Chapter 2: Mathematical Foundations for Optimization

In Chapter 1, we described optimization as the search for the best choice under limits. In this chapter, we build the mathematical language needed to say that idea precisely.

This language matters because quantum optimization algorithms do not solve vague problems. They solve mathematical models. Before a problem can be given to a variational quantum algorithm, we must decide:

- What are the variables?
- What values can they take?
- What number are we trying to minimize or maximize?
- What restrictions must be obeyed?
- Is the search space continuous, discrete, or binary?
- Can the problem be written in a form such as QUBO or Ising?

The purpose of this chapter is not to turn you into a professional mathematician immediately. It is to give you a reliable working foundation. Later chapters will use this foundation when we discuss VQE, QAOA, QUBO models, Ising Hamiltonians, portfolio optimization, routing, scheduling, and benchmarking.

We begin with the most basic building blocks: numbers arranged into vectors and matrices.

2.1 Decision Variables: The Choices We Control

An optimization problem begins with decision variables.

A decision variable is a quantity whose value we are allowed to choose. It represents a controllable part of the problem.

For example, suppose a student wants to decide how many hours to spend studying mathematics and physics this weekend. Let

$$x_1 = \text{hours spent on mathematics}$$

and

$$x_2 = \text{hours spent on physics.}$$

Here, x_1 and x_2 are decision variables.

If the student has 10 available hours, one possible choice is

$$x_1 = 6, \qquad x_2 = 4.$$

Another possible choice is

$$x_1 = 3, \qquad x_2 = 7.$$

The optimization problem asks: which choice is best according to some goal?

In real applications, decision variables can represent many things:

- whether an asset is included in a portfolio,
- how much power a generator should produce,
- which route a delivery truck should take,
- which machine should process a factory job,
- which parameters a machine-learning model should use.

The values allowed for a variable depend on the problem. A variable may be continuous, integer, or binary.

A continuous variable can take values on a continuum, such as

$$x = 2.3, \quad x = 2.31, \quad x = 2.314159.$$

For example, the amount of electricity produced by a generator may be modeled as a continuous variable.

An integer variable must be a whole number:

$$x \in \{0, 1, 2, 3, \dots\}.$$

For example, the number of trucks assigned to a warehouse must be an integer.

A binary variable can take only two values:

$$x \in \{0, 1\}.$$

Binary variables are especially important in quantum optimization. They naturally represent yes/no decisions:

$$x_i = \begin{cases} 1, & \text{if choice } i \text{ is selected,} \\ 0, & \text{if choice } i \text{ is not selected.} \end{cases}$$

For example, in portfolio optimization, $x_i = 1$ may mean "buy asset i," while $x_i = 0$ means "do not buy asset i."

Binary variables will become central in Chapter 11, where we study quadratic unconstrained binary optimization, or QUBO.

2.2 Vectors: Many Variables Written as One Object

When a problem has many decision variables, writing them one by one becomes inconvenient. A vector is a way to collect several numbers into one mathematical object.

If we have three decision variables x_1, x_2, x_3, we can write them as the vector

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

The symbol x now represents the whole list of variables.

For example, suppose a company must decide how many units of three products to make:

$$x_1 = \text{number of chairs},$$

$$x_2 = \text{number of tables},$$

$$x_3 = \text{number of desks.}$$

Then

$$x = \begin{bmatrix} 20 \\ 10 \\ 5 \end{bmatrix}$$

means the company makes 20 chairs, 10 tables, and 5 desks.

A vector has a dimension, which is the number of entries it contains. The vector above has dimension 3.

We often write

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

to mean that x is a vector with n real-number entries. The symbol mathbbR means the set of real numbers, and mathbbR^n means n-dimensional real space.

For example,

$$x \in \mathbb{R}^4$$

means

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

where each x_i is a real number.

Vectors are used throughout optimization because most realistic problems involve many variables, not just one. Numerical optimization is commonly formulated using vectors, objective functions, and constraints, as in standard treatments such as Nocedal and Wright's Numerical Optimization (Nocedal & Wright, 2006).

