Portfolio optimization

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, $i = 1, \ldots, n$.

We call $w \in \mathbf{R}^n$ 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

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

is known as *leverage*.

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 porfolio (fractional) return is $R = r^T w$.

A common model is that r is a random variable with mean $\mathbf{E}r = \mu$ and covariance $\mathbf{E}(\mathbf{r}-\mu)(\mathbf{r}-\mu)^{\mathbf{T}} = \Sigma$. It follows that R is a random variable with $\mathbf{E}R = \mu^T w$ and $\mathbf{var}(R) = w^T \Sigma w$. $\mathbf{E}R$ is the (mean) *return* of the portfolio. $\mathbf{var}(R)$ is the *risk* of the portfolio. (Risk is also sometimes given as $\mathbf{std}(R) = \sqrt{\mathbf{var}(R)}$.)

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

Classical (Markowitz) portfolio optimization

Classical (Markowitz) portfolio optimization solves the optimization problem

$$\begin{array}{ll} \text{maximize} & \mu^T w - \gamma w^T \Sigma w \\ \text{subject to} & \mathbf{1}^T w = 1, \quad w \in \mathcal{W}, \end{array}$$

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

The objective $\mu^T w - \gamma w^T \Sigma w$ 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 [ ]:

```python
# 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 [ ]:

```python
# 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 [ ]:

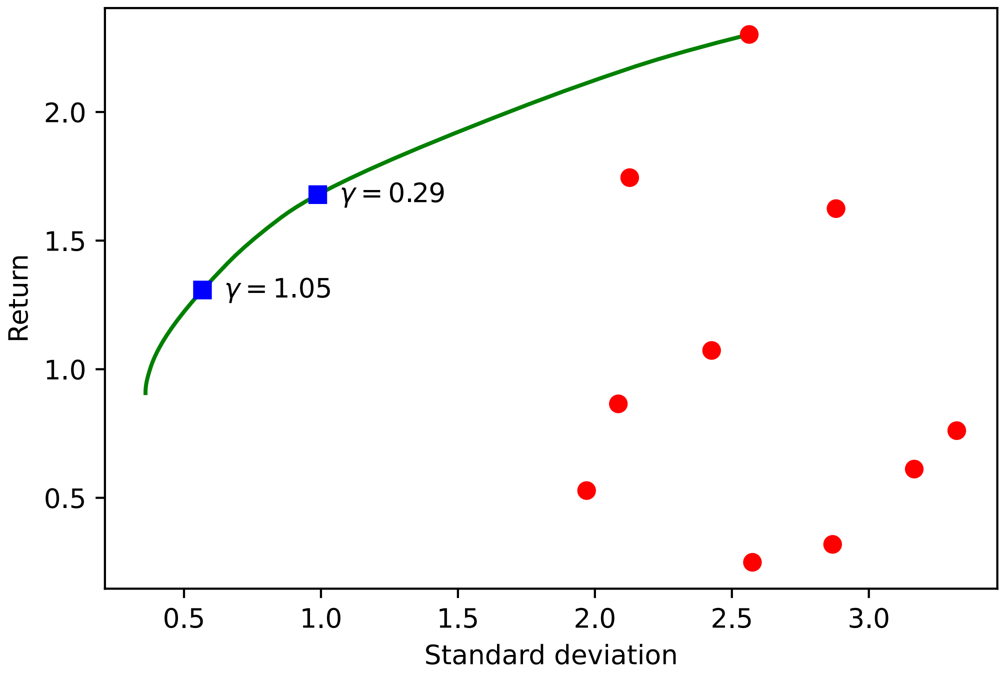
```python
# Compute trade-off curve.
from tqdm.auto import tqdm
SAMPLES = 100
risk_data = np.zeros(SAMPLES)
ret_data = np.zeros(SAMPLES)
gamma_vals = np.logspace(-2, 3, num=SAMPLES)
for i in tqdm(range(SAMPLES)):
    gamma.value = gamma_vals[i]
    prob.solve()
    risk_data[i] = cp.sqrt(risk).value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



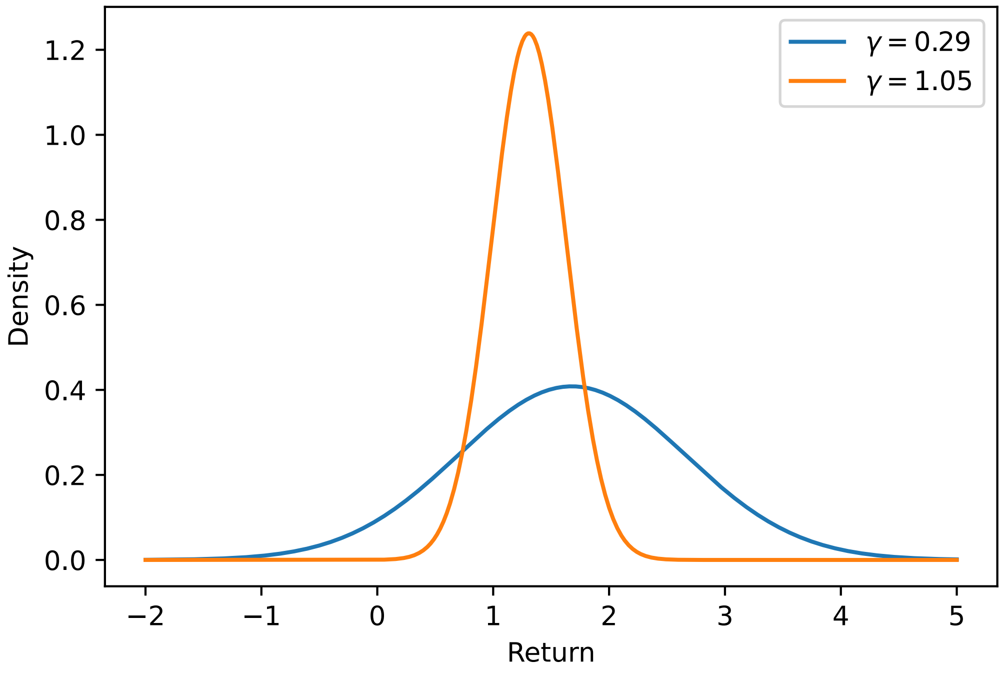
We plot below the return distributions for the two risk aversion values marked on the trade-off curve. Notice that the probability of a loss is near 0 for the low risk value and far above 0 for the high risk value.

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 =

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 $m^T \Sigma w = \mathbf{cov}(M, R)$. 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 [ ]:

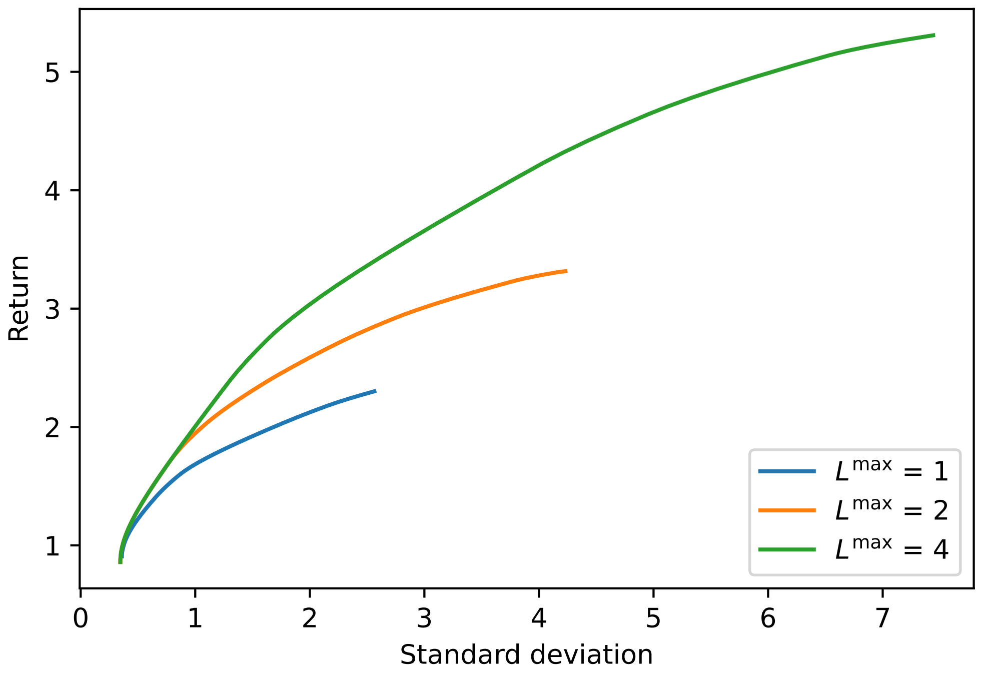
```
# Portfolio optimization with leverage limit.
Lmax = cp.Parameter()
prob = cp.Problem(cp.Maximize(ret - gamma*risk),
               [cp.sum(w) == 1,
                cp.norm(w, 1) <= Lmax])
```

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])
```

