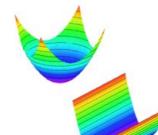
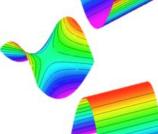
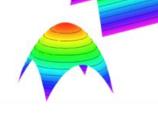
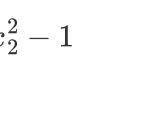
$$\nabla_\nu L(x^*, \nu^*) = 0$$

ECP: Sufficient conditions

$$\langle y, \nabla_{xx}^2 L(x^*, \nu^*) y \rangle > 0,$$

$$\forall y \neq 0 \in \mathbb{R}^n : \nabla h_i(x^*)^\top y = 0$$

Depending on the behavior of the Hessian, the critical points can have a different character.

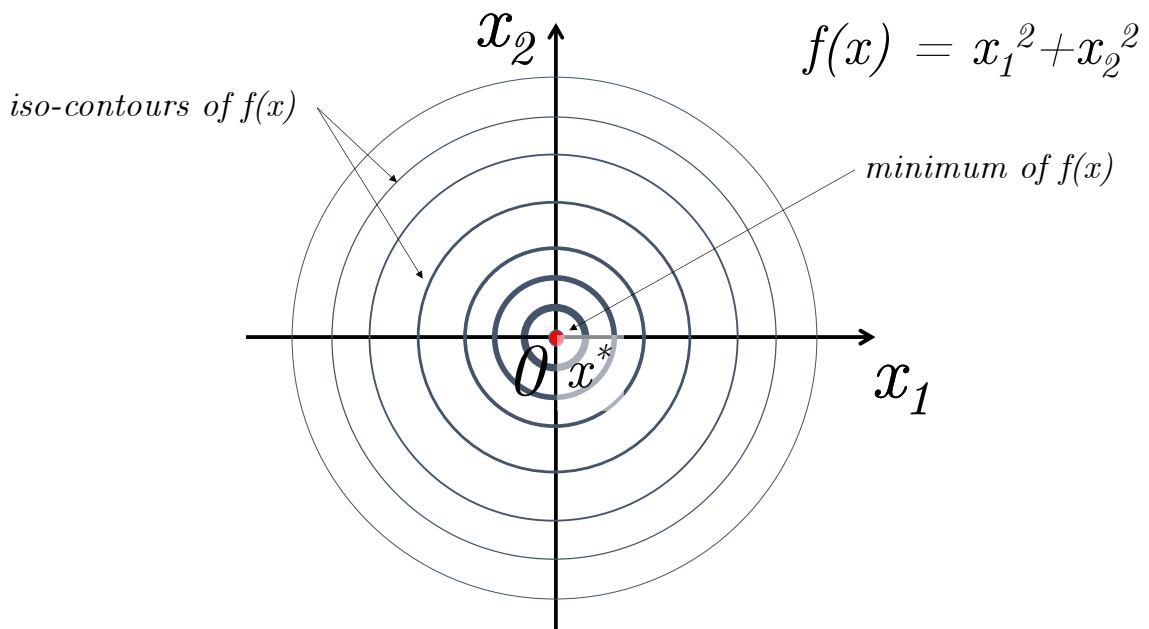
$y^\top Hy \quad \lambda_i$	Definiteness H	Nature x^*	
> 0	Positive d.	Minimum	
≥ 0	Positive semi-d.	Valley	
$\neq 0$	Indefinite	Saddlepoint	
≤ 0	Negative semi-d.	Ridge	
< 0	Negative d.	Maximum	

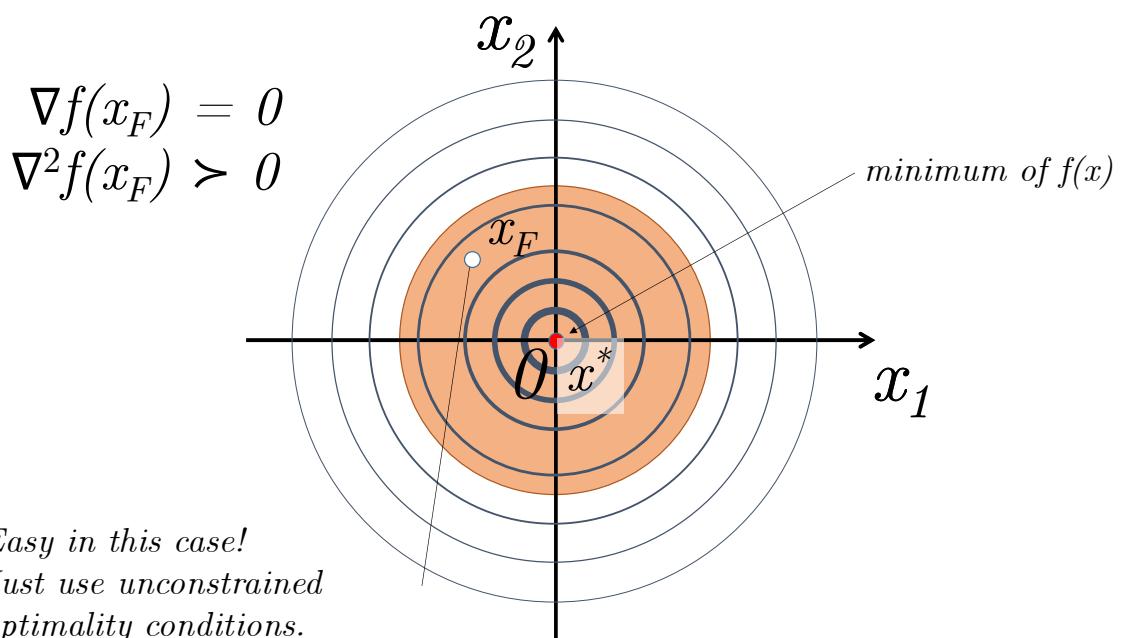
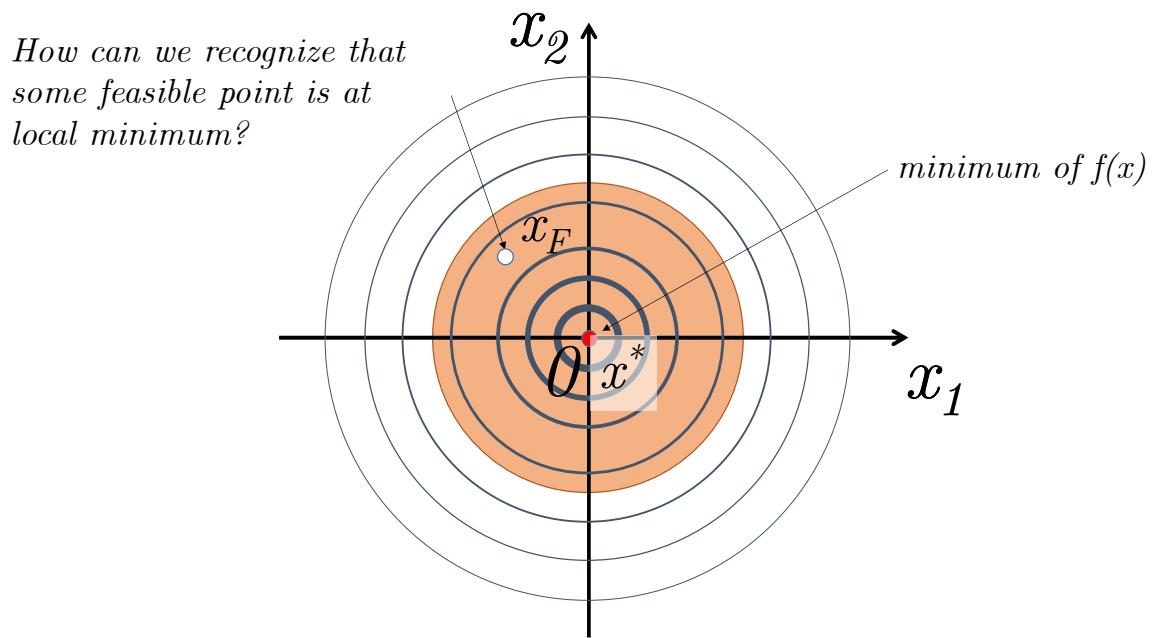
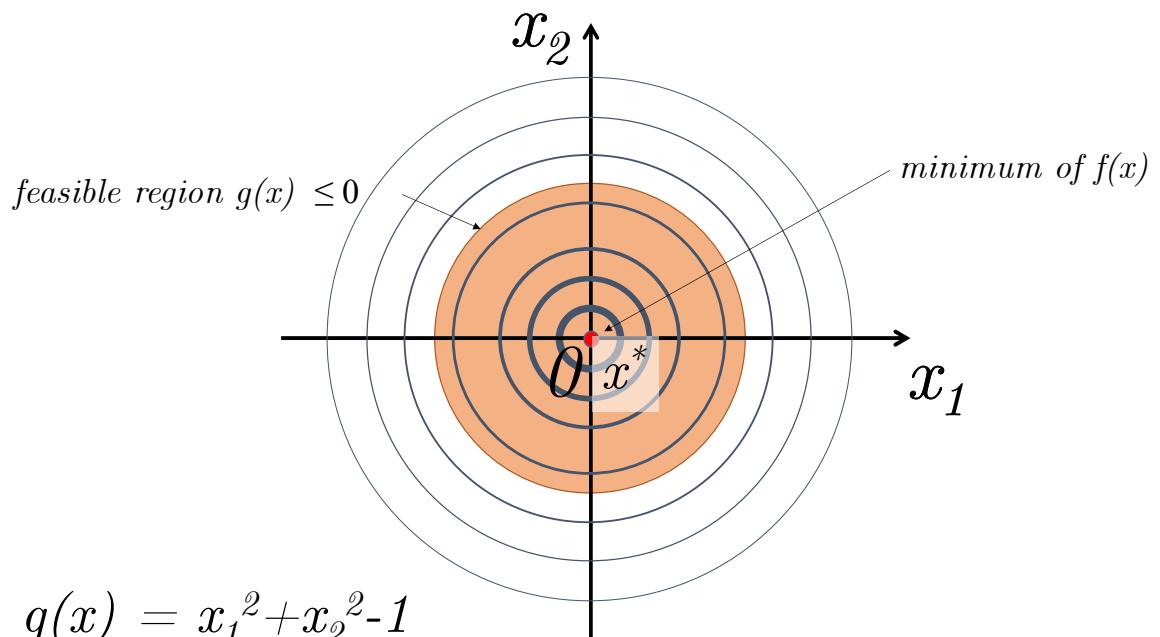
Optimization with inequality conditions