2.3 Matrices: Tables That Transform Vectors

A matrix is a rectangular table of numbers. For example,

$$A = \begin{bmatrix} 2 & 1 \\ 3 & 4 \\ 0 & 5 \end{bmatrix}$$

is a matrix with 3 rows and 2 columns.

Matrices are useful because they can describe relationships between many variables at once.

Suppose a factory makes two products: chairs and tables. Each chair requires 2 units of wood and 3 units of labor. Each table requires 1 unit of wood and 4 units of labor.

Let

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

where x_1 is the number of chairs and x_2 is the number of tables. The resource use can be written as

$$Ax = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

The first row calculates wood use:

$$2x_1 + x_2.$$

The second row calculates labor use:

$$3x_1 + 4x_2.$$

So the matrix equation

$$Ax \leq b$$

can represent resource limits. If the factory has at most 100 units of wood and 180 units of labor, then

$$\begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 100 \\ 180 \end{bmatrix}.$$

This compact expression actually means two inequalities:

$$2x_1 + x_2 \leq 100,$$

$$3x_1 + 4x_2 \leq 180.$$

Matrices let us write large optimization problems in a clean way. Instead of writing thousands of separate equations, we can write them using matrix notation.

2.4 Objective Functions: Turning a Choice into a Score

An objective function is a rule that assigns a number to each possible decision.

If the goal is to minimize cost, the objective function returns a cost. If the goal is to maximize profit, the objective function returns profit. If the goal is to minimize error, it returns an error score.

Mathematically, an objective function is often written as

$$f(x).$$

This means: given the decision vector x, the function f outputs a number.

For example, suppose a factory earns \$30 profit for each chair and \$50 profit for each table. If

$$x_1 = \text{number of chairs},$$

$$x_2 = \text{number of tables},$$

then the profit is

$$f(x) = 30x_1 + 50x_2.$$

If the factory makes 20 chairs and 10 tables, then

$$f(x) = 30(20) + 50(10) = 600 + 500 = 1100.$$

So this decision gives profit 1100 dollars.

If the goal is to maximize profit, we write

$$\max_x f(x).$$

This means: choose x to make f(x) as large as possible.

If the goal is to minimize cost, we write

$$\min_x f(x).$$

For example, if f(x) is total delivery distance, then

$$\min_x f(x)$$

means “choose the route x with the smallest total distance.”

Many optimization books use minimization as the standard form, because maximizing f(x) is equivalent to minimizing -f(x). For example, maximizing profit

$$f(x)$$

is the same as minimizing negative profit

$$-f(x).$$

This simple conversion is widely used in mathematical optimization (Boyd & Vandenberghe, 2004).

2.5 Constraints: Rules That Valid Solutions Must Obey

A constraint is a rule that a solution must satisfy.

Without constraints, the best mathematical answer may be meaningless in real life. If a factory earns profit from every product, then “make infinitely many products” would look best unless we include limits such as labor, material, and storage.

A constrained optimization problem may look like this:

$$\begin{aligned} \text{maximize} \quad & 30x_1 + 50x_2 \\ \text{subject to} \quad & 2x_1 + x_2 \leq 100, \\ & 3x_1 + 4x_2 \leq 180, \\ & x_1 \geq 0, \\ & x_2 \geq 0. \end{aligned}$$

The phrase subject to means “while obeying.”

Here:

- $30x_1 + 50x_2$ is the objective function.
- $2x_1 + x_2 \leq 100$ is the wood constraint.
- $3x_1 + 4x_2 \leq 180$ is the labor constraint.
- $x_1 \geq 0$, $x_2 \geq 0$ say that production quantities cannot be negative.

A solution that satisfies all constraints is called feasible.

A solution that violates at least one constraint is infeasible.

For example, if

$$x = \begin{bmatrix} 20 \\ 10 \end{bmatrix},$$

then wood use is

$$2(20) + 10 = 50,$$

and labor use is

$$3(20) + 4(10) = 100.$$

Both are within the limits, so this solution is feasible.

But if

$$x = \begin{bmatrix} 60 \\ 30 \end{bmatrix},$$

then wood use is

$$2(60) + 30 = 150,$$

which exceeds the wood limit of 100. So this solution is infeasible.

The set of all feasible solutions is called the feasible region or feasible set.