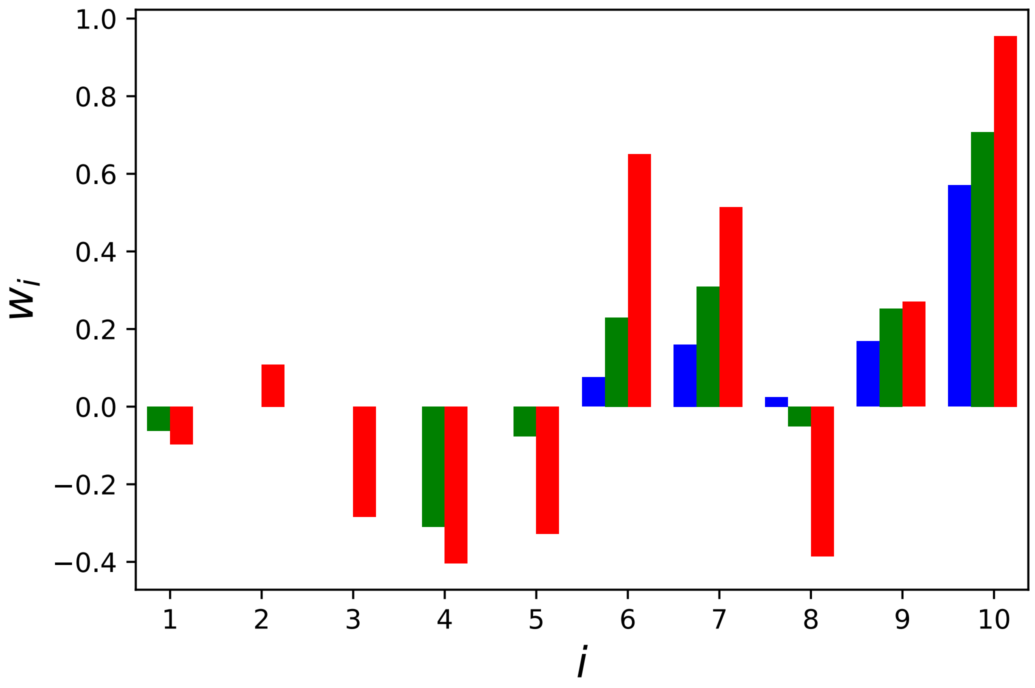
In [ ]:

```
# 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
             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

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

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: 4.606e+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
```

Out[ ]:
```
4.606413077728827
```

In [ ]:
```python
# 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 [ ]:
```python
print('Factor model solve time = {}'.format(prob_factor.solver_stats.solve_ti
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

Optimality conditions. KKT

Background

Extreme value (Weierstrass) theorem

Let $S \subset \mathbb{R}^n$ be compact set and $f(x)$ continuous function on S. So that, the point of the global minimum of the function $f(x)$ on S exists.

Lagrange multipliers

Consider simple yet practical case of equality constraints:

$$
\begin{split}
& f(x) \to \min\limits_{x \in \mathbb{R}^n} \\
\text{s.t. } & h_i(x) = 0, i = 1, \ldots, m
\end{split}
$$

The basic idea of Lagrange method implies switch from conditional to unconditional optimization through increasing the dimensionality of the problem:

$$
L(x, \lambda) = f(x) + \sum\limits_{i=1}^m \lambda_i h_i(x) \to \min\limits_{x \in \mathbb{R}^n, \lambda \in \mathbb{R}^m}
$$

General formulations and conditions

$$
f(x) \to \min\limits_{x \in S}
$$

We say that the problem has a solution if the following set **is not empty**: $x^* \in S$, in which the minimum or the infimum of the given function is achieved.

Unconstrained optimization

General case

Let $f(x) : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function.

$$
f(x) \to \min\limits_{x \in \mathbb{R}^n} \tag{UP}
$$

If x^* - is a local minimum of $f(x)$, then:

$$
\nabla f(x^*) = 0 \tag{UP:Necessary}
$$

$$\nabla f(x^*) = 0 \tag{UP:Necessary}$$

If $f(x)$ at some point x^* satisfies the following conditions:

$$H_f(x^*) = \nabla^2 f(x^*) \succeq (\preceq) 0, \tag{UP:Sufficient}$$

then (if necessary condition is also satisfied) x^* is a local minimum(maximum) of $f(x)$.

Convex case

It should be mentioned, that in **convex** case (i.e., $f(x)$ is convex) necessary condition becomes sufficient. Moreover, we can generalize this result on the class of non-differentiable convex functions.

Let $f(x) : \mathbb{R}^n \to \mathbb{R}$ - convex function, then the point x^* is the solution of (UP) if and only if:

$$0_n \in \partial f(x^*)$$

One more important result for convex constrained case sounds as follows. If $f(x) : S \to \mathbb{R}$ - convex function defined on the convex set S, then:

- Any local minima is the global one.
- The set of the local minimizers S^* is convex.
- If $f(x)$ - strongly convex function, then S^* contains only one single point $S^* = x^*$.

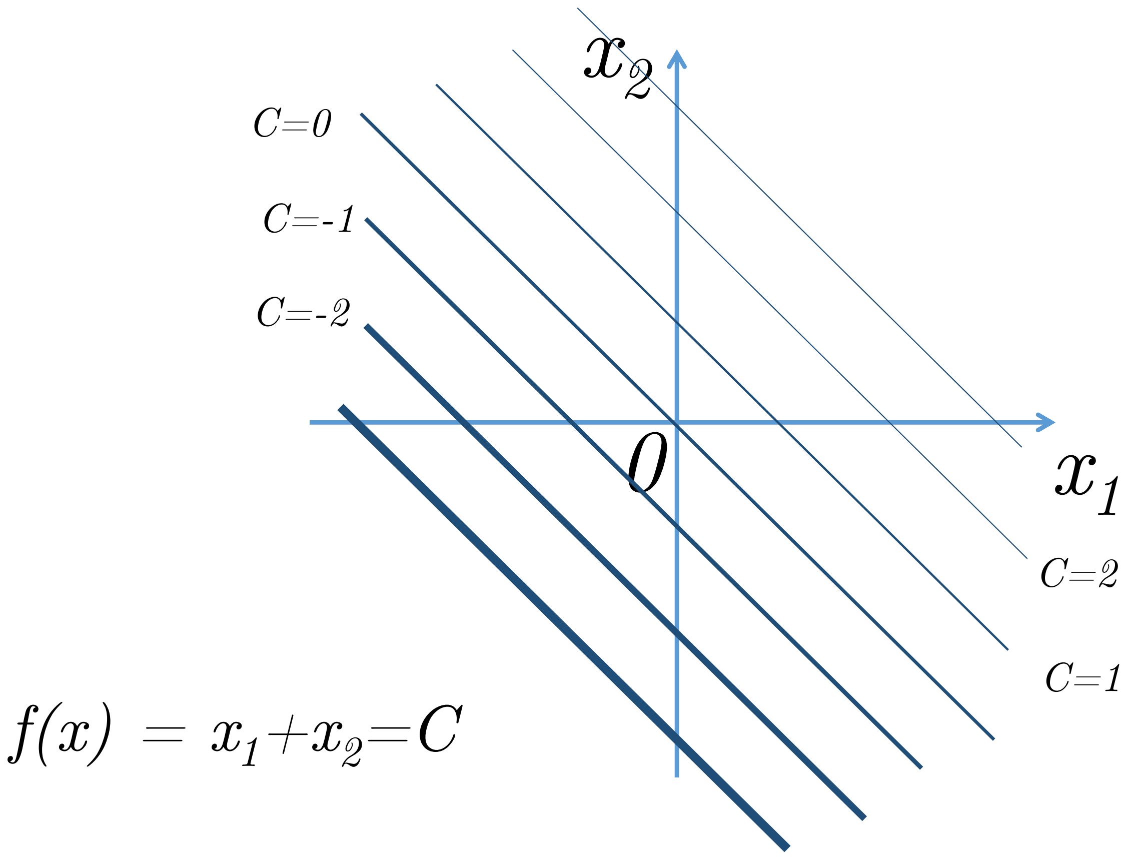
Optimization with equality conditions

Intuition

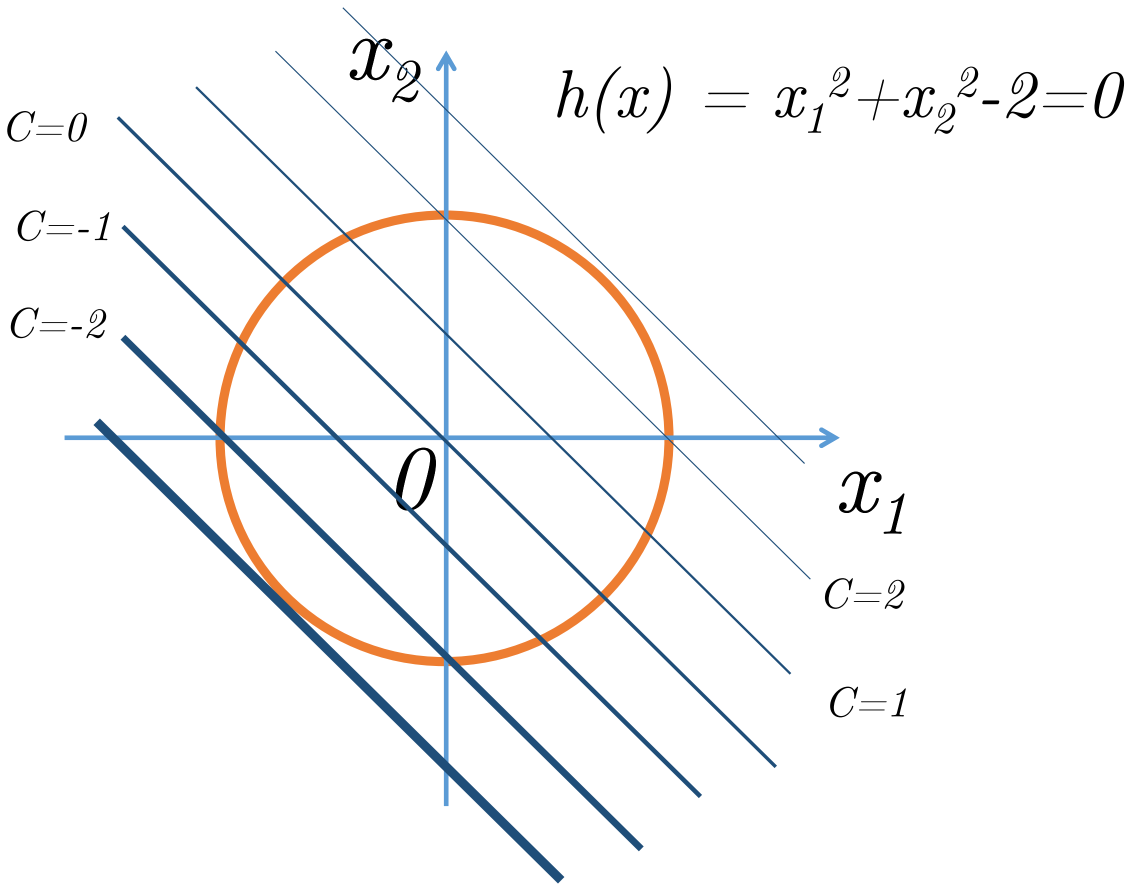
Things are pretty simple and intuitive in unconstrained problem. In this section we will add one equality constraint, i.e.

