Lecture 1

Modern optimization methods

Classical problem of optimization methods

Mathematical programming problem:

$$f(x, y) \rightarrow max$$
$$g(x, y) \leq C$$
$$x, y \geq 0$$

Examples of problems

1. Diet Problem (linear programming problem)

For medical reasons, a person is prescribed to follow a certain daily diet and ensure that the amount of nutrients in the products meets medical requirements. There are three types of products *P1, P2, P3*, which contain certain nutrients *NUT1* and *NUT2*. The content of nutrients in these products (mg) is known (a_{ij}, where *i* is responsible for the *i*-th nutrient, *j* is for the *j*-th product, $i = 1,2$; $j = 1,2,3$) and the minimum need of the body for nutrients (mg) ($b_i, i = 1,2$). The cost of one unit of each type of product is also known ($c_j, j = 1,2,3$).

It is necessary to draw up a person's nutrition plan as follows: all medical requirements must be taken into account and the cost of the food basket must be the lowest. That is, to determine in what quantity a particular product should be purchased, taking into account the existing restrictions.

Examples of problems

2. The problem of optimal use of resources (linear programming problem)

A factory produces three types of products: *P1, P2, P3* (for example, a furniture factory can produce chairs, tables, sofas). Three different types of resources are spent on their production: *R1, R2, R3* (for example, wood, fabric, electricity). The resource stocks for this factory are known b_1, b_2, b_3, where $b_i, i = 1,2,3$ is the stock of the *i*-th type of resource. The number of units of each type of raw material required for the production of each type of product $a_{ij}, i, j = 1,2,3$ is also known, that is, the number of units of the i-th type of resource required for the production of the *j*-th type of product. The profit from the sale of the *j*-th type of product is known and equals $c_j, j = 1,2,3$.

It is required to draw up a production plan that allows maximizing the enterprise's profit, taking into account the existing constraints on resource stocks.

Examples of problems

3. Transport problem (linear programming problem)

The transportation problem is a linear programming problem and its purpose is to determine the most economical plan for transporting homogeneous goods from points of departure (warehouses) to points of destination (stores).

It is necessary to determine the minimum cost of transporting the entire goods

Examples of problems

4. The traveling salesman problem

In the theory of computational complexity, the **travelling salesman problem** (**TSP**) asks the following question:

"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

Examples of problems

5. Optimization algorithms on graphs

Algorithms for finding shortest paths; maximum flow problem (a flow defines a way to transfer certain objects from one point to another).

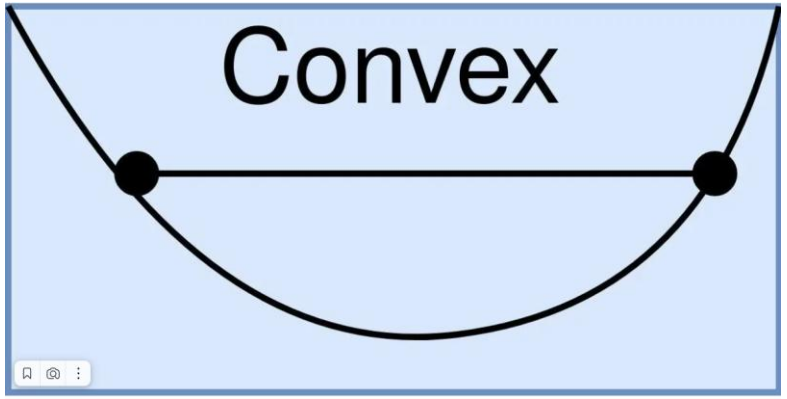
Examples of problems

6. Modern optimization methods in machine learning

Optimization algorithms are the backbone of machine learning models as they enable the modeling process to learn from a given data set. These algorithms are used in order to find the minimum or maximum of an objective function which in machine learning context stands for error or loss.

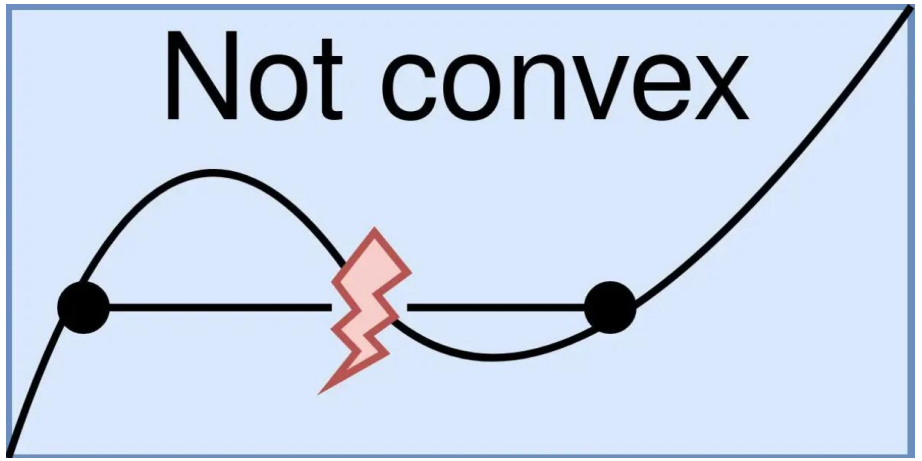
Types of functions used in optimization

1. Convex functions. Convex functions are functions that have one global minimum. A function is called convex if the segment connecting any two points on its graph lies entirely above the graph.



Types of functions used in optimization

2. Non-convex functions. Non-convex functions are functions that have several local minima. The graph of a non-convex function has multiple local minima and/or maxima. It is not guaranteed to have a single global minimum or maximum.



Types of functions used in optimization

3. Limited functions. These functions are unique functions because they have restrictions on their input variables. Limited functions are mathematical expressions that are subject to certain restrictions or rules. These constraints can take the form of equalities or inequalities that the function's inputs and outputs must satisfy.

$$\boldsymbol{Objective\ function} = \max f(x)$$

$$where\ f(x) = x_1^2 + x_2^3$$
$$and\ \ |x_1 + x_2| < 6$$

Optimization algorithms

Optimization algorithms are methods used to find the optimal solution to an optimization problem, which typically involves finding input values that minimize or maximize an objective function.

There are different types of optimization algorithms, including those for convex, non-convex, and constrained functions.

Optimization of convex functions

Convex optimization algorithms are used to solve problems with convex objective functions. These methods are computationally efficient and can find the global optimum.

Gradient descent. Gradient descent is an optimization approach that updates model parameters based on the gradient of an objective function.

Newton's Method. Newton's method is an optimization algorithm used to find local minima and maxima of a function. This is a second-order approach, which means that it uses the gradient of the function and the Hessian to locally approximate the function and find the best solution.

Optimization of non-convex functions

-used to solve problems with non-convex objective functions

-these algorithms cannot be guaranteed to achieve a global optimum, they can be used to find good solutions.

Examples: The SGD, Momentum and Genetic algorithms.

Optimization of non-convex functions

Stochastic Gradient Descent (SGD). SGD is a version of gradient descent that updates parameters using a random subset of the training data, called a mini-batch. This can help speed up the optimization process and reduce the likelihood of hitting a local minimum.

Adagrad. Adagrad is an optimization method that adjusts the learning rate for each parameter based on the previous gradient. This can help in the optimization process for non-convex functions.

Genetic algorithm (GA). GA is a method based on the biological process of natural selection. It uses the concepts of selection, crossover and mutation to develop new solutions and determine which one is best.

Optimization of limited functions

-used to solve problems with input variable constraints

-used to determine the best solution given the constraints

Examples: interior point approach, the simplex method, the Lagrange multiplier method, and sequential quadratic programming.

Optimization of limited functions

Interior Point Method used to solve linear and nonlinear programming problems. It is based on the idea of finding the best solution by choosing a path within a feasible zone rather than along its boundary.

Simplex method used for solving linear programming problems. It is based on the concept of moving from one plausible solution to another by making minor changes to the current solution.

Optimization of limited functions

Lagrange Multiplier Method: used for determining local maxima and minima of a function subject to constraints. It is based on the introduction of a set of auxiliary variables, known as Lagrange multipliers, which are used to express constraints.

Sequential Quadratic Programming (SQP): used to solve nonlinear programming problems. It works by approximating a nonlinear objective function and constraints with a quadratic function and then iteratively solving the resulting quadratic programming problem.