In quantum optimization, constraints are especially important because many quantum algorithms naturally produce candidate bitstrings, and not every bitstring necessarily represents a valid solution. Later, we will see two common strategies:

1. Build the constraints directly into the quantum circuit.
2. Add penalty terms to the objective function so that infeasible solutions receive worse scores.

2.6 Equality and Inequality Constraints

Constraints usually come in two main types.

An equality constraint requires something to be exactly equal:

$$h(x) = 0.$$

For example, suppose a portfolio must choose exactly 3 assets from 10. If $x_i \in \{0,1\}$ indicates whether asset i is selected, then

$$x_1 + x_2 + \cdots + x_{10} = 3$$

is an equality constraint.

An inequality constraint requires something to be less than, greater than, or equal to a limit:

$$g(x) \leq 0.$$

For example, if a truck can carry at most 1000 kg, and package i has weight w_i, then

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \leq 1000$$

is an inequality constraint.

Equality constraints are often stricter than inequality constraints. If a truck can carry at most 1000 kg, it is acceptable to carry 800 kg. But if a scheduling rule says exactly 5 workers must be assigned, then assigning 4 or 6 workers violates the rule.

When modeling real problems, constraints must be chosen carefully. A model that forgets an important constraint can give a mathematically optimal answer that is practically useless.

2.7 Linear and Nonlinear Functions

A function is linear if it is built from variables multiplied by constants and added together, with no powers, products between variables, or curved operations.

For example,

$$f(x) = 3x_1 + 5x_2 - 7x_3$$

is linear.

The constraint

$$2x_1 + x_2 \leq 100$$

is also linear.

A function is nonlinear if it includes curved or interacting terms such as squares, products between variables, exponentials, or logarithms. For example,

$$f(x) = x_1^2 + x_2^2$$

is nonlinear because of the squared terms.

The function

$$f(x) = x_1 x_2$$

is nonlinear because it multiplies two variables together.

The function

f(x) = e□x□□

is nonlinear because it uses an exponential.

Linear optimization problems are often easier to analyze and solve than general nonlinear problems. Nonlinear optimization can be much harder because the objective landscape may contain many hills, valleys, and saddle points. Standard numerical optimization texts study these distinctions carefully, including the role of gradients, curvature, and local solutions (Nocedal & Wright, 2006).

Quantum optimization often focuses on discrete nonlinear-looking objectives, especially quadratic objectives over binary variables. For example,

$$f(x) = 3x_1^2 + 2x_2 - 5x_1x_2$$

is a quadratic function because it includes a product of two variables. If $x_1, x_2 \in \{0,1\}$, this is a small QUBO-style objective.

2.8 Gradients: The Direction of Steepest Increase

For continuous variables, we often want to know how a small change in x changes the objective value f(x). This leads to the idea of a derivative.

For a one-variable function f(x), the derivative f'(x) measures the slope of the function at x. If $f'(x) > 0$, the function is increasing at that point. If $f'(x) < 0$, it is decreasing. If $f'(x) = 0$, the function is flat at that point, at least instantaneously.

For example,

$$f(x) = x^2.$$

The derivative is

$$f'(x) = 2x.$$

At x = 3,

$$f'(3) = 6,$$

so the function is increasing. At x = -3,

$$f'(-3) = -6,$$

so the function is decreasing. At x = 0,

$$f'(0) = 0.$$

The point x=0 is the minimum of x^2.

For a function with many variables, the generalization of the derivative is called the gradient.

If

$$f(x_1, x_2, \ldots, x_n)$$

is a function of n variables, then its gradient is the vector

$$\nabla f(x) = \begin{bmatrix} (\partial f) / (\partial x_1) \\ (\partial f) / (\partial x_2) \\ \vdots \\ (\partial f) / (\partial x_n) \end{bmatrix}.$$

The symbol $(\partial f)/(\partial x_i)$ is called a partial derivative. It means: how does f change when x_i changes, while the other variables are held fixed?