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

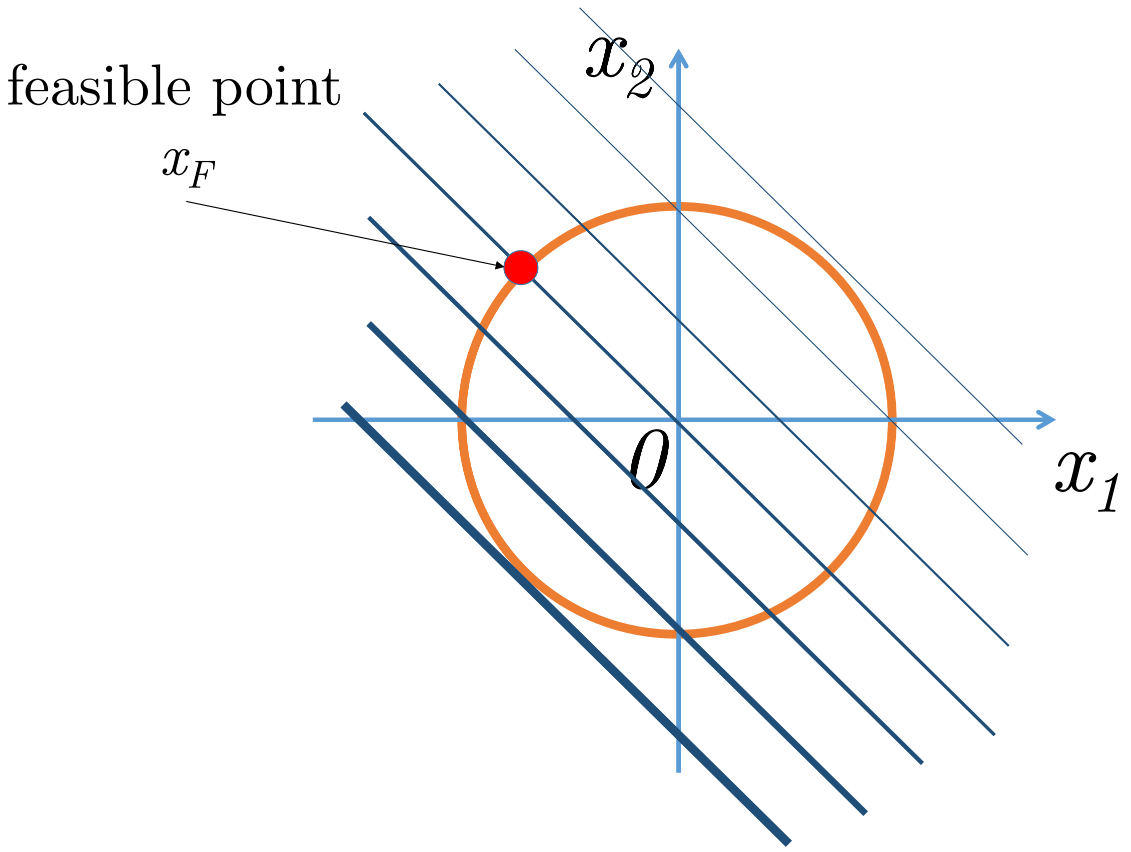
We will try to illustrate approach to solve this problem through the simple example with $f(x) = x_1 + x_2$ and $h(x) = x_1^2 + x_2^2 - 2$