Example

$$f(x) = x_1^2 + x_2^2 \quad g(x) = x_1^2 + x_2^2 - 1$$

$$\begin{aligned} f(x) &\rightarrow \min_{x \in \mathbb{R}^n} \\ \text{s.t. } g(x) &\leq 0 \end{aligned}$$

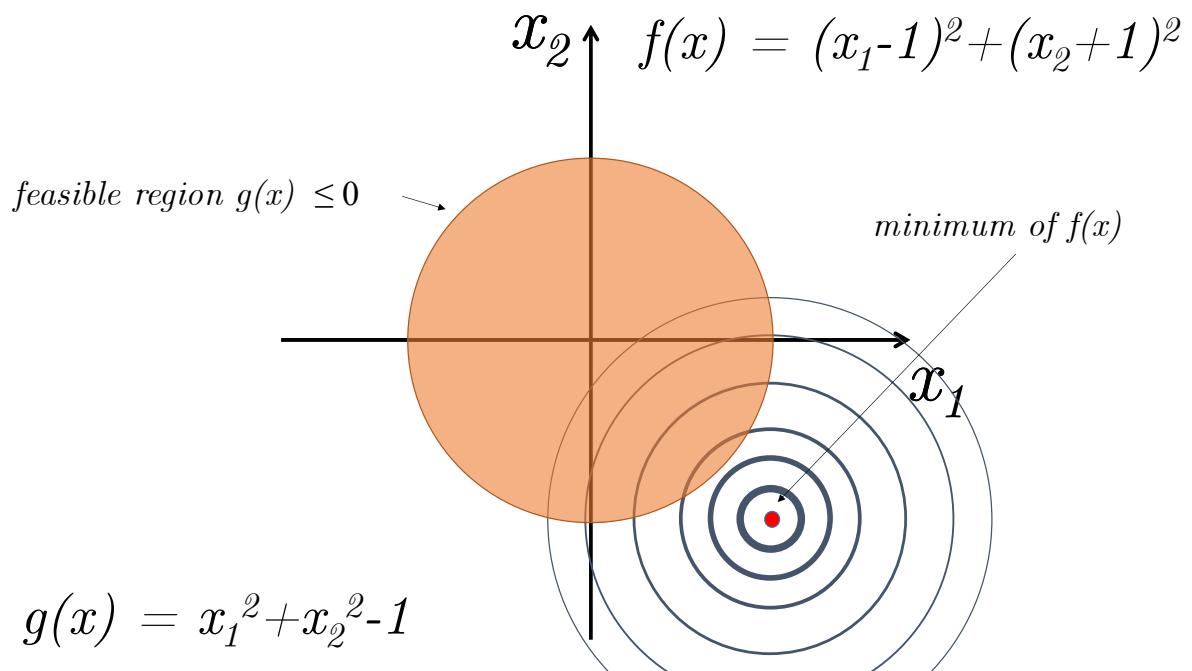
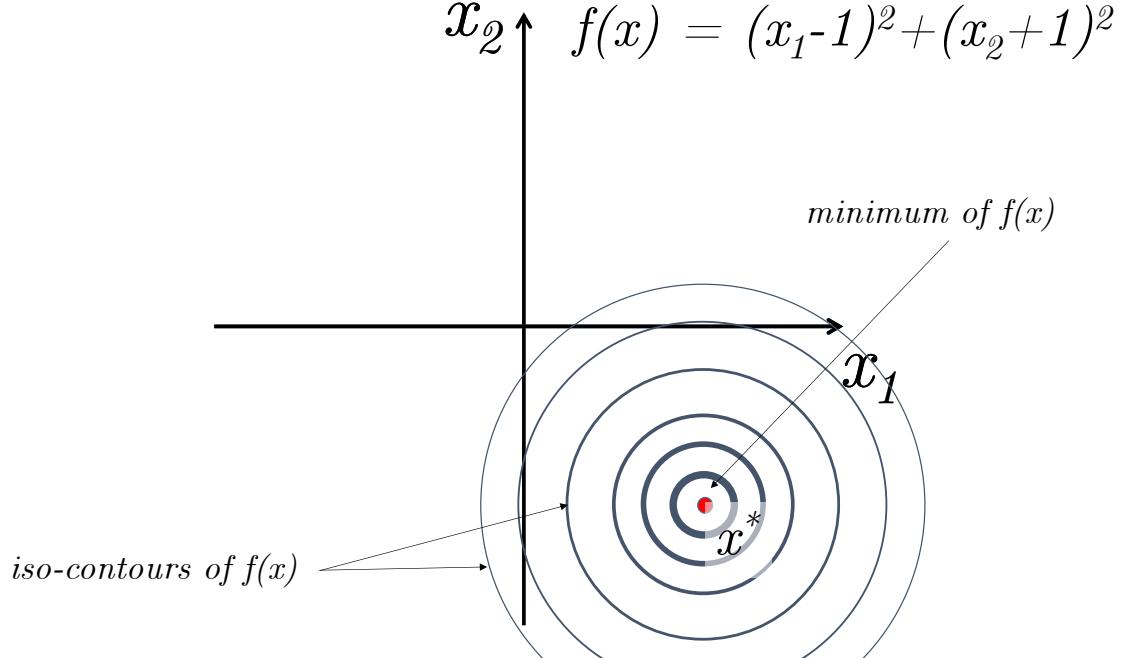