For example, let

$$f(x_1, x_2) = x_1^2 + 3x_1 x_2.$$

Then

$$(\partial f) / (\partial x_1) = 2x_1,$$

and

$$(\partial f) / (\partial x_2) = 6x_1.$$

So

$$\nabla f(x) = \begin{bmatrix} 2x_1 \\ 6x_1 \end{bmatrix}.$$

At the point $x_1=1$, $x_2=2$,

$$\nabla f(1,2) = \begin{bmatrix} 2 \\ 12 \end{bmatrix}.$$

For differentiable functions in ordinary Euclidean space, the gradient points in the direction of steepest local increase, and the negative gradient points in the direction of steepest local decrease; this is a standard fact used in gradient-based optimization methods (Nocedal & Wright, 2006).

This is why gradient descent updates parameters in the direction

$$-\nabla f(x).$$

A simple gradient descent step is

$$x_{new} = x_{old} - \alpha \nabla f(x_{old}),$$

where $\alpha > 0$ is the learning rate or step size. The step size controls how far we move.

For example, if

$$f(x) = x^2,$$

then

$$\nabla f(x) = 2x.$$

Starting at $x=5$, using $\alpha=0.1$,

$$x_{new} = 5 - 0.1(10) = 4.$$

The next step gives

$$x_{new} = 4 - 0.1(8) = 3.2.$$

The algorithm moves toward $x=0$, the minimum.

Gradients will return later when we study variational quantum algorithms. In VQAs, the parameters of a quantum circuit are often continuous, such as rotation angles. The algorithm tries to adjust those parameters to reduce a cost function. Sometimes it estimates gradients directly; sometimes it uses gradient-free methods.

2.9 Local and Global Optima

An optimum is a best solution according to the objective function.

If we are minimizing, an optimum is a point with a small objective value. If we are maximizing, an optimum is a point with a large objective value.

But there are two important kinds of optimum: local and global.

A global minimum is a feasible point whose objective value is less than or equal to the objective value of every other feasible point.

Mathematically, x^star is a global minimum if

$$f(x^\star) \leq f(x)$$

for every feasible x.

For example,

f(x) = x□

has a global minimum at

x^star = 0,

because

x□ ≥ 0

for all real x.

A local minimum is a point that is best compared with nearby feasible points, but not necessarily best compared with all feasible points.

Imagine hiking in a mountain landscape. A local minimum is the bottom of a nearby valley. A global minimum is the lowest valley in the entire landscape.

For example, a function might have several valleys. If you start gradient descent in one valley, it may settle at the bottom of that valley even if another, lower valley exists far away.

This distinction is central in nonconvex optimization. Many practical optimization problems are nonconvex, meaning they can contain multiple local optima or complicated landscapes. Numerical optimization methods must therefore be judged not only by whether they improve a solution, but also by whether they reliably find high-quality solutions (Nocedal & Wright, 2006).

For quantum optimization, the same idea appears in a different form. A variational quantum algorithm has parameters, and those parameters create an optimization landscape. The algorithm may find a good parameter setting, but there is no automatic guarantee that it has found the best possible setting.

2.10 Convexity: The Special Case Where Local Is Global

Some optimization problems have a beautiful structure called convexity.

Convexity is important because convex optimization problems are often much more reliable to solve than general nonconvex problems. A key property is that, under standard definitions, any local minimum of a convex optimization problem is also a global minimum (Boyd & Vandenberghe, 2004).

To understand convexity, first imagine a set of points.

A set is convex if, whenever it contains two points, it also contains the entire straight line segment between them.

For example, a filled circle is convex. If you choose any two points inside the circle and draw a straight line between them, the whole line segment stays inside the circle.

A crescent moon shape is not convex. You can choose two points inside the crescent such that the straight line between them passes outside the crescent.

Now consider a function.

A function f is convex if the line segment between any two points on its graph lies above or on the graph.

A common algebraic definition is:

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$$

for all x,y in the domain and all θ with

$$0 \leq \theta \leq 1.$$

This formula may look abstract, so let us interpret it.

The point

$$\theta x + (1 - \theta)y$$

is a weighted average of x and y. If θ = 0.5, it is the midpoint.

The inequality says:

> The function value at the average point is no larger than the average of the function values.

For example,

$$f(x) = x^2$$

is convex. Its graph is a bowl shape.

By contrast,

$$f(x) = \sin(x)$$

is not convex over the entire real line because it curves up and down repeatedly.