Examples of problems solved by modern optimization methods

Supply Chain Optimization: Companies use optimization algorithms to minimize costs and maximize efficiency in their supply chains. This includes optimizing inventory levels, transportation routes, and production schedules.

Machine Learning Model Training: Optimization techniques, such as gradient descent, are used to minimize the loss function in machine learning models, ensuring that predictions are as accurate as possible.

Portfolio Optimization: In finance, optimization methods help investors select the best portfolio of assets that maximizes returns for a given level of risk, often using methods like the Markowitz Efficient Frontier.

Examples of problems solved by modern optimization methods

Scheduling Problems: Various industries, including airlines and manufacturing, use optimization to create efficient schedules for employees, machines, and flights, ensuring that resources are utilized effectively.

Network Design: Telecommunications companies use optimization to design networks that minimize costs while maximizing coverage and performance, balancing factors like bandwidth and latency.

Energy Management: In renewable energy, optimization algorithms are used to manage energy distribution, maximizing the use of renewable sources while minimizing costs and emissions.

Examples of problems solved by modern optimization methods

Transportation and Logistics: Companies like Uber and FedEx use route optimization algorithms to determine the most efficient paths for delivery and transportation, reducing fuel costs and delivery times.

Healthcare Resource Allocation: Hospitals use optimization techniques to allocate resources like staff and equipment effectively, improving patient care while controlling costs.

Manufacturing Process Optimization: Factories apply optimization to streamline production processes, reduce waste, and improve product quality by adjusting variables such as machine settings and workforce allocation.

Urban Planning: City planners use optimization methods to design efficient public transportation systems, optimize land use, and improve infrastructure, enhancing overall urban living conditions.

Classical problem of optimization methods

Mathematical programming problem:

$$f(x, y) \rightarrow max$$
$$g(x, y) \leq C$$
$$x, y \geq 0$$

Nonlinear programming problems can be divided into two large classes: **unconditional optimization problems** and **conditional optimization problems.**

Classical problem of optimization methods

Vector $x = (x_1, x_2, \dots, x_n)$, satisfying the constraints of the problem is called a **feasible solution**.

The set of feasible solutions to the problem is called the **feasible region**.

A feasible solution that provides the optimal value of the objective function is called an **optimal solution** and is denoted by x^*.

Unconstrained minimization problem

$$f(x) \to min, x \in S, S \subset \mathbb{R}^n$$

where $f(x)$ is a function of several variables.

A point $x^* \in S$ is called a **point of global minimum** of the function $f(x)$ on the set S if $f(x^*) \leq f(x)$ for all $x \in S$.

A point $x^* \in S$ is called a **point of strict global minimum** of the function $f(x)$ on the set S if $f(x^*) \leq f(x)$ for all $x \in S$.

Unconstrained minimization problem

A point $x^* \in S$ is called a **point of local minimum** of the function $f(x)$ on the set S if $f(x^*) \leq f(x)$ for all $x \in S \cap V_\varepsilon(x^*)$, where $V_\varepsilon(x^*) = \{x \in R^n : \|x - x^*\| \leq \varepsilon\}$ is a ball of radius $\varepsilon > 0$ with center at the point x^* (the ε-neighborhood of the point x^*).

A point $x^* \in S$ is called a **point of strict local minimum** of the function $f(x)$ on the set S if $f(x^*) < f(x)$ for all $x \in S \cap V_\varepsilon(x^*)$, where $V_\varepsilon(x^*) = \{x \in R^n : \|x - x^*\| \leq \varepsilon\}$ is a ball of radius $\varepsilon > 0$ with center at the point x^* (the ε-neighborhood of the point x^*).

Unconstrained minimization problem

The **gradient** of a function $f(x)$ is a vector $f'(x)$ whose direction indicates the direction of greatest increase of the function.

The **Hessian matrix** $H(x)$ of a function $f(x)$ that is twice continuously differentiable at a point x is a symmetric square matrix that describes the behavior of the function in the second order.

Unconstrained minimization problem

The **Hessian matrix** $H(x)$ of a function $f(x)$ that is twice continuously differentiable at a point x is a symmetric square matrix that describes the behavior of the function in the second order.

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \cdots & \cdots & & \cdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Unconstrained minimization problem

Theorem 1 (necessary condition for a local minimum). Let $f(x)$ be a function differentiable at a point $x^* \in \mathbb{R}^n$. If x^* is a local minimum point, then $f'(x^*) = 0$.

Theorem 2 (sufficient condition for a local minimum). Let $f(x)$ be a twice differentiable function at a point $x^* \in \mathbb{R}^n$. If $f'(x^*) = 0$ and the Hessian matrix $H(x)$ is positive definite, then x^* is a local minimum point.

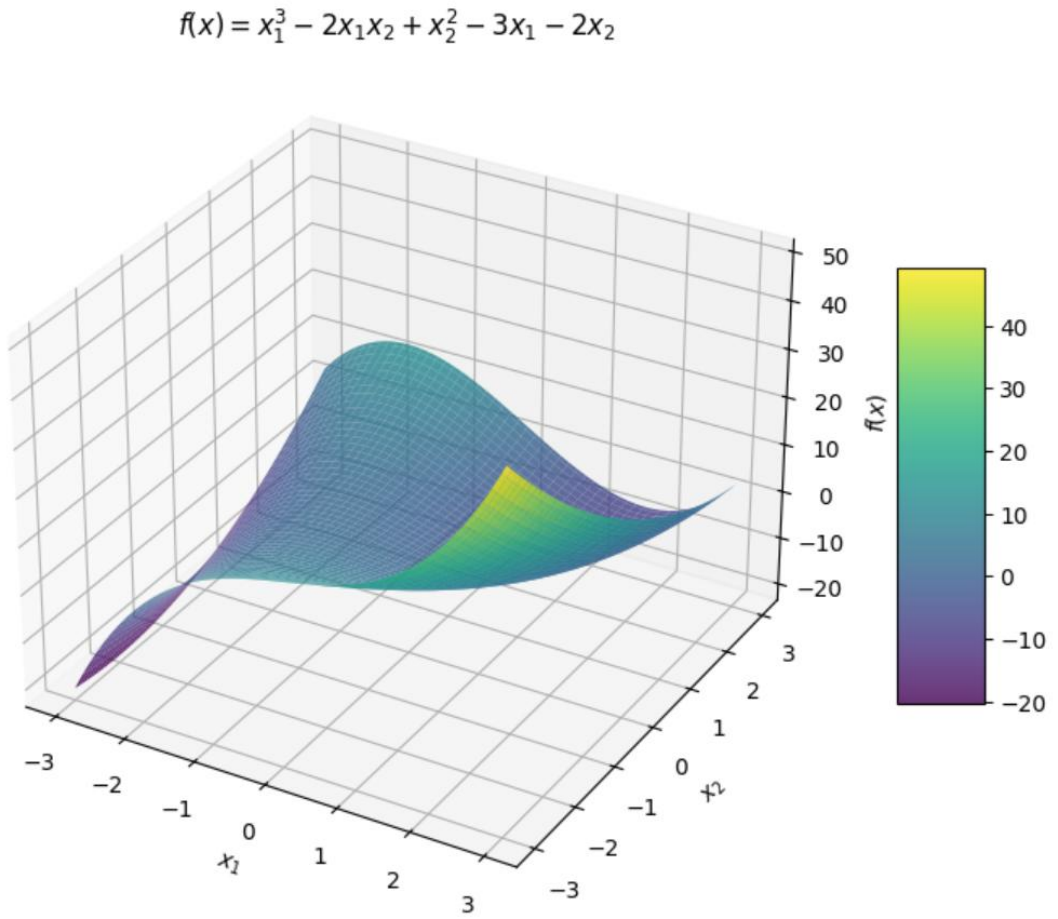
Unconstrained minimization problem

Theorem 3 (Sylvester criterion). For a matrix to be positive definite, it is necessary and sufficient that all leading principal minors of the matrix be positive.

Leading principal minor of order *k* is a determinant composed of elements of a matrix of quadratic form located at the intersection of the first *k* rows and *k* columns of the given matrix.