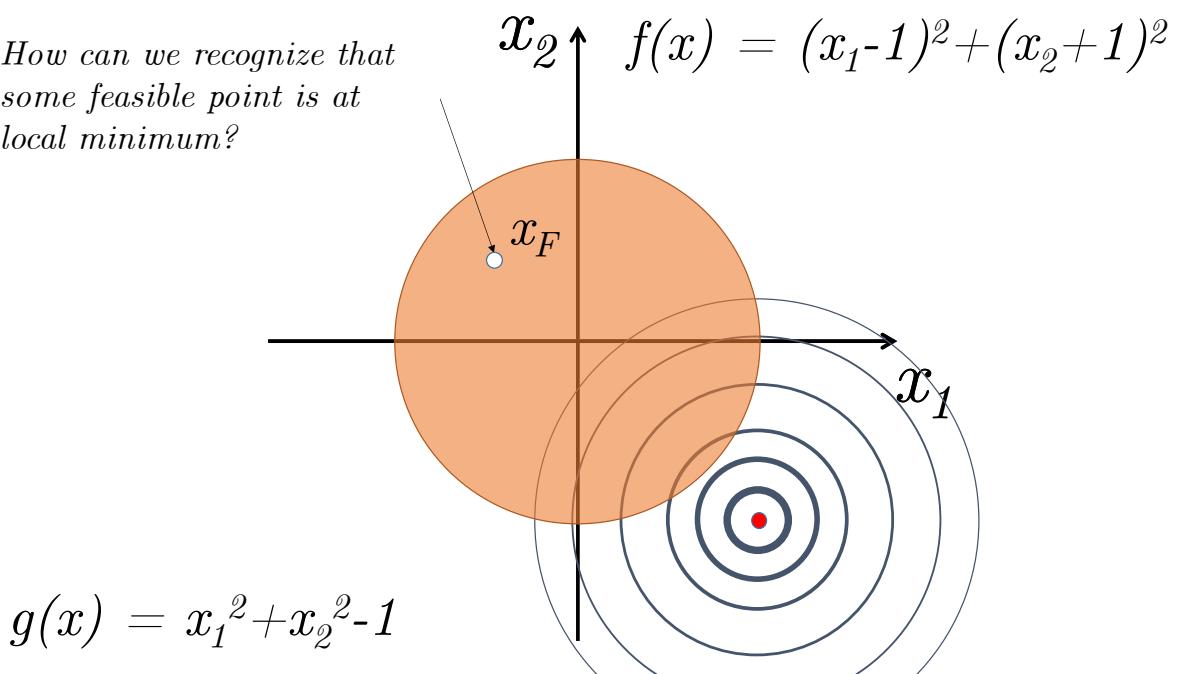
Thus, if the constraints of the type of inequalities are inactive in the constrained problem, then don't worry and write out the solution to the unconstrained problem. However, this is not the whole story 😊. Consider the second childish example

$$f(x) = (x_1 - 1)^2 + (x_2 + 1)^2 \quad g(x) = x_1^2 + x_2^2 - 1$$

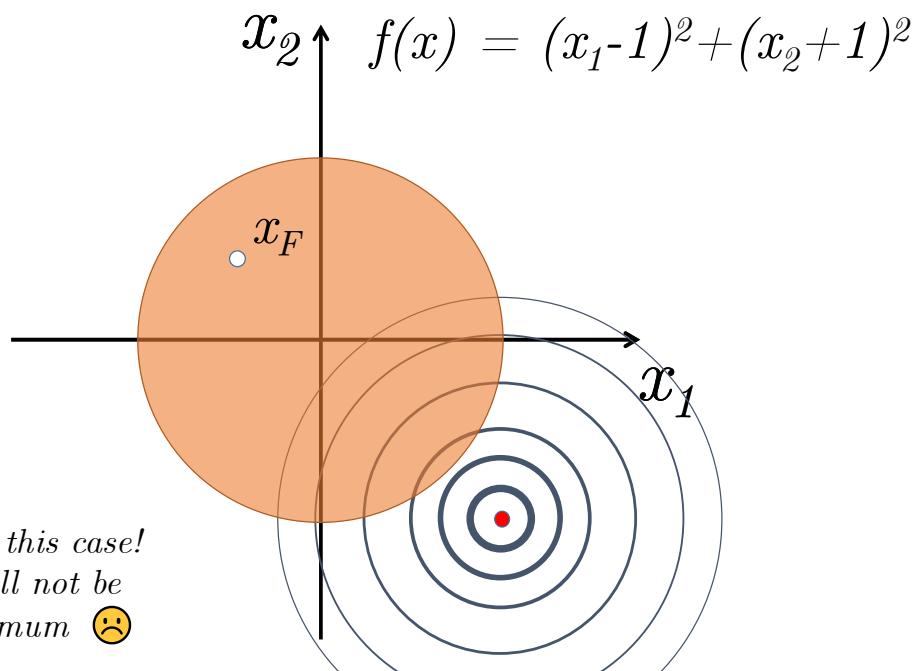
$$\begin{aligned} f(x) &\rightarrow \min_{x \in \mathbb{R}^n} \\ \text{s.t. } g(x) &\leq 0 \end{aligned}$$



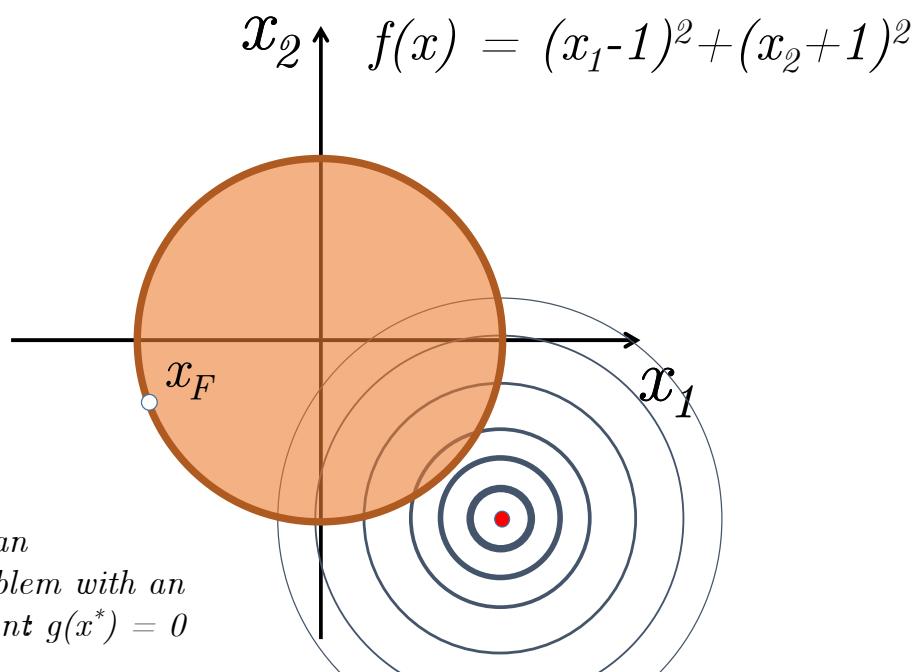
How can we recognize that some feasible point is at local minimum?

