README: Code Guide and Execution Flow

1. Overview

This project is designed to price a "Snowball" / Auto-callable structured product. The codebase is separated into two logical parts:

1. `calculate_fair_value.py`: A high-performance, parallelized pricing **engine** that calculates the Fair Value (FV) of the product.
2. `solver_*.py` / `validator.py`: A series of **solver** scripts that use the engine to find the unknown parameters (like `CP` or `KI`) required to meet a specific profit margin.

2. File Functionality

- **`calculate_fair_value.py` (The Engine)**
  - **Purpose:** To calculate the Fair Value (cost) of the product for a given set of parameters and a *guestimated* coupon rate (`CP_guess`).
  - **Key Features:**
    - **Parallelized:** Uses `multiprocessing` to run simulations on all available CPU cores.
    - **Optimized:** Uses **Antithetic Variates** (`Z` and `-Z`) to reduce variance (noise) and achieve faster, more stable convergence.
    - **Accurate:** Simulates `N=180` daily time steps to correctly monitor for "at any date" knock-in and auto-call events.
  - **Key Functions:** `calculate_fair_value()` (main) and `run_simulation_chunk()` (worker).
- **`solver_i.py` (Solver for Q1)**
  - **Purpose:** Solves for the unknown monthly coupon (`CP`) for the standard HKD product (Q1).
  - **Function:** Calls `calculate_fair_value()` inside a `scipy.optimize.brentq` solver to find the `CP` that makes `Fair_Value = 98.80%` (for Q1.i) and `Fair_Value = 98.40%` (for Q1.ii).
- **`solver_ii.py` (Solver for Q2)**
  - **Purpose:** Solves for the unknown `K0`, `KI`, and `AC` parameters required to maintain the 1.20% profit margin after the coupon is lowered.
  - **Function:** Calls `calculate_fair_value()` inside `brentq` to find the parameter that makes `Fair_Value = 98.80%` *given the new, lower CP*.
- **`solver_iii.py` (Solver for Q3)**
  - **Purpose:** Solves for the unknown monthly coupon (`CP`) for the **Quanto** (CNY) product.
  - **Function:** Identical to `solver_i.py`, but it passes `product_type='Quanto'` to the engine, which correctly switches to the Quanto pricing model (adjusted `r_g` and `r_d` for discounting).
- **`validator.py` (Final Check)**
  - **Purpose:** To verify that all answers from the solver scripts are correct.
  - **Function:** Plugs the final answers (e.g., `CP=3.45...%`) back into `calculate_fair_value()` and prints the resulting profit margin. The resulting margin should be extremely close to the target

(e.g., 1.20%).

3. How to Run & Debug

Follow this exact order. The output of Step 1 is required for Step 2.

1. **Run Q1 Solver:**
    - `python solver_i.py`
    - This will run for several minutes and output the final `CP1` value for the 1.20% margin.
    - **Record this `CP1` value** (e.g., `3.458654`).
2. **Run Q2 Solver:**
    - Open `solver_ii.py`.
    - **Manually paste** the `CP1_VALUE` from Step 1 into the script.
    - Run `python solver_ii.py`.
    - **Debug:** If the solver fails for one parameter (e.g., `KI`), it means the search interval `[a, b]` was too small. As seen in our logs, this is expected for insensitive parameters. Widen the search interval (e.g., `a=0.50`) and re-run.
3. **Run Q3 Solver:**
    - `python solver_iii.py`
    - This will run and find the two coupon rates for the Quanto product.
4. **Run Final Validation:**
    - Open `validator.py`.
    - **Manually paste** all 7 answers from the previous steps (`CP_Q1_I`, `CP_Q1_II`, `K0_Q2_A`, `KI_Q2_B`, `AC_Q2_C`, `CP_Q3_I`, `CP_Q3_II`) into the top of the `main` block.
    - Run `python validator.py`.
    - **Debug:** Check the output. All "Validation: Pass" messages indicate success. A "Warning: Error larger..." (especially for Q3) is acceptable and is due to the higher inherent noise of the Quanto model, not an error in the logic.