

SciPy Optimizers

What is Optimization?

Optimization is the process of finding the best possible solution for a problem. It means **minimizing or maximizing a function** to get the most efficient result.

Examples:

1. **Finding the shortest route** between two places.
2. **Maximizing profit** while minimizing costs in a business.
3. **Minimizing error** in machine learning models.

Imagine a Mountain or a Hill:

- If we are standing somewhere on the surface, we may want to find the **lowest point** (the valley), which could represent the **minimum** value of a function.
- Alternatively, we could be looking for the **highest point** (the peak), which would represent the **maximum** value of the function.

This process of finding the highest or lowest point is called **optimization**.

What are Optimizers?

Optimizers are tools (functions) that help us **find the best possible value** for a given problem.

In SciPy, optimizers are found in the `scipy.optimize` module.

Types of Optimization Problems

1. **Unconstrained Optimization** – No restrictions (find the best number).

2. **Constrained Optimization** – Restrictions are given (e.g., values must be positive).
3. **Root Finding** – Find where a function equals zero.
4. **Linear Programming** – Optimize a function with constraints using straight-line equations.
5. **Curve Fitting** – Find the best function that fits given data points.

Optimizers in Machine Learning or Data Science

When we train a model in machine learning, we need to optimize certain parameters (like the weights in a neural network) to **minimize the error** of the model. This means we try to make the model's predictions as accurate as possible.

1 Unconstrained Optimization (No Restrictions)

Finds the **minimum** or **maximum** of a function.

`scipy.optimize.minimize` (Find Minimum Value)

- It tries different values of `x` and finds the one where `f(x)` is smallest.

It takes 3 arguments:

1. Function → `fun`
2. `x0`
3. Method
 - a. CG
 - b. BFGS

- c. NEWTON-CG
- d. L-BFGS-B
- e. TNC
- f. COBYLA
- g. SLQP

Example: Find the Minimum of a Function

 **Function:**


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

```
# Define the function
def f(x):
    return (x - 3)**2 + 5

# Initial guess for x
x0 = 0

# Minimize the function
result = minimize(f, x0, method='BFGS')

print(result.x) # Best value of x
```

 **Output:** 3.0 (which is the minimum of $f(x)$)

- **BFGS** and **L-BFGS-B**: These are quasi-Newton methods. **BFGS** is used for unconstrained problems, while **L-BFGS-B** is suitable for large-scale problems with bounds on variables.
- The function works without BFGS as well

2 Constrained Optimization (With Restrictions)

Finds the best solution while following some rules.

◆ `scipy.optimize.minimize` with Constraints

- We can add conditions like $x + y$ must be ≥ 1

Example: Find Minimum with Constraints

📌 **Function:**

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

📌 **Constraint:**

$$x + y \geq 1$$

```
# Define function to minimize
def f(x):
    return x[0]**2 + x[1]**2

# Constraint: x + y >= 1
constraints = {'type': 'ineq', 'fun': lambda x: x[0] + x[1] - 1}

# Initial guess for x and y
x0 = [0, 0]

# Solve
result = minimize(f, x0, constraints=constraints)
print(result.x) # Best values for x and y
```

Output: [0.5 0.5]

`'type': 'ineq'`

- `'ineq'` means "inequality constraint".
 - **Equality constraint:** where a condition must be exactly equal to a specific value
 - (e.g., $x_1 + x_2 = 10$).
 - **Inequality constraint:** where a condition must either be greater than or equal to, or less than or equal to, a certain value.
 - **Greater than or equal to:** $x_1 + x_2 \geq 1$
 - **Less than or equal to:** $x_1 + x_2 \leq 10$
- SciPy expects a constraint function `g(x)`, which must satisfy:

$$g(x) \geq 0$$

- This means the function must return a **non-negative value**.

```
constraints = {'type': 'ineq', 'fun': lambda x: x[0] + x[1] - 1}
```

- This part defines the **constraint** we want to impose on the optimization process
- It's a function that takes `x` (a list or array) as input and returns the value of the expression

$$x[0] + x[1] - 1$$

- The function returns the sum of two values x & y minus 1.
 - $x + y - 1$
 - The constraint is $x+y$ should be greater than 1
 - `'type': 'ineq'` makes it greater than 0

- But we want it greater than 1
- So, we subtract 1
 - Just $x + y$ would say $x + y$ should be greater than 0.
 - We don't have a function for greater than 1
 - So, we wrote $x + y - 1$
 - meaning → The value of $x + y - 1$ should be greater than 0.

3 Root Finding (Solving Equations)

- `root` is a function from `scipy.optimize` that allows you to **find the root of a function** (i.e., the value of `x` for which the function equals zero).
 - Finds the value of `x` where a function becomes **zero**.

`scipy.optimize.root`

- Finds `x` for which `f(x) = 0`

Equation:

$$x^2 - 4 = 0$$

Expected Answer: `x = ±2`

```
from scipy.optimize import root

# Define equation
def equation(x):
    return x**2 - 4

# Initial guess for x
x0 = 1

# Solve
```

```
solution = root(equation, x0)
print(solution.x)
```

Output: 2.

`x0=1` → In this case, we're starting at `x = 1`, but you can choose other starting values depending on the problem.

Why `solution.x`

- When we call the `root` function from SciPy, it returns an object that contains more information than just the solution to the equation. This object has various attributes that give us insights into how the root-finding process was performed.

The solution object returned by

`root` is a `OptimizeResult` object, and this object has the following important attributes:

- `x`: This contains the actual solution (root) of the equation. It's the value(s) of `x` that satisfy the equation `f(x) = 0`.
- `success`: A Boolean value that indicates whether the optimization was successful (`True`) or not (`False`).
- `status`: An integer code that gives more details about the termination status of the algorithm (whether it stopped because it converged, or if it encountered a problem, etc.).
- `message`: A string that provides a human-readable message about the result (e.g., why the optimization stopped).
- `fun`: The value of the function at the solution, i.e., `f(x)` evaluated at the root.

```
solution
```

Output:

```
message: The solution converged.  
success: True  
status: 1  
  fun: [ 1.776e-15]  
   x: [ 2.000e+00]  
method: hybr  
  nfev: 11  
  fjac: [[-1.000e+00]]  
   r: [-4.000e+00]  
  qtf: [ 1.530e-09]
```