Example

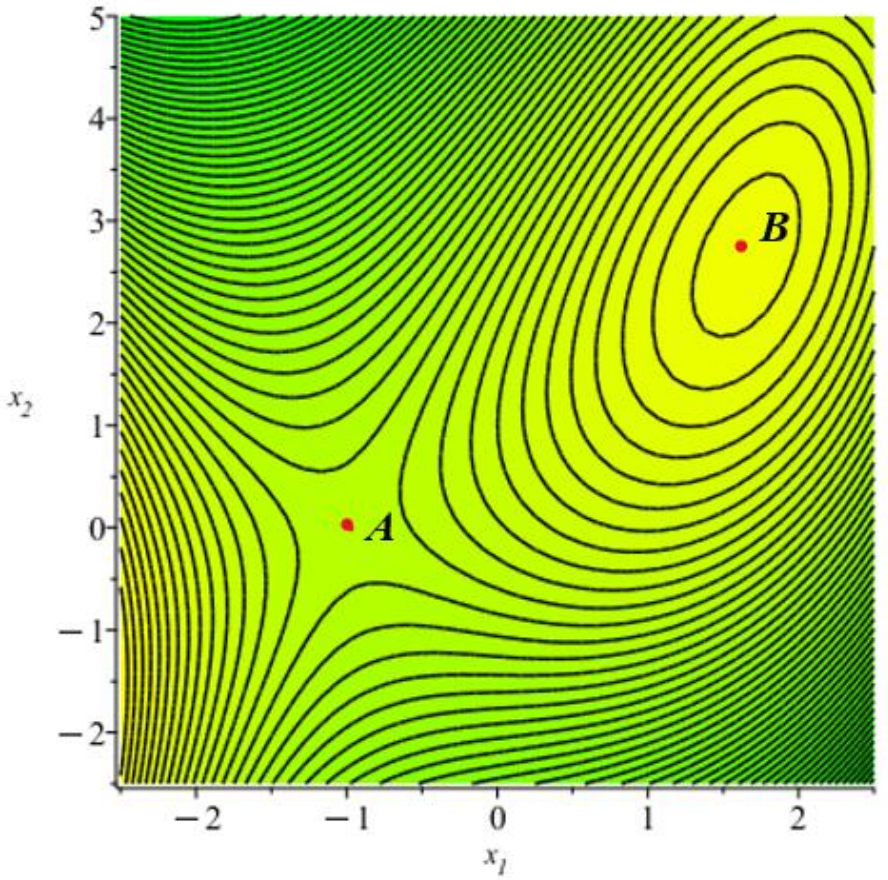
Find the local minimum of the function:
$f(x) = x_1^3 - 2x_1x_2 + x_2^2 - 3x_1 - 2x_2$



Example

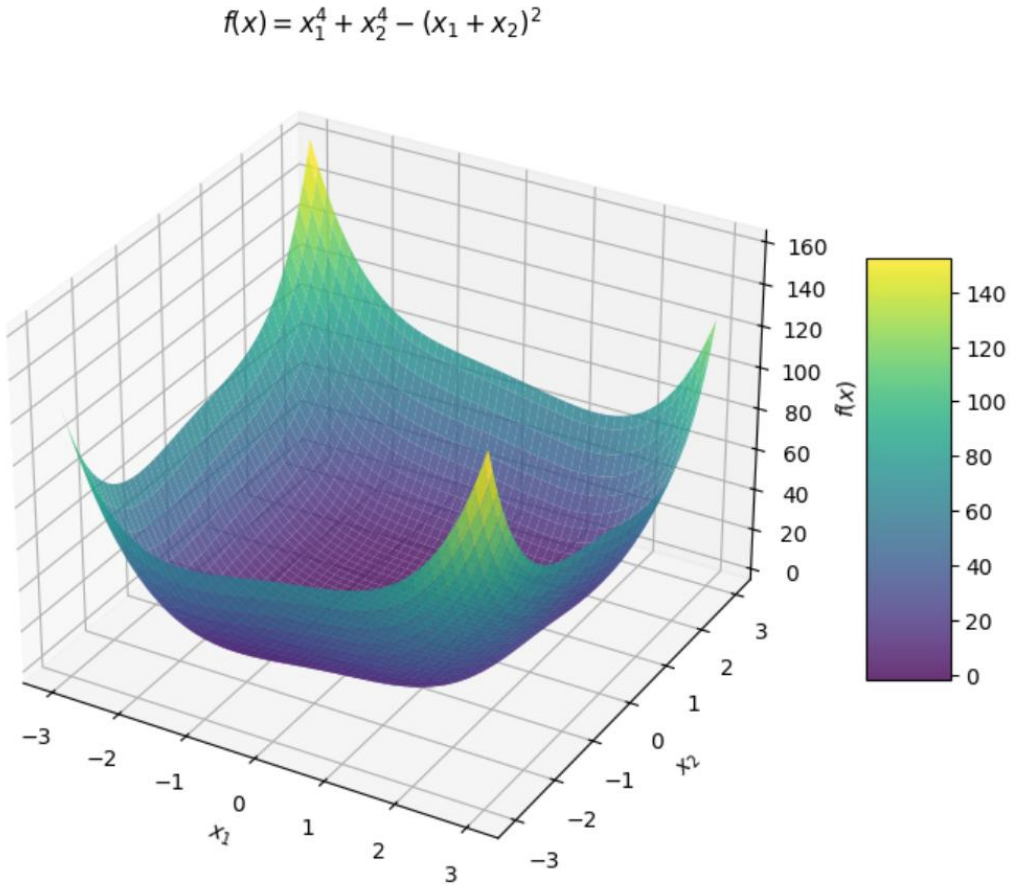
Find the local minimum of the function:
$f(x) = x_1^3 - 2x_1x_2 + x_2^2 - 3x_1 - 2x_2$

The first color value (yellow) corresponds to the smallest the second (green) to the largest.