A convex optimization problem usually means minimizing a convex objective function over a convex feasible set. Convex optimization is a major field because many such problems can be solved efficiently and reliably in practice, especially compared with arbitrary nonconvex problems (Boyd & Vandenberghe, 2004).

However, many problems in quantum optimization are not convex. QUBO problems, routing problems, scheduling problems, and many graph problems are discrete combinatorial problems. Their search spaces are not smooth bowls; they are more like enormous collections of separated points.

This is one reason quantum optimization is interesting. It often targets hard optimization problems where classical methods may struggle as the number of possibilities grows.

2.11 Continuous Optimization

A problem is a continuous optimization problem if its variables can vary continuously.

For example,

$$\min_{x \in \mathbb{R}} (x-3)^2$$

is a continuous optimization problem. The variable x can be 2.9, 2.99, 2.999, and so on. The minimum occurs at

$$x=3.$$

Another example is fitting a line to data. Suppose we want a line

$$y = ax + b$$

that predicts data well. The parameters a and b are continuous variables. We may choose them to minimize the sum of squared prediction errors:

$$f(a,b) = \sum_{i=1}^{n} (y_i - (ax_i+b))^2.$$

This is a continuous optimization problem.

Continuous optimization often uses tools such as derivatives, gradients, Hessians, and step sizes. A Hessian is a matrix of second derivatives. It describes curvature: whether the function bends like a bowl, a hill, or a saddle. Hessians are important in Newton-type methods and second-order optimization methods (Nocedal & Wright, 2006).

In variational quantum algorithms, continuous optimization appears because quantum circuits often contain adjustable angles. For example, a circuit may contain a rotation gate with parameter θ. The algorithm tries values such as

$$\theta = 0.1, 0.2, 0.35, 1.7$$

and searches for parameter values that minimize a measured cost.

So even when the original application is discrete, the training of a variational quantum circuit is often a continuous optimization problem.

2.12 Discrete Optimization

A problem is a discrete optimization problem if its variables can take values from separate, distinct choices rather than a continuum.

For example, suppose

$$x \in \{0,1,2,3,4,5\}.$$

Then x is discrete. It cannot be 2.7.

Binary optimization is a special kind of discrete optimization where each variable is either 0 or 1.

For example,

$$x_i \in \{0,1\}$$

for

$$i = 1,2,\ldots,n.$$

This creates a search space of possible bitstrings:

$$x = x_1 x_2 \ldots x_n.$$

If n=3, there are

$$2^3 = 8$$

possible bitstrings:

$$000,\ 001,\ 010,\ 011,\ 100,\ 101,\ 110,\ 111.$$

If n=10, there are

$$2^{10} = 1024$$

possible bitstrings.

If n=100, there are approximately

$$2^{100} \approx 1.27 \times 10^{30}$$

possible bitstrings.

This explosive growth is one reason discrete optimization can become difficult. Combinatorial optimization studies problems where the feasible solutions are combinations, selections, orderings, or subsets of discrete objects; classic examples include routing, matching, scheduling, and graph problems (Papadimitriou & Steiglitz, 1982).

Quantum optimization methods such as QAOA are often aimed at these discrete search spaces. A quantum computer can naturally represent n binary variables using n qubits. Measurement then produces bitstrings. But this does not mean the quantum computer automatically checks all bitstrings at once and gives the best one. That is a common misunderstanding. Quantum algorithms must use interference, structure, and repeated sampling to increase the probability of useful answers.

2.13 Combinatorial Search Spaces

A combinatorial search space is a set of possible solutions formed by combining discrete choices.

The word “combinatorial” comes from “combinations.” These problems often ask questions such as:

- Which subset should we choose?
- Which order should we use?
- Which assignment is best?
- Which edges should be included in a graph solution?

Let us look at examples.

Example 1: Selecting Items

Suppose there are 5 items, and each item can either be selected or not selected. A solution is a binary vector:

x = beginbmatrix x□ ; x□ ; x□ ; x□ ; x□ endbmatrix, x□ ∈ \0,1\.

The bitstring

$$10110$$

means:

- select item 1,
- do not select item 2,
- select item 3,
- select item 4,
- do not select item 5.