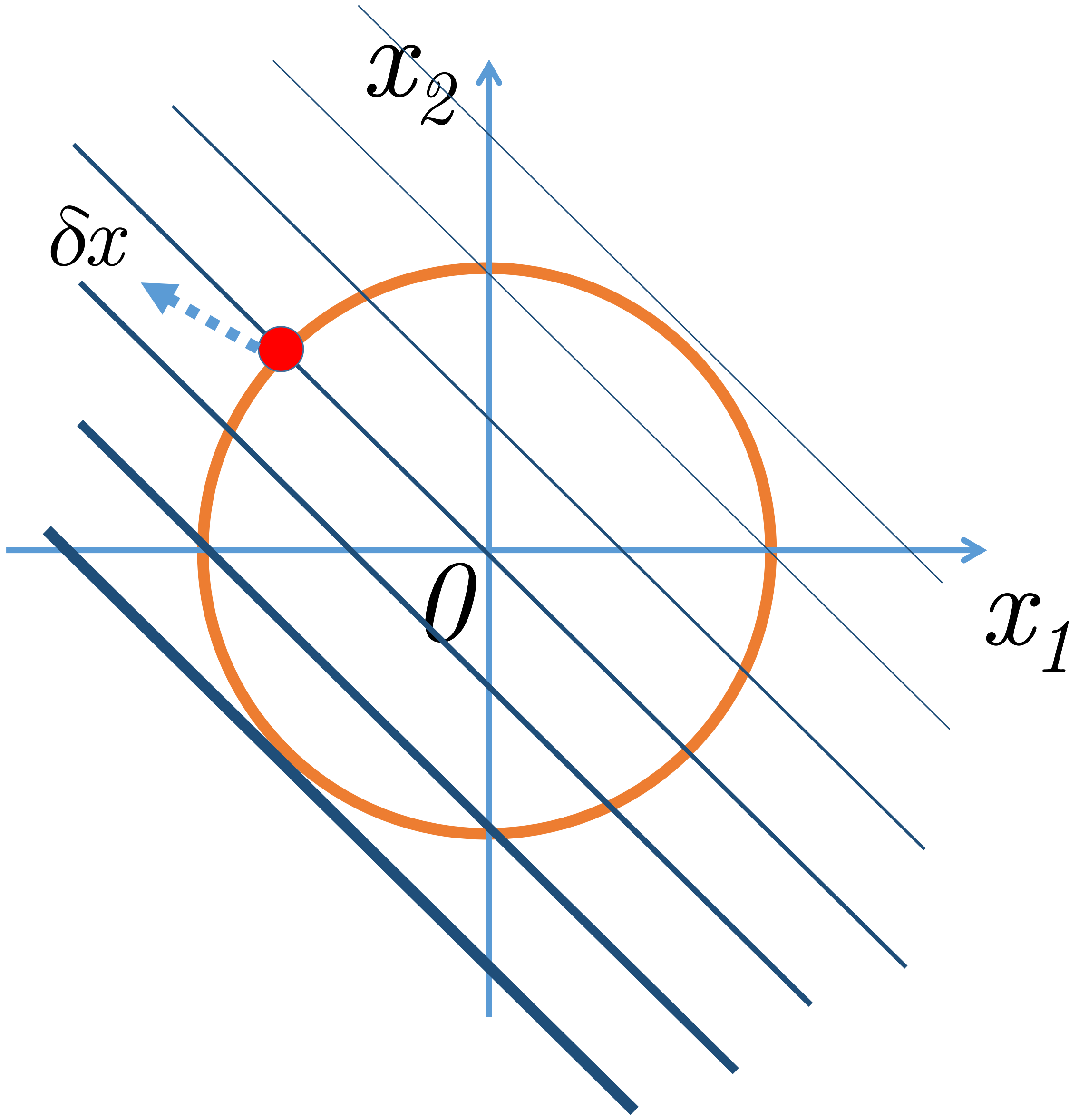


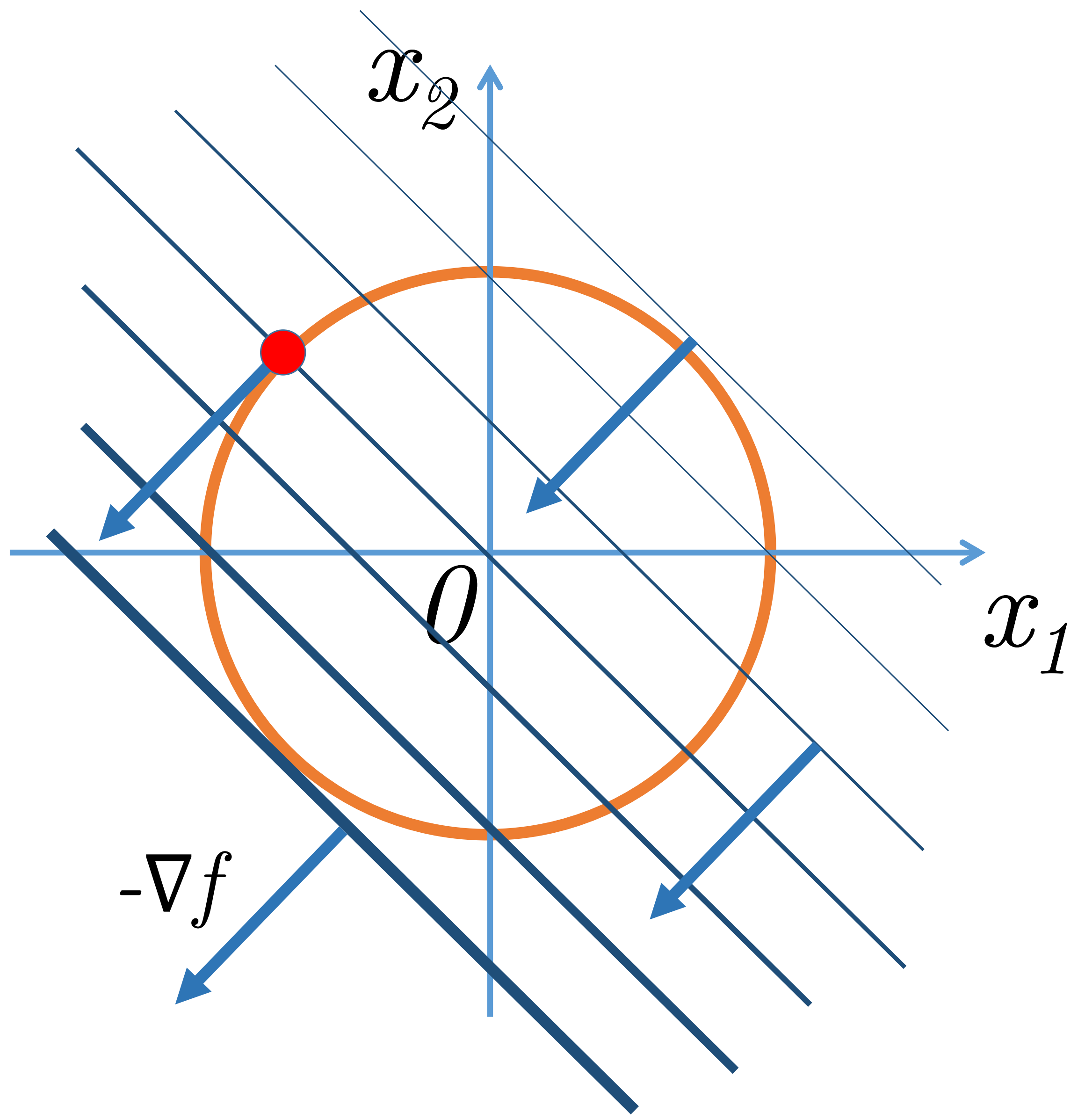
x_2
$C=0$
$C=-1$
$C=-2$
0
x_1
$C=2$
$C=1$
$f(x) = x_1+x_2=C$

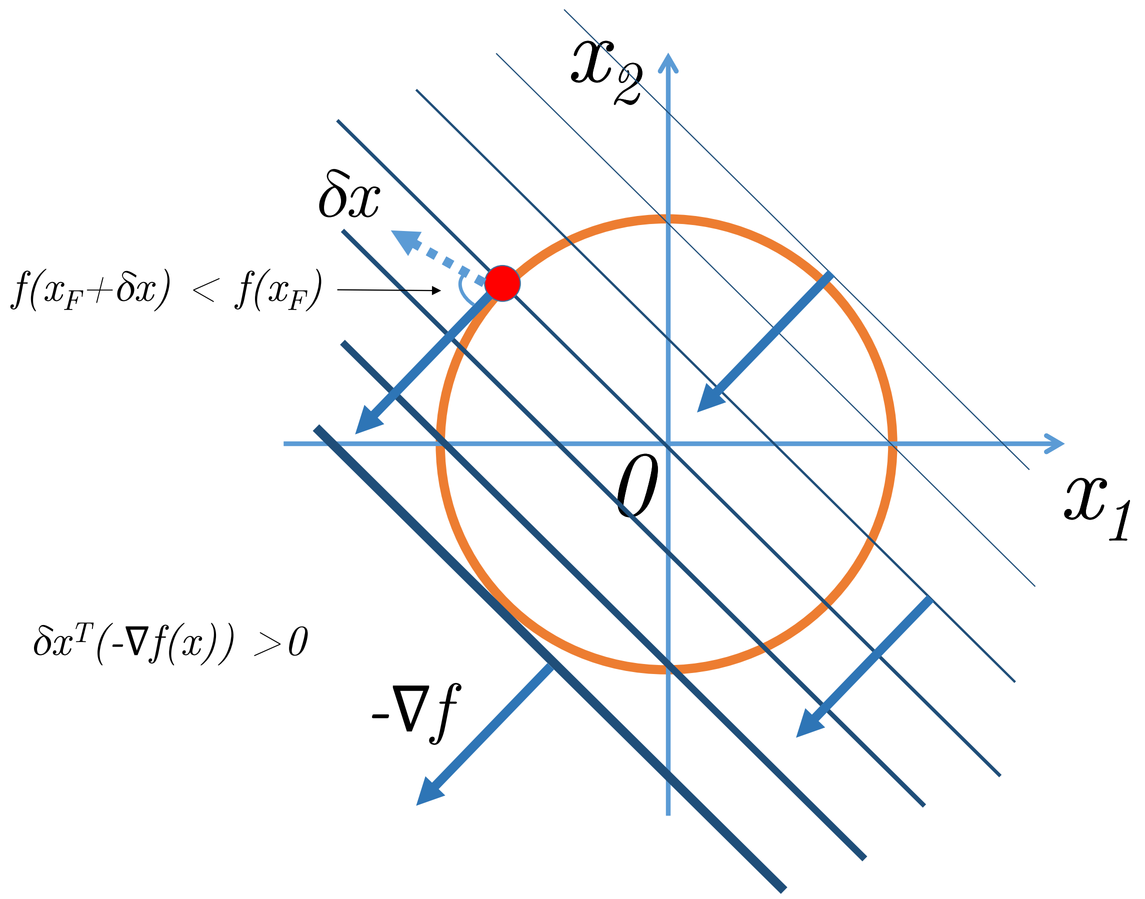


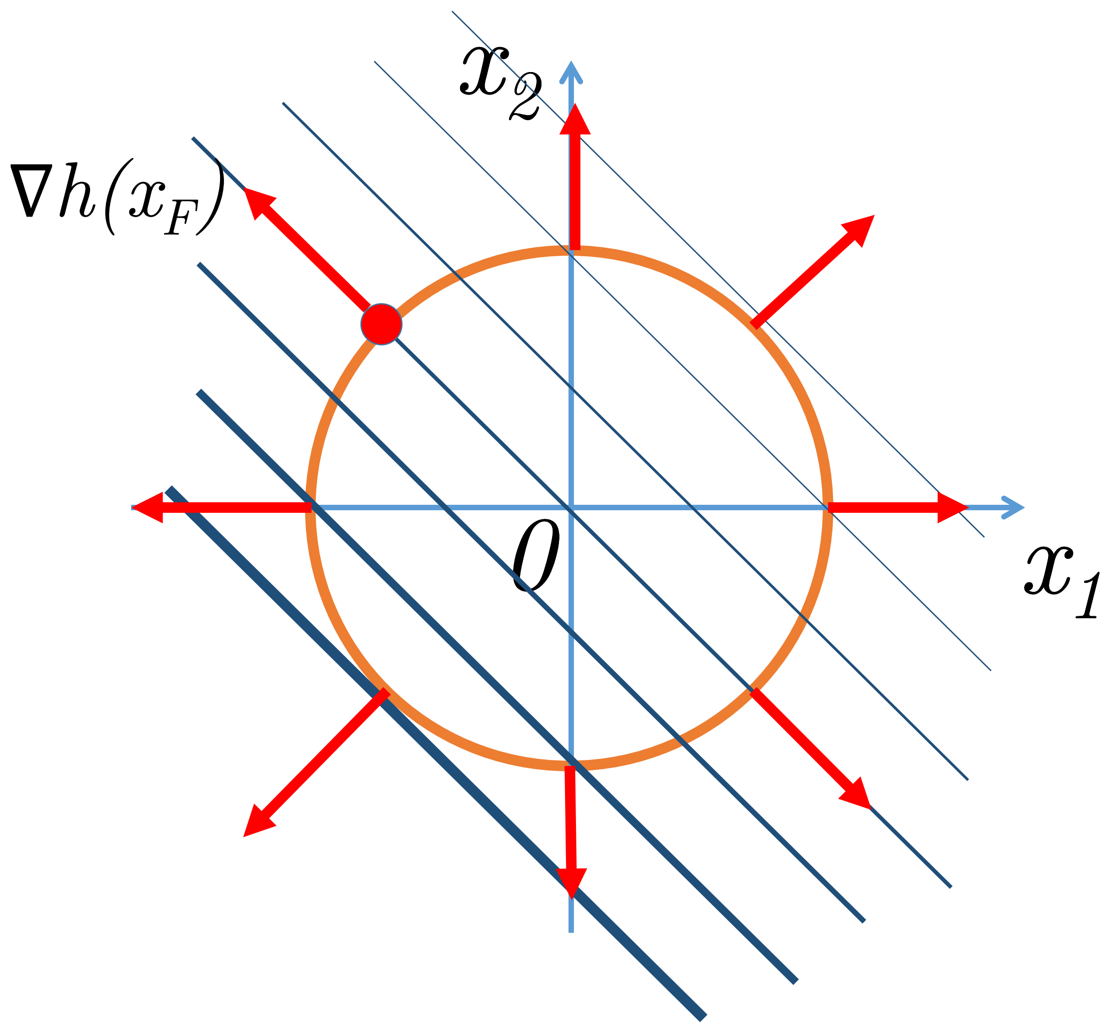
x_2
$h(x) = x_1^2+x_2^2-2=0$
$C=0$
$C=-1$
$C=-2$
0
x_1
$C=2$
$C=1$