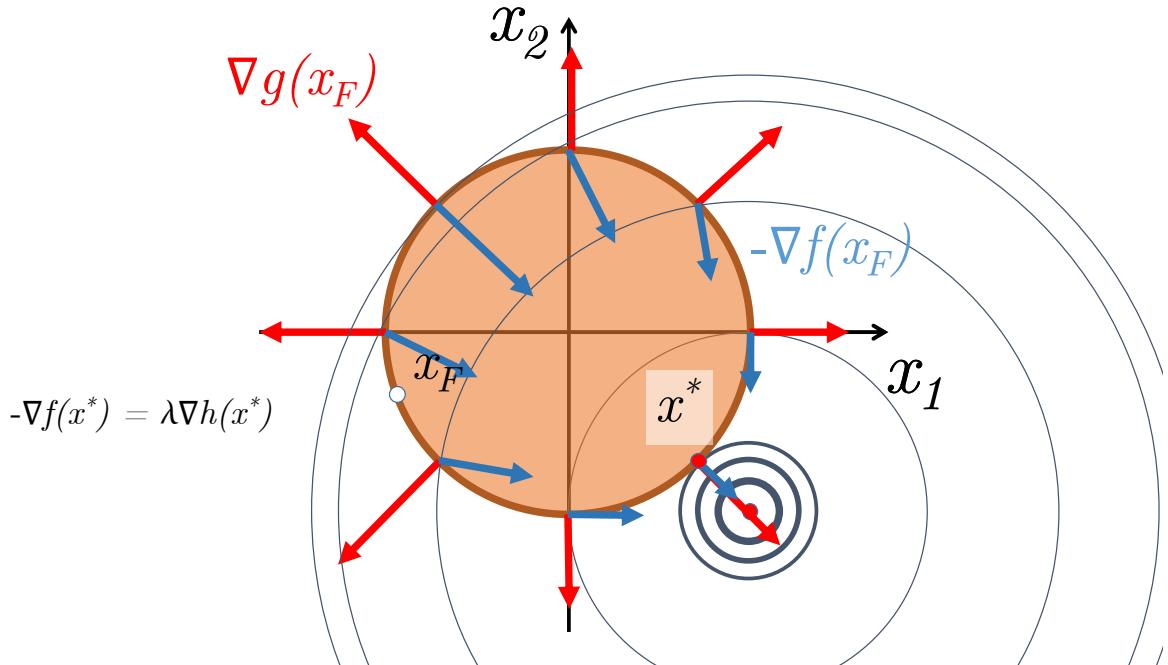


*Not very easy in this case!
Even gradient will not be zero at local optimum 😞*

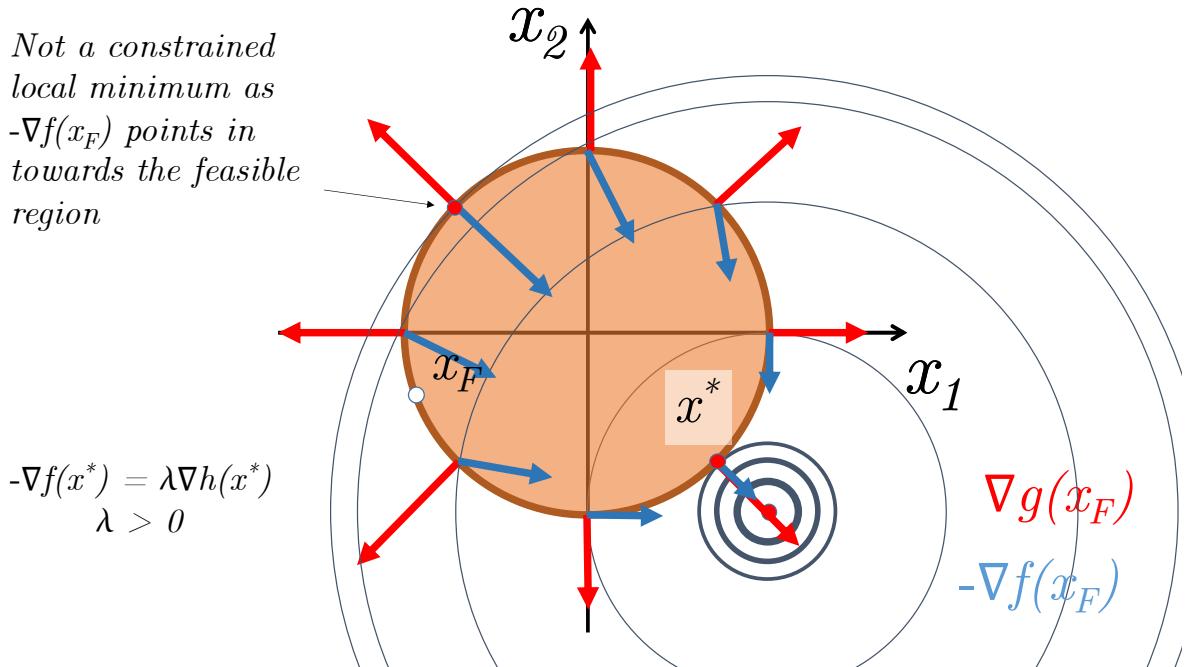


Effectively have an optimization problem with an equality constraint $g(x^) = 0$*





Not a constrained local minimum as $-\nabla f(x_F)$ points in towards the feasible region



So, we have a problem:

$$\begin{aligned} f(x) &\rightarrow \min_{x \in \mathbb{R}^n} \\ \text{s.t. } g(x) &\leq 0 \end{aligned}$$

Two possible cases:

$g(x) \leq 0$ is inactive . $g(x^*) < 0$	$g(x) \leq 0$ is active . $g(x^*) = 0$
$g(x^*) < 0$ $\nabla f(x^*) = 0$ $\nabla^2 f(x^*) > 0$	Necessary conditions $g(x^*) = 0$ $-\nabla f(x^*) = \lambda \nabla g(x^*), \lambda > 0$ Sufficient conditions $\langle y, \nabla^2_{xx} L(x^*, \lambda^*) y \rangle > 0,$ $\forall y \neq 0 \in \mathbb{R}^n : \nabla g(x^*)^\top y = 0$

Combining two possible cases, we can write down the general conditions for the problem:

$$\begin{aligned} f(x) &\rightarrow \min_{x \in \mathbb{R}^n} \\ \text{s.t. } g(x) &\leq 0 \end{aligned}$$

Let's define the Lagrange function:

$$L(x, \lambda) = f(x) + \lambda g(x)$$

Then x^* point - local minimum of the problem described above, if and only if:

- (1) $\nabla_x L(x^*, \lambda^*) = 0$
 - (2) $\lambda^* \geq 0$
 - (3) $\lambda^* g(x^*) = 0$
 - (4) $g(x^*) \leq 0$
 - (5) $\langle y, \nabla_{xx}^2 L(x^*, \lambda^*)y \rangle > 0$
- $\forall y \neq 0 \in \mathbb{R}^n : \nabla g(x^*)^\top y \leq 0$

It's noticeable, that $L(x^*, \lambda^*) = f(x^*)$. Conditions $\lambda^* = 0$, (1), (4) are the first scenario realization, and conditions $\lambda^* > 0$, (1), (3) - the second.

General formulation

$$\begin{aligned} f_0(x) &\rightarrow \min_{x \in \mathbb{R}^n} \\ \text{s.t. } f_i(x) &\leq 0, i = 1, \dots, m \\ h_i(x) &= 0, i = 1, \dots, p \end{aligned}$$

This formulation is a general problem of mathematical programming.

The solution involves constructing a Lagrange function:

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \sum_{i=1}^p \nu_i h_i(x)$$

Karush-Kuhn-Tucker conditions

Necessary conditions

Let $x^*, (\lambda^*, \nu^*)$ be a solution to a mathematical programming problem with zero duality gap (the optimal value for the primal problem p^* is equal to the optimal value for the dual problem d^*). Let also the functions f, f_i, h_i be differentiable.

