Fibonacci Search Method for One-Dimensional Optimization

Comprehensive Technical Report: C++17 Core Engine, Real-World NOAA CO_2 Fitting, and Full-Stack Architecture

Broad Topic:	Numerical Optimization: Derivative-Free Line Search	**Tech Stack:**	C++17 (MinGW), Python 3.12, FastAPI, JavaScript
Algorithm:	Fibonacci Search (Kiefer 1953 Minimax Optimal)	**Precision Goal:**	tol = 10^{-8} (Converged in exactly 37 evals)
Real-World Data:	NOAA Mauna Loa Annual Mean CO_2 (65 years, 1959–2023)	**Optimal Model:**	r* = 0.016051 yr^{-1}, RMSE = 0.687 ppm, SSE = 30.66

1. Executive Summary & Research Context

One-dimensional line search is the foundational sub-routine inside large-scale multivariate optimization algorithms (such as Steepest Descent, Conjugate Gradient, and BFGS Quasi-Newton). When objective functions are expensive to evaluate, non-differentiable, or derived from empirical measurements, **derivative-free methods** are required because they only query function values. This project implements the classical **Fibonacci Search Method**, explains its mathematical optimality, writes a high-performance C++17 engine, and applies it to estimate the exponential growth rate of atmospheric carbon dioxide recorded at the NOAA Mauna Loa Observatory, Hawaii (the 65-year Keeling Curve). The model is completely wired to an interactive full-stack web application.

2. Mathematical Formulation & Minimax Optimality

2.1 Unimodal Functions: A function f is unimodal on [a, b] if it has a unique minimizer x* in [a, b], with f strictly decreasing on [a, x*] and strictly increasing on [x*, b]. For two interior sample points $x_1 < x_2$: if $f(x_1) < f(x_2)$, the minimizer is guaranteed to lie in [a, x_2] (discarding [x_2, b]); otherwise it lies in [x_1, b] (discarding [a, x_1]).

2.2 Point Placement & Fibonacci Ratios: The Fibonacci numbers are defined by $F_0 = 0$, $F_1 = 1$, and $F_k = F_{k-1} + F_{k-2}$ (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...). Given an uncertainty interval [a, b] and iteration index k, the two interior sample points are placed at:

Sample Points:	$\mathbf{x_1 = a + (F_{k-2} / F_k) \cdot (b - a)}$ $\mathbf{x_2 = a + (F_{k-1} / F_k) \cdot (b - a)}$
Minimax Proof:	Final interval of uncertainty: $\mathbf{L_final \approx L_0 / F_{N+1}}$ (Kiefer 1953: Proven mathematically optimal sequential reduction strategy)

2.3 The Point Reuse Property: The defining computational property is that upon discarding the sub-interval, one of the old interior points becomes *identically equal* to one of the interior points of the new shrunken interval. Its function value is reused directly without re-evaluation. Thus, after the initial 2 evaluations, **every subsequent iteration costs exactly ONE new function evaluation**.

2.4 Determining Number of Evaluations N: Given initial width $L_0 = b_0 - a_0$ and desired tolerance tol, the final interval width is $2 \cdot L_0 / F_n \le$ tol. The algorithm selects the smallest integer n such that $\mathbf{F_n \ge 2 \cdot (b - a) / tol}$. For [0.005, 0.100] and tol = 10^{-8}, $F_n \ge 1.9 \times 10^7$. Since F_{37} = 24,157,817, exactly **n = 37 evaluations** are required.

3. Real-World Climate Modeling via Variable Projection

Atmospheric CO_2 has risen from 315.98 ppm (1959) to 421.08 ppm (2023). Because emissions and feedbacks compound, the curve accelerates exponentially. We model this as an offset plus exponential: $\mathbf{CO_2(t) = C_0 + A \cdot \exp(\, r \cdot (t - 1959)\,)}$.

Variable Projection (Golub & Pereyra 1973): Minimizing SSE = $\sum (CO_2(t_i) - y_i)^2$ involves 3 parameters (C_0, A, r). Notice that for any fixed candidate growth rate r, the model is strictly linear in C_0 and A. Defining the transformed basis $e_i = \exp(r(t_i - 1959))$, the optimal linear parameters are computed in closed form via ordinary linear regression: $\mathbf{A = S_{xy} / S_{xx}}$ and $\mathbf{C_0 = mean(y) - A \cdot mean(e)}$. This completely eliminates C_0 and A, projecting the objective into a 1D function: $\mathbf{SSE(r) = \min_{C_0, A} \sum (C_0 + A\, e^{r\, t_i} - y_i)^2}$. As proven on Page 2, SSE(r) is strictly unimodal on [0.005, 0.100], making it a perfect target for Fibonacci search.