Example

Find the local minimum of the function:
$f(x) = x_1^4 + x_2^4 - (x_1 + x_2)^2$

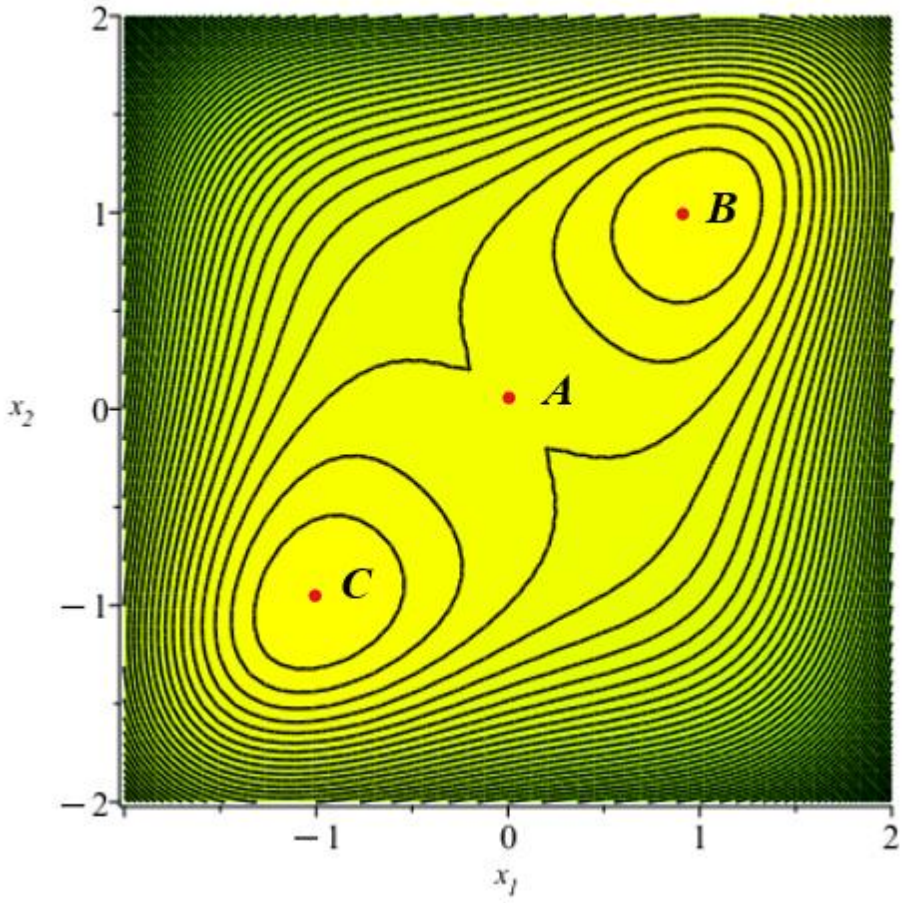


Example

Find the local minimum of the function:
$f(x) = x_1^4 + x_2^4 - (x_1 + x_2)^2$

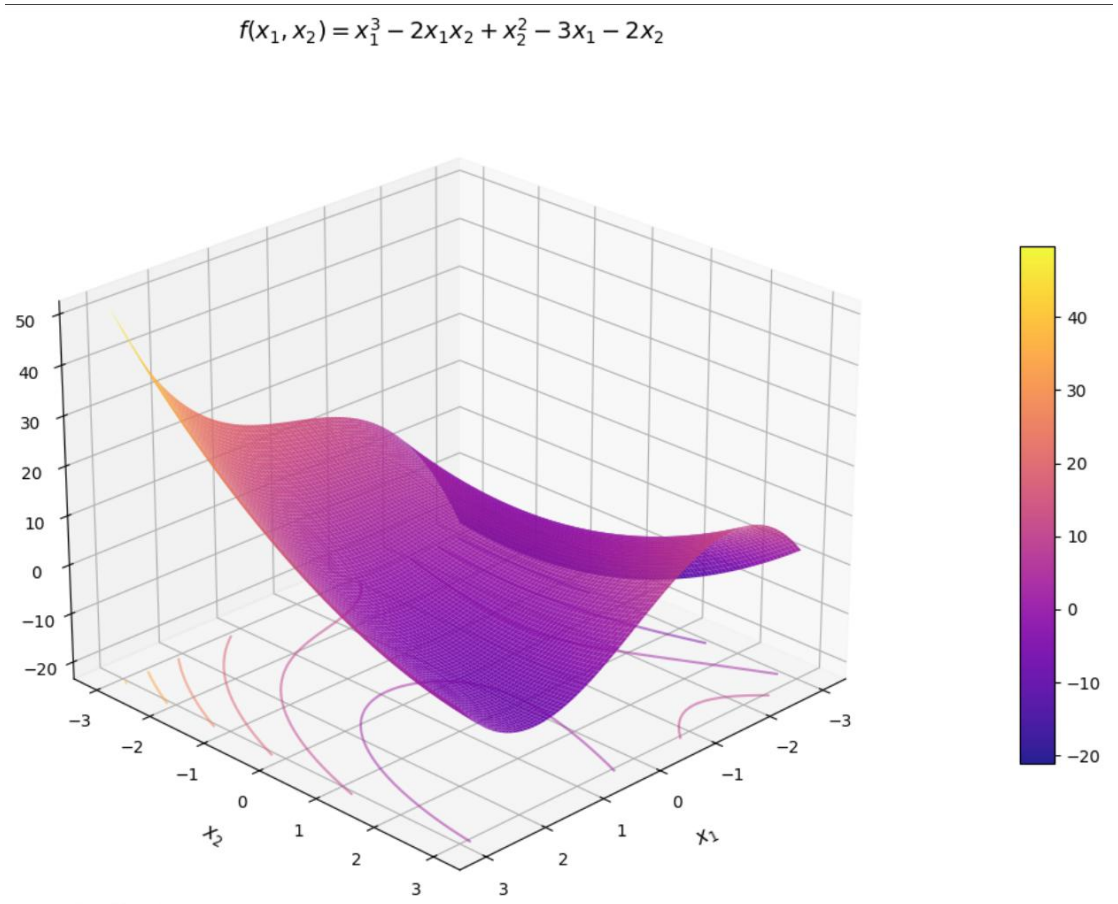
The first color value (yellow) corresponds to the

smallest value of the function, the second (green) to the



Example

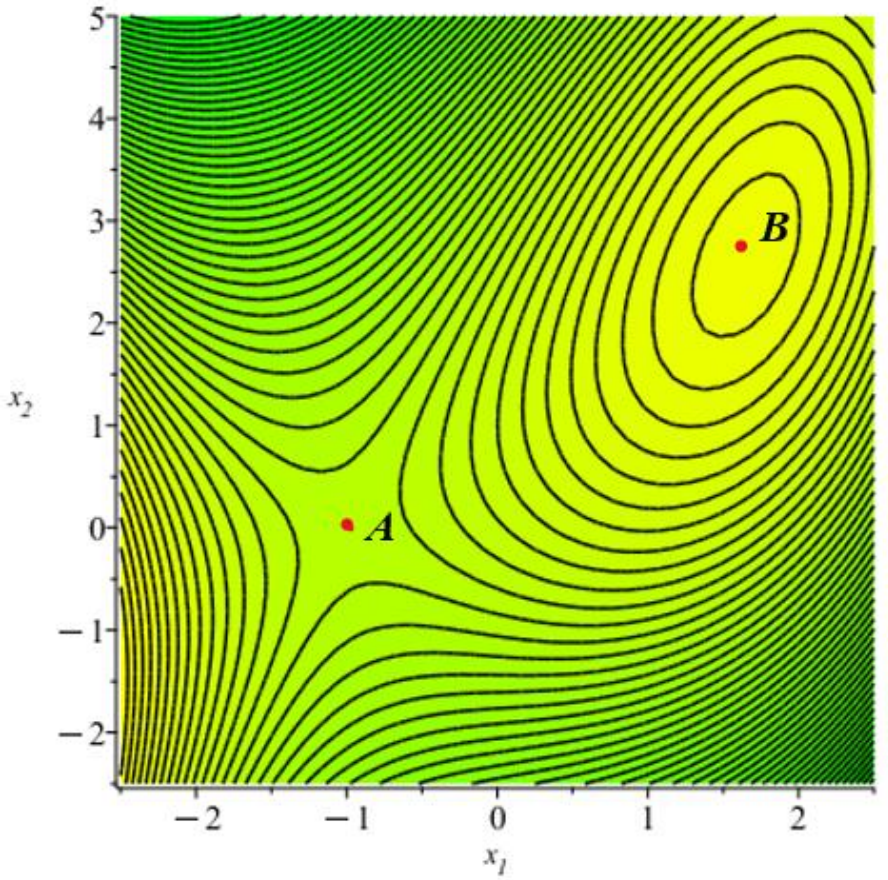
Find the local minimum of the function:
$f(x) = x_1^3 - 2x_1x_2 + x_2^2 - 3x_1 - 2x_2$



Example

Find the local minimum of the function:
$f(x) = x_1^3 - 2x_1x_2 + x_2^2 - 3x_1 - 2x_2$

The first color value (yellow) corresponds to the smallest the second (green) to the largest.



Example (math). Python

Find the local minimum of the function:
$f(x) = x_1^3 - 2x_1x_2 + x_2^2 - 3x_1 - 2x_2$

1. Partial derivatives

```
import sympy as sp

x, y = sp.symbols('x y')

df_dx = sp.diff(f, x)
print("Partial derivative with respect to x:", df_dx)


df_dy = sp.diff(f, y)
print("Partial derivative with respect to y:", df_dy)
```

```
Partial derivative with respect to x: 3*x**2 - 2*y - 3
Partial derivative with respect to y: -2*x + 2*y - 2
```

```python
from scipy.optimize import fsolve
import numpy as np
import sympy as sp

def f(x, y):
    return x**3-2*x*y+y**2-3*x-2*y

def df_dx(x, y):
    return 3*x**2 - 2*y - 3

def df_dy(x, y):
    return -2*x + 2*y - 2

def equations(p):
    x, y = p
    eq1 = df_dx(x, y)
    eq2 = df_dy(x, y)
    return [eq1, eq2]

initial_guess = [1, 1]


solution = fsolve(equations, initial_guess)


print(solution)
```

```
[1.66666667 2.66666667]
```

```python
from scipy.optimize import fsolve
import numpy as np
import sympy as sp

def f(x, y):
    return x**3-2*x*y+y**2-3*x-2*y

def df_dx(x, y):
    return 3*x**2 - 2*y - 3

def df_dy(x, y):
    return -2*x + 2*y - 2

def equations(p):
    x, y = p
    eq1 = df_dx(x, y)
    eq2 = df_dy(x, y)
    return [eq1, eq2]

initial_guess = [-1, 0]


solution = fsolve(equations, initial_guess)


print(solution)
```

```
[-1.  0.]
```