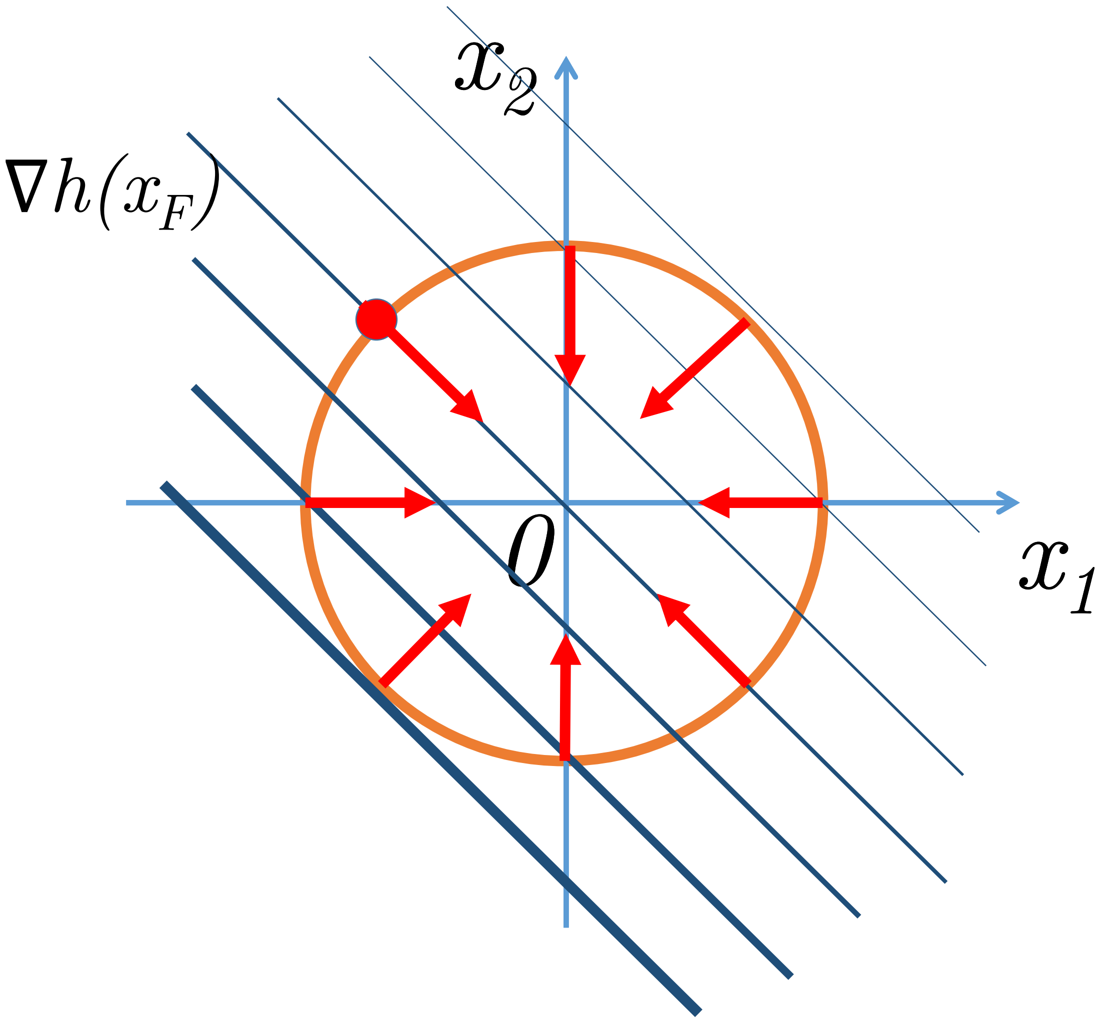

feasible point
x_F
x_2
x_1
0

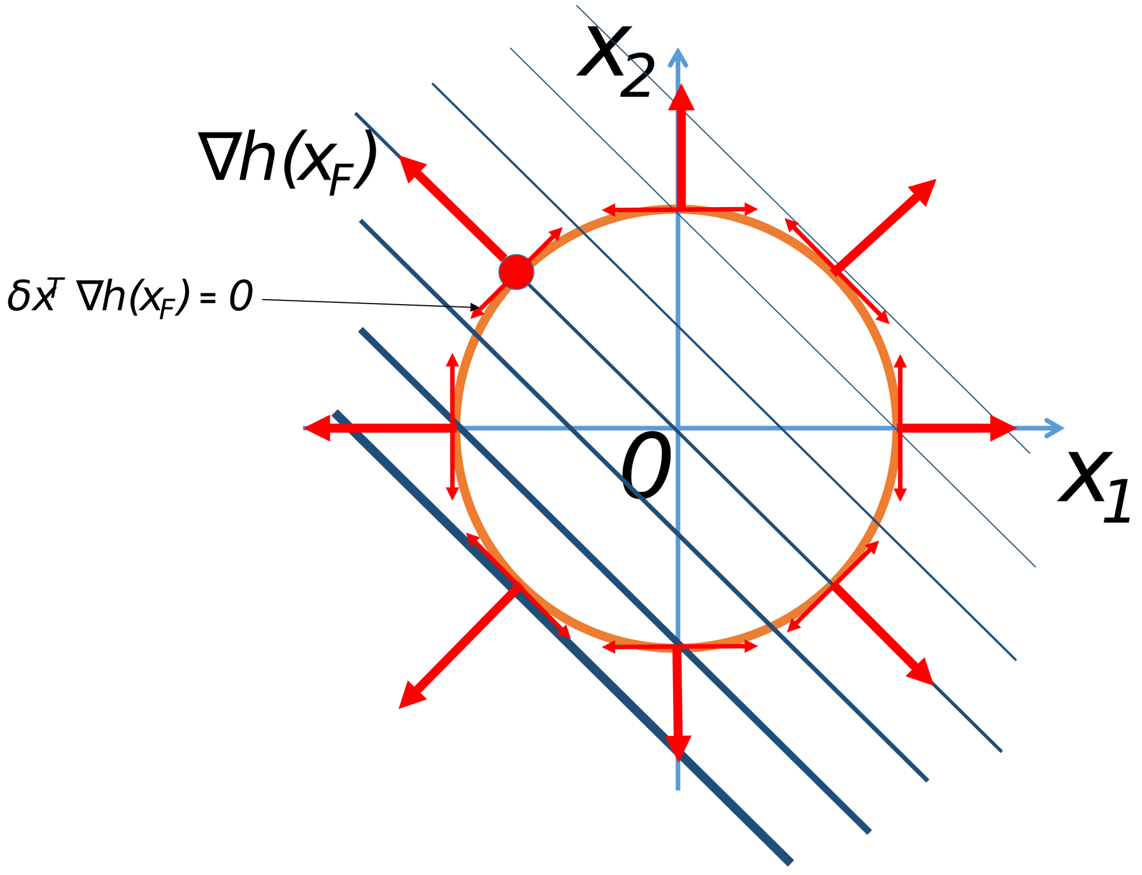

x_2
δx
x_1
0


x_2
x_1
0
$-\nabla f$


x_2
δx
$f(x_F+\delta x) < f(x_F)$
x_1
0
$\delta x^T(-\nabla f(x)) > 0$
$-\nabla f$


x_2
$\nabla h(x_F)$
x_1
0





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

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

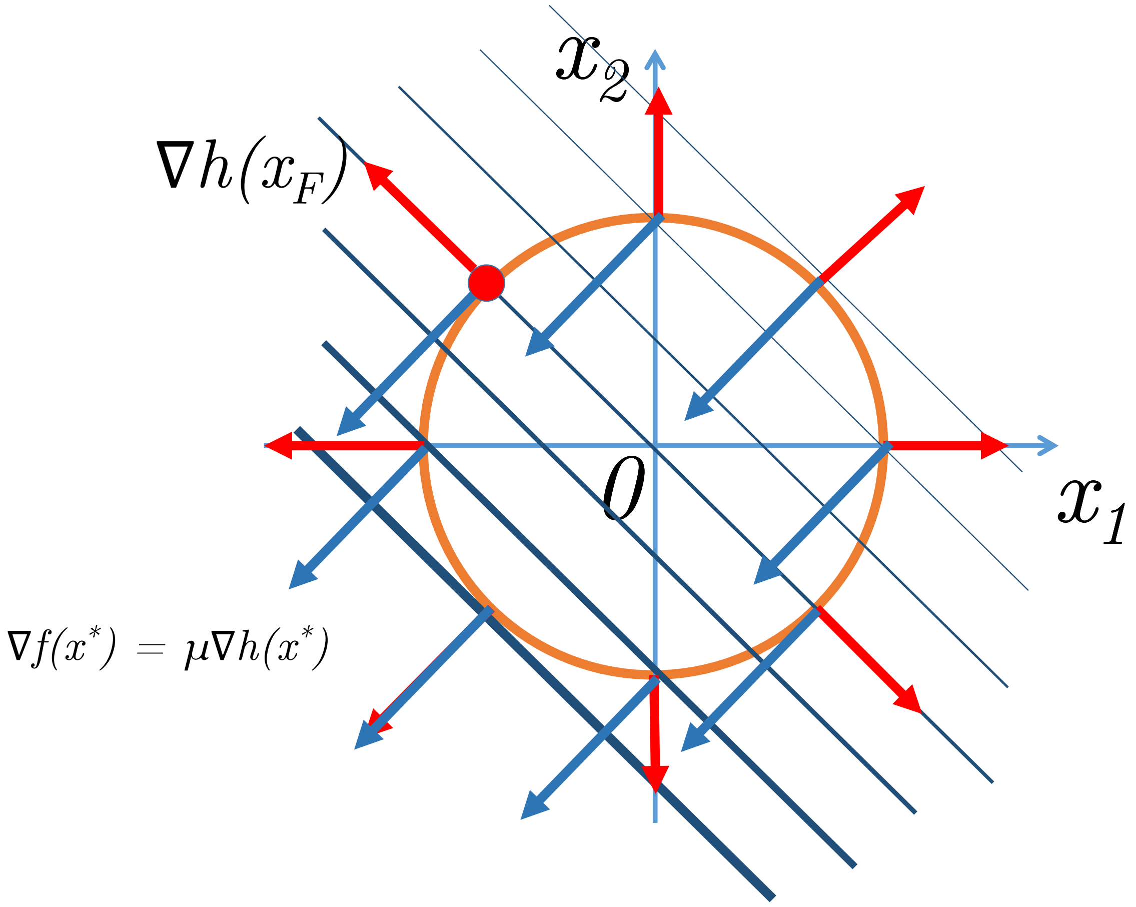
$$\langle \delta x, -\nabla f(x_F) \rangle > 0$$