This type of model appears in portfolio selection, feature selection in machine learning, and knapsack-style resource allocation.

Example 2: Ordering Cities

In a routing problem, a solution may be an ordering of cities.

For 4 cities, one route might be

$$A \to C \to B \to D \to A.$$

Another route might be

$$A \to B \to D \to C \to A.$$

If there are many cities, the number of possible routes grows very quickly. Routing problems are central examples in combinatorial optimization and have been studied extensively in classical optimization and complexity theory (Papadimitriou & Steiglitz, 1982).

Example 3: Assigning Jobs to Machines

Suppose 3 jobs must be assigned to 2 machines. A solution could be:

$$\text{Job } 1 \to \text{Machine A},$$

$$\text{Job } 2 \to \text{Machine B},$$

$$\text{Job } 3 \rightarrow \text{Machine A.}$$

Scheduling and assignment problems become difficult when we add realistic constraints: deadlines, machine capacities, worker shifts, setup times, maintenance windows, and priority rules.

These examples show why modeling is not just a formality. The way we encode the search space affects the size and difficulty of the optimization problem.

2.14 Binary Variables and Bitstrings

Quantum optimization often uses binary variables because qubit measurements naturally produce binary outcomes.

A bit is a classical value that is either 0 or 1.

A bitstring is a sequence of bits, such as

$$01101.$$

If we use n binary variables, a candidate solution can be written as an n-bit bitstring.

For example, suppose we have four possible projects and must choose which ones to fund:

$$x\square = \text{begincases 1, if project i is funded, ; 0, otherwise. endcases}$$

The bitstring

$$1010$$

means fund projects 1 and 3, but not 2 and 4.

This binary representation is simple but powerful. Many real optimization problems can be transformed into binary form by introducing enough binary variables. For example, integer quantities can be represented using binary expansion. If a variable y can be an integer from 0 to 7, we can write

$$y = b_0 + 2b_1 + 4b_2,$$

where

$$b_0,b_1,b_2 \in \{0,1\}.$$

If

$$b_0=1,\ b_1=0,\ b_2=1,$$

then

$$y = 1 + 2(0) + 4(1) = 5.$$

This is one reason binary optimization is a natural bridge between practical optimization and qubit-based algorithms.

However, binary encoding can increase the number of variables. A good model balances expressiveness with problem size.

2.15 Quadratic Objectives: The Shape Behind QUBO

A quadratic function is a function that includes terms involving products of two variables.

For continuous variables, an example is

$$f(x_1,x_2) = 2x_1^2 + 3x_1x_2 + x_2^2.$$

For binary variables, a quadratic objective might be

$$f(x_1,x_2,x_3) = 4x_1 - 2x_2 + 7x_3 + 5x_1x_2 - 3x_2x_3.$$

The terms

$$4x_1,\ -2x_2,\ 7x_3$$

are linear terms.

The terms

$$5x_1x_2, -3x_2x_3$$

are quadratic interaction terms.

The product x_ix_j matters only when both variables are 1. For binary variables,

$$x_ix_j = \begin{cases} 1, & \text{if } x_i=1 \text{ and } x_j=1, \\ 0, & \text{otherwise.} \end{cases}$$

So the term

$$5x_1x_2$$

adds a penalty of 5 if both x_1 and x_2 are selected.

The term

$$-3x_2x_3$$

gives a reward of 3 if both x_2 and x_3 are selected, because it lowers the objective in a minimization problem.

This kind of expression is the foundation of QUBO:

$$\min_{x \in \{0,1\}^n} x^T Q\, x.$$

Here:

- x is a binary vector,
- Q is a matrix of coefficients,
- $x^T Q\, x$ is a compact way to write a quadratic objective.

We will study this carefully in Chapter 11. For now, the key idea is:

> QUBO turns optimization into the task of finding the best binary string according to a quadratic score.

This form is important because QUBO problems can be mapped to Ising Hamiltonians, which are commonly used in quantum optimization models.

2.16 Penalty Terms: Turning Constraints into Costs

Many quantum optimization methods prefer unconstrained problems, meaning problems without explicit constraints. But real problems usually have constraints.

A common modeling trick is to convert constraints into penalty terms.