- $\nabla_x L(x^*, \lambda^*, \nu^*) = 0$
- $\nabla_\nu L(x^*, \lambda^*, \nu^*) = 0$
- $\lambda_i^* \geq 0, i = 1, \dots, m$
- $\lambda_i^* f_i(x^*) = 0, i = 1, \dots, m$
- $f_i(x^*) \leq 0, i = 1, \dots, m$

Some regularity conditions

These conditions are needed in order to make KKT solutions necessary conditions. Some of them even turn necessary conditions into sufficient (for example, Slater's). Moreover, if you have regularity, you can write down necessary second order conditions $\langle y, \nabla_{xx}^2 L(x^*, \lambda^*, \nu^*)y \rangle \geq 0$ with *semi-definite* hessian of Lagrangian.

- **Slater's condition.** If for a convex problem (i.e., assuming minimization, f_0, f_i are convex and h_i are affine), there exists a point x such that $h(x) = 0$ and $f_i(x) < 0$. (Existence of strictly feasible point), than we have a zero duality gap and KKT conditions become necessary and sufficient.
- **Linearity constraint qualification** If f_i and h_i are affine functions, then no other condition is needed.
- For other examples, see [wiki](#).

Sufficient conditions

For smooth, non-linear optimization problems, a second order sufficient condition is given as follows. The solution x^*, λ^*, ν^* , which satisfies the KKT conditions (above) is a constrained local minimum if for the Lagrangian,

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \sum_{i=1}^p \nu_i h_i(x)$$

the following conditions holds:

$$\begin{aligned} \langle y, \nabla_{xx}^2 L(x^*, \lambda^*, \nu^*) y \rangle &> 0 \\ \forall y \neq 0 \in \mathbb{R}^n : \nabla h_i(x^*)^\top y &\leq 0, \nabla f_j(x^*)^\top y \leq 0 \\ i = 1, \dots, p \quad \forall j : f_j(x^*) &= 0 \end{aligned}$$

References

- [Lecture](#) on KKT conditions (very intuitive explanation) in course "Elements of Statistical Learning" @ KTH.
- [One-line proof of KKT](#)

Example 1

Linear Least squares Write down exact solution of the linear least squares problem:

$$\|Ax - b\|^2 \rightarrow \min_{x \in \mathbb{R}^n}, A \in \mathbb{R}^{m \times n}$$

Consider three cases:

1. $m < n$
 2. $m = n$
 3. $m > n$
- ⋮

Portfolio optimization

Портфельная теория
Маркович

source



Portfolio allocation vector

In this example we show how to do portfolio optimization using CVXPY. We begin with the basic definitions. In portfolio optimization we have some amount of money to invest in any of n different assets. We choose what fraction w_i of our money to invest in each asset i , $i = 1, \dots, n$.

$$\mathbf{1}^T w = 1$$

$$w \in \mathbb{R}^n$$

$$w \geq 0$$

- long only portfolio

We call $w \in \mathbb{R}^n$ the *portfolio allocation vector*. We of course have the constraint that

$\mathbf{1}^T w = 1$. The allocation $w_i < 0$ means a *short position* in asset i , or that we borrow shares to sell now that we must replace later. The allocation $w \geq 0$ is a *long only* portfolio. The quantity

норма портфеля

$$\|w\|_1 = \mathbf{1}^T w_+ + \mathbf{1}^T w_-$$

$w_i < 0$

$$= 1$$

is known as *leverage*.

$$\|w\|_1 = \sum_{i=1}^n |w_i|$$

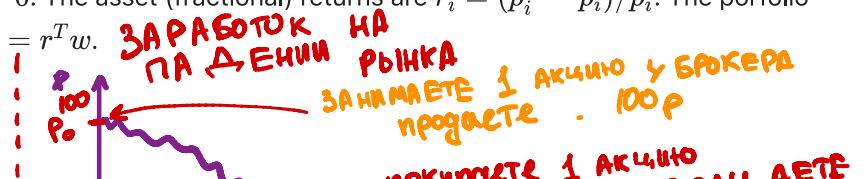
long only

$$> 1$$

Asset returns

We will only model investments held for one period. The initial prices are $p_i > 0$. The end of period prices are $p_i^+ > 0$. The asset (fractional) returns are $r_i = (p_i^+ - p_i)/p_i$. The portfolio (fractional) return is $R = r^T w$.

ЗАРАБОТОК НА
РОСТЕ РЫНКА
предыдущий
период



н Фин.
активов

$w \in \mathbb{R}^n$
ример
 $n = 3$

TC $\begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$
INDX
USDT

$w \begin{pmatrix} 0.7 \\ 0.4 \\ -0.1 \end{pmatrix}$
 $\|w\|_1 = 1$

$\|w\|_1 = 0.7 + 0.4 + 0.1 = 1.2$

ЗАРАБОТОК НА
РОСТЕ РЫНКА
предыдущий
период

1	2	1000

puck:

$$w^T \sum w$$

errorless
no gamma

Classical (Markowitz) portfolio optimization

$$\gamma \in \mathbb{R} \quad \gamma = 0 \quad \gamma = \infty$$

Classical (Markowitz) portfolio optimization solves the optimization problem

$$\min_{\gamma} \gamma w^T \sum w - \mu^T w$$

$$\text{subject to } \begin{cases} \mathbf{1}^T w = 1 \\ w \geq 0 \end{cases}$$

$$\text{maximize } \mu^T w - \gamma w^T \sum w$$

$$\text{subject to } \mathbf{1}^T w = 1, \quad w \in \mathcal{W},$$

where $w \in \mathbb{R}^n$ is the optimization variable, \mathcal{W} is a set of allowed portfolios (e.g., $\mathcal{W} = \mathbb{R}_+^n$ for a long only portfolio), and $\gamma > 0$ is the *risk aversion parameter*.

The objective $\mu^T w - \gamma w^T \sum w$ is the *risk-adjusted return*. Varying γ gives the optimal *risk-return trade-off*. We can get the same risk-return trade-off by fixing return and minimizing risk.

Example

In the following code we compute and plot the optimal risk-return trade-off for 10 assets, restricting ourselves to a long only portfolio.

```
In [ ]:
# Generate data for long only portfolio optimization.
import numpy as np
np.random.seed(1)
n = 10
mu = np.abs(np.random.randn(n, 1))
Sigma = np.random.randn(n, n)
Sigma = Sigma.T @ Sigma
```

```
In [ ]:
# Long only portfolio optimization.
import cvxpy as cp
```

```
w = cp.Variable(n)
gamma = cp.Parameter(nonneg=True)
ret = mu.T @ w
risk = cp.quad_form(w, Sigma)
prob = cp.Problem(cp.Minimize(gamma*risk - ret),
                  [cp.sum(w) == 1,
                   w >= 0])
```

```
In [ ]:
# Compute trade-off curve.
from tqdm.auto import tqdm
SAMPLES = 100
risk_data = np.zeros(SAMPLES)
ret_data = np.zeros(SAMPLES)
gamma_vals = np.logspace(-2, 3, num=SAMPLES)
for i in tqdm(range(SAMPLES)):
    gamma.value = gamma_vals[i]
    prob.solve()
```

$$\text{Ожидаемый доход: } \sum_{i=1}^n \mu_i \cdot w_i = \mu^T w$$

$$\mu^T w \rightarrow m_1$$

$$w_i$$

МАКСИМУЗАЛ
ОЧУГ. ПРИЕМУ

TSLA 0.05 m_1
YNDX 0.01 m_2
BTC -0.02 m_3

$$\mathbf{1}^T w = 1$$

$$w \geq 0$$

$$m_1 w_1 + m_2 w_2 + m_3 w_3$$

Portfolio optimization has two competing objectives: high return and low risk.