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

$$\nabla f(x) = \lambda \nabla h(x)$$

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

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 limited problem :)



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

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

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

$$\begin{split} & \nabla_x L(x^*, \lambda^*) = 0 \text{ that's written above} \\ & \nabla_\lambda L(x^*, \lambda^*) = 0 \text{ condition of being in budget set} \\ & \langle y, \nabla^2_{xx} L(x^*, \lambda^*) y \rangle \geq 0, \quad \forall y \in \mathbb{R}^n : \nabla h(x^*)^\top y = 0 \end{split}$$

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

General formulation

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

$$\text{s.t. } h_i(x) = 0, \; i = 1, \ldots, m$$

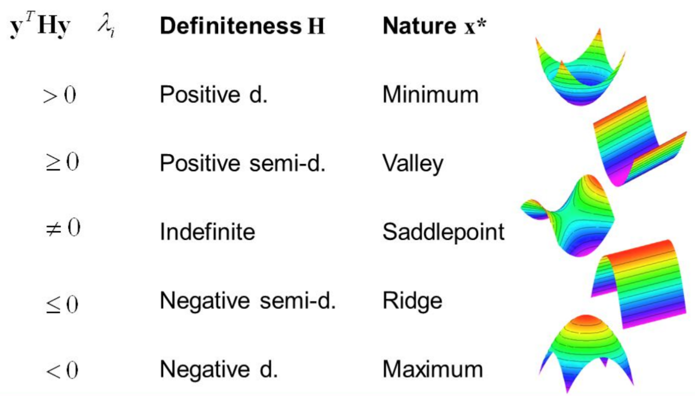
Solution

$$L(x,\lambda) = f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) = f(x) + \lambda^\top h(x)$$

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 $x \in \mathbb{R}^n, \lambda \in \mathbb{R}^m$ are written as

$$\begin{aligned}
&\nabla_x L(x^*,\lambda^*) = 0\\
&\nabla_\lambda L(x^*,\lambda^*) = 0\\
&\langle y, \nabla^2_{xx} L(x^*,\lambda^*) y\rangle \geq 0, \quad \forall y \in \mathbb{R}^n : \nabla h(x^*)^\top y = 0
\end{aligned}$$

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



$\mathbf{y}^T\mathbf{H}\mathbf{y}$ λ_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

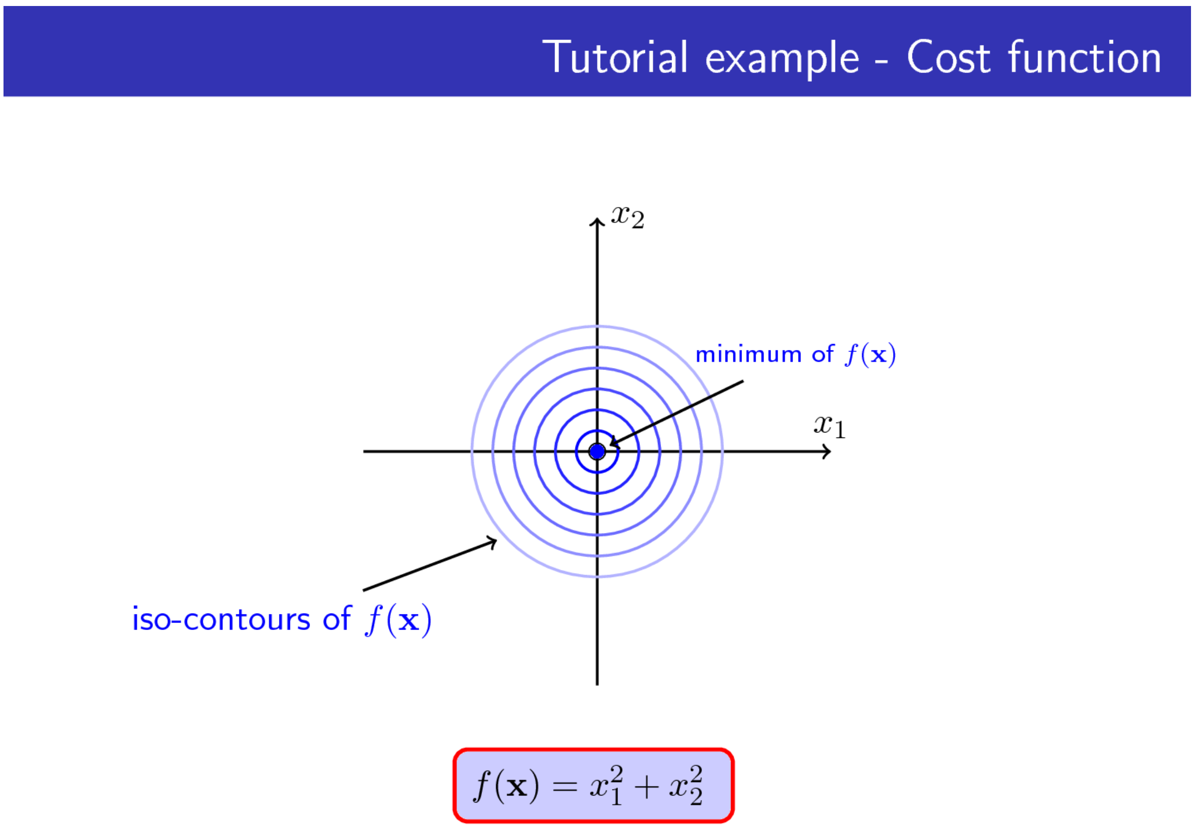
Optimization with inequality conditions

Example

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

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

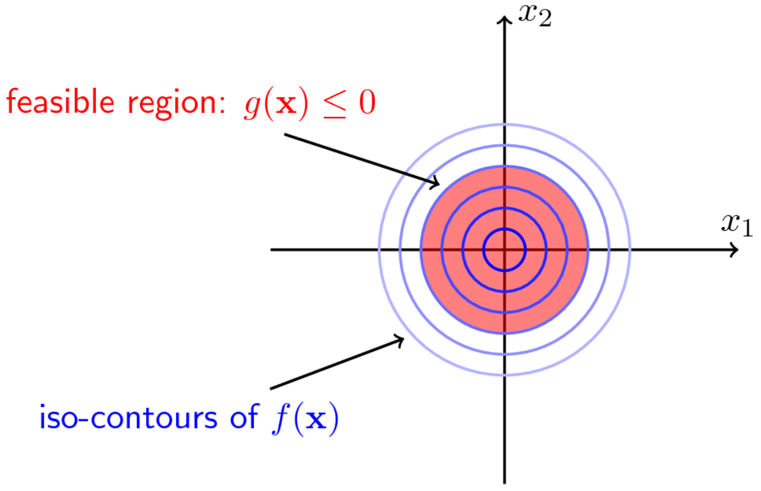
$$\text{s.t. } g(x) \leq 0$$



Tutorial example - Cost function

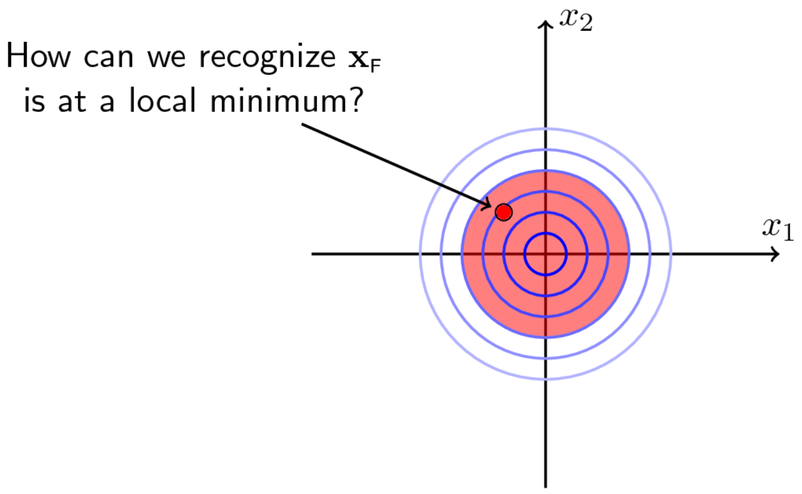
$$\boxed{f(\mathbf{x}) = x_1^2 + x_2^2}$$