Example (math). Python

Find the local minimum of the function:
$f(x) = x_1^3 - 2x_1x_2 + x_2^2 - 3x_1 - 2x_2$

2. Hessian matrix

```
print("Hessian matrix:")
sp.pprint(hessian_matrix)
```

```
Hessian matrix:
⎡6⋅x  -2⎤
⎢      ⎥
⎣-2   2 ⎦
```

```
import sympy as sp

x, y = sp.symbols('x y')

f = x**3-2*x*y+y**2-3*x-2*y

hess_xx = sp.diff(f, x, 2)
hess_xy = hess_yx = sp.diff(f, x, y)
hess_yy = sp.diff(f, y, 2)


hessian_matrix = sp.Matrix([
    [hess_xx, hess_xy],
    [hess_yx, hess_yy]
])
```

Example (math). Python

Find the local minimum of the function:
$f(x) = x_1^3 - 2x_1x_2 + x_2^2 - 3x_1 - 2x_2$

2. Hessian matrix in point (-1;0).

Eigenvalues of the matrix

```
Hessian matrix:
⎡-6  -2⎤
⎢      ⎥
⎣-2   2⎦
```

```python
import numpy as np


matrix = np.array([[-6,-2], [-2,2]])

eigenvalues = np.linalg.eigvalsh(matrix)

if all(eigenvalues > 0):
    print("The matrix is positive definite.")
else:
    print("The matrix is not positive definite.")
```

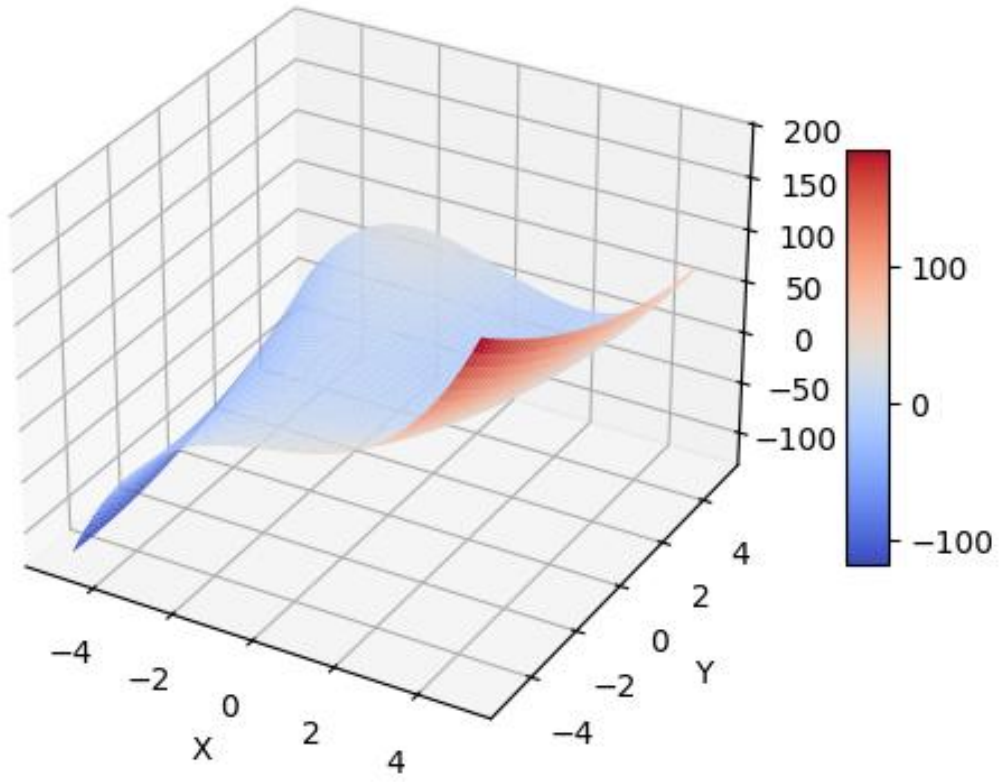
```
The matrix is not positive definite.
```

Find the local minimum of the function:
$f(x) = x_1^3 - 2x_1x_2 + x_2^2 - 3x_1 - 2x_2$

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X, Y = np.meshgrid(x, y)


Z = X**3-2*X*Y+Y**2-3*X-2*Y


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')


surf = ax.plot_surface(X, Y, Z, cmap=plt.cm.coolwarm)


fig.colorbar(surf, shrink=0.5, aspect=10)


ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')


plt.show()
```

Example (SciPy). Python

Find the local minimum of the function:
$f(x) = x_1^3 - 2x_1x_2 + x_2^2 - 3x_1 - 2x_2$



Example (SciPy). Python

Find the local minimum of the function:

$f(x) = x_1^3 - 2x_1x_2 + x_2^2 - 3x_1 - 2x_2$

```
def function_wrapper(z):
    X, Y = z
    return X**3-2*X*Y+Y**2-3*X-2*Y

from scipy.optimize import minimize
x0 = np.array([1, 1])
res = minimize(function_wrapper, x0, method='nelder-mead',
    options={'xtol': 1e-8, 'disp': True})
print(res.x)
```

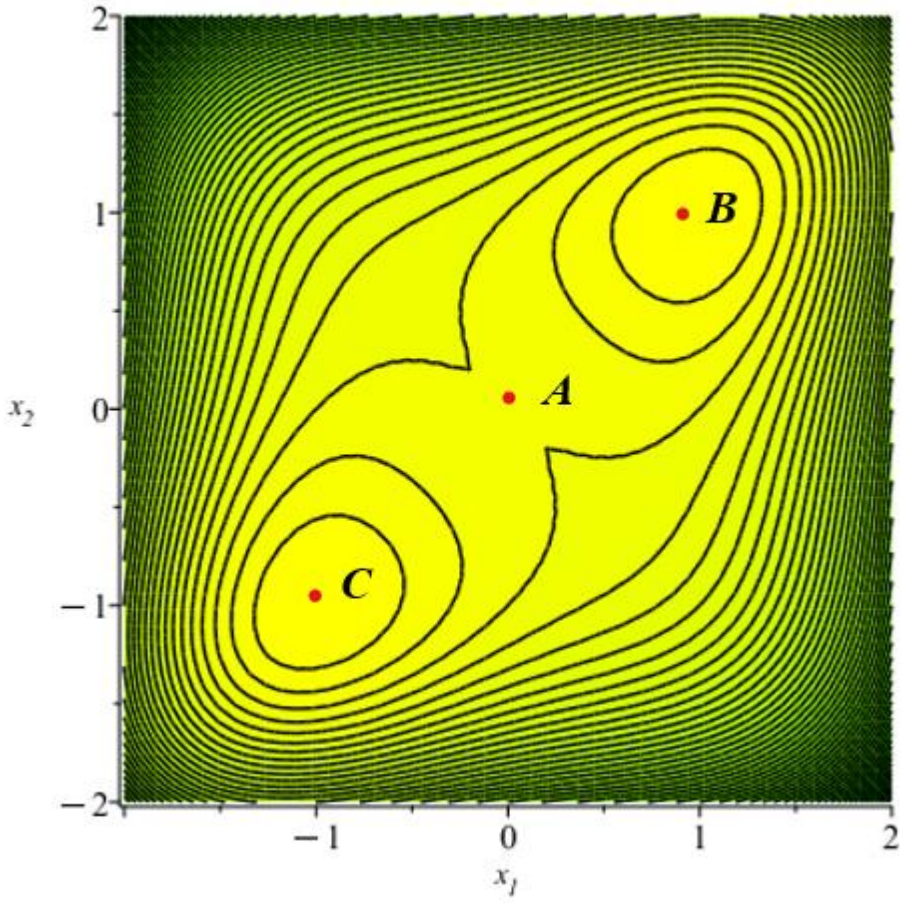
```
Optimization terminated successfully.
         Current function value: -7.481481
         Iterations: 38
         Function evaluations: 73
[1.66666233 2.66668387]
<ipython-input-83-071759682f21>:7: OptimizeWarning: Unknown solver options: xtol
  res = minimize(function_wrapper, x0, method='nelder-mead',
```

Example

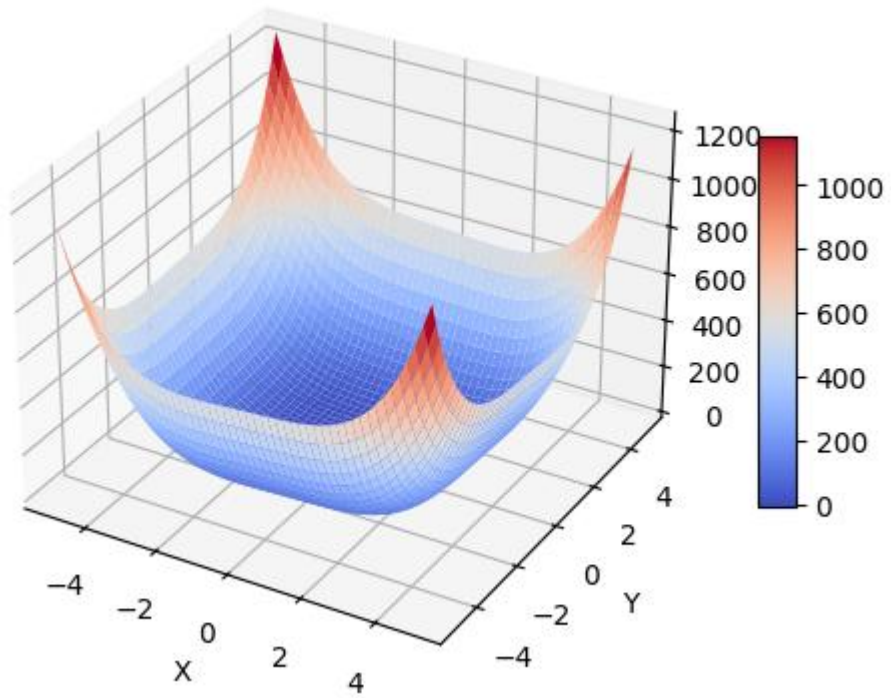
Find the local minimum of the function:
$f(x) = x_1^4 + x_2^4 - (x_1 + x_2)^2$

The first color value (yellow) corresponds to the

smallest value of the function, the second (green) to the