Classical (Markowitz) portfolio optimization

$$\gamma \in \mathbb{R} \quad \gamma = 0 \quad \gamma = \infty$$

Classical (Markowitz) portfolio optimization solves the optimization problem

$$\min_{\gamma} \gamma w^T \sum w - \mu^T w$$

$$\text{subject to } \begin{cases} \mathbf{1}^T w = 1 \\ w \geq 0 \end{cases}$$

$$\text{maximize } \mu^T w - \gamma w^T \sum w$$

$$\text{subject to } \mathbf{1}^T w = 1, \quad w \in \mathcal{W},$$

where $w \in \mathbb{R}^n$ is the optimization variable, \mathcal{W} is a set of allowed portfolios (e.g., $\mathcal{W} = \mathbb{R}_+^n$ for a long only portfolio), and $\gamma > 0$ is the *risk aversion parameter*.

The objective $\mu^T w - \gamma w^T \sum w$ is the *risk-adjusted return*. Varying γ gives the optimal *risk-return trade-off*. We can get the same risk-return trade-off by fixing return and minimizing risk.

Example

In the following code we compute and plot the optimal risk-return trade-off for 10 assets, restricting ourselves to a long only portfolio.

```
In [ ]:
# Generate data for long only portfolio optimization.
import numpy as np
np.random.seed(1)
n = 10
mu = np.abs(np.random.randn(n, 1))
Sigma = np.random.randn(n, n)
Sigma = Sigma.T @ Sigma
```

```
In [ ]:
# Long only portfolio optimization.
import cvxpy as cp
```

```
w = cp.Variable(n)
gamma = cp.Parameter(nonneg=True)
ret = mu.T @ w
risk = cp.quad_form(w, Sigma)
prob = cp.Problem(cp.Minimize(gamma*risk - ret),
                  [cp.sum(w) == 1,
                   w >= 0])
```

```
In [ ]:
# Compute trade-off curve.
from tqdm.auto import tqdm
SAMPLES = 100
risk_data = np.zeros(SAMPLES)
ret_data = np.zeros(SAMPLES)
gamma_vals = np.logspace(-2, 3, num=SAMPLES)
for i in tqdm(range(SAMPLES)):
    gamma.value = gamma_vals[i]
    prob.solve()
```

```

risk_data[i] = cp.sqrt(risk).value
ret_data[i] = ret.value

```

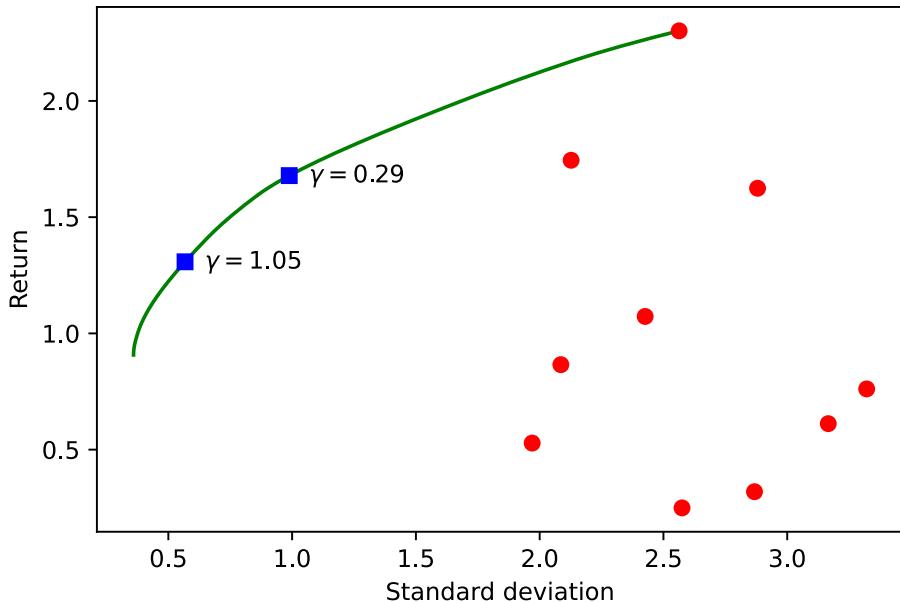
100%|██████████| 100/100 [00:00<00:00, 478.73it/s]

```

In [ ]:
# Plot long only trade-off curve.
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

markers_on = [29, 40]
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(risk_data, ret_data, 'g-')
for marker in markers_on:
    plt.plot(risk_data[marker], ret_data[marker], 'bs')
    ax.annotate(r"$\gamma = %.2f$" % gamma_vals[marker], xy=(risk_data[marker],
for i in range(n):
    plt.plot(cp.sqrt(Sigma[i,i]).value, mu[i], 'ro')
plt.xlabel('Standard deviation')
plt.ylabel('Return')
plt.show()

```



We plot below the return distributions for the two risk aversion values marked on the trade-off curve. Notice that the probability of a loss is near 0 for the low risk value and far above 0 for the high risk value.

```

In [ ]:
# Plot return distributions for two points on the trade-off curve.
import scipy.stats as spstats

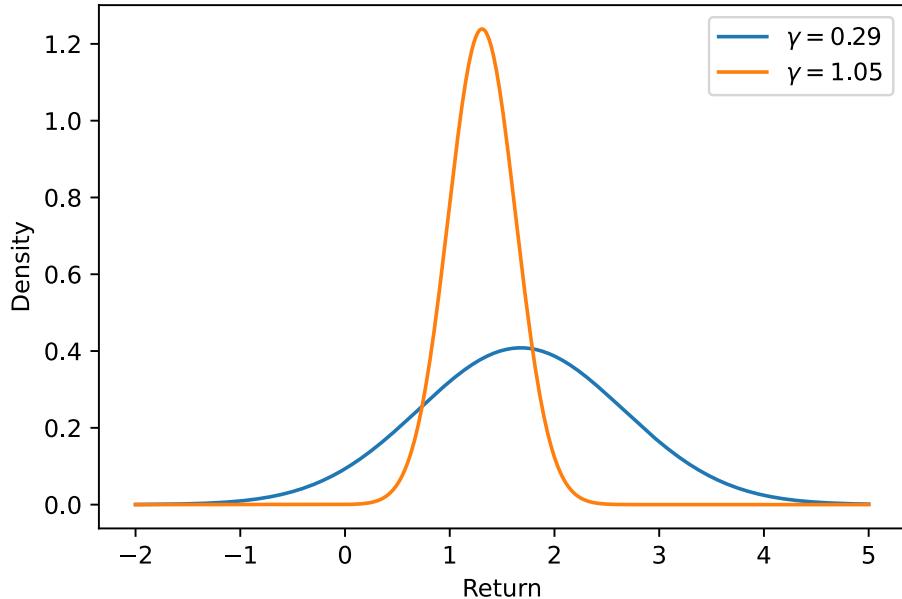
plt.figure()
for midx, idx in enumerate(markers_on):
    gamma.value = gamma_vals[idx]
    prob.solve()
    x = np.linspace(-2, 5, 1000)
    plt.plot(x, spstats.norm.pdf(x, ret.value, risk.value), label=r"$\gamma = %.2f$"
plt.xlabel('Return')

```

```

plt.ylabel('Density')
plt.legend(loc='upper right')
plt.show()

```



Portfolio constraints

There are many other possible portfolio constraints besides the long only constraint. With no constraint ($\mathcal{W} = \mathbf{R}^n$), the optimization problem has a simple analytical solution. We will look in detail at a *leverage limit*, or the constraint that $\|w\|_1 \leq L^{\max}$.

Another interesting constraint is the *market neutral* constraint $m^T \Sigma w = 0$, where m_i is the capitalization of asset i . $M = m^T r$ is the *market return*, and $m^T \Sigma w = \text{cov}(M, R)$. The market neutral constraint ensures that the portfolio return is uncorrelated with the market return.

