How does the brentq library perform binary search optimization

`scipy.optimize.brentq` is an exceptional and widely-used "solver" algorithm.

The term "bisection optimization" is a close approximation, but `brentq` is, in fact, more intelligent and faster than the simple bisection method.

The primary role of `brentq` is not "optimization" (finding a maximum or minimum) but **"root-finding"**—that is, **finding the value `x` that satisfies the equation `f(x) = 0`**.

In the context of the `solver_i.py` script, `brentq` was used to find the `CP` value that made the `Error(CP)` equal to zero.

The power of `brentq` lies in its nature as a **hybrid algorithm**. It ingeniously combines the advantages of three techniques to achieve the optimal balance of **speed** and **reliability**.

---

1. The "Safety Net": The Bisection Method

The foundation and "safety net" of `brentq` is the bisection method.

- **Mechanism:**
    1. An interval `[a, b]` must be provided.
    2. `brentq` **requires** that `f(a)` and `f(b)` have **opposite signs** (one positive, one negative).
    3. This guarantees that the function's graph **must** cross the x-axis between `a` and `b` (i.e., a root `f(x) = 0` is guaranteed to exist within the interval).
    4. The algorithm checks the midpoint `m` of the interval.
    5. If `f(m)` is positive, the root must lie between `[m, b]` (as `f(b)` is negative).
    6. If `f(m)` is negative, the root must lie between `[a, m]` (as `f(a)` is positive).
- **Result:** With each step, the search interval is **halved exactly**.
- **Advantage:** Extremely reliable. It **guarantees** convergence as long as the initial `f(a)` and `f(b)` have opposite signs.
- **Disadvantage:** Relatively "naive" and slow. It does not learn from the function's shape and only ever bisects the interval.

2. The "Accelerator": The Secant Method

`brentq` does not always use the slow bisection method. It attempts to use a "smarter" guessing method known as the **Secant Method**.

- **Mechanism:**
    1. The secant method considers the two current points, `(a, f(a))` and `(b, f(b))`.
    2. It draws a **straight line (a secant line)** between them.
    3. It **guesses** that the root is likely where this straight line intersects the x-axis.
    4. This guess (referred to as `s`) is often **much closer** to the true root than the simple midpoint `m`.
- **Advantage:** Very fast convergence (superlinear convergence).

- **Disadvantage:** Not always reliable. It can sometimes produce a "bad guess" that falls outside the `[a, b]` interval.

3. The "Ultimate Accelerator": Inverse Quadratic Interpolation

When `brentq` has access to three points, it attempts an even more advanced technique:

- **Mechanism:** Instead of drawing a straight line (based on 2 points), it fits a **parabola (a quadratic curve)** through the 3 points.
- **Advantage:** Extremely fast convergence (approximately 1.8-order), even faster than the secant method.
- **Disadvantage:** Even less reliable and can sometimes fail entirely.

The `brentq` "Hybrid Strategy" (The Core)

The genius of `brentq` (developed by Richard Brent) lies in how it combines these three methods:

1. **Prefer the "Fast Lane":** The algorithm always **attempts** to use the fastest methods—Inverse Quadratic Interpolation or the Secant Method—to propose a "candidate root," `s`.
2. **Apply Strict Safety Checks:** Before accepting this candidate `s`, `brentq` performs rigorous checks:
   - Does this `s` fall within the **"safe" bisection interval** `[a, b]`?
   - Is the convergence "fast enough" compared to the previous step?
3. **Make an Intelligent Decision:**
   - **If `s` passes the checks:** `brentq` uses this "smart" guess `s` for the next step, thereby dramatically accelerating the search.
   - **If `s` fails the checks** (e.g., the secant method's guess was too wild): `brentq` **discards** the "smart" guess and **falls back** to performing one step of the **"slow but 100% reliable" bisection method**.

Conclusion: `brentq` operates on a foundation of guaranteed convergence (bisection) while opportunistically using the secant and interpolation methods as "turbo-chargers" to accelerate the process.

This is why it achieves the guaranteed convergence of the bisection method while, in most cases, converging as rapidly as the faster, less-reliable methods. In the context of the `solver` scripts, this allows it to find the precise `CP` value in very few steps (i.e., with few calls to the computationally expensive `calculate_fair_value` function).