```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X, Y = np.meshgrid(x, y)


Z = X**4+Y**4-(X+Y)**2


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')


surf = ax.plot_surface(X, Y, Z, cmap=plt.cm.coolwarm)


fig.colorbar(surf, shrink=0.5, aspect=10)


ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')


plt.show()
```



Example

```
def function_wrapper(z):
    X, Y = z
    return X**4+Y**4-(X+Y)**2

from scipy.optimize import minimize
x0 = np.array([1, 1])
res = minimize(function_wrapper, x0, method='nelder-mead',
    options={'xtol': 1e-8, 'disp': True})
print(res.x)
```

```
Optimization terminated successfully.
         Current function value: -2.000000
         Iterations: 20
         Function evaluations: 41
[1. 1.]
<ipython-input-85-3f8dbdd000a2>:7: OptimizeWarning: Unknown solver options: xtol
  res = minimize(function_wrapper, x0, method='nelder-mead',
```

Constrained minimization problem

Let us consider the problem of nonlinear programming in general form:

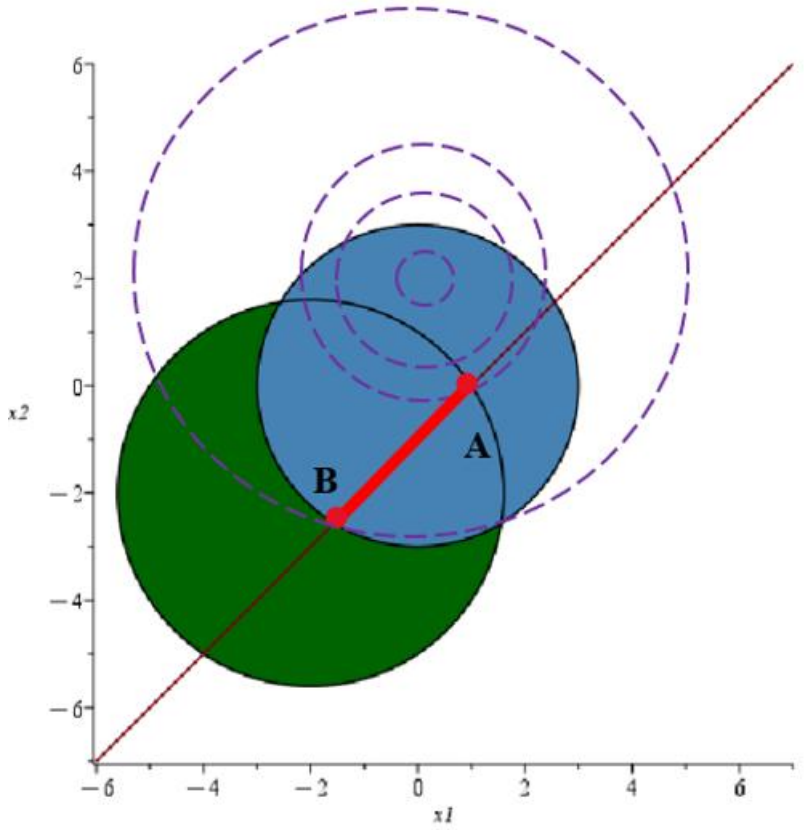
$$f(x) \to \min(\max)$$
$$g_i(x) \le 0, i = 1, \dots, m$$
$$h_i(x) = 0, i = 1, \dots, k$$

For example, task:

$$f(x) = x_1^2 + x_2^2 - 4x_2 \to min$$
$$\begin{cases} x_1^2 + x_2^2 \le 9 \\ x_1^2 + 4x_1 + x_2^2 + 4x_2 \le 5 \\ x_1 - x_2 = 1 \end{cases}$$

Constrained minimization problem. Graphic solution

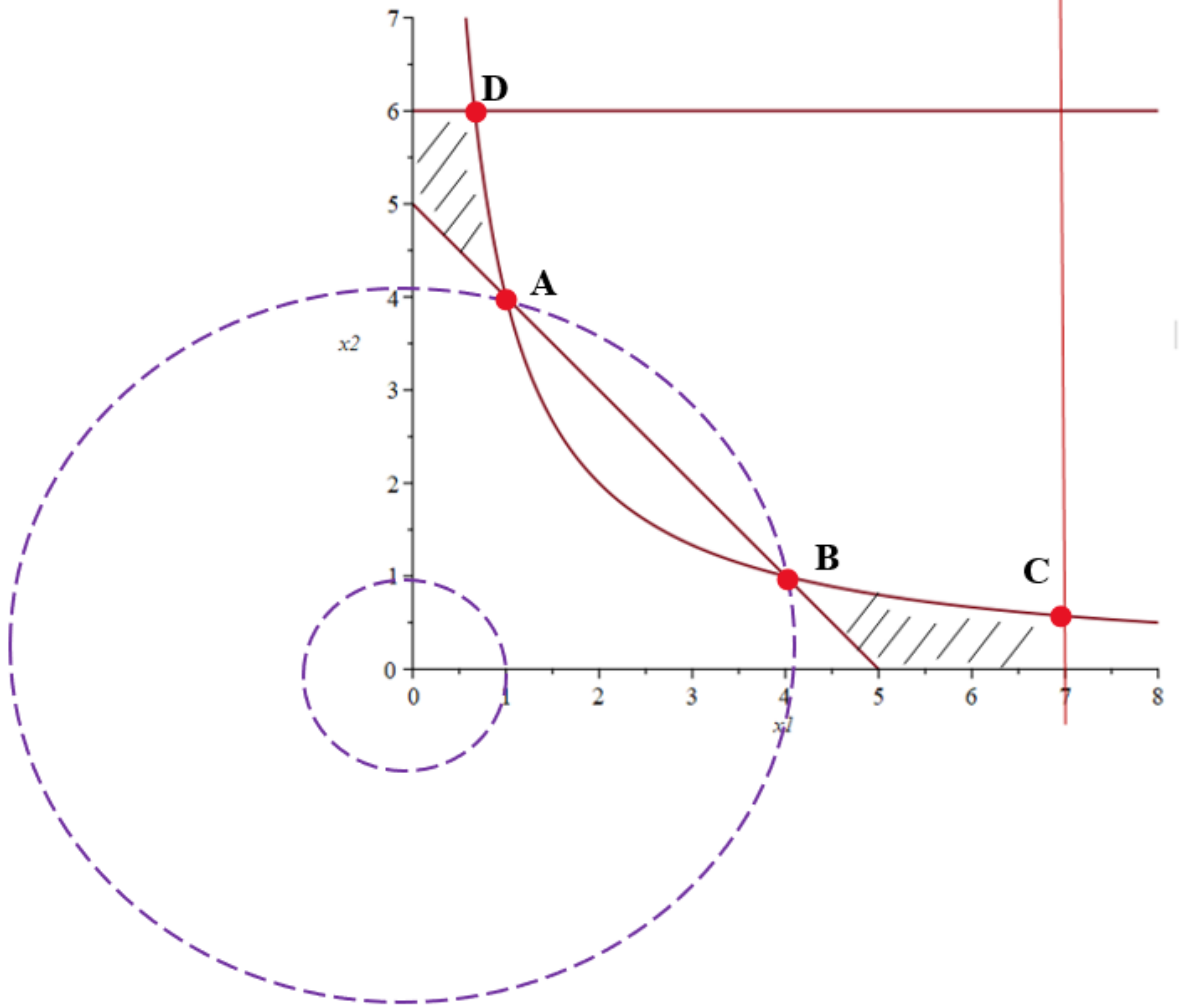
In the case of two variables, the problem can be solved graphically. It is necessary to show the constraints and level lines of the objective function



Constrained minimization problem. Graphic solution

$$f(x) = x_1^2 + x_2^2 \to min$$

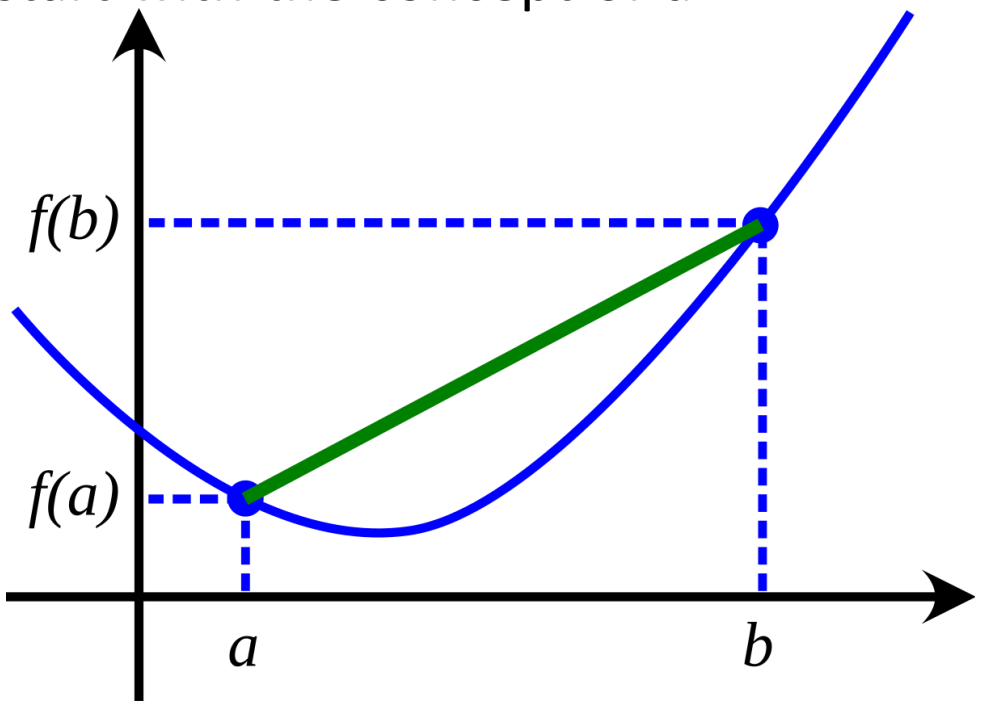
$$\begin{cases} x_1 x_2 \le 4 \\ x_1 + x_2 \ge 5 \\ x_1 \le 7 \\ x_2 \le 6 \\ x \ge 0 \end{cases}$$



Convex functions. Testing a function for convexity

Nonlinear programming problems with convex functions are well studied and belong to the section of convex programming, so let's start with the concept of a convex function.

In mathematics, a real-valued function is called **convex** if the line segment between any two distinct points on the graph of the function lies above or on the graph between the two points



Convex functions. Testing a function for convexity

Algorithm for checking a function for convexity.

1. Construct the Hessian matrix for the function in question.
2. Calculate the principal minors of the Hessian matrix. If all principal minors are non-negative, then conclude that the function is convex.

The principal minor of the *k*-th order is the determinant composed of elements of a matrix of quadratic form located at the intersection of *k* rows and *k* columns of the given matrix with the same numbers.

Convex programming problem

$$f(x) \to min$$

$$\begin{cases} g_i(x) \leq 0, i = 1, \dots, m \\ x \geq 0 \end{cases}$$

where $f(x)$ and $g_i(x), i = 1, \dots, m$ are convex

The **Lagrange function** of a convex programming problem is the function