A penalty term adds extra cost when a constraint is violated.

Suppose we want to minimize

$$f(x)$$

subject to the constraint

$$x_1 + x_2 = 1,$$

where

$$x_1, x_2 \in \{0,1\}.$$

This constraint means exactly one of x_1 and x_2 should be 1.

The valid possibilities are:

$$x_1 = 1,\ x_2 = 0,$$

and

$$x_1=0,\ x_2=1.$$

The invalid possibilities are:

$$x_1=0,\ x_2=0,$$

and

$$x_1=1,\ x_2=1.$$

We can create a penalty:

$$P(x) = \lambda(x_1+x_2-1)^2,$$

where $\lambda > 0$ is a penalty weight.

If the constraint is satisfied, then

$$x_1+x_2-1 = 0,$$

so

$$P(x)=0.$$

If the constraint is violated, then the squared term is positive, so

$$P(x)>0.$$

The constrained problem can be replaced by the unconstrained problem

$$\min \left[f(x) + \lambda(x_1+x_2-1)^2\right].$$

If λ is large enough, violating the constraint becomes unattractive.

But penalty weights require care. If λ is too small, the optimizer may prefer an infeasible solution because the penalty is not strong enough. If λ is too large, the penalty may dominate the original objective and make the optimization landscape harder to search.

Penalty methods are a standard idea in constrained optimization, but choosing penalty parameters well can be problem-dependent (Nocedal & Wright, 2006). In QUBO modeling, penalty design is one of the most important practical skills.

2.17 Continuous Versus Discrete: Why the Difference Matters

Continuous and discrete optimization problems feel different because their search spaces have different shapes.

In continuous optimization, you can move gradually.

If

$$x = 2.0$$

and you want to try a nearby value, you can choose

$$x = 2.01.$$

This makes gradients useful. A gradient tells you how the objective changes under small movements.

In discrete optimization, you often cannot move gradually.

If

$$x \in \0,1$$

there is no value halfway between 0 and 1 that is allowed. You must flip the bit or not flip it.

For a bitstring such as

$$01011,$$

a nearby solution may be one bit flip away:

$$01001.$$

The distance between bitstrings is often measured by Hamming distance. The Hamming distance between two bitstrings is the number of positions where they differ.

For example,

$$01011$$

and

```
11001
```

differ in two positions: the first bit and the fourth bit. So their Hamming distance is 2.

In continuous optimization, geometry often involves slopes and curves. In discrete optimization, geometry often involves neighbors, flips, swaps, and graph structure.

Variational quantum algorithms often combine both worlds:

- The quantum circuit parameters are continuous.
- The measured solutions are often discrete bitstrings.
- The classical optimizer adjusts continuous parameters.
- The final answer may be a discrete candidate solution.

This hybrid nature is one reason VQAs require careful mathematical thinking.

2.18 Feasible Solutions, Optimal Solutions, and Good Solutions

In theory, we often ask for an optimal solution.

In practice, especially for hard combinatorial problems, we may accept a good solution found within reasonable time.

This distinction is essential.

A solution can be:

1. Feasible but poor — it obeys the constraints but has a bad objective value.
2. Infeasible but attractive — it has a good objective value but violates a rule.
3. Feasible and good — it obeys the rules and has a strong objective value.
4. Globally optimal — it is the best feasible solution among all possibilities.

For example, suppose a delivery route is short but misses one customer. It may have a low distance score, but it is infeasible. A longer route that visits all customers may be the better valid solution.

When evaluating optimization algorithms, we should ask:

- Did the algorithm find feasible solutions?
- How good were those solutions?
- How long did it take?
- How many samples or function evaluations were needed?
- How does it compare with strong classical methods?

These questions will become central in Chapter 18, where we discuss honest benchmarking.

2.

Document information

Chapter 2: Mathematical Foundations for Optimization

Project	Variational Quantum Algorithms for Optimization
Document	Document 1.6
Author	phone
Verifier	Not verified
Downloaded	July 04, 2026 21:39 KST
Status	Working
Document link	https://theorytrace.com/projects/variational-quantum-algorithms-for-optimization/documents/chapter-2-mathematical-foundations-for-optimization/