Optimising Sustainable Financial Portfolios

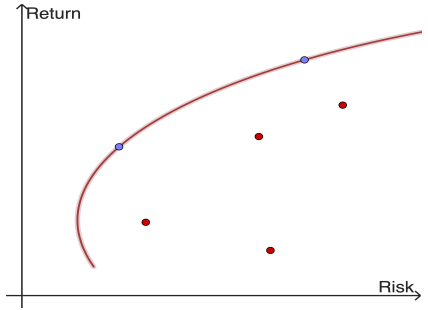
Martin Elsman[1]

[1]Department of Computer Science, University of Copenhagen (DIKU)

September 26, 2023

Financial Portfolio Management

- Investors are faced with a multitude of factors that need to be balanced, including portfolio **returns**, various kinds of **risks**, and **sustainability** concerns.
- Modern portfolio theory (Markowitz) and various derivatives thereof provide techniques for balancing **risk** and **return**.



Modern portfolio theory and "the efficient frontier"

But, how can an investor incorporate sustainability concerns?

What are Sustainable Portfolios?

- How can an investor determine that a portfolio is sustainable?
- How can investors make sustainable financial decisions?

Solution to the Rescue

- Environmental, Social, and Governance (ESG) scores are reported for exchange-traded companies by independent actors.
- ESG data adds another dimension of tradeoff to the efficient frontier!

Existing Solutions

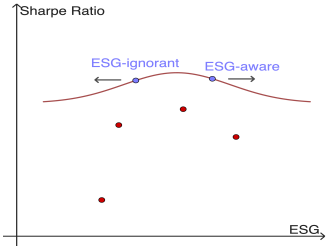
Filtering

An investor may choose to invest only in assets that have certain ESG properties.

The ESG-Efficient Frontier

Given a minimum weighted-ESG-score for a portfolio, an investor may choose to optimise the *Sharpe ratio*, the ratio between the portfolio's return and the portfolio's variance (risk).

- The investor thereby **reduces the dimensionality** of the problem from three to two [Heje Pedersen et al, 2021].
- This approach makes it possible to identify investors who are ESG-ignorant (a portfolio exists that has a higher weighted-ESG-score and the same Sharpe ratio...)



The "ESG-efficient frontier"

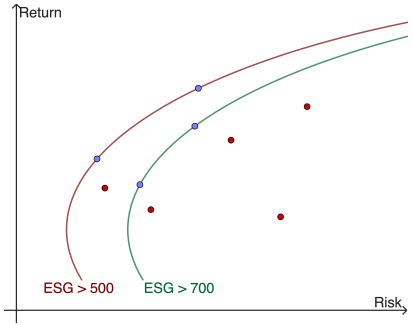
A 3D ESG-Efficient Frontier

In this project,* we seek to compute a 3D ESG-efficient frontier.

- For each ESG-level (e.g., minimum weighted-ESG-score or minimum weighted-E-score), we compute an efficient frontier, maximising return for a particular risk-level.
- This strategy may require us to solve > 10.000 optimisation problems.
- Notice that each problem is an **optimisation problem with inequality constraints** (requiring that the weighted ESG-score is greater than a threshold), which are more difficult to solve than ordinary linear optimisation problems.
- ($*$) *The project "**Optimising Sustainable Financial Portfolios**" is funded in 2023 by Copenhagen Fintech. Company partners are FinE (a Fintech development firm) and Optimal Invest (a family office investment firm).*

Project Activities

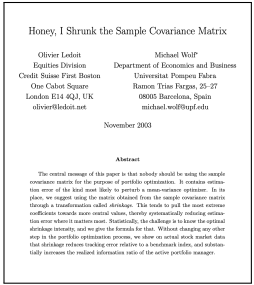
- Supervision of six **computer science and economics BSc projects** in 2023 on the subject of sustainable financial optimisation (14 students). Theoretical developments and implementation of Python libraries for providing investment guidance.
- Supervision of one **computer science MSc thesis project** on data-parallel techniques for optimising sustainable finance.
- Student workshops.