$$L(x, y) = f(x) + \sum_{i=1}^{m} g_i(x)\lambda_i$$

$where \lambda_i, i = 1, \dots, m -$ Lagrange multipliers

Convex programming problem

A point $(x^*, \lambda^*), x^* \geq 0, \lambda^* \geq 0$ is called a **saddle point of the Lagrange function** if:

$$L(x^*, \lambda) \leq L(x^*, \lambda^*) \leq (x, \lambda^*), \forall x, \lambda \geq 0$$

1st Kuhn-Tucker theorem. If (x^*, λ^*) is a saddle point of the Lagrange function, then x^* is an optimal solution to the convex programming problem.

2nd Kuhn-Tucker theorem. For any optimal solution x^* of a regular convex programming problem, there exists a vector of Lagrange multipliers λ^*: (x^*, λ^*) is a saddle point of the Lagrange function.

Classification of optimization problems

Classification of optimization problems:

-Unconstrained optimization;
-LP (linear programming);
-MILP (mixed integer linear programming);
-NLP (nonlinear programming);
-MINLP (mixed integer nonlinear programming)

SciPy

Contains a large set of functions for scientific computing, including tools for solving optimization problems, which are located in the **scipy.optimize** module.

This module provides methods for solving both nonlinear programming (NLP) and linear programming (LP) problems, including mixed integer linear programming (MILP) problems.

SciPy

scipy.optimize module includes the implementation of the following routines:

-Constrained and unconstrained minimization of scalar functions of several variables (minim) using various algorithms (Nelder-Mead simplex, BFGS, conjugate gradient Newton, COBYLA and SLSQP);
-Global optimization (e.g.: basinhopping, diff_evolution);
-Minimization of residuals of least squares (least_squares) and nonlinear curve fitting algorithms of least squares (curve_fit);
-Minimization of scalar functions of one variable (minim_scalar) and root finding (root_scalar);
-Multivariate solvers of a system of equations (root) using various algorithms (hybrid Powell, Levenberg-Marquardt or large-scale methods such as Newton-Krylov).

SciPy. Solving the MILP problem

To solve linear programming problems, the optimize submodule has a **linprog** function. HiGHS is used by default as a solver for LP(MILP) — it implements the simplex method (highs-ds) and the interior point method (highs-ipm). When starting the solution, one of the methods is selected by default; the ability to force a method is also present.

$$f(x) = 3x_1 + 3x_2 \to \min$$
$$x_1 + x_2 \leq 8$$
$$2x_1 - x_2 \geq 1$$
$$x_1 - 2x_2 \leq 2$$
$$x_1 \geq 0, x_2 \geq 0$$

SciPy. Solving the MILP problem

```
from scipy.optimize import linprog

c =  np.array([3., 3.])
A_ub = np.array([[1., 1.],[-2., 1.],[1., -2.]])
b_ub = np.array([8.0,-1.0,2.0])

bounds = [(0, np.inf), (0, np.inf)]
res_milp = linprog(c=c, A_ub=A_ub, b_ub=b_ub, bounds=bounds, method='highs')

print(f"Solution: x = {list(np.round(res_milp['x'], 2))}, f = {-res_milp['fun']}, {res_milp['message']}")
```