Tutorial example - Feasible region


x_2
feasible region: $g(\mathbf{x}) \leq 0$
x_1
iso-contours of $f(\mathbf{x})$

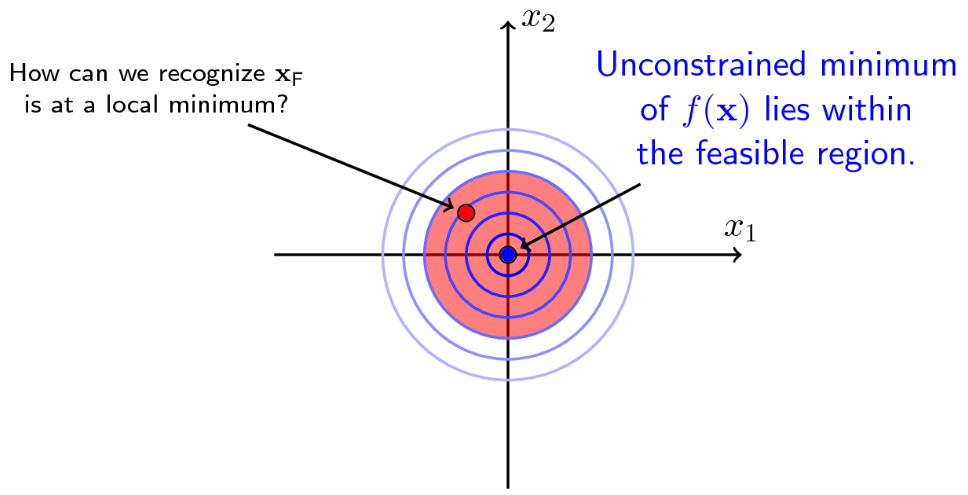
$$\boxed{g(\mathbf{x}) = x_1^2 + x_2^2 - 1}$$

How do we recognize if $\mathbf{x}_F$ is at a local optimum?



Remember $\mathbf{x}_F$ denotes a feasible point.

Easy in this case


How can we recognize $\mathbf{x}_F$ is at a local minimum?
Unconstrained minimum of $f(\mathbf{x})$ lies within the feasible region.
x_2
x_1

$\therefore$ Necessary and sufficient conditions for a constrained local minimum are the same as for an unconstrained local minimum.

$$\boxed{\nabla_{\mathbf{x}} f(\mathbf{x}_F) = \mathbf{0} \quad \text{and} \quad \nabla_{\mathbf{xx}} f(\mathbf{x}_F) \text{ is positive definite}}$$

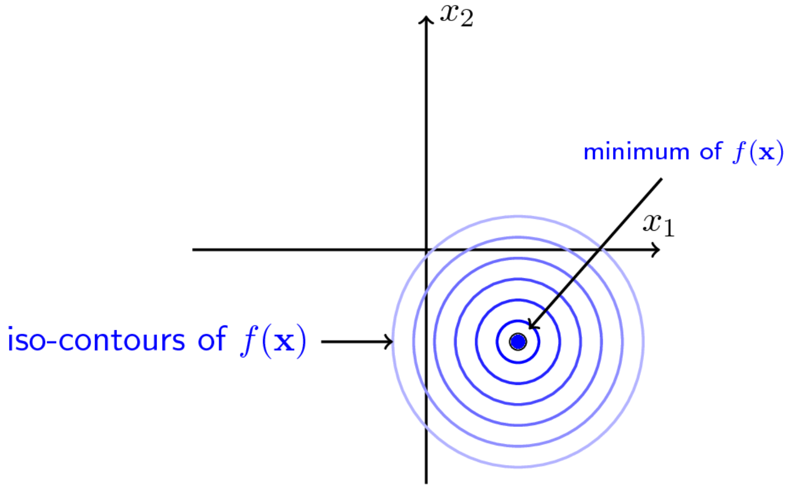
Thus, if the constraints of the type of inequalities are inactive in the UM problem, then don't worry and write out the solution to the UM problem. However, this is not a heal-all :) Consider the second childish example

$$f(x) = (x_1 - 1.1)^2 + (x_2 + 1.1)^2 \quad g(x) = x_1^2 + x_2^2 - 1$$

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

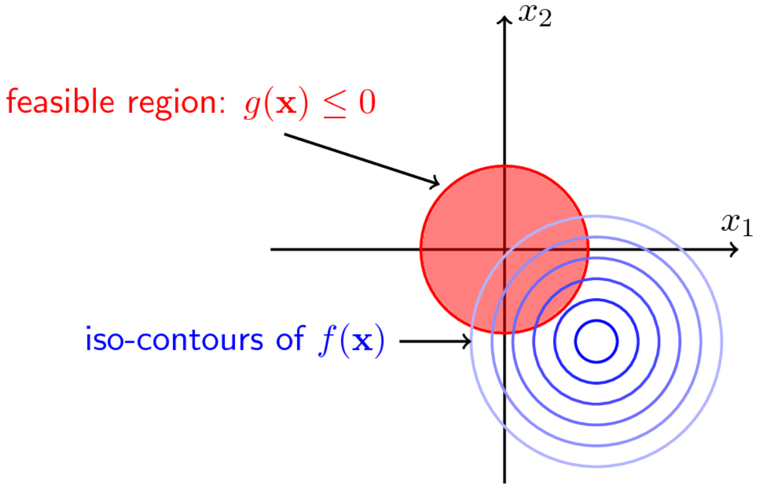
$$\text{s.t. } g(x) \leq 0$$

Tutorial example - Cost function



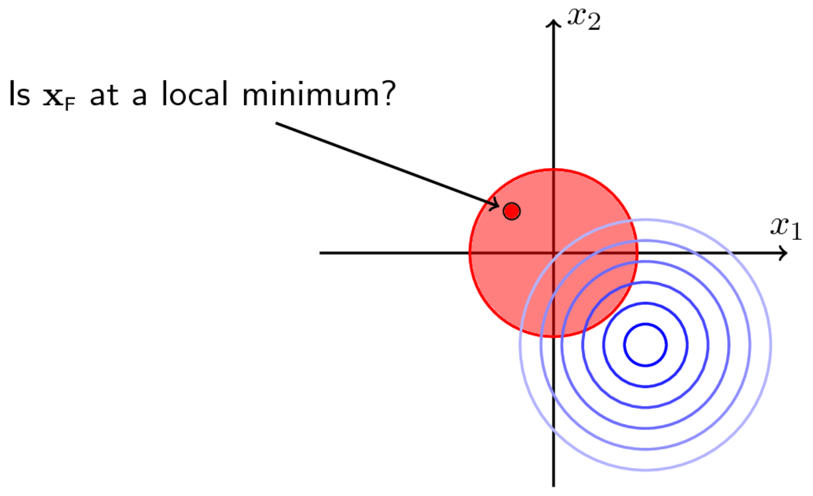
$$\boxed{f(\mathbf{x}) = (x_1 - 1.1)^2 + (x_2 + 1.1)^2}$$

Tutorial example - Feasible region


x_2
feasible region: $g(\mathbf{x}) \leq 0$
x_1
iso-contours of $f(\mathbf{x})$

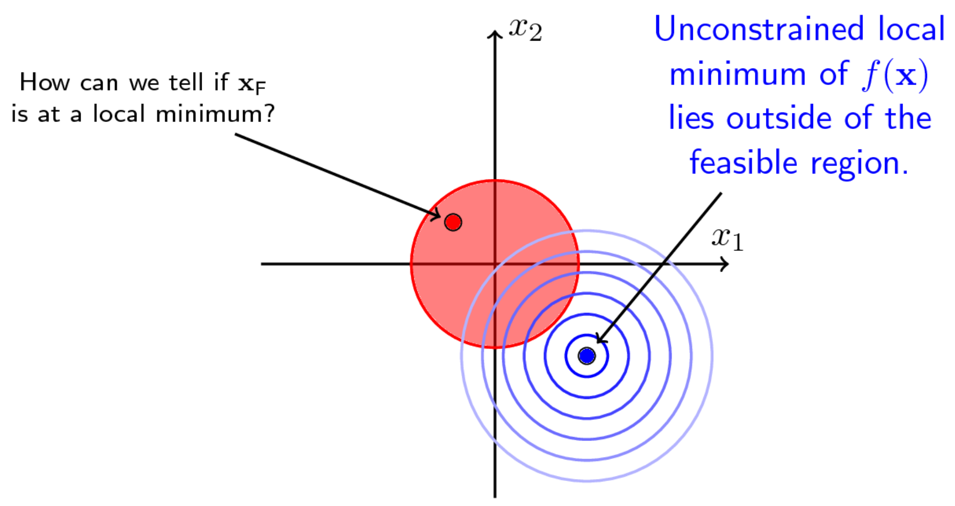
$$\boxed{g(\mathbf{x}) = x_1^2 + x_2^2 - 1}$$

How do we recognize if $\mathbf{x}_F$ is at a local optimum?