Example

In the following code we compute and plot optimal risk-return trade-off curves for leverage limits of 1, 2, and 4. Notice that more leverage increases returns and allows greater risk.

```

In [ ]:
# Portfolio optimization with leverage limit.
Lmax = cp.Parameter()
prob = cp.Problem(cp.Maximize(ret - gamma*risk),
                  [cp.sum(w) == 1,
                   cp.norm(w, 1) <= Lmax])

```

```

In [ ]:
# Compute trade-off curve for each leverage limit.
L_vals = [1, 2, 4]
SAMPLES = 100
risk_data = np.zeros((len(L_vals), SAMPLES))
ret_data = np.zeros((len(L_vals), SAMPLES))
gamma_vals = np.logspace(-2, 3, num=SAMPLES)
w_vals = []
for k, L_val in enumerate(L_vals):
    for i in range(SAMPLES):

```

```

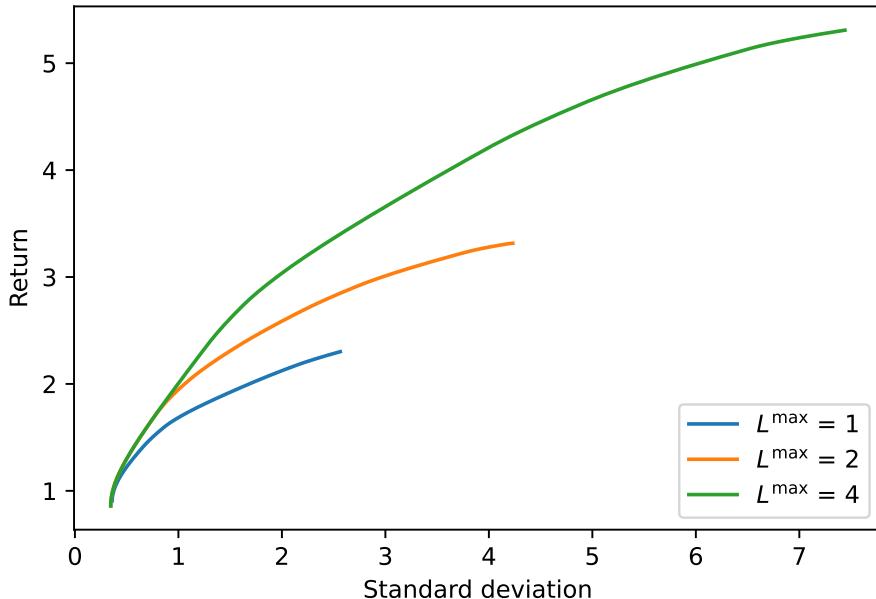
Lmax.value = L_val
gamma.value = gamma_vals[i]
prob.solve(solver=cp.CVXOPT)
risk_data[k, i] = cp.sqrt(risk).value
ret_data[k, i] = ret.value

```

```

In [ ]: # Plot trade-off curves for each leverage limit.
for idx, L_val in enumerate(L_vals):
    plt.plot(risk_data[idx, :], ret_data[idx, :], label=r"$L^{\max} = %d" % L_
for w_val in w_vals:
    w.value = w_val
    plt.plot(cp.sqrt(risk).value, ret.value, 'bs')
plt.xlabel('Standard deviation')
plt.ylabel('Return')
plt.legend(loc='lower right')
plt.show()

```



We next examine the points on each trade-off curve where $w^T \Sigma w = 2$. We plot the amount of each asset held in each portfolio as bar graphs. (Negative holdings indicate a short position.) Notice that some assets are held in a long position for the low leverage portfolio but in a short position in the higher leverage portfolios.

```

In [ ]: # Portfolio optimization with a leverage limit and a bound on risk.
prob = cp.Problem(cp.Maximize(ret),
                  [cp.sum(w) == 1,
                   cp.norm(w, 1) <= Lmax,
                   risk <= 2])

```

```

In [ ]: # Compute solution for different leverage limits.
for k, L_val in enumerate(L_vals):
    Lmax.value = L_val
    prob.solve()
    w_vals.append( w.value )

```

```

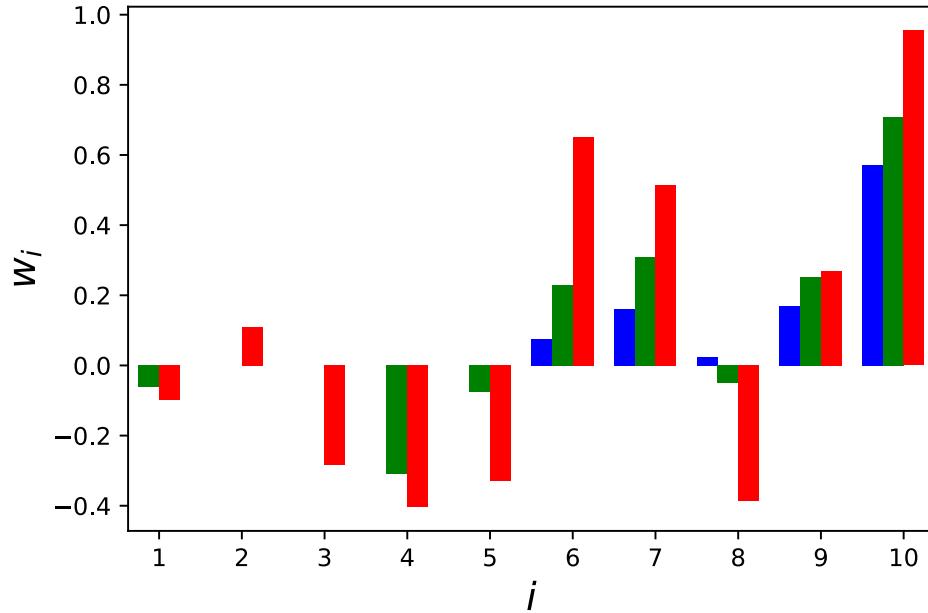
In [ ]: # Plot bar graph of holdings for different leverage limits.
colors = [ 'b', 'g', 'r' ]

```

```

indices = np.argsort(mu.flatten())
for idx, L_val in enumerate(L_vals):
    plt.bar(np.arange(1,n+1) + 0.25*idx - 0.375, w_vals[idx][indices], color=L_color,
            label=r"$L^{\max} = %d" % L_val, width = 0.25)
plt.ylabel(r"$w_i$", fontsize=16)
plt.xlabel(r"$i$", fontsize=16)
plt.xlim([1-0.375, 10+.375])
plt.xticks(np.arange(1,n+1))
plt.show()

```



Variations

There are many more variations of classical portfolio optimization. We might require that $\mu^T w \geq R^{\min}$ and minimize $w^T \Sigma w$ or $\|\Sigma^{1/2} w\|_2$. We could include the (broker) cost of short positions as the penalty $s^T(w)_-$ for some $s \geq 0$. We could include transaction costs (from a previous portfolio w^{prev}) as the penalty

$$\kappa^T |w - w^{\text{prev}}|^\eta, \quad \kappa \geq 0.$$

Common values of η are $\eta = 1, 3/2, 2$.

Factor covariance model

A particularly common and useful variation is to model the covariance matrix Σ as a factor model

$$\Sigma = F \tilde{\Sigma} F^T + D,$$

where $F \in \mathbf{R}^{n \times k}$, $k \ll n$ is the *factor loading matrix*. k is the number of factors (or sectors) (typically 10s). F_{ij} is the loading of asset i to factor j . D is a diagonal matrix; $D_{ii} > 0$ is the *idiosyncratic risk*. $\tilde{\Sigma} > 0$ is the *factor covariance matrix*.

$F^T w \in \mathbf{R}^k$ gives the portfolio *factor exposures*. A portfolio is *factor j neutral* if $(F^T w)_j = 0$.

Portfolio optimization with factor covariance model

Using the factor covariance model, we frame the portfolio optimization problem as

$$\begin{aligned} & \text{maximize} \quad \mu^T w - \gamma (f^T \tilde{\Sigma} f + w^T D w) \\ & \text{subject to} \quad \mathbf{1}^T w = 1, \quad f = F^T w \\ & \quad w \in \mathcal{W}, \quad f \in \mathcal{F}, \end{aligned}$$

where the variables are the allocations $w \in \mathbf{R}^n$ and factor exposures $f \in \mathbf{R}^k$ and \mathcal{F} gives the factor exposure constraints.