```
Solution: x = [3.0, 5.0], f = 24.0, Optimization terminated successfully. (HiGHS Status 7: Optimal)
```

SciPy. Solving the MILP problem

The scipy.optimize.linprog library provides several methods for solving linear programming problems. Each of them has its own features, advantages, and limitations. Here are the main methods:

1. ‘highs’

Advantages:

High performance.

Suitable for large and complex problems.

Supports both minimization and maximization problems.

Disadvantages:

Requires installation of additional dependencies (included in the standard scipy package since version 1.6.0).

Recommendations: Use this method by default if you have a modern version of scipy.

SciPy. Solving the MILP problem

2. 'simplex’

Description: The classic simplex method implemented in scipy. This is one of the first methods available in linprog.

Advantages:

Simple and robust.

Works well for small problems.

Disadvantages:

Slower than modern methods (e.g. highs).

May not handle large problems.

Recommendation: Use for small problems or when a classical approach is required.

SciPy. Solving the MILP problem

3. 'interior-point’

Description: The interior point method. This method solves a linear programming problem by approaching the optimal solution from within a feasible region.

Advantages:

Fast convergence for large problems.

Works well for problems with many variables and constraints.

Disadvantages:

May be less accurate for problems with singular solutions.

Requires more memory than the simplex method.

Recommendation: Use for large problems where speed is important.

SciPy. Solving the MILP problem

4. 'revised simplex’

Description:

An improved version of the simplex method that uses more efficient computational approaches.

Advantages:

More efficient than the classic simplex method.

Suitable for medium-sized problems.

Disadvantages:

Slower than the interior point method for large problems.

Recommendations: Use for medium-sized problems where accuracy is important.

Conditional optimization method="trust-constr"

The **trust-constr method** in the minimize function of the **scipy.optimize** library is a constrained optimization method based on trust-regions. It is designed to solve optimization problems with constraints (both linear and nonlinear) and can work with both continuous and discrete variables.

Conditional optimization method="trust-constr"

Main features of the trust-constr method:

Support for constraints.

The method can work with constraints of different types

Suitable for problems with both smooth and non-smooth functions.

Can work with a large number of variables and constraints.

The method provides high accuracy of the solution, especially for problems with non-linear constraints.

Conditional optimization method="trust-constr"

$$f(x) = 3x_1 + 3x_2 \to \min$$

$$x_1 + x_2 \leq 8$$

$$2x_1 - x_2 \geq 1$$

$$x_1 - 2x_2 \leq 2$$

$$x_1 \geq 0, x_2 \geq 0$$

Conditional optimization method="trust-constr"

$$f(x) = 3x_1 + 3x_2 \to \min$$
$$x_1 + x_2 \le 8$$
$$2x_1 - x_2 \ge 1$$
$$x_1 - 2x_2 \le 2$$
$$x_1 \ge 0, x_2 \ge 0$$

```
from scipy.optimize import Bounds
bounds = Bounds ([0, 1e9], [0, 1e9])
import numpy as np
from scipy.optimize import minimize
from scipy.optimize import LinearConstraint
linear_constraint = LinearConstraint ([[1,1], [-2,1],[1,-2]], [8,-1,2])

[38] def objective(x):
        x1, x2 = x
        return 3 * x1 + 3 * x2

    x0 = np.array([0.5, 0])

    res = minimize(objective, x0, method='trust-constr',
                   constraints=[linear_constraint],
                   options={'verbose': 1}, bounds=bounds)

    print(res.x)
```

```
Number of iterations: 360, function evaluations: 1077, CG iterations: 350, optimality: 2.65e-14, constraint violation: 1.00e+09, execution time: 0.49 s.
[1.49999979 2.9411815 ]
```

CVXPY. Overview

CVXPY - this package was implemented to solve convex optimization problems. To solve the problem, it is necessary to perform several steps: define variables, set the objective function and constraints to form the object of the optimization problem. After the problem is formulated, before running the solver, the convexity of the objective function and constraints is checked using the **DCP** (Disciplined Convex Programming) rules.

$$f(x) = x_1^2 + x_2^2 - 2x_1 \to min$$
$$\begin{cases} x_1^2 + 4x_2^2 - 4x_1 - 4x_2 \leq 0 \\ x_1 + x_2 \geq 4 \\ x \geq 0 \end{cases}$$

```python
import cvxpy as cp

x = cp.Variable()
y = cp.Variable()

objective = cp.Minimize(x**2 + y**2-2*x)

constraints = [
    x**2+4*y**2-4*x-4*y <= 0,
    x+y >= 4,
    x >= 0
]

assert objective.is_dcp(), "Objective function is not convex"
for constraint in constraints:
    assert constraint.is_dcp(), "Constraint is not convex"

problem = cp.Problem(objective, constraints)
problem.solve()

print("Minimal value:", problem.value)
print("Optimal x and y:", x.value, y.value)
```

```
Minimal value: 3.500000000367045
Optimal x and y: 2.5000000056865748 1.4999999944357738
```

Transport problem

The transportation problem is a linear programming problem and its purpose is to determine the most economical plan for transporting goods from points of departure (warehouses) to points of destination (stores).

Let there be m warehouses $A_1, A_2, \dots, A_m$ from which goods should be transported to n stores $B_1, B_2, \dots, B_n$. Let $c_{ij}, i = 1, \dots, m, j = 1, \dots, n$ be the cost of transporting one unit of cargo from the *i*-th warehouse to the j-th store (transportation tariff). Let $a_i, i = 1, \dots, m$ be the cargo stock at the *i*-th warehouse; $b_j, j = 1, \dots, n -$ be the cargo demand at the *j*-th store; $x_{ij}, i = 1, \dots, m, j = 1, \dots, n -$ be the number of cargo units transported from the *i*-th departure point to the *j*-th destination point. The matrix $X = (x_{ij})_{m \times n}$ is called the transportation plan. The matrix $C = (c_{ij})_{m \times n}$ is called the tariff matrix.

Transport problem

As the optimality criterion, we will choose the minimum cost of transporting the entire cargo. Thus, **to solve the transport problem** is to determine the transportation plan $x_{ij}, i = 1, \dots, m, j = 1, \dots, n$, that is, the amount of cargo that should be transported from the i-th departure point to the j-th destination point so that all departure points get rid of cargo reserves, and all destination points receive the required amount of cargo, and at the same time the cost of transporting the entire cargo is the lowest.

Transport problem

The mathematical model of the transport problem will take the form:

$$F = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \rightarrow min$$

$$\sum_{i=1}^{m} x_{ij} = b_j, j = 1, \dots, n$$

$$\sum_{j=1}^{n} x_{ij} = a_i, i = 1, \dots, m$$

$$x_{ij} \geq 0, i = 1, \dots, m, j = 1, \dots, n$$

Transport problem. Example

Warehouses	Stores				Stocks
	B_1	B_2	B_3	B_4	
A_1	2 x_{11}	3 x_{12}	4 x_{13}	2 x_{14}	140
A_2	8 x_{21}	4 x_{22}	1 x_{23}	4 x_{24}	160
A_3	9 x_{31}	7 x_{32}	3 x_{33}	6 x_{34}	120
Needs	150	90	100	80	

Warehouses	Stores				Stocks
	B_1	B_2	B_3	B_4	
A_1	2 x_{11}	3 x_{12}	4 x_{13}	2 x_{14}	140
A_2	8 x_{21}	4 x_{22}	1 x_{23}	4 x_{24}	160
A_3	9 x_{31}	7 x_{32}	3 x_{33}	6 x_{34}	120
Needs	150	90	100	80	

```
Optimal solution: [140.   0.   0.   0.   0.  90.  70.   0.  10.   0.  30.  80.]
Optimal value of the objective function: 1370.0
```

```python
from scipy.optimize import linprog

# objective function coefficients
c = [2, 3, 4, 2, 8, 4, 1, 4, 9, 7, 3, 6]


stocks = [140, 160, 120]
A_ub = [
    [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
]


needs = [150, 90, 100, 80]
A_eq = [
    [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0],
    [0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0],
    [0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]
]

result = linprog(c, A_ub=A_ub, b_ub=stocks, A_eq=A_eq, b_eq=needs)

print("Optimal solution:", result.x)
print("Optimal value of the objective function:", result.fun)
```