- FinE, with Claus Madsen, has provided ESG data from a collaboration partner (ApexESGenterprise; `https://www.esgenterprise.com`).
- Optimal Invest has provided feedback and expressed interest in tools that can guide them about sustainable investments.

How to Solve 10.000 Non-Linear Optimisation Problems in Parallel?

- On a single CPU, solving 10.000 simple non-linear optimisation problems (fewer than 20 stocks on S&P 500) with associated ESG data **takes several minutes** (using CVXPY).
- Larger universes (e.g., S&P 500) result in **much longer compute times** (require also covariance shrinkage [Ledoit & Wolf, 2003]).
- More constraints (e.g., no shortage) give **longer compute times**.



Honey, I Shrunk the Sample Covariance Matrix

Olivier Ledoit
Equities Division
Credit Suisse First Boston
One Cabot Square
London E14 4QJ, UK
olivier@ledoit.net

Michael Wolf*
Department of Economics and Business
Universitat Pompeu Fabra
Ramon Trias Fargas, 25–27
08005 Barcelona, Spain
michael.wolf@upf.edu

November 2003

Abstract

The central message of this paper is that nobody should be using the sample covariance matrix for the purpose of portfolio optimization. It contains estimation error of the kind most likely to perturb a mean-variance optimizer. In its place, we suggest using the matrix obtained from the sample covariance matrix through a transformation called *shrinkage*. This tends to pull the most extreme coefficients towards more central values, thereby systematically reducing estimation error where it matters most. Statistically, the challenge is to know the optimal shrinkage intensity, and we give the formula for that. Without changing any other step in the portfolio optimization process, we show on actual stock market data that shrinkage reduces tracking error relative to a benchmark index, and substantially increases the realized information ratio of the active portfolio manager.

- Other risk measures (e.g., VaR, CVaR, CVA) demand **even more compute power**.
- **Solution:** Data-parallel implementations on GPUs!

Task Parallelism versus Data Parallelism

- **Task parallelism (CPUs)**: Multiple (e.g., 16) cores run different tasks (on different data).
- **Data parallelism (GPUs)**: Many (e.g, 10.000) cores run the same task simultaneously (on different data).

*GPU programming is **different** from CPU programming!*

Many pitfalls with large performance drops and opportunities for errors:

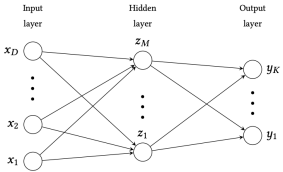
Memory coalescing problems, thread divergence problems, shared- and private memory, memory fences, ...

Increased Focus on Tensor Programming

The concept of **multi-dimensional arrays** (i.e., *tensors*) is currently undergoing a renaissance in terms of available programming libraries and code bases.

The increased attention is primarily driven by:

- The last decade's **machine learning revolution**, which is founded on tensor-programming.
- The move towards **massively data-parallel hardware** for high-performance computing, for which tensor-programming is a natural match.



Futhark,[1] the Language — Purely Functional & Data-Parallel

- Features a selection of array combinators with **parallel semantics**:

```
val map    [n] 'a 'b : (a → b) → [n]a → [n]b
val scan      [n] 'a : (a → a → a) → a → [n]a → [n]a
val reduce       'a : (a → a → a) → a → []a → a
val filter       'a : (a → bool) → []a → []a
val iota            : (n:i64) → [n]i64    -- may fail
```

- Notice that types may be parameterized over types and array sizes.

- Examples:

```
let xs = iota 1000000                  -- create array [0,1,...,999999]
let ys = map (+2) (iota 1000000)       -- add two to each element in xs
let y = reduce (+) 0 xs                -- sum elements
let odds = filter (\x → x%2 == 1) ys -- find the odd elements
```

[1] Futhark is joint work with a number of researchers @ DIKU, including Troels Henriksen, Cosmin Oancea, Fritz Henglein, Ken Friis Larsen, and Philip Munksgaard.