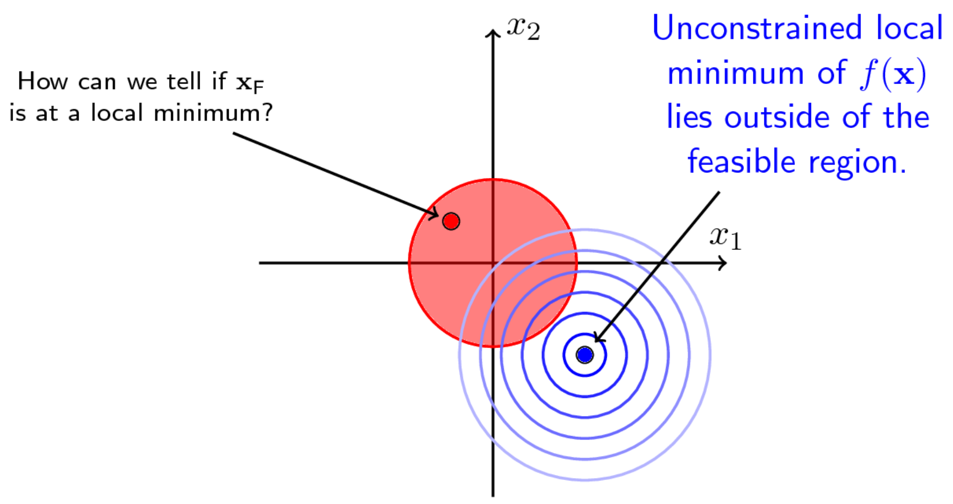
Remember $\mathbf{x}_F$ denotes a feasible point.

How do we recognize if $\mathbf{x}_F$ is at a local optimum?


How can we tell if $\mathbf{x}_F$ is at a local minimum?
x_2
x_1
Unconstrained local minimum of $f(\mathbf{x})$ lies outside of the feasible region.

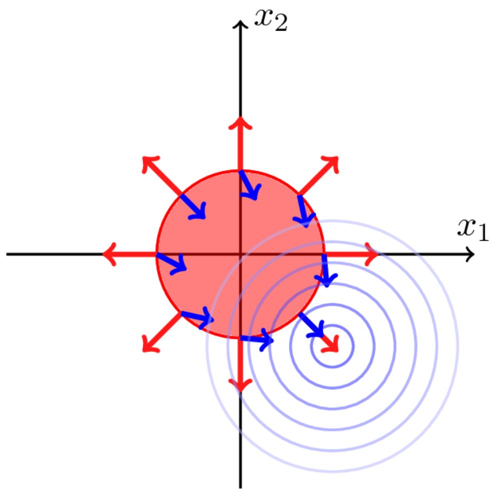
$\therefore$ the constrained local minimum occurs on the surface of the constraint surface.

How do we recognize if $\mathbf{x}_F$ is at a local optimum?


How can we tell if $\mathbf{x}_F$ is at a local minimum?
Unconstrained local minimum of $f(\mathbf{x})$ lies outside of the feasible region.
x_2
x_1

$\therefore$ Effectively have an optimization problem with an **equality constraint**: $g(\mathbf{x}) = 0$.

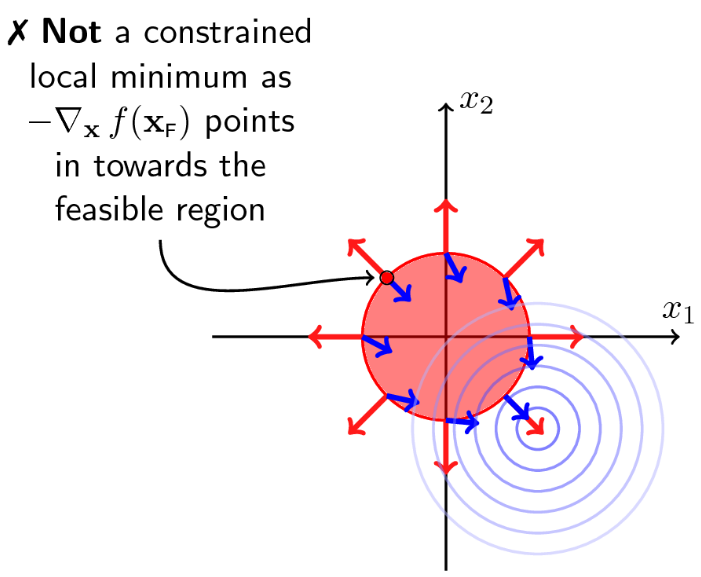
Given an equality constraint



A local optimum occurs when $\nabla_{\mathbf{x}} f(\mathbf{x})$ and $\nabla_{\mathbf{x}} g(\mathbf{x})$ are parallel:

$$\boxed{-\nabla_{\mathbf{x}} f(\mathbf{x}) = \lambda \nabla_{\mathbf{x}} g(\mathbf{x})}$$

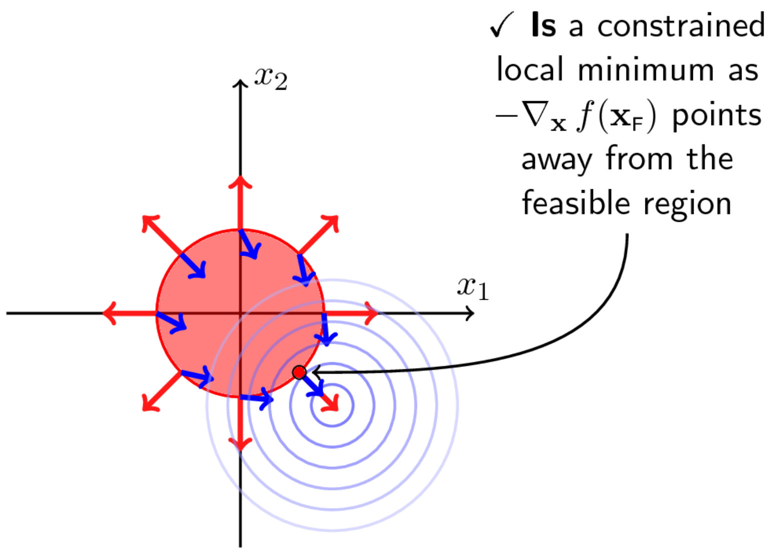
Want a constrained local minimum...


✗ **Not** a constrained local minimum as $-\nabla_{\mathbf{x}} f(\mathbf{x}_F)$ points in towards the feasible region
x_2
x_1

$\therefore$ Constrained local minimum occurs when $-\nabla_{\mathbf{x}} f(\mathbf{x})$ and $\nabla_{\mathbf{x}} g(\mathbf{x})$ point in the same direction:

$$\boxed{-\nabla_{\mathbf{x}} f(\mathbf{x}) = \lambda \nabla_{\mathbf{x}} g(\mathbf{x}) \quad \textbf{and} \quad \lambda > 0}$$

Want a constrained local minimum...


✓ **Is** a constrained local minimum as $-\nabla_{\mathbf{x}} f(\mathbf{x}_F)$ points away from the feasible region
x_2
x_1

$\therefore$ Constrained local minimum occurs when $-\nabla_{\mathbf{x}} f(\mathbf{x})$ and $\nabla_{\mathbf{x}} g(\mathbf{x})$ point in the same direction:

$$\boxed{-\nabla_{\mathbf{x}} f(\mathbf{x}) = \lambda \nabla_{\mathbf{x}} g(\mathbf{x}) \quad \textbf{and} \quad \lambda > 0}$$

So, we have a problem:

$$f(x) \to \min_{x \in \mathbb{R}^n}$$
$$\text{s.t. } g(x) \leq 0$$

Two possible cases:

$g(x^*) < 0$
1. $\nabla f(x^*) = 0$
$\nabla^2 f(x^*) > 0$
$g(x^*) = 0$
2. $-\nabla f(x^*) = \mu \nabla g(x^*), \quad \mu > 0$
$\langle y, \nabla^2_{xx} L(x^*, \mu^*) y\rangle \geq 0, \quad \forall y \in \mathbb{R}^n : \nabla g(x^*)^\top y = 0$

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

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

Let's define the Lagrange function:

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

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

$$\begin{split} & (1) \ \nabla_x L(x^*, \mu^*) = 0 \\ & (2) \ \mu^* \geq 0 \\ & (3) \ \mu^* g(x^*) = 0 \\ & (4) \ g(x^*) \leq 0 \\ & (5) \ \langle y, \nabla^2_{xx} L(x^*, \mu^*) y\rangle \geq 0, \quad \forall y \in \mathbb{R}^n : \nabla g(x^*)^\top y = 0 \end{split}$$

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

General formulation

$$\begin{split} & f(x) \to \min_{x \in \mathbb{R}^n} \\ \text{s.t. } & g_i(x) \leq 0, \ i = 1, \ldots, m \\ & h_j(x) = 0, \ j = 1, \ldots, p \end{split}$$

This formulation is a general problem of mathematical programming. From now, we only consider $\textbf{regular}$ tasks. This is a very important remark from a formal point of view. Those wishing to understand in more detail, please refer to Google.

Solution

$$L(x, \mu, \lambda) = f(x) + \sum\limits_{j=1}^p \lambda_j h_j(x) + \sum\limits_{i=1}^m \mu_i g_i(x)$$

Karush-Kuhn-Tucker conditions

Let x^* be a solution to a mathematical programming problem, and the functions f, h_j, g_i are differentiable. Then there are λ^* and μ^* such that the following conditions are carried out:

- $\nabla_x L(x^*, \lambda^*, \mu^*) = 0$
- $\nabla_\lambda L(x^*, \lambda^*, \mu^*) = 0$
- $\mu^*_j \geq 0$
- $\mu^*_j g_j(x^*) = 0$
- $g_j(x^*) \leq 0$

These conditions are sufficient if the problem is regular, i.e. if:

1. the given problem is a convex optimization problem (i.e., the functions f and g_i are convex, h_i are affine) and the Slater condition is satisfied; or
2. strong duality is fulfilled.

References

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