Using the factor covariance model in the optimization problem has a computational advantage. The solve time is $O(nk^2)$ versus $O(n^3)$ for the standard problem.

Example

In the following code we generate and solve a portfolio optimization problem with 50 factors and 3000 assets. We set the leverage limit = 2 and $\gamma = 0.1$.

We solve the problem both with the covariance given as a single matrix and as a factor model. Using CVXPY with the OSQP solver running in a single thread, the solve time was 173.30 seconds for the single matrix formulation and 0.85 seconds for the factor model formulation. We collected the timings on a MacBook Air with an Intel Core i7 processor.

```
In [ ]: # Generate data for factor model.
n = 3000
m = 50
np.random.seed(1)
mu = np.abs(np.random.randn(n, 1))
Sigma_tilde = np.random.randn(m, m)
Sigma_tilde = Sigma_tilde.T.dot(Sigma_tilde)
D = np.diag(np.random.uniform(0, 0.9, size=n))
F = np.random.randn(n, m)

In [ ]: # Factor model portfolio optimization.
w = cp.Variable(n)
f = F.T*w
gamma = cp.Parameter(nonneg=True)
Lmax = cp.Parameter()
ret = mu.T*w
risk = cp.quad_form(f, Sigma_tilde) + cp.quad_form(w, D)
prob_factor = cp.Problem(cp.Maximize(ret - gamma*risk),
                        [cp.sum(w) == 1,
                         cp.norm(w, 1) <= Lmax])

# Solve the factor model problem.
Lmax.value = 2
gamma.value = 0.1
prob_factor.solve(verbose=True)

=====
=
CVXPY
v1.2.0
=====
```

```

(CVXPY) Mar 24 01:28:51 PM: Your problem has 3000 variables, 2 constraints, an
d 2 parameters.

/Users/bratishka/anaconda3/lib/python3.9/site-packages/cvxpy/expressions/expre
ssion.py:593: UserWarning:
This use of ``*`` has resulted in matrix multiplication.
Using ``*`` for matrix multiplication has been deprecated since CVXPY 1.1.
    Use ``*`` for matrix-scalar and vector-scalar multiplication.
    Use ``@`` for matrix-matrix and matrix-vector multiplication.
    Use ``multiply`` for elementwise multiplication.
This code path has been hit 1 times so far.

    warnings.warn(msg, UserWarning)
/Users/bratishka/anaconda3/lib/python3.9/site-packages/cvxpy/expressions/expre
ssion.py:593: UserWarning:
This use of ``*`` has resulted in matrix multiplication.
Using ``*`` for matrix multiplication has been deprecated since CVXPY 1.1.
    Use ``*`` for matrix-scalar and vector-scalar multiplication.
    Use ``@`` for matrix-matrix and matrix-vector multiplication.
    Use ``multiply`` for elementwise multiplication.
This code path has been hit 2 times so far.

    warnings.warn(msg, UserWarning)
(CVXPY) Mar 24 01:28:51 PM: It is compliant with the following grammars: DCP,
DQCP
(CVXPY) Mar 24 01:28:51 PM: CVXPY will first compile your problem; then, it wi
ll invoke a numerical solver to obtain a solution.

-----
-
                                         Compilation
-----

-
(CVXPY) Mar 24 01:28:51 PM: Compiling problem (target solver=OSQP).
(CVXPY) Mar 24 01:28:51 PM: Reduction chain: FlipObjective -> CvxAttr2Constr -
> Qp2SymbolicQp -> QpMatrixStuffing -> OSQP
(CVXPY) Mar 24 01:28:51 PM: Applying reduction FlipObjective
(CVXPY) Mar 24 01:28:51 PM: Applying reduction CvxAttr2Constr
(CVXPY) Mar 24 01:28:51 PM: Applying reduction Qp2SymbolicQp
(CVXPY) Mar 24 01:28:51 PM: Applying reduction QpMatrixStuffing
(CVXPY) Mar 24 01:28:51 PM: Applying reduction OSQP
(CVXPY) Mar 24 01:28:51 PM: Finished problem compilation (took 1.366e-01 secon
ds).
(CVXPY) Mar 24 01:28:51 PM: (Subsequent compilations of this problem, using th
e same arguments, should take less time.)

-----
-
                                         Numerical solver
-----

-
(CVXPY) Mar 24 01:28:51 PM: Invoking solver OSQP to obtain a solution.

-----
OSQP v0.6.2 - Operator Splitting QP Solver
(c) Bartolomeo Stellato, Goran Banjac
University of Oxford - Stanford University 2021

-----
problem: variables n = 6050, constraints m = 6052
nnz(P) + nnz(A) = 172325
settings: linear system solver = qdldl,
eps_abs = 1.0e-05, eps_rel = 1.0e-05,
eps_prim_inf = 1.0e-04, eps_dual_inf = 1.0e-04,
rho = 1.00e-01 (adaptive),
sigma = 1.00e-06, alpha = 1.60, max_iter = 10000
check_termination: on (interval 25),
scaling: on, scaled_termination: off
warm start: on, polish: on, time_limit: off

```

```

iter    objective    pri res    dua res    rho        time
 1    -2.1359e+03   7.63e+00   3.73e+02   1.00e-01   2.38e-02s
200    -4.1946e+00   1.59e-03   7.86e-03   3.60e-01   1.82e-01s
400    -4.6288e+00   3.02e-04   6.01e-04   3.60e-01   3.18e-01s
600    -4.6444e+00   2.20e-04   7.87e-04   3.60e-01   4.55e-01s
800    -4.6230e+00   1.09e-04   3.70e-04   3.60e-01   5.91e-01s
1000   -4.6223e+00   8.59e-05   1.04e-04   3.60e-01   7.27e-01s
1200   -4.6205e+00   8.56e-05   9.35e-06   3.60e-01   8.65e-01s
1400   -4.6123e+00   6.44e-05   1.54e-04   3.60e-01   1.00e+00s
1575   -4.6064e+00   2.97e-05   4.06e-05   3.60e-01   1.12e+00s

status:          solved
solution polish: unsuccessful
number of iterations: 1575
optimal objective: -4.6064
run time:         1.14e+00s
optimal rho estimate: 3.87e-01

-----
-
Summary
-----
-
(CVXPY) Mar 24 01:28:52 PM: Problem status: optimal
(CVXPY) Mar 24 01:28:52 PM: Optimal value: 4.6064e+00
(CVXPY) Mar 24 01:28:52 PM: Compilation took 1.366e-01 seconds
(CVXPY) Mar 24 01:28:52 PM: Solver (including time spent in interface) took 1.
144e+00 seconds
4.606413077728827

```

Out[]:

```

In [ ]:
# Standard portfolio optimization with data from factor model.
risk = cp.quad_form(w, F.dot(Sigma_tilde).dot(F.T) + D)
prob = cp.Problem(cp.Maximize(ret - gamma*risk),
                  [cp.sum(w) == 1,
                   cp.norm(w, 1) <= Lmax])

# Uncomment to solve the problem.
# WARNING: this will take many minutes to run.
prob.solve(verbose=True, max_iter=30000)

```

```

=====
=
CVXPY
v1.2.0
=====
=
(CVXPY) Mar 24 01:28:54 PM: Your problem has 3000 variables, 2 constraints, an
d 2 parameters.

```

In []:

```

print('Factor model solve time = {}'.format(prob_factor.solver_stats.solve_time))
print('Single model solve time = {}'.format(prob.solver_stats.solve_time))

```

```

Factor model solve time = 2.1817036670000003
Single model solve time = 447.57964334400003

```

Materials

- Portfolio Optimization Algo Trading colab notebook
- Multi objective portfolio optimization