Example: Matrix-Multiplication in Futhark

```
let matmul [n][m][p] (a:[n][m]f64) (b:[m][p]f64) : [n][p]f64 =
  map (\arow →
        map (\bcol → reduce (+) 0 (map2 (*) arow bcol))
            (transpose b)
      ) a
```

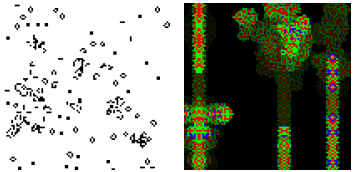
Notice:

- Futhark can assume **compatibility of array dimensions** and generate efficient boundary check-free code.
- When calling `matmul`, Futhark must be able to establish that the array sizes match.
- The programmer may insert *type constraints* (`:>`) for which sizes are checked dynamically.

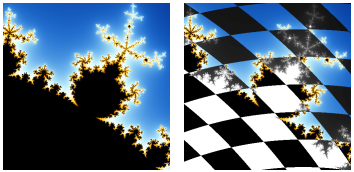
  Such type constraints are rarely needed and when they are, they are explicit (44k lines of benchmark code contains 66 size constraints, mostly in pre- and post-processing code).

Futhark, the Compiler (I)

- Supports **higher-order modules**, which are eliminated at compile time.
- Supports a restricted notion of **higher-order functions**, which are eliminated at compile time. (*functions may not appear in arrays or returned by branches of conditionals*)
- **Other features:** Open source, easy to download and use, used for educational and research purposes, package management… See `http://futhark-lang.org`…



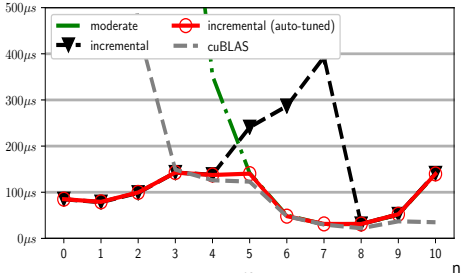
Modularised variants of Conway's Game of Life.



Functional images—Mandelbrot merged with skewed chess board.

Futhark, the Compiler (II)

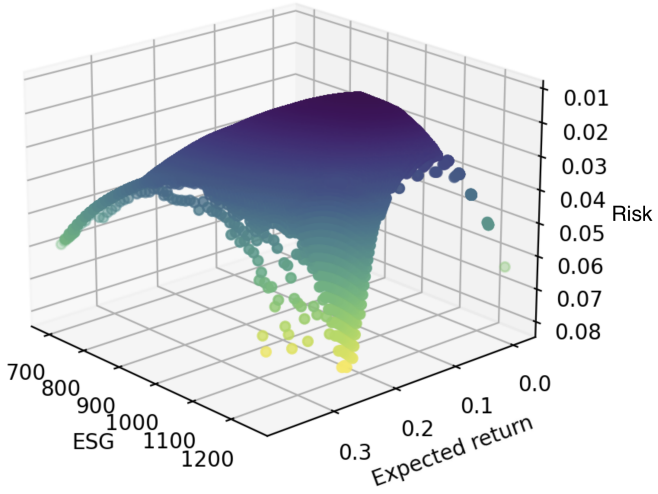
- Targets a series of architectures, including **GPUs** and **CPUs** through CUDA, OpenCL, C, and PyOpenCL.
- Supports nested regular parallelism and generates **multi-versioned** code using a series of aggressive techniques for **fusion**, **flattening**, and **tiling**.



Autotuned Futhark code for multiplying a $2^n \times 2^{(20-2n)}$ matrix with its transposition.

- Irregular nested parallelism must be flattened manually by the user, for instance using segment arrays. Higher-order library functions encapsulate **certain patterns of irregular flattening**.

The 3D ESG-Efficient Frontier in 2.3 Seconds on a GPU



Plot from MSc thesis by Kasper Unn Weihe: Convex Optimization and Parallel Computing for Portfolio Optimization. 2023.

Ongoing Work. More to come...

- Allow for other risk measures (Greeks, VaR, CVaR, CVA, XVA, ...)
- Time series of ESG data...
- Other instruments (bonds, swaps, options, OTCs)...
- Multi-currency considerations...

Other related application areas

- Computational economics (e.g., computing equilibriums in the Danish car fleet).