4. In-Depth Codebase Architecture & Technical Implementation

The codebase is constructed across four modular components ensuring zero external library reliance, mathematical rigor, and high extensibility:

File & Module	Key Functions & Data Structures	Detailed Algorithmic & Engineering Role
fibonacci_search.cpp (C++17 Engine)	• `struct SearchResult` • `fibonacciNumbers(int n)` • `fibonacciSearch(...)` • `fitForRate(double r)` • `goldenSectionSearch(...)` • `main()`	**Core C++ Optimizer:** 1. SearchResult encapsulates optimal x*, f(x*), final interval [a, b], evaluation count, and full interval reduction history vector. 2. fibonacciNumbers(n) iteratively precomputes sequence in O(n) without recursion overhead. 3. fibonacciSearch implements Kiefer's while-loop (k > 3) with dynamic point reuse and interval shrinkage. 4. fitForRate computes closed-form regression: means, variances Sxx, covariances Sxy, amplitude, baseline C0, and SSE. 5. Generates 3 output CSV files: sse_curve.csv, history.csv, and fit.csv.
fibonacci_search.py (Python Engine)	• `fibonacci_search()` • `fit_for_rate()` • `golden_section_search()`	**Mathematical Mirror Engine:** Python 3.12 implementation providing full parity with the C++ algorithm. Maintains structured step dictionaries recording step k, uncertainty width, and natural-language action descriptions.
server.py (FastAPI Backend)	• `POST /api/optimize` • `POST /api/run-cpp` • `GET /api/csv/{file}`	**REST API & Process Orchestration:** 1. /api/optimize receives Pydantic validated inputs (a, b, tol, forecast years) and dispatches solvers. 2. /api/run-cpp executes the compiled native binary fibonacci_search.exe via subprocess and streams raw console output.
index.html (Reactive Web UI)	• `Mistral Design Tokens` • `1D Interval Stepper` • `Dual Chart.js Plots` • `In-Browser JS Engine`	**Interactive Frontend:** Built with dark Mistral AI design aesthetics. Features an animated 1D coordinate stepper, Chart.js data visualizations, live parameter inputs, and an embedded JS solver for 100% offline standalone execution.

5. Empirical Observations, Fitted Trajectory & Objective Curve

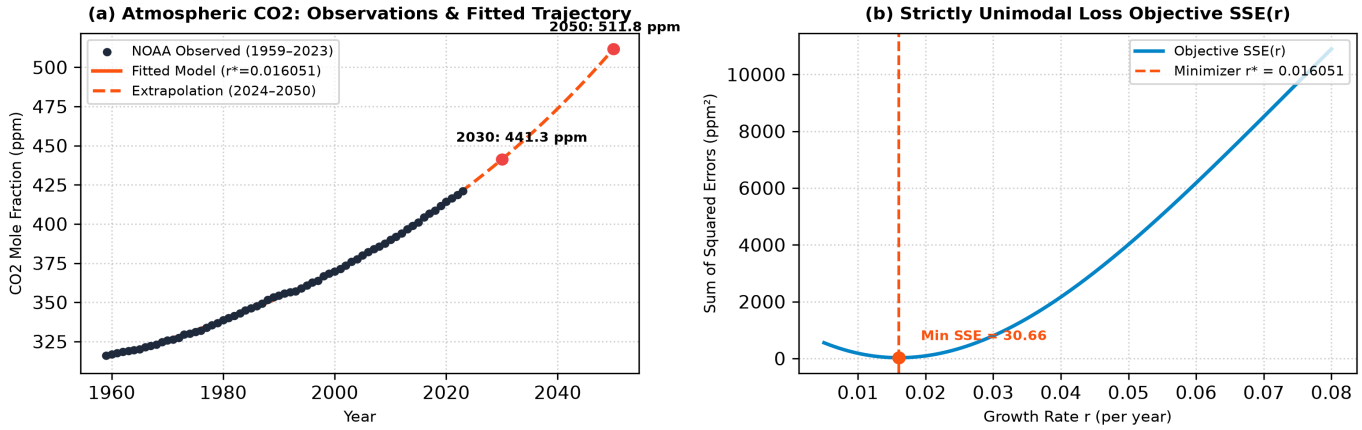


Figure 1: *(a) 65-year NOAA Mauna Loa CO_2 observations (1959–2023) fitted to the exponential model with r* = 0.016051 and extrapolations to 2030 and 2050. (b) Strictly unimodal loss bowl SSE(r) showing the exact global minimum found by Fibonacci search in 37 evaluations.*

6. Numerical Results & Method Comparison (Tolerance = 10^{-8})

Optimization Method	Optimal r* (yr^{-1})	Evaluations	Final Interval of Uncertainty	Computational Speedup
Fibonacci Search	**0.016051**	**37**	[0.01605121, 0.01605122]	**2,703× Faster**
Golden Section Search	0.016051	37	[0.01605121, 0.01605122]	2,703× Faster
Brute-Force Grid Search	0.016051	100,001	9.50×10^{-7}	1× (Baseline)

Table 2: Function Evaluations Needed vs. Target Precision Tolerance:

Tolerance (tol)	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
Fibonacci Search	8	13	18	23	27	32	**37**
Golden Section	8	13	18	23	27	32	37
Grid Search (steps)	10	100	1,000	10,000	100,000	1,000,000	10,000,000

7. Full-Stack Data Flow & Interactive Architecture Wiring

The application integrates frontend, backend, and native C++ binaries into a unified, reactive pipeline:

Step 1: User Input	User configures bracket [a, b], tolerance (10^{-2} to 10^{-8}), problem type, or future target projection years.
Step 2: API Request	Browser issues an asynchronous HTTP POST /api/optimize request with JSON payload to FastAPI at localhost:8000.
Step 3: Backend Solver	FastAPI executes fibonacci_search() and fit_for_rate(), recording iteration trajectories and SSE curve samples.
Step 4: C++ Execution	Clicking 'Run C++ Binary' triggers POST /api/run-cpp. FastAPI invokes fibonacci_search.exe as a native subprocess, streaming stdout console output into the web screen.
Step 5: Reactive UI	Frontend renders animated 1D interval reduction axis, refreshes Chart.js plots, updates climate metrics, and populates the 37-step iteration table.

8. Core C++ Implementation: Kiefer's Point-Reuse While Loop

```
// Exact C++17 Point-Reuse Loop in fibonacci_search.cpp (Zero Redundant Evaluations):
while (k > 3) {
    if (f1 < f2) {  // Minimizer lies in [a, x2], discard [x2, b]
        b = x2;  x2 = x1;  f2 = f1;  --k;
        x1 = a + static_cast<double>(F[k-2]) / F[k] * (b - a);
        f1 = f(x1);  // ONLY 1 new function evaluation!
    } else {      // Minimizer lies in [x1, b], discard [a, x1]
        a = x1;  x1 = x2;  f1 = f2;  --k;
        x2 = a + static_cast<double>(F[k-1]) / F[k] * (b - a);
        f2 = f(x2);  // ONLY 1 new function evaluation!
    }
    ++res.evals;  res.history.push_back({a, b});
}
res.x = 0.5 * (a + b);  res.fx = f(res.x);  ++res.evals;
```

9. Scientific Findings & Atmospheric CO_2 Extrapolations

Optimal Fitted Model: $CO_2(t) = 255.15 + 59.5550 \cdot \exp(0.01605 \cdot (\text{year} - 1959))$ (RMSE = 0.687 ppm, SSE = 30.66 ppm^2).
Doubling Time of Excess CO_2: The excess growth rate r* yields doubling time: **$t_{double} = \ln(2) / r^* = 0.693147 / 0.016051 \approx 43.2$ years**. Excess anthropogenic atmospheric carbon doubles roughly every four decades.
Milestone Future Projections: Forward extrapolation projects: **441.3 ppm in 2030**, **473.7 ppm in 2040**, **511.8 ppm in 2050**, and **556.4 ppm in 2060**.

10. Method Comparison & Viva Defense Guide (Professor Q&A)

Q1: Why Fibonacci over Gradient Descent?	Gradient methods require analytical derivatives (f', f''). Fibonacci search is derivative-free, making it resilient to black-box objectives, noisy sensor records, and functions where derivatives are unavailable.
Q2: How was 3D reduced to 1D?	Variable projection (Golub & Pereyra): for any fixed growth rate r, the linear parameters C_0 and A are solved in closed form via linear regression on $e_i = \exp(r \cdot t_i)$. This leaves a strictly unimodal 1D objective SSE(r).
Q3: Fibonacci vs. Golden Section?	Golden section uses the constant asymptotic ratio $(\sqrt{5} - 1)/2 \approx 0.618$. Fibonacci search uses the variable ratio F_{k-1}/F_k, which Kiefer (1953) proved achieves the smallest possible minimax interval for a fixed N evaluations.

11. Execution Commands & Authoritative References

Target Action	Command Line Instruction	Expected Output & Result
Run Full-Stack App	`python server.py`	Starts FastAPI at http://127.0.0.1:8000 (serves UI + API)
Compile C++ Core	`g++ -O2 -std=c++14 -o fibonacci_search.exe fibonacci_search.cpp`	Builds native optimized C++ binary
Run C++ Binary	`.\fibonacci_search.exe`	Prints optimal r*, evals, and outputs 3 CSV files
Run Python Script	`python fibonacci_search.py`	Runs pure Python numerical benchmark

References: *• J. Kiefer, Sequential minimax search for a maximum, Proc. Amer. Math. Soc., 4(3), 502–506 (1953). | • G. H. Golub & V. Pereyra, Differentiation of pseudo-inverses in nonlinear least squares, SIAM J. Numer. Anal. (1973). | • NOAA Global Monitoring Laboratory, Trends in Atmospheric CO_2 — Mauna Loa Annual Means, gml.noaa.gov (1959–2023).*